An Incremental Syntactic Language Model for Statistical Phrase-based Machine Translation

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Lane Oscar Bingaman Schwartz

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

William Schuler

February, 2012

Acknowledgements

During my time as a PhD student at the University of Minnesota, I often told others that I was the only full-time machine researcher between Pittsburgh and Stanford. While I eventually learned of a handful of exceptions to that statement (most notably in Dayton, Ohio, where I am now employed), in retrospect my assumption that I could successfully pursue research in machine translation at an institution with no machine translation faculty seems remarkably naive.

My eventual success in this achievement is due in no small part to my advisor, William Schuler, who supported and encouraged me in my research. It was on his suggestion that I set out in 2005 to implement the first open source implementation of hierarchical phrase-based machine translation, based solely on the description in the original paper (Chiang, 2005). So it was that I became perhaps the only machine translation researcher ever to be introduced to statistical machine translation through hierarchical phrase-based models, and only later become aware of the more straightforward word-based and phrase-based models. In the spring of 2008, I learned of an upcoming event — the 2nd Machine Translation Marathon, to be held in May in Wandlitz, Germany. I owe my advisor a tremendous debt of gratitude for providing funds for me to attend. My decision to attend this event was the single most important decision of my graduate career. That time in Wandlitz provided me with an invaluable opportunity to meet and learn from some of the finest researchers in our field. It was there that I met Chris Callison-Burch, who at the end of the week offered me a job working on Joshua, his lab's hierarchical machine translation system. I am immensely grateful to Chris for the opportunity to collaborate with him both then and now, and for the doors that that collaboration helped to open.

I am a firm believer that as scientists, we have a duty to ensure that the research we perform is documented and reproducible. As a computer scientist, I owe a great deal to the many researchers and developers who have developed the great wealth of tools that allow me to do the research that I love:

- Thanks to Richard Stallman, Linus Torvalds, and the many other developers of GNU and Linux. You guys rock. Thanks to Steve Wozniak and Steve Jobs for the computers that got me hooked. Thanks to Bill Gates for Applesoft BASIC and for the opportunity to study at the University of Cambridge.

- Thanks to Yukihiro "Matz" Matsumoto. Ruby was an amazing sancuary for this object-oriented refugee from Perl. To this day I know of no better programming language for natural language processing.

- Thanks to Marian Olteanu, for Phramer, an open source Java implementation of a statistical phrase-based machine translation decoder. This code, especially the MERT implementation, was extremely helpful as I wrote my own hierarchical machine translation decoder.

- Thanks to the members of the 2006 JHU Summer Workshop who worked on the initial implementation of Moses, the de facto standard for phrase-based machine translation: Philipp Koehn, Nicola Bertoldi, Ondřej Bojar, Chris Callison-Burch, Alexandra Constantin, Brooke Cowan, Chris Dyer, Marcello Federico, Evan Herbst, Hieu Hoang, Christine Moran, Wade Shen and Richard Zens. Thanks also to all of the many, many other researchers and developers who have contributed to Moses and supported the many users of Moses.

- Thanks to the attendees of the 2nd Machine Translation Marathon, especially Chris Callison-Burch, Philipp Koehn, Phil Blunsom, Adam Lopez, Chris Dyer, Hieu Hoang, Barry Haddow, Ondřej Bojar, Josh Schroeder, and Martin Kay. You guys provided me with my first real opportunity to collaborate with other machine translation researchers. I am extremely grateful to Josh Schroeder and Martin Kay for our extensive conversations regarding multi-source machine translation.

- Thanks to Nitin Madnani for the iBLEU visualization tool. The translation visualizations in Chapter 5 were produced using that tool, and would not otherwise have been possible.

- Thanks to Mark Przybocki for providing the uncased NIST OpenMT results.

- Thanks to Jon Clark, for continuing to provide me with tools that are tantalizingly close to being incredibly useful. ⌣

Thanks to all of my colleagues and instructors who have supported me throughout this long journey:

- Thanks to Tim Anderson, Ray Slyh, Eric Hansen and my other colleagues at Air Force Research Lab, who shepherded me through the federal hiring process, made me feel welcome, shielded me from red tape, and allowed me the time and freedom to finish the research and writing for this dissertation.

- Thanks to my labmates at the University of Minnesota. I really miss our time together, not to mention tacos at Sally's. Thanks also to my excellent instructors, especially George Karypis, Gopalan Nadathur, and John Riedl. And special thanks to Georganne Tolaas, without whom nothing is possible.

- Thanks to my classmates and instructors in the M.Phil program in Computer Speech, Text & Internet Technology at the University of Cambridge. That experience provided me with a firm foundation in natural language processing that I use in my research every day.

- Thanks to my managers and colleagues at IBM Rochester. My time at IBM provided me with outstanding positive examples of how to build and develop good professional relationships, as well as the benefits of regular 1-on-1 meetings with your manager. I am incredibly grateful for the opportunity to go on educational leave of absence to pursue my dream of a PhD. One of my only regrets from graduate school is that it didn't work out for me to return to IBM when I finished.

- Thanks to my friends and instructors at Luther College and the University of Nottingham, especially Steve Hubbard, Dave Ranum, Kent Lee, Paul Gardner, and Dave Elliman.

- Thanks to all of the instructors in Gambell, Elim, Wrangell, and Greenfield who encouraged and guided me through the years. I want to thank my Yupik language instructors — though I didn't know it at the time, the seeds that were sown then grew into my love of language and my research in machine translation. I especially want to thank Mary Guthridge, Jerri Moore, Butch Schmidt, and Bob Daut. Thanks also to Delma Apassingok, who provided the text and audio recording of the pledge of allegiance in Central Siberian Yupik for my final dissertation defense.

Thank you to Matt, Chris, Sarah, Tara, Mike, Andy, and all my other friends and fellow board gamers who helped keep me sane throughout the years.

And last, but most certainly not least, thanks to my family. Without your love, encouragement, and support, this work would not have been possible. I want to thank my parents, my siblings, and my wonderful extended family. And most of all, thanks to my incredible wife and to our children. Thank you for your love, your patience, and your understanding. Thank you for waiting for me while I spent time apart, first in Rochester, then to conferences, and finally through two different summers in Baltimore. Thank you for believing in me and following me on this incredible journey.

Dedication

I dedicate this work to the voices that shaped me and the voices to shape tomorrow:

To the winds and snow of Sivuqaq

To the roiling surf of Norton Sound

To the hidden gems of the inside passage

To my wife, who followed me around the world

And to my children, that the world you inherit might better understand

Abstract

Modern machine translation techniques typically incorporate both a *translation model*, which guides how individual words and phrases can be translated, and a *language model* (LM), which promotes fluency as translated words and phrases are combined into a translated sentence. Most attempts to inform the translation process with linguistic knowledge have focused on infusing syntax into translation models. We present a novel technique for incorporating syntactic knowledge as a language model in the context of statistical phrase-based machine translation (Koehn et al., 2003), one of the most widely used modern translation paradigms.

The major contributions of this work are as follows:

- We present a formal definition of an incremental syntactic language model as a Hierarchical Hidden Markov Model (HHMM), and detail how this model is estimated from a treebank corpus of labelled data.
- The HHMM syntactic language model has been used in prior work involving parsing, speech recognition, and semantic role labelling. We present the first complete algorithmic definition of the HHMM as a language model.
- We develop a novel and general method for incorporating any generative incremental language model into phrase-based machine translation. We integrate our HHMM incremental syntactic language model into Moses, the prevailing phrase-based decoder.
- We present empirical results that demonstrate substantial improvements in perplexity for our syntactic language model over traditional n-gram language models; we also present empirical results on a constrained Urdu-English translation task that demonstrate the use of our syntactic LM.

A standard measure of language model quality is average per-word perplexity. We present empirical results evaluating perplexity of various n-gram language models and our syntactic language model on both in-domain and out-of-domain test sets. On an in-domain test set, a traditional 5-gram language model trained on the same data as our syntactic language model outperforms the syntactic language model in terms of perplexity. We find that interpolating

the 5-gram LM with the syntactic LM results in improved perplexity results, a 10% absolute reduction in perplexity compared to the 5-gram LM alone.

On an out-of-domain test set, we find that our syntactic LM substantially outperforms all other LMs trained on the same training data. The syntactic LM demonstrates a 58% absolute reduction in perplexity over a 5-gram language model trained on the same training data. On this same out-of-domain test set, we further show that interpolating our syntactic language model with a large Gigaword-scale 5-gram language model results in the best overall perplexity results — a 61% absolute reduction in perplexity compared to the Gigaword-scale 5-gram language model alone, a 76% absolute reduction in perplexity compared to the syntactic LM alone, and a 90% absolute reduction in perplexity compared to the original smaller 5-gram language model.

A language model with low perplexity is a theoretically good model of the language; it is expected that using an LM with low perplexity as a component of a machine translation system should result in more fluent translations. We present empirical results on a constrained Urdu-English translation task and perform an informal manual evaluation of translation results which suggests that the use of our incremental syntactic language model is indeed serving to guide the translation algorithm towards more fluent target language translations.

Contents

List of Tables

List of Figures

List of Algorithms

Disclaimer

Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Air Force. This dissertation was cleared for public release on 19 Jan 2012 with originator reference number RH-12-109026 and case number 88ABW-2012-0302.

Chapter 1

Introduction

Modern machine translation techniques typically incorporate both a *translation model*, which guides how individual words and phrases can be translated, and a *language model* (LM), which promotes fluency as translated words and phrases are combined into a translated sentence. Most attempts to inform the translation process with linguistic knowledge have focused on infusing syntax into translation models. We present a novel technique for incorporating syntactic knowledge as a language model in the context of statistical phrase-based machine translation (Koehn et al., 2003), one of the most widely used modern translation paradigms.

The primary novel contribution of this dissertation is a method for using an incremental syntactic language model to guide a machine translation decoder as it translates sentences into a target language. A syntactic language model is a formal probabilistic model of the syntax, or structure, of a human language, and is defined in terms of a *grammar* and a *parsing* algorithm.

Formal theories of dependency grammar (Tesnière, 1959; Mel'čuk, 1988) represent sentence structure as a dependency graph. Words in a sentence are modelled as nodes in a graph, connected by (possibly labelled) directed arcs. In such models, each arc represents a dependency between a head word and a dependent argument of that word. Arcs might exist, for example, between the noun head of a noun phrase and an adjective modifying that noun, or from a verb to an object of the verb. Other formal theories, such as the prominent X-Bar theory (Chomsky, 1970; Jackendoff, 1977; Kornai and Pullum, 1990) of language structure, model the hierarchical structure of phrases within sentences. Using such phrase structure grammars, sentence structure can be annotated in the form of phrase structure trees. Phrase structure grammars are well-suited to adaptation as computational language models.

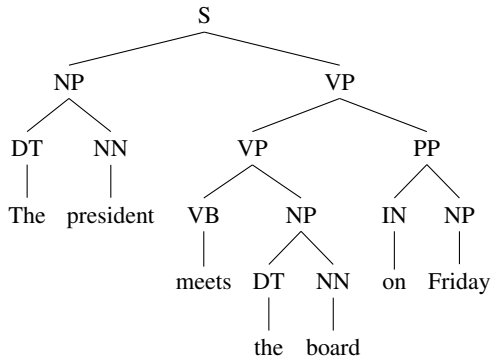


Figure 1.1: Graphical representation of a phrase structure tree. This tree illustrates the hierarchical structure of the underlying sentence (S), which is comprised of a noun phrase (NP) followed by a verb phrase (VP). Each of these child sub-trees is also hierarchically structured.

Parsing is the task of selecting the representation $\hat{\tau}$ (in our case a phrase structure tree defined by a phrase structure grammar) that best models the structure of sentence $\mathbf{e}$, out of all such possible representations $\boldsymbol{\tau}$. This set of representations may be all phrase structure trees allowed by the grammar. Typically, tree $\hat{\tau}$ is taken to be:

$$\hat{\tau} = \underset{\tau}{\operatorname{argmax}}\, \mathsf{P}(\tau \mid \mathbf{e}) \tag{1.1}$$

In other words, the parser's task is to select the tree that is most probable according to the grammar, given the sentence $\mathbf{e}$. Figure 1.1 depicts a phrase structure tree that results from parsing the English sentence *The president meets the board on Friday*.

Our ultimate goal is not to obtain the most likely tree representation of a sentence using a grammar and a parser. Rather, it is to use the syntactic language model defined by a grammar and parser to guide a machine translation decoding algorithm towards translations which are syntactically well-formed in the target language. To that end, we propose that our syntactic language model should calculate the total probability mass over all possible phrase structure trees for sentence $\mathbf{e}$. This is shown in Equation 1.2:

$$\mathsf{P}(\mathbf{e}) = \sum_{\tau \in \boldsymbol{\tau}} \mathsf{P}(\tau, \mathbf{e}) \tag{1.2}$$

It is typically not tractable in practice to sum over all possible trees licensed by the grammar. Typically, pruning is performed during parsing, which results in some portion of the possible

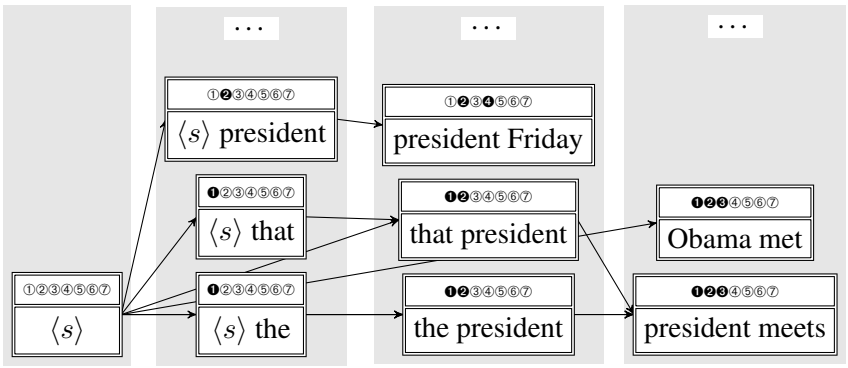


Figure 1.2: Partially constructed translation lattice resulting from statistical phrase-based translation algorithm for the German sentence *Der Präsident trifft am Freitag den Vorstand.*

trees being discarded. We define the syntactic language model probability after pruning in Equation 1.3, where $\tilde{\boldsymbol{\tau}}$ represents those trees which remain after pruning:

$$\mathsf{P}(\mathbf{e}) = \sum_{\tau \in \tilde{\boldsymbol{\tau}}} \mathsf{P}(\tau, \mathbf{e}) \tag{1.3}$$

Traditional bottom-up parsers (Kasami, 1965; Younger, 1967; Cocke and Schwartz, 1970; Chappelier and Rajman, 1998) and top-down parsers (Earley, 1968, 1970) typically require a completed string as input. This requirement makes it difficult to incorporate them into phrase-based translation, which generates partial hypothesized translations incrementally, from left-to-right. Incremental translation construction in statistical phrase-based translation is shown in Figure 1.2, where each vertical decoding stack (shaded grey) represents nodes in the translation lattice in which the same number of source words have been translated. The lattice is expanded by incrementally adding new nodes representing target language phrase translations of previously untranslated source language phrases.

Beginning with the first statistical approaches to machine translation (Brown et al., 1990), statistical machine translation research has typically followed widespread practice in speech recognition by using n-gram language models (Shannon, 1948, 1951) to model the probability of a contiguous sequence of words. These n-gram models are relatively simple finite-state models, and do not incorporate knowledge about the syntax of the language.

$$\mathsf{P}(e_n \,|\, e_1 \ldots e_{n-1}) = \frac{C(e_1 \ldots e_n)}{C(e_1 \ldots e_{n-1})} \tag{1.4}$$

Equation 1.4 defines the probability of the n^{th} word in a phrase, given the preceding $n-1$ words.[1]

We observe that n-gram language models are incremental in nature, and as such can be easily integrated into the incremental phrase-based translation algorithm. Each node in the phrase-based machine translation search graph stores a coverage vector, denoting which words from the source sentence have been translated in the partial translation at that node, and an n-gram language model state, representing the most recent $n-1$ words in the partially constructed translation. For example, the left-most node in Figure 1.2 represents an empty translation, where no source words have yet been translated; the n-gram language model state is $\langle s \rangle$, representing the start of a sentence. The node in the third decoding stack with n-gram language model state *president Friday* represents a partial translation where only the second and fourth source words (*Präsident Freitag*) have been translated.

Modern phrase-based translation using large scale n-gram language models generally performs well in terms of lexical choice, but still often produces ungrammatical output. The additional use of a syntactic language model may help produce more grammatical output by better modelling structural relationships and long-distance dependencies. In contrast to n-gram language models, traditional bottom-up and top-down parsers, which require a completed sentence, cannot be integrated into the phrase-based translation algorithm in such a straightforward manner. Instead, we must define an incremental syntactic language model capable of processing one word at a time. An incremental syntactic language model is defined by a grammar and an incremental parser.

After processing the t^{th} token in string $\mathbf{e}$, an incremental parser has some internal representation of possible hypothesized (incomplete) trees, $\boldsymbol{\tau_t}$. In practice, a parser may constrain the set of trees under consideration to $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t}}$, that subset of analyses or partial analyses that remains after any pruning is performed. An incremental syntactic language model can then be presented as a probability mass function (Equation 1.5) and a transition function δ (Equation 1.6).

$$\mathsf{P}(e_1 \dots e_t) \approx \mathsf{P}(\tilde{\boldsymbol{\tau}}_t) = \sum_{\tau \in \tilde{\boldsymbol{\tau}}_t} \mathsf{P}(e_1 \dots e_t \,|\, \tau)\mathsf{P}(\tau) \tag{1.5}$$

$$\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1}) \rightarrow \tilde{\boldsymbol{\tau}}_t \tag{1.6}$$

[1] Count function $C(\cdot)$ provides the number of times a word sequence is observed in a training corpus of word sequences. In practice, n-gram language model probability values are smoothed using various backoff techniques.

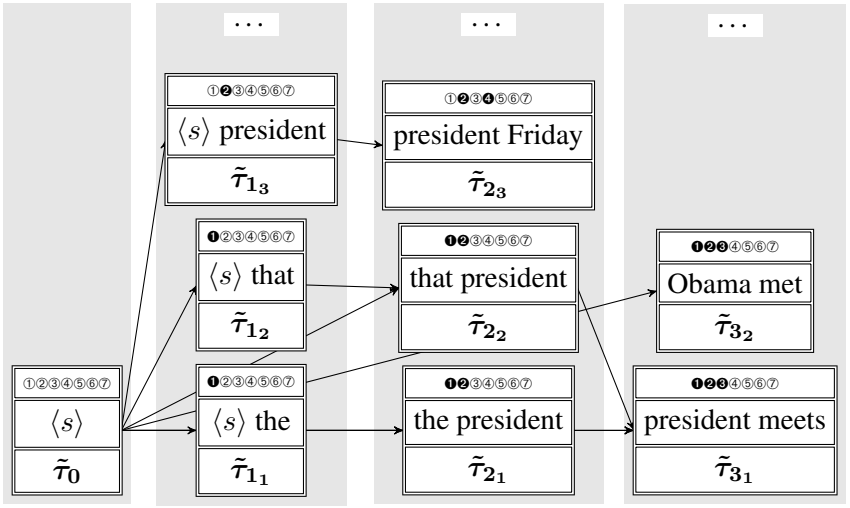


Figure 1.3: Partially constructed translation lattice resulting from statistical phrase-based translation algorithm for the German sentence *Der Präsident trifft am Freitag den Vorstand*. Each node t_n in the translation lattice represents a partial translation of t words from the source sentence. Each node is augmented with syntactic language model state $\tilde{\boldsymbol{\tau}}_{t_n}$, representing the set of syntactic analyses of the translation at node t_n that remain after pruning. State $\tilde{\boldsymbol{\tau}}_{3_2}$, for example, represents a set of syntactic analyses of the English partial translation *Obama met* of the German source words *Der Präsident trifft*.

We augment each node in the translation lattice with a syntactic language model state $\tilde{\boldsymbol{\tau}}$. A syntactic language model state represents all unpruned syntactic analyses of the partial translation at that node. As each node is added to the translation lattice, transition function δ (Equation 1.6) is applied to construct a syntactic language model state for the new node. The use of an incremental syntactic parser provides a straightforward mechanism for introducing syntax to phrase-based machine translation.

Incremental parsing is also attractive from a psycholinguistic perspective. It is well established that humans process language incrementally. Eye-tracking studies show that humans listening to spoken language are able to actively attend to the entities that the spoken words might refer to, even while the words are still being pronounced (Tanenhaus et al., 1995; Brown-Schmidt et al., 2002). There is evidence that humans' incremental syntactic processing abilities may occur entirely within general-purpose short-term memory (Sachs, 1967; Jarvella, 1971;

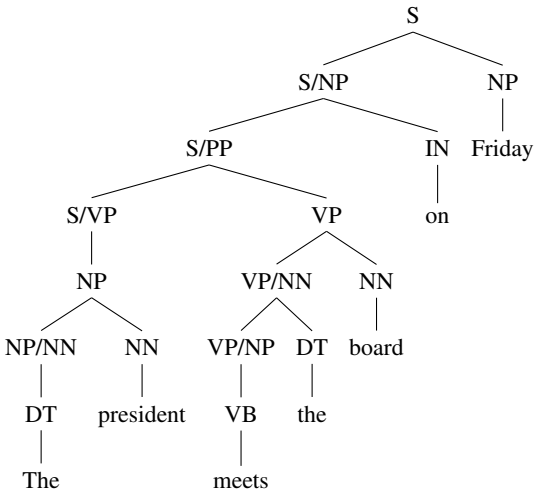


Figure 1.4: Graphical representation of a phrase structure tree in right-corner form.

Just and Carpenter, 1992). This is especially intriguing given strong evidence that general-purpose short-term memory has severe capacity limits of perhaps no more than three to four distinct elements (Miller, 1956; Cowan, 2001).

We propose to use an incremental syntactic language model designed within these psycholinguistic constraints in mind. An incremental syntactic language model is defined by a grammar and an incremental parser. For our grammar, we use a probabilistic phrase-structure grammar in a normalized right-corner form. Figure 1.4 shows a phrase structure tree in right-corner form; this tree was obtained by performing the normalizing right-corner transform on the phrase structure tree from Figure 1.1.

Constituent categories in right-corner trees may correspond to traditional phrase structure constituent categories (such as DT, NP, VP) or to categories representing incomplete constituent structure (such as NP/NN, S/VP, VP/NN). Such incomplete constituent categories are very similar in spirit to those from Combinatorial Categorial Grammars (Steedman, 2000) of the form C_1/C_2. These incomplete categories are interpreted as C_1 lacking C_2 on the right. For example, a determiner (DT) could alternatively be categorized as NP/NN, meaning a noun phrase lacking a common noun. Two subtrees rooted at C_1/C_2 and C_2, respectively, are expected to combine to form a larger subtree rooted at C_1. In Figure 1.4, we see subtrees rooted at NP/NN and NN combining to form a larger subtree rooted at NP.

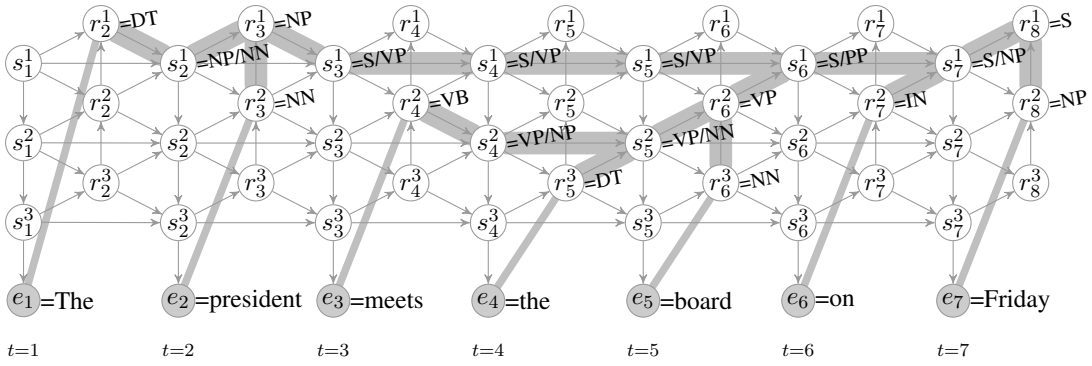


Figure 1.5: Graphical representation of the Hierarchic Hidden Markov Model after parsing input sentence *The president meets the board on Friday*. The shaded path through the parse lattice illustrates the recognized right-corner tree structure of Figure 1.4.

Incremental parsing using a right-corner grammar is designed to minimize the number of memory elements in a short-term memory store required for parsing. We use an incremental parser implemented as a Hierarchical Hidden Markov Model — HHMM (Murphy and Paskin, 2001) — to incrementally recognize phrase structure trees in right-corner form. An HHMM is a factored time series model, equivalent to a a bounded-memory probabilistic push-down automaton. Each vertical layer in our HHMM acts acts one element of a bounded-memory store. Figure 1.5 depicts an HHMM with three levels of bounded memory after successfully parsing the English sentence *The president meets the board on Friday*. Our HHMM can be defined to implement the Equations 1.5 and 1.6 required to define an incremental syntactic language model in the context of statistical phrase-based translation.

We incorporate our HHMM incremental syntactic language model into a phrase-based machine translation decoder with the goal of guiding the translation algorithm towards translations which are more syntactically well-formed. Below we see two translations of a sample Urdu sentence. Translation 1 was produced by a phrase-based decoder configured using an n-gram language model. Translation 2 was produced by a decoder configured using an HHMM syntactic language model in addition to an n-gram language model.

Translation 1 (n-gram) We are on this decision for .

Translation 2 (n-gram & HHMM) We congratulations on this decision .

Four human reference translations for this sample sentence are shown below:

Reference We congratulate on this decision .

Reference We congratulate on this judgement .

Reference We congratulate the court on this decision .

Reference We congratulate them on the ruling .

When we examine the n-gram and HHMM language model scores for the two translations, we see that the n-gram language model gives Translation 1 a higher score, indicating that the model prefers Translation 1 over Translation 2. However, we observe that translation 2 is much more syntactically well-formed than translation 1, and represents a better overall translation. The HHMM syntactic language model gives a higher score to the more syntactically well-formed Translation 2.

Translations and references of a second sample Urdu sentence are shown below:

Translation 1 (n-gram) In the meeting , is not .

Translation 2 (n-gram & HHMM) No harm in the meeting .

Reference There is nothing wrong in meeting .

Reference There is no problem in meeting .

Reference There is no harm in meeting with him .

Reference There are no problems with this meeting .

Translations and references of a third sample Urdu sentence are shown below:

Translation 1 (n-gram) Apart from this , three extremists have been brought Lucknow arrested the Calcutta .

Translation 2 (n-gram & HHMM) Apart from this , three extremists have been arrested from Calcutta to Lucklow .

Reference Moreover , 3 terrorists have been arrested and brought to Lucklow from Calcutta .

Reference Besides , three extremists were arrested from Kolkata and brought to Lucklow .

Reference Additionally , three extremists have been arrested in Kolkata and brought to Lucklow .

Reference Additionally , three extremists have been arrested in Kolkata and brought to Lucklow .

In both cases, we again observe more syntactically well-formed translations when the HHMM syntactic language model is allowed to guide the translation algorithm.

1.1 Main Contributions

- We present a formal definition of an incremental syntactic language model as a Hierarchical Hidden Markov Model (HHMM), and detail how this model is estimated from a treebank corpus of labelled data.
- The HHMM syntactic language model has been used in prior work involving parsing, speech recognition, and semantic role labelling. We present the first complete algorithmic definition of the HHMM as a language model.
- We develop a novel and general method for incorporating any generative incremental language model into phrase-based machine translation. We integrate our HHMM incremental syntactic language model into Moses, the prevailing phrase-based decoder.
- We present empirical results for language model perplexity that show our incremental syntactic language model, implemented as an HHMM, is a good model of language. We present empirical results on an Urdu-English translation task that demonstrate the use of our syntactic LM, and we perform an informal manual evaluation of translation results which suggests that the use of our incremental syntactic language model is indeed serving to guide the translation algorithm towards more fluent target language translations.

1.2 Outline

The remainder of this dissertation is structured as follows:

- In Chapter 2, we present a brief introduction to machine translation. We begin with a historical overview of machine translation. Next, we present foundational background material on phrase-based statistical machine translation, the translation paradigm on which we build in this dissertation. We conclude the chapter by surveying the existing literature on incorporating syntax into statistical machine translation.

- In Chapter 3, we describe the process by which a probabilistic context-free grammar is obtained from the Wall Street Journal treebank corpus of labelled data. We describe several tree transformations required to convert the labelled training data into the optimal form required to train our incremental syntactic language model. We formally define how to train a probabilistic context-free grammar from these transformed phrase structure trees.

- In Chapter 4, we formal define the incremental syntactic language model used in this work. We begin by defining Hidden Markov Models, following Markov (1913). We formally define our incremental syntactic language model as a Hierarchical Hidden Markov Model (Murphy and Paskin, 2001), using the probabilistic context-free grammar from Chapter 3. We present the first complete algorithmic description of the HHMM parser operating as an incremental syntactic language model. We document important speed optimizations.

- In Chapter 5, we present the core of our novel contribution, the incorporation of an incremental syntactic language model into standard phrase-based translation. We begin by defining a general method for integrating any generative incremental syntactic language model into phrase-based translation. We describe how we integrate the incremental syntactic language model from Chapter 4, in conjunction with the probabilistic context-free grammar from Chapter 3, into Moses, the prevailing phrase-based decoder. Integration of our syntactic language model into phrase-based translation comes with a cost to translation speed; we examine this issue and present a mechanism for alleviating the problem. Finally, we present empirical results that demonstrate substantial improvements in perplexity for our syntactic language model over traditional n-gram language models; we also present empirical results on a constrained Urdu-English translation task that demonstrate the use of our syntactic LM.

- Chapter 6 presents a final discussion of contributions of this work.

1.3 Related Publications

The core novel contribution of this dissertation, the incorporation of an incremental syntactic language model into standard phrase-based translation (introduced in Chapter 1 and presented in Chapter 5), directly extends Schwartz et al. (2011), joint work with Chris Callison-Burch, William Schuler, and Stephen Wu. That work was presented at the Annual Meeting of the Association for Computational Linguistics, the premier international venue for research in machine translation and natural language processing.

The incremental syntactic language model we introduce in Chapter 1, train in Chapter 3, and present in Chapter 4 builds on our earlier published work in HHMM language models for spoken language interfaces (Miller et al., 2007; Wu et al., 2008a,b; Schwartz et al., 2009; Schuler et al., 2009) and parsing (Schuler et al., 2008, 2010). These publications are joint work with the other members of the University of Minnesota Natural Language Processing group — William Schuler, Tim Miller Stephen Wu, Andey Exley, Samir AbdelRahman, and Luan Nguyen. In this work, we fully document for the first time the complete algorithmic definition of the HHMM as a language model

Other of our publications represent more indirect contributions to this dissertation. Schwartz (2008b) explores the use of phrase-based machine translation when the source document to be translated is available in multiple languages. Schwartz (2008a) presents the first open source implementation of a hierarchical machine translation system, capable of translating using a formally syntactic hierarchical translation model. This work was continued in our collaboration in the Joshua machine translation system (Li et al., 2009a,b; Schwartz and Callison-Burch, 2010; Schwartz, 2010; Li et al., 2010), a similar translation system which added support for linguistically syntactic translation models. Our joint work at the Johns Hopkins University 2009 Summer Camp for Advanced Language Exploration — SCALE (Baker et al., 2009) — examined various techniques for syntactic and semantic translation model augmentation. Research on Joshua was joint work with the Joshua research group, including Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Anne Irvine, Sanjeev Khudanpur, Wren Thornton, Ziyuan Wang, Jonathan Weese, and Omar Zaidan, as well as the members of SCALE 2009.

Chapter 2

Literature Review: Syntax in Machine Translation

The primary novel contribution of this dissertation is the development of an incremental syntactic language model for use in statistical phrase-based machine translation. In Chapters 3–5, we present a formal language model of English syntax and incorporate that model into a contemporary machine translation system. This chapter presents the historical and contemporary background for our contribution.

2.1 History

Machine translation is one of the oldest disciplines within computer science. Computers excel at retrieving and presenting large amounts of stored data. The process of translating a document requires retrieving data (translated words, phrases, possibly even whole sentences) using presented key values in the source language. It is not surprising that some of the first electronic and digital computers were programmed to perform such tasks. In 1933, patents were issued to Petr Trojanskij in Russia and Georges Artsrouni in France for mechanical dictionaries (Hutchins, 2004). Some of the early work in machine translation also drew inspiration from Allied successes in cryptology during World War II (Weaver, 1949). Researchers at IBM and Georgetown University later collaborated to produce the first public demonstration of translation using computers (IBM, 1954); the presentation of this small Russian-English system became known as the Georgetown Experiment.

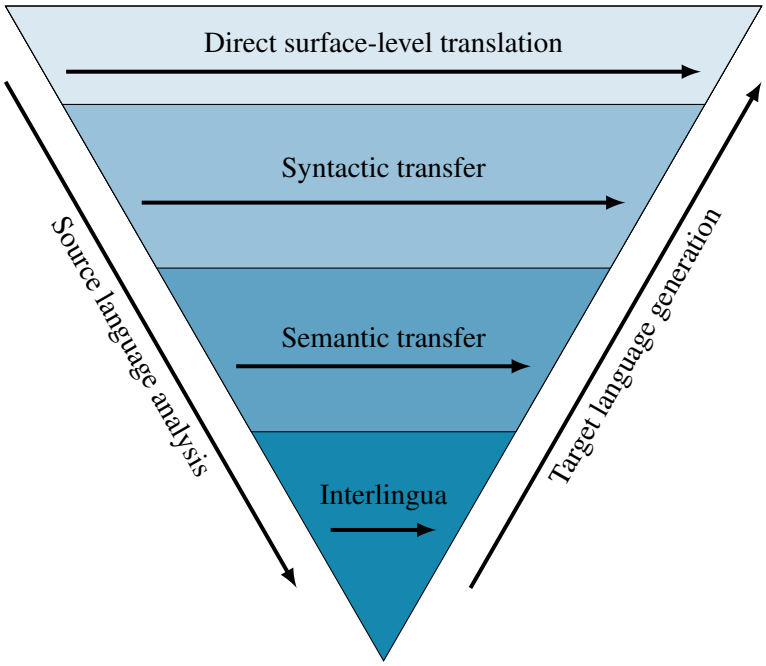


Figure 2.1: Machine translation triangle, adapted from (Vauquois, 1968). The triangle depicts various levels of depth that machine translation systems may adopt during source language text analysis, representational source to target transfer, and target language text generation. This triangle is traditionally drawn with surface analysis at the base and interlingua at the top. We prefer this inverted structure, as increasingly deep analysis techniques descend the levels of the triangle. Direct translation approaches translate directly from source language surface tokens into the target language. Transfer approaches typically perform deeper source language analysis, involving syntax and possibly semantics; the resulting structural representations are transfered into target language structural representations, and a separate generation phase produces the final target language surface text. The deepest approaches attempted to analyze the source text into a truly language-independent interlingua format, from which a generation phase would produce the final target language surface text.

Research continued over the next decade, with many researchers expressing optimism that fully automatic, high-quality machine translation results would soon be realized. Machine translation research coalesced around three main competing approaches: direct, transfer, and interlingua. Direct translation, typified by Reifler (1961), attacked translation at a shallow surface level. At its simplest, direct translation used bilingual dictionaries to perform word-for-word substitution from source language to target language, with no attempt made at word reordering. At the other extreme, interlingua research, seen in Ceccato (1956), proposed to deeply analyze the source text into a truly language-independent meaning representation, with the hope that target text could then be directly generated from this interlingual representation. Straddling the divide between direct and interlingua translation lay transfer methods, such as Lehmann (1957, 1998), which decompose translation into three phases: analysis, transfer, and generation. The analysis phase transformed source language tokens into an intermediate structural representation; the depth of analysis varied between systems, but could be morphological, syntactic, or even semantic. The transfer phase utilized encoded linguistic rules to construct a target language structural representation derived from the source language structural representation. The final generation phase produced target language text from the target language structural representation. Figure 2.1 illustrates the varying depths at which analysis, transfer, and generation occur in various methodologies.

The optimism of the first decade of machine translation research was not universal. Bar-Hillel (1960) argued that semantic ambiguities inherent in human language present an insurmountable barrier to high quality fully automatic translation, regardless of the approach. The Automatic Language Processing Advisory Committee later reported to the U.S. government that "there is no immediate or predictable prospect of useful machine translation" (ALPAC, 1966), leading to a rapid halt in nearly all U.S. government funding of machine translation research.

Machine translation research slowed, but did not stall completely. Over the next thirty-five years, machine translation research became dominated by transfer-based methods (Hutchins, 2003). The dominant commercial systems that emerged (SYSTRAN, Logos, and METAL) all incorporated syntactic analysis, transfer, and generation. Transfer translation methods encode language-specific and language pair-specific linguistic knowledge into analysis rules, transfer rules, and generation rules. Such rule-based systems tend to be expensive to develop (in terms of both time and money), as linguists must be closely consulted to properly develop and update the system rule database.

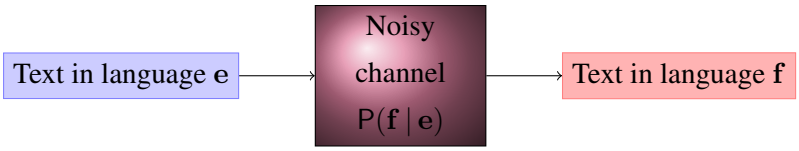


Figure 2.2: Noisy channel process, adapted from Shannon (1948). A message travels from the origin (left) to the destination (right). At its destination, the message appears to be written in foreign language $\mathbf{f}$. The noisy channel model postulates that at the origin, the message was originally written in language $\mathbf{e}$, but was corrupted by noise according to translation model $\mathsf{P}(\mathbf{f} \mid \mathbf{e})$.

2.2 Statistical Translation Models

By the early 1980s, speech recognition researchers had successfully applied statistical pattern recognition techniques (Baker, 1979; Ferguson, 1980; Bahl et al., 1983). These techniques trained statistical models of real-world phenomena from collected speech and transcription data. These models could be used in conjunction with decoding software to transcribe new, previously unseen, speech data.

2.2.1 Noisy Channel Framework

Drawing on the success of statistical approaches to speech recognition, and inspired by Weaver (1949), researchers at IBM proposed a statistical direct translation model (Brown et al., 1988). Given a parallel corpus of existing translated documents and sufficient computing resources, such word-based statistical models could be trained automatically (Brown et al., 1990, 1993), without consulting a linguist or domain expert. For a given language pair, a statistical word translation model, $\mathsf{P}(\mathbf{f} \mid \mathbf{e})$, defines a probability distribution for translating individual words from language $\mathbf{f}$ into language $\mathbf{e}$. Figure 2.2 illustrates this word-based translation as a noisy channel process (Shannon, 1948). Viewing translation as a noisy channel process, a given text in foreign source language $\mathbf{f}$ can be transformed into its translation in target language $\mathbf{e}$ according to Equation 2.1.

$$\mathsf{P}(\mathbf{e} \mid \mathbf{f}) = \frac{\mathsf{P}(\mathbf{f} \mid \mathbf{e})\mathsf{P}(\mathbf{e})}{\mathsf{P}(\mathbf{f})} \tag{2.1}$$

Equation 2.1 defines translation model probability $\mathsf{P}(\mathbf{e} \mid \mathbf{f})$ in terms of word translation model $\mathsf{P}(\mathbf{f} \mid \mathbf{e})$, as well as $\mathsf{P}(\mathbf{e})$ and $\mathsf{P}(\mathbf{f})$, which model the prior probability of the target language $\mathbf{e}$ and source language $\mathbf{f}$, respectively. Statistical translation software which searches for the most probable target language sentence $\hat{e}$ given source language sentence f is commonly refered to as a decoder. Equation 2.2 formalizes the search task for a decoder:

$$\hat{e} = \arg\max_{e} \mathsf{P}(\mathbf{e} \mid \mathbf{f}) = \arg\max_{e} \frac{\mathsf{P}(\mathbf{f} \mid \mathbf{e})\mathsf{P}(\mathbf{e})}{\mathsf{P}(\mathbf{f})} \tag{2.2}$$

Prior probability models $\mathsf{P}(\mathbf{f})$ and $\mathsf{P}(\mathbf{e})$ are called the source language model and target language model, respectively. Because the denominator in Equation 2.2 is constant with respect to e, source language model $\mathsf{P}(\mathbf{f})$ is not required for decoding, and Equation 2.2 can be simplified to Equation 2.3:

$$\hat{e} = \arg\max_{e} \mathsf{P}(\mathbf{f} \mid \mathbf{e})\mathsf{P}(\mathbf{e}) \tag{2.3}$$

2.2.2 N-gram language models

To define target language model $\mathsf{P}(\mathbf{e})$, machine translation research typically follows widespread practice in speech recognition by using n-gram language models (Shannon, 1948, 1951) to model the prior probability of a contiguous sequence of words. These n-gram models are relatively simple finite-state models, and do not incorporate knowledge about the syntax of the language. While alternate language model formulations that do incorporate syntactic knowledge have been developed (Chelba et al., 1997; Charniak et al., 2003) and used in speech recognition (Chelba and Jelinek, 2000) and machine translation (Galley and Manning, 2009), the vast majority of statistical speech recognition and machine translation systems still rely on n-gram language models.

Given a sequence of words (such as a sentence or a speech utterance), an n-gram language model provides the conditional probability of the next word in the sequence given the previous $n-1$ words. This conditional probability model can be estimated using relative frequency estimation from a corpus of word sequences. Let count function $C(\cdot)$ provide the number of times a word sequence is observed in the training corpus of word sequences. Then, Equation 2.4

defines an n-gram language model $\mathsf{P}(\mathbf{e})$:

$$\mathsf{P}(e_n \,|\, e_1 \ldots e_{n-1}) = \frac{C(e_1 \ldots e_n)}{C(e_1 \ldots e_{n-1})} \tag{2.4}$$

Various discounting, backoff, and smoothing techniques have been developed to allow n-gram language models to more elegantly handle data sparsity issues and word sequences not observed during training (Lidstone, 1920; Good, 1953; Jelineck and Mercer, 1980; Katz, 1987; Witten and Bell, 1991; Church and Gale, 1991; Ney et al., 1994; Kneser and Ney, 1995). The n-gram language models we use in Chapter 5 are trained using modified Kneser-Ney smoothing (Chen and Goodman, 1998).

2.2.3 Maximum Entropy Framework

The noisy channel model used in word-based translation is somewhat counter-intuitive, in that its translation model works in a reverse direction. That is, given a document to be translated from language $\mathbf{f}$ to language $\mathbf{e}$ requires the use of word-based translation model $\mathsf{P}(\mathbf{f} \,|\, \mathbf{e})$, which models the translation probability of words in language $\mathbf{f}$ given words in language $\mathbf{e}$. Because of this formulation, the early statistical translation literature confusingly refers to $\mathbf{f}$ as the target language and $\mathbf{e}$ as the source language, despite the fact that the document to be translated is in language $\mathbf{f}$. While researching extensions to word-based translation, Och et al. (1999) changed the direction of the trained word-based translation model from $\mathsf{P}(\mathbf{f} \,|\, \mathbf{e})$ in Equation 2.3 to the more intuitive $\mathsf{P}(\mathbf{e} \,|\, \mathbf{f})$:

$$\hat{e} = \arg\max_{e} \mathsf{P}(\mathbf{e} \,|\, \mathbf{f}) \mathsf{P}(\mathbf{e}) \tag{2.5}$$

Using this mathematically unsound reformulation (Equation 2.5) as the decoding search criterion, Och et al. obtained translation results equivalent in quality to those resulting when using Equation 2.3. Following this finding, Och and Ney (2002) proposed a generalization of the noisy channel model for statistical machine translation; all models informing the translation process, including translation model $\mathsf{P}(\mathbf{f} \,|\, \mathbf{e})$ and language model $\mathsf{P}(\mathbf{e})$, are considered to be feature functions h_m in a maximum entropy (Berger et al., 1996) framework, with weights λ_m:

$$\mathsf{P}(\mathbf{e} \,|\, \mathbf{f}) = \frac{\exp\left[\sum_{m=1}^{M} \lambda_m h_m(e, f)\right]}{\sum_{e'} \exp\left[\sum_{m=1}^{M} \lambda_m h_m(e', f)\right]} \tag{2.6}$$

Equation 2.6 can exactly emulate the behavior of the noisy channel Equation 2.1 when Equation 2.7 and Equation 2.8 are defined as the translation model and language model feature functions, respectively:

$$h_1(e, f) = \log \mathsf{P}(\mathbf{f} \mid \mathbf{e}) \tag{2.7}$$

$$h_2(e, f) = \log \mathsf{P}(\mathbf{e}) \tag{2.8}$$

The denominator in Equation 2.6 normalizes over all possible translations e' for source sentence f. This normalization constant is invariant with respect to the decoding search, and so can be eliminated in Equation 2.9:

$$\hat{e} = \arg\max_{e} \left\{ \exp \left[\sum_{m=1}^{M} \lambda_m h_m(e, f) \right] \right\} \tag{2.9}$$

Values for feature function weights λ_m are typically obtained by optimizing translation quality on a development set for which known translations are available. The optimization method most commonly used is MERT — Minimum Error Rate Training (Och, 2003). During optimization, translation quality is measured by an automatic evaluation criteria such as word error rate, position independent word error rate, BLEU (Papineni et al., 2001), METEOR (Banerjee and Lavie, 2005), or TER (Snover et al., 2006). BLEU, which is essentially an n-gram precision measure, is the most widely used automatic evaluation metric.

This formulation (Equation 2.9) is the most widely used decoding search criterion in current statistical machine translation research. Arbitrary feature functions, including target language model and translation models in both directions, can be defined and incorporated into this framework. This formulation allows an unlimited number of component feature functions to be defined and used, including the definiton of multiple target language models which are all used at translation time.

2.2.4 Phrase-Based Translation

Where the rule-based transfer approaches to translation in Section 2.1 made heavy use of syntax and other linguistic components through hand-crafted rules, the statistical word-based approach

to translation in Section 2.2 deals only with word-form tokens, eschewing any attempt at syntactic analysis.

Wang and Waibel (1998) and Och and Weber (1998) propose enhancing word-based translation with very basic structural information; these approaches use automatic word class clustering techniques to posit *phrases* — contiguous word sequences which may or may not correspond to any linguistic constituent. The alignment template approach (Och et al., 1999; Och, 2002; Och and Ney, 2004) that followed represents the first statistical translation system where the basic unit of translation is the *phrase* rather than the *word*. Other early research examining the use of phrases include Marcu (2001), Venugopal et al. (2003), Watanabe et al. (2003), Zens et al. (2002), and Zens and Ney (2004).

The standard phrase-based model of statistical machine translation (Koehn et al., 2003) incorporates phrase translation models (in both directions), a target n-gram language model, a distortion model (for phrase reordering) and several other models in a maximum entropy log-linear framework (Equation 2.9). The open source Moses (Koehn et al., 2007) statistical machine translation system is the de-facto standard implementation for phrase-based translation. We have contributed implementations of our novel work from Chapter 5 into Moses. Cunei (Phillips and Brown, 2009) is another open source phrase-based translation system; the models in Cunei incorporate additional information about phrase context, drawing substantially from the example-based machine translation literature.

Phrase-based translation incorporates minimal structural knowledge lacking in word-based translation — contiguous strings of adjacent words are grouped into phrases. However, no information is incorporated regarding the hierarchical structure of language. Hierarchical phrase-based translation (Chiang, 2005, 2007) uses the formal framework of a weighted synchronous context-free grammar, SCFG (Aho and Ullman, 1969), to allow phrases with wildcard gaps. A related formalism is translation using Inversion Transduction Grammars, ITGs (Wu, 1997); an ITG can be considered to be a special case of a hierarchical SCFG. Hierarchical SCFG models are formally syntactic, but do not use linguistic phrase structure categories; the only nonterminal categories are wildcard category X, and S, which allows sequences of (possibly hierarchical) phrases to be combined. Hierarchical phrase-based decoding typically operates via bottom-up parsing; Watanabe et al. (2006) develop an alternate incremental decoding algorithm. Schwartz (2008b), Joshua (Li et al., 2009a), Moses (Hoang et al., 2009), cdec (Dyer et al., 2010), and Jane (Vilar et al., 2010) are open-source implementations for hierarchical phrase-based translation.

Neither phrase-based nor hierarchical phrase-based translation take explicit advantage of the syntactic structure of either source or target language. The phrases (with gaps allowed in the case of hierarchical phrases) defined in these translation models may or may not correspond to any linguistic constituent. Koehn et al. (2003) considered whether restricting phrase-based translation models to use only syntactically well-formed constituents might improve translation quality but found such restrictions failed to do so.

2.3 Incorporating Syntax into Statistical Machine Translation

Statistical machine translation techniques typically incorporate many statistical models via a maximum entropy framework (Equation 2.9). Attempts to incorporate linguistic syntax into statistical translation may focus on the translation model, the target language model, or other models such as the reordering model. This section surveys various approaches which incorporate syntax into statistical translation.

2.3.1 Syntax through Reranking

The most straightforward techniques for incorporating syntax reorder the n-best list produced by a decoder according to one or more syntactic features. Chelba et al. (1997) define a syntactic language model based on dependency parsing, and use this model to rerank n-best transcription results from an automatic speech recognition system. Collins et al. (2005) similarly use a discriminative syntactic language model to rerank speech recognition results.

Syntactic language models have also been explored in conjunction with tree-based translation models such as Yamada and Knight (2001). Charniak et al. (2003) use a phrase-structure parser (Charniak, 2000) as a syntactic language model to rescore the output of a tree-to-string translation system (Yamada and Knight, 2002).

Och et al. (2004) use a reranking approach to explore a number of syntactic features for phrase-based translation. Hasan et al. (2006) and Wang et al. (2007) rerank n-best lists according to several different models of syntactic well-formedness, including supertagging (Bangalore and Joshi, 1999) with lightweight dependency analysis (Bangalore, 2000).

It is relatively straightforward to use a syntactic model to rerank an n-best list. However, reranking is not the ideal mechanism to incorporate knowledge of syntax. Even when n is very large, the hypothesized translations present in an n-best list represent only a tiny fraction

of the hypothesis space considered during translation. A syntactic model used to rerank can only choose among the n translations present in the n-best list. A syntactic model incorporated as a feature function (Equation 2.9) in a maximum entropy framework is much more powerful influence; such a model can influence the seach process directly, allowing many more (hopefully more syntactically well-formed) hypotheses to be explored. The following sections examine syntactic models that have been directly incorporated in this way.

2.3.2 Syntax in the Translation Model

Rule-based transfer approaches to machine translation (Section 2.1) analyze or parse source sentences, resulting in a structural representation of the sentences. This structural representation is transfered, using rules, into an equivalent target language structural representation, from which the target language string is generated.

Tree-based approaches to statistical machine translation follow this same general outline, but utilize automatically generated weighted rules for parsing, transfer, and generation. Significant research has examined the extent to which syntax can be usefully incorporated into statistical tree-based translation models. Various research has examined the use of source language syntax (tree-to-string models), source and target language syntax (tree-to-tree models), and target language syntax (string-to-tree models).

As a notational issue, when we use the terms *source language* and *target language*, they are used in the obvious sense (a text to be translated begins in the source language; after translation it is in the target language), not in the inverted (noisy channel) sense. Much of the statistical tree-based translation literature uses *source* and *target* in the inverted sense (see Section 2.2.1). Our usage of *tree-to-string* and *string-to-tree* is in the straightforward sense (string-to-tree translation takes as input a source language string and produces a target language tree), and is therefore opposite the usage of such authors as Yamada and Knight (2001) and Graehl and Knight (2004) who use the noisy channel terminology.

We define tree-to-string, tree-to-tree, and string-to-tree translation models according to three factors: the input format of the source language sentence, and the respective structural formats of the source language and target language components in the model's synchronous transfer process. Tree-to-string translation requires the source language text to already be parsed into syntactic trees. In tree-to-string translation models, the source language components of transfer rules are tree fragments, and the target language components are flat target language string

fragments. Tree-to-tree models are similar to tree-to-string models except in the target language components of the translation models, which use target language tree fragments. String-to-tree models accept unparsed source language text. String-to-tree translation parses the source text using a synchronous grammar which represents the target language components as tree fragments.

Tree-to-Tree Translation Models

Tree-to-tree translation makes use of a set of related synchronous grammar formalisms for modelling pairs of source and target language trees. These related grammar formalisms have been used to construct decoders, such as Abeillé et al. (1990), that accept a parsed source language tree and produce a parsed target language tree.

Synchronous tree adjoining grammars, STAGs (Shieber and Schabes, 1990; Shieber, 2004; Zhang et al., 2007), extend the well-establish tree adjoining grammar (TAG) formalism (Joshi, 1985) to allow for simultaneous derivation of a source tree and a target tree. The formal generative power of tree adjoining grammars is mildly context-sensitive. Using an STAG model, the decoder of Cowan et al. (2006) operates in a discriminative framework instead of the more commonly used noisy channel or maximum entropy frameworks. Translation models over dependency treelets have also been used for tree-to-tree translation (Ding and Palmer, 2005; Quirk et al., 2005).

Eisner (2003) presents synchronous tree substitution grammars, STSGs, a closely related formalism which lacks the tree adjunction operation of STAGs. Some authors, particularly those with experience in example-based translation, use the term Data-Oriented Translation (DOT) for certain types of translation using STSGs (Poutsma, 1998, 2000, 2003; Hearne and Way, 2003; Hearne, 2005; Bod, 2007).

Synchronous tree-insertion grammars, STIGs (Nesson et al., 2006; DeNeefe and Knight, 2009), similarly lack adjunction and also place restrictions on the form of elementary trees.

Ding and Palmer (2004b) define synchronous dependency insertion grammars, SDIGs, as a formal generative model which allows alignment between structurally divergent dependency trees; such dependency insertion grammars are weakly context-free equivalent, providing context-free formal generative power. The SDIG formalism builds on alignment techniques for non-isomorphic parallel dependency trees (Ding et al., 2003; Ding and Palmer, 2004a). An improved algorithm for inducing SDIGs from parallel corpora is developed in Ding and

Palmer (2005), along with a polynomial time tree transduction algorithm for translation decoding. Further refinements to grammar induction and decoding algorithms are presented in Ding and Palmer (2006). Ding (2006) reports translation quality using SDIG for Chinese-English on par with standard phrase-based models as implemented in Pharaoh (Koehn, 2004).

Alshawi (1996a) defines monolingual head automata for analyzing and generating dependency trees in the context of a probabilistic tree-to-tree transfer decoder. Closely related translation models (Alshawi, 1996b; Alshawi et al., 1997, 1998, 2000) utilize head automata as tree transducers to directly perform translation decoding. These probabilistic head automata are closely related to traditional finite-state automata, but are capable of handling some context-free phenomena.

DeNeefe et al. (2007) observe that phrase-based translation models often contain phrases not found in tree-based translation models, and explore how tree-based models can be enhanced to address this deficiency; Chiang (2010) generalizes tree-to-tree rule extraction (Lavie et al., 2008; Hanneman and Lavie, 2009) into fuzzy tree-to-tree rule extraction, increasing the number of extractable rules. Chiang allows the decoder to apply any learned STSG rule at any point, learning when and to what extent to trust the available syntatic rules through feature optimization.

Tree-to-String Translation Models

Tree-to-string translation proceeds from a parsed source language sentence via statistical tree transform operations, resulting in a target language string. Liu et al. (2006) define a tree-to-string alignment template model, merging ideas from Och and Ney's phrase-based alignment templates into tree-based translation; speed enhancements through efficient incremental decoding are developed in Huang and Mi (2010). Huang et al. (2006) present one system which operates by internally parsing the source text, then translating through a tree transducer.

In tree-to-tree and tree-to-string translation, the quality of the source language parse can be a factor that affects the quality of the translations (Quirk and Corston-Oliver, 2006). Such approaches typically begin translation with a single parse for each input sentence; by contrast Liu et al. (2007) define a tree-to-string model which accepts a set of parse trees for each input sentence. Zhang et al. (2008) take a similar approach providing sets of source parse trees in tree-to-tree translation. Using sets of parse trees instead of a single best parse mitigates the problem of parse quality by allowing some parse ambiguity to propigate, enabling a broader

range of possible translations. A packed forest (Billot and Lang, 1989) is a compact representation of many or all of the possible parse tree derivations for a parsed input sentence. Mi and Huang (2008) and Mi et al. (2008) extend the work of Liu et al. (2007) to accept packed forest representations of input sentence parses. Liu et al. (2009) enable packed forest input in tree-to-tree translation.

String-to-Tree Translation Models

Synchronous context-free grammars, SCFGs (Aho and Ullman, 1969), were developed in the context of translating computer programming languages into compiled machine instructions. Like context-free rules, rules in SCFGs contain a left-hand side (consisting of a single nonterminal symbol) and a right-hand side (consisting of one or more nonterminal and/or terminal symbols); SCFG rules also contain a second parallel right-hand side. The terminals in the first right-hand side come from the source language; the terminals in the second right-hand side come from the target language. SCFGs can be directly used as translation models; by parsing a source language sentence with dotted chart parsing (Earley, 1968, 1970; Chappelier and Rajman, 1998), a corresponding target language tree is constructed.

Bracketing inversion transduction grammars in binary-normal form, ITGs (Wu, 1997), represent a special case of SCFGs. Such grammars are interesting in that they contain only two nonterminal types, a start symbol (typically called S) and a dummy symbol (typically called X). Hierarchical phrase-based translation (Chiang, 2005, 2007) similarly uses a formally syntactic SCFG translation model using nonterminals S and X without using linguistic constituent categories. Syntax augmented machine translation (SAMT) grammars (Zollmann and Venugopal, 2006) expand on hierarchical models by incorporating enhanced nonterminal category types, in addition to more traditional phrase-structure nonterminal categories; these enhanced nonterminal types include incomplete category constituents (such as C_1/C_2 and $C_2\backslash C_1$ types found in CCG, Combinatorial Category Grammar (Ades and Steedman, 1982)) and concatenated category constituents (such as $C_1 + C_2$). Recent work has shown that parsing-based machine translation using syntax-augmented hierarchical translation grammars with rich nonterminal sets can demonstrate substantial gains over hierarchical grammars for certain language pairs (Baker et al., 2009). Hoang and Koehn (2010) present a similar approach, extracting hierarchical SCFG rules and rules containing syntactic constituent categories. Hoang and Koehn differ from Zollmann and Venugopal by allowing syntactic constituent categories only on the

source side (the target side categories are restrained to X); rather than posit incomplete or contatenated constituent category labels, this approach simply assign X as the source side label in cases where an extracted rule does not fit neatly within the training source parse tree.

Marton and Resnik (2008) and Chiang et al. (2008) present an alternate mechanism for augmenting hierarchical translation models by incorporating features that represent soft constraints based on linguistic syntax; even more structural features are added to string-to-tree and hierarchical translation models by Chiang et al. (2009). Yet another approach (Zhou et al., 2008) incorporates syntax into hierarchical models though the use of prior derivation models applied using tree kernels.

Yamada and Knight (2001) present a string-to-tree translation model; as each source language input sentence is decoded (Yamada and Knight, 2002) a corresponding target language syntax tree is constructed. Gildea (2003) presents an extension to Yamada and Knight's tree-to-string model which he further adapts into a tree-to-tree translation model. Galley et al. (2004, 2006) and Imamura et al. (2004) develop a string-to-tree translation models which map source language strings to aligned target language parse trees. Graehl and Knight (2004) examine how tree transducers can be trained for such string-to-tree (and tree-to-tree) translation models.

Like hierarchical translation models, these models are commonly formalized as SCFGs; however, unlike hierarchical models, these string-to-tree grammars utilize linguistically motivated nonterminal categories. Translation decoding proceeds by parsing according to these synchronous grammars. Wang et al. (2007) shows how such models can be generalized through binarization, and Melamed (2004) generalizes translation by parsing to take advantage of multitexts. Where the above string-to-tree techniques operate over synchronous phrase-structure grammars, Marcu et al. (2006) and Shen et al. (2008) translate using a string-to-dependency tree translation model.

String-to-String Translation Models

Phrase-based decoders are formally equivalent to finite-state string transducers (Kumar and Byrne, 2003), and as such make use of string-to-string translation models. When Koehn et al. (2003) forced their phrase-based system to use only those phrases which corresponded to syntactic constituents, translation quality decreased. Since then, various approaches have attempted to improve the quality of phrase-based translation by integrating syntactic knowledge.

Phrase-based translation models are typically trained from word-aligned parallel corpus.

Tinsley et al. (2007a) augment this model through the use of additional phrase alignments obtained from a tree-to-tree alignment (Tinsley et al., 2007b) of the parsed parallel training corpus; the additional phrase pairs extracted using the tree-to-tree alignments are added to the phrase table. Similarly, ITG alignments of parallel training corpora (Zhao and Vogel, 2003; Zhao and Gildea, 2005; Chao and Li, 2007) have also been used when extracting phrase pairs for phrase-based translation (Sánchez and Benedí, 2006; Cherry and Lin, 2007; Saers and Wu, 2009; Haghighi et al., 2009). Mylonakis and Sima'an (2008) and Sima'an and Mylonakis (2008) develop an alternate mechanism for estimating phrase translation model probabilities, making use of an ITG prior model.

The Moses phrase-based decoder allows multiple layers of information to be incorporated as translation model factors (Koehn and Hoang, 2007; Koehn et al., 2007; Hoang and Koehn, 2009; Hoang, 2011). Lexicalized tree adjoining grammar (LTAG) supertags (Bangalore and Joshi, 1999) and CCG supertags (Clark and Curran, 2004) have been successfully employed (Hassan et al., 2007) as factors within Moses (Birch et al., 2007).

The vast majority of research seeking to incorporate syntactic models as translation features have focused on the translation model rather than the language model. As discussed above, this research includes tree-to-string translation models which require parsed input, string-to-tree translation models which posit syntactic structure on hypothesized translations, and tree-to-tree translation models which both require parsed input and posit syntactic structure on hypothesized translations. Syntactic models have been used to guide the construction of phrase-based translation models, and as a mechanism to augment those translation models with syntactic super-tags. In contrast, the novel work we develop in this dissertation maintains a standard (non-syntactic) phrase-based translation model. Instead, we incorporate syntax into the language model.

2.3.3 Syntax in the Language Model

Traditional approaches to language models in speech recognition and statistical machine translation focus on the use of n-grams, which provide a simple finite-state model approximation of the target language. Speech recognition and phrase-based translation decoding algorithms process input incrementally, from the beginning of a sentence to the end. The finite-state nature of n-gram language models allows for straightforward integration of such models into incremental speech recognition and phrase-based translation algorithms.

Phrase-based translation using factored translation models can take advantage of n-gram

language models over factors other than surface word forms (Koehn and Hoang, 2007; Hoang and Koehn, 2009). As an example, the Czech-English translation system presented in Bojar and Hajič (2008) makes use of high-order 7-gram language models over morphological tags; this is in addition to the use of traditional n-gram language models over surface word forms. In a similar spirit, Hassan et al. (2007) and Birch et al. (2007) apply n-gram language models over supertag sequences to phrase-based translation. Riezler and Maxwell (2006) use n-gram language models defined over functional structures from the Lexical Functional Grammar (LFG) formalism (Kaplan and Bresnan, 1982) in the context of a tree-to-tree transfer-based translation system. Graham and van Genabith (2010) examine LFG n-gram models in more detail, and describe how they could be used in phrase-based and hierarchical translation.

The use of n-gram language models in conjunction with tree-based translation models is much more difficult than the use of n-gram language models in phrase-based translation. Phrase-based translation generates partial translation hypotheses incrementally, from the beginning of a translation to the end; this allows for easy incorporation of n-gram language models. Translation systems with tree-based translation models, however, typically formulate partial translation hypotheses in a bottom-up parsing fashion. While n-gram language models can be incorporated into tree-based translation techniques (Chiang, 2007), doing so requires additional bookkeeping and heuristics not required for n-gram LM incorporation with phrase-based translation.

Chelba et al. (1997) proposed that syntactic structure could be used as an alternative to n-grams in language modelling, defining a syntactic dependency language model. Chelba and Jelinek (1998) define a related model to operate in an incremental manner; this model is incorporated into incremental speech recognition (Chelba and Jelinek, 2000). Shen et al. (2008) incorporate a language model based on target language dependency structure with a string-to-dependency tree translation model.

Post and Gildea (2008) investigate the integration of parsers as syntactic language models during binary bracketing transduction translation (Wu, 1997); under these conditions, both syntactic phrase-structure and dependency parsing language models were found to improve oracle-best translations, but did not improve actual translation results. Post and Gildea (2009) report the use of tree substitution grammar parsing for language modelling, but do not use this language model in a translation system. Galley and Manning (2009) adapt a standard probabilistic dependency parser to provide a language model score based on the 1-best partial dependency

parse at each node in a phrase-based translation system.

2.3.4 Syntax in Other Models

Zens et al. (2004) and Yamamoto et al. (2008) use source language syntax in conjunction with an inversion transduction grammar to impose constraints on phrase-based reordering. The syntactic cohesion features of Cherry (2008) encourages the use of syntactically well-formed translation phrases, also in the context of standard phrase-based translation. The syntax-driven reordering model of Ge (2010) uses syntax-driven features to influence word order within standard phrase-based translation. These approaches are fully orthogonal to our proposed incremental syntactic language model, and could be applied in concert with our work.

2.4 Conclusion

Substantial research efforts have explored how syntactic information can be used to improve the results of machine translation. In Section 2.3, we surveyed the major techniques for incorporating syntax into modern statistical machine translation. We now briefly review these techniques, and show where our work fits into this context.

Syntax through Reranking It is relatively straightforward to rerank an n-best list produced by a translation system according to one or more syntactic features. Reranking is far from ideal; even when n is very large, the hypothesized translations present in an n-best list represent only a tiny fraction of the hypothesis space considered during translation. A syntactic model incorporated directly (into the translation model, language model, or reordering model) is much more powerful, influencing the seach process directly and allowing many more (hopefully more syntactically well-formed) hypotheses to be explored.

Syntax in the Translation Model The vast majority of statistical machine translation techniques which directly incorporate syntax make use of tree-based syntactic translation models or factored phrase-based translation models. This research includes tree-to-string translation models which require parsed input, string-to-tree translation models which posit syntactic structure on hypothesized translations, and tree-to-tree translation models

which both require parsed input and posit syntactic structure on hypothesized translations. Syntactic models have been used to guide the construction of phrase-based translation models, and as a mechanism to augment those translation models with syntactic super-tags. These techniques typically do not make use of syntax in the language model, instead using traditional n-gram language models.

Syntax in the Language Model Far less research has examined how syntax can be incorporated into language models. Syntactic language models were developed initially to model syntactic structure in the context of speech recognition. A small amount of more recent research has attempted to integrate syntactic language models (based primarily on dependency grammar parsers and tree substitution grammar parsers) into tree-based and phrase-based translation, with mixed levels of success.

Our research in this dissertation develops a novel technique for integrating an incremental phrase-structure parser as a syntactic language model in phrase-based translation. The novel work we develop is similar in spirit to the work of Galley and Manning (2009), but ours provides a much more natural fit by using a parser designed from the start to parse incrementally. Our incremental syntactic language model calculates language model scores over all available phrase structure parses at each translation hypothesis rather than simply the 1-best parse.

Syntax in Other Models A few researchers have used syntactic constraints to inform the reordering models in statistical phrase-based translation. These approaches are fully orthogonal to our proposed incremental syntactic language model, and could be applied in concert with our work.

The remaining content of this dissertation is structured as follows. In Chapter 3, we describe the training process by which the model parameters for our syntactic language model are estimated from a corpus of manually annotated phrase structure trees; this model is an incremental syntactic language model which fits into the family of linear-time dynamic programming parsers described by Huang and Sagae (2010). In Chapter 4 we present the core algorithms required to parse and score target language text using our trained incremental syntactic language model. Finally, in Chapter 5 we show how our incremental syntactic language model can be directly and easily integrated into a phrase-based translation decoder.

Chapter 3

Learning an Incremental Parsing Model from Phrase Structure Trees

The primary novel contribution of this dissertation is the development of an incremental syntactic language model for use in statistical phrase-based machine translation. A syntactic language model is a formal probabilistic model of the syntax, or structure, of a human language. This chapter describes the training process by which the model parameters for our incremental syntactic language model are estimated from a corpus of manually annotated phrase structure trees.

The training process for our model requires a training corpus where all phrase structure trees are in a normalized maximally left-recursive form. This normalized maximally left-recursive form is called *right-corner form*, and is a variant of the left-corner form of Johnson (1998). We therefore present our syntactic language model training process in two parts: a sequence of tree transformations (Section 3.2) designed to transform the trees in the Wall Street Journal treebank corpus into right corner form, followed by model parameter estimation for a probabilistic context-free grammar from the transformed treebank corpus (Section 3.3). This two part training process is similar to the work of Hockenmaier (2003), which presents tree transformations and model estimation for parsing under Combinatory Categorial Grammar (CCG). In brief, the tree transformation component of our training process transforms the Wall Street Journal Penn Treebank corpus from a set of unnormalized, relatively flat trees into a set of binary-branching depth-bounded trees in a maximally left-recursive normal right-corner form.

Figure 3.1 shows a sample tree from the WSJ treebank corpus before any processing has

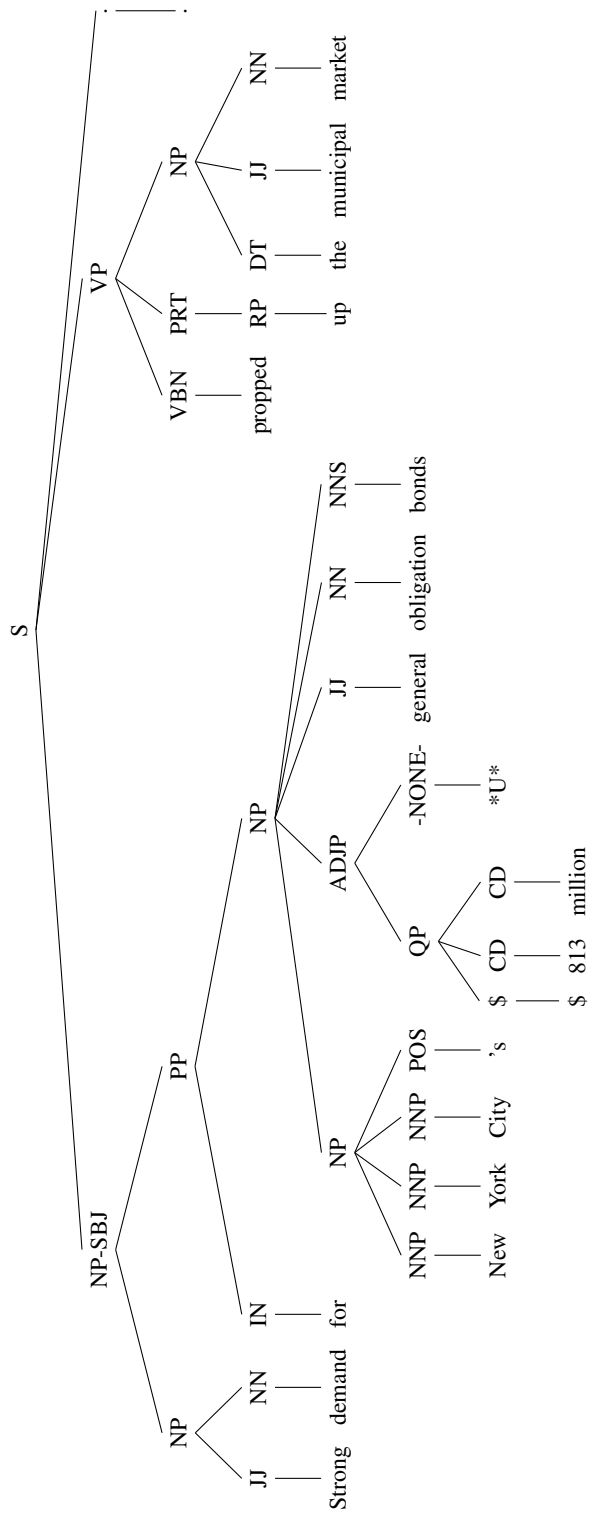


Figure 3.1: Graphical representation of a sample tree from the WSJ treebank, before any processing.

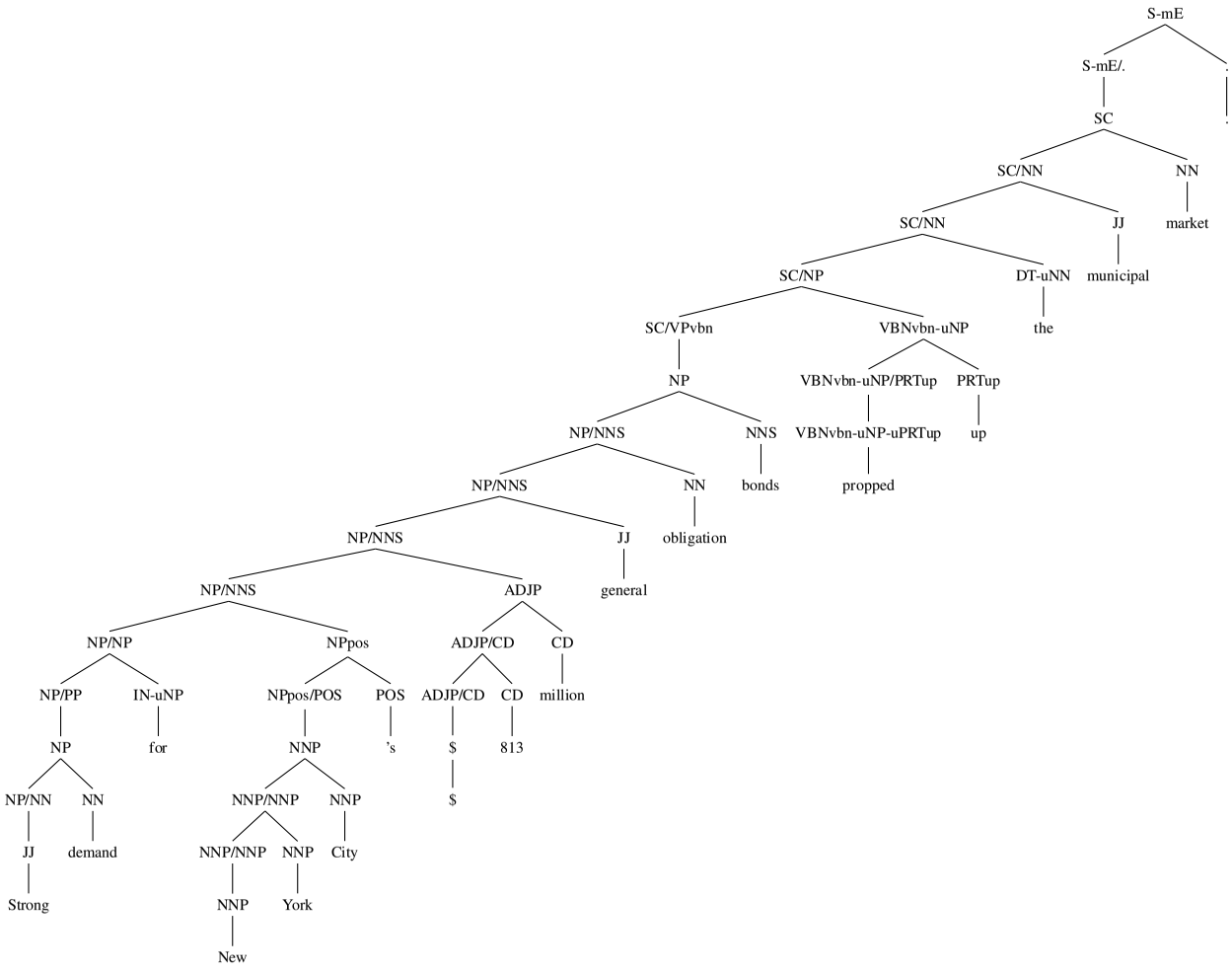


Figure 3.2: Graphical representation of a sample tree from the WSJ treebank, in right-corner form, after all processing.

been performed. Figure 3.2 illustrates this same tree in normal right-corner form, after all processing has been performed. This chapter describes the tree transformation (Section 3.2) and syntactic language model parameter estimation (Section 3.3) procedures necessary for learning an incremental parsing model from a corpus of sentences, manually annotated with phrase structure trees.[1] The Wall Street Journal (WSJ) Penn Treebank corpus (Marcus et al., 1993) is a collection of sentences in the news domain taken from the Wall Street Journal; each sentence in the corpus is annotated with a phrase structure tree that was assigned by a human annotator. This is the corpus we will use to train our incremental syntactic language model.

We now briefly enumerate the tree transformations necessary to train our incremental syntactic language model:

1. **Remove annotation errors** The WSJ treebank contains certain annotation errors which we correct.

2. **Binarization** The WSJ trees are not distributed in a binary-branching form. We attempt to identify the constituent head of high-arity branches, and use that information to rewrite those branches in binary form.

3. **Argument structure** We attempt to identify phrasal argument structure. Each child element identified as a syntactic head is subcategorized to mark its argument structure.

4. **Traces** The WSJ treebank contains empty constituents, annotated following government and binding theory (Haegeman, 1994). We remove empty constituents, along with any unary projections that arise from this removal.

5. **Punctuation** The WSJ treebank annotation scheme utilizes certain (arguably arbitrary) conventions in annotating punctuation. We re-annotate punctuation in such a way as to minimize its effect on our parsing model.

6. **Normalization** All trees are transformed into Chomsky Normal Form (Chomsky, 1963).

[1] The model training processes described here (and numerous variants) were developed and used extensively by members of the University of Minnesota Natural Language Processing group (Miller et al., 2007; Miller and Schuler, 2008a,b,c; Miller, 2009a,b; Miller et al., 2009; Miller and Schuler, 2010; Schuler and Miller, 2005; Schuler et al., 2006, 2008, 2009, 2010; Schuler, 2010; Schwartz et al., 2009, 2011; Wu et al., 2008a,b, 2010), but until now have not been fully described.

7. **Depth annotation & bounding** Our parsing model utilizes a bounded memory store. We identify the depth in this bounded memory store at which each node in a tree will be placed, and annotate the tree nodes with this information. Trees that exceed a specified maximum depth are removed from the training set.

8. **Right-corner transform** We transform trees into a maximally left-recursive right-corner form. This two-part process first flattens right-recursive structure into flat structure, and then transforms these flattened structures into equivalent left-recursive structures.

3.1 Notation

Throughout this chapter, we describe various transforms as applied to phrase structure trees. We will make use of the following notation:

- Roman uppercase letters (A_i) are variables matching constituent labels,
- Roman lowercase letters (a_i) are variables matching terminal symbols,
- Greek lowercase letters (α_i) are variables matching entire subtree structure,
- Roman letters followed by colons, followed by Greek letters (A_i:α_i) are variables matching the label and structure, respectively, of the same subtree, and
- ellipses (. . .) are taken to match zero or more subtree structures, preserving the order of ellipses in cases where there are more than one

3.2 Tree Transformations

While models for common top-down (Earley, 1968) and bottom-up (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967) parsing techniques can be trained directly on such treebanks, ours cannot. The limited pre-processing requirement for their parsing techniques is that trees be transformed into a canonical binary-branching normal form, typically Chomsky Normal Form, CNF (Chomsky, 1963).[2] In order to parse with the Hierarchical Hidden Markov Model

[2] A grammar is in CNF if and only if all grammar production rules are of the form $A \rightarrow BC$ or $A \rightarrow \alpha$, where A, B, and C represent grammar nonterminal constituent labels, and α represents any non-empty terminal language symbol.

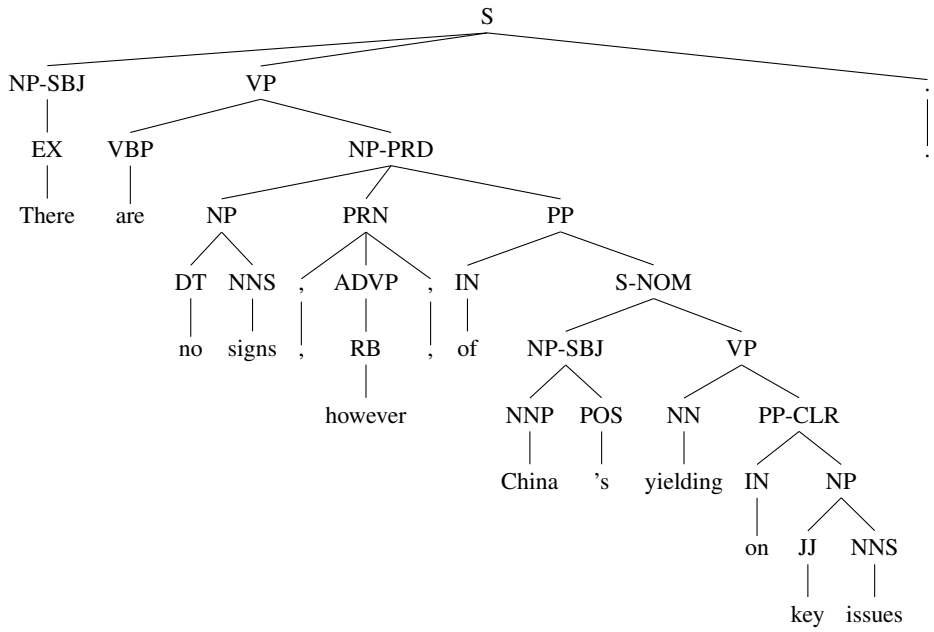


Figure 3.3: Graphical representation of a sample phrase structure tree in the WSJ treebank, before any processing.

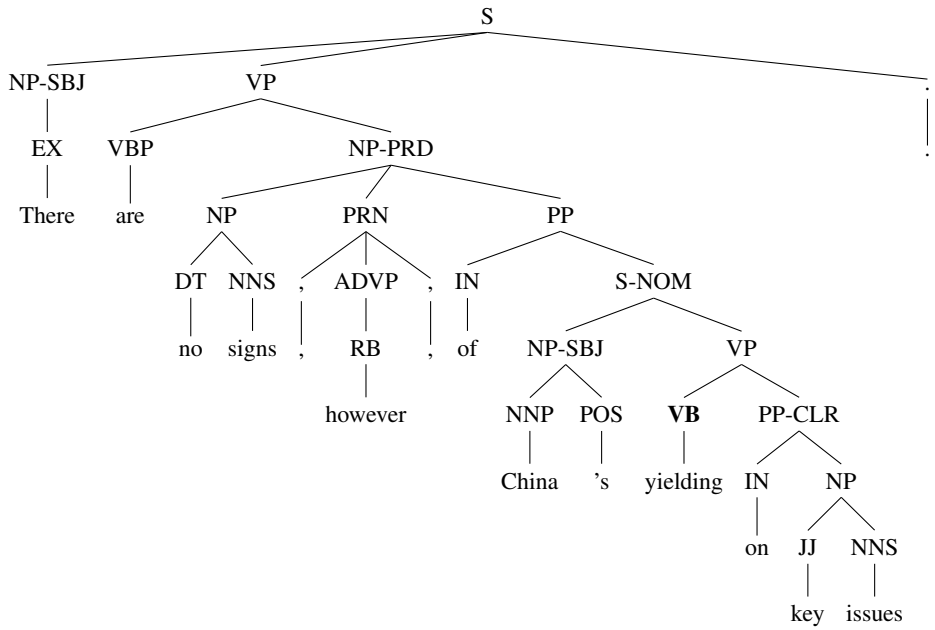


Figure 3.4: Graphical representation of a sample phrase structure tree in the WSJ treebank, after pre-processing to fix annotation errors (see Appendix A.1). Changes from the tree in Figure 3.3 are shown in **bold**.

(HHMM) (Murphy and Paskin, 2001) parsing algorithms we present in Chapter 4, more extensive transformations are required. The end result of these transformations will be a depth-limited, right-corner transformed grammar in Chomsky Normal Form.

The WSJ treebank corpus is distributed as a collection of 25 non-overlapping sections (sections 0–24). The syntactic language model which we define is trained on this treebank. Following standard practice in the parsing literature, we train on sections 2–21. The WSJ treebank contains some annotation errors; we begin processing the treebank by correcting certain of these annotation errors (see Appendix B.2). Figure 3.3 illustrates a sample phrase structure tree from the WSJ treebank, before any processing has been performed. We will use this tree as a running example throughout this chapter, to illustrate the effects of our transformations. Figure 3.4 shows this same tree after pre-processing to fix annotation errors. In Figure 3.4 and in the Figures that follow, nodes in the tree which have changed are highlighted in bold.

3.2.1 Binarization

The WSJ treebank trees are not distributed in a binary-branching normal form. The subtree shown in Figure 3.4 dominating *no signs , however , of China 's yielding on key issues* and headed by NP-PRD has three immediate children (NP, PRN, PP). It is convenient for all training trees to be transformed into a canonical binary-branching normal form. To this end, after all input training trees have been pre-processed to correct annotation errors, these trees are binarized. At the same time, all terminal tokens are lowercased. Punctuation marks are retained.

The binarization script (Appendix B.3) contains over 180 tree transform rules. These rules define how to binarize tree fragments when dealing with conjunctions, noun phrases, temporal noun phrases, verb phrases, sentential projections, adjectival and adverbial phrases, prepositional phrases, empty phrase structure categories, punctuation, and terminal symbols. The selection of head constituents in these rewrite rules is similar to the Magerman-Black head rules (Magerman, 1995). The most important tree transform rules are presented in more detail in Appendix A.2. Figure 3.5 depicts the sample tree from Figure 3.4 after the binarization transform.

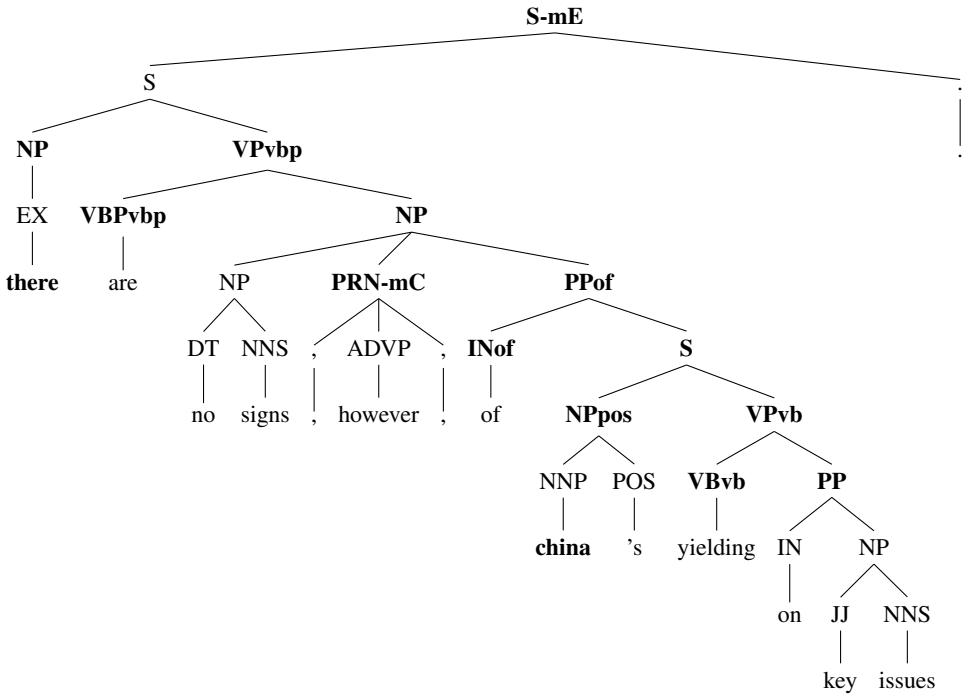


Figure 3.5: Graphical representation of a sample phrase structure tree in the WSJ treebank, after binarization (see Appendix A.2). Changes from the tree in Figure 3.4 are shown in **bold**.

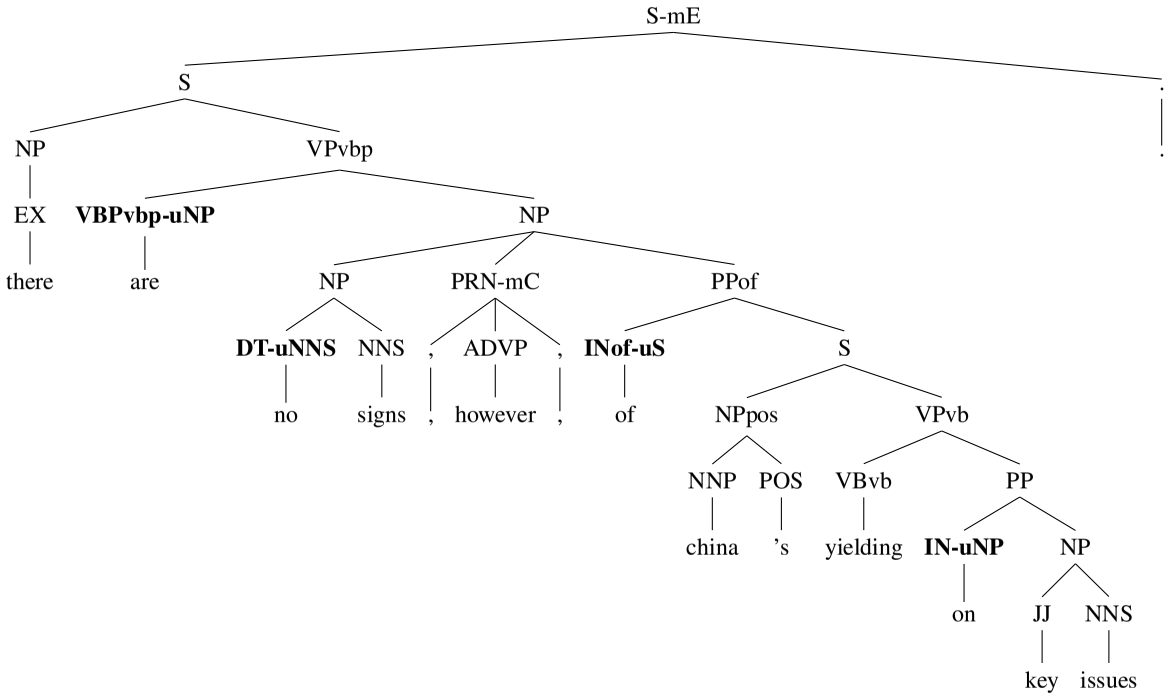


Figure 3.6: Graphical representation of a sample phrase structure tree in the WSJ treebank, after argument structure processing (see Appendix A.3). Changes from the tree in Figure 3.5 are shown in **bold**.

3.2.2 Argument Structure

In any binary subtree fragment, one child is typically the head while the other typically serves as either an argument or an adjunct.[3] The trees resulting from the linguistically motivated binarization processing in Section 3.2.1 are next examined to identify heads and arguments. Each child identified as a syntactic head is subcategorized to mark its argument category. The subcategory X-uY is used to mark that head category X will take an (as yet unsatisfied) argument of category Y. For example, in Figure 3.6 the subtree (NP $\rightarrow$ DT NNS) is transformed into (NP $\rightarrow$ DT-uNNS NNS).

Figure 3.6 shows the binarized phrase structure tree from Figure 3.5 after argument processing has been performed. Implementation details for argument structure processing are documented in Appendix A.3.

3.2.3 Traces

The WSJ treebank contains empty constituents, annotated following government and binding theory (Haegeman, 1994). After processing trees for argument structure (Section 3.2.2) we remove these empty constituents, along with any unary projections that arise from this removal (following Schuler et al. (2010)). In the case of empty constituents representing traces, the extracted category label is annotated onto the lowest nonterminal dominating the trace using the suffix -gX_{extr} where X_{extr} is the category of the extracted constituent. To preserve grammaticality, this annotation is then passed up the tree and eliminated when a $wh-$, topicalized, or other moved constituent is encountered, in a manner similar to that used in Head-driven Phrase Structure Grammar (Pollard and Sag, 1994), but without affecting branching structure.

Figure 3.7 shows the phrase structure tree from Figure 3.6 after this processing has been performed. The processing rules are listed in Appendix A.4, with code implementation listed in Appendix B.5.

3.2.4 Punctuation

The Wall Street Journal treebank annotation scheme utilizes certain (arguably arbitrary) conventions in annotating punctuation. These conventions can lead to undesirable artifacts in the

[3] Adjuncts are sometimes also called modifiers.

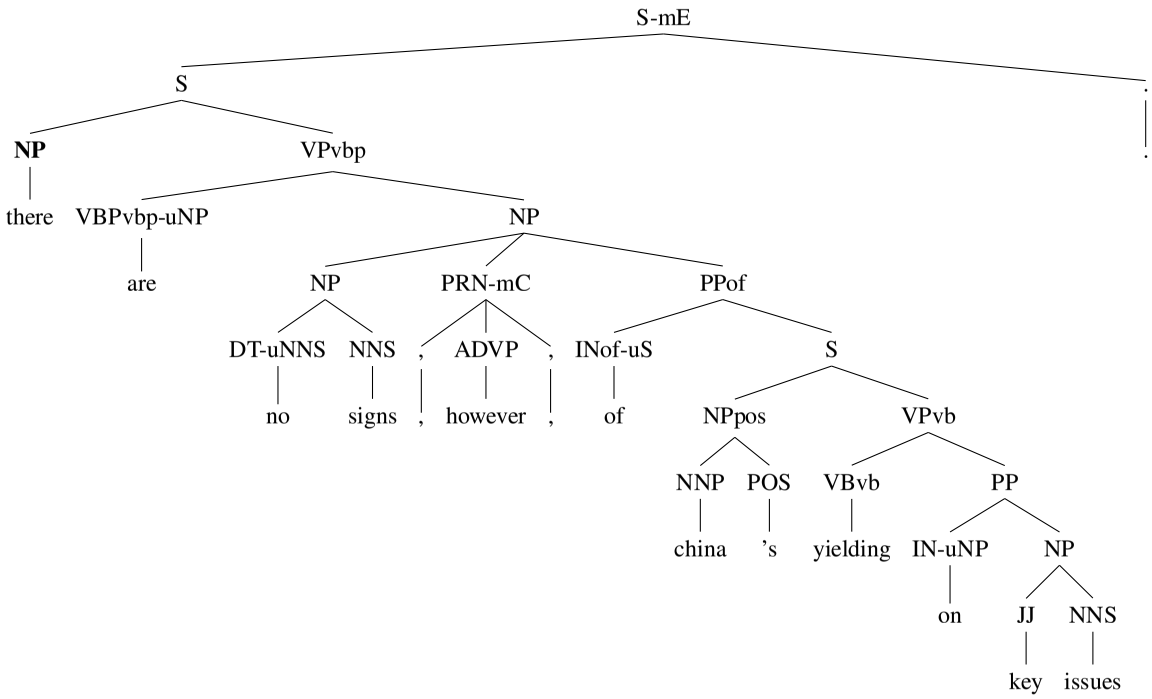


Figure 3.7: Graphical representation of a sample phrase structure tree in the WSJ treebank, after handling traces (see Appendix A.4). Changes from the tree in Figure 3.6 are shown in **bold**. Here, the unary chain (NP $\rightarrow$ EX $\rightarrow$ there) present in Figure 3.6 has been reduced to (NP $\rightarrow$ there).

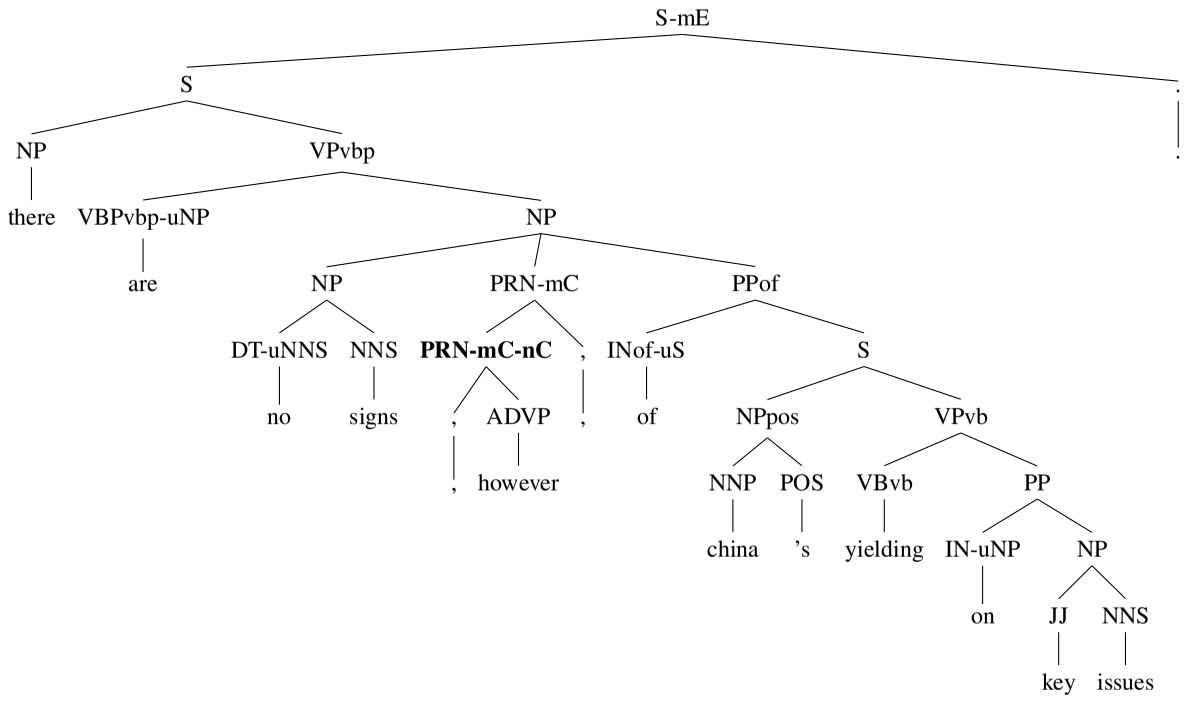


Figure 3.8: Graphical representation of a sample phrase structure tree in the WSJ treebank, after handling punctuation (see Appendix A.5). Changes from the tree in Figure 3.7 are shown in **bold**.

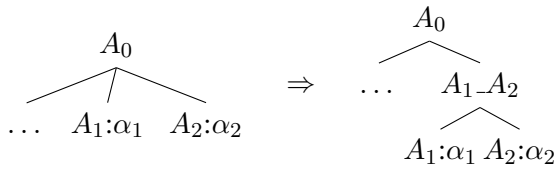
resulting annotated trees which are both psycholinguistically implausable and potentially detrimental during parsing. As an example, in Figure 3.8 the annotation of commas in the tree causes the word *however* to be artificially center-embedded by lone punctuation marks. In general, branching structure for punctuation can be difficult to motivate on linguistic grounds, because punctuation marks do not have lexical projections or argument structure in most linguistic theories.

The HHMM which we use as a model of language is a pycholinguistic model with an explicit bounded memory store. It is highly questionable to account for punctuation marks in such a psycholinguistic model as composable elements in the memory store. We therefore re-annotate punctuation in the treebank in such a way as to minimize the effect of punctuation with regard to the memory store by allowing punctuation to rise in the tree. This re-annotation, in conjunction with the right-corner transform, serves to minimize the effect of punctuation on the memory store during parsing.[4]

Figure 3.8 shows the phrase structure tree from Figure 3.7 after punctuation processing has been performed. The punctuation processing rules are listed in Appendix A.5, with code implementation listed in Appendix B.6.

3.2.5 Normalization

The procedures described in the subsections above (and the corresponding rules as implemented in the relevant appendices) remove about 65% of super-binary branches from the treebank training trees. All remaining super-binary branches are processed as described in Schuler et al. (2010). Tree fragments with super-binary branching arity are decomposed into right-branching structures by introducing intermediate nodes, each with a label concatenated from the labels of its children, delimited by underscores.



[4] Note that this process does **not** remove punctuation. Rather, tree structure is modified to minimize the effect of punctuation on the parser's memory constraints. Punctuation has been shown to have significant effects during translation, and it is important that our syntactic language model robustly handle punctuation.

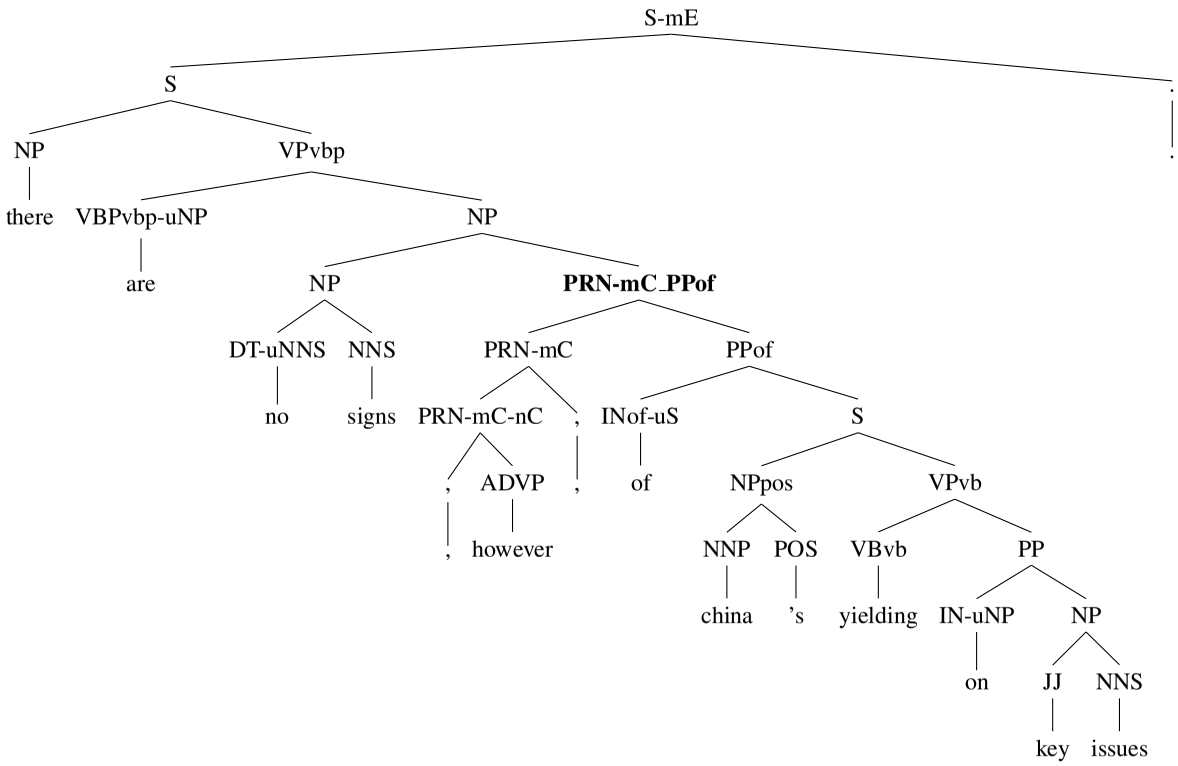


Figure 3.9: Graphical representation of a sample phrase structure tree in the WSJ treebank, after conversion to Chomsky Normal Form (see Appendix A.6). Changes from the tree in Figure 3.8 are shown in **bold**.

This label concatenation process removes all remaining super-binary structure from the training trees, leaving all trees in Chomsky Normal Form. The steps to perform this normalization are shown in Appendix A.6. While technically, no super-binary structure remains, the concatenated labels at the root of the newly binarized branches represent the exact same super-binary structure as existed before; this process is equivalent to leaving super-binary branches intact and using dot rules in parsing (Earley, 1970).

Figure 3.9 shows the phrase structure tree from Figure 3.8 after normalization to Chomsky Normal Form has been performed. Details for performing normalization are in Appendix A.6 with implementation details listed in Appendix B.7.

3.2.6 Depth Annotation and Depth Bounding

After normalization (Section 3.2.5) all training trees from the WSJ treebank are binary-branching trees in Chomksy Normal Form. These trees will be used to estimate probability values in the HHMM for use in parsing. The nodes in a training tree can be mapped onto hidden variables in the HHMM. In order to correctly estimate probability values for the HHMM hidden variables, it is necessary to annotate each node in the training trees with the depth in the HHMM of the hidden variable to which the node can be mapped. During model training, this data will allow depth-specific models to be correctly estimated.

Given the root element of a training tree in Chomsky Normal Form, Algorithm 3.1 annotates each node in the tree with the depth at which it will reside when the tree is mapped onto an HHMM. The algorithm also determines the *side* of the node; that is, whether it is a left or right child of its parent. Unary children are considered to have a side value of left. The nonterminal category of each node annotated with this depth and side information.

The HHMM is a bounded memory model of language. In any use of the HHMM, a concrete value must be selected as the maximum depth (or bound) of the memory model.Schuler et al. (2010) found that 99.54% of trees in the WSJ treebank training section can be accounted for with a maximum depth of 4 memory elements. This coverage increases to 99.96% if punctuation is discarded. For model training and in all experiments, we set the HHMM maximum memory depth to 4 memory elements. Training trees that require more than 4 elements in the HHMM memory store are discarded.

Figure 3.10 shows the first phrase structure tree in the WSJ treebank after this processing has been performed. Details are shown in Appendix A.7, with code listed in Appendix B.8.

Algorithm 3.1 Given the root element of a tree in Chomsky Normal Form, annotate each node in the tree with the depth at which it will reside when the tree is mapped onto an HHMM. Additionally, annotate each child node as a right (R) or left (L) child of its parent. The root element is considered to be a left child.

```
Require: Input tree is in Chomksy Normal Form
  function ANNOTATE-BRANCH-DEPTH(node)
    if node==root then
      node.depth ← 1
      node.side ← L
    end if
    if node.children.size > 0 then
      if node.side == R then
        node.children[0].depth ← node.depth+1
      else
        node.children[0].depth ← node.depth
      end if
      node.children[0].side ← L
      ANNOTATE-BRANCH-DEPTH(node.children[0])
      if node.children.size > 1 then
        node.children[1].depth ← node.depth
        node.children[1].side ← R
        ANNOTATE-BRANCH-DEPTH(node.children[1])
      end if
    end if
  end function
```

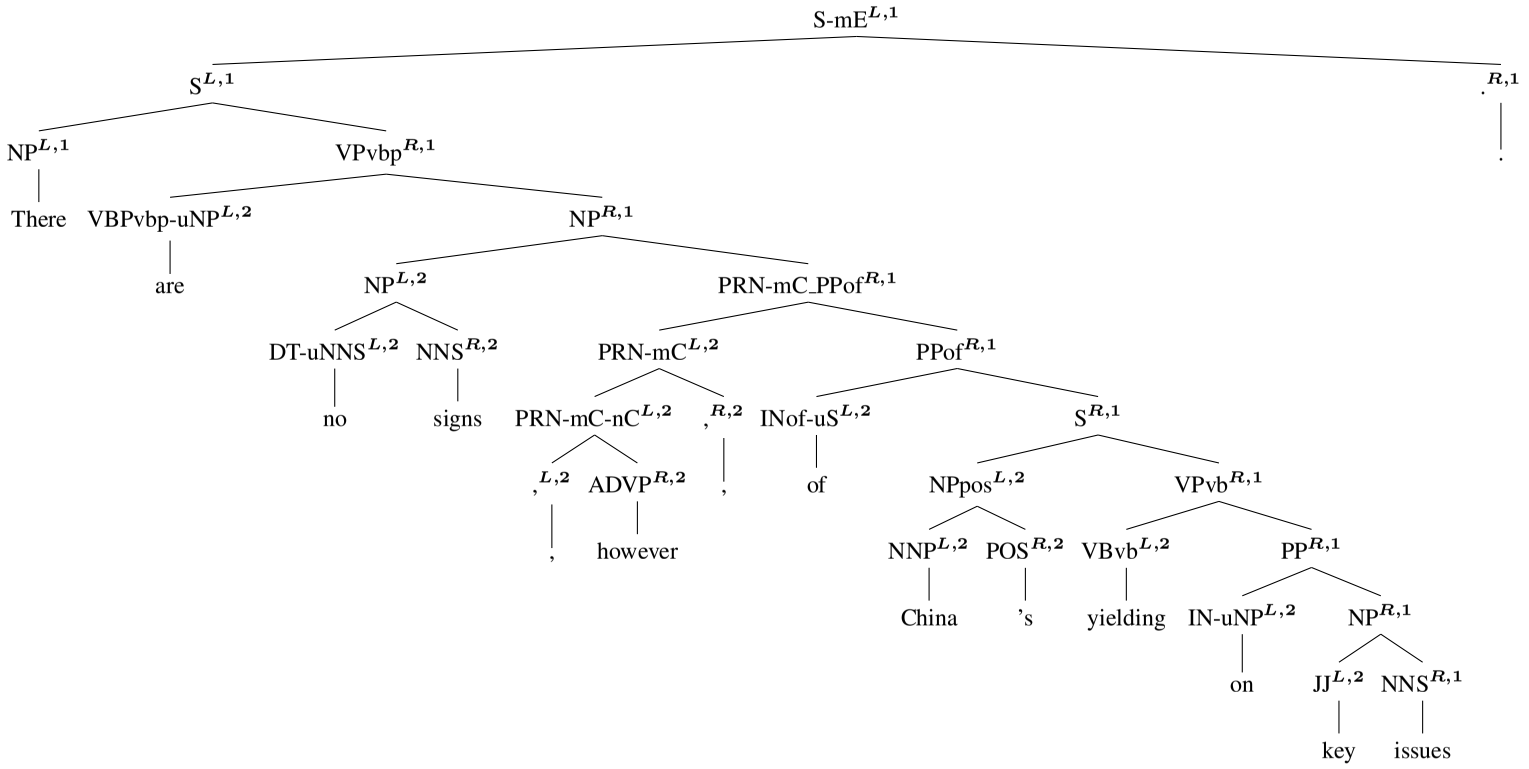


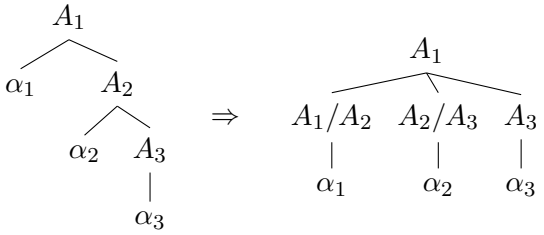
Figure 3.10: Graphical representation of a sample phrase structure tree in the WSJ treebank after depth and side annotation. The tree structure shown here is identical to the CNF tree in Figure 3.9; the labels at each node in this tree are also identical, with the depth and side information annotated as superscript. These superscript changes from the tree in Figure 3.9 are shown in **bold**.

3.2.7 Right Corner Transform

Following our work in (Schuler et al., 2010), we define the right-corner transform as a process in which right-recursive structure is first flattened and is then replaced with left-recursive structure.

Flatten Right-Recursive Structure

1. Transform right-recursive sequences of completed constituents into flat sequences of incomplete constituents



2. Transform right-recursive sequences of completed and incomplete constituents into flat sequences of incomplete constituents

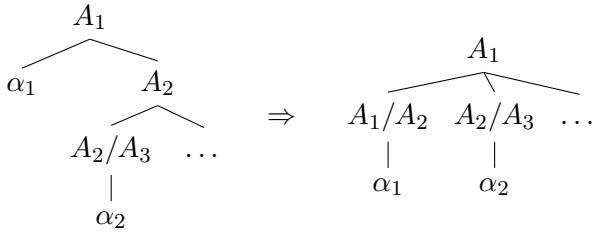
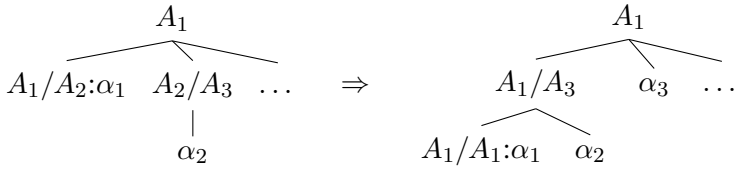


Figure 3.11 shows the sample tree from Figure 3.10 after transformations to flatten right-recursive structure.

Transform Flattened Right-Recursive Structure into Left-Recursive Structure

3. Transform flattened sequences into left-recursive sequences



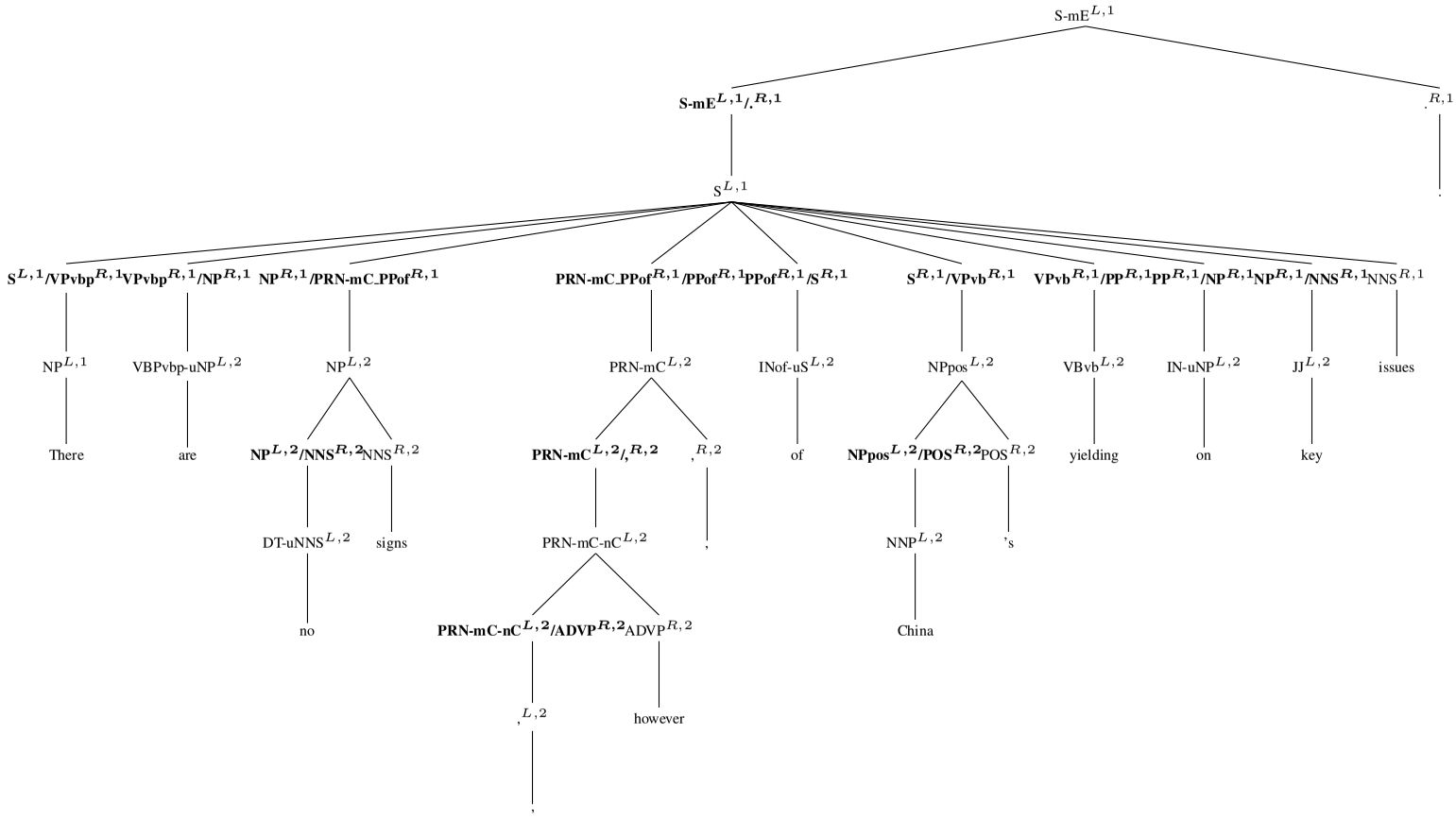


Figure 3.11: Graphical representation of a sample phrase structure tree in the WSJ treebank after the first stage of the right corner transform. Changes from the tree in Figure 3.10 are shown in **bold**.

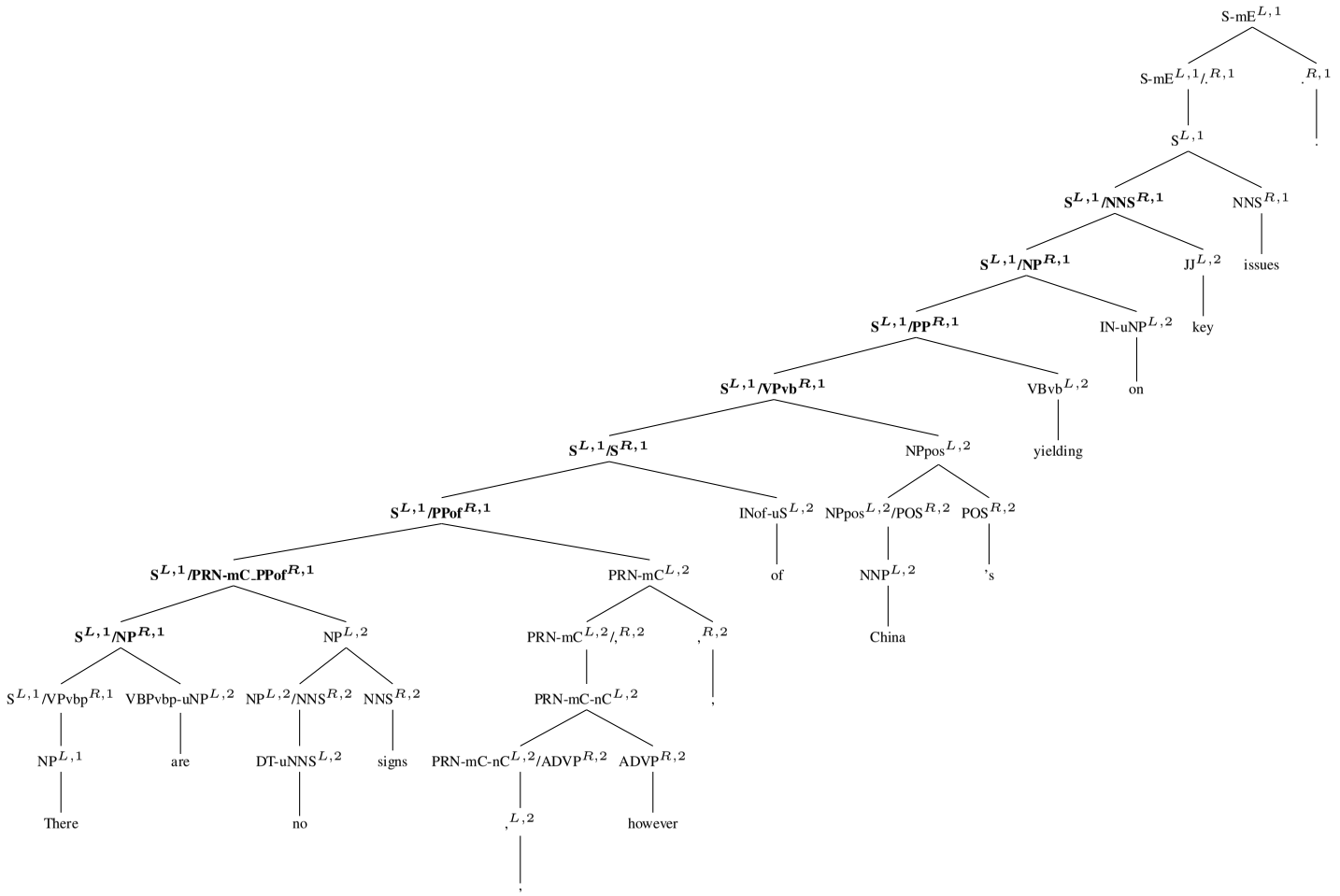


Figure 3.12: Graphical representation of a sample phrase structure tree in the WSJ treebank after the right corner transform is complete. Changes from the tree in Figure 3.11 are shown in **bold**.

Figure 3.12 shows the sample tree from Figure 3.11 after flattened structures have been transformed into left-recursive (right-corner) form.

3.3 Estimating Model Parameters from Transformed Trees

The preceding Section 3.2 defined various transformation processes over the phrase structure trees of the Wall Street Journal Treebank. Given a set of transformed phrase structure trees (WSJ sections 2-21) as a training set of positive examples of valid English sentences, our goal is to define a formal model of the English language and estimate parameter values for this model from the training data.

3.3.1 Formal Definition: Probabilistic Context-Free Grammar

A probability model of a language can be defined in terms of a probabilistic context-free grammar, or PCFG, (Booth, 1969). A context-free grammar G is defined by the tuple $\langle N, \Sigma, R, S \rangle$. A probabilistic context free grammar is a CFG augmented by the probability model θ_D defined by function D.

$\boldsymbol{N}$ A finite set of nonterminal symbols representing all valid phrase-structure categories

$\boldsymbol{\Sigma}$ A finite set of terminal symbols representing all valid words

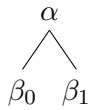
$\boldsymbol{R}$ A finite set of production rules of the form $\zeta \rightarrow \gamma$, where ζ is an element of N and γ is a sequence of elements of $N \bigcup \Sigma$. We will be dealing exclusively with context-free grammars in Chomsky Normal Form. In such grammars, elements of R are restricted to those of the form $\alpha \rightarrow \beta_0 \beta_1$ or $\alpha \rightarrow x$, where α, β_0 and β_1 are elements of N, and x is an element of Σ.

$\boldsymbol{S}$ Those elements of N that which may serve as the start symbols for the grammar

$\boldsymbol{D}$ A function that assigns a real value between 0 and 1 to each production rule in R, such that for each left-hand side (ζ), the sum over all right-hand sides (γ) sum to 1

A phrase structure tree representing a valid sentence in a grammar G represents a sequence of rule applications from R. Given this view of phrase structure trees, a treebank of phrase structure trees in language $\mathcal{L}$ can be used to obtain N, Σ, R, S, and θ_D for a grammar G of $\mathcal{L}$.

Each binary branch in a tree represents the application of a binary rule $\alpha \rightarrow \beta_0 \beta_1$. Each unary branch in a tree represents the application of a unary rule $\alpha \rightarrow x$.



$$\Rightarrow \quad \alpha \rightarrow \beta_0 \beta_1 \tag{3.1}$$

$$\Rightarrow \quad \alpha \rightarrow x \tag{3.2}$$

A traversal of all depth-bounded training phrase structure trees (from Section 3.2.6) using Equations 3.1 and 3.2 results in a multi-set R' of rules. Set R is obtained by removing all duplicate elements from R'. Set N is the set of all nonterminals α that appear on the left-hand side of any rule in R unioned with the set of all nonterminals β that appear on the right-hand side of any binary rule. Set Σ is the set of all terminals x that appear on the right-hand side of any unary rule. The set S is the set of nonterminals which appear at the root of any training tree. Count function C provides the number of times a rule occurs in R'. Given R' and C, probability model θ_D can be obtained via relative frequency estimation:

$$\mathsf{P}_{\theta_D}(\alpha \rightarrow \gamma \,|\, \alpha) = \frac{C(\alpha \rightarrow \gamma)}{\sum_{\gamma'} C(\alpha \rightarrow \gamma')} \tag{3.3}$$

If the goal is to train a PCFG over trees in right-corner form, one might wonder why we train on the depth-bounded phrase structure trees resulting from Section 3.2.6 rather than the right-corner form trees resulting from Section 3.2.7. Much of our prior work indeed did train the incremental syntactic language model in exactly this way, from a corpus of right-corner form trees.

However, we choose to make use of an equivalent, but more general option. Schuler (2009) defines a model transform which converts a standard probabilistic context-free grammar into an equivalent right-corner grammar. In our case, the end result is the same. Had we chosen to train a PCFG directly over the right-corner form trees resulting from Section 3.2.7, that grammar would be equivalent to the grammar obtained by first training on the depth-bounded phrase structure trees resulting from Section 3.2.6 and then performing the right-corner model transform of Schuler (2009).

In other cases, we may have a sophisticated PCFG available whose training procedure cannot easily incorporate the right-corner transform. By defining our training process in terms of the right-corner model transform rather than the right-corner tree transform, we allow for the direct use of any arbitrary PCFG.

3.3.2 Reformulation as Component Probability Models

Chapter 4 defines a grammar transform over the PCFG from Section 3.3.1. When defining that grammar transform, it is most convenient for the original PCFG to be reformulated as a set of component prior and conditional probability models. These component models can be directly estimated from the training trees using relative frequency estimation. Together, they contain all information required for the PCFG original formulation from Section 3.3.1. The component probability models are listed below:

θ_C Prior probability model over all nonterminal symbols

θ_P Prior probability model over preterminal symbols. Preterminals are that subset of nonterminal symbols which appear on the left-hand side of unary rules

θ_{Cr} Prior probability that a nonterminal symbol may be a start symbol. Only start symbols may serve as the root of a tree

θ_{CC} Conditional probability model of the nonterminal children (right-hand side) of a rule given the parent (left-hand side). Because unary rules have only terminal children, for the purpose of this model, unary branching rules ($\alpha \rightarrow x$) are treated as binary branching rules with empty nonterminal values ($-$) for both children ($\alpha \rightarrow --$)

θ_{Pw} Conditional probability model of the parent preterminal (left-hand side) of a unary branching rule given the child word (right-hand side)

θ_{Pc} Conditional probability model of a preterminal given itself (as a nonterminal). In our models this values is deterministic with value 1. In other formulations this model could potentially have more interesting values.

During parsing we may be interested in the conditional probability of a terminal given a preterminal. This value can be easily calculated using Bayes's rule given the above models.[5] Appendix A.8 illustrates the use of the scripts which train these models using relative frequency estimation from the depth-limited training trees.

3.3.3 Part of Speech Model

Any robust model of language needs a mechanism to handle words not seen during training. Words not present in the training corpus are termed out-of-vocabulary, or OOV. Given an OOV word, it is necessary to define a probability distribution over the possible preterminal categories. We use a preterminal, or part-of-speech (POS), model defined using a decision tree. This implementation is an existing component of ModelBlocks. Any other POS model implementation could be substituted, provided it can output probability values in the appropriate ModelBlocks file format. Implementation details for estimating the POS model are listed in Appendix A.9.

3.4 An Incremental Parsing Model

In statistical machine translation, the target language model plays a critical role guiding the order and choice of target language words. Statistical machine translation research typically follows widespread practice in speech recognition by using n-gram language models to model the prior probability of a contiguous sequence of words.

Syntactic parsing may help produce more grammatical output by better modelling structural relationships and long-distance dependencies. Bottom-up and top-down parsers typically require a completed string as input; this requirement makes it difficult to incorporate these parsers into phrase-based translation, which generates hypothesized translations incrementally, from left-to-right.[6]

On the other hand, incremental parsers (Roark, 2001; Henderson, 2004; Schuler et al., 2010; Huang and Sagae, 2010) process input in a straightforward left-to-right manner. We observe that incremental parsers, used as syntactic language models, provide an appropriate algorithmic

[5] Technically, the prior probability of a terminal is also needed in this calculation. However, at parse time all terminals are observed values. As such, this term may be safely dropped, resulting in the prior probability of a terminal being proportional to the conditional probability of the preterminal given the terminal, divided by the prior probability of the preterminal.

[6] While not all languages are written left-to-right, we will refer to incremental processing which proceeds from the beginning of a sentence as left-to-right.

match to incremental phrase-based decoding. The model we have defined in this chapter, by the nature of the right-corner form over which it is defined, fits naturally into an incremental parsing framework.

We now present a general definition of incremental parsing. An incremental parser processes each token of input sequentially from the beginning of a sentence to the end. Tokens will commonly be words, but may be morphemes (used in psycholinguistics sentence processing research) or characters (used in Chinese parsing). After processing the t^{th} token in string $\mathbf{e}$, an incremental parser has some internal representation of possible hypothesized (incomplete) trees, $\boldsymbol{\tau_t}$. The syntactic language model probability of a partial sentence $e_1 \dots e_t$ is defined:

$$\mathsf{P}(e_1 \dots e_t) = \sum_{\tau \in \boldsymbol{\tau_t}} \mathsf{P}(e_1 \dots e_t \,|\, \tau)\mathsf{P}(\tau) \tag{3.4}$$

In practice, a parser may constrain the set of trees under consideration to $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t}}$, that subset of analyses or partial analyses that remains after any pruning is performed. An incremental syntactic language model can then be defined by a probability mass function (Equation 3.5) and a transition function δ (Equation 3.6). Any combination of grammar and parser which implement these two functions can serve as an incremental syntactic language model.

$$\mathsf{P}(e_1 \dots e_t) \approx \mathsf{P}(\tilde{\boldsymbol{\tau}}_t) = \sum_{\tau \in \tilde{\boldsymbol{\tau}}_t} \mathsf{P}(e_1 \dots e_t \,|\, \tau)\mathsf{P}(\tau) \tag{3.5}$$

$$\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1}) \rightarrow \tilde{\boldsymbol{\tau}}_t \tag{3.6}$$

The equations above provide a general definition of an incremental syntactic language model. In Chapter 4 that follows, we provide a much more detailed and specific definition of one particular incremental syntactic language model. We will define our specific incremental syntactic language model in terms of an incremental Hierarchical Hidden Markov Model parsing algorithm. Our parsing algorithm is defined in terms of a right-corner form grammar obtained from the PCFG defined in Section 3.3 above. Chapter 4 provides a very detailed formal description of the right-corner model transform and the HHMM parsing algorithm. In Chapter 5, we show how our incremental syntactic language model can be directly integrated into phrase-based machine translation.

Chapter 4

Algorithms for Parsing with an Incremental Syntactic Language Model

The core idea we present in this dissertation is a method for using an incremental syntactic language model to guide a machine translation decoder as it translates sentences into a target language. The probabilitistic phrase structure grammar defined in Chapter 3 is a necessary component, but is not sufficient to completely define an incremental syntactic language model. Given a sentence, we need some mechanism to posit and rank possible phrase structure trees. Because the phrase-based machine translation algorithm generates target language words incrementally, we would like the mechanism we define to be capable of processing not just complete sentences, but partially complete sentences, where the final words of the sentence are not yet available. In this chapter, we use a Hierarchical Hidden Markov Model — HHMM (Murphy and Paskin, 2001) — to define the computational mechanism in our incremental syntactic language model which satisfies these desiderata.

The phrase structure grammar presented in detail in Chapter 3 represents the syntactic structure of sentences in terms of phrase structure trees. Given a sentence in the language, such a probabilistic model assigns a probability value between zero and one to every possible phrase structure tree. In other words, the model describes the probability that the syntactic structure of a given sentence can be correctly described by a particular phrase structure tree.

This chapter presents the formal computational mechanism of our incremental syntactic language model in terms of detailed parsing algorithms for an HHMM. Readers of this work who are interested primarily in our novel contribution of how an incremental syntactic language model can be integrated into statistical phrase-based translation may find much of the material presented in this chapter to be at a considerably greater level of detail than they may find necessary to understand the primary contributions of this dissertation. For such readers who choose to skim the remainder of this chapter, the most important highlights of the previous chapter and of this chapter are summarized below.

Sections 3.2–3.3 A probabilistic context-free grammar over trees in right-corner form can be estimated from a transformed corpus of phrase structure trees.

Section 3.4 An incremental parser processes each token of input sequentially, from the first word in a sentence to the last. After processing the t^{th} token in string $\mathbf{e}$, an incremental parser has some internal representation of possible hypothesized (incomplete) trees, $\boldsymbol{\tau_t}$. Generically, any incremental parser and grammar which together implement an appropriate probability mass function $\mathsf{P}(\tilde{\boldsymbol{\tau}}_t)$ (Equation 3.5 on page 55) and transition function $\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1})$ (Equation 3.6 on page 55) together define an incremental syntactic language model.

Section 4.1 Hidden Markov Models can be used to model unobserved (or hidden) states that underlie sequences of observed data. We choose a Hierarchical Hidden Markov Model (shown in Figure 4.3 on page 61) to model the unobserved (right-corner form) syntactic structure that underlies an observed sequence of words.

Section 4.2 We formally redefine the right-corner model transform of Schuler (2009) in terms of the probabilistic context-free grammar we trained in Section 3.3. The resulting right-corner PCFG fully defines the model parameters of our HHMM.

Section 4.3 We formally define the algorithms necessary to incrementally parse using our HHMM with our right-corner PCFG. Algorithm 4.12 on page 91 defines the required transition function $\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1})$ declared in Equation 3.6. Algorithm 4.14 on page 93 defines the required probability mass function $\mathsf{P}(\tilde{\boldsymbol{\tau}}_t)$ declared in Equation 3.5. Together, these components constitute our specific incremental syntactic language model.

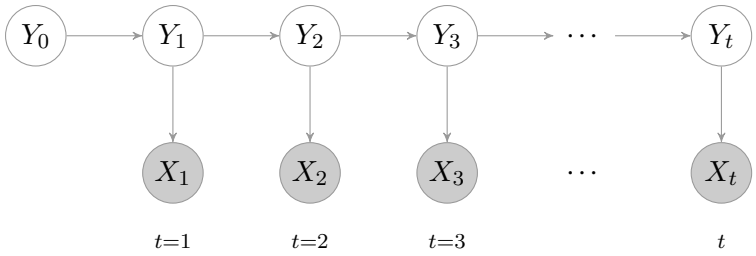


Figure 4.1: A Hidden Markov Model (HMM) models a sequence of $X_{1...t}$ observations where each observation has a corresponding unobserved hidden state $Y_{1...t}$. Each observation X_t is dependent only on its corresponding hidden state Y_t. Each hidden state Y_t is dependent only on the immediately previous hidden state Y_{t-1}.

Section 4.4 In practice, exhaustive parsing with no pruning as defined in Section 4.3 is prohibitively slow. We define important parser optimization techniques which allow the parser to operate with practical runtimes. These techniques include beam pruning, parallel processing, A* uniform cost search, and lazy priority queue expansion.

Chapter 5 We integrate the incremental syntactic language model defined in Sections 4.2–4.4 into the decoding algorithm for statistical phrase-based machine translation.

4.1 Hidden Markov Model

In many problems, it is useful to define models of events that produce sequential output. Such models can be refered to as graphical sequence models. The first prominent use of a graphical sequence model to model and analyze human language was presented by Markov (1913). The resulting family of graphical sequence models that follow his work are called Markov models.

Markov models can be used to address problems in which an unknown (or hidden) state must be postulated for each observation in a sequence of data. The joint probability distribution $\mathsf{P}_{\theta_H}(X_{1...t}, Y_{1...t})$ over the sequence of t observations $X_{1...t}$ and the corresponding sequence of hidden states $Y_{1...t}$ represents a Hidden Markov Model, or HMM. Figure 4.1 depicts a Hidden Markov Model drawn as an unrolled dynamic Bayesian network over the t observations. Each observation is considered to take place at a discrete point in time, and as such each observation with its corresponding hidden state is refered to as a time slice.

Formally, an HMM is defined by tuple $\langle \boldsymbol{Y}, \boldsymbol{X}, \theta_\Pi, \theta_A, \theta_B \rangle$, where $\boldsymbol{X}$ represents the set (or vocabulary) of all possible observations and $\boldsymbol{Y}$ represents the set of all possible hidden state values. The probability distribution over hidden state values at HMM node Y_t is defined by transition model θ_A. Transition model θ_A is defined such that each hidden state Y_t is dependent only on the immediately previous hidden state Y_{t-1}. The initial hidden state Y_0 has no preceding hidden state; as such it is modelled by the prior probability model θ_Π. The probability distribution over the observation state values at HMM node X_t is defined by observation model θ_B. Observation model θ_B is defined such that each observed state X_t is dependent only on the corresponding hidden node Y_t at that time slice. The joint probability of the sequence of observations $X_{1...t}$ and hidden states $Y_{1...t}$ can be defined in terms of the hidden state prior model θ_Π, transition model θ_A, and observation model θ_B:

$$\mathsf{P}(X_{1...t}, Y_{1...t}) = \mathsf{P}_{\theta_\Pi}(Y_0) \sum_{1}^{t} \mathsf{P}_{\theta_A}(Y_t \mid Y_{t-1}) \cdot \mathsf{P}_{\theta_B}(X_t \mid Y_t) \tag{4.1}$$

Hidden Markov Models have been widely used in natural language processing and speech recognition. In many applications of HMMs, the desired goal is to hypothesize the most likely sequence of hidden state values $Y_{1...t}$ given a sequence of observations $X_{1...t}$. This sequence of hidden states can be efficiently calculated using via dynamic programming (Viterbi, 1967). However, we are interested in determining the total probability mass that our model assigns to the observation sequence $X_{1...t}$.

In our task, the observation set $\boldsymbol{X}$ is the vocabulary of words in a human language, specifically the target language into which we wish to translate. The set of hidden state values $\boldsymbol{Y}$ will represent hypothesized structure over a sequence of words in the language. We will precisely define what kind of "hypothesized structure" the hidden states will store over the remainder of this chapter. Regardless of what form we develop for the hypothesized structure, we calculate the probability of an observed sequence of words $X_{1...t}$ given our model of the language, as specified by an HMM. To do so, we must sum over all possible sequences $Y'_{1...t}$ of hidden state values:

$$\mathsf{P}(X_{1...t}) = \sum_{Y'_{1...t}} \mathsf{P}(X_{1...t}, Y'_{1...t}) \tag{4.2}$$

If the values of the hidden states represent the syntax of the modelled target language, Equation 4.2 defines a *syntactic language model* of the target language.

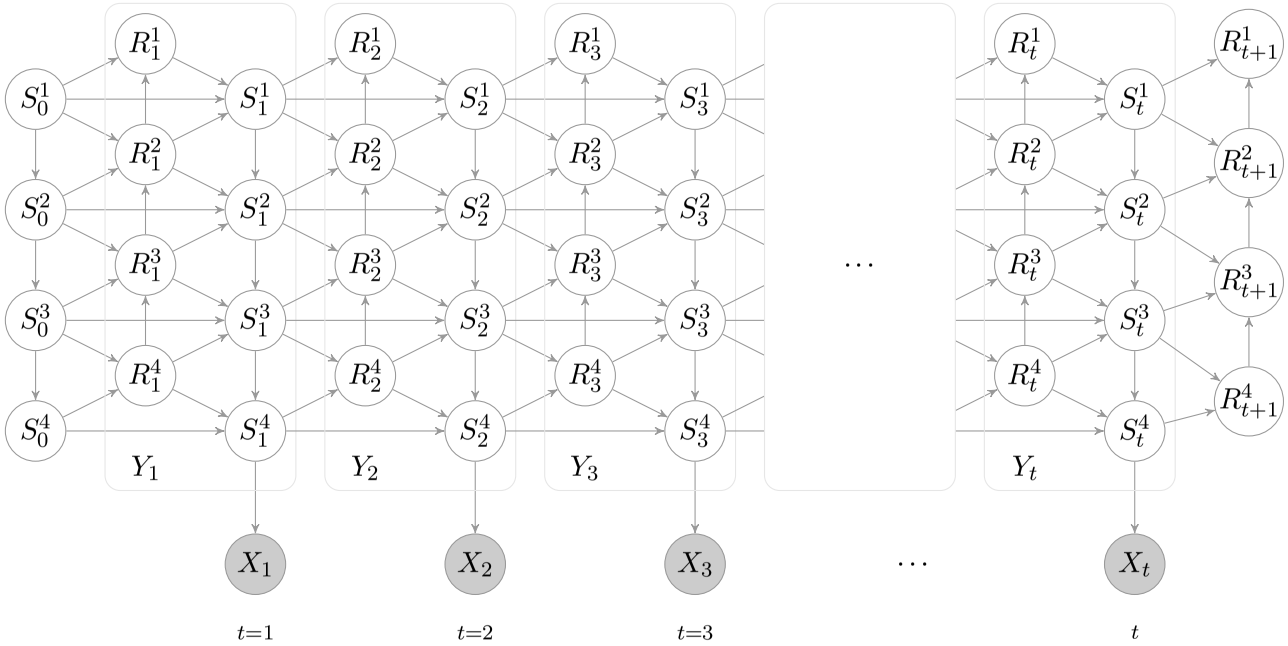


Figure 4.2: A Hierarchical Hidden Markov Model (HHMM) models a sequence of $X_{1...t}$ observations where each observation has a corresponding unobserved hidden state Y_t. Each hidden state Y_t is factored into nodes $S_t^{1...d}$ and $R_t^{1...d}$. The factored nodes in each hidden state Y_t are dependent only on other nodes in Y_t and on nodes in the immediately previous hidden state Y_{t-1}. Observed variables are shown as shaded nodes.

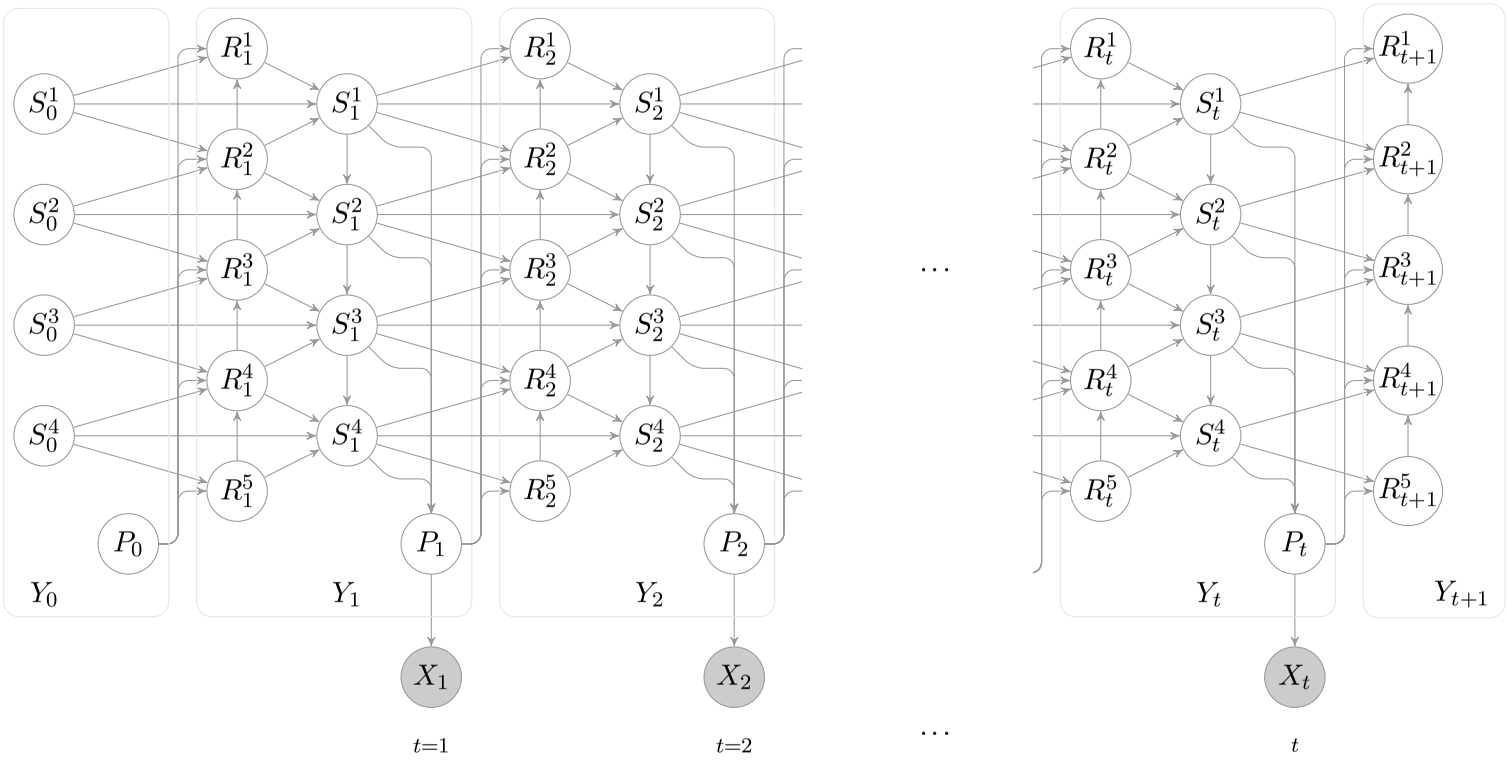


Figure 4.3: The basic HHMM from Figure 4.2 is shown here augmented at each time step with a hidden preterminal node P_t. Each node P_t models the completed phrase structure category that serves as the parent (in a hypothesized phrase structure tree) of the word observed at node X_t.

4.1.1 Hierarchical Hidden Markov Model

This chapter seeks to define a syntactic language model for a target language. To do so we define an HMM where the domain of hidden states encompasses a specific type of phrase structure trees. To efficiently represent structured data in the hidden domain of an HMM, each hidden state can be explicitly factored (Fine et al., 1998) into a Hierarchical Hidden Markov Model, or HHMM. As with HHMs, inference in an HHMM can be accomplished in linear time (Murphy and Paskin, 2001). We follow Murphy and Paskin (2001) in our representation of HHMMs as unrolled Bayesian networks.

Figure 4.2 depicts the standard architecture of a Hierarchical Hidden Markov Model. The HHMM models a sequence of $X_{1...t}$ observed word tokens. Each observation has a corresponding unobserved hidden state Y_t, modelling a probability distribution over possible syntactic representation values. Each hidden state Y_t is factored into component parts, denoted as nodes $S_t^{1...d}$ and $R_t^{1...d}$.The factored nodes in each hidden state Y_t are dependent only on other nodes in Y_t and on nodes in the immediately previous hidden state Y_{t-1}.

We augment this standard HHMM architecture by explicitly factoring the preterminal of each observed word token into a hidden node P_t. Each node P_t models the completed phrase structure category that serves as the parent (in a hypothesized phrase structure tree) of the word observed at node X_t. This augmented HHMM is shown in Figure 4.3.

4.2 Right-Corner Model Transform

Using the model transform of Schuler (2009), a PCFG can be transformed into the specific type of right-cornerized, depth-limited PCFG used to parse with our HHMM. Section 3.3 defines a PCFG in Chomsky Normal Form. The rules for a PCFG trained on transformed WSJ trees are generated as shown in Appendix A.8, with a selection shown in Figure 4.4. The ModelBlocks file format we utilize can store probability values directly, or as count values which can be used to calculate probabilities. The model transform process requires a PCFG model file in the latter form. This model file provides values for the count function $C(\cdot)$, from which probability values can be calculated by simple normalization. The various lines provide the count values over the following:

C $C_{\theta_C}(\alpha^{side,depth})$ for all binary rules $\alpha^{side,depth} \rightarrow \beta_0^{L,depth_0}\beta_1^{R,depth}$

and unary rules $\alpha^{side,depth} \rightarrow x$

$\boldsymbol{P}$ $C_{\theta_P}(\alpha)$ for all unary rules $\alpha^{side,depth} \rightarrow x$

$\boldsymbol{Cr}$ $C_{\theta_{Cr}}(\alpha^{side,depth})$ for all nonterminals α at the root of a training tree

$\boldsymbol{CC}$ $C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0^{L,depth_0}\beta_1^{R,depth})$ for all rules. Binary branching rules $\alpha^{side,depth} \rightarrow \beta_0^{L,depth_0}\beta_1^{R,depth}$ are counted directly. As mentioned previously, unary branching rules have only terminal children, so for the purpose of this model such rules ($\alpha^{side,depth} \rightarrow x$) are treated as binary branching rules with empty nonterminal values ($\alpha^{side,depth} \rightarrow --$) for children $\beta_0^{L,depth_0}$ and $\beta_1^{R,depth}$

$\boldsymbol{Pw}$ $C_{\theta_{Pw}}(\alpha \rightarrow x)$ for all unary rules $\alpha^{side,depth} \rightarrow x$

$\boldsymbol{Pc}$ $C_{\theta_{Pc}}(\alpha^{side,depth})$ for all unary rules $\alpha^{side,depth} \rightarrow x$

Using the binary branching rule counts $C_{CC}(\cdot)$, the depth-specific conditional probability of the left and right childrens' categories can be calculated, given the side and category of the parent. Recall that $C_{CC}(\cdot)$ includes counts for unary rules, where each unary rule $\alpha^{side,depth} \rightarrow x$ is treated as a binary branching rule with empty child categories ($\alpha^{side,depth} \rightarrow --$). In the following equations, summation over any variable marked $\prime$ (such as X') indicates summing over all values of that variable.

$$\mathsf{P}_{\theta_{CC}}(\beta_0, \beta_1 \mid \alpha, side, depth) = \frac{C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0{}^{L,depth_0}\beta_1^{R,depth})}{\sum_{\beta_0',\beta_1'} C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0'{}^{L,depth_0}\beta_1'{}^{R,depth})} \tag{4.3}$$

By further marginalizing over all category values for the left child, the depth-specific conditional probability of the right child's category can be calculated, given the side and category of the parent:

$$\mathsf{P}_{\theta_{CC}}(\beta_1 \mid \alpha, side, depth) = \frac{\sum_{\beta_0',depth_0} C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0'{}^{L,depth_0}\beta_1^{R,depth})}{\sum_{\beta_0',\beta_1',depth_0} C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0'{}^{L,depth_0}\beta_1'{}^{R,depth})} \tag{4.4}$$

Likewise, by marginalizing over all category values for the right child, the depth-specific conditional probability of the left child's category can be calculated, given the side and category

C : ADJP^{L,1} = 148
C : ADJP^{L,2} = 2543
C : ADJP^{L,3} = 2559
C : ADJP^{L,4} = 447
C : ADJP^{L,5} = 1
C : ADJP^{R,1} = 4484
C : ADJP^{R,2} = 3126
C : ADJP^{R,3} = 380
C : ADJP^{R,4} = 13
...
C : NN-tmp^{L,1} = 1
C : NN-tmp^{L,2} = 33
C : NN-tmp^{L,3} = 22
...
CC ADJP^{L,2} : JJR^{L,2} JJ^{R,2} = 7
CC ADJP^{L,2} : JJR^{L,2} NN^{R,2} = 2
...
CC NP^{R,1} : DT-uJJ^{L,2} JJ^{R,1} = 81
CC NP^{R,1} : DT-uNNPS^{L,2} NNPS^{R,1} = 77
CC NP^{R,1} : DT-uNNP^{L,2} NNP^{R,1} = 1199
CC NP^{R,1} : DT-uNNS^{L,2} NNS^{R,1} = 2011
CC NP^{R,1} : DT-uNN^{L,2} NN^{R,1} = 10845
...
Cr : S-mE^{L,1} = 29403
...
Cr : SINV-mE^{L,1} = 1658
...

...
P : INof-uSprovbg = 547
P : INthat = 280
P : INthat-uS = 4082
P : INthat-uSC = 13
P : IT = 554
P : JJ = 56276
P : JJ-uNP = 45
...
Pc ADJP^{L,1} : ADJP = 18
Pc ADJP^{L,2} : ADJP = 286
Pc ADJP^{L,3} : ADJP = 122
Pc ADJP^{L,4} : ADJP = 14
Pc ADJP^{L,5} : ADJP = 1
Pc ADJP^{R,1} : ADJP = 1292
Pc ADJP^{R,2} : ADJP = 1487
Pc ADJP^{R,3} : ADJP = 201
Pc ADJP^{R,4} : ADJP = 4
...
Pw altered : VBNvbn-uNP = 3
Pw altered : VBNvbn-v = 2
Pw altering : VBGvbg-uNP = 1
Pw alternate : JJ = 3
Pw alternates : VBZvbz = 1
Pw alternating : VBGvbg-uNP = 2
Pw alternative : ADJP = 1
...

Figure 4.4: Selected lines from a ModelBlocks PCFG file trained on transformed WSJ training trees (Appendix A.8). Because we use the generated ModelBlocks PCFG file in a model transform (Section 4.2), we call `scripts/relfreq.pl` with the `-f` flag, generating unnormalized raw counts instead of probabilities. Without this flag, actual probability values are generated. Note that the file in the format shown (with counts) provides all the information required to calculate the probability values.

of the parent:

$$\mathsf{P}_{\theta_{CC}}(\beta_0 \,|\, \alpha, side, depth_0) = \frac{\sum_{\beta'_1} C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow \beta_0^{L,depth_0} {\beta'_1}^{R,depth})}{\sum_{\beta'_0,\beta'_1} C_{\theta_{CC}}(\alpha^{side,depth} \rightarrow {\beta'_0}^{L,depth_0} {\beta'_1}^{R,depth})} \tag{4.5}$$

In Equation 4.5, the summations should formally also marginalize over $depth$; however, since the value of $depth$ is deterministic given $side$ and $depth_0$ (and vice versa - see Algorithm 3.1), in practice this marginalization can be omitted. Likewise, the marginalization over $depth_0$ can be omitted in Equation 4.3 and Equation 4.4 for the same reason.

4.2.1 Left progeny

Following the model notation of Schuler (2010), we alias the depth-specific probability distribution of Equation 4.3 into two cases, depending on the $side$ of parent α — $\theta_{\text{G-L},d}$ for $side = L$ and $\theta_{\text{G-R},d}$ for $side = R$.

$$\mathsf{P}_{\theta_{\text{G-L},d}}(\alpha \rightarrow \beta_0\beta_1) = \mathsf{P}_{\theta_{CC}}(\beta_1, \beta'_2 \,|\, \alpha, side = L, depth = d) \tag{4.6}$$

$$\mathsf{P}_{\theta_{\text{G-R},d}}(\alpha \rightarrow \beta_0\beta_1) = \mathsf{P}_{\theta_{CC}}(\beta_1, \beta'_2 \,|\, \alpha, side = R, depth = d) \tag{4.7}$$

An expected count model $\theta_{\text{G-LR}^*,d}$ can be estimated from $\theta_{\text{G-L},d}$ and $\theta_{\text{G-R},d}$. This model is an estimate of the number of times each possible category label is encountered, when examining only the left descendents (or progeny) of parent α (whose $side = R$). We begin by summing over all possible right child values β'_2.

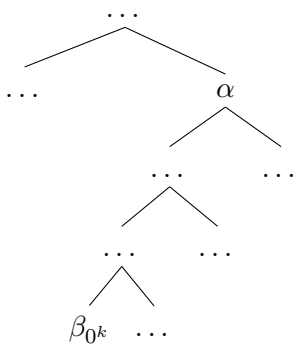
$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \overset{0}{\rightarrow} \beta_0 \ldots) = \sum_{\beta'_1} \mathsf{P}_{\theta_{\text{G-R},d}}(\alpha \rightarrow \beta_0\beta'_1) \tag{4.8}$$

Equation 4.8 models the number of subtree fragments in the training corpus where β_0 is the left child of α, and where $side = R$ for α:

$$\alpha \overset{0}{\rightarrow} \beta_0\beta_1 \;\Rightarrow\; \begin{array}{ccc} & \ldots & \\ \ldots & & \alpha \\ & \beta_0 & \ldots \end{array} \tag{4.9}$$

Equation 4.8 provides a count estimate for the case when β_0 is a direct left child of α. For a complete definition of expected count model $\theta_{\text{G-LR}^*,d}$ we must also count the subtrees where β_0 is an indirect left descendant of α.

$$\alpha \xrightarrow{k} \beta_{0^k} \ldots \Rightarrow \tag{4.10}$$



We define notation β_{0^k} to indicate that k intermediate left progeny ancestors separate β_0 from ancestor α. In Equation 4.9, $k = 0$. In the sample tree drawn in Equation 4.10, $k = 2$. We can now recursively define $\theta_{\text{G-LR}^*,d}$ for subtrees with separation value k:

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{k} \beta_{0^k} \ldots) = \sum_{\beta'_0} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{k-1} \beta'_{0^{k-1}} \ldots) \cdot \sum_{\beta'_1} \mathsf{P}_{\theta_{\text{G-L},d}}(\alpha \rightarrow \beta'_0 \beta'_1) \tag{4.11}$$

The expected count of finding left descendent β_0 at any level in the direct left progeny of ancestor α is defined as the summation of finding β_0 at level k for all values of k.

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{*} \beta_0 \ldots) = \sum_{k=0}^{\infty} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{k} \beta_{0^k} \ldots) \tag{4.12}$$

In practice, Equation 4.12 is approximated using value iteration (Bellman, 1957) by substituting a constant upper bound K in place of infinity. We use $K = 20$; this value should be sufficiently large to account for actual depth values of k encountered in the training corpus.

By normalizing we can obtain a probability model from the expected count model.

$$\mathsf{P}_{\theta_{\text{G-RL}^*,d}}(\beta_0 \mid \alpha) = \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{*} \beta_0 \ldots)}{\sum_{\beta'_0} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha \xrightarrow{*} \beta'_0 \ldots)} \tag{4.13}$$

For a node with constituent label α whose $side = R$ and $depth = d$, Equation 4.13 defines the conditional probability that constituent label β_0 can be found somewhere in the direct left progeny of node α.

Finally, it is sometimes useful to know how often a constituent label β_0 is in the left progeny of α, but is not the immediate left child of α. By subtracting, we can determine this expected count of β_0 as an indirect left descendent of α.

$$\mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha \xrightarrow{+} \beta_0 \ldots) = \mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha \xrightarrow{*} \beta_0 \ldots) - \mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha \xrightarrow{0} \beta_0 \ldots) \tag{4.14}$$

4.2.2 Transformed Component Models

Given a model of a PCFG in ModelBlocks file format (with counts instead of probabilities) as specified in Section 4.2 (and shown in Figure 4.4) we transform this model into an equivalent right-corner model using the model transform of Schuler (2009).

Murphy and Paskin (2001) note that the top layer of an HHMM can be equivalently defined as either a special case in the definition of the relevant random variables, or alternatively as a dummy layer component HMM with constant values. Figure 4.5 depicts an HHMM following the latter convention; such dummy nodes are shaded. The component models that comprise the transformed model in the ModelBlocks file format are listed below.

Model Constants

The models defined below incorporate certain edge case dependencies, which are omitted in the HHMM shown in Figure 4.3. The omitted dependencies are not true random variables, but rather are determined constant values that are required dependencies of nodes S_t^1, P_t, and R_t^5. These constant values are shown in Figure 4.5 as shaded nodes in order to present a complete depiction of the HHMM as a dynamic Bayesian network whose dependencies correspond completely and exactly with the equations below which define models θ_S, θ_R, and θ_P (for the S nodes, R nodes, and P nodes, respectively).

At time t, constant node S_t^0 is a required dependency of random variable representing node S_t^1. The active component $\alpha^0_{\mathrm{A}^t}$ of node S_t^0 is fixed with phrase structure category value *ROOT*; the awaited component $\alpha^0_{\mathrm{A}^t}$ of node S_t^0 is fixed with phrase structure category value *REST*. These categories are defined in terms of a streaming sequence of sentences; *ROOT* represents the root of the sequence while each instance of *REST* is the parent of one sentence in the sequence. This is illustrated in Figure 4.6.

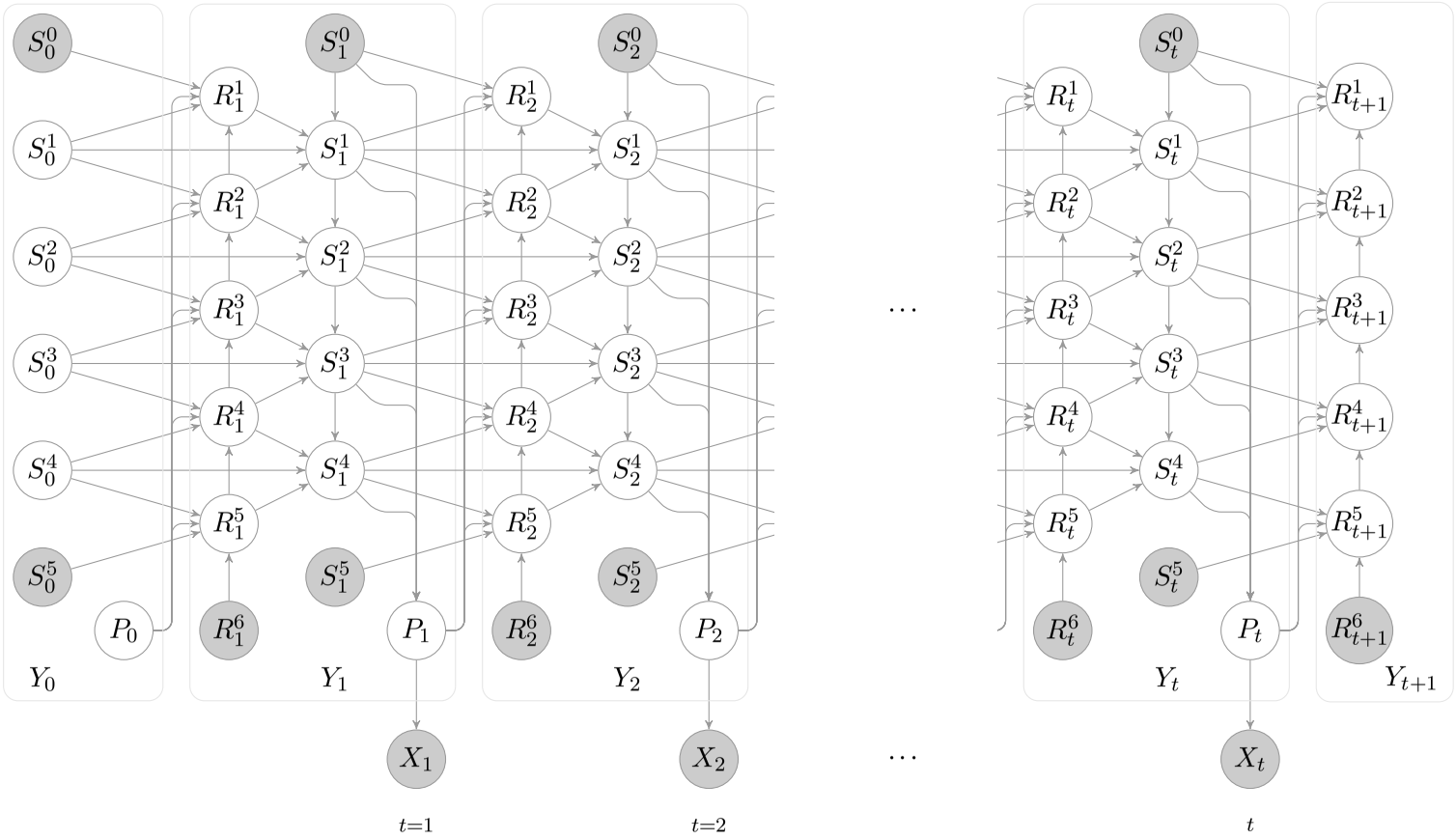


Figure 4.5: The full definitions of the S, R, and P nodes (models θ_S, θ_R, and θ_P, respectively) include certain edge case dependencies, which are omitted in the HHMM shown in Figure 4.3. The omitted dependencies are not true random variables, but rather are determined constant values that are required dependencies of nodes S_t^1, P_t, and R_t^5. These constant values are shown here as shaded nodes in order to present a complete depiction of the HHMM as a dynamic Bayesian network whose dependencies correspond completely and exactly with the equations that define θ_S, θ_R, and θ_P.

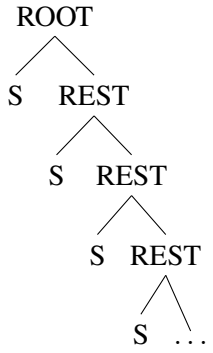


Figure 4.6: A meta-parse tree connects a sequence of complete sentences. Each complete sentence, S, acts as the left child of the ROOT of the sequence or the REST of the sequence.

Constant nodes R_t^{d+2} and S_{t-1}^{d+1} are required dependencies of the random variable representing node R_t^{d+1}. An empty phrase structure constituent value is indicated in the ModelBlocks files as a literal underscore "_". The active $\alpha^{d+1}_{\mathrm{A}^{t-1}}$ and awaited $\alpha^{d-1}_{\mathrm{W}^t}$ components of constant node S_{t-1}^{d+1} are each fixed with this value. The completed constituent α_t^{d+2} of constant node R_t^{d+2} is also fixed with this empty phrase structure constituent value. The boolean component b_t^{d+2} of R_t^{d+2} is fixed to value $\mathbf{1}$.

Observation Model

Observation model θ_X represents the conditional probability distribution of each observed X node. The definition of θ_X depends on several component models. These component models are part of the transformed ModelBlocks model file, and are defined below.

$\boldsymbol{Pc}$ As defined previously, $\mathsf{P}_{\theta_{Pc}}(\alpha \mid \alpha^{side,depth})$ for all unary rules $\alpha^{side,depth} \rightarrow x$.

In our models this values is deterministic with value 1. In other formulations this model could potentially have more interesting values. These Pc lines are obtained by relative frequency estimation - normalizing the counts from the Pc lines in the PCFG ModelBlocks file.

$\boldsymbol{Pw}$ As defined previously, $\mathsf{P}_{\theta_{Pw}}(\alpha \mid x)$ for all unary rules $\alpha^{side,depth} \rightarrow x$.

These Pw lines are obtained by relative frequency estimation - normalizing the counts from the Pw lines in the PCFG ModelBlocks file.

PwDT As defined previously, $\mathsf{P}_{\theta_{PwDT}}(x_{OOV} \mid \alpha)$ for all unary rules $\alpha \rightarrow w_{oov}$ for all out-of-vocabulary words x_{oov}.

These $PwDT$ lines are obtained from the decision tree process in Section 3.3.3.

P As defined previously, $\mathsf{P}_{\theta_P}(\alpha)$ for all unary rules $\alpha^{side,depth} \rightarrow x$.

These P lines are obtained by relative frequency estimation - normalizing the counts from the P lines in the PCFG ModelBlocks file.

The observation model must define a probability distribution θ_X of words given preterminal phrase structure categories. That is, $\mathsf{P}_{\theta_X}(x \mid \alpha)$. We can define this distribution $\mathsf{P}_{\theta_X}(x \mid \alpha)$ using Bayes's rule, but to do so requires some prior probability distribution θ_W over words in the target language. At parse time all words are observed; as such we define an arbitrary uniform prior distribution θ_W which assigns a constant very low probability value to every word:

$$\mathsf{P}_{\theta_W}(x) = \exp(-1000) \tag{4.15}$$

Having defined θ_W, we can now define θ_X using Bayes's rule in conjunction with the component models defined above. Observation model θ_X is first defined for the case when word x is known, that is, it was seen in the training corpus during model training.

$$\mathsf{P}_{\theta_X}(x \mid \alpha) = \frac{\mathsf{P}_{\theta_{Pc}}(\alpha \mid x) \cdot \mathsf{P}_{\theta_W}(x)}{\mathsf{P}_{\theta_P}(\alpha)} \tag{4.16}$$

Observation model θ_X is similarly defined for the case when word x was not seen during training, but in this case the out-of-vocabulary decision tree model θ_{PwDT} is used in place of θ_W.

$$\mathsf{P}_{\theta_X}(x \mid \alpha) = \frac{\mathsf{P}_{\theta_{Pc}}(\alpha \mid x) \cdot \mathsf{P}_{\theta_{PwDT}}(x)}{\mathsf{P}_{\theta_P}(\alpha)} \tag{4.17}$$

Preterminal Model

Model θ_P represents the conditional probability distribution at each hidden P node. The definition of θ_P depends on component model θ_{Ce} and node selection function SELECT-S-NODE. The component model θ_{Ce} is part of the transformed ModelBlocks model file, and is defined below.

$\boldsymbol{Ce}$ Preterminal expansion model $\mathsf{P}_{\theta_{Ce}}(\beta_0 \mid \alpha_{\mathrm{W}}, d)$.

Let HHMM node S_t^d at depth d be the deepest node at time t that contains a non-empty constituent. Given an incomplete constituent $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ at node S_t^d, probability model $\mathsf{P}_{\theta_{Ce}}(\beta_0 \mid \alpha_{\mathrm{W}}, d)$ is the probability that the awaited component constituent α_{W} will expand as preterminal β_0 at HHMM node P_t. Model θ_{Ce} is formally defined in Equation 4.18.[1]

$$\mathsf{P}_{\theta_{Ce}}(\beta_0 \mid \alpha_{\mathrm{W}}, d) = \frac{\mathsf{P}_{\theta_{CC}}(- \mid \alpha_{\mathrm{W}}, R, d) \cdot [\![\alpha_{\mathrm{W}} = \beta_0]\!] + \mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha_{\mathrm{W}} \xrightarrow{*} \beta_0 \ldots) \cdot \mathsf{P}_{\theta_{CC}}(- \mid \beta_0, L, d)}{\mathsf{P}_{\theta_{CC}}(- \mid \alpha_{\mathrm{W}}, R, d) + \sum_{\beta_0'} \mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha_{\mathrm{W}} \xrightarrow{*} \beta_0' \ldots) \cdot \mathsf{P}_{\theta_{CC}}(- \mid \beta_0', L, d)} \tag{4.18}$$

To obtain the θ_{Ce} model probability, we begin by obtaining the expected count of β_0 in the direct left progeny of α_{W} — $\mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha_{\mathrm{W}} \xrightarrow{*} \beta_0 \ldots)$. That count value is multiplied by the probability that β_0 will serve as a preterminal (that is, that it will expand to a terminal) — $\mathsf{P}_{\theta_{CC}}(- \mid \beta_0, L, d)$. If α_{W} can serve directly as a preterminal (indicated by $[\![\alpha_{\mathrm{W}} = \beta_0]\!]$), we add on the probability that α_{W} will serve as a preterminal $\mathsf{P}_{\theta_{CC}}(- \mid \alpha_{\mathrm{W}}, R, d)$ given $side = R$ for α_{W}.

At each time slice, there exist d variable S nodes, plus the constant nodes S_t^0 and S_t^{d+1}. The deepest S node, constant node S_t^{d+1}, is guaranteed to contain only the empty incomplete constituent __/__. The shallowest S node, constant node S_t^0, is guaranteed to contain only the non-empty incomplete constituent ROOT/REST. Selection function SELECT-S-NODE, shown in Algorithm 4.1, returns the depth x of the deepest S node at time t that contains a non-empty constituent.

Model θ_P is defined in terms of selection function SELECT-S-NODE and component model θ_{Ce}:

$$x = \textsc{Select-S-Node}(S_t^{0 \ldots d+1}) \tag{4.19}$$

$$\mathsf{P}_{\theta_P}(\beta_0 \mid \alpha^x_{\mathbf{W}^t}, x) = \mathsf{P}_{\theta_{Ce}}(\beta_0 \mid \alpha^x_{\mathbf{W}^t}, x) \tag{4.20}$$

[1] An indicator function $[\![\cdot]\!]$ is used to denote deterministic probabilities: $[\![\phi]\!] = 1$ if ϕ is true, 0 otherwise.

Algorithm 4.1 Given nodes $S_t^{0\ldots d+1}$ at time t, select node S_t^x such that x is deepest depth at time t which contains a non-empty constituent. A non-empty constituent is defined as one whose active component is not "__".

Require: Nodes $S_t^{0\ldots d+1}$ at time t

function SELECT-S-NODE($S_t^{0\ldots d+1}$)

for $x = d+1 \ldots 0$ **do**

if $\alpha^x_{\mathrm{A}t} \neq \_\_$ **then** ▷ $\alpha^x_{\mathrm{A}t}$ is the active component of node S_t^x

return x

end if

end for

end function

Shift Model

Model θ_S represents the conditional probability distribution at each hidden S node. The definition of θ_S depends on three component transition models. These component transition models are part of the transformed ModelBlocks model file, with definitions listed below.

Ctaa Model $\mathsf{P}_{\theta_{Ctaa}}(\alpha_{\mathrm{A}} \mid \alpha_{\mathrm{W}}^{d-1}, \beta_0, d)$ for active component of active transition.

Given the awaited component $\alpha_{\mathrm{W}}^{d-1}$ of an incomplete constituent at node S_t^{d-1} at depth $d-1$ and a completed constituent β_0 at node R_t^d, probability model $\mathsf{P}_{\theta_{Ctaa}}(\alpha_{\mathrm{A}} \mid \alpha_{\mathrm{W}}^{d-1}, \beta_0, d)$ provides the probability of transitioning to α_{A} as the active component of incomplete constituent $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ at node S_t^d.

$$\mathsf{P}_{\theta_{Ctaa}}(\alpha_{\mathrm{A}} \mid \alpha_{\mathrm{W}}^{d-1}, \beta_0, d) = \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \overset{*}{\rightarrow} \alpha_{\mathrm{A}} \ldots) \cdot \mathsf{P}_{\theta_{CC}}(\beta_0, \alpha_{\mathrm{W}} \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \cdot [\![\beta_0 \neq -]\!]}{\sum_{\alpha'_{\mathrm{A}}} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \overset{*}{\rightarrow} \alpha'_{\mathrm{A}} \ldots) \cdot \mathsf{P}_{\theta_{CC}}(\beta_0, \alpha_{\mathrm{W}} \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \cdot [\![\beta_0 \neq -]\!]} \tag{4.21}$$

Ctaw Model $\mathsf{P}_{\theta_{ctaw}}(\alpha_{\mathrm{W}} \mid \alpha_{\mathrm{A}}, \beta_0, d)$ for awaited component of active transition.

Given the active component α_{A} of an incomplete constituent at node S_t^d at depth d and a completed constituent β_0 at node R_t^d, probability model $\mathsf{P}_{\theta_{ctaw}}(\alpha_{\mathrm{W}} \mid \alpha_{\mathrm{A}}, \beta_0, d)$ provides the probability of α_{W} as the awaited component of incomplete constituent $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ at node S_t^d.

$$\mathsf{P}_{\theta_{ctaw}}(\alpha_{\mathrm{W}} \mid \alpha_{\mathrm{A}}, \beta_0, d) = \frac{\mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha_{\mathrm{W}}^{d-1} \overset{*}{\rightarrow} \alpha_{\mathrm{A}} \ldots) \cdot \mathsf{P}_{\theta_{CC}}(\beta_0, \alpha_{\mathrm{W}} \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \cdot [\![\beta_0 \neq -]\!]}{\sum_{\alpha'_{\mathrm{W}}} \mathsf{E}_{\theta_{\text{G-RL*},d}}(\alpha_{\mathrm{W}}^{d-1} \overset{*}{\rightarrow} \alpha_{\mathrm{A}} \ldots) \cdot \mathsf{P}_{\theta_{CC}}(\beta_0, \alpha'_{\mathrm{W}} \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \cdot [\![\beta_0 \neq -]\!]} \tag{4.22}$$

As an example for models θ_{Ctaa} and θ_{Ctaw}, consider an HHMM with incomplete constituent S/VP at node S_t^1 ($\alpha_{\mathrm{W}}^{d-1}$=VP) and completed constituent VB at node R_t^2 (β_0=VB). An active transition might transition from node R_t^d with context S_t^1 to incomplete constituent VP/NP at node S_t^2 (α=VP and α_{W}=NP). This example transition is shown in Figure 5.6.

Ctww Awaited transition model $\mathsf{P}_{\theta_{Ctww}}(\beta_1 \mid \alpha_{\mathrm{W}}, \beta_0, d)$.

Given an incomplete constituent $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ and a completed constituent β_0 , probability model $\mathsf{P}_{\theta_{Ctww}}(\beta_1 \mid \alpha_{\mathrm{W}}, \beta_0, d)$ provides the probability of transitioning to β_1 as the awaited component of incomplete constituent $\alpha_{\mathrm{A}}/\beta_1$.

$$\mathsf{P}_{\theta_{Ctww}}(\beta_1 \mid \alpha_{\mathrm{W}}, \beta_0, d) = \frac{\mathsf{P}_{\theta_{CC}}(\beta_0, \beta_1 \mid \alpha_{\mathrm{W}}, R, d) \cdot [\![d > 0]\!] \cdot [\![\beta_0 \neq -]\!]}{\sum_{\beta'_1} \mathsf{P}_{\theta_{CC}}(\beta_0, \beta_1 \mid \alpha_{\mathrm{W}}, R, d) \cdot [\![d > 0]\!] \cdot [\![\beta_0 \neq -]\!]} \tag{4.23}$$

As an example, consider an HHMM with incomplete constituent S/PP at node S_{t-1}^1 (α_{W}=PP) and completed constituent IN at node R_t^2 (β_0=IN). An awaited transition might transition from these values at nodes S_{t-1}^1 and R_t^2 to incomplete constituent S/NP at node S_t^1 (β_1=NP). This example transition is shown in Figure 5.6.

The θ_S model is defined as a set of component parts which interact to define the full model; it is composed from the awaited (θ_{Ctww}) and active (θ_{Ctaa} and θ_{Ctaw}) transition models, with certain edge cases hard-coded in the parser implementation itself. The formal definitions for the component parts of model θ_S are now presented.

1.

Node	S_t^d		R_t^{d-1}		R_t^d		S_{t-1}^d		S_t^{d-1}
	α_{A}	α_{W}	b^{d-1}	α^{d-1}	b	α	$\alpha_{\mathrm{A}^{t-1}}$	$\alpha_{\mathrm{W}^{t-1}}$	$\alpha_{\mathrm{W}}^{d-1}$
θ_{S_1}	$= \alpha_{\mathrm{A}^{t-1}}$	$= \alpha_{\mathrm{W}^{t-1}}$	$= \mathbf{0}$						

Model θ_S components										
θ_{S_1}	$= \alpha_{\mathrm{A}^{t-1}}$	$= \alpha_{\mathrm{W}^{t-1}}$	$= \mathbf{0}$						
θ_{S_2}		θ_{Ctww}	$= \mathbf{1}$	$\neq \_\_$					
θ_{S_3}	θ_{Ctaa}	θ_{Ctaw}		$= \_\_$	$= \mathbf{0}$				
θ_{S_4}	$= \_\_$	$= \_\_$			$= \mathbf{1}$				
	α_{A}	α_{W}	b^{d-1}	α^{d-1}	b	α	$\alpha_{\mathrm{A}^{t-1}}$	$\alpha_{\mathrm{W}^{t-1}}$	$\alpha_{\mathrm{W}}^{d-1}$
Node	S_t^d		R_t^{d-1}		R_t^d		S_{t-1}^d		S_t^{d-1}

Table 4.1: Constraints over which the component parts of model θ_S are defined. In each row, the columns indicate the node values for which that model component is defined to be potentially non-zero. Blank cells indicate that there is no restriction on that node value with respect to the given model component.

$$\theta_{S_1}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) = \\ [\![b^{d-1} = \mathbf{0}]\!] \cdot [\![\alpha_{\mathrm{A}} = \alpha_{\mathrm{A}^{t-1}}]\!] \cdot [\![\alpha_{\mathrm{W}} = \alpha_{\mathrm{W}^{t-1}}]\!] \quad (4.24)$$

2.

Node	S_t^d		R_t^{d-1}		R_t^d		S_{t-1}^d		S_t^{d-1}
	α_{A}	α_{W}	b^{d-1}	α^{d-1}	b	α	$\alpha_{\mathrm{A}^{t-1}}$	$\alpha_{\mathrm{W}^{t-1}}$	$\alpha_{\mathrm{W}}^{d-1}$
θ_{S_2}	$= \alpha_{\mathrm{A}^{t-1}}$	θ_{Ctww}	$= \mathbf{1}$	$\neq \_\_$					

$$\theta_{S_2}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) = \\ [\![b^{d-1} = \mathbf{1}]\!] \cdot [\![\alpha_{\mathrm{A}} \neq \_\_]\!] \\ \cdot \mathsf{P}_{\theta_{Ctww}}(\alpha_{\mathrm{W}} \mid \alpha_{\mathrm{W}^{t-1}}, \alpha^{d-1}, d) \quad (4.25)$$

3.

Node	S_t^d		R_t^{d-1}		R_t^d		S_{t-1}^d		S_t^{d-1}
	α_{A}	α_{W}	b^{d-1}	α^{d-1}	b	α	$\alpha_{\mathrm{A}^{t-1}}$	$\alpha_{\mathrm{W}^{t-1}}$	$\alpha_{\mathrm{W}}^{d-1}$
θ_{S_3}	θ_{Ctaa}	θ_{Ctaw}	$= \mathbf{1}$	$= \_\_$	$= \mathbf{0}$				

$$\theta_{S_3}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) = \\ [\![b^{d-1} = \mathbf{1}]\!] \cdot [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![b = \mathbf{1}]\!] \\ \mathsf{P}_{\theta_{Ctaa}}(\alpha_{\mathrm{A}} \mid \alpha_{\mathrm{W}}^{d-1}, \alpha, d) \cdot \mathsf{P}_{\theta_{Ctaw}}(\alpha_{\mathrm{W}} \mid \alpha_{\mathrm{A}}, \alpha, d) \quad (4.26)$$

4.

Node	S_t^d		R_t^{d-1}		R_t^d		S_{t-1}^d		S_t^{d-1}
	α_{A}	α_{W}	b^{d-1}	α^{d-1}	b	α	$\alpha_{\mathrm{A}^{t-1}}$	$\alpha_{\mathrm{W}^{t-1}}$	$\alpha_{\mathrm{W}}^{d-1}$
θ_{S_4}	$= \_\_$	$= \_\_$	$= \mathbf{1}$	$= \_\_$	$= \mathbf{1}$				

$$\theta_{S_4}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) = \\ [\![b^{d-1} = \mathbf{1}]\!] \cdot [\![\alpha^{d-1} = \_\_]\!] \cdot [\![b = \mathbf{1}]\!] \cdot [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![\alpha_{\mathrm{W}} = \_\_]\!] \quad (4.27)$$

The conditions under which each component model is active are mutually exclusive. These components models are combined as follows to define θ_S as a complete well-formed probability model:

$$\mathsf{P}_{\theta_S}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) = \\ \sum_{i \in \{1 \ldots 4\}} \theta_{S_i}(\alpha_{\mathrm{A}}, \alpha_{\mathrm{W}} \mid b^{d-1}, \alpha^{d-1}, b, \alpha, \alpha_{\mathrm{A}^{t-1}}, \alpha_{\mathrm{W}^{t-1}}, \alpha_{\mathrm{W}}^{d-1}) \quad (4.28)$$

Reduction Model

Model θ_R represents the conditional probability distribution at each hidden node R. The definition of θ_R depends on the component model specified by the F lines in the transformed ModelBlocks file. The definition of model θ_R is listed below.

F Reduction model $\mathsf{P}_{\theta_R}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1})$.

Given the active $\alpha_{\mathrm{A}}^{d-1}$ and awaited $\alpha_{\mathrm{W}}^{d-1}$ components of an incomplete constituent at node S_{t-1}^{d-1} at depth $d-1$, the active component α_{A} of an incomplete constituent at node S_{t-1}^d at depth d, the completed constituent β_0 at node P_{t-1}, and the completed constituent α^{d+1} at node R_t^{d+1}, probability model $\mathsf{P}_{\theta_R}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1})$ provides the probability of b, α at node R_t^d for completed constituent α and boolean reduction indicator value b. The ModelBlocks file format for the reduction model lines is defined as $F\ d\ \alpha_{\mathrm{W}}^{d-1}\ \alpha_{\mathrm{A}}\ \beta_0 : b$,$\alpha = p$.

The reduction model θ_R provides a model for the dependencies of a reduction node R_t^d in an HHMM at time t and depth d. Each node R_t^d interacts with the preceding S nodes at depths d and $d-1$ (nodes S_{t-1}^d and S_{t-1}^{d-1}, respectively). Node R_t^d also depends on the

Model θ_R components								
	θ_{R_1}	$= \mathbf{0}$	$= -$	$\neq \_\_$				
	θ_{R_2}	$= \mathbf{1}$	$= \alpha_{\mathrm{A}}$	$= \_\_$	$\neq \_\_$			
	θ_{R_3}	$= \mathbf{0}$						
	θ_{R_4}	$= \mathbf{1}$	$= \_\_$		$= \_\_$	$= \_\_$		
	θ_{R_5}					$\neq \_\_$	$= \beta_0$	
	θ_{R_6}		$= \beta_0$				$\neq \beta_0$	
	θ_{R_7}	$= \mathbf{0}$						
		b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
	Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}

Table 4.2: Constraints over which the component parts of model θ_R are defined. In each row, the columns indicate the node values for which that model component is defined to be potentially non-zero. Blank cells indicate that there is no restriction on that node value with respect to the given model component.

preceding preterminal node P_{t-1}, as well as the R node below it (R_t^{d+1}). Model θ_R is defined as a set of component parts which interact to define a full well-formed conditional probability model. While this model is primarily constructed by the script commands shown in Figure A.3, certain edge cases are hard-coded in the parser implementation itself.

The formal definitions for the component parts of model θ_R are now presented. Table 4.2 provides a summary, displaying the constraint conditions under which each component model is active.

1.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_1}	$= \mathbf{0}$	$= -$	$\neq \_\_$				

$$\theta_{R_1}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) = [\![b = \mathbf{0}]\!] \cdot [\![\alpha = -]\!] \cdot [\![\alpha^{d+1} \neq \_\_]\!] \quad (4.29)$$

2.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_2}	$= \mathbf{1}$	$= \alpha_{\mathrm{A}}$	$= \_\_$	$\neq \_\_$			

$$\theta_{R_2}(b, \alpha \,|\, d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) =$$
$$[\![b = \mathbf{1}]\!] \cdot [\![\alpha = \alpha_{\mathrm{A}}]\!] \cdot [\![\alpha^{d+1} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}} \neq \_\_]\!]$$
$$\cdot \, \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \xrightarrow{0} \alpha \ldots) \quad (4.30)$$

3.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_3}	$= \mathbf{0}$	$= \alpha_{\mathrm{A}}$	$= \_\_$	$\neq \_\_$			

$$\theta_{R_3}(b, \alpha \,|\, d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) =$$
$$[\![b = \mathbf{0}]\!] \cdot [\![\alpha = \alpha_{\mathrm{A}}]\!] \cdot [\![\alpha^{d+1} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}} \neq \_\_]\!]$$
$$\cdot \, \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \xrightarrow{+} \alpha \ldots) \quad (4.31)$$

4.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_4}	$= \mathbf{1}$	$= \_\_$	$= \_\_$	$= \_\_$	$= \_\_$		

$$\theta_{R_4}(b, \alpha \,|\, d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) =$$
$$[\![b = \mathbf{1}]\!] \cdot [\![\alpha = \_\_]\!] \cdot [\![\alpha^{d+1} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}}^{d-1} = \_\_]\!] \quad (4.32)$$

5.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_5}	$= \mathbf{1}$	$= \_\_$	$= \_\_$	$= \_\_$	$\neq \_\_$	$= \beta_0$	

$$\theta_{R_5}(b, \alpha \,|\, d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) =$$
$$[\![b = \mathbf{1}]\!] \cdot [\![\alpha = \_\_]\!] \cdot [\![\alpha^{d+1} = \_\_]\!]$$
$$\cdot \, [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}}^{d-1} \neq \_\_]\!] \cdot [\![\alpha_{\mathrm{W}}^{d-1} = \beta_0]\!]$$
$$\cdot \, \mathsf{P}_{\theta_{CC}}(- \,|\, \alpha_{\mathrm{W}}^{d-1}, R, d) \quad (4.33)$$

6.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_6}	$= \mathbf{1}$	$= \beta_0$	$= \_\_$	$= \_\_$	$\neq \_\_$	$\neq \beta_0$	

$$\theta_{R_6}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) = \\ [\![b = \mathbf{1}]\!] \cdot [\![\alpha = \_\_]\!] \cdot [\![\alpha^{d+1} = \_\_]\!] \\ \cdot [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}}^{d-1} \neq \_\_]\!] \cdot [\![\alpha_{\mathrm{W}}^{d-1} \neq \beta_0]\!] \\ \cdot \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \xrightarrow{0} \alpha \ldots) \cdot \mathsf{P}_{\theta_{CC}}(- \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \quad (4.34)$$

7.

Node	F_t^d		F_t^{d+1}	S_{t-1}^d	S_{t-1}^{d-1}		P_{t-1}
	b	α	α^{d+1}	α_{A}	$\alpha_{\mathrm{A}}^{d-1}$	$\alpha_{\mathrm{W}}^{d-1}$	β_0
θ_{R_7}	$= \mathbf{0}$	$= \beta_0$	$= \_\_$	$= \_\_$	$\neq \_\_$	$\neq \beta_0$	

$$\theta_{R_7}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) = \\ [\![b = \mathbf{0}]\!] \cdot [\![\alpha = \_\_]\!] \cdot [\![\alpha^{d+1} = \_\_]\!] \\ \cdot [\![\alpha_{\mathrm{A}} = \_\_]\!] \cdot [\![\alpha_{\mathrm{A}}^{d-1} \neq \_\_]\!] \cdot [\![\alpha_{\mathrm{W}}^{d-1} \neq \beta_0]\!] \\ \cdot \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(\alpha_{\mathrm{W}}^{d-1} \xrightarrow{+} \alpha \ldots) \cdot \mathsf{P}_{\theta_{CC}}(- \mid \alpha_{\mathrm{W}}^{d-1}, L, d) \quad (4.35)$$

The conditions under which each component model is active are mutually exclusive. These components models are combined as follows to define θ_R as a complete well-formed probability model:

$$\mathsf{P}_{\theta_R}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1}) = \\ \frac{\sum_{i \in \{1 \ldots 7\}} \theta_{R_i}(b, \alpha \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1})}{\sum_{b', \alpha'} \sum_{i \in \{1 \ldots 7\}} \theta_{R_i}(b', \alpha' \mid d, \alpha_{\mathrm{A}}, \alpha_{\mathrm{A}}^{d-1}, \alpha_{\mathrm{W}}^{d-1}, \beta_0, \alpha^{d+1})} \quad (4.36)$$

4.3 Parsing with an HHMM

We now define an HHMM parsing algorithm capable of estimating the language model probability of a sequence of words in the target language. This definition makes use of the previously defined hierarchical hidden component transition model θ_A and observed component model θ_B.[2]

[2] The full HHMM definition also requires hidden state prior model θ_Π. We use a trivial fixed deterministic distribution for θ_Π. Consequently for simplicity, we elide θ_Π from all further equations.

We begin with the HHMM (with depth d) shown as a dynamic Bayes network in Figure 4.5. At each time step t, this HHMM is comprised of hidden reduction nodes $R_t^{1...d+1}$, hidden shift nodes $S_t^{1...d}$, hidden preterminal node P_t, and observed node X. Each node in the HHMM serves as a random variable.[3] The respective domains of the random variables are defined in Section 4.2.

4.3.1 Belief Propagation through Message Passing

Our hidden models define probability distributions for transitions between hidden states. However, to obtain language model probability estimates requires the full probability distribution at each hidden node. Pearl (1982) defines belief propagation through message passing over nodes in a directed acyclic graph as a mechanism to calculate the full probability distribution at each node, given relevant transition and prior models (Bayes, 1763). Full probability distributions are directly available at nodes which have no parents. The message passing algorithm begins at these nodes; at each child node whose parents' probability distributions are fully known, parent distributions and transition distributions are used to calculate the child node's probability distribution. As each child node is processed, the algorithm repeats, until all nodes in the network have been fully specified. If the graph of nodes is singly connected (that is, has no directed or undirected cycles), message passing is guaranteed to correctly calculate the probability distribution at each node in the graph.

Unfortunately the graph topology in our HHMM is multiply connected (contains undirected cycles). Belief propagation through message passing is not guaranteed to calculate correct probability distributions at nodes in multiply connected networks. To circumvent this shortcoming, multiply connected graphs can be transformed into equivalent singly connected graphs by merging nodes (Pearl, 1986). Message passing can then be performed on the transformed network. In HMM terms, this node clustering technique can be thought of as a mechanism for unfactoring the factored hidden nodes (at a given time step) in an HHMM into a single huge non-factored hidden node. The relative complexity of our HHMM topology, coupled with the large domain of the hidden variables means that in practice, node clustering is impractical for calculating hidden node probability distributions in our HHMM parser.

[3] The HHMM also contains determined constant values at S_t^0, S_t^{d+1}, and R_t^{d+2}. These nodes are required dependencies which are necessary for the complete model definition, but are not random variables.

Belief propagation through message passing is well-defined over multiply connected networks of nodes (Pearl, 1988). Even though correct calculation is not guaranteed, in practice the use of message passing in multiply connected networks results in calculated distributions of sufficiently high quality for certain problems (McEliece et al., 1998). Whether message passing would result in useful results when applied to the multiply connected nodes in our HHMM is an empirical question which we leave as future work.

4.3.2 HHMM Parsing as a Search Problem

The prominent traditional task in parsing is the estimation of the 1-best parse tree, given a parsing model and a sequence of words to parse in the target language. In an HHMM parser, we define each path through the dynamic Bayes network as a tree with leaves at the observed nodes $X_{1...t}$. At each node in a path, the random variable defined by that node takes on one concrete value from its respective domain. With this perspective, 1-best parsing can be cast as a search problem: find that path and associated random variable value assignments which are most probable according to the defined models. This path will correspond to the 1-best parse tree.

This search can be performed through exhaustive enumeration of all possible path and random variable assignment combinations through the network of nodes defined by the HHMM. In belief propagation through message passing, the full probability distribution is calculated at each child node, given the fully defined probability distribution for all the child's parents. Because of the multiply connected topology of the HHMM, belief propagation through message passing would not guarantee correctly calculated probability distributions; consequently we must pursue an alternate technique.

Each random variable is assigned a single value from its respective domain; given concrete values at every node, the probability of the path corresponding to that combination of variable assignments is calculated given the HHMM models and the observed sequence of words. Successively iterating through all combinations of variable assignments over all nodes exhaustively enumerates all possible path and variable assignments. The path and variable assignments with the highest probability corresponds to the 1-best parse tree.

We now formally define the algorithm for exhaustively enumerating all possible path and random variable assignment combinations through the network of nodes defined by the HHMM. For notational convenience, we define store state σ_t as an ordered list containing the random

variables that comprise the nodes at $S_t^{1\ldots D}$ and P_t. Recall that each node S_t^d at time t and depth d contains active component $\alpha_{\mathrm{A}^t}^d$ and awaited component $\alpha_{\mathrm{W}^t}^d$ and each node P_t contains preterminal β_t. The random variables in σ_t are all initialized to be *unbound*.

$$\sigma_t = \left\langle \alpha_{\mathrm{A}^t}^1, \alpha_{\mathrm{W}^t}^1, \ldots, \alpha_{\mathrm{A}^t}^d, \alpha_{\mathrm{W}^t}^d, \beta_t \right\rangle \tag{4.37}$$

We define an unbound random variable to be one which has not yet been assigned a value. A useful analogy might be a hardware register in a computer in which no value has yet been stored. Our use here of the term unbound is distinct from and should not be confused with the separate meaning of bound and unbound variables in the context of quantifiers in predicate logic.

Similarly, we define reduction state ρ_t as an ordered list containing the random variables that comprise the nodes at $R_t^{D+1\ldots 1}$. Recall that each node R_t^d at time t and depth d contains boolean flag b_t^d and nonterminal α_t^d. The random variables in ρ_t are all initialized to be unbound.

$$\rho_t = \left\langle b_t^{d+1}, \alpha_t^{d+1}, \ldots, b_t^1, \alpha_t^1 \right\rangle \tag{4.38}$$

Algorithm 4.2 Assign next values to random variables b_t^d and α_t^d at node R_t^d

```
1:  function AssignNextValues(R_t^d, σ_{t-1}, ρ_t, σ_t)
2:      Array b contains all values in the boolean domain
3:      Array N contains all values in the nonterminal domain
4:      i,j = map_R[R_t^d, σ_{t-1}, ρ_t, σ_t]
5:                          ▷ Map lookup returns ⟨0, −1⟩ if key ⟨R_t^d, σ_{t-1}, ρ_t, σ_t⟩ is not present
6:      if j + 1 < |N| then
7:          j += 1
8:      else
9:          j = 0
10:         i += 1
11:     end if
12:     map_R[R_t^d, σ_{t-1}, ρ_t, σ_t] = i, j      ▷ Store new indices in map at key ⟨R_t^d, σ_{t-1}, ρ_t, σ_t⟩
13:     return b[i], N[j]
14: end function
```

The search task is performed by successively enumerating over the possible values in the respective domain of each random variable, first in ρ_t and then in σ_t. At the beginning of the

search, each random variable is unbound, meaning that no value has yet been assigned to it from its respective domain. Once each random variable in the hidden domain at a time step has been bound to a respective value, the combination of these variable bindings constitutes one path through the search space. We now formally define the functions that comprise this algorithm. We present the functions in bottom-up order, so that all functions used in an algorithm are defined before they are used in subsequent algorithms. The complete algorithm for parsing at a time step is listed in Algorithm 4.12. The algorithm formally defining the syntactic language model score at a store state is Algorithm 4.14.

Algorithm 4.2 formally defines the search enumeration function for the random variables b_t^d and α_t^d at hidden reduction node R_t^d. The first required parameter to this function is the store state σ_{t-1} from the previous time step. Store state σ_t and ρ_t are also required parameters. In the first call to the function, the random variables in σ_t and ρ_t are all unbound. In subsequent calls, some or all of the random variables in σ_t and ρ_t will be bound. Given values for σ_{t-1}, σ_t, and ρ_t, Algorithm 4.2 calculates and returns a new combination of values to bind to random variables b_t^d and α_t^d at reduction node R_t^d.

Algorithm 4.3 Assign next values to random variables $\alpha_{\mathrm{A}^t}^d$ and $\alpha_{\mathrm{W}^t}^d$ at node S_t^d

1: **function** ASSIGNNEXTVALUES(S_t^d,σ_{t-1},ρ_t,σ_t)
2: Array $\boldsymbol{N}$ contains all values in the nonterminal domain
3: i,j = $map_S[S_t^d,\sigma_{t-1},\rho_t,\sigma_t]$
4: ▷ Map lookup returns $\langle 0, -1 \rangle$ if key $\langle S_t^d,\sigma_{t-1},\rho_t,\sigma_t \rangle$ is not present
5: **if** $j + 1 < |\boldsymbol{N}|$ **then**
6: j += 1
7: **else**
8: j = 0
9: i += 1
10: **end if**
11: $map_S[S_t^d,\sigma_{t-1},\rho_t,\sigma_t] = i, j$ ▷ Store new indices in map at key $\langle S_t^d,\sigma_{t-1},\rho_t,\sigma_t \rangle$
12: **return** $\boldsymbol{N}[i], \boldsymbol{N}[j]$
13: **end function**

Algorithm 4.2 defines and uses a globally scoped associative array, map_R, capable of storing key-value pairs. In Algorithm 4.2, the tuple $\langle R_t^d,\sigma_{t-1},\rho_t,\sigma_t \rangle$ serves as the key; the values

stored in the array represent a pair of indices, i (into the boolean domain) and j (into the non-terminal domain). Similar globally scoped associative arrays map_S and map_P are used in Algorithm 4.3 and Algorithm 4.4, respectively.

Similarly, Algorithm 4.3 formally defines the search enumeration function for the random variables $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ at hidden store node S^d_t. As before, the store state values σ_{t-1} from the previous time step are required, along with existing values ρ_t and σ_t for the current time step. In the first call to the function, the random variables in σ_t and ρ_t are all unbound. In subsequent calls, some or all of the random variables in σ_t and ρ_t will be bound. Given values for σ_{t-1}, σ_t, and ρ_t, Algorithm 4.3 calculates and returns a new combination of values to bind to random variables $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ at store node S^d_t.

Algorithm 4.4 Assign next values to random variable β_t at node P^d_t

```
1: function AssignNextValues(P_t,σ_{t-1},ρ_t,σ_t)
2:     Array N contains all values in the nonterminal domain
3:     i = map_P[P_t,σ_{t-1},ρ_t,σ_t]
4:                          ▷ Map lookup returns −1 if key ⟨P_t,σ_{t-1},ρ_t,σ_t⟩ is not present
5:     i += 1
6:     map_P[P_t,σ_{t-1},ρ_t,σ_t] = i        ▷ Store new indices in map at key ⟨P_t,σ_{t-1},ρ_t,σ_t⟩
7:     return N[i]
8: end function
```

Finally, Algorithm 4.4 formally defines the search enumeration function for the random variable β_t at hidden preterminal node P^d_t. As before, the store state values σ_{t-1} from the previous time step are required, along with existing values ρ_t and σ_t for the current time step. In the first call to the function, the random variables in σ_t and ρ_t are all unbound. In subsequent calls, some or all of the random variables in σ_t and ρ_t will be bound. Given values for σ_{t-1}, σ_t, and ρ_t, Algorithm 4.4 calculates and returns a new value to bind to random variable β_t at preterminal node P^d_t.

Algorithms 4.2, 4.3, and 4.4 define elementary search enumeration functions for random variables at R, S, and P nodes, respectively. These functions will be used in later functions that drive the actual enumeration.

The topology of the HHMM dynamic Bayes network defines a traversal order over nodes for both message passing and exhaustive enumeration. All nodes at time step t have dependencies

Algorithm 4.5 Given the random variables in ρ_t, identify the deepest level d in the HHMM dynamic Bayes network at time t where the random variables b_t^d and α_t^d for node R_t^d are unbound in ρ_t.

```
1: function DEEPESTUNBOUND(ρ_t,D)
2:     for d in D + 1 ... 1 do
3:         if b_t^d,α_t^d from R_t^d are unbound in ρ_t then
4:             return d
5:         end if
6:     end for
7:     return 0
8: end function
```

on nodes in the previous store state σ_{t-1}. The traversal order is defined such that a node is visited only once all of its parents have been visited.

In our use case, visiting a node means using Algorithm 4.2, 4.3, or 4.4 (as appropriate given the node type) to bind a value to the random variables at the node. For reduction nodes, this implies that if the random variables at R_t^d are bound, then so are the random variables at all deeper nodes $R_t^{d'}$, where $d' > d$. If zero or more R nodes at time t have been bound, the function defined by Algorithm 4.5 identifies the depth d of the deepest node R_t^d that has not yet been bound, returning 0 if the random variables at all nodes $R_t^{1\ldots D+1}$ have been bound. Recall that b_t^{D+1} and α_t^{D+1} at R_t^{D+1} are constants, not random variables, and as such are by definition always bound.

For store nodes, the traversal order implies that if the random variables at S_t^d are bound, then so are the random variables at all shallower nodes $S_t^{d'}$, where $d' < d$. If zero or more S nodes at time t have been bound, the function defined by Algorithm 4.6 identifies the depth d of the shallowest node S_t^d that has not yet been bound, returning $D + 1$ if the random variables at all nodes $S_t^{0\ldots D}$ and P_t have been bound. Recall that $\alpha_{\mathbf{A}^t}^0$ and $\alpha_{\mathbf{W}^t}^0$ at S_t^0 are constants, not random variables, and as such are by definition always bound.

Algorithms 4.5 and 4.6 define functions that identify the next unbound node in the node traversal order. Algorithms 4.2, 4.3, and 4.4 define elementary search enumeration functions for random variables at R, S, and P nodes, respectively. These functions are integrated in Algorithm 4.7.

Algorithm 4.6 Given the random variables in σ_t, identify the shallowest level d in the HHMM dynamic Bayes network at time t where the random variables $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ for node S^d_t are unbound in σ_t. If the random variables for nodes $S^{1\ldots D}_t$ are all bound in σ_t, this function returns a depth value of $D+1$, which logically corresponds to the depth of preterminal node P_t.

```
1: function SHALLOWESTUNBOUND(σ_t,D)
2:     for d in 1 ... D do
3:         if α^d_{A^t},α^d_{W^t} from S^d_t are unbound in σ_t then
4:             return d
5:         end if
6:     end for
7:     return D + 1
8: end function
```

Given previous store state σ_{t-1}, the FORWARDSTEP function defined in Algorithm 4.7 uses Algorithms 4.5 and 4.6 to determine the next unbound node (from ρ_t and σ_t) in the node traversal order. Once the next unbound node is identified, the appropriate enumeration function (Algorithm 4.2 for R nodes, Algorithm 4.3 for S nodes, or Algorithm 4.4 for P nodes) is called to determine the first value(s) from the node's domain(s). The returned values are then bound to the random variable(s) at that node. We refer to this process of selecting the next unbound node and binding its random variables as taking a *forward step* through the dynamic Bayes network.

As each forward step is taken, the relevant probability models are consulted, and a cumulative score is maintained for the partial path though the nodes:

- If a forward step is taken into hidden node R^d_t, reduction model θ_R is consulted to calculate the conditional joint probability of b^d_t and α^d_t given the random variables at the parent nodes of R^d_t.

- If all nodes $R^{1\ldots D+1}_t$ are bound, and a forward step is taken into hidden node S^d_t, shift model θ_S is consulted to calculate the conditional joint probability of $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ given the random variables at the parent nodes of S^d_t.

- If all nodes $R^{1\ldots D+1}_t$ and $S^{0\ldots D}_t$ are bound, and a forward step is taken into hidden node P_t, preterminal expansion model θ_{Ce} is consulted to calculate the conditional probability of β_t given the random variables at the parent nodes of P_t.

Algorithm 4.7 Step forward through reduce state ρ_t, store state σ_t, and observation x_t

function FORWARDSTEP(D,σ_{t-1},ρ_t,σ_t,x_t,score)
  d = DEEPESTUNBOUNDLEVEL(ρ_t,D)
  if $d > 0$ **then** ▷ Step forward to R_t^d
    b_t^d, α_t^d = ASSIGNNEXTVALUES(R_t^d,σ_{t-1},ρ_t,σ_t)
    score′ = score $- \log \mathsf{P}_{\theta_R}(b_t^d, \alpha_t^d \mid d, \alpha^d_{\mathrm{A}^{t-1}}, \alpha^{d-1}_{\mathrm{A}^{t-1}}, \alpha^{d-1}_{\mathrm{W}^{t-1}}, \beta_{t-1}, \alpha_t^{d+1})$
    ρ_t' = substitute newly bound b_t^d, α_t^d into a copy of ρ_t
    return ρ_t', σ_t, score′, **false**
  else
    d = SHALLOWESTUNBOUNDLEVEL(σ_t,D)
    if $d < D$ **then** ▷ Step forward to S_t^d
      $\alpha^d_{\mathrm{A}^t}$, $\alpha^d_{\mathrm{W}^t}$ = ASSIGNNEXTVALUES(S_t^d,σ_{t-1},ρ_t,σ_t)
      score′ = score $- \log \mathsf{P}_{\theta_S}(\alpha^d_{\mathrm{A}^t}, \alpha^d_{\mathrm{W}^t} \mid b_t^{d-1}, \alpha_t^{d-1}, b_t^d, \alpha_t^d, \alpha^d_{\mathrm{A}^{t-1}}, \alpha^d_{\mathrm{W}^{t-1}}, \alpha^{d-1}_{\mathrm{W}^t})$
      σ_t' = substitute newly bound $\alpha^d_{\mathrm{A}^t}$, $\alpha^d_{\mathrm{W}^t}$ into a copy of σ_t
      return ρ_t, σ_t', score′, **false**
    else if $d = D$ **then** ▷ Step forward to P_t
      β_t = ASSIGNNEXTVALUES(P_t,σ_{t-1},ρ_t,σ_t)
      score′ = score $- \log \mathsf{P}_{\theta_{Ce}}(\beta_t \mid \alpha^d_{\mathrm{W}^t}, d)$
      σ_t' = substitute newly bound β_t into a copy of σ_t
      return ρ_t, σ_t', score′, **false**
    else ▷ Step forward to X_t
      score′ = score $- \log \mathsf{P}_{\theta_X}(x \mid \beta_t)$
      return ρ_t, σ_t, score′, **true**
    end if
  end if
end function

- Finally, if all nodes $R_t^{1\ldots D+1}$, $S_t^{0\ldots D}$, and P_t are bound, and a forward step is taken into observed node X_t, observation model θ_X is consulted to calculate the conditional probability of the observed value at X_t given the preterminal β_t at parent node P_t.

Whatever probability value is calculated, the negative log of that value is summed with the negative log probabilities previously calculated for the node bindings along the (partial) path.

Algorithm 4.8 Given the random variables in ρ_t, identify the shallowest level d in the HHMM dynamic Bayes network at time t where the random variables b_t^d and α_t^d for node R_t^d are bound in ρ_t.

```
1: function SHALLOWESTBOUND(ρ_t,D)
2:     for d in 1 ... D + 1 do
3:         if b_t^d,α_t^d from R_t^d are bound in ρ_t then
4:             return d
5:                 ▷ Constants b_t^{D+1},α_t^{D+1} from R_t^{D+1} by definition always have bound values
6:         end if
7:     end for
8: end function
```

Each forward step to each random variable in the selected node binds the first value of that domain and that random variable. Full enumeration through all values of all hidden random variables at a time step obviously requires a mechanism for binding all other values from each random variable's domain to the respective random variable. A forward step takes ρ_t and σ_t and calculates ρ'_t and σ'_t by binding a previously unbound random variable from ρ_t or σ_t.

We define a *side step* to take ρ_t and σ_t and calculates ρ'_t and σ'_t by rebinding a new value to the most recently bound random variables from ρ_t and σ_t. To perform a side step, we must be able to identify the most recently bound random variables from ρ_t and σ_t. The functions defined by Algorithms 4.5 and 4.6 identify the next unbound node in the node traversal order. We now define variants of these which identify the most recently bound node in the node traversal order for a given σ_{t-1}, ρ_t and σ_t.

Recall that for reduction nodes $R_t^{1\ldots D+1}$, if the random variables at R_t^d are bound, then so are the random variables at all deeper nodes $R_t^{d'}$, where $d' > d$. Algorithm 4.8 identifies the depth d of the shallowest node R_t^d whose b_t^d and α_t^d have been bound; this is done by iterating through the random variables in ρ_t.

Algorithm 4.9 Given the random variables in σ_t, identify the deepest level d in the HHMM dynamic Bayes network at time t where the random variables $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ for node S^d_t are bound in σ_t. If the random variable β_t in preterminal nodes P_t is also bound, this function returns a depth value of $D+1$.

```
1:  function DEEPESTBOUND(σ_t,D)
2:      if β_t from P_t is bound in σ_t then
3:          return D + 1
4:      else
5:          for d in D ... 0 do
6:              if α^d_{A^t},α^d_{W^t} from S^d_t are bound in σ_t then
7:                  return d
8:                      ▷ Constants α^0_{A^t},α^0_{W^t} from S^0_t by definition always have bound values
9:              end if
10:         end for
11:     end if
12: end function
```

Similarly, Algorithm 4.9 identifies the depth d of the deepest node in σ_t whose random variables have been bound. If β_t from preterminal node P_t has been bound, depth $D+1$ is returned. Otherwise, d is returned, where d is the depth of the deepest node S^d_t whose $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ have been bound.

Together, Algorithms 4.8 and 4.9 define functions that identify the most recently bound node in the node traversal order. These functions are integrated with the elementary search enumeration functions defined in Algorithms 4.2, 4.3, and 4.4 to form the SIDESTEP function in Algorithm 4.10.

As each side step is taken, the relevant probability models are consulted, and a cumulative score is maintained for the partial path through the nodes:

- If a forward step is taken at hidden node R^d_t, reduction model θ_R is consulted to calculate the conditional joint probability of b^d_t and α^d_t given the random variables at the parent nodes of R^d_t.
- If all nodes $R^{1\ldots D+1}_t$ are bound, and a forward step is taken at hidden node S^d_t, shift

Algorithm 4.10 Step sideways through reduce state ρ_t or store state σ_t

function SIDESTEP(D,σ_{t-1},ρ_t,σ_t,score)
  d = DEEPESTBOUNDLEVEL(σ_t,D)
  if $d = 0$ **then**
    d = SHALLOWESTBOUNDLEVEL(ρ_t,D) ▷ Side step at R_t^d
    b_t^d, α_t^d = ASSIGNNEXTVALUES(R_t^d,σ_{t-1},ρ_t,σ_t)
    score′ = score $- \log \mathsf{P}_{\theta_R}(b_t^d, \alpha_t^d \mid d, \alpha^d_{\mathrm{A}^{t-1}}, \alpha^{d-1}_{\mathrm{A}^{t-1}}, \alpha^{d-1}_{\mathrm{W}^{t-1}}, \beta_{t-1}, \alpha_t^{d+1})$
    ρ'_t = substitute newly re-bound b_t^d, α_t^d into a copy of ρ_t
    return ρ'_t, σ_t, score′
  else
    if $d \leq D$ **then** ▷ Side step at S_t^d
      $\alpha^d_{\mathrm{A}^t}$, $\alpha^d_{\mathrm{W}^t}$ = ASSIGNNEXTVALUES(S_t^d,σ_{t-1},ρ_t,σ_t)
      score′ = score $- \log \mathsf{P}_{\theta_S}(\alpha^d_{\mathrm{A}^t}, \alpha^d_{\mathrm{W}^t} \mid b_t^{d-1}, \alpha_t^{d-1}, b_t^d, \alpha_t^d, \alpha^d_{\mathrm{A}^{t-1}}, \alpha^d_{\mathrm{W}^{t-1}}, \alpha^{d-1}_{\mathrm{W}^t})$
      σ'_t = substitute newly re-bound $\alpha^d_{\mathrm{A}^t}$, $\alpha^d_{\mathrm{W}^t}$ into a copy of σ_t
    else
      β_t = ASSIGNNEXTVALUES(P_t,σ_{t-1},ρ_t,σ_t) ▷ Side step at P_t
      score′ = score $- \log \mathsf{P}_{\theta_{Ce}}(\beta_t \mid \alpha^d_{\mathrm{W}^t}, d)$
      σ'_t = substitute newly re-bound β_t into a copy of σ_t
    end if
    return ρ_t, σ'_t, score′
  end if
end function

Algorithm 4.11 Determine whether a side step is well defined at reduce state ρ_t with store state σ_t

```
1:  function CANSTEPSIDEWAYS(D,ρ_t,σ_t)
2:      Array b contains all values in the boolean domain
3:      Array N contains all values in the nonterminal domain
4:      d = DEEPESTBOUNDLEVEL(σ_t,D)
5:      if d = 0 then
6:          d = SHALLOWESTBOUNDLEVEL(ρ_t,D)
7:          i,j = map[R_t^d,σ_{t-1},ρ_t,σ_t]
8:          if i < |b| or j < |N| then
9:              return true                          ▷ Can side step at R_t^d
10:         else
11:             return false                         ▷ Can not side step at R_t^d
12:         end if
13:     else if d <= D then
14:         i,j = map[S_t^d,σ_{t-1},ρ_t,σ_t]
15:         if i < |N| or j < |N| then
16:             return true                          ▷ Can side step at S_t^d
17:         else
18:             return false                         ▷ Can not side step at S_t^d
19:         end if
20:     else
21:         i = map[P_t,σ_{t-1},ρ_t,σ_t]
22:         if i < |N| then
23:             return true                          ▷ Can side step at P_t
24:         else
25:             return false                         ▷ Can not side step at P_t
26:         end if
27:     end if
28: end function
```

Algorithm 4.12 Process one observed word.

```
1:  function ParseWord(D,σ_{t−1},x_t,queue)
2:      Initialize reduction state ρ_t with all variables unbound
3:      Initialize store state σ_t with all variables unbound
4:      for all σ_{t−1},score in σ_{t−1} do
5:          Enqueue(queue,⟨σ_{t−1},ρ_t,σ_t,score,false⟩)        ▷ Seed queue with initial tuples
6:      end for
7:      Initialize empty store state collection σ_t for storing results
8:      repeat
9:          ⟨σ_{t−1}, ρ_t, σ_t, score, complete⟩ = Dequeue(queue)    ▷ Get next tuple from queue
10:         if complete then
11:             StoreResult(σ_t,⟨σ_t,score⟩)                  ▷ Record a fully bound store state
12:         else
13:             Expand(queue,σ_{t−1},ρ_t,σ_t,score)           ▷ Perform forward and side steps
14:         end if
15:     until StoppingCondition(queue,σ_t)
16:     return σ_t
17: end function
```

Algorithm 4.13 Perform forward step and side steps

```
1:  function Expand(queue,σ_{t−1},ρ_t,σ_t,score)
2:      ⟨ρ'_t, σ'_t, score', complete'⟩ = ForwardStep(D,σ_{t−1},ρ_t,σ_t,x_t,score)
3:      Enqueue(queue,⟨σ_{t−1},ρ'_t, σ'_t, score', complete'⟩)    ▷ Add new tuple to queue
4:      while CanStepSideways(D,ρ'_t,σ'_t) do
5:          ⟨ρ'_t, σ'_t, score'⟩ = SideStep(D,σ_{t−1},ρ'_t,σ'_t,score)
6:          Enqueue(queue,⟨σ_{t−1},ρ'_t, σ'_t, score', false⟩)    ▷ Add new tuple to queue
7:      end while
8:  end function
```

model θ_S is consulted to calculate the conditional joint probability of $\alpha^d_{\mathrm{A}^t}$ and $\alpha^d_{\mathrm{W}^t}$ given the random variables at the parent nodes of S^d_t.

- If all nodes $R_t^{1\ldots D+1}$ and $S_t^{0\ldots D}$ are bound, and a forward step is taken into hidden node P_t, preterminal expansion model θ_{Ce} is consulted to calculate the conditional probability of β_t given the random variables at the parent nodes of P_t.

Given ρ_t and σ_t, if a side step is taken, the random variables in the most recently bound node will be bound to new values. For such a side step to be well defined, there must be some value in the respective domains of the random variables at that node which has not yet been enumerated (given the combination of other variable bindings in ρ_t and σ_t).

Algorithm 4.11 formally defines the conditions under which a side step is well defined. This function, CANSTEPSIDEWAYS, first identifies the most recently bound node. Using the maps defined in Algorithms 4.2, 4.3, and 4.4, the function determines whether there exists some new value which can be assigned to one of the random variables at that node. If there is, the function returns **true**, indicating that a side step is well defined at ρ_t and σ_t. Otherwise, the function returns **false**, indicating that a side step is not well defined at ρ_t and σ_t.

We now have defined all component functions that we will use to fully define the task of successively enumerating all possible value combinations over the random variables in the HHMM hidden nodes at time t. Algorithm 4.12 formally defines this process as function PARSEWORD. This function takes as input the collection of store state values $\boldsymbol{\sigma}_{t-1}$ from the previous time step, the observation x_t from the current time step, and a queue. For exhaustive enumeration, this queue is defined as an initially empty unbounded first-in, first-out queue.

The queue elements are defined as a tuple containing σ_{t-1}, ρ_t, σ_t, the most recently calculated score for this tuple, and a boolean flag. The boolean flag indicates whether all required calculations for this tuple are complete; the flag will be **true** only once all variables in σ_{t-1}, ρ_t, and σ_t are bound, and the probability of the observation given the hidden state has been incorporated into the score (see line 22 of Algorithm 4.7).

Algorithm 4.12 begins by seeding the queue initial tuples. For each store state value σ_{t-1} in the collection $\boldsymbol{\sigma}_{t-1}$ of previous store state values, a new tuple is added to the queue (lines 4–6). The new tuple contains σ_{t-1}, along with an empty reduction state ρ_t and an empty store state σ_t; all variables in ρ_t and σ_t are unbound at this point. Additionally, an empty store state collection $\boldsymbol{\sigma}_t$ is initialized to store results as they are calculated (line 7).

The algorithm proceeds by removing elements from the queue (line 9). If the tuple's boolean flag is **true**, indicating that all required calculations for the tuple are complete and all variables in the tuple are bound, then the tuple constitutes a complete path through the hidden state at time t. The completed store state from the tuple and its corresponding score are added to the results collection. The function STORERESULTS can be defined as a summing map function, for each store state σ_t in $\boldsymbol{\sigma}_t$, the associated score is added to a running sum for that store state.

If the tuple's boolean flag is **false**, then the tuple represents a partial path through the hidden state at time t. Algorithm 4.13 is called to add new elements to the queue built by expanding the current tuple. A forward step is performed first, with the tuple as a starting point (line 2). The resulting tuple will either have one (previously unbound) variable bound, or will be complete. In either case, the resulting tuple is added to the queue. The algorithm next attempts to perform all possible side steps (lines 4–7) from the tuple created by stepping forward; each tuple created by a side step is also added to the queue.

Algorithm 4.12 continues until a STOPPINGCONDITION is satisfied. For exhaustive enumeration, the STOPPINGCONDITION simply checks to see if the queue is empty. Once the stopping condition has been satisfied, the resulting store state collection $\boldsymbol{\sigma}_t$ is returned.

4.3.3 Syntactic Language Model Score

The score associated with a complete path through ρ_t and σ_t represents the conditional joint negative log probability of ρ_t and σ_t given the previous store state σ_{t-1} and the current observation x_t. By summing over all scores in $\boldsymbol{\sigma}_t$(Algorithm 4.14), we obtain the total probability mass for the collection. This value represents a syntactic language model probability of the observed sequence of words $x_1 \ldots x_t$ given the grammar of the language.

Algorithm 4.14 Sum the combined log probability mass from an entire store state collection.

```
1: function SCORE(σ_t)
2:     sum = 0
3:     for all σ_t,score in σ_t do
4:         sum += score
5:     end for
6:     return sum
7: end function
```

4.4 Faster Parsing

In practice, exhaustive enumeration with no pruning is prohibitively slow. When calculating the syntactic language model probability $\mathsf{P}(X_{1\ldots t})$ for an observed sequence of words $x_1 \ldots x_t$, $|\boldsymbol{Y}|$ possible states must be examined at each of t hidden nodes $Y_{1\ldots t}$. If syntactic states are very basic, consisting for example of simple part-of-speech tags, then $|\boldsymbol{Y}|$ is small and the overall syntactic language model calculation is quite tractable. However, in some cases $\boldsymbol{Y}$ may encompass all possible phrase structure trees in the target language. In these cases, $|\boldsymbol{Y}|$ is extremely large.

We now examine the worst case computational complexity of the hidden state transition from Y_{t-1} to Y_t at each time step t. Hidden state Y_t is factored into d S nodes, $d+1$ R nodes, and one P node.

Each hidden S node maintains an active and an awaited component, and there are d random variable S nodes at each of t time steps. The set of phrase structure nonterminal symbols $\boldsymbol{N}$ defines the domain of possible values for the active component α_{A} and awaited component α_{W} of each S node and for the preterminal β_0 of each P node. The domain size $|\boldsymbol{N}|$ is roughly 400 nonterminal categories.

In the worst case, there may be $|\boldsymbol{N}|^{2d+1}$ unique element combinations stored at $S_{t-1}^{1\ldots d}$ and P_{t-1}. Another 2^d combinations of boolean values may be stored at $R_t^{1\ldots d}$; the nonterminal value α stored at each R node can is deterministic given the boolean value and the value of the S nodes at the previous time step, and so can be ignored in the complexity calculations. Finally, another $|\boldsymbol{N}|^{2d+1}$ unique element combinations may be stored at $S_{t-1}^{1\ldots d}$ and P_{t-1}. The resulting worst case complexity for a hidden state transition is $\boldsymbol{O}(|\boldsymbol{N}|^{4d+2} \cdot 2^d)$.

4.4.1 Beam Pruning

To speed parsing, beam pruning is employed in the hidden nodes at each time step. We now define beam pruning for the hidden nodes of our HHMM.

The prior probability distribution at the S nodes at time 0 is deterministic such that each random variable has exactly one value. As the parser encounters the t^{th} observed word, values for the hidden nodes at time step t are calculated through exhaustive enumeration. Once all $|\boldsymbol{N}|^{2d+1}$ possible values are calculated for the hidden state at the time step t, the values are

sorted according to their current probabilities. The b most probable nonterminal value combinations are maintained in the S nodes at the time step t. All other nonterminal value combinations are pruned. Algorithm 4.15 defines the pruning function.

When the parser encounters the next observed word, there will be no more than n combinations of values at the previous hidden nodes. This drops the worst case number of transitions at each time step from $\boldsymbol{O}(|\boldsymbol{N}|^{4d+2} \cdot 2^d)$ to $\boldsymbol{O}(|\boldsymbol{N}|^{2d+1} \cdot b \cdot 2^d)$. While the "1-best" parse calculated in this way is no longer guaranteed to be the most probable of all possible parses (because of possible errors introduced by pruning), we nevertheless use beam pruning in all experiments involving our syntactic language model.

Algorithm 4.16 defines a complete linear-time HHMM parser with beam pruning.

Algorithm 4.15 Beam pruning for store state σ_t

function PRUNE($\boldsymbol{\sigma}_t$,n)
    Sort elements of $\boldsymbol{\sigma}_t$ in increasing order according to score (negative log probability)
    Initialize empty store state collection $\boldsymbol{\sigma}'_t$ for storing pruned results
    for i **in** $1 \ldots n$ **do**
        σ_t,score = $\boldsymbol{\sigma}_t[i]$
        STORERESULT($\boldsymbol{\sigma}'_t$,σ_t,score)
    end for
    return $\boldsymbol{\sigma}'_t$
end function

Algorithm 4.16 Process a sequence of observed words

function PARSESEQUENCE(D,n,$\boldsymbol{\sigma}_0$,$x_1 \ldots x_t$)
    Initialize queue as an initially empty, unbounded first-in, first-out queue
    score = 0
    for i **in** $1 \ldots t$ **do**
        $\boldsymbol{\sigma}_i$ = PARSEWORD(D,$\boldsymbol{\sigma}_{i-1}$,$x_i$,queue)
        PRUNE($\boldsymbol{\sigma}_i$,n)
    end for
    return $\boldsymbol{\sigma}_t$
end function

4.4.2 Parallel Processing

The enumeration of paths through the HHMM is an embarrassingly parallel task. This is especially true in exhaustive enumeration, but remains true even when beam pruning is used. Parsing time may be reduced by employing multiple threads of computation. To run Algorithm 4.12 in parallel, each instance of the main loop (lines 8–15) should be run in its own thread. As long as a thread-safe queue implementation is used, this change is sufficient to parallelize the algorithm. Before the implementation of uniform cost search (Section 4.4.3), earlier implementations of the HHMM parser implemented parallel processing using this approach (in conjunction with beam pruning).

4.4.3 A* and Uniform Cost Search

Algorithm 4.14 calculates the total syntactic language model probability mass of all unpruned paths though the HHMM at one time step. When the HHMM is used as an explicit language model, this behavior is desired. However, when the HHMM is used as a parser, rather than as a language model, the desired result is the 1-best parse or the set of k-best parses. For parsing, it is therefore desirable to identify a search strategy alternative to exhaustive search. Exhaustive search is optimal, because it is guaranteed to find the lowest-cost path, but is not optimally efficient, because unnecessary suboptimal paths are explored while identifying the lowest-cost path. The ideal search strategy would be one that is both optimal and optimally efficient.

The search process can be viewed as a search through an implicit graph structure. Given a starting point in the implicit graph (the set of store states at time t-1) and a definition of the goal (the lowest-cost store state at time t where all variables are bound), Dijkstra's algorithm can be used to find the minimum cost path between the two specified points in the graph (Dijkstra, 1959).

A* search is a generalization of Dijkstra's algorithm which allows heuristic functions to guide the search path. In A* search (Equation 4.39), the score at a node n in the search space is the actual calculated cost $g(n)$ to that node plus a heuristic estimated cost $h(n)$ from that node to the goal. A* search is both optimal and optimally efficient for any given admissible and consistent heuristic (Dechter and Pearl, 1985).

$$f(n) = g(n) + h(n) \tag{4.39}$$

Uniform cost search (Russell and Norvig, 2003) is a special case of A* search (Hart et al., 1968, 1972) where the A* heuristic function $h(\cdot)$ always returns a constant (possibly zero) value c, regardless of the node n being evaluated by the heuristic function (Equation 4.40).

$$h(n) = c \tag{4.40}$$

We employ uniform cost search with $h(n) = 0$ as an alternative to exhaustive search. Uniform cost search can be implemented by employing a priority queue in Algorithm 4.12 instead of a first-in first-out queue. This search process could be further enhanced by processing multiple queue elements in parallel; we leave this speed enhancement as future work. When removing an item (line 9), the priority queue always removes the lowest cost item from the priority queue. In our case, a tuple's score serves as its cost.

Additionally, a new STOPPINGCONDITION function must be defined for uniform cost search. This STOPPINGCONDITION function returns **true** if the priority queue is empty, or if k unique store states have been stored in the results collection $\boldsymbol{\sigma}_t$.

4.4.4 Inexact Estimation of Syntactic Language Model Scores

In practice, HHMM parsing using uniform cost search (Section 4.4.3) is substantially faster than HHMM parsing using exhaustive search (Section 4.3.2), even when parallel processing is employed (Section 4.4.2).

The use of uniform cost search is problematic when the HHMM is used as a syntactic language model, rather than as a parser. In parsing, the desired result is the highest scoring single parse or k-best parses. Parsing via uniform cost search takes advantage of this fact; parsing stops once the k highest-scoring store states are obtained.

The syntactic language model score is calculated in Algorithm 4.14 by summing the scores in the store state collection $\boldsymbol{\sigma}_t$. In parsing terminology, this value is called the forward score, and represents the total probability mass of all parses. The exact estimation of the forward score requires all possible parse derivations to be enumerated via exhaustive search. The collection $\boldsymbol{\sigma}_t$ returned from uniform cost search represents only the k best derivations. The score of the 1 best derivation represents, in parsing terminology, the Viterbi score.

In our experiments, we use uniform cost search, and accept the sum over the k items in $\boldsymbol{\sigma}_t$ as an inexact estimation of the syntactic language model score. It is a common assumption in

parsing that the Viterbi score dominates the probability mass in the forward score, but whether this assumption is true is an empirical question which we have not evaluated.

It should be possible to obtain a more accurate estimate of the syntactic language model score by attempting to capture more of the forward score probability mass. To do so, a new STOPPINGCONDITION function could be defined. Rather than stopping when k unique store states have been stored in the collection $\boldsymbol{\sigma}_t$, the STOPPINGCONDITION function could continue until some specified percentage of the total possible probability mass has been accounted for in the store state collection $\boldsymbol{\sigma}_t$. Additionally (or alternatively), the STOPPINGCONDITION function could return **true** once some maximum amount of processing time has elapsed. In any case, we leave the implementation of such a STOPPINGCONDITION function into the HHMM as future work.

4.4.5 From Cube Pruning to Lazy Queue Expansion

Cube pruning (Chiang, 2007) is a method for efficiently integrating an n-gram language model into the translation search space of a hierarchical phrase-based decoder (Chiang, 2005), using the lazy k-best algorithm of Huang and Chiang (2005). The key insight of cube pruning is a technique for efficiently searching through a three-dimensional space in a best-first manner for the k highest-scoring results. This search strategy has been adapted to other related problems, including k-best monolingual parsing (Huang and Chiang, 2005) and bilingual word alignment (Riesa and Marcu, 2010).

Gesmundo and Henderson (2010) concisely present the common core of cube pruning as follows. Given an ordered set of random variables, cube pruning attempts to find the most probable combination of assignments for the random variables. For each random variable, a list is maintained containing all elements from the domain of that random variable, sorted from most probable to least probable. The algorithm begins by taking the first (most probable) element from each list and assigning that element to the respective random variable. Since each list is sorted, this is guaranteed to result in the most probable combination of random variable assignments[4] The neighboring combinations of variable assignments are assigned to a priority queue, and the algorithm proceeds by processing subsequent assignment combinations from the priority queue.

[4] Assuming that the search space is monotonic. In our case, the search space is monotonic. In Chiang (2007), the search space is non-monotonic, and consequently this guarantee does not hold there.

Hopkins and Langmead (2009) show that this algorithm is highly similar to A* search, and argue that cube pruning in its original context (Chiang, 2007) is in fact simply A* search applied to a particular machine translation problem, making use of a particular search heuristic.

Pust and Knight (2009) examine whether cube pruning in a machine translation context can be sped up through what they call "pervasive laziness." During A* search in general, and cube pruning in particular, after a search space element is examined, all possible successor states are immediately added to the priority queue. In the worst case, all elements added to the priority queue will eventually be examined. However, in practice, the A* stopping condition often triggers far before all search states have been exhaustively enumerated. When this happens, many elements remain on the priority queue; each of these elements represents a potential result that was neither needed nor examined. Gesmundo and Henderson (2010) attempt to minimize the number of elements added to the priority queue to only those needed to find the final results. When a search space element is examined, only the immediate successor states are added. Others will be added later, but each element will be added to the priority queue on an as-needed basis.

Algorithm 4.17 Perform forward step and side step

```
1: function LazyExpand(queue,σ_{t−1},ρ_t,σ_t,score,parentScore)
2:     ⟨ρ'_t, σ'_t, score', complete'⟩ = ForwardStep(D,σ_{t−1},ρ_t,σ_t,x_t,score)
3:     Enqueue(queue,⟨σ_{t−1},ρ'_t, σ'_t, score',score, complete'⟩)      ▷ Add new tuple to queue
4:     if CanStepSideways(D,ρ'_t,σ'_t) then
5:         ⟨ρ'_t, σ'_t, score'⟩ = SideStep(D,σ_{t−1},ρ'_t,σ'_t,parentScore)
6:         Enqueue(queue,⟨σ_{t−1},ρ'_t, σ'_t, score',parentScore, false⟩)      ▷ Add new tuple to q.
7:     end if
8: end function
```

We use a very similar strategy to lazily expand elements onto our HHMM priority queue during uniform cost search. When parsing (line 13 of Algorithm 4.12), we replace the call to Expand (as implemented by Algorithm 4.13) with a call to LazyExpand, implemented by Algorithm 4.17. Previously, each tuple was expanded by taking a forward step followed by all possible side steps from that forward step. In this alternative approach, each tuple is expanded by taking a single side step and a single forward step. While we have not evaluated parsing speed using LazyExpand against uniform cost search using Expand, a comparable change

resulted in an 11% speed increase when applied to cube pruning during translation (Gesmundo and Henderson, 2010). In the worst case, LAZYEXPAND will explore the same search space as EXPAND, and in more optimistic settings it will explore a smaller search space, resulting in faster parsing times.

As with cube pruning, using LAZYEXPAND requires that the domains of random variables be enumerated in sorted order. For each random variable, the values with the highest probability (given the requisite conditioning values) must be the first value enumerated. Subsequent values must be presented in decreasing order of conditional probability. To this end, we define new implementations of the ASSIGNNEXTVALUES function previously defined in Algorithm 4.2, Algorithm 4.3, and Algorithm 4.4. The new implementations of ASSIGNNEXTVALUES are defined such that given values of the appropriate conditioning variables, the possible values of the modelled variables are enumerated in sorted order, according the the relevant conditional probability model (θ_R, θ_S, or θ_P).

4.5 Conclusion

An incremental parser processes each input token text sequentially, from the word in a sentence to the last. After processing the t^{th} token in string $\mathbf{e}$, an incremental parser has some internal representation of possible hypothesized (incomplete) trees, $\boldsymbol{\tau_t}$. Generically, any incremental parser and grammar which together implement an appropriate probability mass function $\mathsf{P}(\tilde{\boldsymbol{\tau}}_t)$ and transition function $\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1})$ together define an incremental syntactic language model.

We obtain the grammar component of our incremental syntactic language model by formally redefining the right-corner model transform of Schuler (2009) in terms of the probabilistic context-free grammar we trained in Section 3.3. The resulting right-corner PCFG fully defines the model parameters of a Hierarchical Hidden Markov Model. We formally defined the algorithms necessary to incrementally parse using our HHMM with our right-corner PCFG. Algorithm 4.12 defines the required transition function $\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1})$ declared in Equation 3.6. Algorithm 4.14 defines the required probability mass function $\mathsf{P}(\tilde{\boldsymbol{\tau}}_t)$ declared in Equation 3.5. Together, these components constitute our incremental syntactic language model.

In Chapter 5 that follows, we integrate our incremental syntactic language model into the decoding algorithm for statistical phrase-based machine translation.

Chapter 5

Applying Incremental Syntactic Language Models to Phrase-based Translation

We argue that incremental parsers, used as syntactic language models, provide an appropriate algorithmic match to incremental phrase-based machine translation. In this chapter, we integrate an incremental syntactic language model directly into phrase-based translation.[1] The incremental syntactic language model we use is defined by the HHMM parsing algorithms described in Chapter 4, using the probabilistic phrase structure grammar in right-corner form trained on the transformed treebank from Chapter 3. The method we describe is both novel and general; in principle this method can be used to incorporate any generative incremental language model into phrase-based machine translation. This method re-exerts the role of the language model as a mechanism for encouraging syntactically fluent translations.

Early work in statistical machine translation viewed translation as a noisy channel process comprised of a translation model, which functioned to posit adequate translations of source language words, and a target language model, which guided the fluency of generated target language strings (Brown et al., 1990). The noisy channel approach was later generalized (Och and Ney, 2002), allowing multiple models to be combined in a maximum entropy framework.

[1] This chapter is based on Schwartz et al. (2011), originally published as "Incremental Syntactic Language Models for Phrase-based Translation" in the Proceedings of the Annual Meeting of the Association for Computational Linguistics.

Most current statistical machine translation research uses the maximum entropy framework. We formally define the noisy channel and maximum entropy frameworks in Section 2.2. In both approaches, the target language model plays a critical role guiding the order and choice of target language words.

To define the target language model, statistical machine translation research typically follows widespread practice in speech recognition by using n-gram language models (Shannon, 1948, 1951) to model the prior probability of a contiguous sequence of words. These n-gram models are relatively simple finite-state models, and do not incorporate knowledge about the syntax of the language. Modern phrase-based translation using large scale n-gram language models generally performs well in terms of lexical choice, but still often produces ungrammatical output.

Syntactic parsing may help produce more grammatical output by better modelling structural relationships and long-distance dependencies. Bottom-up and top-down parsers typically require a completed string as input; this requirement makes it difficult to incorporate these parsers into phrase-based translation, which generates hypothesized translations incrementally, from left-to-right.[2] In some cases, it is possible to take a parser which was not designed to be incremental and adapt it to approximate incremental parsing behavior for use as a syntactic language model in phrase-based machine translation (Galley and Manning, 2009); however such adaptation is far from straightforward and may require non-trivial modification of the parsing algorithm.

As a workaround, parsers can rerank the translated output of translation systems (Och et al., 2004). It is relatively straightforward to use a syntactic model to rerank an n-best list. However, reranking is not the ideal mechanism to incorporate knowledge of syntax. Even when n is very large, the hypothesized translations present in an n-best list represent only a tiny fraction of the hypothesis space considered during translation. A syntactic model used to rerank can only choose among the n translations present in the n-best list.

A syntactic model incorporated as a feature function (Equation 2.9 on page 18) in a maximum entropy framework is much more powerful influence; such a model can influence the seach process directly, allowing many more (hopefully more syntactically well-formed) hypotheses to be explored. In this work, we present a mechanism to allow incremental parsers

[2] While not all languages are written left-to-right, we will refer to incremental processing which proceeds from the beginning of a sentence as left-to-right.

(Roark, 2001; Henderson, 2004; Schuler et al., 2010; Huang and Sagae, 2010), which process input in a straightforward left-to-right manner, to be directly incorporated as a syntactic language model feature function into the maximum entropy framework of phrase-based machine translation.

The remainder of this chapter is organized as follows:

- Section 5.1 formally defines the primary novel contribution of this dissertation, a general method for integrating syntactic language models into phrase-based translation.
- Section 5.2 briefly reviews the incremental HHMM parser from Chapter 4. We will use this parser as an incremental syntactic language model.
- Section 5.3 describes how our syntactic language model is integrated into Moses, a phrase-based statistical machine translation system.
- Section 5.4 presents empirical results that demonstrate substantial improvements in perplexity for our syntactic language model over traditional n-gram language models.
- Integration of our syntactic language model into phrase-based translation comes with a cost to translation speed. Section 5.4.2 examines this issue and presents a mechanism for alleviating the problem.

5.1 Parser as Syntactic Language Model in Phrase-based Translation

Parsing is the task of selecting the representation $\hat{\tau}$ (typically a tree) that best models the structure of sentence $\mathbf{e}$, out of the set of all such possible structural representations $\boldsymbol{\tau}$. This set of representations may be all phrase structure trees or all dependency trees allowed by the parsing model. Typically, tree $\hat{\tau}$ is taken to be:

$$\hat{\tau} = \underset{\tau}{\operatorname{argmax}}\, \mathsf{P}(\tau \mid \mathbf{e}) \tag{5.1}$$

In our case, $\boldsymbol{\tau}$ represents the set of all depth-limited right-corner phrase structure trees licensed for sentence $\mathbf{e}$ according to the grammar and parser defined in Chapters 3 and 4.

We formally define a syntactic language model $\mathsf{P}(\mathbf{e})$ based on the total probability mass over all possible trees for string $\mathbf{e}$. This is shown in Equation 5.2 and decomposed in Equation 5.3.

$$\mathsf{P}(\mathbf{e}) = \sum_{\tau \in \boldsymbol{\tau}} \mathsf{P}(\tau, \mathbf{e}) \tag{5.2}$$

$$\mathsf{P}(\mathbf{e}) = \sum_{\tau \in \boldsymbol{\tau}} \mathsf{P}(\mathbf{e} \,|\, \tau)\mathsf{P}(\tau) \tag{5.3}$$

5.1.1 Incremental Syntactic Language Model

An incremental parser processes each token of input sequentially from the beginning of a sentence to the end, rather than processing input in a top-down (Earley, 1968) or bottom-up (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967) fashion. Tokens will commonly be words, but may be morphemes (used in psycholinguistics sentence processing research) or characters (used in Chinese parsing). After processing the t^{th} token in string $\mathbf{e}$, an incremental parser has some internal representation of possible hypothesized (incomplete) trees, $\boldsymbol{\tau_t}$. The syntactic language model probability of a partial sentence $e_1 \ldots e_t$ is defined:

$$\mathsf{P}(e_1 \ldots e_t) = \sum_{\tau \in \boldsymbol{\tau_t}} \mathsf{P}(e_1 \ldots e_t \,|\, \tau)\mathsf{P}(\tau) \tag{5.4}$$

In practice, a parser may constrain the set of trees under consideration to $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t}}$, that subset of analyses or partial analyses that remains after any pruning is performed. An incremental syntactic language model can then be defined by a probability mass function (Equation 5.5) and a transition function δ (Equation 5.6). The role of δ is explained in Section 5.1.3. Any parser which implements these two functions can serve as a syntactic language model.

$$\mathsf{P}(e_1 \ldots e_t) \approx \mathsf{P}(\tilde{\boldsymbol{\tau}}_t) = \sum_{\tau \in \tilde{\boldsymbol{\tau}}_t} \mathsf{P}(e_1 \ldots e_t \,|\, \tau)\mathsf{P}(\tau) \tag{5.5}$$

$$\delta(e_t, \tilde{\boldsymbol{\tau}}_{t-1}) \rightarrow \tilde{\boldsymbol{\tau}}_t \tag{5.6}$$

5.1.2 Decoding in Phrase-based Translation

Given a source language input sentence $\mathbf{f}$, a trained source-to-target translation model, and a target language model, the task of translation is to find the maximally probable translation $\hat{\boldsymbol{e}}$

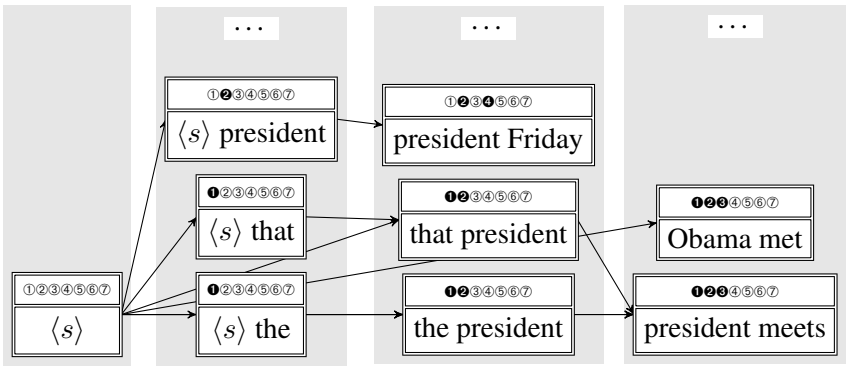


Figure 5.1: Partial decoding lattice for standard phrase-based decoding stack algorithm translating the German sentence *Der Präsident trifft am Freitag den Vorstand*. Each node h in decoding stack t represents the application of a translation option, and includes the source sentence coverage vector, target language n-gram state. Hypothesis combination is also shown, indicating where lattice paths with identical n-gram histories converge. We use the English translation *The president meets the board on Friday* as a running example.

using a linear combination of j feature functions $\boldsymbol{h}$ weighted according to tuned parameters $\boldsymbol{\lambda}$ (Och and Ney, 2002).

$$\hat{\boldsymbol{e}} = \arg\max_{\mathbf{e}} \exp(\sum_{j} \lambda_j h_j(\boldsymbol{e}, \boldsymbol{f})) \tag{5.7}$$

Phrase-based translation constructs a set of translation options — hypothesized translations for contiguous portions of the source sentence — from a trained phrase table, then incrementally constructs a lattice of partial target translations (Koehn, 2010). To prune the search space, lattice nodes are organized into beam stacks (Jelinek, 1969) according to the number of source words translated. An n-gram language model history is also maintained at each node in the translation lattice. The search space is further trimmed with hypothesis recombination, which collapses lattice nodes that share a common coverage vector and n-gram state.

Figure 5.1 illustrates the lattice of partial target translations that is constructed as source words are translated. Each node in the lattice stores the source coverage vector, which indicates which source words have been translated. Additionally, each node stores the target language n-gram state. The node in the first (left-most) stack has an empty coverage vector, indicating

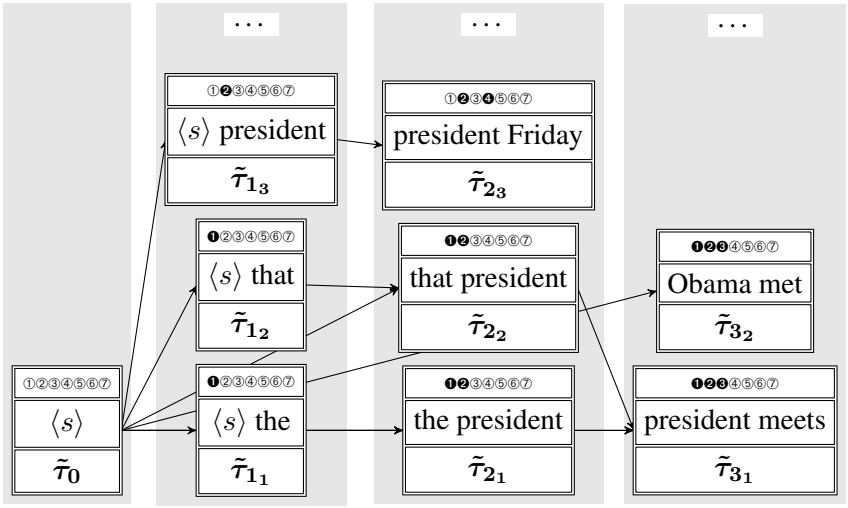


Figure 5.2: Partial decoding lattice for standard phrase-based decoding stack algorithm translating the German sentence *Der Präsident trifft am Freitag den Vorstand*. Each node h in decoding stack t is augmented with syntactic language model state $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t_h}}$.

that no words have been translated; the n-gram language model state $\langle s \rangle$ signifies the beginning of a sentence. In the next node to the right (at the bottom of the second stack), the first German word, *Der*, has been translated as *the*. This partial translation is continued in the next node to the right (at the bottom of the third stack), where the second German word, *Präsident*, is translated as *president*.

Moving one node up in the lattice (the middle node of the third stack), we see an alternate translation, where the first two source words, *Der Präsident*, are collectively translated as *the president*. The node at the bottom of the right-most stack represent a case of hypothesis recombination; the translations for each of the bottom two nodes from the third stack are extended by translating the third German word, *trifft*, as *meets*. The bigram language model history for the alternate translations *the president meets* and *that president meets* is the same: *president meets*. Consequently the alternate paths converge at this new node.

5.1.3 Incorporating a Syntactic Language Model

Phrase-based translation produces target language words in an incremental left-to-right fashion, generating words at the beginning of a translation first and words at the end of a translation last.

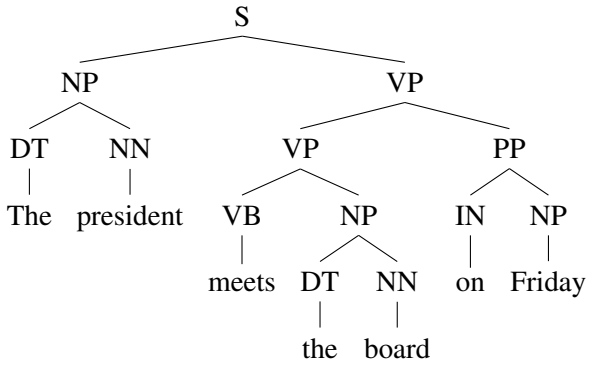


Figure 5.3: Binarized phrase structure tree for English translation *The president meets the board on Friday* of the German source language sentence *Der Präsident trifft am Freitag den Vorstand*.

Similarly, incremental parsers process sentences in an incremental fashion, analyzing words at the beginning of a sentence first and words at the end of a sentence last. As such, an incremental parser with transition function δ can be incorporated into the phrase-based decoding process in a straightforward manner. Each node in the translation lattice is augmented with a syntactic language model state $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t}}$.

The hypothesis at the root of the translation lattice is initialized with $\tilde{\tau}_0$, representing the internal state of the incremental parser before any input words are processed. The phrase-based translation decoding process adds nodes to the lattice; each new node contains one or more target language words. Each node contains a backpointer to its parent node, in which $\tilde{\boldsymbol{\tau}}_{t-1}$ is stored. Given a new target language word e_t and $\tilde{\boldsymbol{\tau}}_{t-1}$, the incremental parser's transition function δ calculates $\tilde{\boldsymbol{\tau}}_t$. Figure 5.2 illustrates a sample phrase-based decoding lattice where each translation lattice node is augmented with syntactic language model state $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t}}$.

In phrase-based translation, many translation lattice nodes represent multi-word target language phrases. For such translation lattice nodes, δ will be called once for each newly hypothesized target language word in the node. Only the final syntactic language model state in such sequences need be stored in the translation lattice node.

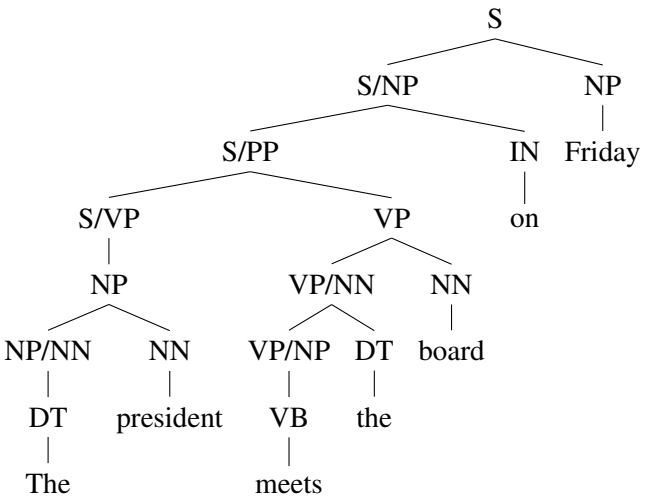


Figure 5.4: Binarized phrase structure tree after application of right-corner transform for English sentence *The president meets the board on Friday*. Constituent nonterminals in right-corner transformed tree take the form of incomplete constituents $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ consisting of an 'active' constituent α_{A} lacking an 'awaited' constituent α_{W} yet to come, similar to non-constituent categories in a Combinatory Categorial Grammar (Steedman, 2000).

5.2 Incremental Bounded-Memory Parsing with a Time Series Model

Having defined (in Section 5.1.3) the framework by which any incremental parser may be incorporated into phrase-based translation, we now review the incremental bounded-memory HHMM parser from Chapter 4 as a specific incremental syntactic language model for use in our experiments.

The parser must process target language words incrementally as the phrase-based decoder adds hypotheses to the translation lattice. To facilitate this incremental processing, recall that ordinary phrase-structure trees can be transformed into right-corner recursive phrase structure trees using the tree transforms in Schuler et al. (2010). Constituent nonterminals in right-corner transformed trees take the form of *incomplete constituents* $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$ consisting of an 'active' constituent α_{A} lacking an 'awaited' constituent α_{W} yet to come, similar to non-constituent categories in a Combinatory Categorial Grammar (Ades and Steedman, 1982; Steedman, 2000). As an example, the parser might consider VP/NN as a possible category for input "meets the".

A sample phrase structure tree is shown before and after the right-corner transform in Figure 5.3 and Figure 5.4. Our parser operates over a right-corner transformed probabilistic context-free grammar (PCFG). Parsing runs in linear time on the length of the input. This model of incremental parsing is implemented as a Hierarchical Hidden Markov Model, HHMM (Murphy and Paskin, 2001), and is equivalent to a probabilistic pushdown automaton with a bounded pushdown store. The parser runs in O(n) time, where n is the number of words in the input. This model is shown graphically in Figure 5.5 and formally defined in Chapter 4.

The incremental parser assigns a probability (Equation 5.5) for a partial target language hypothesis, using a bounded store of incomplete constituents $\alpha_{\mathrm{A}}/\alpha_{\mathrm{W}}$. The phrase-based decoder uses this probability value as the syntactic language model feature score.

5.3 Phrase-based Translation with an Incremental Syntactic Language Model

The phrase-based decoder is augmented by adding additional state data to each hypothesis in the decoder's hypothesis stacks. Figure 5.2 illustrates an excerpt from a standard phrase-based translation lattice. Within each decoder stack t, each hypothesis h is augmented with a syntactic language model state $\tilde{\tau}_{t_h}$. Each syntactic language model state is a random variable store,

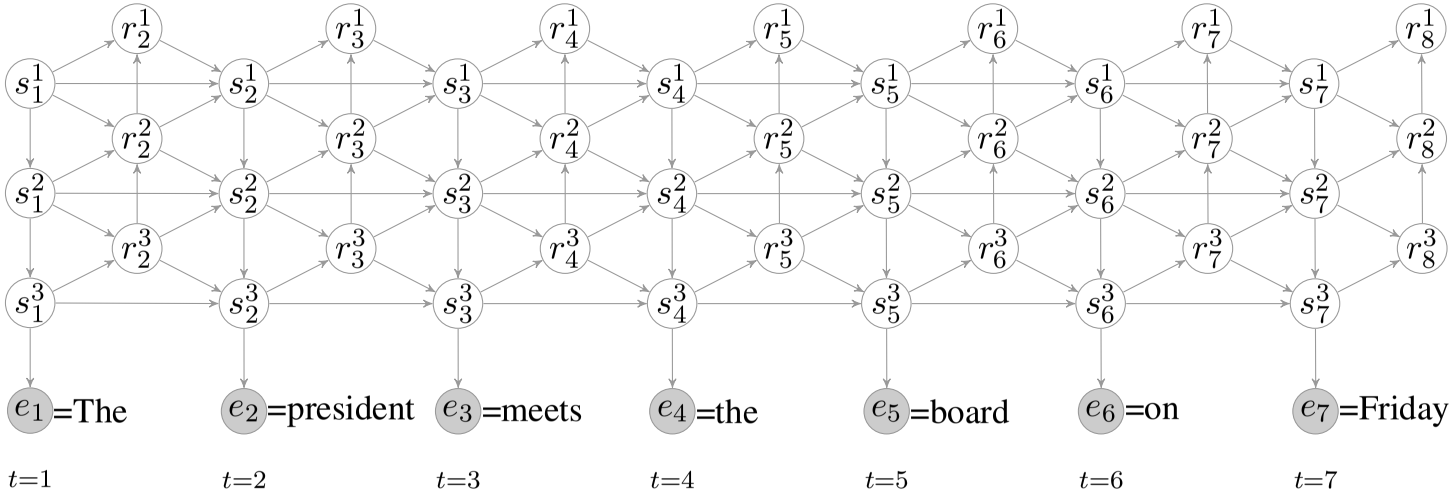


Figure 5.5: Graphical representation of a Hierarchic Hidden Markov Model with $D = 3$ hidden levels after parsing input sentence *The president meets the board on Friday*. Circles denote random variables, and edges denote conditional dependencies. Shaded circles denote variables with observed values.

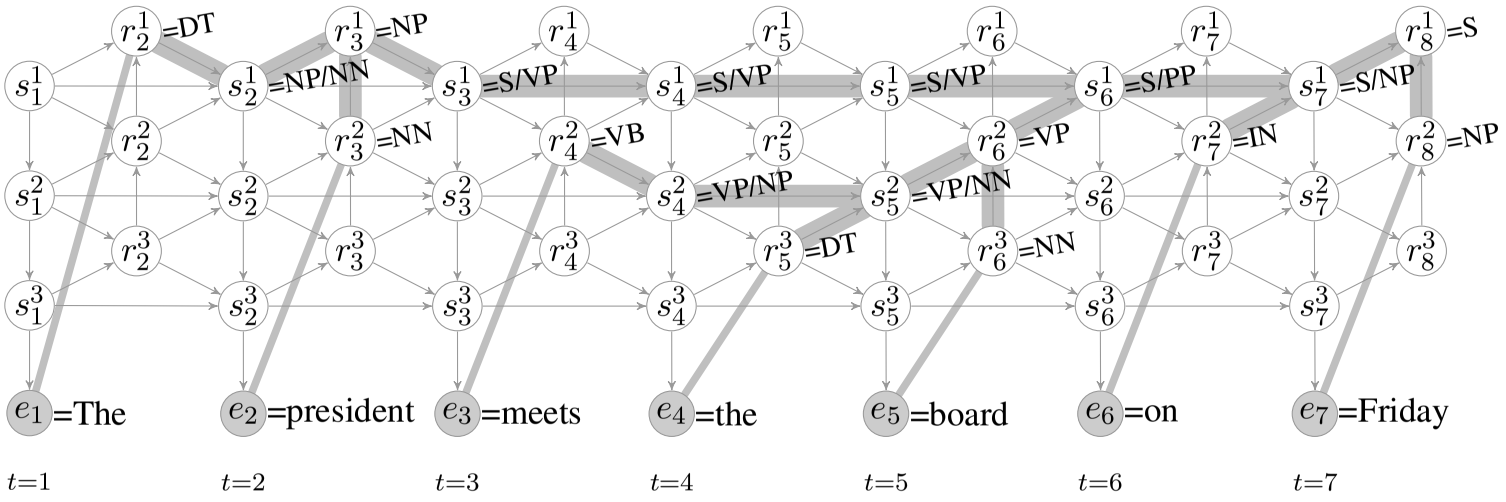


Figure 5.6: Graphical representation of the Hierarchic Hidden Markov Model after parsing input sentence *The president meets the board on Friday*. The shaded path through the parse lattice illustrates the recognized right-corner tree structure of Figure 5.4.

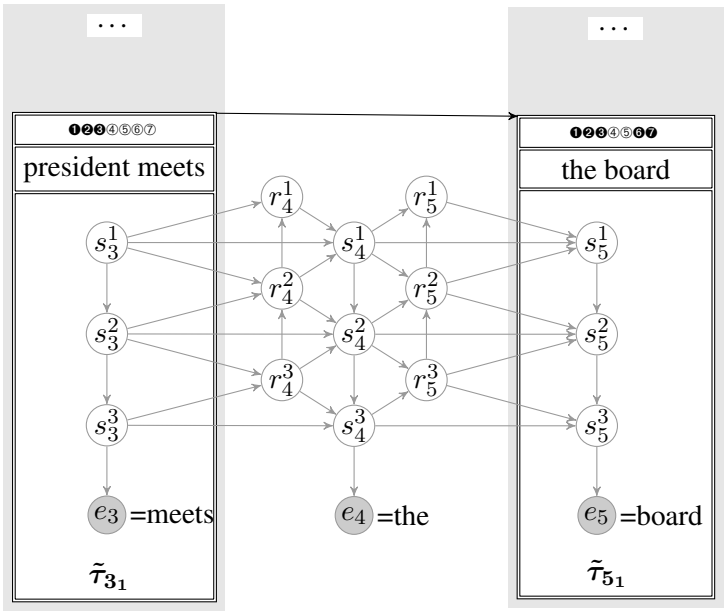


Figure 5.7: A hypothesis in the phrase-based decoding lattice from Figure 5.2 is expanded using translation option *the board* of source phrase *den Vorstand*. Syntactic language model state $\tilde{\boldsymbol{\tau}}_{\mathbf{3_1}}$ contains random variables $s_3^{1..3}$; likewise $\tilde{\boldsymbol{\tau}}_{\mathbf{5_1}}$ contains $s_5^{1..3}$. The intervening random variables $r_4^{1..3}$, $s_4^{1..3}$, and $r_5^{1..3}$ are calculated by transition function δ (Equation 5.6), but are not stored. Observed random variables ($e_3..e_5$) are shown for clarity, but are not explicitly stored in any syntactic language model state.

containing a slice of random variables from the HHMM. Specifically, $\tilde{\boldsymbol{\tau}}_{\boldsymbol{t_h}}$ contains those random variables $s_t^{1..D}$ that maintain distributions over syntactic elements.

By maintaining these syntactic random variable stores, each hypothesis has access to the current language model probability for the partial translation ending at that hypothesis, as calculated by an incremental syntactic language model defined by the HHMM. Specifically, the random variable store at hypothesis h provides $\mathsf{P}(\tilde{\boldsymbol{\tau}}_{\boldsymbol{t_h}}) = \mathsf{P}(e_{1..t}^h, s_{1..t}^{1..D})$, where $e_{1..t}^h$ is the sequence of words in a partial hypothesis ending at h which contains t target words, and where there are D syntactic random variables in each random variable store (Equation 5.5).

During stack decoding, the phrase-based decoder progressively constructs new hypotheses by extending existing hypotheses. New hypotheses are placed in appropriate hypothesis stacks. In the simplest case, a new hypothesis extends an existing hypothesis by exactly one target word. As the new hypothesis is constructed by extending an existing stack element, the store and reduction state random variables are processed, along with the newly hypothesized word. This results in a new store of syntactic random variables (Equation 5.6) that are associated with the new stack element.

When a new hypothesis extends an existing hypothesis by more than one word, this process is first carried out for the first new word in the hypothesis. It is then repeated for the remaining words in the hypothesis extension. Once the final word in the hypothesis has been processed, the resulting random variable store is associated with that hypothesis. The random variable stores created for the non-final words in the extending hypothesis are discarded, and need not be explicitly retained.

Figure 5.7 illustrates this process, showing how a syntactic language model state $\tilde{\boldsymbol{\tau}}_{\mathbf{5_1}}$ in a phrase-based decoding lattice is obtained from a previous syntactic language model state $\tilde{\boldsymbol{\tau}}_{\mathbf{3_1}}$ (from Figure 5.2) by parsing the target language words from a phrase-based translation option.

Our syntactic language model is integrated into the current version of Moses (Koehn et al., 2007).

5.4 Results

We trained the syntactic language model from Section 5.2 (HHMM) and an interpolated n-gram language model with modified Kneser-Ney smoothing (Chen and Goodman, 1998); models were trained on sections 2-21 of the Wall Street Journal (WSJ) treebank (Marcus et al., 1993).

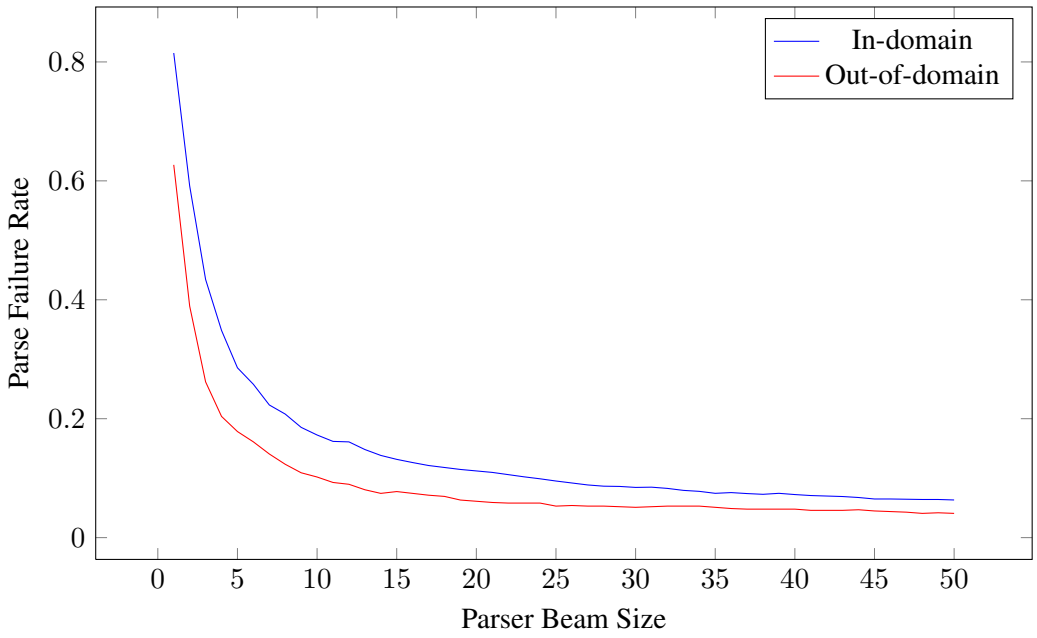


Figure 5.8: Ratio of sentences which trigger parse failure, at all parser beam sizes from 1–50.

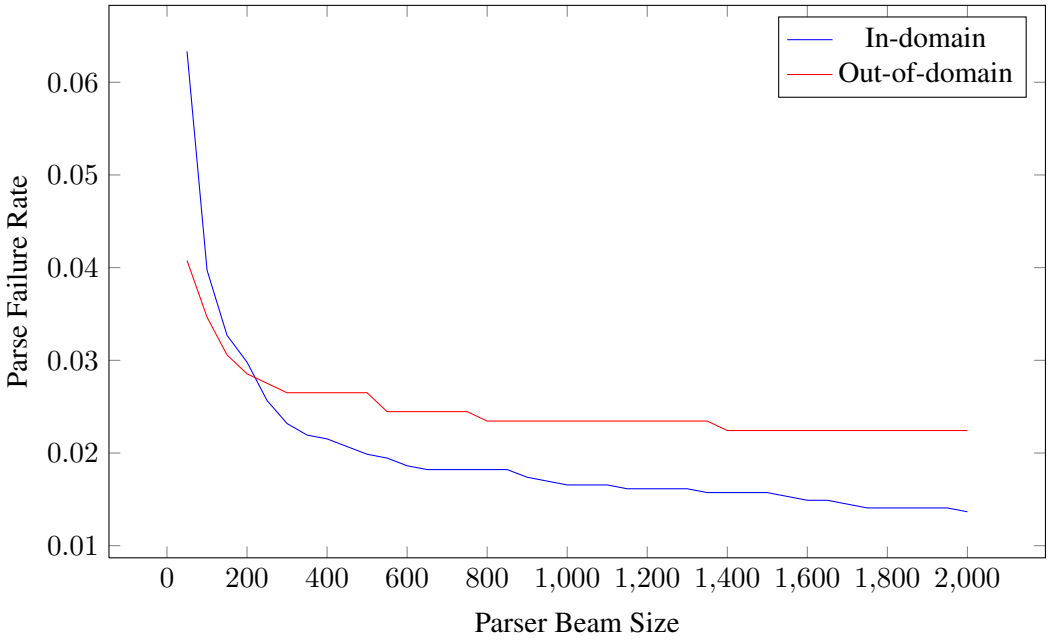


Figure 5.9: Ratio of sentences which trigger parse failure, at various beam sizes 50–2000.

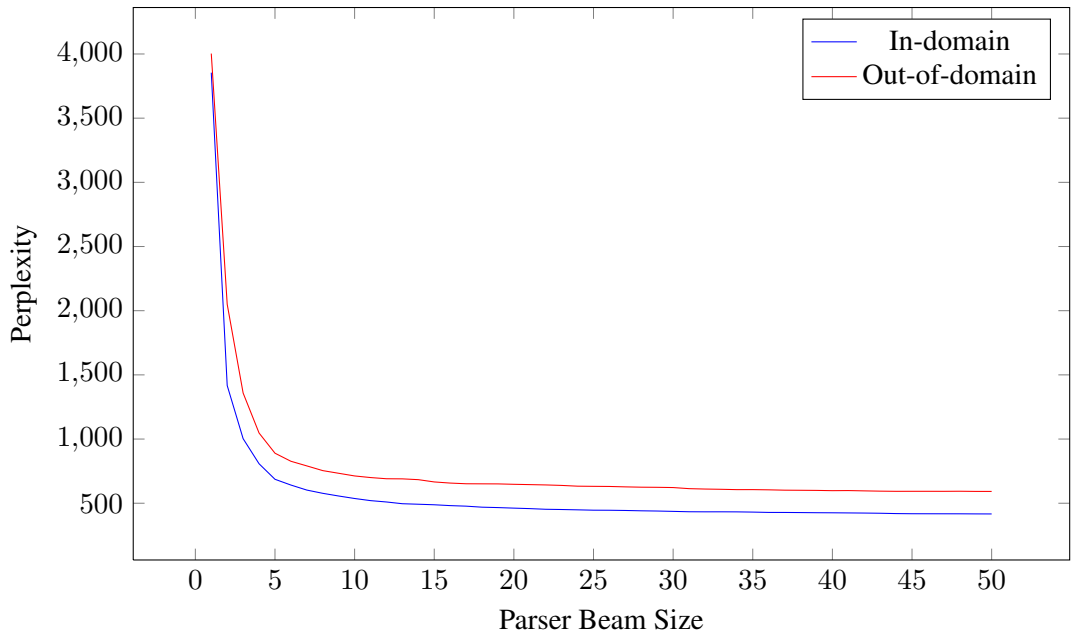


Figure 5.10: Perplexity for in-domain and out-of-domain test sets, at parser beam sizes 1–50.

5.4.1 Perplexity Results

A standard measure of language model quality is average per-word perplexity, ppl. This measure reports how surprised a model is by test data. Equation 5.8 calculates ppl using log base b for a test set of T tokens. Equation 5.8 calculates ppl using log base b for a test set of T tokens.

$$ppl = b^{\frac{-log_b \mathsf{P}(e_1 \ldots e_T)}{T}} \tag{5.8}$$

For in-domain perplexity tests we use Section 23 of the WSJ corpus. For out-of-domain perplexity tests, we use the English reference translations of the dev section, set aside in Baker et al. (2009) for parameter tuning, of the NIST Open MT 2008 Urdu-English task. This is the same dev section used for translation parameter optimization.

Because averege per-word perplexity as defined in Equation 5.8 is well-defined for only those sentences $\mathbf{e}$ with a non-zero language model probabilty $\mathsf{P}(\mathbf{e})$, in all perplexity results that we present we exclude any sentences with zero language model probability. Ideally, a language model should provide a non-zero language model probability for all sentences in the language. In practice, we apply beam pruning (as presented in Section 4.4) during syntactic

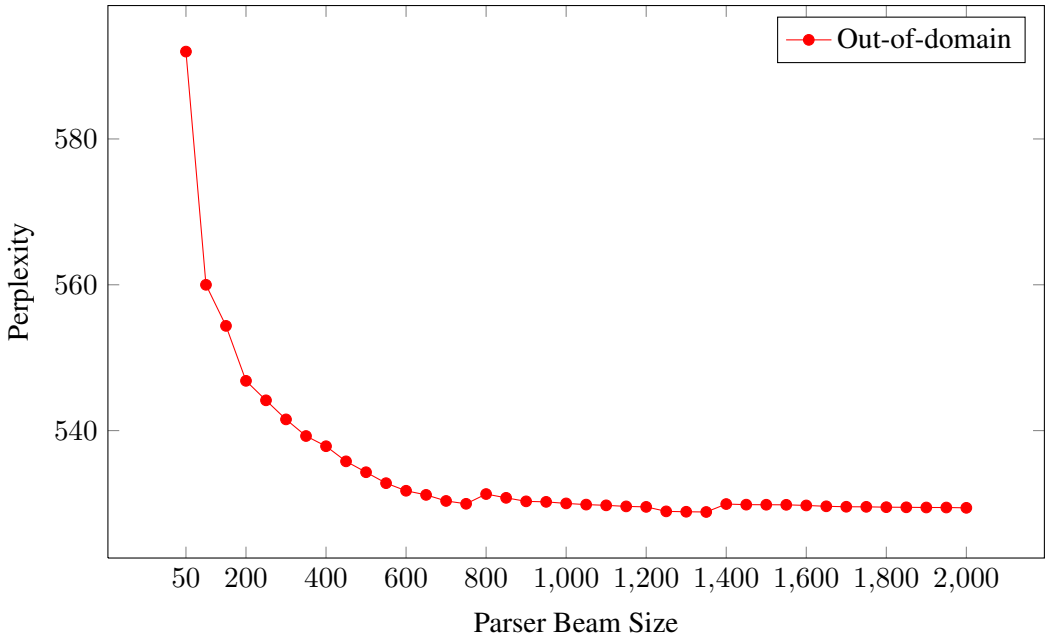


Figure 5.11: Perplexity for out-of-domain test set, at various parser beam sizes from 50–2000.

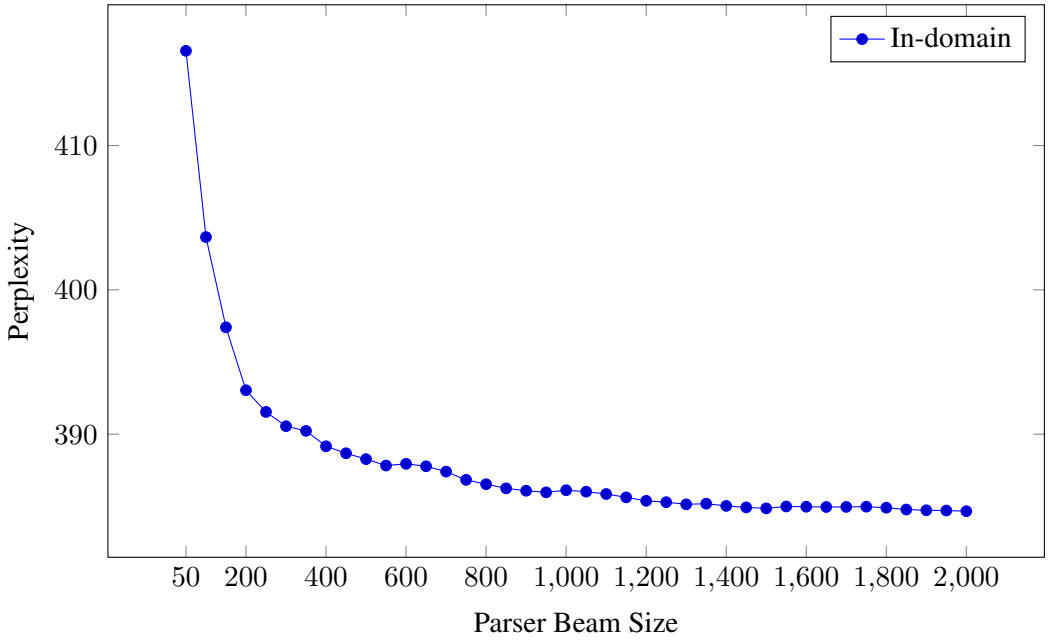


Figure 5.12: Perplexity for in-domain test set, at various parser beam sizes from 50–2000.

LM	In-domain WSJ 23 *ppl*	Out-of-domain ur-en dev *ppl*
WSJ 1-gram	1973.57	3581.72
WSJ 2-gram	349.18	1312.61
WSJ 3-gram	262.04	1264.47
WSJ 4-gram	244.12	1261.37
WSJ 5-gram	**232.08**	1261.90
WSJ HHMM	384.66	**529.41**
Interpolated WSJ 5-gram + HHMM	***209.13***	225.48
Giga 5-gram	258.35	312.28
Interp. Giga 5-gr + WSJ HHMM	222.39	***123.10***
Interp. Giga 5-gr + WSJ 5-gram	174.88	321.05

Figure 5.13: Average per-word perplexity values. HHMM was run with beam size of 2000. **Bold** indicates best single-model results for LMs trained on WSJ sections 2-21. Best overall in ***italics***.

language model parsing. Figures 5.8 and 5.9 depict the percentage of sentences which fail to parse (resulting in zero probability according to the syntactic language model) for small and large beam sizes, respectively. When the most extreme level of beam pruning is applied (beam size of 1), the parse failure rate is 63% for out-of-domain sentences and 81% for in-domain sentences. At more the more realistic beam sizes shown in Figure 5.9, the parse failure rate approaches 1% for in-domain sentences and 2% for out-of-domain sentences.

Figure 5.10 presents perplexity results obtained by applying our syntactic language model to the in-domain and out-of-domain test sets at all parser beam sizes from 1–50. In Figure 5.8 we saw a high rate of parse failure during extreme pruning with very small beam sizes; under those same conditions, sentences which do parse are assigned relatively low syntactic language

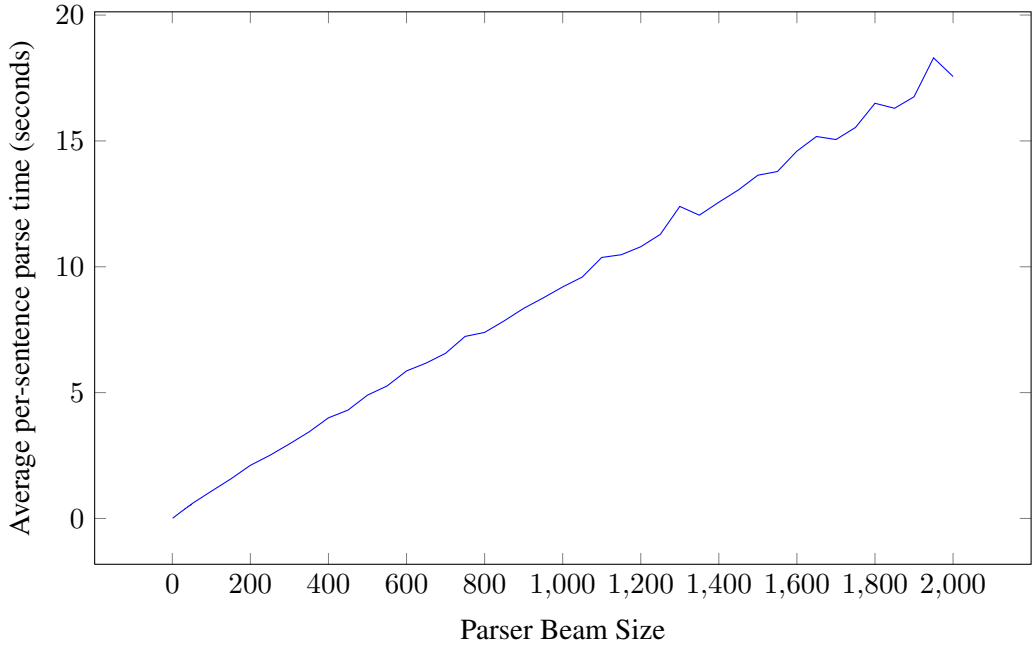


Figure 5.14: Average per-sentence parse time, measured in seconds, for beam sizes from 1–2000.

model probabilities, resulting in extremely high perplexity values for both in-domain and out-of-domain test sets. As beam size in increased, the perplexity values quickly drop for both in-domain and out-of-domain test sets. Figure 5.11 shows an out-of-domain perplexity value of 591 at beam size 50, stabilizing to near 530 at beam sizes of 500–2000. A similar result is shown in Figure 5.12, with a perplexity value of 416 at beam size 50, stabilizing to near 385 at beam sizes of 500–2000.

The HHMM outperforms the n-gram model in terms of out-of-domain test set perplexity when trained on the same WSJ data; the best perplexity results for in-domain and out-of-domain test sets are found by interpolating HHMM and n-gram LMs (Figure 5.13). To show the effects of training an LM on more data, we also report perplexity results on the 5-gram LM trained for the GALE Arabic-English task using the English Gigaword corpus. In all cases, including the HHMM significantly reduces perplexity.

5.4.2 Speed Results

As the parser beam size is increased, the time required to parse a sentence increases. Figure 5.14 plots the linear increase in average per-sentence parsing time over beam sizes ranging from 1–2000. When the parser is integrated into phrase-based translation as a syntactic language model, the beam size setting directly affects the time it takes to translate a source language sentence. In Figure 5.15, we observe that integration of our syntactic language model into phrase-based translation comes at a substantial cost to translation speed; in this section, we examine this issue and present a mechanism for alleviating the problem by distributing translation jobs across a cluster of parallel computational nodes.

Sentence length	Moses	+HHMM beam=50	+HHMM beam=2000
10	0.21	533	1143
20	0.53	1193	2562
30	0.85	1746	3749
40	1.13	2095	4588

Figure 5.15: Mean per-sentence decoding time (in seconds) for dev set using Moses with and without syntactic language model. HHMM parser beam sizes are indicated for the syntactic LM.

Ideally, all parts of a document should take the same amount of time to translate. While naive splitting techniques reduce the time required for each translation iteration by splitting the work between p computational nodes, in practice some parts may take much longer to complete than others. This can lead to significant computational slack time. To address this problem, we develop three novel algorithms for splitting translation tasks in a parallel computing environment, drawing on research in parallel machine scheduling.

Related Work: Machine Scheduling

While machine translation models could, in theory, condition on previously translated sentences, in practice virtually no widely used models do so. It is therefore very straightforward to split the data into p parts, and translate each part independently on p computational nodes.

Scripts implemented in Moses (Koehn et al., 2007) do exactly that, simply splitting the data into p arbitrary parts such that each part contains the same number of lines (Algorithm 5.1).

Algorithm 5.1 Split input text into n parts such that each part contains the same number of lines.

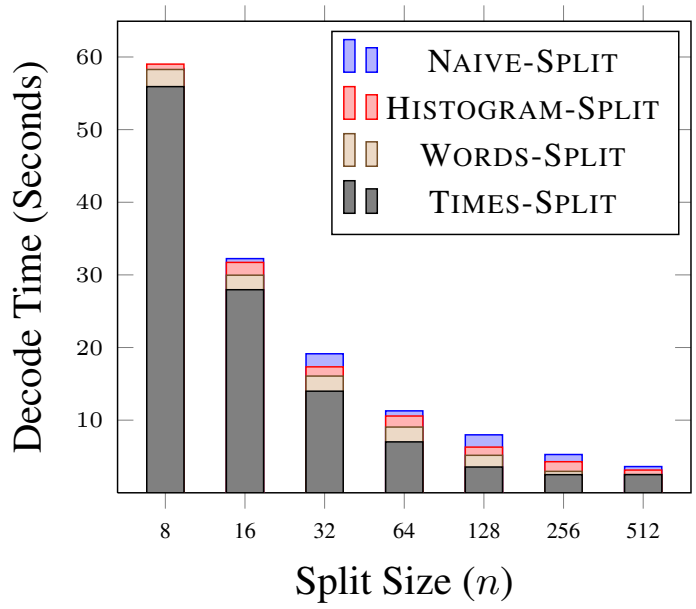
```
function NAIVE-SPLIT(n,input)
    ℓ ← input.length / n
    for p ← 0 ... (n − 1) do
        i ← ℓ × p
        k ←min(i + ℓ − 1,input.length)
        for j ← i ... k do
            output[p].append(input[j])
        end for
    end for
    return output
end function
```

Research into parallel machine scheduling problems constitutes a wide and well-studied field, ranging through various disciplines of engineering, manufacturing, and management in addition to computer science and applied mathematics (Cheng and Sin, 1999), spanning a wide range of scheduling techniques (Panwalkar and Iskander, 1977).

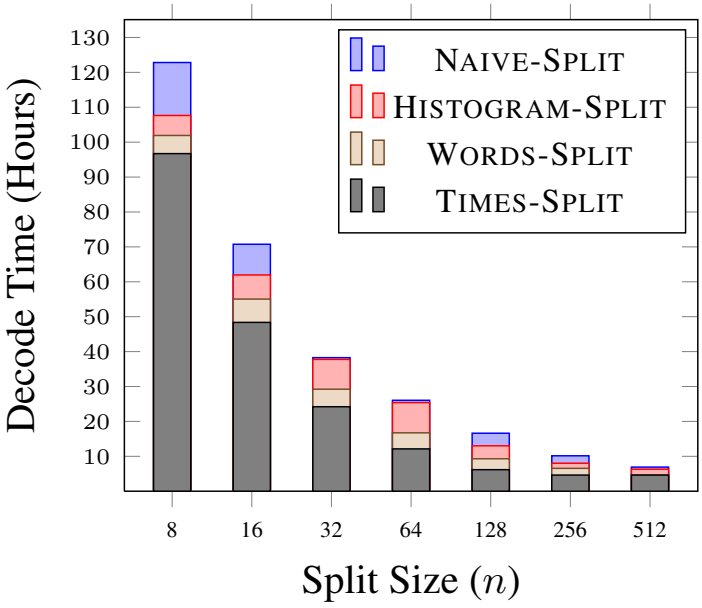
We now briefly examine the existing research most relevant to our task. Hu (1961) and Graham (1966, 1969) develop various list scheduling algorithms. This family of algorithms prioritizes jobs into a queue, then assigns jobs to machines in queue order. This approach attempts to evenly balance the load on each execution host (De and Morton, 1980; Cheng and Sin, 1999). Both Algorithm 5.1 and techniques we develop fall into this family of algorithms.

Better Splitting for Faster Results

To observe the effects of splitting algorithms on decoding speed, we translated Urdu-English data using Moses in a parallel computing cluster, distributing work using the Sun Grid Engine. We ran two decoding setups: a standard configuration using a 5-gram language model, and a much slower configuration that also used an incremental syntactic language model. Using Algorithm 5.1, the runtimes of the slowest of n translation jobs in each configuration is illustrated



(a) Decoding times in **seconds** for standard decoder configured using a 5-gram language model.



(b) Decoding times in **hours** for decoder configured using a syntactic language model in addition to a 5-gram language model.

Figure 5.16: Decoding times for the slowest translation job in a translation task split into n decoding jobs using various splitting algorithms (NAIVE-SPLIT, HISTOGRAM-SPLIT, WORDS-SPLIT, and TIMES-SPLIT).

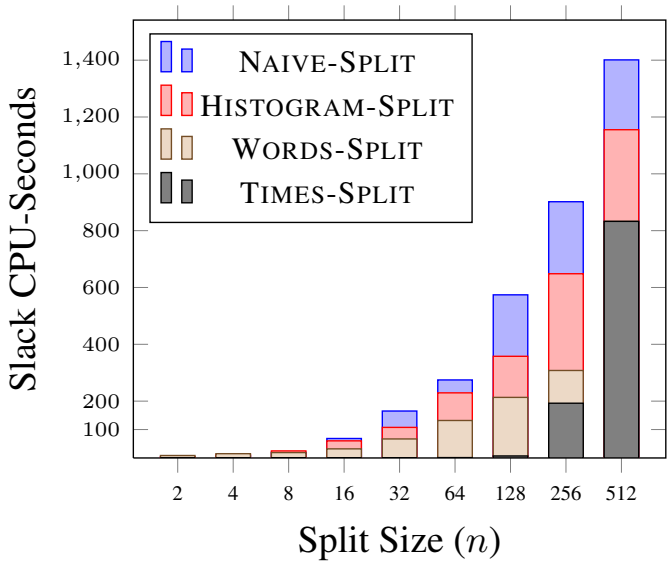
Split Size (n)	NAIVE-SPLIT		HISTOGRAM-SPLIT		WORDS-SPLIT		TIMES-SPLIT	
	Min	Max	Min	Max	Min	Max	Min	Max
2	222.9	224.4	221.5	225.8	219.3	228.0	*223.7*	***223.7***
4	109.2	113.7	110.0	114.6	108.7	115.6	*111.8*	***111.8***
8	51.4	58.4	52.2	59.0	53.2	58.3	*55.9*	***55.9***
16	24.9	32.2	25.0	31.7	25.3	30.0	27.9	**28.0**
32	11.3	19.1	11.9	17.3	11.7	16.1	*14.0*	***14.0***
64	5.4	11.3	5.4	10.6	5.7	9.1	*7.0*	***7.0***
128	1.3	8.0	2.2	6.3	2.3	5.2	*3.5*	***3.5***
256	0.3	5.3	0.7	4.3	0.8	3.0	1.7	**2.5**
512	0.0	3.6	0.2	3.1	0.3	**2.5**	0.6	**2.5**

(a) Decoding times in **seconds** for standard decoder configured using a 5-gram language model.

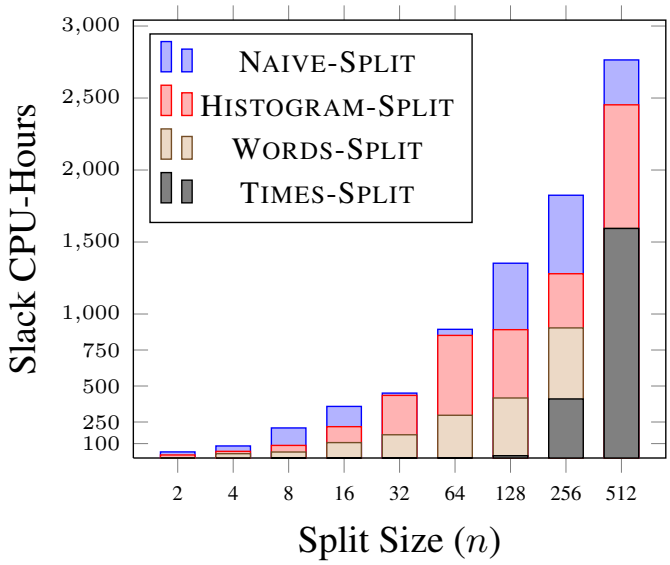
Split Size (n)	NAIVE-SPLIT		HISTOGRAM-SPLIT		WORDS-SPLIT		TIMES-SPLIT	
	Min	Max	Min	Max	Min	Max	Min	Max
2	365.7	408.0	376.3	397.5	386.1	387.6	*386.9*	***386.9***
4	176.1	214.4	186.8	205.0	184.6	200.9	*193.4*	***193.4***
8	84.6	122.8	84.8	107.6	88.4	101.9	*96.7*	***96.7***
16	40.8	70.8	40.5	62.0	45.3	55.0	*48.4*	***48.4***
32	19.2	38.3	18.8	37.8	20.5	29.2	*24.2*	***24.2***
64	9.2	26.1	9.2	25.4	9.4	16.7	*12.1*	***12.1***
128	2.7	16.6	4.1	13.0	3.7	9.3	5.9	**6.2**
256	0.7	10.2	1.3	8.0	1.4	6.6	2.9	**4.6**
512	0.0	6.9	0.3	6.3	0.6	**4.6**	1.1	**4.6**

(b) Decoding times in **hours** for decoder configured using a syntactic language model in addition to a 5-gram language model.

Figure 5.17: Decoding times for the fastest (min) and slowest (max) decoding jobs when a translation task is split into n decoding jobs. *Italics* indicate balanced task times. **Bold** indicates fastest max time at that split.



(a) Slack **CPU-Seconds** for standard decoder configured using a 5-gram language model.



(b) Slack **CPU-Hours** for decoder configured using a syntactic language model in addition to a 5-gram language model.

Figure 5.18: Cumulative slack CPU time for n processing cores when processing a parallel translation task split into n jobs using various splitting algorithms. Slack CPU time is caused when some jobs finish before others. Zero slack time indicates conditions where all jobs complete simultaneously.

in Figure 5.16 for various values of n. Figure 5.17 shows that in all cases, there is a significant difference between the fastest and slowest translation job, leading to computational slack time (Figure 5.18).

In examining these results, we observe that the slack time results primarily from situations where some jobs are assigned a disproportionate number of short sentences, and thus finish much faster than jobs that are assigned many longer sentences. To remedy this imbalance, we propose Algorithm 5.2. This technique examines the word lengths of each sentence prior to splitting the data into jobs. Sentences are sorted according to length, then assigned in turns to jobs. This results in the sentence length histograms for each job being approximately equal.

Algorithm 5.2 Split input text into n parts to balance the histograms of line lengths for all parts.

```
function HISTOGRAM-SPLIT(n,input)
    for i ← 0 ... ( input.length −1) do
        sentence[i].length ← input[i].length
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| x.length ⇔ y.length}        ▷ Sort sentences by length
    p ← 0
    for i ← 0 ... (input.length −1) do
        output[p].append(input[sentence[i].index])
        p ← (p + 1) mod n
    end for
    return output
end function
```

Figures 5.16a–5.18a fail to show improvment in speed for a standard Moses configuration for small values of n. But for values of $n > 8$, and for all values of n using the slow syntactic language model (Figures 5.16b–5.18b), Algorithm 5.2 results in significant decreases in total decoding time over the naive Algorithm 5.1.

While Algorithm 5.2 balances short and long sentences across jobs, we may be able to improve runtimes by balancing the total number of words in each job. In Algorithm 5.3, sentences are sorted by length into a queue, with longest sentences at the head of the queue. Initially, no sentences have been assigned to any job. The longest sentence, at the head of the queue, is

assigned first to a job. As each sentence is assigned to a job, the total number of words assigned to that job is recorded. Each subsequent sentence is removed from the queue and assigned to the job with the least work assigned to it, as measured by number of words. Results for Algorithm 5.3 show substantial speedups over Algorithms 5.1 and 5.2 when using the slower decoder configuration. Similar speedups are seen where $n > 8$ for the faster configuration.

Algorithm 5.3 Split input text into n parts to balance the number of words for all parts.

```
function WORDS-SPLIT(n,input)
    for i ← 0 . . . ( input.length −1) do
        sentence[i].length ← input[i].length
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| y.length ⇔ x.length}
                                    ▷ Sort sentences by length, in reverse order
    for i ← 0 . . . ( input.length −1) do
        p ← LEAST(words)                    ▷ Find partition with fewest words
        output[p].append(input[sentence[i].index])
        words[p] ←words[p] + sentence[i].length
    end for
    return output
end function
```

When assigning sentences to jobs, we would ideally like to know how long each sentence will take to process. Algorithms 5.2 and 5.3 use the number of words in each sentence as a proxy for processing time. During MERT, the same set of development sentences are translated multiple times. Since each decoding process differs only by the $\boldsymbol{\lambda}$ weights used, it is reasonable to expect similar runtimes for each run. With this in mind, we record the time required to translate each sentence during the first iteration of MERT. In subsequent iterations, Algorithm 5.4 uses the time recorded to translate a sentence as an estimate of the time it will take to translate that sentence again. Algorithm 5.4 differs from Algorithm 5.3 by sorting using these times instead of sentence length.

The use of Algorithm 5.4 results in speedups under all conditions (Figure 5.17). In nearly all conditions, the speedup is substantial over the baseline. Slack time is at or near zero in

Algorithm 5.4 Split input text into n parts to balance the estimated translation time of all parts.

```
function TIMES-SPLIT(n,input,estimate)
    for i ← 0 ... ( input.length −1) do
        sentence[i].time ← estimate[i]
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| y.time ⇔ x.time}
                                    ▷ Sort sentences by time, in reverse order
    for i ← 0 ... ( input.length −1) do
        p ← LEAST(times)                    ▷ Find partition with least time
        output[p].append(input[sentence[i].index])
        times[p] ← times[p] + sentence[i].time
    end for
    return output
end function
```

all cases where $n < 256$. Where slack time remains, $n = (256, 512)$, a few jobs were each assigned a single long sentence that took longer to translate than the other jobs which were assigned multiple shorter sentences.

5.4.3 Translation Results

We trained a phrase-based translation model on the full NIST Open MT08 Urdu-English translation model using the full training data. We trained the HHMM and n-gram LMs on the WSJ data in order to make them as similar as possible. To tune the log-linear feature parameter weights, we used MERT — Minimum Error Rate Training (Och, 2003). During tuning, Moses was first configured to use just the n-gram LM, then configured to use both the n-gram LM and the syntactic HHMM LM. MERT consistently assigned positive weight to the syntactic LM feature, typically slightly less than the n-gram LM weight. This suggests that the MERT optimization procedure found our syntactic LM to be a useful feature.

In our integration with Moses, incorporating a syntactic language model dramatically slows the decoding process. Figure 5.15 illustrates a slowdown around three orders of magnitude. Although speed remains roughly linear to the size of the source sentence (ruling out exponential

Moses LM(s)	BLEU
n-gram only	18.78
HHMM + n-gram	**19.78**

Figure 5.19: Results for Ur-En devtest (only sentences with 1-20 words) with HHMM beam size of 2000 and Moses settings of distortion limit 10, stack size 200, and ttable_limit 20. The n-gram only condition used a dev set constrained to sentences with 1-20 words. The HHMM + n-gram condition used a dev set constrained to sentences with 1-40 words.[3]

behavior), it is with an extremely large constant time factor. Due to this slowdown, we tuned the parameters using a constrained dev set, and tested using a constrained devtest set (only sentences with 1-20 words). The n-gram only condition used a dev set constrained to sentences with 1-20 words. The HHMM + n-gram condition used a dev set constrained to sentences with 1-40 words.[3]

We test our tuned translation systems on a test set of sentences not seen during MERT. Our test set is the devtest section, set aside in Baker et al. (2009), of the NIST Open MT 2008 Urdu-English task. In our initial experiment, we restrict our test set to only those sentences 1–20 words in length. For translating the test set, Moses was first configured to use just the n-gram LM, then configured to use both the n-gram LM and the syntactic HHMM LM. Translation results, as measured by BLEU, are shown in Figure 5.19.

Figure 5.19 shows a statistically significant improvement to the BLEU score when using the HHMM and the n-gram LMs together on this reduced test set. Our perplexity results indicated that the HHMM language model is a good model of the target language. This positive translation result provides some evidence indicating that the use of our incremental syntactic language model is indeed serving to guide the translation algorithm towards more fluent target language translations.

We next examine whether these results hold when we translate a larger section of our test set, including sentences 1–40 words in length. We re-run MERT, this time constraining the dev set for both conditions to sentences with 1–20 words. Again during tuning, Moses was first configured to use just the n-gram LM, then configured to use both the n-gram LM and

[3] Schwartz et al. (2011) incorrectly reported that both conditions used a dev set constrained to sentences with 1-20 words.

Moses LM(s)	BLEU
n-gram only	21.43
HHMM + n-gram	21.72

Figure 5.20: Results for Ur-En devtest (only sentences with 1-40 words) with HHMM beam size of 2000 and Moses settings of distortion limit 10, stack size 200, and ttable_limit 20. Both conditions used a dev set constrained to sentences with 1-20 words.

Moses LM(s)	BLEU	
	distortion limit=10	distortion limit=20
n-gram only	21.67	21.88
HHMM + n-gram	21.44	21.93

Figure 5.21: Results for Ur-En devtest (only sentences with 1-40 words) with HHMM beam size of 2000 and Moses settings of distortion limit 10 and 20, stack size 200, and ttable_limit 20. Both conditions used a dev set constrained to sentences with 1-40 words.

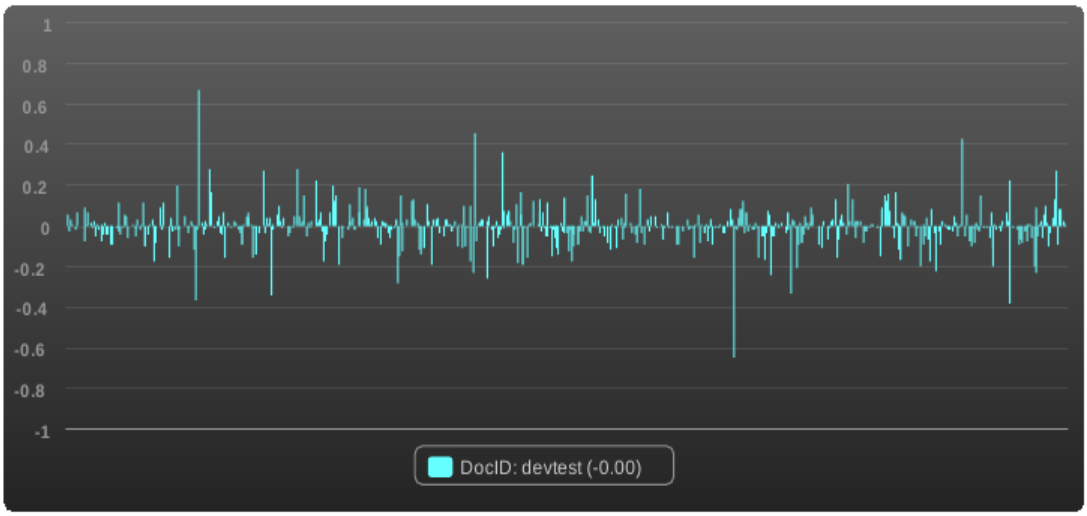


Figure 5.22: Graphical representation of differences in BLEU scores at the segment-level BLEU scores. Graph was produced by the iBLEU visualization tool (Madnani, 2011).

System	BLEU
NIST OpenMT System 2	12.13
Systran version 5.05	14.81
NIST OpenMT System 3	21.24
Google	21.36
NIST OpenMT System 1	23.77
Moses, n-gram LM only	24.13
Moses, HHMM + n-gram LMs	24.18
NIST OpenMT System 7	24.86
NIST OpenMT System 6	24.97
NIST OpenMT System 5	25.17
NIST OpenMT System 8	25.92
NIST OpenMT System 4	26.97
Commercially available system with syntactic TM, circa 2010	27.48
SCALE	32.24
NIST OpenMT System 9	32.88

Figure 5.23: Translation results for NIST 2009 Open Machine Translation Urdu-English comparison test set of 1220 sentences. The Moses runs used a distortion limit of 20, and an HHMM beam size of 2000.

System	BLEU
Systran version 5.05	14.69
Google	22.05
Moses, n-gram LM only	23.91
Moses, HHMM + n-gram LMs	24.12
Commercially available system with syntactic TM, circa 2010	27.18
SCALE	32.46
Top System, NIST OpenMT 2009	33.10

Figure 5.24: Translation results for NIST 2009 Open Machine Translation Urdu-English full test set of 1792 sentences. The Moses runs used a distortion limit of 20, and an HHMM beam size of 2000.

the syntactic HHMM LM. Translation results for this experimental condition, as measured by BLEU, are shown in Figure 5.20. Under this condition, we find no significant difference in translation quality as measured by BLEU between the translation system using the syntactic LM and the n-gram only system.

For the experimental results shown in Figure 5.20, tuning was performed on a dev set constrained to 1–20 words, and testing was performed on a test set constrained to 1–40 words. We next examine a condition where both dev set and test set were constrained to sentences 1–40 words in length. For this experiment, tuning was again run with Moses was first configured to use just the n-gram LM, then configured to use both the n-gram LM and the syntactic HHMM LM. Translation results for this experimental condition, as measured by BLEU, are shown in Figure 5.21. Under this condition, we find no significant difference in translation quality as measured by BLEU between the translation system using the syntactic LM and the n-gram only system.

Using the iBLEU visualization tool (Madnani, 2011), it is possible to examine BLEU score results at a finer, sentence level of granularity. Figure 5.22 graphically depicts the difference in component BLEU scores for each sentence in a test set. The translation results shown in Figure 5.22 are for the condition where dev set and test set were constrained to sentences with 1–40 words, with a distortion limit of 20, as shown in Figure 5.21. In this visualization, vertical

lines above the center indicate sentences where the n-gram only condition outperforms the syntactic LM condition, according to BLEU score components. Likewise, vertical lines below the center indicate sentences where the syntactic LM condition outperforms the n-gram only condition, according to BLEU score components. The magnitude of each vertical line depicts the relative difference in BLEU score components between the two conditions. Figure 5.22 shows an approximately equal number of vertical lines above and below the center; this result confirms the results shown in Figure 5.20 and Figure 5.21 — namely that BLEU fails to show a significant difference between the two conditions.

In the statistical machine translation literature, it is standard practice to report changes in BLEU score as an indicator of whether a new experimental condition produces significantly better results over a baseline. According to that measure, the experimental results reported in Figure 5.20 and Figure 5.21 fail to show a significant difference between phrase-based machine translation results produced with a syntactic LM and those produced with only an n-gram LM. However, the fact that BLEU scores between two compared conditions are not significantly different does not necessarily mean that the translation quality of the two systems are the same; Figure 5.25 illustrates this fact with translations of two sentences selected from our test set. BLEU is an n-gram precision metric, and therefore tends to prefer translations produced by systems which emphasize high-order n-gram matches. In particular, BLEU has been shown to be a poor indicator of relative translation quality when comparing translation systems that utilize substantially different methods (Callison-Burch et al., 2007), such as rule-based translation systems and statistical phrase-based systems. Even worse, BLEU scores have been shown to correlate poorly with human assessments of translation quality (Callison-Burch et al., 2006). In recent years, the shared translation task at the annual Workshop on Statistical Machine Translation has adopted human assessment as the primary metric of translation quality, in preference over BLEU and other automatic evaluation metrics (Callison-Burch et al., 2007, 2008, 2009, 2010, 2011).

In the translation experiments above, tuning via MERT was performed only once for each experimental condition. This was due primarily to time constraints; running MERT is a time-intensive procedure, since each run of MERT involves translating a dev set multiple times. When decoding is slowed by the use of the syntactic LM, tuning time is obviously affected. This methodology, running MERT once for each experimental condition, is common practice in the statistical machine translation literature. However, it is well known in the statistical machine

ID	Segment 103, Document "devtest" [Δ_{BLEU}=0.68]
Source	حکومت کے وعدے ? ? ?
Reference (reference0)	the promises of the government ? ? ?
Reference (reference1)	government 's promises ? ? ?
Reference (reference2)	the promises of the govt . ? ? ?
Reference (reference3)	government promises ? ? ?
Hypothesis (ngram)	the government ? **? ?** [1.00]
Hypothesis (hhmm)	the government ? **promise . .** [0.32]

ID	Segment 512, Document "devtest" [Δ_{BLEU}=-0.65]
Source	لاجواب لکھہ ہے .
Reference (reference0)	written wonderfully well .
Reference (reference1)	you have written a matchless piece !
Reference (reference2)	undoubtedly , it was written very well .
Reference (reference3)	splendid writing
Hypothesis (ngram)	**how is written** . [0.19]
Hypothesis (hhmm)	**matchless** . [0.84]

Figure 5.25: Translation of segments 103 & 512. BLEU is not always a good judge of translation quality. In segment 103, the translation from the n-gram only system has a higher trigram match, and hence a higher BLEU score, but the translation from the HHMM syntactic LM system translates more content words. In segment 512, the translations from both systems are very bad, but the HHMM syntactic LM system receives a much higher score according to BLEU.

translation community that there is substantial instability in the MERT process. Recent work has shown that this instability leads not just to a difficulty in exactly replicating prior results, but can easily lead to substantial random variation in translation quality; Clark et al. (2011) examines this issue and concludes that experiments should strive to run multiple replications of tuning to control for optimizer instability. In future work we plan to re-run our above experiments using multiple optimization runs.

Our initial translation results, shown in Figure 5.19, present a statistically significant improvement in translation quality, as measured by BLEU, when our phrase-based translation utilizes our syntactic language model. Further results, shown in Figure 5.20 and Figure 5.21 fail to show a significant difference between phrase-based machine translation results produced with a syntactic LM and those produced with only an n-gram LM. All of these experimental conditions were each performed with only one run of MERT, and it is therefore even more important to be careful in drawing conclusions from the data.

Ultimately, the most important metric of translation quality is human judgement. To better examine whether our syntactic language model is impacting translation quality, we perform

ID	Segment 561, Document "devtest" [Δ_{BLEU}=-0.21]
Source	ہر انسان کو معاشرے میں اپنی ذمے داری سمجھنا چاہئے
Reference (reference0)	everyone must recognize his responsibility in the society
Reference (reference1)	every person should realize ones responsibility in the society .
Reference (reference2)	everyone in society should do his duty .
Reference (reference3)	every man should understand his responsibilities to society .
Hypothesis (ngram)	**the society should understand their in** every human being claimed responsibility [0.13]
Hypothesis (hhmm)	every human being claimed responsibility **in the society should understand** [0.34]

Figure 5.26: Translation of segment 561. The translation from the HHMM syntactic LM system is more syntactically well-formed.

ID	Segment 624, Document "devtest" [Δ_{BLEU}=-0.15]
Source	فیصلہ کا تاریخ گا لکھے وقت ' پر ' .
Reference (reference0)	but ' time will recount the judgment of history ' .
Reference (reference1)	but ' time will write the judgment of history ' .
Reference (reference2)	but time will decide what history will write in the end .
Reference (reference3)	but time will write the decision of the history .
Hypothesis (ngram)	the decision of history **written on ' time will '** . [0.29]
Hypothesis (hhmm)	**' time will write on** the decision of history . [0.44]

ID	Segment 744, Document "devtest" [Δ_{BLEU}=-0.23]
Source	ملاقات میں حرج نہیں .
Reference (reference0)	there is nothing wrong in meeting .
Reference (reference1)	there is no problem in meeting .
Reference (reference2)	there is no harm in meeting with him .
Reference (reference3)	there are no problems with this meeting .
Hypothesis (ngram)	in the meeting **, is not** . [0.09]
Hypothesis (hhmm)	**no harm** in the meeting . [0.32]

Figure 5.27: Translation of segments 624 & 744. The translations from the HHMM syntactic LM system are more syntactically well-formed.

an informal examination of our translation results. The example translations we highlight are taken from the experimental condition shown in Figure 5.21, where dev set and test set were constrained to sentences with 1–40 words, with a distortion limit of 20.

While translation results are nowhere near perfect (recall the bad translations shown in Figure 5.25), a manual examination of the results shows that in many cases, the translations produced using the syntactic language model do appear to be more syntactically well-formed than those produced using only an n-gram language model. Figures 5.26–5.31 show a selection of sentences translated under each experimental condition. These example sentences were selected using the iBLEU visualization tool (Madnani, 2011); the visualization of these sentences shown in Figures 5.26–5.31, along with with their machine translation results and reference translations, are taken from the relevant iBLEU visualizations.

In Figure 5.26, a sample sentence (segment 561) from the test set is shown. Along with the sentence, there are four human reference translations of the original Urdu sentence. Finally,

ID	Segment 158, Document "devtest" [Δ_{BLEU}=-0.34]
Source	, موجودہ چیف جسٹس کے خلاف دوبارہ ریفرنس دائر نہیں کیا جاسکتا سعیدالزماں صدیقی
Reference (reference0)	reference can not be filed again against the present chief justice , saeed uz zaman siddiqui
Reference (reference1)	another reference can not be filed against present chief justice : saeeduz zaman siddiqi
Reference (reference2)	saiduz zaman siddiqi : a second reference can not be filed against the present chief justice
Reference (reference3)	saiduz zaman siddiqi : a second reference can not be filed against the present chief justice
Hypothesis (ngram)	the chief justice **of سعیدالزماں siddiqui** can not **file a** reference again . [0.10]
Hypothesis (hhmm)	the chief justice can not **be filed against the** reference again , **سعیدالزماں siddiqui** [0.44]

Figure 5.28: Translation of segment 158. The translation from the HHMM syntactic LM system is more syntactically well-formed and represents an slightly better translation.

ID	Segment 100, Document "devtest" [Δ_{BLEU}=-0.36]
Source	انہوں نے بتایا کہ نیٹو نے پاکستان سے سیاسی مذاکرات شروع کر دیئے ہیں اور وہ درست سمت میں آگے بڑھ رہے ہیں .
Reference (reference0)	he said that nato had started political negotiations with pakistan and that they were heading in the right direction .
Reference (reference1)	he said that nato has started political negotiations with pakistan and they are moving in the right direction .
Reference (reference2)	she said that nato has begun political negotiations with pakistan and that they are headed in a positive direction .
Reference (reference3)	she said that nato has begun political negotiations with pakistan and that they are headed in a positive direction .
Hypothesis (ngram)	he said that nato and they are moving forward in the right direction **and political talks started from pakistan** . [0.39]
Hypothesis (hhmm)	he said that nato **started political negotiations with pakistan** and they are moving forward in the right direction . [0.75]

ID	Segment 144, Document "devtest" [Δ_{BLEU}=-0.16]
Source	لال بازار سے لال مسجد تک
Reference (reference0)	from red light area to lal masjid
Reference (reference1)	from lal bazar to lal masjid
Reference (reference2)	from lal bazaar to lal masjid
Reference (reference3)	from lal masjid to lal bazar .
Hypothesis (ngram)	lal market lal masjid **by** [0.19]
Hypothesis (hhmm)	lal market **to** lal masjid [0.35]

Figure 5.29: Translation of segments 100 & 144. The translations from the HHMM syntactic LM system are more syntactically well-formed and represent overall better translations.

ID	Segment 210, Document "devtest" [Δ_{BLEU}=-0.19]
Source	جس سے جمہوری عمل میں عوام کے عمل دخل میں مزید اضافہ ہوا .
Reference (reference0)	which enhanced public participation in the democratic process .
Reference (reference1)	this will increase public participation in the democratic process .
Reference (reference2)	which increased the part the public takes in democratic action .
Reference (reference3)	which increased the part the public takes in democratic action .
Hypothesis (ngram)	**further increase in** which played in **democratic** process of the **people** . [0.11]
Hypothesis (hhmm)	which played **an increase** in **the** process of **people in** the **democratic process** . [0.30]

ID	Segment 481, Document "devtest" [Δ_{BLEU}=-0.16]
Source	اس کے علاوہ تین انتہا پسند کولکتہ سے گرفتار کرکے لکھنؤ لائے گئے ہیں .
Reference (reference0)	moreover , 3 terrorists have been arrested and brought to lucknow from calcutta .
Reference (reference1)	besides , three extremists were arrested from kolkata and brought to lucknow .
Reference (reference2)	additionally , three extremists have been arrested in kolkata and brought to lucknow .
Reference (reference3)	additionally , three extremists have been arrested in kolkata and brought to lucknow .
Hypothesis (ngram)	apart from this , three extremists have been **brought lucknow** arrested **the** calcutta . [0.34]
Hypothesis (hhmm)	apart from this , three extremists have been arrested **from** calcutta **to lucknow** . [0.50]

Figure 5.30: Translation of segments 210 & 481. The translations from the HHMM syntactic LM system are more syntactically well-formed and represent overall better translations.

machine translations of the sentence using the HHMM syntactic language model condition and n-gram only condition are shown. In the translation from the n-gram only condition, we observe that sequences of words are reasonably fluent within windows of three or four words, but that the sentence is far from globally coherent. This effect is expected; the n-gram language model works to ensure fluency within an n-word window, but lacks any mechanism to influence fluency outside that window.

While the semantic adequacy of the syntactic LM translation in Figure 5.26 may not necessarily be an improvement over the n-gram only translation, the syntactic LM has the ability to influence fluency over the entire sentence. In this case we do indeed observe that the syntactic structure of this alternate translation appears to be more fluent. We observe similar results in test set segments 624 and 744, shown in Figure 5.27. In each of these examples, the semantic adequacy of the syntactic LM translations is not necessarily superior to that of the n-gram only translations, but the syntactic LM translations are more fluent and more syntactically well-formed.

ID	Segment 323, Document "devtest" [Δ_{BLEU}=-0.26]
Source	ہم اس فیصلہ پر مبارکباد پیش کرتے ہیں .
Reference (reference0)	we congratulate on this decision .
Reference (reference1)	we congratulate on this judgment .
Reference (reference2)	we congratulate the court on this decision .
Reference (reference3)	we congratulate them on the ruling .
Hypothesis (ngram)	we **are** on this decision **for** . [0.28]
Hypothesis (hhmm)	we **congratulations** on this decision . [0.54]

Figure 5.31: Translation of segment 323. The translation from the HHMM syntactic LM system is more syntactically well-formed and represents an overall better translation. Note the trailing preposition in the translation from the n-gram only system.

We observe a number of examples in the test set where the use of our syntactic language model results in translations that are both more syntactically well-formed and also are clearly more semantically accurate with respect to the reference translations. Test set segment 158, shown in Figure 5.28, represents an example where the use of our syntactic LM results in a translation that is more syntactically well-formed; in this example, the resulting translation also comes somewhat closer to the semantics of the reference translations than does the n-gram only translation. We observe this effect even more clearly in test set segments 100 and 144, shown in Figure 5.29. In these examples, the syntactic LM translations are clearly more semantically accurate with respect to the reference translations, and are also clearly more syntactically well-formed. This is easy to see in segment 144; here the n-gram only model incorrectly places the incorrect preposition at the end of the translation, while the syntactic LM model places the correct preposition at the correct location in the middle of the translation. We observe similar effects in test set segments 210 and 481, shown in Figure 5.30, and in test set segment 323, shown in Figure 5.31.

5.5 Conclusion

In this chapter, we have argued that incremental parsers, used as syntactic language models, provide an appropriate algorithmic match to incremental phrase-based machine translation. We integrated an incremental syntactic language model directly into the Moses phrase-based translation decoder. The incremental syntactic language model we use is defined by the HHMM parsing algorithms described in Chapter 4, using the probabilistic phrase structure grammar in right-corner form trained on the transformed treebank from Chapter 3. The method we describe is both novel and general; in principle this method can be used to incorporate any generative incremental language model into phrase-based machine translation. Our method re-exerts the role of the language model as a mechanism for encouraging syntactically fluent translations.

The incremental syntactic language model we present is a good model of the English language. A standard measure of language model performance is the perplexity metric, which measures how surprised a model is by new data. We shown, in Section 5.4.1, that our syntactic language model substantially outperfoms traditional n-gram language models in terms of perplexity on out-of-domain data. When our syntactic language model is interpolated with an

n-gram language model, we find even better perplexity results, on both in-domain and out-of-domain data.

Integration of our syntactic language model into phrase-based translation comes with a cost to translation speed. In Section 5.4.2 we examine this issue and develop mechanisms for alleviating the problem by distributing translation jobs across a cluster of computational nodes.

Ultimately, we hypothesize that our syntactic language model should influence the phrase-based translation decoder toward translations which are more syntactically well-formed. We test this hypothesis under several experimental conditions. We find no significant difference in translation quality as measured by BLEU between the translation system using the syntactic LM and the n-gram only system. We conclude our experimentation with an informal manual examination of a number of sentences from our test set translations. Our manual examination suggests that the use of our incremental syntactic language model is indeed serving to guide the translation algorithm towards more fluent target language translations.

Chapter 6

Conclusion

Modern machine translation techniques typically incorporate both a *translation model*, which guides how individual words and phrases can be translated, and a *language model* (LM), which promotes fluency as translated words and phrases are combined into a translated sentence. Most attempts to inform the translation process with linguistic knowledge have focused on infusing syntax into translation models. This dissertation presents a novel technique for incorporating syntactic knowledge as a language model in the context of statistical phrase-based machine translation (Koehn et al., 2003), one of the most widely used modern translation paradigms.

This dissertation argues that incremental syntactic language models are a straightforward and appropriate algorithmic fit for incorporating syntax into phrase-based statistical machine translation, since both process sentences in an incremental left-to-right fashion. This means incremental syntactic LM scores can be calculated during the decoding process, rather than waiting until a complete sentence is posited, which is typically necessary in top-down or bottom-up parsing.

In summary, the major contributions of this work are as follows:

- We present a formal definition of an incremental syntactic language model as a Hierarchical Hidden Markov Model (HHMM), and detail for the first time exactly how this model is estimated from a treebank corpus of labelled data (Chapter 3).

- The HHMM syntactic language model has been used in prior work involving parsing, speech recognition, and semantic role labelling. For the first time, we fully document the complete algorithmic definition of the HHMM as a language model (Chapter 4).

- We develop a novel and general method for incorporating any generative incremental language model into phrase-based machine translation (Chapter 5). We integrate our HHMM incremental syntactic language model into Moses, the prevailing phrase-based decoder.

- We present empirical results for language model perplexity that show our incremental syntactic language model, implemented as an HHMM, is a good model of language. We present empirical results on a constrained Urdu-English translation task that demonstrate the use of our syntactic LM. We present three novel techniques for effectively speeding up translation through the use of parallel computational resources.

6.1 Experimental Results

A standard measure of language model quality is average per-word perplexity. We present empirical results evaluating perplexity of various n-gram language models and our syntactic language model on both in-domain and out-of-domain test sets. On an in-domain test set, a traditional 5-gram language model trained on the same data as our syntactic language model outperforms the syntactic language model in terms of perplexity. We find that interpolating the 5-gram LM with the syntactic LM results in improved perplexity results, a 10% absolute reduction in perplexity compared to the 5-gram LM alone.

On an out-of-domain test set, we find that our syntactic LM substantially outperforms all other LMs trained on the same training data. The syntactic LM demonstrates a 58% absolute reduction in perplexity over a 5-gram language model trained on the same training data. On this same out-of-domain test set, we further show that interpolating our syntactic language model with a large Gigaword-scale 5-gram language model results in the best overall perplexity results — a 61% absolute reduction in perplexity compared to the Gigaword-scale 5-gram language model alone, a 76% absolute reduction in perplexity compared to the syntactic LM alone, and a 90% absolute reduction in perplexity compared to the original smaller 5-gram language model.

A language model with low perplexity is a theoretically good model of the language; it is expected that using an LM with low perplexity as a component of a machine translation system should result in more fluent translations. We present empirical results on a constrained Urdu-English translation task that demonstrate the use of our syntactic LM, and we perform an informal manual evaluation of translation results which suggests that the use of our incremental

syntactic language model is indeed serving to guide the translation algorithm towards more fluent target language translations.

Integration of our syntactic language model into phrase-based translation comes at a substantial cost to translation speed. We present three novel techniques for effectively speeding up translation through the use of parallel computational resources. We show that use of these techniques results in a substantial reduction in slack time, where parallel computational resources are available but not used. By using the best of these techniques, we found a complete elimination of slack time in most cases.

6.2 Future Work

We provided a rigorous formal definition of incremental syntactic languages models, and detailed what steps are necessary to incorporate such LMs into phrase-based decoding. The perplexity improvements suggest that interpolating between n-gram and syntactic LMs may hold promise on larger data sets. In future work, we will explore whether the positive results seen on this constrained task hold when translating longer sentences and other data sets.

In our translation experiments, we chose to use an n-gram model trained only on data from the Wall Street Journal treebank corpus. This is the same corpus on which our syntactic language model was trained. This choice allowed us to directly compare translations performed with and without the syntactic language model, without having the complicating factor of different respective training corpora for the n-gram and syntactic language models. That said, we acknowledge that the n-gram language model trained only on WSJ data is a small language model. The use of very large n-gram language models is typically a key ingredient in the best-performing machine translation systems (Brants et al., 2007). Our future work seeks to incorporate large-scale n-gram language models in conjunction with incremental syntactic language models.

The added decoding time cost of our syntactic language model is very high. It could be argued that a phrase-based decoder without a syntactic language model that explored a much larger search space (and consequently also took much more time to run) might find and select translations of equal or greater quality than those selected when the syntactic language model is used. By increasing the beam size and distortion limit of the baseline system (where no syntactic language model is used), future work may examine whether a baseline system with

comparable runtimes can achieve comparable translation quality.

A more efficient implementation of the HHMM parser would speed decoding and make more extensive and conclusive translation experiments possible. Various additional improvements could include caching the HHMM LM calculations, and exploiting properties of the right-corner transform that limit the number of decisions between successive time steps.

In this dissertation, we integrated our incremental syntactic language model into Moses, the leading phrase-based translation system. However, Moses is not the only machine translation system in which target language words are generated incrementally, from left to right. The methods we developed here could be directly applied to any system in which target language words are generated incrementally, from left to right. In future work, we may integrate our incremental syntactic language model into Cunei (Phillips and Brown, 2009), a phrase-based decoder which draws heavily on ideas from the example-based machine translation literature. Our method is applicable to other, more divergent MT algorithms as well; Huang and Mi (2010) present an incremental tree-to-string translation algorithm which could potentially benefit from the additional target language syntactic information provided by our incremental syntactic language model.

References

Anne Abeillé, Yves Schabes, and Aravind K. Joshi. Using lexicalized tree adjoining grammars for machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*, Helsinki, Finland, August 1990.

Anthony E. Ades and Mark Steedman. On the order of words. *Linguistics and Philosophy*, 4: 517–558, 1982.

Alfred V. Aho and Jeffery D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of computer and system sciences*, 3:37–56, 1969.

Automatic Language Processing Advisory Committee ALPAC. *Language and Machines: Computers in Translation and Linguistics*. National Research Council, 1966.

Hiyan Alshawi. Head automata and bilingual tiling: Translation with minimal representations (invited talk). In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 167–176, Santa Cruz, California, USA, June 1996a. URL `http://www.aclweb.org/anthology/P96-1023`.

Hiyan Alshawi. Head automata for speech translation. In *Proceedings of the International Conference on Spoken Language Processing*, Philadelphia, Pennsylvania, 1996b.

Hiyan Alshawi, Adam Buchsbaum, and Fei Xia. A comparison of head transducers and transfer for a limited domain translation application. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 360–365, Madrid, Spain, July 1997. URL `http://www.aclweb.org/anthology/P97-1046`.

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Automatic acquisition of hierarchical transduction models for machine translation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 41–47, Montreal, Quebec, Canada, August 1998. URL `http://www.aclweb.org/anthology/P98-1006`.

Hiyan Alshawi, Shona Douglas, and Srinivas Bangalore. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1):45–60, March 2000.

Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190, 1983.

James K. Baker. Stochastic modeling for automatic speech understanding. In R.A. Reddy, editor, *Speech Recognition*, pages 521–541. Academic Press, New York, 1979.

Kathy Baker, Steven Bethard, Michael Bloodgood, Ralf Brown, Chris Callison-Burch, Glen Coppersmith, Bonnie Dorr, Wes Filardo, Kendall Giles, Anni Irvine, Mike Kayser, Lori Levin, Justin Martineau, Jim Mayfield, Scott Miller, Aaron Phillips, Andrew Philpot, Christine Piatko, Lane Schwartz, and David Zajic. Semantically informed machine translation (SIMT). SCALE summer workshop final report, Human Language Technology Center Of Excellence, 2009.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proc. ACL*, 2005. URL `http://www.aclweb.org/anthology-new/W/W05/W05-0909.pdf`.

Srinivas Bangalore. A lightweight dependency analyzer for partial parsing. *Natural Language Engineering*, 6, June 2000.

Srinivas Bangalore and Arivand Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25:237–265, June 1999.

Yehoshua Bar-Hillel. The present status of automatic translation of languages. *Advances in Computers*, 1:91–163, 1960.

Thomas Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Adam Berger, Vincent Della Pietra, and Stephen Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–72, March 1996. URL `http://www.aclweb.org/anthology-new/J/J96/J96-1002.pdf`.

Sylvie Billot and Bernard Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL '89)*, pages 143–151, 1989.

Alexandra Birch, Miles Osborne, and Philipp Koehn. CCG supertags in factored statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 9–16, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W07/W07-0702`.

Rens Bod. Unsupervised syntax-based machine translation: The contribution of discontiguous phrases. In *Proceedings of MT Summit 2007*, Copenhagen, 2007.

Ondřej Bojar and Jan Hajič. Phrase-based and deep syntactic English-to-Czech statistical machine translation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 143–146, Columbus, Ohio, USA, June 2008.

Taylor L. Booth. Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pages 74–81, Waterloo, Ontario, Canada, October 1969.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

Peter Brown, John Cocke, Stephen Della Pietra, Vincent Della Pietra, Frederick Jelinek, Robert Mercer, and Paul Roossin. A statistical approach to language translation. In *Proc. of COLING*, pages 71–76, 1988.

Peter Brown, John Cocke, Stephen Della Pietra, Vincent Della Pietra, Frederick Jelinek, John Lafferty, Robert Mercer, and Paul Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16:79–85, 1990. URL `http://www.aclweb.org/anthology-new/J/J90/J90-2002.pdf`.

Peter Brown, Vincent Della Pietra, Stephen Della Pietra, and Robert Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19:263–311, 1993. URL `http://www.aclweb.org/anthology-new/J/J93/J93-2003.pdf`.

Sarah Brown-Schmidt, Ellen Campana, and Michael K. Tanenhaus. Reference resolution in the wild: Online circumscription of referential domains in a natural interactive problem-solving task. In *Proceedings of the 24th Annual Meeting of the Cognitive Science Society*, pages 148–153, Fairfax, VA, August 2002.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the role of Bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, Trento, Italy, 2006.

Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. (Meta-)Evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 136–158, Prague, Czech Republic, June 2007.

Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further meta-evaluation of machine translation. In *Proceedings of the Third Workshop on Statistical Machine Translation (WMT08)*, Colmbus, Ohio, 2008.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation (WMT09)*, March 2009. URL `http://www.aclweb.org/anthology/W/W09/W09-0x01`.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Kay Peterson, Mark Przybocki, and Omar Zaidan. Findings of the 2010 joint workshop on statistical machine translation and matrics for machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MatricsMATR*, pages 17–53, Uppsala, Sweden, July 2010.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, Scotland, July 2011.

Silvio Ceccato. La grammatica insegnata alle machine. *Civiltà delle Machine*, Nos. 1 and 2, 1956.

Wen-Han Chao and Zhou-Jun Li. Incorporating constituent structure constraint into discriminative word alignment. In *Proceedings of MT Summit XI*, pages 97–103, Copenhagen, Denmark, 2007.

J.-C. Chappelier and M. Rajman. A generalized CYK algorithm for parsing stochastic CFG. In *Proceedings of the First Workshop on Tabulation in Parsing and Deduction*, pages 133–137, Paris, France, 1998.

Eugene Charniak. A maximum-entropy inspired parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (ANLP-NAACL'00)*, pages 132–139, Seattle, Washington, 2000.

Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In *In MT Summit IX. Intl. Assoc. for Machine Translation*, 2003.

Ciprian Chelba and Frederick Jelinek. Exploiting syntactic structure for language modeling. In *Proc. COLING/ACL*, pages 225–231, Montreal, Canada, 1998.

Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech and Language*, 14:283–332, 2000.

Ciprian Chelba, David Engle, Frederick Jelinek, Victor M. Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, and Dekai Wu. Structure and performance of a dependency language model. In *Proc. Eurospeech*, pages 2775–2778, Rhodes, Greece, 1997.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical report, Harvard University, 1998.

T.C.E. Cheng and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1999.

Colin Cherry. Cohesive phrase-based decoding for statistical machine translation. In *Proceedings of ACL-08: HLT*, pages 72–80, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P08/P08-1009.

Colin Cherry and Dekang Lin. Inversion transduction grammar for joint phrasal translation modeling. In *Proceedings of SSST, NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, pages 17–24, Rochester, New York, April 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W07/W07-0403.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proc. ACL*, pages 263–270, 2005.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

David Chiang. Learning to translate with source and target syntax. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1443–1452, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P10-1146.

David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 224–233, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1024.

David Chiang, Kevin Knight, and Wei Wang. 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N09/N09-1025.

Noam Chomsky. Formal properties of grammars. In R. Luce, R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. John Wiley, 1963.

Noam Chomsky. Remarks on nominalization. In Roderick Jacobs and Peter Rosenbaum, editors, *Readings in English Transformational Grammar*, pages 184–221. Georgetown University Press, Washington, D.C., 1970.

Kenneth W. Church and William A. Gale. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54, 1991.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Techologies*, pages 176–181, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P11/P11-2031`.

Stephen Clark and James Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING 2004*, pages 282–288, Geneva, Switzerland, August 2004.

J. Cocke and J. I. Schwartz. Programming languages and their compilers. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

Michael Collins, Brian Roark, and Murat Saraclar. Discriminative syntactic language modeling for speech recognition. In *Proc. ACL*, 2005.

Brooke Cowan, Ivona Kučerová, and Michael Collins. A discriminative model for tree-to-tree translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 232–241, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-1628`.

Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24:87–185, 2001.

Prabuddha De and Thomas E. Morton. Scheduling to minimum makespan on unequal parallel processors. *Decision Sciences*, 11(4):586–602, October 1980.

Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, 32:505–536, 1985.

Steve DeNeefe and Kevin Knight. Synchronous tree adjoining machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 727–736, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D/D09/D09-1076`.

Steve DeNeefe, Kevin Knight, Wei Wang, and Daniel Marcu. 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 Computational Natural Language Learning (EMNLP-CoNLL)*, pages 755–763, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D/D07/D07-1079`.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

Yuan Ding. *Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars*. PhD thesis, University of Pennsylvania, 2006.

Yuan Ding and Martha Palmer. Automatic learning of parallel dependency treelet pairs. In *Proceedings of the 1st International Joint Conference on Natural Language Processing*, Hainan Island, China, March 2004a.

Yuan Ding and Martha Palmer. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *Proceedings of the Workshop on Recent Advances in Dependency Grammars, COLING-04*, Geneva, Switzerland, August 2004b.

Yuan Ding and Martha Palmer. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 541–548, Ann Arbor, Michigan, USA, June 2005.

Yuan Ding and Martha Palmer. Better learning and decoding for syntax based SMT using PSDIG. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas*, pages 37–45, Cambridge, Massachusetts, USA, August 2006.

Yuan Ding, Daniel Gildea, and Martha Palmer. An algorithm for word-level alignment of parallel dependency trees. In *Proceedings of MT Summit IX*, New Orleans, September 2003.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philp Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12, Uppsala, Sweden, July 2010.

Jay Earley. *An efficient context-free parsing algorithm*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 1968.

Jay Earley. An efficient context-free parsing algorithm. *CACM*, 13(2):94–102, 1970.

Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics*, pages 205–208, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075178.1075217. URL `http://www.aclweb.org/anthology/P03-2039`.

J.D. Ferguson. Hidden Markov analysis: An introduction. In J.D. Ferguson, editor, *Hidden Markov Models for Speech*, pages 8–15. IDA-CRD, Princeton, New Jersey, October 1980.

Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

Michel Galley and Christopher D. Manning. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 773–781, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P09/P09-1087`.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 273–280, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve Deneefe, Wei Wang, and Ignacio Thayer. Scalable inference and training of context-rich syntactic translation models. In *Proc. ACL*, 2006. URL http://www.aclweb.org/anthology-new/P/P06/P06-1121.pdf.

Niyu Ge. A direct syntax-driven reordering model for phrase-based machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 849–857, Los Angeles, California, June 2010. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N10-1127.

Andrea Gesmundo and James Henderson. Faster cube pruning. In *Proceedings of the Seventh International Workshop on Spoken Language Translation (IWSLT)*, pages 267–274, Paris, France, December 2010.

Daniel Gildea. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 80–87, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075107. URL http://www.aclweb.org/anthology/P03-1011.

Irving J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3–4):237–264, 1953.

Jonathan Graehl and Kevin Knight. Training tree transducers. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 105–112, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Ron L. Graham. Bounds on certain multiprocessing timing anomalies. *The Bell Systems Technical Journal*, 45(9):1563–1581, November 1966.

Ron L. Graham. Bounds on certain multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

Yvette Graham and Josef van Genabith. Deep syntax language models and statistical machine translation. In *Proceedings of the 4th Workshop on Syntax and Structure in Statistical Translation*, pages 118–126, Beijing, China, August 2010. Coling 2010 Organizing Committee. URL http://www.aclweb.org/anthology/W10-3815.

Liliane Haegeman. *Introduction to Government and Binding Theory*. Blackwell, Oxford, England, 2nd edition edition, 1994.

Aria Haghighi, John Blitzer, John DeNero, and Dan Klein. Better word alignments with supervised ITG models. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 923–931, Suntec, Singapore, August 2009. Association for Computational Linguistics.

Greg Hanneman and Alon Lavie. Decoding with syntactic and non-syntactic phrases in a syntax-based machine translation system. In *Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation (SSST-3) at NAACL HLT 2009*, pages 1–9, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W09-2301`.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2): 100–107, July 1968.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bulletin*, pages 28–29, December 1972.

Saša Hasan, Oliver Bender, and Hermann Ney. Reranking hypotheses using structural properties. In *Proceedings of the Workshop on Learning Structured Information in Natural Language Processing at the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 41–48, April 2006.

Hany Hassan, Khalil Sima'an, and Andy Way. Supertagged phrase-based statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 288–295, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P07-1037`.

Mary Hearne. *Data-Oriented Models of Parsing and Translation*. PhD thesis, Dublin City University, Dublin, Ireland, 2005.

Mary Hearne and Andy Way. Seeing the wood for the trees: Data-oriented translation. In *Proceedings of MT Summit IX*, New Orleans, September 2003.

James Henderson. Lookahead in deterministic left-corner parsing. In *Proc. Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 26–33, Barcelona, Spain, 2004.

Hieu Hoang. *Improving Statistical Machine Translation with Linguistic Information*. PhD thesis, University of Edinburgh, 2011.

Hieu Hoang and Philipp Koehn. Improving mid-range reordering using templates of factors. In *Proceedings of the 12th Conference of the European Chapter of the ACL*, pages 372–379, Athens, Greece, April–May 2009.

Hieu Hoang and Philipp Koehn. Improving translation with source syntax labels. In *Proceedings of the Joint 5th Workshop on Statistical Machine Translation and MetricsMATR*, pages 409–417, Uppsala, Sweden, July 2010.

Hieu Hoang, Philipp Koehn, and Adam Lopez. A unified framework for phrase-based, hierarchical, and syntax-based statistical machine translation. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 152–159, Tokyo, Japan, 2009.

Julia Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2003.

Mark Hopkins and Greg Langmead. Cube pruning as heuristic search. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 62–71, Singapore, August 2009.

T.C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9(6):841–848, 1961.

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

Liang Huang and Haitao Mi. Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283, Cambridge, MA, October 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D10-1027`.

Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, 2010.

Liang Huang, Kevin Knight, and Aravind Joshi. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas*, pages 66–73, Cambridge, USA, August 2006.

W. John Hutchins. Machine translation: half a century of research and use. Prepared for UNED Summer School, July 2003.

W. John Hutchins. Two precursors of machine translation: Artsrouni and Trojanskij. *International Journal of Translation*, 16(1):11–31, Jan–Jun 2004.

IBM. The Georgetown-IBM experiment. IBM Press Release, January 1954.

Kenji Imamura, Hideo Okuma, Taro Watanabe, and Eiichiro Sumita. Example-based machine translation based on syntactic transfer with statistical models. In *Proceedings of Coling 2004*, pages 99–105, Geneva, Switzerland, Aug 23–Aug 27 2004. COLING.

Ray Jackendoff. *$\bar{X}$ Syntax*, volume 2 of *Linguistic Inquiry Monographs*. MIT Press, Cambridge, Massachusetts, USA, 1977.

Robert J. Jarvella. Syntactic processing of connected speech. *Journal of Verbal Learning and Verbal Behavior*, 10:409–416, 1971.

Frederick Jelineck and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands, May 1980.

Frederick Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, pages 675–685, 1969.

Mark Johnson. Finite state approximation of constraint-based grammars using left-corner grammar transforms. In *Proceedings of COLING/ACL*, pages 619–623, Montreal, Canada, 1998.

Aravind K. Joshi. How much context sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars. In L. Karttunen D. Dowty and A. Zwicky, editors,

Natural language parsing: Psychological, computational and theoretical perspectives, pages 206–250. Cambridge University Press, Cambridge, U.K., 1985.

Marcel Adam Just and Patricia A. Carpenter. A capacity theory of comprehension: Individual differences in working memory. *Psychological Review*, 99:122–149, 1992.

Ronald M. Kaplan and Joan Bresnan. Lexical functional grammar, a formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Massachusetts, USA, 1982.

T. Kasami. An efficient recognition and syntax analysis algorithm for context free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Tranactions on Acoustics, Speech and Signal Processing*, 35 (3):400–401, March 1987.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 181–184, 1995.

Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proc. AMTA*, 2004.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

Philipp Koehn and Hieu Hoang. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 868–876, Prague, Czech Republic, June 2007.

Philipp Koehn, Franz Joseph Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of HLT-NAACL 2003*, pages 127–133, Edmonton, Canada, 2003. URL `http://www.iccs.inf.ed.ac.uk/~pkoehn/publications/phrase2003.pdf`.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej

Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL 2007 Demo and Poster Sessions*, 2007.

Andás Kornai and Geoffrey Pullum. The X-Bar theory of phrase structure. *Language*, 66: 24–50, March 1990.

Shankar Kumar and William Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of HLT-NAACL 2003*, pages 63–70, Edmonton, Canada, May–June 2003.

Alon Lavie, Alok Parlikar, and Vamshi Ambati. Syntax-driven learning of sub-sentential translation equivalents and translation rules from parsed parallel corpora. In *Proceedings of the ACL-08: HLT Second Workshop on Syntax and Structure in Statistical Translation (SSST-2)*, pages 87–95, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W08/W08-0411`.

Winfred Lehmann. Structure of noun phrases in german. In Léon Dostert, editor, *Report of the Eighth Round Table Meeting on Linguistics and Language Study: Research in Machine Translation*, volume 10 of *Monograph Series on Languages and Linguistics*, pages 125–133. Georgetown University Press, Washington, D.C., 1957.

Winfred Lehmann. Machine translation at Texas: The early years. http://www.utexas.edu/cola/centers/lrc/mt/earlymt.html, May 1998.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March 2009a. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0x24`.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Demonstration of Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL'09), Software Demonstrations*, Singapore, August 2009b.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Anne Irvine, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Ziyuan Wang, Jonathan Weese, and Omar Zaidan. Joshua 2.0: A toolkit for parsing-based machine translation with syntax, semirings, discriminative training and other goodies. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 133–137, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

George Lidstone. Notes on the general case of the Bayes-Laplace formula for inductive or *a posteriori* probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

Yang Liu, Qun Liu, and Shouxun Lin. Tree-to-string alignment templates for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL-CoLing-2006)*, Sydney, Australia, 2006.

Yang Liu, Yun Huang, Qun Liu, and Shouxun Lin. Forest-to-string statistical translation rules. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 704–711, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P07-1089`.

Yang Liu, Yajuan Lü, and Qun Liu. Improving tree-to-tree translation with packed forests. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 558–566, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P09/P09-1063`.

Nitin Madnani. iBLEU: Interactively debugging and scoring statistical machine translation systems. In *Proceedings of the IEEE Fifth International Conference on Semantic Computing*, pages 213–214, Palo Alto, California, September 2011.

David Magerman. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 276–283, Cambridge, MA, 1995.

Daniel Marcu. Towards a unified approach to memory- and statistical-based machine translation. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 386–393, Toulouse, France, July 2001. Association for Computational Linguistics. doi: 10.3115/1073012.1073062. URL http://www.aclweb.org/anthology/P01-1050.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. Spmt: Statistical machine translation with syntactified target language phrases. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 44–52, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W06/W06-1606.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Andrei A. Markov. An example of statistical investigation in the text of 'Eugene Onyegin' illustrating coupling of 'tests' in chains. In *Proceedings of the Academy of Sciences, St. Petersburg*, volume 7, pages 153–162, 1913.

Yuval Marton and Philip Resnik. Soft syntactic constraints for hierarchical phrased-based translation. In *Proceedings of ACL-08: HLT*, pages 1003–1011, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P08/P08-1114.

Robert McEliece, David MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of Pearl's "Belief Propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2): 140–152, February 1998.

Dan Melamed. Statistical machine translation by parsing. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-2004)*, Barcelona, Spain, 2004.

Igor Mel'čuk. *Dependency syntax: theory and practice*. SUNY Series in Linguistics. State University of New York Press, Albany, New York, USA, 1988.

Haitao Mi and Liang Huang. Forest-based translation rule extraction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 206–214, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D08-1022`.

Haitao Mi, Liang Huang, and Qun Liu. Forest-based translation. In *Proceedings of ACL-08: HLT*, pages 192–199, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P08/P08-1023`.

George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

Tim Miller. Word buffering models for improved speech repair parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore, 2009a.

Tim Miller. Improved syntactic models for parsing speech with repairs. In *Proceedings of the North American Association for Computational Linguistics*, Boulder, CO, 2009b.

Tim Miller and William Schuler. A unified syntactic model for parsing fluent and disfluent speech. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL '08), short papers*, pages 105–108, 2008a.

Tim Miller and William Schuler. A syntactic time-series model for parsing fluent and disfluent speech. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING'08)*, 2008b.

Tim Miller and William Schuler. An empirical evaluation of HHMM parsing time. In *Proceedings of the Midwest Computational Linguistics Colloquium*, 2008c.

Tim Miller and William Schuler. Hhmm parsing with limited parallelism. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics '10)*, 2010.

Tim Miller, Lane Schwartz, and William Schuler. Incremental semantic models for continuous context-sensitive speech recognition. In *Proceedings of the Workshop on Semantic Representation of Spoken Language (SRSL'07)*, Salamanca, Spain, November 2007. URL `http://www-users.cs.umn.edu/~schuler/paper_srsl07.pdf`.

Tim Miller, Luan Nguyen, and William Schuler. Parsing speech repair without specialized grammar symbols. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 277–280, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P09/P09-2070.

Kevin P. Murphy and Mark A. Paskin. Linear time inference in hierarchical HMMs. In *Proc. NIPS*, pages 833–840, Vancouver, BC, Canada, 2001.

Markos Mylonakis and Khalil Sima'an. Phrase translation probabilities with ITG priors and smoothing as learning objective. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 630–639, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1066.

Rebecca Nesson, Stuart Shieber, and Alexander Rush. Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proc. AMTA*, 2006.

Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, 8:1–38, 1994.

Franz Och. Minimum error rate training in statistical machine translation. In *Proc. ACL*, pages 160–167, Sapporo, Japan, July 2003. URL http://www.fjoch.com/acl03.pdf.

Franz Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. ACL*, 2002.

Franz Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449, 2004. URL http://www.aclweb.org/anthology-new/J/J04/J04-4002.pdf.

Franz Och, Christoph Tillman, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proc. SIGDAT-EMNLP*, 1999.

Franz Josef Och and Hans Weber. Improving statistical natural language translation with categories and rules. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 985–989, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics. doi: 10.3115/980691.980731. URL `http://www.aclweb.org/anthology/P98-2031`.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. A smorgasbord of features for statistical machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 161–168, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Franz Joseph Och. *Statistical Machine Translation: From Single-Word Models to Alignment Templates*. PhD thesis, RWTH Aachen, Germany, 2002.

S.S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1): 45–61, 1977.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*, pages 311–318, 2001. URL `http://acl.ldc.upenn.edu/P/P02/P02-1040.pdf`.

Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pages 133–136, Pittsburgh, Pennsylvania, 1982. Morgan Kaufmann.

Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29: 241–288, 1986.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

Aaron Phillips and Ralf Brown. Cunei machine translation platform: System description. In *Proceedings of the 3rd Workshop on Example-Based Machine Translation*, Dublin, Ireland, November 2009.

Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994.

Matt Post and Daniel Gildea. Parsers as language models for statistical machine translation. In *Proc. AMTA*, October 2008.

Matt Post and Daniel Gildea. Language modeling with tree substitution grammars. In *NIPS workshop on Grammar Induction, Representation of Language, and Language Learning*, Whistler, British Columbia, Dec 2009.

Arjen Poutsma. Data-oriented translation. In *Ninth Conference of Computational Linguistics in the Netherlands*, Leuven, Belgium, 1998.

Arjen Poutsma. Data-oriented translation. In *Proc. COLING*, 2000.

Arjen Poutsma. Machine translation with Tree-DOP. In R. Bod, R. Scha, and K. Sima'an, editors, *Data-Oriented Parsing*, pages 339–357. CSLI Publications, Stanford, CA, 2003.

Michael Pust and Kevin Knight. Faster MT decoding through pervasive laziness. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 141–144, Boulder, Colorado, June 2009.

Chris Quirk and Simon Corston-Oliver. The impact of parse quality on syntactically-informed statistical machine translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 62–69, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-1608`.

Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*, Ann Arbor, Michigan, 2005. URL `http://research.microsoft.com/users/chrisq/t2s_acl2005.pdf`.

Erwin Reifler. The machine translation project at the University of Washington. *Mechanical Translation*, 6:25–32, November 1961.

Jason Riesa and Daniel Marcu. Hierarchical search for word alignment. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 157–166, Uppsala, Sweden, July 2010.

Stefan Riezler and John T. Maxwell, III. Grammatical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 248–255, New York, June 2006.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276, 2001.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Pearson Education, Upper Saddle River, New Jersey, second edition, 2003.

Jacqueline Sachs. Recognition memory for syntactic and semantic aspects of connected discourse. *Perception and Psychophysics*, 2:437–442, 1967.

Markus Saers and Dekai Wu. Improving phrase-based translation via word alignments from Stochastic Inversion Transduction Grammars. In *Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation (SSST-3) at NAACL HLT 2009*, pages 28–36, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W09-2304`.

Joan Andreu Sánchez and José Miguel Benedí. Stochastic inversion transduction grammars for obtaining word phrases for phrase-based statistical machine translation. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 130–133, New York City, June 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-3117`.

William Schuler. Parsing with a bounded stack using a model-based right-corner transform. In *Proceedings of NAACL*, pages 344–352, Boulder, Colorado, 2009.

William Schuler. Incremental parsing in bounded memory. In *Proceedings of the 10th International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+10)*, New Haven, CT, 2010.

William Schuler and Tim Miller. Integrating denotational meaning into a DBN language model. In *Proceedings of the 9th European Conference on Speech Communication and Technology / 6th Interspeech Event (Eurospeech/Interspeech'05)*, pages 901–904, Lisbon, Portugal, 2005.

William Schuler, Tim Miller, Andrew Exley, and Stephen Wu. Dynamic evidence models in a dbn phone recognizer. In *Proceedings of the 9th International Conference on Spoken Language Processing (ICSLP/Interspeech'06)*, pages 1221–1224, Pittsburgh, Pennsylvania, 2006.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. Toward a psycholinguistically-motivated model of language. In *Proceedings of COLING*, pages 785–792, Manchester, UK, August 2008.

William Schuler, Stephen Wu, and Lane Schwartz. A framework for fast incremental interpretation during speech decoding. *Computational Linguistics*, 35(3):313–343, 2009.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. Broad-coverage incremental parsing using human-like memory constraints. *Computational Linguistics*, 36(1), 2010.

Lane Schwartz. Multi-source translation methods. In *Proc. AMTA*, October 2008a. URL `http://www-users.cs.umn.edu/~lane/papers/amta08.pdf`.

Lane Schwartz. An open-source hierarchical phrase-based translation system. In *Proceedings of the 5th Midwest Computational Linguistics Colloquium (MCLC'08)*, East Lansing, Michigan, May 2008b. URL `http://www-users.cs.umn.edu/~lane/papers/mclc08.pdf`.

Lane Schwartz. Reproducible results in parsing-based machine translation: The JHU shared task submission. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 177–182, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Lane Schwartz and Chris Callison-Burch. Hierarchical phrase-based grammar extraction in Joshua: Suffix arrays and prefix trees. *The Prague Bulletin of Mathematical Linguistics*, 93: 157–166, January 2010.

Lane Schwartz, Luan Nguyen, Andrew Exley, and William Schuler. Positive effects of redundant descriptions in an interactive semantic speech interface. In *Proceedings of the 2009 International Conference on Intelligent User Interfaces (IUI'09)*, pages 217–226, Sanibel Island, FL., 2009.

Lane Schwartz, Chris Callison-Burch, William Schuler, and Stephen Wu. Incremental syntactic language models for phrase-based translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Techologies*, pages 620–631, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P11/P11-1063`.

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

Claude Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64, 1951. URL `http://www.princeton.edu/~wbialek/rome/refs/shannon_51.pdf`.

Libin Shen, Jinxi Xu, and Ralph Weischedel. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proc. ACL*, 2008.

Stuart M. Shieber. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, 2004.

Stuart M. Shieber and Yves Schabes. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*, Helsinki, Finland, August 1990.

Khalil Sima'an and Markos Mylonakis. Better statistical estimation can benefit all phrases in phrase-based statistical machine translation. In *Proceedings of the IEEE Workshop on Spoken Language Translation*, Goa, India, 2008.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proc. AMTA*, pages 223–231, 2006. URL `http://www.cs.umd.edu/~snover/pub/amta06/ter_amta.pdf`.

Mark Steedman. *The syntactic process*. MIT Press/Bradford Books, Cambridge, MA, 2000.

Michael K. Tanenhaus, Michael J. Spivey-Knowlton, Kathy M. Eberhard, and Julie E. Sedivy. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268:1632–1634, 1995.

Lucien Tesnière. *Éléments de syntaxe structurale*. Librarie Klincksieck, Paris, 1959.

John Tinsley, Mary Hearne, and Andy Way. Exploiting parallel treebanks to improve phrase-based statistical machine translation. In *Proceedings of the Sixth International Workshop on Treebanks and Linguistic Theories*, pages 175–187, Bergen, Norway, December 2007a.

John Tinsley, Ventsislav Zhechev, Mary Hearne, and Andy Way. Robust language-pair independent sub-tree alignment. In *Proceedings of Machine Translation Summit XI*, pages 467–474, Copenhagen, Denmark, September 2007b.

Bernard Vauquois. A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In J.H. Morrell, editor, *Proceedings of the International Federation for Information Processing Congress (IFIP-68)*, volume 2, pages 1114–1122, Edinburgh, Scotland, August 1968.

Ashish Venugopal, Stephan Vogel, and Alex Waibel. Effective phrase translation extraction from alignment models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 319–326, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075137. URL `http://www.aclweb.org/anthology/P03-1041`.

David Vilar, Daniel Stein, Matthias Huck, and Hermann Ney. Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 262–270, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W10-1738`.

Andrew Viterbi. Error bounds for convolution codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

Wei Wang, Kevin Knight, and Daniel Marcu. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *Proceedings of the 2007 Joint Conference on Empirical*

Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 746–754, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D07/D07-1078.

Ye-Yi Wang and Alex Waibel. Modeling with structures in statistical machine translation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 1357–1363, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics. doi: 10.3115/980691.980790. URL http://www.aclweb.org/anthology/P98-2090.

Taro Watanabe, Eiichiro Sumita, and Hiroshi G. Okuno. Chunk-based statistical translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 303–310, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075135. URL http://www.aclweb.org/anthology/P03-1039.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. Left-to-right target generation for hierarchical phrase-based translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 777–784, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220273. URL http://www.aclweb.org/anthology/P06-1098.

Warren Weaver. Translation. Memo, 1949. Published 1955 in *Machine translation of languages: fourteen essays*.

Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probability of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4): 1085–1094, July 1991.

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403, 1997.

Stephen Wu, Lane Schwartz, and William Schuler. Referential semantic language modeling for data-poor domains. In *Proc. ICASSP*, 2008a.

Stephen Wu, Lane Schwartz, and William Schuler. Exploiting referential context in spoken language interfaces for data-poor domains. In *Proc. International Conference on Intelligent User Interfaces (IUI'08)*, Canary Islands, Spain, January 2008b. URL `http://www-users.cs.umn.edu/~schuler/paper_iui08.pdf`.

Stephen Wu, Asaf Bachrach, Carlos Cardenas, and William Schuler. Complexity metrics in an incremental right-corner parser. In *Proceedings of the 49th Annual Conference of the Association for Computational Linguistics (ACL'10)*, 2010.

Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001)*, Toulouse, France, 2001. URL `http://www.isi.edu/natural-language/projects/rewrite/syntax.ps`.

Kenji Yamada and Kevin Knight. A decoder for syntax-based statistical mt. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, Philadelphia, Pennsylvania, 2002. URL `http://www.isi.edu/natural-language/projects/rewrite/syndec.ps`.

Hirofumi Yamamoto, Hideo Okuma, and Eiichiro Sumita. Imposing constraints from the source tree on ITG constraints for SMT. In *Proceedings of the ACL-08: HLT Second Workshop on Syntax and Structure in Statistical Translation (SSST-2)*, pages 1–9, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W08/W08-0401`.

D.H. Younger. Recognition and parsing of context-free languages in time n cubed. *Information and Control*, 10(2):189–208, 1967.

Richard Zens and Hermann Ney. Improvements in phrase-based statistical machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 257–264, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Richard Zens, Frans Joseph Och, and Hermann Ney. Phrase-based statistical machine translation. In M. Jarke, J. Koehler, and G. Lakemeyer, editors, *KI — 2002: Advances in Artificial*

Intelligence. 25. Annual German Conference on AI, KI 2002, volume LNAI 2479, pages 18–32. Springer Verlag, September 2002.

Richard Zens, Hermann Ney, Taro Watanabe, and Eiichiro Sumita. Reordering constraints for phrase-based translation. In *Proceedings of COLING 2004*, pages 205–211, Geneva, Switzerland, August 2004.

Min Zhang, Hongfei Jiang, Ai Ti Aw, Jun Sun, Seng Li, and Chew Lim Tan. A tree-to-tree alignment-based model for statistical machine translation. In *Proc. MT Summit*, 2007.

Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan, and Sheng Li. A tree sequence alignment-based tree-to-tree translation model. In *Proceedings of ACL-08: HLT*, pages 559–567, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P08/P08-1064`.

Bing Zhao and Stephan Vogel. Word alignment based on bilingual bracketing. In Rada Mihalcea and Ted Pedersen, editors, *Proceedings of the HLT-NAACL 2003 Workshop on Building and Using Parallel Texts: Data Driven Machine Translation and Beyond*, pages 15–18, 2003. URL `http://www.aclweb.org/anthology/W03-0303.pdf`.

Hao Zhao and Daniel Gildea. Stochastic lexicalized inversion transduction grammar for alignment. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*, pages 475–482, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

Bowen Zhou, Bing Xiang, Xiaodan Zhu, and Yuqing Gao. Prior derivation models for formally syntax-based translation using linguistically syntactic parsing and tree kernels. In *Proceedings of the ACL-08: HLT Second Workshop on Syntax and Structure in Statistical Translation (SSST-2)*, pages 19–27, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W08/W08-0403`.

Andreas Zollmann and Ashish Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 138–141, New York City, June 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-3119`.

Appendix A

Tree & Model Transformations: Implementation Details

A.1 Initial Preprocessing

Trees from the WSJ treebank (sections 2-21) are preprocessed. After preprocessing, all training data has been concatenated into a single file, `genmodel/wsjTRAIN.linetrees`, containing one tree per line.

```
for i in 02 .. 21; do
    perl scripts/tbtrees2linetrees.pl  \
         < wsj/${i}/*.mrg              \
         > genmodel/wsj${i}.linetrees
done
```

In the interest of maximal reproducibility, the script that performs this processing step, `scripts/tbtrees2linetrees.pl`, is listed in Appendix B.1.

```
cat genmodel/*.linetrees                    \
             | perl scripts/annotateFixes   \
             > genmodel/wsjTRAIN.linetrees
```

The WSJ treebank contains some annotation errors; the script `scripts/annotateFixes.pl`, listed in Appendix B.2, addresses certain of these annotation errors.

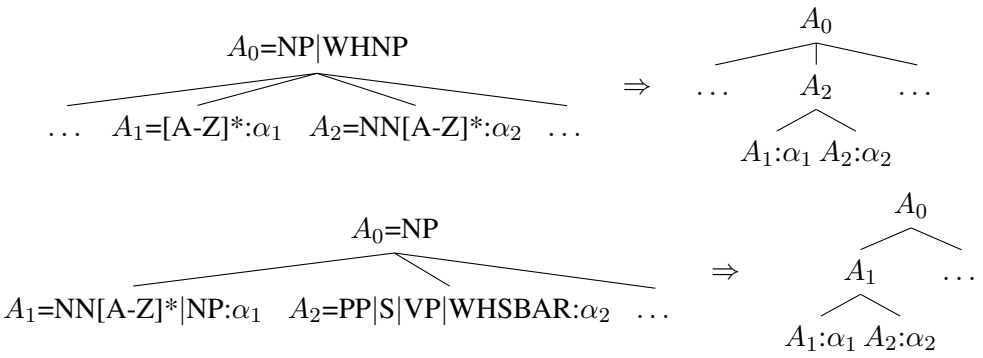
A.2 Binarization

After initial preprocessing (Appendix A.1), trees from the WSJ treebank (sections 2-21) are binarized, and terminal tokens are lowercased. Punctuation is retained. The resulting binary trees are stored in a single file, `genmodel/wsjTRAIN.projtrees`. In the interest of maximal reproducibility, the script that performs this processing step, `scripts/ annotateProjs.pl`, is listed in Appendix B.3.

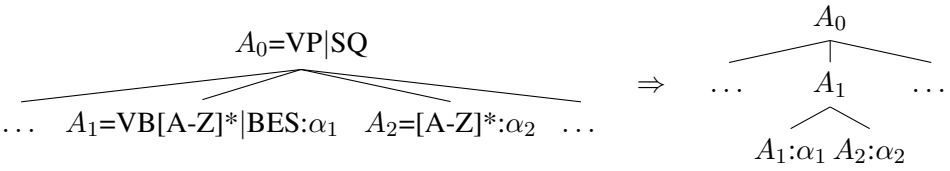
```
perl scripts/annotateProjs.pl < genmodel/wsjTRAIN.linetrees \
                              > genmodel/wsjTRAIN.projtrees
```

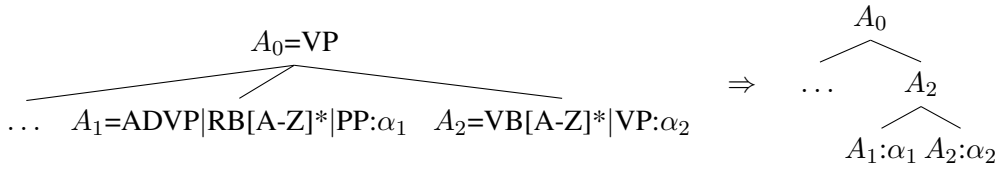
Binarization rules are documented in the implementing script, where they are implemented in Perl as regular expression substitution operations. Following Schuler et al. (2010) we list the most important binarization rules below. These binarization rules are designed to convert flat constituents into linguistically motivated head projections. In the rules below, vertical bar |, square brackets [], Kleene star *, and optional marker ? should be read as regular expression operators.

1. NP: right-binarize basal NPs as much as possible; then left-binarize NPs after left context reduced to nil:

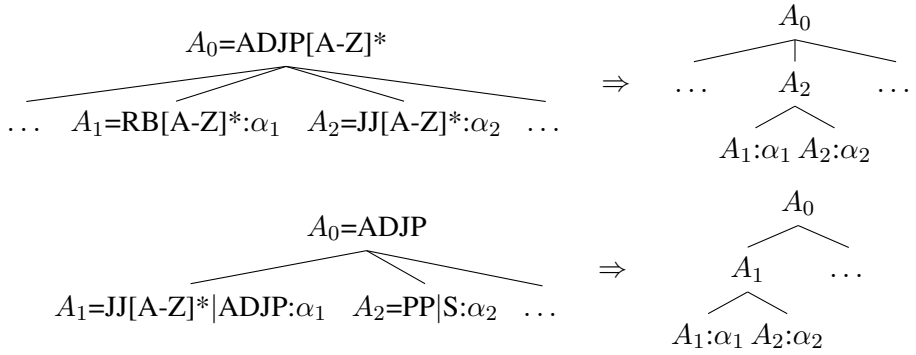


2. VP: left-binarize basal VPs as much as possible; then right-binarize VPs after right context reduced to nil:

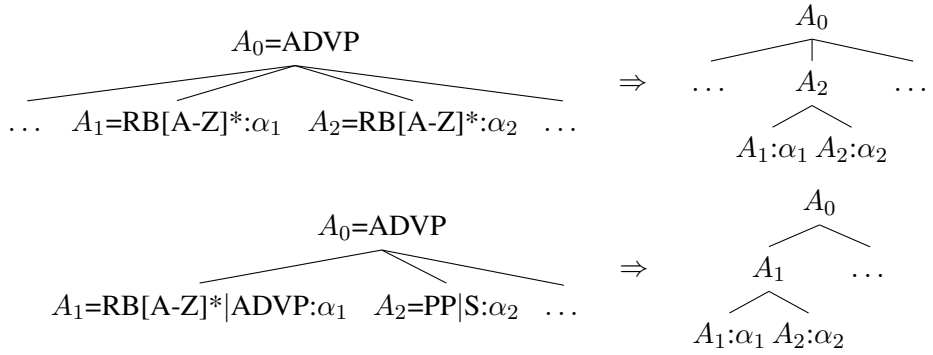




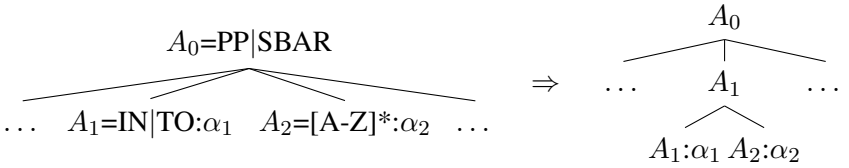
3. ADJP: right-binarize basal ADJPs as much as possible; then left-binarize ADJPs after left context reduced to nil:

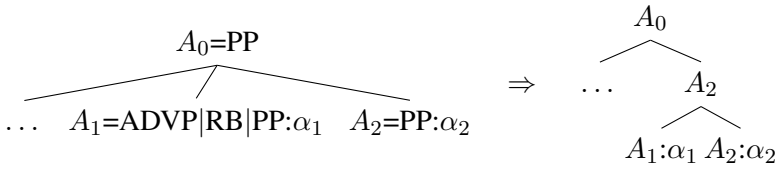


4. ADVP: right-binarize basal ADVPs as much as possible; then left-binarize ADVPs after left context reduced to nil:

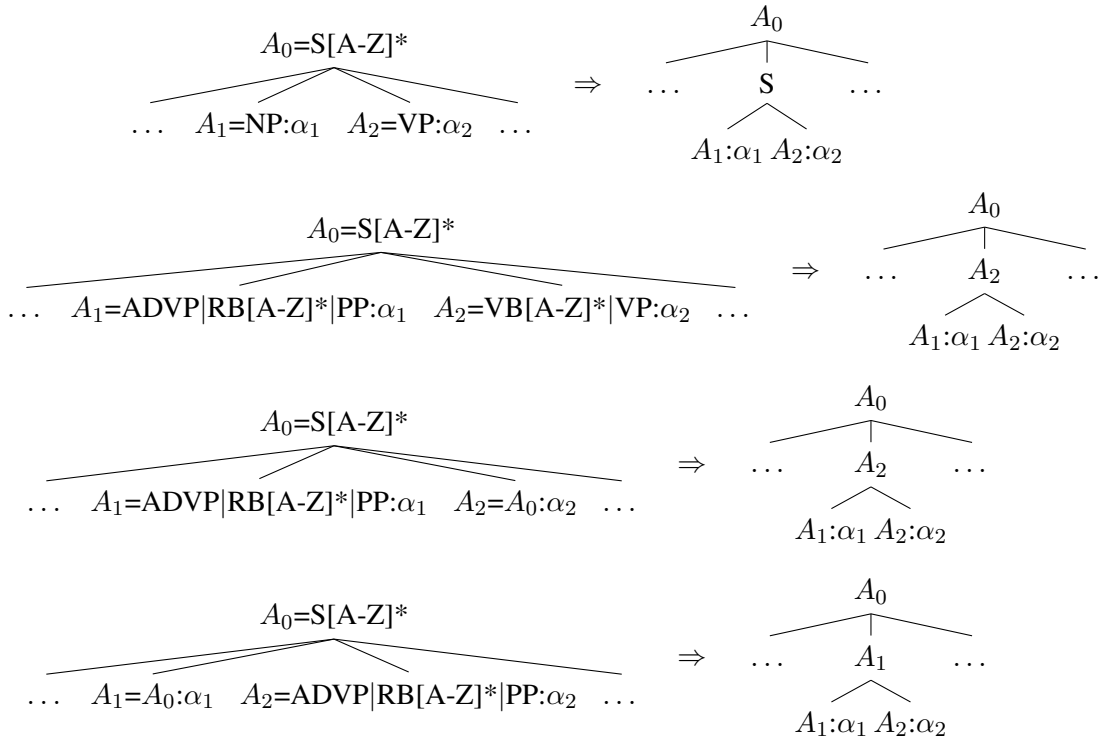


5. PP: left-binarize PPs as much as possible; then right-binarize PPs after right context reduced to nil:





6. S: group subject NP and predicate VP of a sentence; then group modifiers to right and left:

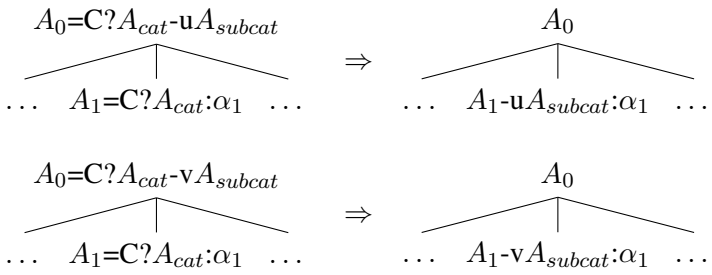


A.3 Argument Structure

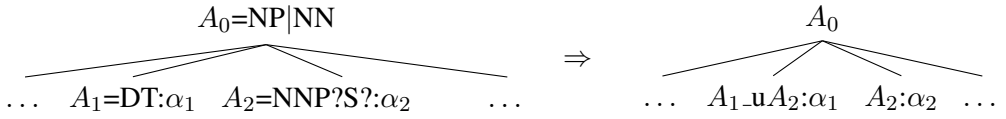
After binarization (Appendix A.2), trees from the WSJ treebank (sections 2-21) are processed; argument structure is identified, and subcategorization tags are added. In the interest of maximal reproducibility, the script that performs this processing step, `scripts/ annotateArgs.pl`, is listed in Appendix B.4.

```
perl scripts/annotateArgs.pl < genmodel/wsjTRAIN.projtrees \
                             > genmodel/wsjTRAIN.argtrees
```

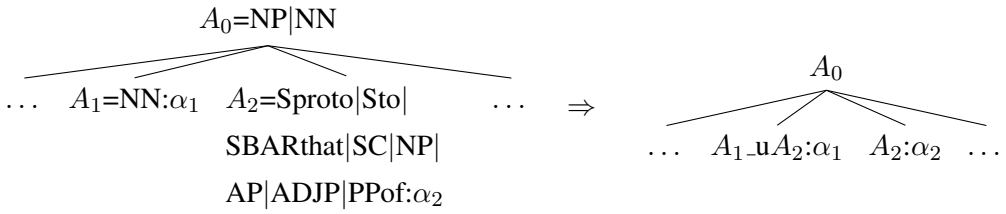
1. Pass existing subcategorization argument (-u) and passive (-v) tags from parent to each child whose catagory exactly matches the parent's, or whose category differs from the parent's only by the presence or absence of the C- conjunction-marking prefix.



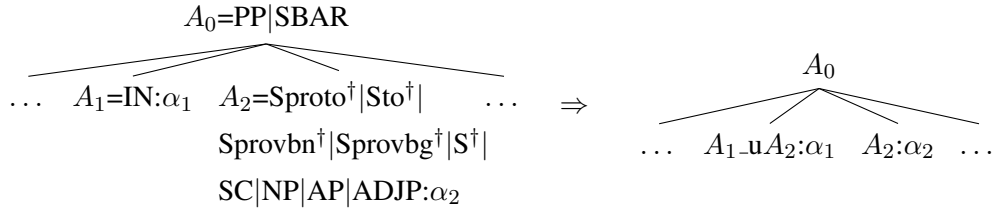
2. Subcategorize the head of an determiner-headed noun phrase with its noun argument type



3. Subcategorize the noun head of a noun-headed phrase with its argument type

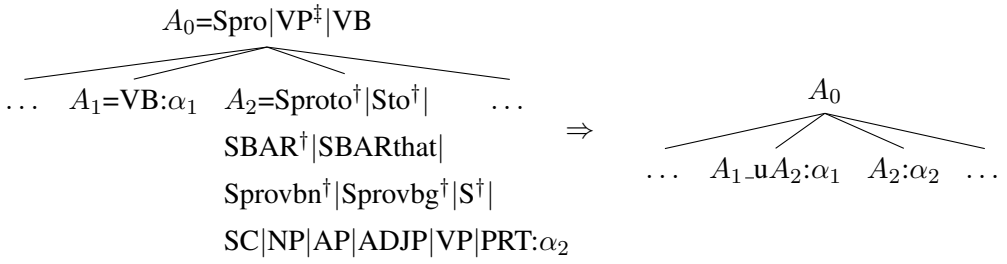


4. Subcategorize the preposition of a prepositional-headed phrase with its argument type

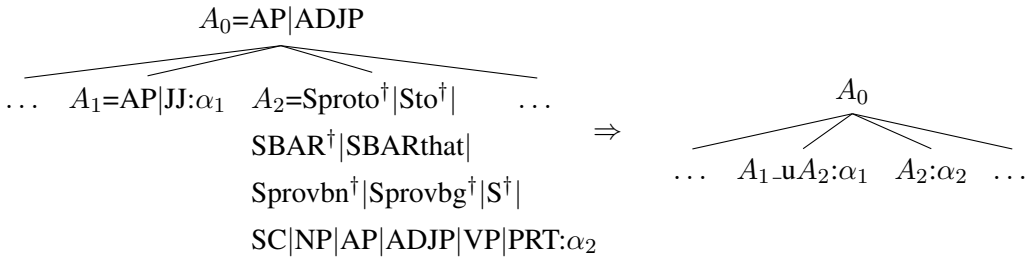


† Indicates that category may not end in suffix -adv.
‡ Indicates that category may not end in suffix -it.

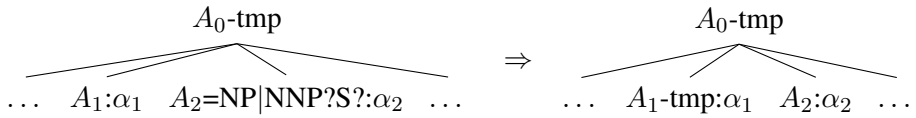
5. Subcategorize the verb of a verb-headed phrase with its argument type



6. Subcategorize the head of an adjective-/adverb-headed phrase with its argument type



7. In addition to the above rules, temporal markers are passed down from parent to child



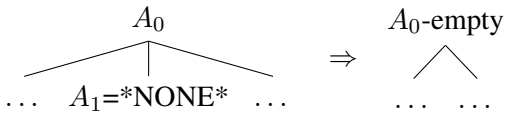
A.4 Traces

After argument structure processing (Appendix A.3), trees from the WSJ treebank (sections 2-21) are further processed to handle traces. In the interest of maximal reproducibility, the script that performs this processing step, `scripts/annotateGaps.pl`, is listed in Appendix B.5.

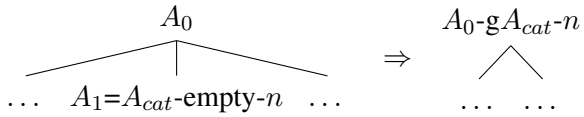
```
perl scripts/annotateGaps.pl < genmodel/wsjTRAIN.argtrees \
                             > genmodel/wsjTRAIN.gaptrees
```

In the rules below, n=[0-9]+ and the suffix X-g indicates that the subtree headed by the constituent contains an unfilled gap.

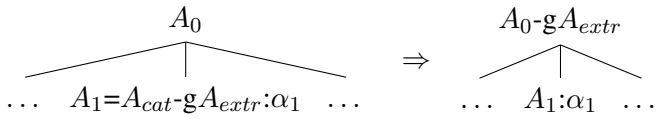
1. Fold trace corresponding to an empty child node into the parent as an extraction, removing the empty child from the tree



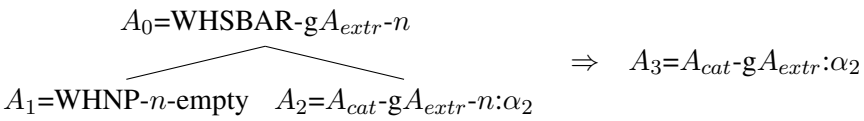
2. Fold child empty category into parent nonterm as extraction.



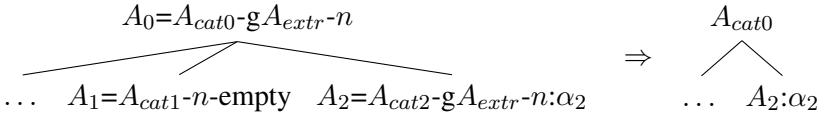
3. Project any extraction upward in the tree, from child to parent



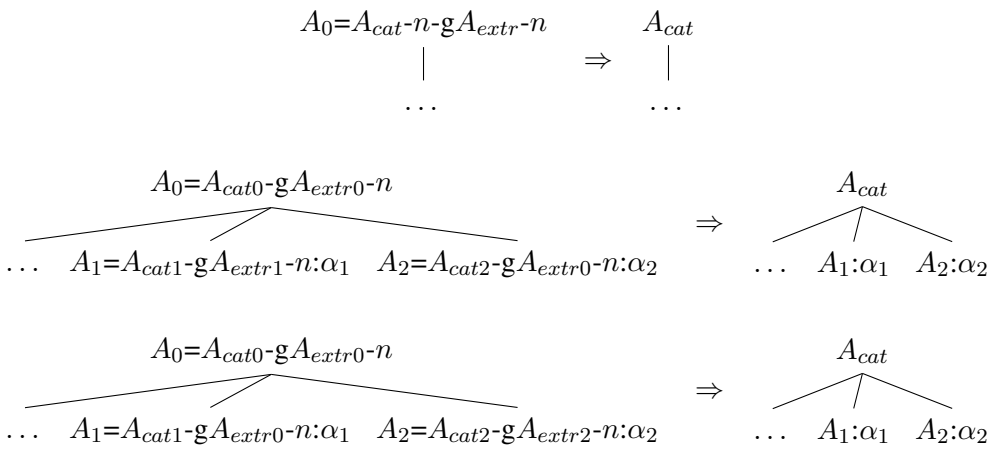
4. When an extraction is projected onto a WHSBAR, the children match (n in left child matches right child) and the left child is an empty extraction, eliminate the projection, the parent and the left child



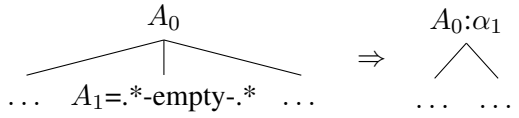
5. When the children match (n in left child matches right child) and the left child is an empty extraction, eliminate the projection and the left child



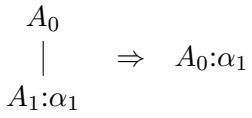
6. Undo extraction projection in the parent when n of children indicates the extraction gap has been filled below this parent



7. Remove any remaining empty constituents



8. Remove the child of any remaining unary subtrees



A.5 Punctuation

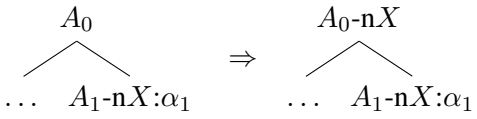
After trace processing (Appendix A.4), trees from the WSJ treebank (sections 2-21) are further processed to handle punctuation. In the interest of maximal reproducibility, the script that performs this processing step, `scripts/annotateMarks.pl`, is listed in Appendix B.6.

```
perl scripts/annotateMarks.pl < genmodel/wsjTRAIN.gaptrees \
                              > genmodel/wsjTRAIN.marktrees
```

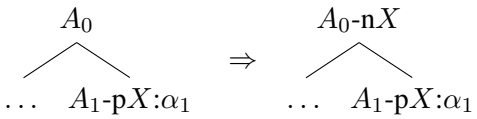
In the rules below, suffix X is a variable that indicates the type of punctuation. It can be bound to value B (bracket or parenthesis)[1] [2] , C (comma), D (dash)[3] , E (end of sentence punctuation), or S (semicolon).

[1] In prior preprocessing, each left bracket and left parenthesis was replaced with the literal string !LRB!
[2] In prior preprocessing, each right bracket and right parenthesis was replaced with the literal string !RRB!
[3] In prior preprocessing, each dash was replaced with the literal string !dash!

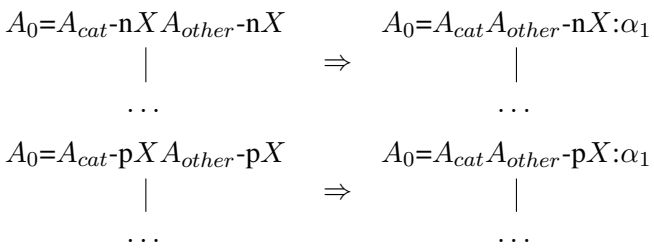
1. Propogate existing annotation of punctuation following constituent (-n) upward in the tree, from child to parent



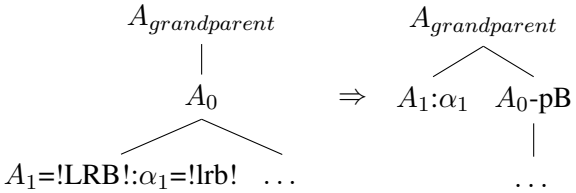
2. Propogate existing annotation of punctuation preceding constituent (-p) upward in the tree, from child to parent

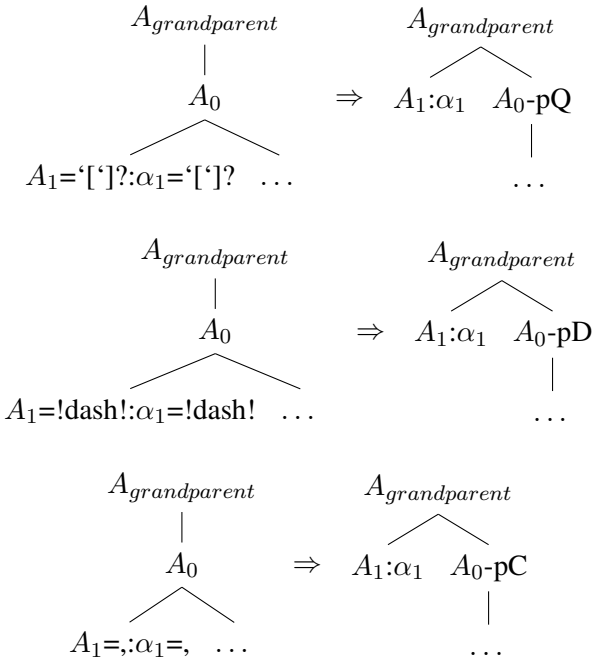


3. Remove any duplicate punctuation tags from parent. In this rule, A_{other} represents any other tags with which the parent may be annotated. A_{other} may be empty

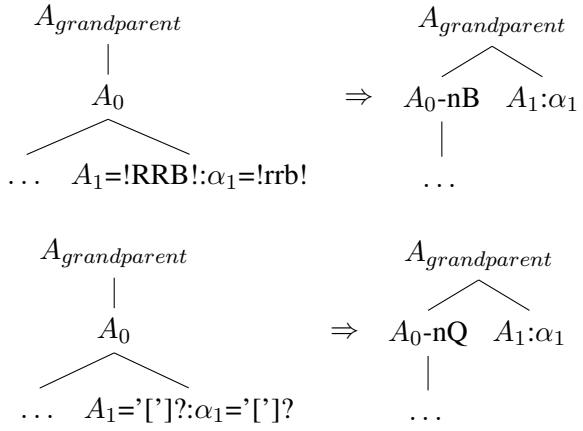


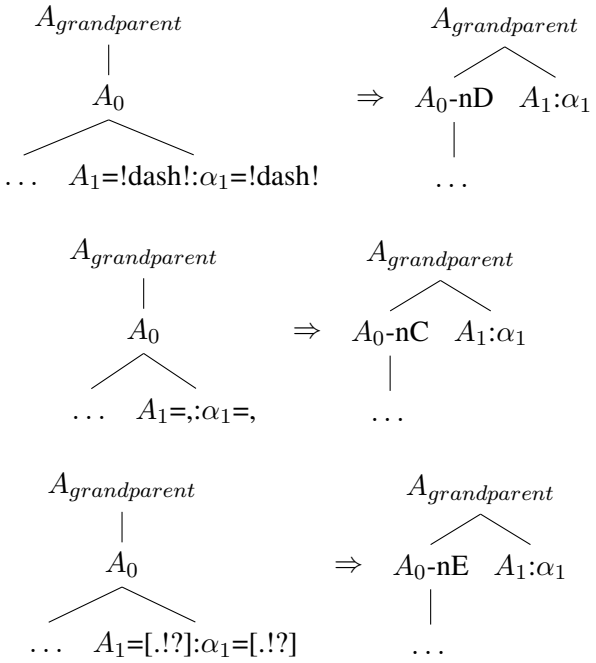
4. Move left punctuation marks up in the tree. Given a node containing a left punctuation mark, and its parent, the node is moved to become a left sibling of its (now former) parent. The former parent is annotated with -pX, indicating that it is preceded by a left punctuation mark of type X.



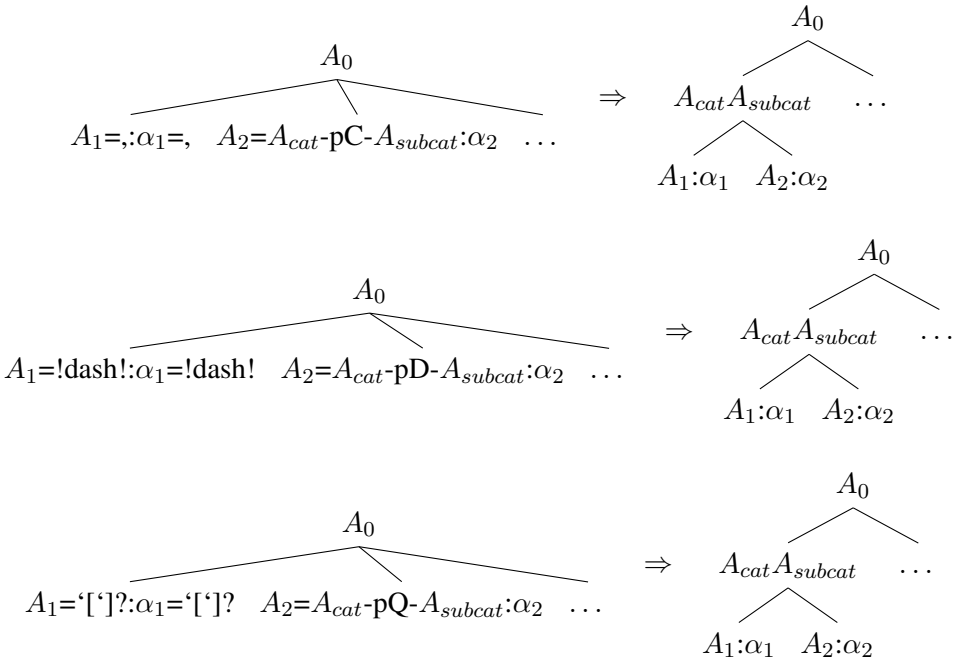


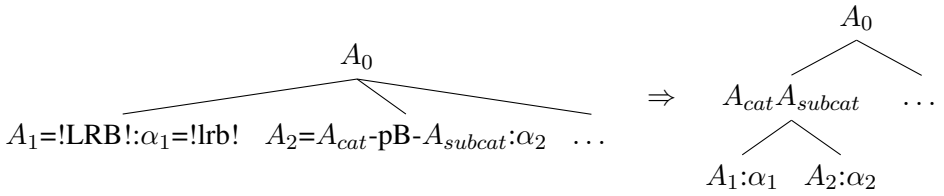
5. Move right punctuation marks up in the tree. Given a node containing a right punctuation mark, and its parent, the node is moved to become a right sibling of its (now former) parent. The former parent is annotated with -nX, indicating that it is followed by a right punctuation mark of type X.



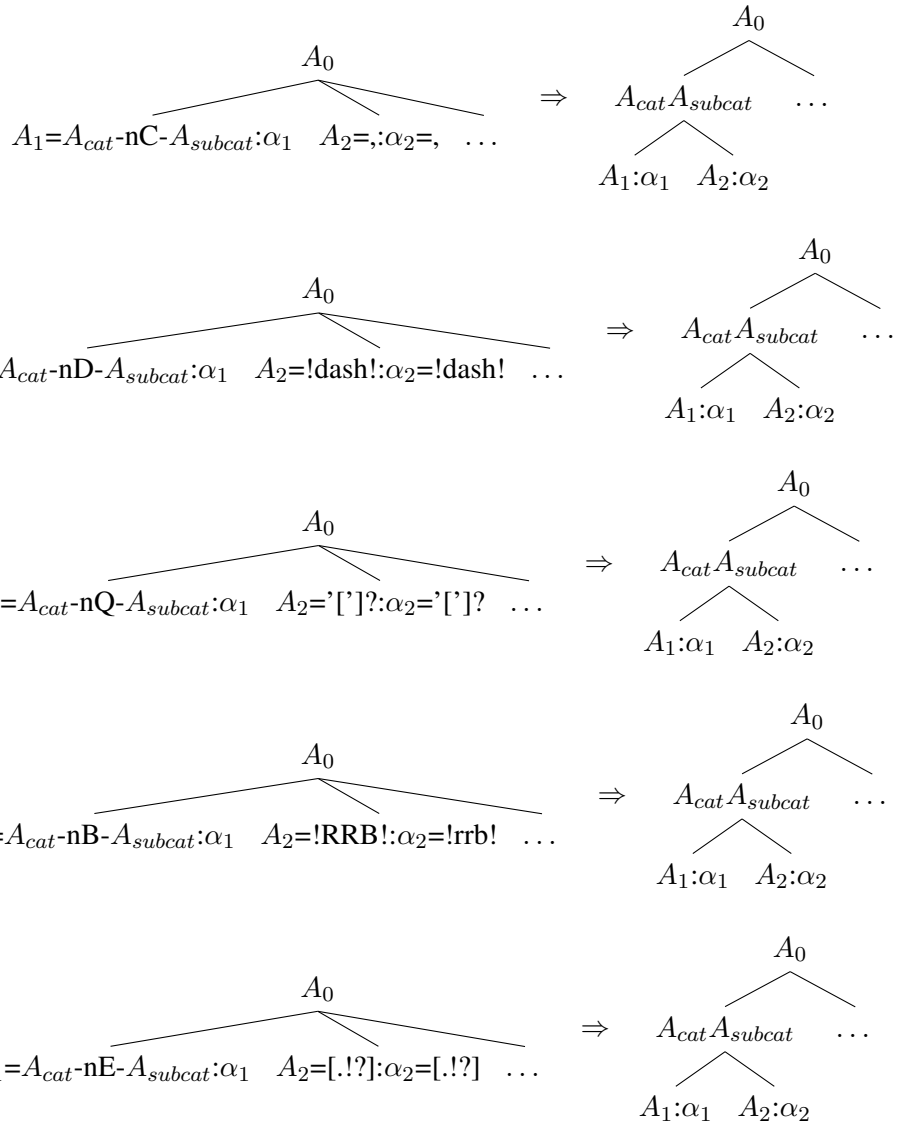


6. After any left punctutation mark has been propagated as high in the tree as possible, clean up by pushing it back down one level in the tree





7. After any right punctutation mark has been propagated as high in the tree as possible, clean up by pushing it back down one level in the tree



8. Remove the parent of any unary subtrees that remain

$$\begin{array}{c} A_0 \\ | \\ A_1{:}\alpha_1 \end{array} \quad \Rightarrow \quad A_1{:}\alpha_1$$

A.6 Normalization

After punctuation processing (Appendix A.5), trees from the WSJ treebank (sections 2-21) are processed to ensure they are in Chomsky Normal Form. In the interest of maximal reproducibility, the script that performs this processing step, `scripts/ensureCnf.pl`, is listed in Appendix B.7.

```
perl scripts/ensureCnf.pl < genmodel/wsjTRAIN.marktrees \
                          > genmodel/wsjTRAIN.cnftrees
```

A.7 Depth Bounding

After conversion to Chomsky Normal Form (Appendix A.6), trees from the WSJ treebank (sections 2-21) are processed to annotate each node with depth and side data. Trees which require more than 4 memory elements in the HHMM are discarded from training (the `grep` command below). In the interest of maximal reproducibility, the script that performs this processing step, `scripts/cnftrees2cedepths.rb`, is listed in Appendix B.8.

```
cat genmodel/wsjTRAIN.cnftrees                    \
          | ruby scripts/cnftrees2cedepths.rb     \
          | grep -v '\^R,5'                       \
          > genmodel/wsjTRAIN-tdepth.cnftrees
```

A.8 Relative Frequency Estimation

After final preprocessing (Appendix A.7), trees from the WSJ treebank (sections 2-21) are used in conjunction with Equation 3.1 and Equation 3.2 to estimate a PCFG. In the interest of maximal reproducibility, the scripts that perform this step, `scripts/trees2rules.pl` and

```
Z b = 0.15
Z c = 0.23
  ⋮
Z : q = 0.01
```

Figure A.1: ModelBlocks file format for representing prior probability models.

```
Xy a : b c = 0.27
Xy d : b c = 0.73
  ⋮
Xy q : e f = 1.0
```

Figure A.2: ModelBlocks file format for representing conditional probability models.

`scripts/relfreq.pl`, are listed in Appendix B.9. A sample of the resulting file is shown in Figure 4.4.

```
cat genmodel/wsjTRAIN-tdepth.cnftrees          \
          | perl scripts/trees2rules.pl -p     \
          | perl scripts/relfreq.pl -f         \
          > genmodel/wsjTRAIN-tdepth.pw-cc.counts
```

The preprocessing scripts and HHMM implementation used in this work are all components of ModelBlocks, an open source probability modelling toolkit originally implemented by the members of the University of Minnesota Natural Language Processing group.[4] ModelBlocks defines a common file format for representing probability models. This file format is illustrated in Figures A.1 and A.2.

ModelBlocks files can define both prior probability models and conditional probability models. A line for a prior probability model consists of $M\ \alpha = p$. In the first line of Figure A.1, the modelled variable is Z, Z=b, and $\mathsf{P}_{\theta_Z}(b) = 0.15$.

Similarly, a line for a conditional probability model consists of $Cm\ \alpha : \gamma = p$. Cm is the name of the model; by convention this name is based on the conditioning variable C and the modelled variable M. α is a value for the modelled variable. γ is a value for the conditioning

[4] The code for ModelBlocks is hosted at `http://www.sourceforge.net/projects/modelblocks`

```
cat genmodel/wsjTRAIN-tdepth.pw-cc.counts       \
            | python scripts/calc-cfp-hhmm.py   \
            | perl scripts/sortbyprob.pl        \
            > genmodel/wsjTRAIN-tdepth.pwdt-cfp.model

cat genmodel/wsjTRAIN-tdepth.dt.model           \
          >> genmodel/wsjTRAIN-tdepth.pwdt-cfp.model
```

Figure A.3: Given a model of a PCFG in ModelBlocks file format (with counts instead of probabilities) as specified in Section 4.2 we transform this model into an equivalent right-corner model using the model transform of Schuler (2009). In the interest of maximal reproducibility, the script that performs this step, `scripts/calc-cfp-hhmm.py`, is listed in Appendix B.10

variable. p is a number between zero and one, representing $\mathsf{P}_{\theta_{CM}}(m \mid c)$. In the first line of Figure A.2, the conditioning variable is X, the modelled variable is Y, Y=a, X=b c, and $\mathsf{P}_{\theta_{XY}}(a \mid bc) = 0.27$.

A.9 Part of Speech Model

Given a model of a PCFG in ModelBlocks file format (with counts instead of probabilities) as specified in Section 4.2 and Appendix A.8, we estimate a part of speech model.

```
cat genmodel/wsjTRAIN-tdepth.pw-cc.counts       \
            | sed 's/\.0*$//g'                  \
            | grep '^Pw .* = [1-5]$'            \
            | bin/postrainer                    \
            > genmodel/wsjTRAIN-tdepth.dt.model
```

A.10 Transformed Component Models

Given a model of a PCFG in ModelBlocks file format (with counts instead of probabilities) as specified in Section 4.2 (and shown in Figure 4.4) we transform this model into an equivalent right-corner model using the model transform of Schuler (2009). The command to run this transform are illustrated in Figure A.3, with the script listed in Appendix B.10.

Appendix B

Syntactic Language Model Training Scripts

B.1 scripts/tbtrees2linetrees.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################

#!/usr/bin/perl

use strict;
my $num_parens=0;
my $tree = "";

while(<STDIN>){
  chomp;
  ## Get rid of leading spaces
  s/^\s*//;

  if($_ eq ""){
    next;
```

```
  }

#  s/PRT\|//g;

#  ## Get rid of the dash in *-UNF categories
#  s/-UNF/UNF/g;
#  ## Get rid of everything after the dash (eg. VP-PRD but not -DFL-)
#  s/(\([^ -]+)-[^ ]+/\1/g;
#  ## Escape some difficult terminal symbols to non-word letter combinations
#  s/\\\]/SUBSTITUTERIGHTBRACKET/g;

##  s/\\\[/SUBSTITUTELEFTBRACKET/g;
##  s/\\\+/SUBSTITUTEPLUS/g;
##  s/\.\)/SUBSTITUTEDOT)/g;

##  # Eliminate empty categories (and the unary branch above them) and change right-recursive branch above that to
      reflect deletion
##  s/\(([^ ]+) \(-NONE\- [^ ]+\) \)//;
##  # Eliminate all remaining empty categories (without unary branches above)
##  s/\(-NONE\- [^ ]+\)//;

#  ## Remove cues that give away edited-ness
#  s/\(RM \(-DFL\- \\\[\) \)//g;
#  s/\(IP \(-DFL\- \\\+\) \)//g;
#  s/\(-DFL- E_S\)//g;
#  s/\(-DFL- N_S\)//g;
#  s/\(RS \(-DFL\- SUBSTITUTERIGHTBRACKET\) \)//g;

#  # Remove punctuation
#  s/\(\. \.\)//g;
#  s/\(\. \?\)//g;
#  s/\(, ,\)//g;

  # Accumulate tree lines until parens match
  $tree .= "$_";
  my @left_parens = m/(\(/g;
  my @right_parens = m/(\))/g;
  $num_parens += ($#left_parens + 1);
  $num_parens -= ($#right_parens + 1);
  if($num_parens == 0){
    ## Get rid of extra parentheses around the tree
    $tree =~ s/^\s*\(\s*\((.*)\s*\)\s*\)\s*$/(\1)/;
##    print substr($tree, 2, length($tree) - 4) . "\n"; # "$tree\n";

#    ## Now that tree is on one line, eliminate / push upward many kinds of empty category crapola...
#    $tree =~ s/\(([^ ]+) +\(([^ ]+) +\(-NONE\- +[^ ]+ *\) *\)/\(\1startingwithempty\2/g;
#    $tree =~ s/\(([^ ]+) +\(-NONE\- +[^ ]+ *\) *\)/\(empty\1 -NONE-\) /g;
#    $tree =~ s/\(([^ ]+) +\(-NONE\- +[^ ]+ *\)/\(\1startingwithempty/g;
#    $tree =~ s/\(([^ ]+)startingwithempty[^ ]* +\(empty[^ ]+ +\-NONE\- *\) *\)/\(empty\1 -NONE-\)/g;
#    $tree =~ s/\(-NONE\- [^ ]+ *\)/\(emptysomething -NONE-\)/g;

##    $tree =~ s/ +\(([^ ]+) +\(-NONE\- +[^ ]+\) *\)/endingwithempty\1/g;

    print "$tree\n";
    $tree = "";
  }
#  print $_;
}
```

B.2 scripts/annotateFixes.pl

```
################################################################################
##                                                                            ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                            ##
##    ModelBlocks is free software: you can redistribute it and/or modify     ##
##    it under the terms of the GNU General Public License as published by    ##
##    the Free Software Foundation, either version 3 of the License, or       ##
##    (at your option) any later version.                                     ##
##                                                                            ##
##    ModelBlocks is distributed in the hope that it will be useful,          ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of          ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           ##
##    GNU General Public License for more details.                            ##
##                                                                            ##
##    You should have received a copy of the GNU General Public License       ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.    ##
##                                                                            ##
################################################################################


# cat wsj_0001.trees | perl scripts/treesed.pl
use Getopt::Std;

getopts("d");

$DEBUG = 0;
if ($opt_d) {
  $DEBUG = 1;
}

sub debug {
  if ($DEBUG) {
    $msg = $_[1];
    print stderr $_[0] , "␣" , $msg, "\n";
  }
}

# luan: commented out to simplify b/c now model role as '-1' tag
#$SRL = "\w+\!ldelim\!";

## for each tree...
while ( <> ) {

  ## trace errors...
  # line 1628:
  s/\(WHNP \(WDT which\) \) \(S  \(NP-SBJ-1 \(DT the\) \(JJ Soviet\) \(NN leader\) \)/\(WHNP-1 \(WDT which\) \) \(S
        \(NP-SBJ \(DT the\) \(JJ Soviet\) \(NN leader\) \)/;
  # line 3642:
  s/\(WHNP \(IN that\) \) \(S  \(NP-SBJ-1 \(DT the\) \(NNS Japanese\) \) \(VP \(MD might\)  \(VP \(VB want\)  \(S
       \(NP-SBJ \(-NONE- \*-1\) \) \(VP \(TO to\)  \(VP \(VB buy\)/\(WHNP-2 \(IN that\) \) \(S  \(NP-SBJ-1 \(DT
       the\) \(NNS Japanese\) \) \(VP \(MD might\)  \(VP \(VB want\)  \(S  \(NP-SBJ \(-NONE- \*-1\) \) \(VP \(TO
       to\)  \(VP \(VB buy\)  \(NP \(-NONE- \*T\*-2\) \)/;
  # line 5642:
  s/\(WHNP-1 \(WDT which\) \) \(S  \(NP-SBJ \(NNP Bush\) \) \(VP \(VBD allowed\)  \(S  \(NP-SBJ \(-NONE- \*-1\)/\(
       WHNP-1 \(WDT which\) \) \(S  \(NP-SBJ \(NNP Bush\) \) \(VP \(VBD allowed\)  \(S  \(NP-SBJ \(-NONE- \*T
       \*-1\)/;
  # line 7797:
  s/\(NP \(DT The\) \(JJ same\) \(NN evening\) \) \(SBAR  \(WHNP-4 \(IN that\) \)/\(NP \(DT The\) \(JJ same\) \(NN
       evening\) \) \(SBAR  \(WHPP-4 \(IN that\) \)/;
```

```
# line 17204:
s/\(WHNP-3 \(WDT which\) \) \(S  \(NP-SBJ \(JJ many\) \(NNS investors\) \) \(VP \(VBP say\)  \(SBAR \(-NONE- 0\)
      \(S  \(NP-SBJ-1 \(-NONE- \*-3\)/\(WHNP-3 \(WDT which\) \) \(S  \(NP-SBJ \(JJ many\) \(NNS investors\) \)
     \(VP \(VBP say\)  \(SBAR \(-NONE- 0\)  \(S  \(NP-SBJ-1 \(-NONE- \*T\*-3\)/;
# line 17563:
s/\(WHNP \(WP what\) \) \(S  \(NP-SBJ \(JJ other\) \(JJ prosecutorial\) \(NNS abuses\) \) \(VP \(MD may\)  \(VP
     \(VB have\)  \(VP \(VBN occurred\)/\(WHNP-1 \(WP what\) \(JJ other\) \(JJ prosecutorial\) \(NNS abuses\) \)
      \(S \(NP-SBJ \(-NONE- \*T\*-1\) \) \(VP \(MD may\)  \(VP \(VB have\)  \(VP \(VBN occurred\)/;
# line 25018:
s/\(NP  \(NP \(DT a\) \(NN year\) \) \(SBAR  \(WHNP-1 \(IN that\) \)/\(NP  \(NP \(DT a\) \(NN year\) \) \(SBAR
     \(WHPP-1 \(IN that\) \)/;
# line 27938:
s/\(WHNP \(WP what\) \) \(S  \(NP-SBJ \(DT some\) \(NNS economists\) \) \(VP \(VBP call\) /\(WHNP-1 \(WP what\)
     \) \(S  \(NP-SBJ \(DT some\) \(NNS economists\) \) \(VP \(VBP call\) \(NP \(-NONE- \*T\*-1\) \) /;
# line 29529:
s/\(WHNP \(WDT which\) \) \(S  \(NP-SBJ \(PRP it\) \) \(VP \(MD will\)  \(VP \(VB occupy\) /\(WHNP-1 \(WDT which
     \) \) \(S  \(NP-SBJ \(PRP it\) \) \(VP \(MD will\)  \(VP \(VB occupy\) \(NP \(-NONE- \*T\*-1\) \) /;
# line 36352:
s/\(WHNP \(IN that\) \) \(S  \(NP-SBJ \(DT the\) \(NN government\) \) \(VP \(VBD feared\)  \(SBAR  \(WHNP-1 \(-
     NONE- 0\) \)/\(WHNP-1 \(IN that\) \) \(S  \(NP-SBJ \(DT the\) \(NN government\) \) \(VP \(VBD feared\)  \(
     SBAR  \(WHNP \(-NONE- 0\) \)/;
# line 36736:
s/\(WHNP \(IN that\) \) \(S  \(NP-SBJ \(PRP he\) \) \(VP \(VBD estimated\)  \(SBAR  \(WHNP-2 \(-NONE- 0\) \)/\(
     WHNP-2 \(IN that\) \) \(S  \(NP-SBJ \(PRP he\) \) \(VP \(VBD estimated\)  \(SBAR  \(WHNP \(-NONE- 0\) \)/;
# line 37010:
s/\(WHNP \(IN that\) \) \(S  \(NP-SBJ \(DT some\) \(NNS analysts\) \) \(VP \(VBP say\)  \(SBAR  \(WHNP-1 \(-NONE-
      0\) \) /\(WHNP-1 \(IN that\) \) \(S  \(NP-SBJ \(DT some\) \(NNS analysts\) \) \(VP \(VBP say\)  \(SBAR  \(
     WHNP \(-NONE- 0\) \) /;
# line 37777:
s/\(SBAR  \(WHNP-1 \(WDT which\) \) \(S  \(NP-SBJ-2 \(-NONE- \*-1\) \)/\(SBAR  \(WHNP-1 \(WDT which\) \) \(S  \(
     NP-SBJ-2 \(-NONE- \*T\*-1\) \)/;
# line 39576:
s/\(WHNP \(WDT which\) \) \(S  \(NP-SBJ \(PRP they\) \) \(VP \(VBP sell\) /\(WHNP-1 \(WDT which\) \) \(S  \(NP-
     SBJ \(PRP they\) \) \(VP \(VBP sell\) \(NP \(-NONE- \*T\*-1\) \)/;

## VBN errors -> VBD...
# line 41:
s/\(, ,\)  \(VP \(VBN reached\)  \(NP \(-NONE- \*T\*-/\(, ,\)  \(VP \(VBD reached\)  \(NP \(-NONE- \*T\*-/;
# line 4257:
s/2\) \) \(VP \(VBN forced/2\) \) \(VP \(VBN forced/;
# line 6124:
s/\(VP \(VBN suspended\)  \(NP \(NN stock-index\)/\(VP \(VBD suspended\)  \(NP \(NN stock-index\)/;
# line 6815:
s/\(VP \(VBN involved\)  \(NP \(JJ critical\)/\(VP \(VBD involved\)  \(NP \(JJ critical\)/;
# line 7676:
s/\(NN week\) \) \(VP \(VBN succeeded\)/\(NN week\) \) \(VP \(VBD succeeded\)/;
# line 10076:
s/\(NNS completions\) \) \(VP \(VBN lagged\)/\(NNS completions\) \) \(VP \(VBD lagged\)/;
# line 10572:
s/\(RB ago\) \) \(VBN acquired\)/\(RB ago\) \) \(VBD acquired\)/;
# line 11963:
s/\(, ,\) \) \(VP \(VBN announced\)/\(, ,\) \) \(VP \(VBD announced\)/;
# line 12946:
s/3\) \) \(VP \(VBN generated/3\) \) \(VP \(VBD generated/;
# line 13609:
s/NNS researchers\) \) \(VP \(VBN agreed/NNS researchers\) \) \(VP \(VBD agreed/;
# line 15066:
s/1\) \) \(VP \(VBN started/1\) \) \(VP \(VBD started/;
# line 15591, 28900:
s/1\) \) \(VP \(VBN changed/1\) \) \(VP \(VBD changed/;
# line 15961:
s/1\) \) \(VP \(VBN opened/1\) \) \(VP \(VBD opened/;
```

```
# line 18772:
s/NNP KKR\) \) \(VP \(VBN restructured/NNP KKR\) \) \(VP \(VBD restructured/;
# line 19265:
s/1\) \) \(VP \(VBN included/1\) \) \(VP \(VBD included/;
# line 19680:
s/VBN become\)  \(NP-PRD \(JJ overnight/VBD become\)  \(NP-PRD \(JJ overnight/;
# line 19748:
s/NN bank\) \) \(VP \(VBN disclosed\)/NN bank\) \) \(VP \(VBD disclosed\)/;
# line 21201:
s/NP Goodson\) \) \(VP \(VBN bought/NP Goodson\) \) \(VP \(VBD bought/;
# line 23901:
s/2\) \) \(VP \(VBN opened/2\) \) \(VP \(VBD opened/;
# line 24689:
s/1\) \) \(VP \(VBN traded/1\) \) \(VP \(VBD traded/;
# line 24700:
s/3\) \) \(VP \(VBN stretched/3\) \) \(VP \(VBD stretched/;
# line 26286:
s/VBN set\)  \(NP \(NNS plans/VBD set\)  \(NP \(NNS plans/;
# line 28510:
s/1\) \) \(VP \(VBN made/1\) \) \(VP \(VBD made/;
# line 32752:
s/NNS eggs\) \) \(VP \(VBN come/NNS eggs\) \) \(VP \(VBD come/;
# line 33883:
s/NNP II\) \) \(VP \(VBN ended/NNP II\) \) \(VP \(VBD ended/;
# line 38619:
s/1\) \) \(VP \(VBN set\)  \(PRT \(RP off/1\) \) \(VP \(VBD set\)  \(PRT \(RP off/;
# line 38773:
s/NN company\) \) \(VP \(VBN valued/NN company\) \) \(VP \(VBD valued/;
# line 38814:
s/1\) \) \(VP \(VBN induced/1\) \) \(VP \(VBD induced/;
# line 39070:
s/NN market\) \) \(VP \(VBN stabilized/NN market\) \) \(VP \(VBD stabilized/;
# line 39264:
s/PRP it\) \) \(VP \(VBN opened/PRP it\) \) \(VP \(VBD opened/;
# line 39579:
s/NN analyst\) \) \(VP \(VBN estimated/NN analyst\) \) \(VP \(VBD estimated/;
# line 39619:
s/RB finally\) \) \(VP \(VBN opened/RB finally\) \) \(VP \(VBD opened/;
# line 39623:
s/NN market\) \) \(VP \(VBN opened/NN market\) \) \(VP \(VBD opened/;
# line 39779:
s/RB never\) \) \(VBN reopened/RB never\) \) \(VBD reopened/;

# NN errors -> VB/VBZ
s/\(VP  *\(NNS  *([^\)]*)\)/\(VP \(VBZ \1\)/g;
s/\(VP  *\(NN  *([^\)]*)\)/\(VP \(VB \1\)/g;


# probably fine, but not the problem...
# s/((?: be| being| been| is| was|VBZ 's| are| were| 're)[ \)]*[^ \)]*)\(VBD [^ ]*ed\)/\1\(VBNmodified [^ ]*ed\)/
    g;
# probably fine, but not the problem...
# s/\(VBD ([^\)]*)\) *\(NP *\(-NONE- \*\) \)/\(VBN \1\)  \(NP \(-NONE- \*\) \)/g;

## translate to parens...
s/\[/\(/g;
s/\]/\)/g;
## for each constituent...
while ( $_ =~ /\([^\(\)]*\)/ ) {
  ## convert outer parens to braces...
  s/\(([^\(\)]*)\)/{\1}/;
  ################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
```

```
    debug($step++, "___$_");

    # line 24513:
    s/{(SBAR) *<(WHNP-1 [^>]*)> *<(S) ([^>]*)> <(VP [^>]*)> *}/{\1 <\2> <\3 \4 [\5]>}/;

#    # VBN errors:
#    # any vb[dgn] projection following BE-verb is AP
#    s/((?: be| being| been| is| was|VBZvbz 's| are| were| 're)[ \)>\]]*)\{(VPvb[ndg]|VB[NDG]vb[ndg])(.*)\}/\1\{AP
     \3\}/;
#    # any vb[dgn] projection post-modifying NP
#    s/{(NP.*)<(NP)([^>]*)> +<(Sprovb[ndg]|VPvb[ndg]|VB[NDG]vb[ndg])([^>]*)>(.*)}/{\1<\2\3> <AP\5>\6}/;

    ####################
    ## convert inner angles (if any) to bracks...
    while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}
    ## convert outer braces to angles...
    s/{(.*)}/<\1>/;
  }

  ## finish up...
  s/</[/;
  s/>/]/;
  ## translate to parens again...
  s/\[/\(/g;
  s/\]/\)/g;
  ## output...
  print $_;
}
```

B.3 scripts/annotateProjs.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################


# cat wsj_0001.trees | perl scripts/treesed.pl
use Getopt::Std;


getopts("pd");
```

```
$remove_punct = 0;
if($opt_p){
  $remove_punct = 1;
}

$DEBUG = 0;
if ($opt_d) {
  $DEBUG = 1;
}

sub debug {
  if ($DEBUG) {
    $msg = $_[1];
    print stderr $_[0] , "␣" , $msg, "\n";
  }
}

$SRL = "!colon![^>]*";

## for each tree...
$lineNum=1;
while ( <> ) {
  debug("***TB***␣line␣", $lineNum++);

  # moved from tbtrees2linetrees.pl
  s/PRT\|//g;

  ## Make parentheses look standard...
  s/\( */(/g;
  s/ *\)/)/g;

  ## Remove repair annotation
  s/\(RM \(\-DFL\- \\\[\) \)//g;
  s/\(IP \(\-DFL\- \\\+\) \)//g;
  s/\(\-DFL\- E_S\)//g;
  s/\(\-DFL\- N_S\)//g;
  s/\(RS \(\-DFL\- \\\]\) \)//g;
  ## Sometimes that creates empty categories (especially in mis-annotated data)
  ## Remove those empty categories
  s/\([^ ]+[ ]+\)//;

  ## Collapse out everything between cat. and UNF
  s/\(([^ ]+) -[^ ]+-UNF/(\1-UNF/g;
  s/-UNF/UNF/g;
  # We wil undo this shift later, but for right now it's done so INTJ is not
  # confused with IN (I think 4 underscores should be enough)
  s/INTJ/____INTJ/g;

  ## translate to parens...
  s/\[/\(/g;
  s/\]/\)/g;

  if($remove_punct == 1){
      # Remove/change punctuation...
      s/\([^ ]+ \.\.\.\)//g;
      s/\([^ ]+ \.\)//g;
      s/\([^ ]+ \!\)//g;
      s/\([^ ]+ \?\)//g;
      s/\([^ ]+ \,\)//g;
      s/\([^ ]+ *\([^ ]+ *\-\-\) *\)//g;  ## dash is nested... don't know why
      s/\([^ ]+ \-\-\)//g;
      s/\([^ ]+ \-\)//g;
```

```
        s/\([^ ]+ \;\)//g;
        s/\([^ ]+ \:\)//g;
        s/\([^ ]+ \‘\)//g;
        s/\([^ ]+ \’\)//g;
␣␣␣␣␣␣s/\([^␣]+␣\‘\‘\)//g;
␣␣␣␣␣␣s/\([^␣]+␣\’\’\)//g;
␣␣␣␣␣␣s/\([^␣]+␣–L.B–\)//g;
␣␣␣␣␣␣s/\([^␣]+␣–R.B–\)//g;
␣␣}

␣␣#␣Get␣rid␣of␣typos␣(categories␣starting␣with␣^)
␣␣s/\(\^/\(/g;

␣␣#␣Distinguish␣punctuation...
␣␣s/\([^␣]+␣\.\.\.(:[^\)]*)?\)/\(\.\.\.␣\.\.\.\1\)/g;
␣␣s/\([^␣]+␣\?(:[^\)]*)?\)/\(\?␣\?\1\)/g;
␣␣s/\([^␣]+␣*\([^␣]+␣*\–\–(:[^\)]*)?\)␣*\)/\(\–\–␣\–\–\1\)/g;␣␣##␣dash␣is␣nested...␣don't know why
  s/\([^ ]+ \–\–(:[^\)]*)?\)/\(\–\– \–\–\1\)/g;
  s/\([^ ]+ \–(:[^\)]*)?\)/\(\–\– \–\–\1\)/g;
  s/\([^ ]+ \:(:[^\)]*)?\)/\(\: \:\1\)/g;
  s/\([^ ]+ \;(:[^\)]*)?\)/\(\; \;\1\)/g;
  s/\([^ ]+ \‘(:[^\)]*)?\)/\(\‘ \‘\1\)/g;
  s/\([^ ]+ \’(:[^\)]*)?\)/\(\’␣\’\1\)/g;

␣␣##########␣GLOBAL␣SYMBOL␣SHIFT
␣␣#s/\!/\!exclamation\!/g;
␣␣s/\~/\!tilde\!/g;
␣␣#s/\‘\‘/\!openquote\!/g;
␣␣#s/\‘/\!openscare\!/g;
␣␣s/\@/\!at\!/g;
␣␣s/\#/\!pound\!/g;
␣␣#s/\$/\!dollar\!/g;
␣␣#s/\%/\!percent\!/g;
␣␣#s/\^/\!carat\!/g;
␣␣s/\&/\!ampersand\!/g;
␣␣#s/\*/\!star\!/g;
␣␣s/\–\–/\!dash\!/g;
␣␣s/\+/\!plus\!/g;
␣␣#s/\=/\!equals\!/g;
␣␣s/\:/\!colon\!/g;
␣␣s/\;/\!semi\!/g;
␣␣#s/\”/\!dblquote\!/g;
␣␣#s/\’\’/\!closequote\!/g;
␣␣#s/\’/\!closescare\!/g;
␣␣#s/\./\!period\!/g;
␣␣#s/\,/\!comma\!/g;
␣␣#s/([^\*])\?/\1\!question\!/g;
␣␣s/\//\!slash\!/g;
␣␣##########

␣␣#␣substitute␣–LRB–␣to␣\!LRB\!
␣␣s/–L.B–/\!LRB\!/g;
␣␣#␣substitute␣–RRB–␣to␣\!RRB\!
␣␣s/–R.B–/\!RRB\!/g;
␣␣#␣substitute␣–NONE–␣to␣\*NONE\*
␣␣s/–NONE–/\*NONE\*/g;
␣␣#␣substitute␣NP–TMP␣to␣NP–tmp
␣␣s/NP[^␣]*\–TMP/NP–tmp/g;
␣␣#␣substitute␣S*–ADV␣to␣S*–adv
␣␣s/(S[^␣]*)\–ADV/\1–adv/g;
␣␣s/(S[^␣]*)\–PRP/\1–adv/g;
␣␣s/(S[^␣]*)\–TMP/\1–adv/g;
```

```
  # substitute SBAR-NOM,SBAR-PRD to NP
  s/SBAR-NOM/NP/g;
  s/SBAR-PRD/NP/g;
  # remove all other dash specifications beginning with capital letter
  s/\-[A-Z][A-Z]*([^ \)\]]*) /\1 /g;
  s/\-[A-Z][A-Z]*([^ \)\]]*) /\1 /g;
  s/\-[A-Z][A-Z]*([^ \)\]]*) /\1 /g;


  ## for each constituent ...
  $step = 0;
  while ( $_ =~ /\([^\(\)]*\)/ ) {
    ## convert outer parens to braces ...
    $_ =~ s/\(([^\(\)]*)\)/{\1}/;
    ################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent) ...
    debug(++$step, " $_");

    #### EOS PUNCT
    # kick eos punct up out of constit (should have been annotated this way!)
    s/{(?![^ ]*-mE)([^ ]*) +(.*) +(<[\.\!\?] [\.\!\?]>)}(?=.*\))/{\1 \2} \3/;
    # kick eos punct + right quote up out of constit (should have been annotated this way -- quote delimits can be obtained from left, but punct is needed at root)
    s/{(?![^ ]*-mE)(?![^ ]*-nQ)([^ ]*) +(.*) +(<``?``?>)(.*) +(<[\.\!\?] [\.\!\?]> +<''?''?>)}(?=.*\))/{\1-nQ-mQ \2\3\4} \5/;
    # kick eos punct + right quote up out of constit (should have been annotated this way -- quote delimits can be obtained from left, but punct is needed at root)
    s/{(?![^ ]*-mE)(?![^ ]*-nQ)([^ ]*) +(.*) +(<[\.\!\?] [\.\!\?]> +<''?''?>)}(?=.*\))/{\1-nQ \2} \3/;
    # isolate quote + ... + period + quote (allows other junk after, like rest of line in case of embedded quote)
    s/{(?![^ ]*-mE)([^ ]*) +(<``?``?>) +(<.*) +(<[\.\!\?] [^>]*>) +(<''?''?>.*)}/\(\1-mE-mQ \(\1-nQ-mE-mQ \2 \{\1 \3\} \4\) \5\)/;
    # isolate ... + period + quote (allows other junk after, like rest of line in case of embedded quote)
    s/{(?![^ ]*-mE)([^ ]*) +(<.*) +(<[\.\!\?] [^>]*>) +(<''?''?>.*)}/\(\1-mE \(\1-nQ-mE \{\1 \2\} \3\) \4\)/;
    # isolate period after S (allows other junk after period, like quotes or rest of line in case of embedded quote)
    s/{(?![^ ]*-mE)([^ ]*) +(<.*) +(<[\.\!\?] [^>]*>)(.*)}/\(\1-mE \{\1 \2\} \3\4\)/;

    #### EMPTY CATEGORIES
    # delete ICH (moved -- not just raised -- modifier) and U (transposed currency symbol) traces; too hard to reconstruct
    s/{\*NONE\* .*ICH.*}//;
    s/{\*NONE\* *\*U\*(!colon!.*)? *}//;
    # eliminate expletive it cleft (a shame, but sent is too deeply attached)
    s/<NP[^ ]* +\[(NP[^ ]*)\] \[[^ ]*-expl-([0-9]*)_0> \((VP[^ ]*) (.*) \{([^ ]*)-\2 (.*)\}([ \)]*)\)/<\1> \(\3-aIT \4\7\) \{\5\6\}/;
    s/{(.*)<NP[^ ]* +\[(NP[^>]*)\] +\[S[^ ]*-empty-([0-9]*)_0\]>(.*)\[\]/{\1<\2>\3
    s/{NP[^ ]* +<(NP[^>]*)>+<S[^ ]*-empty_0($SRL)?> *}/{\1}/;
    # delete unary projections of deleted empty constituents
    s/{[^ ]* *}//;
    # fold empty category for currency into nonterm
    s/{QP([^ ]*) +(.*<\!pound\!.*)}/{NP-currunit \1 \2}/;
    s/{QP([^ ]*) +(.*<\$.*)}/{NP-currunit \1 \2}/;
    s/{QP([^ ]*) +(.*<\%.*)}/{NP-currunit \1 \2}/;
    s/{(NP[^ ]*|ADJP[^ ]*) (.*)<NP-currunit[^ ]* ([^>]*)>*<\*NONE\* [^>]*>([^}]*)}/{\1 \2 \3\4}/; ## probably redundant now with *U* removed
    # fold empty X categories into X-empty nonterm
    s/{([^ ]*) +<\*NONE\* +\*T\*\-([0-9]+)($SRL)?> *}/{\1-empty-\2_0\3}/;
    debug($step, " $_");
    s/{([^ ]*) +<\*NONE\* +\*EXP\*\-([0-9]+)($SRL)?> *}/{\1-expl-\2_0\3}/;
    s/{([^ ]*) +<\*NONE\* +\*RNR\*\-([0-9]+)($SRL)?> *}/{\1-empty-\2_0\3}/;
    s/{([^ ]*) +<\*NONE\* +\*($SRL)?> *}/{\1-empty-ctrl_0\2}/;
    if ( $_ =~ /\!colon\!(REL|rel)/ ) { # use this to check if input is srl tree
      s/{([^ ]*) +<\*NONE\* +[^>]*($SRL)> *}/{\1-empty_0\2}/;
```

```
    }␣else␣{
␣␣␣␣␣␣␣␣s/{([^␣]*)␣+<\*NONE\*␣+[^>]*>␣*}/{\1−empty␣0};
    }
    #␣fold␣NP␣trace␣in␣VP␣into␣passive␣VP␣nonterm
    s/{(VP[^␣]*|VBN[^␣]*|VBD[^␣]*)␣+(.*)<(VB[ND](?![^␣]*−v)[^␣]*)␣+([^>]*)>␣*<NP[^␣]*−empty(−ctrl)?␣0($SRL)?>(.*)
        }/{\1␣\2<\3−v␣\4>␣\7};
    #␣set␣passive␣for␣VBN␣following␣BE−verb
    s/((?:␣be|␣being|␣been|␣is|␣was|VBZvbz␣'s|␣are|␣were|␣'re)[␣\)>\]]*[^\)>\]]*)<((?:VB[ND]|VPvb[nd])(?![^␣]*−v)[^
        ␣]*)␣([^␣]*)␣*>([^\{]*\})/\1<\2−v␣\3>\4/;
    ##␣s/{.*␣(?:be|being|been|is|was|are|were)[␣\)>\]]*[^\)>\]]*)<(VB[ND](?![^␣]*−v)[^␣]*)␣([^␣]*)␣*>(.*)}/{\1<\2−
        v␣\3>\4};
    #␣set␣passive␣for␣VBN␣following␣BE−verb␣+␣adverb
    s/((?:␣be|␣being|␣been|␣is|␣was|VBZvbz␣'s|␣are|␣were|␣'re)[␣\)>\]]*[^\)>\]]*<ADVP[^␣]*␣+[^>]*>[␣\)>\]]*[^\)
        >\]]*)<((?:VB[ND]|VPvb[nd])(?![^␣]*−v)[^␣]*)␣([^␣]*)␣*>([^\{]*\})/\1<\2−v␣\3>\4/;
    ##␣s/{.*␣(?:be|being|been|is|was|are|were)[␣\)>\]]*[^\)>\]]*<ADVP[^␣]*␣+[^>]*>[␣\)>\]]*[^\)>\]]*)<(VB[ND](?![^
        ␣]*−v)[^␣]*)␣([^␣]*)␣*>(.*)}/{\1<\2−v␣\3>\4};
    #␣set␣passive␣for␣VBN␣adjectives
    s/{(N.*)<(VB[ND](?![^␣]*−v)[^␣]*)([^>]*)>␣+<(NN[^␣]*)([^>]*)>(.*)}/{\1<\2−v\3>␣<\4\5>\6};
    #␣set␣passive␣for␣VBN␣adjectives
    s/{(ADJP.*)<(RB[^␣]*)([^>]*)>␣+<(VB[ND](?![^␣]*−v)[^␣]*)([^>]*)>(.*)}/{\1<\2\3>␣<\4−v\5>\6};
    #␣set␣passive␣for␣VBN␣adverbial␣clauses
    s/(\(S[^␣]*−adv[^␣]*␣+<NP[^>]*−empty[^>]*>␣+\{VP[^␣]*␣+)<(VB[ND](?![^␣]*−v)[^␣]*)([^>]*)>/\1<\2−v\3>/;
    #␣shift␣passive␣down␣if␣not␣at␣leaf
    s/{([^␣]*)−v␣+<(VB[ND][^␣]*)␣([^>]*)>(.*)}/{\1␣<\2−v␣\3>\4};
    #␣propagate␣passive␣tag␣up␣unary
    s/{([^␣]*)␣+<([^␣]*)−v␣([^␣]*)>␣*}/{\2−v␣\3};
    #␣fold␣empty␣NP␣in␣S␣with␣VPvb|VPvbp␣into␣S␣imperative
    s/{(S[^␣]*|PRN[^␣]*)␣+(.*)<[^␣]*NP[^␣]*−empty−ctrl␣0($SRL)?>␣*<(VPvbp|VPvb)([^a−z][^>]*)>(.*)}/(\1␣\2\{Simp
        \5\}\6);
    #␣fold␣empty␣NP␣in␣S␣with␣VPvbg␣into␣S␣pro␣progressive␣/␣passive␣/␣infinitive
    s/{(S[^␣]*)␣+(.*)<[^␣]*NP[^␣]*−empty(−ctrl)?␣0($SRL)?>␣*<VP(vbg|vbn|prd|vbd|vb|to)([^>]*)>(.*)}/(\1␣\2\{Spro
        \5\6\}\7);
    #␣fold␣empty␣NP␣in␣S␣with␣VPto␣into␣S␣pro␣stative
    s/{(S[^␣]*)␣+(.*)<[^␣]*NP[^␣]*−empty(−ctrl)?␣0($SRL)?>␣*(.*)<(ADJP|NP|PP)([^>]*)>(.*)}/(\1␣\2\5\{Sproprd\7\}\8)
        ;
    #␣fold␣NP␣trace␣in␣SBAR␣into␣S␣nonterm
    s/{SBAR[^␣\^]*([^␣]*)␣+<\*NONE\*␣+[^>]*>␣*<S([^␣\^]*)[^␣\−]*([^␣]*)␣([^>]*)>␣*}/{S\2\1\3␣\4};
    s/{SBAR[^␣\^]*([^␣]*)␣+<[^␣]*−empty−([0−9]*)␣0($SRL)?>␣*<S([^␣\^]*)[^␣\−]*([^␣]*)␣([^>]*)>␣*}/{S\2\1\5−\4␣\6};
    #␣redo␣SBAR␣as␣WHSBAR
    s/{(SBAR[^␣]*)␣+(<WH.*)␣+(<S.*)}/{WH\1␣\2␣\3};

    #␣turn␣NN␣in␣VP␣into␣VBG␣−−␣weird␣way␣to␣annotate␣gerunds
    s/{(VP[^␣]*)␣<NN[^␣\^]*([^␣]*)␣([^>]*)>(.*)}/(\1␣\{VBGvbg\2␣\3\}\4);

    ####␣TERMINALS
    #␣give␣terminal␣symbols␣lower␣case␣and␣add␣lowercase␣pos␣to␣category
    if␣($_␣=~␣/{(VB[A−Z]*|MD|TO|BES|HVS)[a−z]*([^␣]*)␣*([^␣\/]*)␣*}/)␣{␣$c=$1;␣$u=$2;␣$p=lc($1);␣$w=$3;␣s/{.*}/{
        $c$p$u␣$w};␣}
    #if␣($_␣=~␣/{(VB[A−Z]*|MD|TO|BES|HVS)[a−z]*([^␣]*)␣*([^␣\/]*)␣*}/)␣{␣$c=$1;␣$u=$2;␣$p=lc($1);␣$w=lc($3);␣s
        /{.*}/{$c$p$u␣$w};␣}
    #␣give␣terminal␣symbols␣lower␣case
    if␣($_␣=~␣/{([A−Z\$]*)([^␣]*)␣*([^␣\/]*)␣*}/)␣{␣$c="$1$2";␣$w=$3;␣s/{.*}/{$c␣$w};␣}
    #if␣($_␣=~␣/{([A−Z\$]*)([^␣]*)␣*([^␣\/]*)␣*}/)␣{␣$c="$1$2";␣$w=lc($3);␣s/{.*}/{$c␣$w};␣}
    #␣give␣terminal␣symbols␣lower␣case␣and␣add␣word␣to␣category
    #if␣($_␣=~␣/{(PRT)[a−z]*([^␣]*)␣*([^␣\/]*)␣*}/)␣{␣$c=$1;␣$u=$2;␣$p=lc($1);␣$w=lc($3);␣s/{.*}/{$c$w$u␣$w};␣}
    s/{PRT([^>\]]*)␣([a−z]+)([^<\[]*)}/{PRT\2\1␣\2\3};

    $CONJABLE␣=␣"[^␣]*";
    $STRICTCONJABLE␣=␣"S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*";
    ####␣CONJUNCTION
    #␣XP␣CC␣XP␣−>␣XP␣CXP
    s/{([^␣]*)␣+(.*)<($CONJABLE)␣([^>]*)>␣*<(CC␣[^>]*)>␣*<(\3[^>]*)>(.*)}/\(\1␣\2<\3␣\4>␣\{C\3␣<\5>␣<\6>\}\7\)/;
```

```
␣␣␣␣#␣XP␣XP␣CXP␣␣␣␣␣->␣␣XP␣CXP
␣␣␣␣s/{([^␣]*)␣+(.*)<($CONJABLE)␣([^>]*)>␣*<(\3[^>]*)>␣*<(C\3[^>]*)>(.*)}/\(\1␣\2<\3␣\4>␣\{C\3␣<\5>␣<\6>\}\7\)/;
␣␣␣␣#␣XP␣XP␣XP␣␣␣␣␣␣->␣␣XP␣CXP␣␣at␣end␣of␣constit
␣␣␣␣s/{([^␣]*)␣+(.*)<($STRICTCONJABLE)␣([^>]*)>␣*<(\3[^>]*)>␣*<(\3[^>]*)>(␣*)}/\(\1␣\2<\3␣\4>␣\{C\3␣<\5>␣<\6>\}\7\)
     /;
␣␣␣␣#␣;␣␣CC␣XP␣␣␣␣␣␣->␣␣;␣␣CSXP–pS
␣␣␣␣s/{([^␣]*)␣+(.*)<(\!semi\![^>]*)>␣*<(CC␣[^>]*)>␣*<([^␣]*)␣([^>]*)>(.*)}/\(\1␣\2<\3>␣\{CS\5–pS␣<\4>␣<\5␣
     \6>\}\7\)/;
␣␣␣␣#␣;␣␣XP␣CSXP␣␣␣␣->␣␣;␣␣CSXP–pS
␣␣␣␣s/{([^␣]*)␣+(.*)<(\!semi\![^>]*)>␣*<($CONJABLE)␣([^>]*)>␣*<(CS?\4[^>]*)>(.*)}/\(\1␣\2<\3>␣\{CS\4–pS␣<\4␣\5>␣
     <\6>\}\7\)/;
␣␣␣␣#␣XP␣;␣␣CSXP–ps␣␣->␣␣XP␣CSXP
␣␣␣␣s/{([^␣]*)␣+(.*)<($CONJABLE)␣([^>]*)>␣*<(\!semi\![^>]*)>␣*<(CS\3–pS[^>]*)>(.*)}/\(\1␣\2<\3␣\4>␣\{CS\3␣<\5>␣
     <\6>\}\7\)/;
␣␣␣␣#␣;␣␣XP␣;␣␣XP␣␣␣->␣␣;␣␣XP␣CSXP␣␣at␣end␣of␣constit
␣␣␣␣s/{([^␣]*)␣+(.*)<(\!semi\![^>]*)>␣*<($CONJABLE)␣([^>]*)>␣*<(\!semi\![^>]*)>␣*<(\4[^>]*)>(␣*)}/\(\1␣\2<\3>␣<\4␣
     \5>␣\{CS\4␣<\6>␣<\7>\}\8\)/;
␣␣␣␣#␣,␣␣CC␣XP␣␣␣␣␣␣->␣␣,␣␣CCXP–pC
␣␣␣␣s/{([^␣]*)␣+(.*)<(,[^>]*)>␣*<(CC␣[^>]*)>␣*<([^␣]*)([^>]*)>(.*)}/\(\1␣\2<\3>␣\{CC\5–pC␣<\4>␣<\5\6>\}\7\)/;
␣␣␣␣#␣,␣␣XP␣CCXP␣␣␣␣->␣␣,␣␣CCXP–pC
␣␣␣␣s/{([^␣]*)␣+(.*)<(,[^>]*)>␣*<($CONJABLE)␣([^>]*)>␣*<(CC?\4[^>]*)>(.*)}/\(\1␣\2<\3>␣\{CC\4–pC␣<\4␣\5>␣<\6>\}\7\)
     /;
␣␣␣␣#␣XP␣,␣␣CCXP–ps␣␣->␣␣XP␣CCXP
␣␣␣␣s/{([^␣]*)␣+(.*)<($CONJABLE)␣([^>]*)>␣*<(,[^>]*)>␣*<(CC\3–pC[^>]*)>(.*)}/\(\1␣\2<\3␣\4>␣\{CC\3␣<\5>␣<\6>\}\7\)
     /;
␣␣␣␣#␣,␣␣XP␣,␣␣XP␣␣␣->␣␣,␣␣XP␣CCXP␣␣at␣end␣of␣constit
␣␣␣␣s/{([^␣]*)␣+(.*)<(,[^>]*)>␣*<($CONJABLE)␣([^>]*)>␣*<(,[^>]*)>␣*<(\4[^>]*)>(␣*)}/\(\1␣\2<\3>␣<\4␣\5>␣\{CC\4␣<\6>
     ␣<\7>\}\8\)/;
␣␣␣␣#␣X␣␣CX␣␣␣␣␣->␣␣X␣␣(finalize)
␣␣␣␣s/{(?!C)([^␣]*)␣+(.*)<([^␣]*)␣([^>]*)>␣*<(C[CS]?\3)␣([^>]*)>(.*)}/\(\1␣\2\{\3␣<\3␣\4>␣<\5␣\6>\}\7\)/;
␣␣␣␣#␣X␣␣CC␣␣X␣␣->␣␣X␣␣####(do␣generic␣conj␣first␣,␣as␣more␣specific␣one␣will␣appear␣beside␣it␣and␣ultimately␣
     replace␣it␣via␣"undo␣unary␣identity␣projection")
␣␣␣␣s/{([^␣]*)␣+(.*)<([^␣]*)␣([^>]*)>␣*<(CC)([^>]*)>␣*<\3␣([^>]*)>(.*)}/\(\1␣\2\{\3␣<\3␣\4>␣<C\3␣[\5\6]␣[\3␣
     \7]>\}\8\)/;
#␣␣␣␣#␣XP␣CC␣XP␣␣␣␣␣␣␣->␣␣XP
#␣␣␣␣s/{([^␣]*)␣+(.*)<([^␣]*)␣([^>]*)>␣*<(C\3[^>]*)>(.*)}/\(\1␣\2\{CC\3␣<\3␣\4>␣<\5>\}\6\)/;

#␣␣␣␣␣#␣grab␣XP␣;␣conj␣XP␣as␣LISTXP–semi
#␣␣␣␣␣s/{([^␣]*)␣+(.*)<(S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*)([^>]*)>␣*<(\!semi
     \![^>]*)>␣*<(CC[^>]*)>␣*<\3([^>]*)>(.*)}/\(\1␣\2\{LIST\3–semi␣<\3\4>␣<\5>␣<\6>␣<\3\7>\}\8\)/;
#␣␣␣␣␣#␣grab␣XP␣;␣XP␣;␣XP␣as␣LISTXP–semi␣at␣end␣of␣constit
#␣␣␣␣␣s/{([^␣]*)␣+(.*)<(NP[^␣]*)([^>]*)>␣*<(\!semi\![^>]*)>␣*<\3([^>]*)>␣*<(\!semi\![^>]*)>␣*<\3([^>]*)>(␣*)}/\(\1
     ␣\2\{LIST\3–semi␣<\3\4>␣<\5>␣<LIST\3–semi␣\[\3\6\]␣\[\7\]␣\[\3\8\]>\}\9\)/;
#␣␣␣␣␣#␣grab␣XP␣;␣LISTXP–semi␣as␣LISTXP–semi
#␣␣␣␣␣s/{(?!LIST)([^␣]*)␣+(.*)<(S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*)([^>]*)>␣*<(\!
     semi\![^>]*)>␣*<LIST\3–semi([^>]*)>(.*)}/\(\1␣\2\{LIST\3–semi␣<\3\4>␣<\5>␣<LIST\3–semi\6>\}\7\)/;
#␣␣␣␣␣#␣grab␣XP␣,␣conj␣XP␣as␣LISTXP
#␣␣␣␣␣s/{([^␣]*)␣+(.*)<(S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*)([^>]*)>␣*<(,[^>]*)>␣*<(
     CC[^>]*)>␣*<\3([^>]*)>(.*)}/\(\1␣\2\{LIST\3␣<\3\4>␣<\5>␣<\6>␣<\3\7>\}\8\)/;
#␣␣␣␣␣#␣grab␣XP␣,␣XP␣conj␣XP␣as␣LISTXP
#␣␣␣␣␣s/{([^␣]*)␣+(.*)<(S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*)([^>]*)>␣*<(,[^>]*)>
     ␣*<\3([^>]*)>␣*<(CC[^>]*)>␣*<\3([^>]*)>(.*)}/\(\1␣\2<\3\4>␣<\5>␣\{LIST\3␣<\3\6>␣<\7>␣<\3\8>\}\9\)/;
#␣␣␣␣␣#␣grab␣XP␣,␣XP␣,␣XP␣as␣LISTXP␣at␣end␣of␣constit
#␣␣␣␣␣s/{([^␣]*)␣+(.*)<(NP[^␣]*)([^>]*)>␣*<(S[^␣]*)>␣*<(,[^>]*)>␣*<\3([^>]*)>␣*<(,[^>]*)>␣*<\3([^>]*)>(␣*)}/\(\1␣\2\{LIST\3␣
     <\3\4>␣<\5>␣<LIST\3␣\[\3\6\]␣\[\7\]␣\[\3\8\]>\}\9\)/;
#␣␣␣␣␣#␣grab␣XP␣,␣LISTXP␣as␣LISTXP
#␣␣␣␣␣s/{(?!LIST)([^␣]*)␣+(.*)<(S[^␣]*|NP[^␣]*|VP[^␣]*|PP[^␣]*|ADJP[^␣]*|ADVP[^␣]*|CD[^␣]*)([^>]*)>␣
     *<(,[^>]*)>␣*<LIST\3([^>]*)>(.*)}/\(\1␣\2\{LIST\3␣<\3\4>␣<\5>␣<LIST\3\6>\}\7\)/;
#␣␣␣␣␣#␣grab␣XP␣XP␣conj␣XP␣as␣XP␣LISTXP
#␣␣␣␣␣s/{(SBAR|SQ|SINV|S|NP|VP|PP|ADJP|ADVP|CD)([^␣]*)␣+(.*)<\1([^␣]*)␣([^>]*)>␣*<\1\4␣([^>]*)>␣*<(CC[^>]*)>␣*<\1\4
     ␣([^>]*)>(.*)}/\(\1\4␣\3<\1\4␣\5>␣\{LIST\1\4␣<\1\4␣\6>␣<\7>␣<\1\4␣\8>\}\9\)/;
#␣␣␣␣␣#␣grab␣XP␣XP␣XP␣as␣LISTXP␣at␣end␣of␣constit
```

```
#     s/{([^ ]*) +(.*)<(NP[^ ]*) ([^>]*)>*<\3 ([^>]*)>*<\3 ([^>]*)>(.*)}/\(\1 \2<\3 \4>\{LIST\3 <\3 \5> <\3
      \6>\}\7\)/;
#     #grab XP XP LISTXP as XP LISTXP
#     s/{(?!LIST)([^ ]*) +(.*)<(S[^ ]*|NP[^ ]*|VP[^ ]*|PP[^ ]*|ADJP[^ ]*|ADVP[^ ]*|CD[^ ]*)([^>]*)>*<\3([^>]*)>*<
      LIST\3([^>]*)>(.*)}/\(\1 \2<\3\4>\{LIST\3 <\3\5>  <LIST\3\6>\}\7\)/;
#     #grab X conj X as X #### (do generic conj first , as more specific one will appear beside it and ultimately
      replace it via "undo unary identity projection")
#     s/{([^ ]*) +(.*) <([^ ]*) ([^>]*)>*<(CC)([^>]*)>*<\3([^>]*)>(.*)}/\(\1 \2\{\3 <\3\4> <\5\6> <\3\7>\}\8\)
      /;

    $LEXTPU="[,'' \\!\\.\\?^][ \\)>\\]]*[^\\)>\\]]*|^[^\\)>\\]]*";
    $REXTPU="[ \\)>\\]]*[^\\)>\\]]*[,'' \\!\\.\\?]";

    ####  NOUN PHRASES
    #  right-binarize (basal) NPs as much as possible
    s/{(NP|WHNP)[a-z]*([^ ]*) +(.*)<(?![-,'' \!])(?!')([A-Z]*)([^ ]* [^>]*)> *<(NN[A-Z]*)([a-z]*)([^ ]* [^>]*)>(.*)
      }/\(\1\2 \3\{\6 <\4\5> <\6\7\8>\}\9\)/;
    # left-binarize NP genitive marker if left and right context reduced to nil
    s/{(NP|WHNP)[a-z]*([^ ]*) +(*)<(NN[A-Z]*|NP)([a-z]*)([^ ]* [^>]*)> *<(POS)([^ ]* [^>]*)(*)}/\{\1pos <\4\5\6>
      <\7\8>\}/;
#   # grab NN|NP + comma + mod + internal comma as NP
#   s/{(NP)[a-z]*([^ ]*) +<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<,> *<((?=ADVP|PP|S|VP|WHSBAR|ADJP|NP)(?![^
    ]*-nC)[^ ]*)([^>]*)>(*<,>.*)}/\(\1\2 \{\3 <\3nn\4\5> <\6-nC-mC [,] [\6\7]>\}\8\)/;
#   # grab NN|NP + comma + mod + external punct as NP
#   s/{(NP)[a-z]*([^ ]*) +<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<,> *<((?=ADVP|PP|S|VP|WHSBAR|ADJP|NP)(?![^
    ]*-nC)[^ ]*)([^>]*)> *}($REXTPU)/\(\1\2 \{\3 <\3nn\4\5> <\6-nC-mC [,] [\6\7]>\}\)\8/;
#   # grab mod to right of NN or NP, if left context reduced to nil
#   s/{(NP)[a-z]*([^ ]*) +(*)<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<(ADVP|PP|S|VP|WHSBAR|ADJP|NP)([^ ]*
    [^>]*)>(.*)}/\(\1\2 \3\{\4 <\4nn\5\6> <\7\8>\}\9\)/;  ##>(*<.*)}
    # grab NN|NP + comma + mod + internal comma as NP
    s/{(NP)[a-z]*([^ ]*) +<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<,> *<((?=ADVP|PP|S|VP|WHSBAR|ADJP|NP)(?![^
      ]*-nC)[^ ]*)([^>]*)>(*<,>.*)}/\(\1\2 \{\3 <\3\4\5> <\6-nC-mC [,] [\6\7]>\}\8\)/;
    # grab NN|NP + comma + mod + external punct as NP
    s/{(NP)[a-z]*([^ ]*) +<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<,> *<((?=ADVP|PP|S|VP|WHSBAR|ADJP|NP)(?![^
      ]*-nC)[^ ]*)([^>]*)> *}($REXTPU)/\(\1\2 \{\3 <\3\4\5> <\6-nC-mC [,] [\6\7]>\}\)\8/;
    # grab mod to right of NN or NP, if left context reduced to nil
    s/{(NP)[a-z]*([^ ]*) +(*)<(NN[A-Z]*|NP)n?n?([a-z]*)([^ ]* [^>]*)> *<(ADVP|PP|S|VP|WHSBAR|ADJP|NP)([^ ]* [^>]*)
      >(.*)}/\(\1\2 \3\{\4 <\4\5\6> <\7\8>\}\9\)/;  ##>(*<.*)}
    # undo last unary N bar projection, carrying empty up
    s/\((NP)[a-z]*([^ ]*) +\{(NP|NN[A-Z]*)([a-z]*)([^ ]*empty[^ ]*) +(.*)\} *\)/{\1\2\5 \6}/;  ## +(\[
    # undo last unary N bar projection
    s/\((NP)[a-z]*([^ ]*) +\{(NP|NN[A-Z]*)([a-z]*)(?![^ ]*empty)([^ ]*) +(.*)\} *\)/{\1\2 \6}/;  ## +(\[

debug($step, "\\$_");

    #### TIME NPS
    # substitute NP-tmp
    s/{NP-[^ ]*tmp[^ \-\=]*[^ ]* (.*)}/\(PP-tmp \{NP \1\}\)/g;

    #### VERB PHRASES
    # grab internal comma + S-adv|ADVP|RB|PP + comma + VB|VP as VP
    s/{(VP|VB[A-Z]*)[a-z]*([^ ]*)(.*<,>) *<((?=S-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<,> *<(VB[A-Z]*|VP)([a-z]*)([^
      ]*)([^>]*)>(*| *<[^A-Z].*)}/\(\1\7\2\3 \{\6\7\8 <\4-pC-mC [\4\5] [,]> <\6\7\8\9>\}$10\)/;
    # grab external punct + S-adv|ADVP|RB|PP + comma + VB|VP as VP
    s/($LEXTPU){(VP|VB[A-Z]*)[a-z]*([^ ]*) *<((?=S-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<,> *<(VB[A-Z]*|VP)([a-z]*)([^
      ]*)([^>]*)>(*| *<[^A-Z].*)}/\1\(\2\7\3 \{\6\7\8 <\4-pC-mC [\4\5] [,]> <\6\7\8\9>\}$10\)/;
    # right-binarize VPs after right context reduced to nil or punct constituent
    s/{(VP|VB[A-Z]*)[a-z]*([^ ]*) +(.*)<(S-adv|ADVP|RB|PP)([^ ]* [^>]*)> *<(VB[A-Z]*|VP)([a-z]*)([^ ]* [^>]*)>(*|
      *<[^A-Z].*)}/\(\1\7\2 \3\{\6\7 <\4\5> <\6\7\8>\}\9\)/;  ##(<.*)<
    # left-binarize VPs headed by VB or BES as much as possible
    s/{(VP|VB[A-Z]*|SQ)[a-z]*([^ ]*) +(.*)<(VB[A-Z]*|BES)([a-z]*)([^ ]* [^>]*)> *<(?!CC)([A-Z]+)([^ ]* [^>]*)>(.*)
      }/\(\1\5\2 \3\{\4\5 <\4\5\6> <\7\8>\}\9\)/;
```

```
    # left-binarize VPs headed by TO or MD as much as possible
    s/{(VP|VB[A-Z]*|SQ)[a-z]*([^ ]*) +(.*)<(TO|MD)([a-z]*)([^ ]* [^>]*)> *<(VP|VB)([^ ]* [^>]*)>(.*)}/\(\1\5\2 \3\{
        VP\5 <\4\5\6> <\7\8>\}\9\)/;
    # grab VP|VB + comma + mod + internal comma as VP
    s/{(VP|VB[A-Z]*|SQ)[a-z]*([^ ]*) +(.*)<(VP[a-z]*|VB[A-Za-z]*)([^ ]*)([^>]*)> *<, ,> *<((?=PP|VP|SBAR|Spro|S[^
        ]*-adv|PRN)[^ ]*)([^>]*)>( *<, ,>.*)}/\(\1\5\2 \3\{\4 <\4\5\6> <\7-nC-mC [ , ,] [\7\8]>\}\9\)/;
    # grab VP|VB + comma + mod + external punct as VP
    s/{(VP|VB[A-Z]*|SQ)[a-z]*([^ ]*) +(.*)<(VP[a-z]*|VB[A-Za-z]*)([^ ]*)([^>]*)> *<, ,> *<((?=PP|VP|SBAR|Spro|S[^
        ]*-adv|PRN)[^ ]*)([^>]*)> *}($REXTPU)/\(\1\5\2 \3\{\4 <\4\5\6> <\7-nC-mC [ , ,] [\7\8]>\}\9\)/;
    # left-binarize VPs headed by VP as much as possible
    s/{(VP|VB[A-Z]*|SQ)[a-z]*([^ ]*) +(.*)<(VP)([a-z]*)([^ ]* [^>]*)> *<(PP|VP)([^ ]* [^>]*)>(.*)}/\(\1\5\2
        \3\{\4\5 <\4\5\6> <\7\8>\}\9\)/;
    # percolate it-cleft up until uncle of expl
    s/(-expl-([0-9]+).*)\{([A-Z]*)(.*) <([^ ]*)[-=]\2([^>]*)> *\}/\1\{\3it\4 <\5\6>\}/;
#     s/(-expl-([0-9]+).*)\{(.*) <([^ ]*[-=]\2[^>]*)> *\}/\1\{\3\} <\4>/;  ### raise
#     #s/(-expl-([0-9]*).*)\{[^ ]* +<([^<]*)> +<([^ ]*-\2[^>]*)> *\}/\1\{\3\} <\4>/;
    s/{(.*)<NP[^ ]* +\[NP[a-z]*([^>]*)\] +\[[^ ]*-expl-[0-9]+ 0\]> <([A-Z]*it[^>]*)>(.*)}/{\1<IT\2> <\3>\4}/;
    s/{(.*)<NP[^ ]* +\[NP[a-z]*([^>]*)\] +\[[^ ]*-expl-[0-9]+ 0\]> <([A-Z]*)([^>]*)>(.*)}/{\1<IT\2> <\3it\4>\5}/;
    s/{(.*)<NP[^ ]* +\[NP[a-z]*([^>]*)\] +\[[^ ]*-expl-[0-9]+ 0\]>(.*)}/{\1<IT\2>\3}/;
#     s/{(.*)<NP[^ ]* +\[NP[a-z]*([^>]*)\] +\[[^ ]*-expl-([0-9]+) 0\]> <([^ ]*)([^>]*)> <([^ ]*)-\3([^>]*)>(.*)
      }/\(\1<IT\2> \{\4-aIT <\4\5> <\6\7>\}\8\)/;
    # undo last unary V bar projection
    s/\((VP|VB[A-Z]*)[a-z]*([^ ]*) +\{(VP|VB[A-Z]*)([a-z]*)([^ ]*) +(.*)\} *\)/{\1\4\2 \6}/;  ## +(\[

    #### SENTENTIAL PROJECTIONS
    # right-binarize NP-SBJ VP as S
    s/{(S[A-Z]*)[a-z]*([^ ]*) +(.*)<(NP|IT)([^ ]* [^>]*)> *<(VP)([a-z]*)([^ ]* [^>]*)>(.*)}/\(\1\2 \3\{S <\4\5>
        <\6\7\8>\}\9\)/;
    # under SINV: right-binarize MOD VP as VP
    s/{(SINV)[a-z]*([^ ]*) +(.*)<(S-adv|ADVP|RB[A-Z]*|PP)([^ ]* [^>]*)> *<(VP|VB[A-Z]*)([a-z]*)([^>]*)>(.*)}/\(\1\2
        \3\{\6\7 <\4\5> <\6\7\8>\}\9\)/;
    # under SINV: combine VB NP as SINV-uS
    s/{(SINV.*<S.*)<(VP[^>]*S-empty[^>]*)> *<(NP[^>]*)>(.*)}/(\1\{SINV <\2> <\3>\}\4)/;       ## NOTE: lower SINV
        will become SINV-gS
    # under S: S-# , S --> (S-# S-nC ,) S    (topicalization with comma)
    s/{(S.*)<([^ ]*)(-[0-9]+) ([^>]*)> *<(, ,)> *<(S[a-z]*[^>]*empty\3[^>]*)>(.*)}/\(\1\{\2\3 <\2-nC \4> <\5>\}
        <\6>\7\)/;
    # under S: grab internal comma + S-adv|ADVP|RB|PP + comma + VB|VP as VP
    s/{(S[A-Z]*)[a-z]*([^ ]*)(.*<, ,>) *<((?=S[a-z]*-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<, ,> *<(VB[A-Z]*|VP)([a-z]*)
        ([^ ]*)([^>]*)>(.*)}/\(\1\7\2\3 \{\6\7\8 <\4-pC-mC [\4\5] [ ,] > <\6\7\8\9>\}$10\)/;
    # under S: grab external punct + S-adv|ADVP|RB|PP + comma + VB|VP as VP
    s/($LEXTPU){(S[A-Z]*)[a-z]*([^ ]*) *<((?=S[a-z]*-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<, ,> *<(VB[A-Z]*|VP)([a-z]*)
        ([^ ]*)([^>]*)>(.*)}/\1\(\2\7\3 \{\6\7\8 <\4-pC-mC [\4\5] [ ,] > <\6\7\8\9>\}$10\)/;
    # under S: right-binarize MOD VP as VP
    s/{(S[A-Z]*)[a-z]*([^ ]*) +(<.*)<(S-adv|ADVP|RB[A-Z]*|PP)([^ ]* [^>]*)> *<(VP|VB[A-Z]*)([a-z]*)([^>]*)>(.*)
        }/\(\1\2 \3\{\6\7 <\4\5> <\6\7\8>\}\9\)/;
    # under S: grab internal comma + S-adv|ADVP|RB|PP + comma + S as S
    s/{(S[A-Z]*)[a-z]*([^ ]*)(.*<, ,>) *<((?=PRN|SBAR|S[^-]*-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<, ,> *<(\1)([a-z]*)([^
        ]*)([^>]*)>(.*)}/\(\1\7\2\3 \{\6\7\8 <\4-pC-mC [\4\5] [ ,] > <\6\7\8\9>\}$10\)/;
    # under S: grab external punct + S-adv|ADVP|RB|PP + comma + S as S
    s/($LEXTPU){(S[A-Z]*)[a-z]*([^ ]*) *<((?=PRN|SBAR|S[^-]*-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<, ,> *<(\2)([a-z]*)([^
        ]*)([^>]*)>(.*)}/\1\(\2\7\3 \{\6\7\8 <\4-pC-mC [\4\5] [ ,] > <\6\7\8\9>\}$10\)/;
    # recover from forgotten comma before pre-sentence modifier
    s/{(S[A-Z]*)[a-z]*([^ ]*)(.*) *<((?=PRN|SBAR|S[^-]*-adv|ADVP|RB|PP)[^ ]*)([^>]*)> *<, ,> *<(\1)([a-z]*)([^ ]*)
        ([^>]*)>(.*)}/\(\1\7\2\3 \{\6\7\8 <\4-mC [\4\5] [ ,] > <\6\7\8\9>\}$10\)/;
    # grab comma between NP and VP
    s/{(S.*)<(NP.*)> +<(, ,)> +<(VP[^ ]*) (.*)>(.*)}/(\1<\2> \{\4 <\3> <\4 \5>\}\6)/;
    # under S: right-binarize MOD + S as S
    s/{(S[A-Z]*)[a-z]*([^ ]*) +(.*)<(PRN|S-adv|ADVP|RB[A-Z]*|PP)([^ ]* [^>]*)> *<(\1)([a-z]*)( [^>]*)>(.*)
        }/\(\1\7\2 \3\{\6\7 <\4\5> <\6\7\8>\}\9\)/;
    # left-binarize SINV
```

```
s/{(SINV)[a-z]*([^ ]*) +(.*)<(\1)([^ ]*)( [^>]*)> *<(PRN|ADVP|RB[A-Z]*|PP)([^ ]* [^>]*)>(.*)}/\(\1\5\2 \3\{\4\5
      <\4\5\6> <\7\8>\}\9\)/;
# left-binarize S
s/{(S[A-Z]*)[a-z]*([^ ]*) +(.*)<(\1)([a-z]*)( [^>]*)> *<(PRN|ADVP|RB[A-Z]*|PP)([^ ]* [^>]*)>(.*)}/\(\1\5\2
     \3\{\4\5 <\4\5\6> <\7\8>\}\9\)/;
# redo Sto
s/{(S) +(<NP.*) +(<VPto.*)}/{\1to \2 \3}/;
# undo last unary S bar projection
s/\((S)[a-z]*([^ ]*) +\{(S)([a-z]*)([^ ]*) +(.*)\} *\)/{\1\4\2 \6}/;  ## +(\[
# identify small clause
s/{S([^ ]*) +<(NP[^>]*|IT[^>]*)> +<(VPvbn|VBNvbn|VPvbg|VBGvbg|ADJP|PP)([^>]*)> *}/{SC\1 <\2> <\3\4>}/;

#### ADJECTIVAL / ADVERBIAL PHRASES
# right-binarize ADJPs as much as possible
s/{(ADJP)[a-z]*([^ ]*) +(.*)<(RB[A-Z]*|ADVP)([^ ]* [^>]*)> *<(JJ[A-Z]*)([a-z]*)([^ ]* [^>]*)>(.*)}/\(\1\2
     \3\{\6 <\4\5> <\6\7\8>\}\9\)/;
# left-binarize ADJPs after left context reduced to nil
s/{(ADJP)[a-z]*([^ ]*) +( *)<(JJ[A-Z]*|ADJP)([a-z]*)([^ ]* [^>]*)> *<(PRN|PP|S)([^ ]* [^>]*)>(.*)}/\(\1\2
     \3\{\4 <\4\5\6> <\7\8>\}\9\)/;  ##>( *<.*)}
# undo last unary A bar projection
s/\((ADJP)[a-z]*([^ ]*) +\{(JJ[A-Z]*)([a-z]*)([^ ]*) +(.*)\} *\)/{\1\2 \6}/;  ## +(\[
# right-binarize ADVPs as much as possible
s/{(ADVP)[a-z]*([^ ]*) +(.*)<(RB[A-Z]*)([^ ]* [^>]*)> *<(RB[A-Z]*)([a-z]*)([^ ]* [^>]*)>(.*)}/\(\1\2 \3\{\6
     <\4\5> <\6\7\8>\}\9\)/;
# left-binarize ADVPs after left context reduced to nil
s/{(ADVP)[a-z]*([^ ]*) +( *)<(RB[A-Z]*|ADVP)([a-z]*)([^ ]* [^>]*)> *<(PRN|PP|S)([^ ]* [^>]*)>(.*)}/\(\1\2
     \3\{\4 <\4\5\6> <\7\8>\}\9\)/;  ##>( *<.*)}
# undo last unary Ad bar projection
s/\((ADVP)[a-z]*([^ ]*) +\{(RB[A-Z]*)([a-z]*)([^ ]*) +(.*)\} *\)/{\1\2 \6}/;  ## +(\[
# annotate unary rb
s/{(ADVP[^ ]*) *<(RB[^ ]*) ([^ >]*)> *}/{\1 <\2-unary \3>}/;

#### PREPOSITIONAL PHRASES
# annotate prepositions with word instead of pos
s/{(IN)[a-z]*([^ ]*) *(of|that)(\!colon\!.*)?}/{\1\3\2 \3\4}/;
#s/{(PP|SBAR)[a-z]*([^ ]*) *<(IN|TO)(of|that|to)([^ ]* .*)}/{\1\4 <\3\4\5}/;
# left-binarize PPs/SBARs headed by IN or TO as much as possible
s/{(PP|SBAR)[a-z]*([^ ]*) +(.*)<(IN|TO)([a-z]*)([^ ]* [^>]*)> *<([A-Z]+)([^ ]* [^>]*)>(.*)}/\(\1\5\2 \3\{\1\5\2
      <\4\5\6> <\7\8>\}\9\)/;
# right-binarize PPs after right context reduced to nil
s/{(PP)[a-z]*([^ ]*) +(.*)<(ADVP|RB|PP)([^ ]* [^>]*)> *<(PP)([a-z]*)([^ ]* [^>]*)>( *)}/\(\1\7\2 \3\{\6\7
     <\4\5> <\6\7\8>\}\9\)/; ##(<.*)<
# undo last unary P bar projection
s/\((PP|SBAR)[a-z]*([^ ]*) +\{(\1)([a-z]*)([^ ]*) +(.*)\} *\)/{\1\4\2 \6}/;  ## +(\[

#### TERMINAL SYMBOLS
# propagate unary head pos at terminal
s/{(NP)[a-z]*([^ ]*) +<(NN[A-Z]*)([a-z]*)([^ ]*) +([^<>]*)> *}/{\1\2 <\3\5 \6>}/;
# propagate unary head pos at terminal
s/{(VP)[a-z]*([^ ]*) +<(VB[A-Z]*)([a-z]*)([^ ]*) +([^<>]*)> *}/{\1\4\2 <\3\4\5 \6>}/;
# propagate unary head pos at terminal
s/{(ADJP)[a-z]*([^ ]*) +<(JJ[A-Z]*)([a-z]*)([^ ]*) +([^<>]*)> *}/{\1\2 <\3\5 \6>}/;
# propagate unary head pos at terminal
s/{(ADVP)[a-z]*([^ ]*) +<(RB[A-Z]*)([a-z]*)([^ ]*) +([^<>]*)> *}/{\1\2 <\3\5 \6>}/;
# undo unary identity projection
s/{([^ ]*) +<\1([^ ]*) ([^>]*)> *}/{\1\2 \3}/;

s/{(.*)<(CD [^>]*)> *<(CD [^>]*)>(.*)}/\(\1\{CD <\2> <\3>\4\}\)/;
s/{(.*)<(RB [^>]*)> *<(QP [^>]*)>(.*)}/\(\1\{QP <\2> <\3>\4\}\)/;

#### BRACKETS / PARENS
# 1a. introduce, from matched brackets / parens at edges of constituent, delimited tag
```

```
    s/{(?![^ ]*-mB)([^ ]*) +(<\!LRB\! \!lrb\!> .* <\!RRB\! \!rrb\!>) *}/{\1-mB \2}/;
    # 1b. introduce, from matched brackets / parens inside constituent, delimited tag on new constituent
    s/{(?![^ ]*-mB)(.*) (<\!LRB\! \!lrb\!>) +<(?!')([^ ]*?)(-[pn]B)?([^ ]*)([^>]*)>+(<\!RRB\!\!rrb\!>)(.*)}/\(\1 \{\3\5-mB \2<\3\4\5\6> \7\}\8\)/;
____#2a. introduce, from beginning of sentence to bracket / paren at end of constituent, delimited tag
____s/^([^\)>\]]*){(?![^]*-mB)([^]*) +(<(?![^]*-nB)(?!')[^>]*> +<\!RRB\! \!rrb\!>) *}/\1\{\2-pB-mB \3\}/;
    # 2b. introduce, from beginning of sentence to bracket / paren at end of constituent, delimited tag on new constituent
    s/^([^\)>\]]*){(?![^]*-mB)([^]*) +<(?![^]*-nB)(?!')(?!NP)([^]*?)(-pB)?([^]*)([^>]*)>+(<\!RRB\!\!rrb\!>)(.*)}/\1\(\2 \{\3\5-pB-mB <\3\4\5\6> \7\}\8\)/;
____#3a. introduce, from external punct to bracket / paren at end of constituent, delimited tag
____s/([,''\!\.\?][\)>\]]+[^\)>\]]*){(?![^]*-mB)([^]*) +(<(?![^]*-nB)(?!')[^>]*>+<\!RRB\!\!rrb\!>)*}/\1\{\2-pB-mB \3\}/;
____#3b. introduce, from external punct to bracket / paren at end of constituent, delimited tag on new constituent
____s/([,''\!\.\?][\)>\]]+[^\)>\]]*){(?![^]*-mB)([^]*) +<(?![^]*-nB)(?!')(?!NP)([^]*?)(-pB)?([^]*)([^>]*)> +(<\!RRB\!\!rrb\!>)(.*)}/\1\(\2 \{\3\5-pB-mB <\3\4\5\6> \7\}\8\)/;
____#4a. introduce, from bracket / paren at edge of constituent to external punct, delimited tag
____s/{(?![^]*-mB)([^]*)+(<\!LRB\!\!lrb\!>+<(?![^]*-pB)(?!')([^]*?)(-nB)?([^]*) ([^>]*)> *}([\)>\]]*[^\)>\]]*[,''\!\.\?])/\{\1-nB-mB \2\}\3/;
____#4b. introduce, from bracket / paren within the constituent to external punct, delimited tag on new constituent
____s/{(?![^]*-mB)(.*)+(<\!LRB\!\!lrb\!>)+<(?![^]*-pB)(?!')([^]*?)(-nB)?([^]*) ([^>]*)> *}([\)>\]]*[^\)>\]]*[,''\!\.\?])/\(\1 \{\3\5-nB-mB \2<\3\4\5\6>\}\)\7/;

____#### QUOTES
____# grab complete quote
____s/{(?![^]*-mQ)(.*)(<''?''?>)+<(?![^]*-mQ)([^]*?)(-[pn]Q)?([^]*)([^>]*)>+((<,,>+)?<''?''?>)(.*)}/\(\1 \{\3\5-mQ \2<\3\4\5\6> \7\}\9\)/;
____# grab incomplete quote: '' X as X-nQ-nC-mQ
____s/{([^]*)(.*)(<''>)+<([^]*)([^>]*)>*}/\(\1 \2\{\4-nQ-nC-mQ \3<\4 \5\>\}\)/;
____# tag incomplete quote: ... '' ... as -nQ-nC
____s/{(?![^]*-mQ)([^]*)(.*<''>((?!'')*).*)}/{\1-nQ-nC \2}/;
____# grab incomplete quote: X '' as X-pQ-mQ
____s/{([^]*)+<([^,]*)([^>]*)>(<''>)(.*)*}/\(\1 \{\2-pQ-mQ <\2\3> \4\}\5\)/;
____# tag incomplete quote: ... '' ... as -pQ
____s/{(?![^]*-mQ)([^]*)((?!''*).)*<''>.*)}/{\1-pQ \2}/;

____#### DASHES
____# 1a. introduce, from matched dashes at edges of constituent, delimited tag
____s/{(?![^]*-mD)([^]*)+(<\!dash\!\!dash\!>.*<\!dash\!\!dash\!>)*}/{\1-mD \2}/;
____# 1b. introduce, from matched dashes inside constituent, delimited tag on new constituent
____s/{(?![^]*-mD)(.*)(<\!dash\!\!dash\!>)+<(?!')([^]*?)(-[pn]D)?([^]*) ([^>]*)> +(<\!dash\! \!dash\!>)(.*)}/\(\1 \{\3\5-mD \2 <\3\4\5 \6> \7\}\8\)/;
    # 2a. introduce, from beginning of sentence to dash at end of constituent, delimited tag
    s/^([^\)>\]]*){(?![^]*-mD)([^]*) +(<(?![^]*-nD)(?!')[^>]*>+<\!dash\!\!dash\!>)*}/\1\{\2-pD-mD \3\}/;
____# 2b. introduce, from beginning of sentence to dash at end of constituent, delimited tag on new constituent
____s/^([^\)>\]]*){(?![^]*-mD)([^]*)+<(?![^]*-nD)(?!')(?!NP)([^]*?)(-pD)?([^]*) ([^>]*)> +(<\!dash\! \!dash\!>)(.*)}/\1\(\2 \{\3\5-pD-mD <\3\4\5 \6> \7\}\8\)/;
    # 3a. introduce, from external punct to dash at end of constituent, delimited tag
    s/([,''\!\.\?][\)>\]]+[^\)>\]]*){(?![^]*-mD)([^]*)+(<(?![^]*-nD)(?!')[^>]*> +<\!dash\! \!dash\!>)*}/\1\{\2-pD-mD \3\}/;
    # 3b. introduce, from external punct to dash at end of constituent, delimited tag on new constituent
    s/([,''\!\.\?][\)>\]]+[^\)>\]]*){(?![^]*-mD)([^]*)+<(?![^]*-nD)(?!')(?!NP)([^]*?)(-pD)?([^]*) ([^>]*)> +(<\!dash\! \!dash\!>)(.*)}/\1\(\2 \{\3\5-pD-mD <\3\4\5 \6> \7\}\8\)/;
    # 4a. introduce, from dash at edge of constituent to external punct, delimited tag
    s/{(?![^]*-mD)([^]*) +(<\!dash\! \!dash\!> +<(?![^]*-pD)(?!')[^>]*>)*}([\)>\]]*[^\)>\]]*[,''\!\.\?])/\{\1-nD-mD \2\}\3/;
    # 4b. introduce, from dash within the constituent to external punct, delimited tag on new constituent
    s/{(?![^]*-mD)(.*) +(<\!dash\! \!dash\!>) +<(?![^]*-pD)(?!')([^]*?)(-nD)?([^]*)([^>]*)>*}([\)>\]]*[^\)>\]]*[,''\!\.\?])/\(\1 \{\3\5-nD-mD \2 <\3\4\5 \6>\}\)\7/;

    #### COMMAS
```

```
    # 1a. introduce, from matched dashes at edges of constituent, delimited tag
    s/{(?![^ ]*-mC)([^ ]*) +(<, ,> .* <, ,>) *}/{\1-mC \2}/;


    # make sure punct tags precede everything else
    while ( s/{([^ ]*)(?!-[pn][a-z])(-[^- ]+)(-[pn][a-z])/{\1\3\2/g ){}
    #while ( s/<([^ ]*)(?!-[pn][a-z])(-[^- ]+)(-[pn][a-z])/<\1\3\2/g ){}
    # delete redundant tags
    #while ( s/{([^ ]*)(?!-[pn][a-z])(-[^- ]+)(-[pn][a-z])/{\1\3\2/g ){}
    while ( s/<([^ ]*)(-[^- ]+)([^ ]*)\1/<\1\2\3/g ){}


    #print stderr ":::$_\n";
    ####################

    ## convert inner angles (if any) to bracks...
    while ( s/{(.*)<([^>]*)>(.*)}/{\1\[\2\]\3}/ ){}
    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }
  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;
  ## translate to parens again...
  $_ =~ s/\[/\(/g;
  $_ =~ s/\]/\)/g;

  $_ =~ s/____INTJ/INTJ/g;

  print $_;
}
```

B.4 scripts/annotateArgs.pl

```
###############################################################################
##                                                                           ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                           ##
##    ModelBlocks is free software: you can redistribute it and/or modify    ##
##    it under the terms of the GNU General Public License as published by   ##
##    the Free Software Foundation, either version 3 of the License, or      ##
##    (at your option) any later version.                                    ##
##                                                                           ##
##    ModelBlocks is distributed in the hope that it will be useful,         ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of         ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          ##
##    GNU General Public License for more details.                           ##
##                                                                           ##
##    You should have received a copy of the GNU General Public License      ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.   ##
##                                                                           ##
###############################################################################

use Getopt::Std;

getopts("d");
```

```
$DEBUG = 0;
if ($opt_d) {
  $DEBUG = 1;
}

sub debug {
  if ($DEBUG) {
    $msg = $_[1];
    print stderr $_[0] , "␣" , $msg, "\n";
  }
}

## for each tree...
while ( <> ) {

  # this is not really the place for it, but should be done somewhere
  s/ +/ /g;

  ## mark top-level constituent as current...
  s/^ *\((.*)\) *$/{\1}/;
  ## mark all other constituents as internal...
  s/\(/\[/g;
  s/\)/\]/g;
  ## for each constituent...
  while ( $_ =~ /{/ ) {
    ## mark all children of current...
    for ( $i=index($_,'{'),$d=0; $i<index($_,'}'); $i++ ) {
      if ( substr($_,$i,1)eq'[' ) { if($d==0){substr($_,$i,1)='<';} $d++; }
      if ( substr($_,$i,1)eq']' ) { $d--; if($d==0){substr($_,$i,1)='>';} }
    }
    ##################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
    debug(++$step, "␣␣␣$_");

    # pass time-np tags down to each head
    while ( s/{([^ ]*)(-tmp)(.*) <(?![^ ]*-tmp)(NP[a-z]*|NNP?S?[a-z]*)(.*)>( *)}/{\1\2\3<\4\2\5>\6}/ ) {}

    # pass subcat tags down to each identical child (modulo optional conjunction prefix)
    while ( s/{(C?)(((?!-v|-u)[^ ])*)((-v|-u[^- ]+)+)(.*) <(?![^ ]*-v|[^ ]*-u)(C?\2)([- ].*)>(.*)
          }/{\1\2\4\6<\7\4\8>\9}/ ) {}

    # identify arguments and add subcat tags
    s/{(NP|NN)([^<]*)<(DT[^ ]*)([^>]*)> *<(NNP?S?|JJR?S?)([- ][^>]*)> *}/{\1\2<\3-u\5\4> <\5\6>}/;
    s/{(NP|NN)(.*)<(NN[^ ]*)([^>]*)> *<(Sproto|Sto|SBARthat|SC|NP|AP|ADJP|PPof)([- ][^>]*)> *}/{\1\2<\3-u\5\4>
          <\5\6>}/;
    s/{(PP|SBAR)(.*)<(IN[^ ]*)([^>]*)> *<(Sproto(?![^ ]*-adv)|Sto(?![^ ]*-adv)|Sprovbn(?![^ ]*-adv)|Sprovbg(?![^
          ]*-adv)|S(?![^ ]*-adv)|SC|NP|AP|ADJP)([- ][^>]*)> *}/{\1\2<\3-u\5\4> <\5\6>}/;
    s/{(Spro|VP(?!it)|VB)(.*)<(VB[^ ]*)([^>]*)> *<(Sproto(?![^ ]*-adv)|Sto(?![^ ]*-adv)|SBAR(?![^ ]*-adv)|SBARthat|
          Sprovbn(?![^ ]*-adv)|Sprovbg(?![^ ]*-adv)|S(?![^ ]*-adv)|SC|NP|AP|ADJP|VP[a-z]*|PRT[a-z]*)([- ][^>]*)>
          *}/{\1\2<\3-u\5\4> <\5\6>}/;
    s/{(AP|ADJP)(.*)<(AP|JJ[^ ]*)([^>]*)> *<(Sproto(?![^ ]*-adv)|Sto(?![^ ]*-adv)|SBAR(?![^ ]*-adv)|SBARthat|
          Sprovbn(?![^ ]*-adv)|Sprovbg(?![^ ]*-adv)|S(?![^ ]*-adv)|SC|NP|AP|ADJP|VP[a-z]*|PRT[a-z]*)([- ][^>]*)>
          *}/{\1\2<\3-u\5\4> <\5\6>}/;

    #s/{(VP|VB)(.*)<(VB[^ ]*)([^>]*)> *<(Sproto(?![^ ]*-adv)|Sto(?![^ ]*-adv)|SBAR(?![^ ]*-adv)|SBARthat|Sprovbn
          (?![^ ]*-adv)|Sprovbg(?![^ ]*-adv)|S(?![^ ]*-adv)|NP|AP|ADJP|VP[a-z]*|PRT[a-z]*)([- ][^>]*)> *}/{\1\2<\3-
          u\5\4> <\5\6>}/;

    ####################
    ## mark current as external...
    s/{(.*?)}/\(\1\)/;
    ## mark first unexpanded child as current...
    s/<(.*?)>/{\1}/;
```

```
  }
  # output
  print $_;
}
```

B.5 scripts/annotateGaps.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################


# cat wsj_0001.trees | perl scripts/treesed.pl
use Getopt::Std;

getopts("d");

$DEBUG = 0;
if ($opt_d) {
  $DEBUG = 1;
}

sub debug {
  if ($DEBUG) {
    $msg = $_[1];
    print stderr $_[0] , "␣" , $msg, "\n";
  }
}

# luan: commented out to simplify b/c now model role as '-1' tag
#$SRL = "\w+\!ldelim\!";

## for each tree...
while ( <> ) {
  ## translate to parens...
  s/\[/\(/g;
  s/\]/\)/g;
  ## for each constituent...
  while ( $_ =~ /\([^\(\)]*\)/ ) {
    ## convert outer parens to braces...
    $_ =~ s/\(([^\(\)]*)\)/{\1}/;
    ################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
```

```
    debug($step++, "   $_");

    #### delete unhandled traces
    # fold child trace into parent nonterm as extraction
    s/{([^ ]*) +<\*NONE\*[^>]*> *}/{\1-empty}/;
    #s/{([^ ]*) +<($SRL)?\*NONE\*[^>]*> *}/{\1-empty}/;
    # fold child trace into parent nonterm as extraction
    s/{([^ ]*) +( *)<\*NONE\*[^>]*>(.*<.*)}/\3/;
    #s/{([^ ]*) +( *)<($SRL)?\*NONE\*[^>]*>(.*<.*)}/\4/;
    # fold child trace into parent nonterm as extraction
    s/{([^ ]*) +(.*<.*)<\*NONE\*[^>]*>( *)}/\2/;
    #s/{([^ ]*) +(.*<.*)<($SRL)?\*NONE\*[^>]*>( *)}/\2/;
    # fold child trace into parent nonterm as extraction
    s/{([^ ]*) +(.*)<\*NONE\*[^>]*>(.*)}/\2\3/;
    #s/{([^ ]*) +(.*)<($SRL)?\*NONE\*[^>]*>(.*)}/\2\4/;

    # fold child empty category into parent nonterm as extraction (trace chain)
    s/{([^ ]*) +(.*)<([^ \^]*)[^ ]*-([0-9]+)-empty-([0-9]+) *0 *>(.*<[^ ]*-g[^ ]*-\4.*)}/{\1-g\3-\5 \2\6}/;
    #s/{([^ ]*) +(.*)<($SRL)?([^ \^]*)[^ ]*-([0-9]+)-empty-([0-9]+) *0 *>(.*<[^ ]*-g[^ ]*-\5.*)}/{\1-g\4-\6 \2\7}/;
    # fold child empty category into parent nonterm as extraction
    s/{([^ ]*) +(.*)<([^ \^\-]*)[^ ]*-empty-([0-9]+) *0 *>(.*)}/{\1-g\3-\4 \2\5}/;
    #s/{([^ ]*) +(.*)<($SRL)?([^ \^\-]*)[^ ]*-empty-([0-9]+) *0 *>(.*)}/{\1-g\4-\5 \2\6}/;

    # project extraction up
    s/{([^ ]*) +(.*<[^ ]*)-g([^ ]+) +(.*>.*)}/{\1-g\3 \2-g\3 \4}/;

#debug($step++, " > $_");
#     # remove WHSBAR dominating just S-gNP
#     s/{WHSBAR[^ ]* <> <(S[^ ]*-gNP[^ ]*) ([^>]*)> *}/{\1 \3}/;
    # undo extr projection and nuke WHSBAR if '-[0-9]' numbers of siblings match (left child is empty extr: e.g. "
        the car I saw")
    s/{(WHSBAR[^ ]*)-g([^ ]*)-([0-9]+) +(.*<WHNP[^ ]*-\3-empty *0 *)> +<([^ ]*-g\2-\3[^0-9][^>]*)> *}/{ERASEME
        <\5>}/;
    # undo extr projection if '-[0-9]' numbers of siblings match + fold child empty category into parent (trace
        chain ending in pro)
    s/{([^ ]*)-g([^ ]*)-([0-9]+) +(.*)<[^ \^]*[^ ]*-\3-empty *0 *>(.*<[^ ]*-g[^ ]*-\3.*)}/{\1 \4\5}/;
    # undo extr projection if '-[0-9]' numbers of siblings match (parent is extr)
    s/{([^ ]*)-([0-9]+)-g[^ ]*-\2 +(.*<.*)}/{\1 \3}/;
    # undo extr projection if '-[0-9]' numbers of siblings match (left child is extr)
    s/{([^ ]*)-g([^ ]*)-([0-9]+) +(.*<(?![^ ]*-g)[^ ]*-\3[^0-9].*) +(<[^ ]*-g\2-\3[^0-9].*)}/{\1 \4 \5}/;
    # undo extr projection if '-[0-9]' numbers of siblings match (right child is extr)
    s/{([^ ]*)-g([^ ]*)-([0-9]+) +(.*<[^ ]*-g\2-\3[^0-9].*) +(<(?![^ ]*-g)[^ ]*-\3[^0-9].*)}/{\1 \4 \5}/;

    # last resort: nuke all remaining empty constituents w/o trace
    s/{([^ ]*) +<[^>]*-empty[^>]*> *<[^>]* ([^>]*)> *}/{\1 \2}/;
    s/{([^ ]*) +<[^>]* ([^>]*)> +<[^>]*-empty[^>]*> *}/{\1 \2}/;
    s/{([^ ]*) (.*) <[^>]*-empty[^>]*>(.*)}/{\1 \2\3}/;

    # remove child of unary constituents that remain
    s/{((?!ERASEME)[^ ]*) +<([^ ]*) ([^>]*)> *}/{\1 \3}/;
    # leave gap for empty-extr rel clause
    s/{(.*) <ERASEME ([^>]*)>(.*)}/{\1 \2\3}/;

    ####################
    ## convert inner angles (if any) to bracks...
    while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}
    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }

  ################### FINALLY: delete numbers...
  s/[-=][0-9]+([^ \)\]]*) /\1 /g;
```

```
  s/[-=][0-9]+([^ \)\]]*) /\1 /g;
  s/[-=][0-9]+([^ \)\]]*) /\1 /g;

  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;
  ## translate to parens again...
  $_ =~ s/\[/\(/g;
  $_ =~ s/\]/\)/g;
  ## output...
  print $_;
}
```

B.6 scripts/annotateMarks.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################

use Getopt::Std;

getopts("d");

$DEBUG = 0;
if ($opt_d) {
  $DEBUG = 1;
}

sub debug {
  if ($DEBUG) {
    $msg = $_[1];
    print stderr $_[0] , "␣" , $msg, "\n";
  }
}

## for each tree...
while ( <> ) {

  ## translate to parens...
  s/\[/\(/g;
  s/\]/\)/g;
  ## for each constituent...
```

```
while ( $_ =~ /\([^\(\)]*\)/ ) {
  ## convert outer parens to braces...
  $_ =~ s/\(([^\(\)]*)\)/{\1}/;
  ##################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
  debug(++$step, "___$_");


  # propagate right punct
  s/{([^- ]*)([^ ]*)(.*)<([^ ]*?)((-n.)+)([^>]*)> *}/{\1\5\2\3<\4\5\7>}/;
  # propagate left punct
  s/{([^- ]*)([^ ]* *)<([^ ]*?)((-p.)+)([^>]*)>(.*)}/{\1\4\2<\3\4\6>\7}/;
  # undo duplicates
  while ( s/{([^ ]*)(-[pn].)([^ ]*)\2/{\1\3\2/ ) {}


  #### PUNCT
  # kick right eos punct out of constit
  s/{(?![^ ]*-nE)([^ ]*) +(.*) +<([\.\!\?] [\.\!\?])>}(?=.*\))/{\1-nE \2} <\3>/;

  #### BRACKETS / PARENS
  # kick left  brack/paren up out of constit
  s/{(?![^ ]*-pB)([^ ]*) +<(\!LRB\! \!lrb\!)> +(.*)}(?=.*\))/<\2> {\1-pB \3}/;
  # kick right brack/paren up out of constit
  s/{(?![^ ]*-nB)([^ ]*) +(.*) +<(\!RRB\! \!rrb\!)>}(?=.*\))/{\1-nB \2} <\3>/;

  #### QUOTES
  # kick left  quote up out of constit
  s/{(?![^ ]*-pQ)([^ ]*) +<(``? ``?)> +(.*)}(?=.*\))/<\2> {\1-pQ \3}/;
  # kick right quote up out of constit
  s/{(?![^ ]*-nQ)([^ ]*) +(.*) +<('' ? '' ?)>}(?=.*\))/{\1-nQ \2} <\3>/;

  #### DASHES
  # kick left  dash up out of constit
  s/{(?![^ ]*-pD)([^ ]*) +<(\!dash\! \!dash\!)> +(.*)}(?=.*\))/<\2> {\1-pD \3}/;
  # kick right dash up out of constit
  s/{(?![^ ]*-nD)([^ ]*) +(.*) +<(\!dash\! \!dash\!)>}(?=.*\))/{\1-nD \2} <\3>/;

  #### COMMAS
  # kick left  comma out of constit
  s/{(?![^ ]*-pC)([^ ]*) +<(, ,)> +(.*)}(?=.*\))/<\2> {\1-pC \3}/;
  # kick right comma out of constit
  s/{(?![^ ]*-nC)([^ ]*) +(.*) +<(, ,)>}(?=.*\))/{\1-nC \2} <\3>/;


  ####################
  ## convert inner angles (if any) to bracks...
  while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}
  ## convert outer braces to angles...
  $_ =~ s/{(.*)}/<\1>/;
}
## finish up...
$_ =~ s/</[/;
$_ =~ s/>/]/;
## translate to parens again...
$_ =~ s/\[/\(/g;
$_ =~ s/\]/\)/g;

## for each constituent...
while ( $_ =~ /\([^\(\)]*\)/ ) {
  ## convert outer parens to braces...
  $_ =~ s/\(([^\(\)]*)\)/{\1}/;
  ##################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
```

```
    debug(++$step, "␣␣␣$_");

    # put back any remaining left  comma
    s/{(.*)(<, ,>) +<([^ ]*)–pC([^ ]*) ([^>]*)>(.*)}/\(\1\{\3\4 \2 <\3–pC\4 \5>\}\6\)/;
    # put back any remaining right comma
    s/{(.*)<([^ ]*)–nC([^ ]*) ([^>]*)> +(<, ,>)(.*)}/\(\1\{\2\3 <\2–nC\3 \4> \5\}\6\)/;
    # put back any remaining left  dash
    s/{(.*)(<\!dash\! \!dash\!>) +<([^ ]*)–pD([^ ]*) ([^>]*)>(.*)}/\(\1\{\3\4 \2 <\3–pD\4 \5>\}\6\)/;
    # put back any remaining right dash
    s/{(.*)<([^ ]*)–nD([^ ]*) ([^>]*)> +(<\!dash\! \!dash\!>)(.*)}/\(\1\{\2\3 <\2–nD\3 \4> \5\}\6\)/;
    # put back any remaining left  quote
    s/{(.*)(<``? ``?>) +<([^ ]*)–pQ([^ ]*) ([^>]*)>(.*)}/\(\1\{\3\4 \2 <\3–pQ\4 \5>\}\6\)/;
    # put back any remaining right quote
    s/{(.*)<([^ ]*)–nQ([^ ]*) ([^>]*)> +(<''? ''?>)(.*)}/\(\1\{\2\3 <\2–nQ\3 \4> \5\}\6\)/;
    # put back any remaining left  brack/paren
    s/{(.*)(<\!LRB\! \!lrb\!>) +<([^ ]*)–pB([^ ]*) ([^>]*)>(.*)}/\(\1\{\3\4 \2 <\3–pB\4 \5>\}\6\)/;
    # put back any remaining right brack/paren
    s/{(.*)<([^ ]*)–nB([^ ]*) ([^>]*)> +(<\!RRB\! \!rrb\!>)(.*)}/\(\1\{\2\3 <\2–nB\3 \4> \5\}\6\)/;
    # put back any remaining right punct
    s/{(.*)<([^ ]*)–nE([^ ]*) ([^>]*)> +(<[\.\!\?] [\.\!\?]>)(.*)}/\(\1\{\2\3 <\2–nE\3 \4> \5\}\6\)/;

    ####################
    ## convert inner angles (if any) to bracks...
    while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}
    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }
  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;
  ## translate to parens again...
  $_ =~ s/\[/\(/g;
  $_ =~ s/\]/\)/g;

  ## for each constituent...
  while ( $_ =~ /\([^\(\)]*\)/ ) {
    ## convert outer parens to braces...
    $_ =~ s/\(([^\(\)]*)\)/{\1}/;
    ##################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
    debug(++$step, "␣␣␣$_");

    # remove parent of (nonterminal?) unary constituents that remain
#     s/{[^ ]* +<([^ ]* [^>]*\[[^>]*]> *}/{\1}/;
    s/{[^ ]* +<([^ ]* [^>]*]> *}/{\1}/;
#     s/{([^ ]*)(-.dlt|-p.|-n.)+ +<\1 ([^>]*]> *}/{\1 \3}/;

    ####################
    ## convert inner angles (if any) to bracks...
    while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}
    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }
  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;
  ## translate to parens again...
  $_ =~ s/\[/\(/g;
  $_ =~ s/\]/\)/g;



  #### NOTE!!!!!!
```

```
  s/-[mnp]Q//g;



  print $_;
}
```

B.7 scripts/ensureCnf.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################

use Getopt::Std;

getopts("dn");

$DEBUG = ($opt_d) ? 1 : 0;
$PRINT_POS = ($opt_n) ? 0 : 1;

sub debug {
  if ($DEBUG) {
    print stderr $_[0] , "␣" , $_[1], "\n";
  }
}


## for each tree...
while ( <> ) {

  ## translate to parens...
  s/\[/\(/g;
  s/\]/\)/g;

  ## for each constituent...
  while ( $_ =~ /\([^\(\)]*\)/ ) {

    ## convert outer parens to braces...
    $_ =~ s/\(([^\(\)]*)\)/{\1}/;

    ################### ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
    debug($step++, "␣␣␣$_");
```

```
    # create underscore cats
    if ( s/{([^ ]*) *(<.*) *<([^ ]*)([^>]*)> *<([^ ]*)([^>]*)> *}/\(\1 \2 {\3\_\5 <\3\4> <\5\6>}\)/ ) {

      # clean up -l tags in new underscore cat...
      s/{([^ \_]*)-l[^- \_]+([^ ]*)-l[^- \_]+([^ \_]*) (.*)}/{\1\2\3-l1 \4}/;

      # clean up -p tags in first of new underscore cat...
      while ( s/{([^ \_]*)(-p[^- \_]+)([^ ]*) (.*)}/{\1\3\2 \4}/ ) {}
      # clean up -p tags in rest of new underscore cat...
      while ( s/{([^ ]*)(-p[^- \_]+)([^ \_]*) (.*)}/{\1\3 \4}/ ) {}
      # clean up -n tags in first of new underscore cat...
      while ( s/{([^ \_]*)(-n[^- \_]+)([^ ]*) (.*)}/{\1\3 \4}/ ) {}
      # # clean up -n tags in rest of new underscore cat...
      # while ( s/{([^ ]*)(-n[^- \_]+)([^ \_]*) (.*)}/{\1\3\2 \4}/ ) {}

      # clean up -u tags in first of new underscore cat...
      while ( s/{([^ \_]*)(-u[^- \_]+)([^ ]*) (.*)}/{\1\3 \4}/ ) {}
      # # clean up -u tags in rest of new underscore cat...
      # while ( s/{([^ ]*)(-u[^- \_]+)([^ \_]*) (.*)}/{\1\3\2 \4}/ ) {}

      # clean up -g tags in first of new underscore cat...
      while ( s/{([^ \_]*)(-g[^- \_]+)([^ ]*) (.*)}/{\1\3\2 \4}/ ) {}
      # # clean up -g tags in rest of new underscore cat...
      # while ( s/{([^ ]*)(-g[^- \_]+)([^ \_]*) (.*)}/{\1\3\2 \4}/ ) {}
    }

    if ($PRINT_POS) {
        # create pos tags at leaves
        s/\(([^ ]*) *\{([^ ]*) ([^>\[\#]*)\}\)/{\1 \1\#\3}/;
        s/{([^ ]*) ([^ \[\#]*)}/{\1 \1\#\2}/;
    }
    ####################

    ## convert inner angles (if any) to bracks...
    while ( s/({[^{}]*)<([^<>]*)>/\1\[\2\]/ ){}

    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }

  # ??!!
  # ##################### FINALLY: delete numbers...
  # s/[-=][0-9]+([^ \)\]]*) /\1 /g;
  # s/[-=][0-9]+([^ \)\]]*) /\1 /g;
  # s/[-=][0-9]+([^ \)\]]*) /\1 /g;

  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;

  ## translate to parens again...
  $_ =~ s/\[/\(/g;
  $_ =~ s/\]/\)/g;

  print $_;
}
```

B.8 scripts/cnftrees2cedepths.rb

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################

#!/usr/bin/ruby

#########################################################################
# cnftrees2cedepths.rb
#  looks for and annotates center-embedding depths and Left-Right
#   constituency
#  only works on binary-branching trees
#
#########################################################################

require "scripts/umnlp.rb"

class Tree
  def propDown

    if @head =~ /.*\^([LR]),([0-9]+)/
      depth = $2.to_i
      side  = $1
    else
      depth = 1
      side  = "L"
      @head += "^#{side},#{depth}"
    end

    if @children.size() <= 1
      return
    end
    if side=="R"
      ldepth = depth+1
    else
      ldepth = depth
    end
    @children[0].head += "^L,#{ldepth}"
    @children[0].propDown

    if @children.size() <= 1
      return
    end
    #depth += 1
    @children[1].head += "^R,#{depth}"
```

```
    @children[1].propDown

  end
end

while(line = gets)
  t = Tree.new(line)
  t.propDown
  if t.prob == nil
    print t.to_s + "\n"
  else
    puts "#{t.to_s} : #{t.prob}"
  end
end
```

B.8.1 scripts/umnlp.rb

```
################################################################################
##                                                                            ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                            ##
##    ModelBlocks is free software: you can redistribute it and/or modify     ##
##    it under the terms of the GNU General Public License as published by    ##
##    the Free Software Foundation, either version 3 of the License, or       ##
##    (at your option) any later version.                                     ##
##                                                                            ##
##    ModelBlocks is distributed in the hope that it will be useful,          ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of          ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           ##
##    GNU General Public License for more details.                            ##
##                                                                            ##
##    You should have received a copy of the GNU General Public License       ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.    ##
##                                                                            ##
################################################################################

##########################################
## A Tree object consists of a string "head" (e.g. NP, S, JJ),
## a Tree object which is its parent (possibly nil)
## and an array of children that are Tree objects
## You initialize a tree by passing it a string
## containing parentheses delimited tree structure.
## For files containing multi-line trees (like treebank)
## you need to use the tree slurper class
##
##########################################

$disallowed = {"-NONE-" => 1, "-DFL-" => 1, "DISFL" => 1, "XX" => 1,
               "N_S" => 1, "E_S" => 1}

class Tree
  attr_reader :str, :head, :children, :num_rules, :parent, :prob
  attr_writer :str, :head, :children, :num_rules, :parent, :prob

  def initialize(str="", parent=nil)
    @str = str
    @children = Array.new
    @parent = parent
    @num_rules = 0
```

```
    @prob = nil
    if str != ""
      ## Check if the stupid user passed in a stupid string with
      ## stupid brackets instead of parentheses
      if (str.length - str.gsub(/\)/,"").length) < (str.length - str.gsub(/\]/,"").length)
        str.gsub!(/\[/,"(")
        str.gsub!(/\]/,")")
      end
      if str =~ /: ([0-9\.]+)$/
        @prob = $1
      end
      begin
        buildStructure(@str)
      rescue
        raise "Caught exception when tree #{@str} was passed in...\nError message:\n#{$!}"
      end
    else
      head = ""
    end
    if str == ""
      str = to_s
    end
    @num_rules = getNumRules #+= @children[i].num_rules
  end

  def buildStructure(str)
    if str == ""
      @head = ""
      return ""
    else
      ## First let's check if its one of those weird switchboard
      ## trees that start with 2 open parens: ( (S
      if str =~ /^ *\( *\((.*)\) *$/
        str = "(" + $1
      end

      #Pluck off the head if it's there
      if str =~ /^ *\( *([^ ()]+)/
        ## Start of a rule
        @head = $1
        str = $'#
        while true
          if str =~ /^ *\(/
            child = Tree.new("", self)
            begin
              str = child.buildStructure(str)
            rescue
              ## Catching downstream exception we'll pass it up...
              #$stderr.puts "Error caught and being passed upwards"
              raise $!
            end
            @children << child
          elsif str =~ /^ *([^ ()]+) *\)/
            ## we've reached a leaf - i.e. a word and its close paren
            child = Tree.new("", self)
            child.head = $1 #.downcase
            child.num_rules = 1
            @children << child
            str = $'#
            return str
          elsif str =~ /^ *\)/
            ## End of a tree
```

```
            str = $'#'
            return str
          else
            raise "Erroneouse part of tree: #{str}."
          end
        end
      end
    end
  end

  ## Convert a chomsky normal form (cnf) tree to a right-corner (rc) tree
  def cnf2rc
    if getNumLeafs() == 1
      return self
    end
    if @children[0].class == Tree and @children.size() == 1
#      puts "**Tree " + to_s + "... has only one child, but many leaves"
      @children[0] = @children[0].cnf2rc
      return self
    end
    curTree = nil
    next_rc = nil
    rc = nil

    ## Outer loop over all right children
    while true
      if getNumLeafs() == 1
        break
      end

      if next_rc != nil
        rc = next_rc
        next_rc = rc.parent
      else
        ## Inner loop to find current right corner
        rc = self
        while true
          if rc.getNumLeafs() == 1
            break
          end
          rc = rc.children.last
        end
        next_rc = rc.parent
      end

      if rc == self
        @children[0] = @children[0].cnf2rc
        break
      end

      ## Okay, now we have the "right corner"
      ## Get its parent, remove it, and add it to new children list
      tempParent = rc.parent
      if tempParent != nil
          tempParent.children.pop()
      end

      newBigTree = Tree.new
      newBigTree.head = String.new(@head)
      newBigTree.prepend(rc.cnf2rc)
#       rc.cnf2rc
      newBigTree.prepend(self)
```

```
      if curTree != nil
        curTree.children[0] = newBigTree
        newBigTree.parent = curTree
      end
      curTree = newBigTree

      ## Find out what we're "missing" by relocating the right corner
      if rc.head =~ /(.*)\/(.*)/
        newHead = $1
      else
        newHead = String.new(rc.head)
      end

#       puts "newHead = " + newHead
#       puts "My tree (prelim) is: " + to_s
#       puts "Curtree parent (prelim) is: " + curTree.parent.to_s
      ## Now adjust the labels in the tree to reflect the missing item
      temp = self
      while temp != nil
        if temp.head =~ /.*\/.*/
          temp.head.gsub!(/(.*)\/(.*)/, '\1/' + newHead)
        else
          temp.head += "/" + newHead
        end
        if temp.children.size() < 2
          break
        end
        temp = temp.children.last
      end

#       puts "My tree is: " + to_s
#       puts "Next rc is: " + next_rc.to_s
#       puts "Curtree is: " + curTree.to_s
#       puts "Curtree parent is: " + curTree.parent.to_s
    end

    #@head = curTree.head
    #@children = curTree.children
    while curTree.parent != nil
      curTree = curTree.parent
    end
    return curTree


  end

  def rc2cnf

    if getNumLeafs == 1
        return self #@children[0]
    end
#   begin
#     puts "rc2cnf called with:"
#     puts to_s + "\n\n"

    r_sub = self.children[1]
    cur_tree = self.children[0]
    while true
      if cur_tree.getNumLeafs == 1
        newTree = Tree.new
        newTree.head = @head
        newTree.children << cur_tree.children[0]
```

```
        newTree.children << r_sub
        return newTree
      end

      if cur_tree.children.size() == 1
#         $stderr.puts "I'm in here and cur_tree = " + cur_tree.to_s
#         $stderr.puts "and r_sub = " + r_sub.to_s
        newTree = Tree.new
        newTree.head = cur_tree.head.gsub(/(.*)\/(.*)/, '\1')
        newTree.children << cur_tree.children[0].rc2cnf
#         $stderr.puts "I've returned from rc2cnf on my child"
        newTree.children << r_sub
        return newTree
#         cur_tree = cur_tree.children[0]
      else
        cur_tree.children[1].parent = nil
        newTree = Tree.new
        newTree.head = cur_tree.children[0].head.gsub(/(.*)\/(.*)/, '\2')
        newTree.children << cur_tree.children[1]
        newTree.children << r_sub
        cur_tree.children[1].parent = newTree
        r_sub.parent = newTree
        r_sub = newTree
        cur_tree.children.pop()
        if r_sub.children[0].class == Tree
            r_sub.children[0] = r_sub.children[0].rc2cnf
        end
        cur_tree = cur_tree.children[0]
      end
    end
    return cur_tree
#   rescue
#     puts "I've been rescued: cur_tree: " + cur_tree.to_s
#     puts "     and r_sub: " + r_sub.to_s
#     return nil
#   end
  end

  def nominalBinarize!()
    ra = Array.new
    cats = Array.new
    if @children.size > 2
      @children.each{ |child|
        if not $disallowed.has_key?(child.head)
          ra << child
          cats << child.head
        end
      }
    end

    if ra.size > 2
      nt = Tree.new
      nt.head = cats[1..-1].join("_")
      nt.children = ra[1..-1]
      @children[1] = nt
      @children.slice!(2..-1)
    end
    @children.each{ |child|
      child.nominalBinarize!
    }
  end
```

```
  def nominalUnbinarize!()
    if @children[0] == nil
      return
    elsif @children[0].head =~ /_/
      left_children = @children[0].children
    else
      left_children = @children[0]
    end

    if @children[1] == nil
      right_children = nil
    elsif @children[1].head =~ /_/
      right_children = @children[1].children
    else
      right_children = @children[1]
    end
    if right_children != nil
      @children = [left_children, right_children].flatten
    else
      @children = [left_children].flatten
    end
    @children.each{ |child|
      child.nominalUnbinarize!
    }
  end
  ####################################
  # unbinarize
  # Takes a treebank tree that has been binarized with the magerman head
  # rules and unbinarizes them for easy comparison to treebank gold standard
  #
  ####################################
  def unbinarize
    if getNumLeafs == 1
      return
    end
    @children.each_index { |i|
      @children[i].unbinarize
      if @children[i].head =~ /-bin/
        movin_on_up = @children[i].children
        @children.delete_at(i)
        @children.insert(i, movin_on_up)
      end
    }

  end

  def to_s
    if @children.length == 0
      return "#{head}"
    else

      s = "(#{@head} "
      @children.each{ |child|
        s += child.to_s
#         print "#{child.head}"
        #s += " "
      }
      s += ")"
    end
    return s.gsub(/\)\(/, ") (")
  end
```

```
  def prepend(t)
    @children.unshift(t)
  end

  ## getNumRules
  ## Computes the total number of rules in this tree
  ## If this tree is the gold standard, it is the denominator
  ## in the recall calculation.
  ## If this tree is the hypothesis, it is the denominator
  ## in the precision calculation.
  def getNumRules()
    if @children.size() > 1
      @num_rules = 1
#   elsif @children.size() == 1 and @children[0].children.size > 0
#     @num_rules = 1
    else
      @num_rules = 0
    end
    @children.each_index{ |i|
      @num_rules += @children[i].getNumRules
    }
    return @num_rules
  end

  ## getNumCorrect - This method treats the tree it belongs
  ## to as a gold standard, takes in a tree argument as a
  ## hypothesis, and returns the number correct, as the
  ## numerator for the labeled precision/recall calculation
  def getNumCorrect(t)
    count = 0
    gold_rules = getRulesHash
    hypoth_rules = t.getRulesArray(0)
    hypoth_rules.each{ |v|
      if (v.j - v.i) > 1 && gold_rules.has_key?(v.hash)
#     if v.first != "" && (not v.first =~ /[a-z]/)  && gold_rules.has_key?(v.hash)
        count += 1
      end
    }
    return count
  end

  def getRulesHash
    rules = getRulesArray(0)
    rules_hash = Hash.new
    rules.each{ |v|
      rules_hash[v.hash] = v
    }
    return rules_hash
  end

  def getRulesArray(start_ind)
    rules = Array.new
    ind = start_ind
    if(@children.size > 0)
      @children.each_index{ |i|
        i_rules = @children[i].getRulesArray(ind)
        if i_rules.size > 0
          rules << i_rules
          rules.flatten!
          ind = rules.last.j
        end
      }
```

```
    else
      ## Don't increment count for 0, *T*-1, and word fragments
      if not @head =~ /^[\*\d]/ and not @head =~ /\-$/
#         if @children.size() > 1
#           rules << Rule.new(@head, start_ind, start_ind+1, @children[0].head)
#         else
          rules << Rule.new(@head, start_ind, start_ind+1)
#         end
#         puts "Created new rule " + rules.last.to_s
      end
      return rules
    end
    if rules.size > 0
      end_ind = rules.last.j
    else
      end_ind = start_ind
    end
    #rules.delete_if{ |rule| (rule.j - rule.i) < 2}
    if(@children.size() > 1)
      rules << Rule.new(@head, start_ind, end_ind, @children[0].head)
    end
#    puts "Created new rule " + rules.last.to_s
    return rules
  end

  def getPrecision(t)
    return t.getNumCorrect / t.getNumRules
  end

  def getRecall(t)
    return t.getNumCorrect / self.getNumRules
  end

  def getWordString
   begin
    if @children.size == 0
      return @head
    end
    str = ""
    @children.each{ |child|
      str += child.getWordString + " "
    }
    str.gsub!(/ +/, " ")
    str.gsub!(/ $/, "")
    return str
   rescue Exception
     $stderr.puts "Exception caught when str = #{str}: " + $!
   end
  end

  def getNumLeafs
    if @children.size() == 0
      return 1
    end
    ret = 0
    @children.each{ |child|
      ret += child.getNumLeafs
    }
    return ret
  end

# get the depth of the current tree
```

```
        def getDepth
                if @children.size == 0
                        return 1
                else
                        ret = 0;
                        @children.each{ |child|
                                if child.getDepth > ret
                                        ret = child.getDepth
                                end
                        }
                        return 1 + ret
                end
        end

        # returns true if the tree includes the given word
        def include?(str)
                if(head.include?("#") and head.split("#")[1] == str)
                        return true
                else
                        @children.each { |child|
                                if(child.include?(str))
                                        return true
                                end
                        }
                        return false
                end
        end

        # returns true if the tree includes the given POS
        def posinclude?(str)
                if(head.include?("#") and (head.split("#")[0]).include?(str))
                        return true
                else
                        @children.each { |child|
                                if(child.posinclude?(str))
                                        return true
                                end
                        }
                        return false
                end
        end

        def find(str)
                rArray = Array.new
                if(head =~ /##{str}$/ or head =~ /^\S*#{str}\S*#/) then
#                       puts head
                        rArray.push(self)
                        return rArray
                elsif(@children.size == 0)
                        return nil
                else
                        @children.each { |child|
                                tmp = child.find(str)
                                if (tmp != nil) then
                                        if (tmp.class == Array) then
                                                rArray += tmp
                                        else
                                                rArray.push(tmp)
                                        end
                                end
                        }
                        if(rArray.size == 0) then
```

```
                        return nil
                else
                        return rArray
                end
        end
    end

    # gets the words in this tree that are contained in the given pos
    def getWords(pos)
        arr = Array.new
        if(head.include?("#") and (head.split("#")[0]).include?(pos)) then
                        arr[0] = head.split("#")[1]
                        return arr
        else
                @children.each { |child|
                        arr += child.getWords(pos)
                }
        end
        return arr
    end

# get the depth of the stack if it were interpreted as a right-corner tree
    def getRCStackDepth(str = "")
        if str != "" then
                if head.include?("#" + str + "␣)") or head.include?(str + "#") then
                        return self.getRCStackDepth
                elsif @children.size == 0
                        return -1
                else
                        max = 0;
                        @children.each_index{ |x|
                                depth = @children[x].getRCStackDepth(str)
                                if x != 0 then
                                        depth += 1
                                end
                                if depth > max and (@children[x].include?(str) or @children[x].posinclude?(
                                        str)) then
                                        max = depth
                                end
                        }
                        return max
                end
        elsif @children.size == 0
                return 0
        else
                max = 0;
                @children.each_index { |x|
                        depth = @children[x].getRCStackDepth
                        if x != 0 then
                                depth += 1
                        end
                        if depth > max then
                                max = depth
                        end
                }
                return max
        end
    end

# gets the depth of the stack for the path from root to leaf containing the given str
#     def getRCStackDepth(str)
#         if head.include?("#" + str + " )")
```

```
#                       return self.getRCStackDepth
#               elsif @children.size == 0
#                       return -1
#               else
#                       max = 0
#                       @children.each_index { |x|
#                               depth = @children[x].getRCStackDepth
#                               if x != 0
#                                       depth += 1
#                               end
#                               if depth > max then
#                                       max = depth
#                               end
#                       }
#                       return max
#               end
#       end

  def getOps
  end

  def getNumEditedLeafs
    if not @head =~ /\/EDITED/ and @head =~ /EDITED/
      ## We have a EDITED label
      return getNumLeafs
    else
      count = 0
      @children.each{ |child|
        count += child.getNumEditedLeafs
      }
      return count
    end
  end

  def getEditedArray
    if not @head =~ /\/EDITED/ and @head =~ /EDITED/
      ## We have a EDITED label
      return Array.new(getNumLeafs,1)
    else
      a = Array.new
      if @children.size == 0
        a[0] = 0
      else
        @children.each{ |child|
          a << child.getEditedArray
        }
      end
      return a.flatten
    end
  end

  def getNumEditedCorrect(t2)
    count = 0
    gold = getEditedArray
    hypoth = t2.getEditedArray
    hypoth.each_index{ |i|
      if hypoth[i] == 1 and gold[i] == 1
        count += 1
      end
    }
    return count
  end
```

```
end

##################################
## TreeSlurper
## This class is used to read in files
## containing trees spanning multiple
## lines.  getNext reads just to the next
## tree and returns a Tree object.
## getAll returns an array containing all
## trees in the file.  This may be prohibitively
## large for certain large corpora.
##
##################################

class TreeSlurper
  def initialize(file_name)
    @file_name = file_name
    @file = File.open(@file_name)
  end

  def getNext
    ## Get the next tree
    num_parens = 0
    str = ""
    while (line = (@file.gets)) != nil
      line.chomp!
      str += line
      num_left_parens = line.length - line.gsub(/\(/,'').length
      num_right_parens = line.length - line.gsub(/\)/,'').length
      num_parens += (num_left_parens - num_right_parens)
      if num_parens == 0 && line != ""
        break
      end
    end
    if str != ""
      return Tree.new(str)
    else
      return nil
    end
  end

  def getAll
    ## Get all trees
    ra = Array.new
    while (t = getNext) != nil
      ra << t
    end
    return ra
  end

  def close
    File.close(@file_name)
  end

end

class Rule
  attr_reader :head, :i, :j, :first

  def initialize(name, i, j, first="")
    @head = name
    @i = i
```

```
    @j = j
    @first = first
  end

  def to_s
    s = i.to_s + "␣" + @head + "␣" + j.to_s + "␣" + @first
  end

  def hash
    return @i.to_s + @head + @j.to_s
  end
end

########################
# class HypothPath
#
# Represents the output of a DBN viterbi trace, all the
# operations at each stack depth and time step.
# This is most importantly used for recreating a rc tree
# that one can compare to a gold standard for computing
# accuracy results
#
#########################

class HypothPath
  attr_reader :S, :R, :F

  def initialize(hypoth_file)
    @S = Array.new
    @R = Array.new
    @F = Array.new
    t = -1
    @file = File.open(hypoth_file, "r")
#    old_line = ""
    while (line = @file.gets) != nil
      line.chomp!
#      if line == old_line
        ## didn't recognize correctly
      if line =~ /no most likely sequence/
        if t > -1
          break
        else
          @R[0] = nil
          @F[0] = nil
          @S[0] = nil
          break
        end
      end
#     end
      if (line =~ /HYPOTH (\d+)> R: ([^ ]+) *F: ([^ ]+) *S: ([^ ]+)/)
        rs = $2
        fs = $3
        ss = $4
        t = $1.to_i
        @R[t] = rs.split(";")
        @F[t] = fs.split(";")
        @F[t] << 1          ## Add a 1 to the end
        @S[t] = ss.split(";")  # S already has the LX at the end
        if line =~ /S: null/
          break
        end
      end
```

```
#       old_line = line
    end
  end

  def nextTree
    treeSoFar = nil #Tree.new
    subTree = nil
    subsubTree = nil
    ## Iterate over time steps
    @R.each_index { |t|
      if @R[0] == nil
        treeSoFar = Tree.new
        break
      end
      if t == 0
        next
      end
      ## This is the 011 case or 111 case - the latter just being a special
      ## case of the former for the end of utterance
      if @F[t][1].to_i == 1
        ## Create a new tree at depth 0 with the old subtree as the left and the new stuff as the right
        newBigTree = Tree.new
        newBigTree.head = @R[t][0]
        ## Add the tree so far as the left child, if it exists
        if treeSoFar != nil
          newBigTree.children << treeSoFar
        end

        ## Create a new tree at depth 1 that will be the right child of the newBigTree
        newSmallTree = Tree.new
        newSmallTree.head = @R[t][1] #Tree.new("( #{@R[t][1]} )")
        ## If we've just finished a bunch of stuff at level 1, add it as the left child
        if subTree != nil
          ## FIXME? (this _is_ a test fix... not sure it's right)
          if subsubTree != nil
            superSubTree = Tree.new
            superSubTree.head = @R[t-1][1]
            superSubTree.children << subTree
            superSubTree.children << subsubTree
            subTree = superSubTree
          end
          newSmallTree.children << subTree
        end

        ## If there is any non-trivial thing going on between levels 1 and 2 in R, make a
        ## a tree out of it (trivial case is a word going to the same word), and add that
        ## tree to the
        ## Non-trivial case is NT going to another NT, or NT/word going to word basically
        if @R[t][1] =~ /[A-Z]/ || @R[t][1] != @R[t][2]
          newSmallTree.children << Tree.new("(_#{@R[t][2]}_)")
          if @R[t][2] != @S[t-1][3]
            newSmallTree.children.last.children << Tree.new("(_#{@S[t-1][3]}_)")
          end
        end
        ## Append the right hand (small) tree to the new big tree, and make the
        ## treeSoFar point at it
        newBigTree.children << newSmallTree
        if not @R[t][0].eql?(@S[t][0]) and @F[t][0].to_i == 0
          temp = Tree.new
          temp.head = @S[t][0]
          temp.children << newBigTree
          newBigTree = temp
```

```
      end
      subTree = nil
      subsubTree = nil
      treeSoFar = newBigTree
    elsif @F[t][2].to_i == 1  ## Here we have 0 0 1
      if subTree == nil
        ## Just starting a new subtree off the main tree
        subTree = Tree.new
        subTree.head = @R[t][1]
        subTree.children << Tree.new("(␣#{@R[t][2]}␣)")
        if @S[t-1][3] != @R[t][2]
          subTree.children[0].children << Tree.new("(␣#{@S[t-1][3]}␣)")
        end
      elsif subsubTree == nil
        ## Continuing a subtree
        newSubTree = Tree.new
        newSubTree.head = @R[t][1]
        newSubTree.children << subTree
        newSubTree.children << Tree.new("(␣#{@R[t][2]}␣)")
        if @S[t-1][3] != @R[t][2]
          newSubTree.children[1].children << Tree.new("(␣#{@S[t-1][3]}␣)")
        end
        subTree = newSubTree
      else ## We have to deal with a subsubtree
        newSubTree = Tree.new
        newSubTree.head = @R[t][1]
        newSubTree.children << subTree
        newSubTree.children << Tree.new("(␣#{@R[t][2]}␣)")
        newSubTree.children[1].children << subsubTree
        newsubstr = "(␣" + @S[t-1][2].gsub(/^[^\/]+\/(.*)$/, '\1') + "␣" + @S[t-1][3] + "␣)"
        newSubTree.children[1].children << Tree.new(newsubstr)
        subTree = newSubTree
      end
      if not @R[t][1].eql?(@S[t][1])
        temp = Tree.new
        temp.head = @S[t][1]
        temp.children << subTree
        subTree = temp
      end
      subsubTree = nil
    else   ## Here we have 0 0 0 for the F nodes
      if subsubTree == nil
        subsubTree = Tree.new
        subsubTree.head = @R[t][2]
        subsubTree.children << Tree.new("(␣#{@S[t-1][3]}␣)")
      else
        newsubsubTree = Tree.new
        newsubsubTree.head = @R[t][2]
        newsubsubTree.children << subsubTree;
        newsubsubTree.children << Tree.new("(␣#{@S[t-1][3]}␣)")
        subsubTree = newsubsubTree
      end
      if not @R[t][2].eql?(@S[t][2])
        temp = Tree.new
        temp.head = @S[t][2]
        temp.children << subsubTree
        subsubTree = temp
      end
    end
  }
  @R = Array.new
  @S = Array.new
```

```
    @F = Array.new
    t = -1
    ## Advance to the next tree
    while (line = @file.gets) != nil
      line.chomp!
      if line =~ /no most likely sequence/
        if t > -1
          break
        else
          @R[0] = nil
          @F[0] = nil
          @S[0] = nil
          break
        end
#         break
      end
      if (line =~ /HYPOTH (\d+)> R: ([^ ]+) *F: ([^ ]+) *S: ([^ ]+)/)
        rs = $2
        fs = $3
        ss = $4
        t = $1.to_i
        @R[t] = rs.split(";")
        @F[t] = fs.split(";")
        @F[t] << 1           ## Add a 1 to the end
        @S[t] = ss.split(";")  # S already has the LX at the end
        if line =~ /S: null/
          break
        end
      end
    end
    return treeSoFar
  end
end


class ValueProb
  attr_writer :value, :prob
  attr_reader :value, :prob

  def initialize(v,p)
    @value = v
    @prob = p
  end
end

class CounterHash < Hash
  def [] (key)
    if not has_key?(key)
      store(key, 0)
    end
    super
  end
end

class ArrayHash < Hash
  def initialize
    super
    @max_val = 0
  end

  def [] (key)
    if not has_key?(key)
```

```
      @max_val += 1
      store(key, @max_val)
    end
    super
  end
end
```

B.9 scripts/trees2rules.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################

use Getopt::Std;

getopts("p");

$PWDT = 0;
if($opt_p){
  $PWDT = 1;
}


########## FOR TREES IN GENERAL...
#print "GG ROOT : S eos\n";
##########

## for each tree...
while (<>) {

  ########## FOR TREES IN GENERAL...
  print "C : ROOT\n";
  ##########

  ## translate to parens...
  s/\[/\(/g;
  s/\]/\)/g;

  ## for each constituent...
  while ($_ =~ /\([^\(\)]*\)/) {

    ## convert outer parens to braces...
```

```
    $_ =~ s/\(([^\(\)]*)\)/{\1}/;

    ########## ADD SED RULES HERE: apply rules to angles (children) within braces (consituent)...
    #print stderr "A== $_";
    ## if terminal branch...
    if ( ($c,$p,$w) = ($_ =~ /{([^ ]+) +([^ ]+)\#([^ ]+) *}/) ) {
#       $c=lc($c);
      s/{([^ ]*) +([^ <]*\#[^ <}]*)}/{$c $p\#$w}/;
      print "C: $c\n";
      print "CC $c : --\n";
      if ($PWDT) {
        print "Pc $c : $p\n";
        print "Pw $w : $p\n";
        print "P : $p\n";
        #print "W : $w\n";
        #print "PW $p : $c\#$w\n";
      } else {
        print "X $c : $w\n";
      }
    }
    ## for version without pos's (so without # symbol)...
    elsif ( ($c,$w) = ($_ =~ /{([^ ]+) +([^ ]+) *}/) ) {
      $p = $c;
      s/{([^ ]*) +([^ <}]*)}/{$c $p\#$w}/;
      print "C: $c\n";
      print "CC $c : --\n";
      if ($PWDT) {
        print "Pc $c : $p\n";
        print "Pw $w : $p\n";
        print "P : $p\n";
      } else {
        print "X $c : $w\n";
      }
    }
    ## if nonterminal branch...
    if ( ($c,$cc) = ($_ =~ /{([^ ]*)(.*<[^ ]*) [^<]*}/) ) {
      #print "$p$cc\n";
      print "C: $c\n";
      @A = split ( / [^<]*</, $cc );
      print "CC $c :@A\n";
    }
    ##########

    ## convert inner angles (if any) to bracks...
    while ($_ =~ /{[^{}]*</) {
      $_ =~ s/({[^{}]*)<([^<>]*)>/\1\[\2\]/;
    }

    ## convert outer braces to angles...
    $_ =~ s/{(.*)}/<\1>/;
  }

  if ( ($c) = ($_ =~ /<([^ ]*) /) ) {
    print "Cr: $c\n";
  }

  ## finish up...
  $_ =~ s/</[/;
  $_ =~ s/>/]/;

  ## translate to parens again...
  #$_ =~ s/\[/\(/g;
```

```
  #$_ =~ s/\]/\)/g;

#  print $_;
}
```

B.9.1 scripts/relfreq.pl

```
##############################################################################
##                                                                          ##
## This file is part of ModelBlocks. Copyright 2009, ModelBlocks developers. ##
##                                                                          ##
##    ModelBlocks is free software: you can redistribute it and/or modify   ##
##    it under the terms of the GNU General Public License as published by  ##
##    the Free Software Foundation, either version 3 of the License, or     ##
##    (at your option) any later version.                                   ##
##                                                                          ##
##    ModelBlocks is distributed in the hope that it will be useful,        ##
##    but WITHOUT ANY WARRANTY; without even the implied warranty of        ##
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         ##
##    GNU General Public License for more details.                          ##
##                                                                          ##
##    You should have received a copy of the GNU General Public License     ##
##    along with ModelBlocks.  If not, see <http://www.gnu.org/licenses/>.  ##
##                                                                          ##
##############################################################################


use Getopt::Std;

getopts('c:lfr:');
if (defined($opt_c)) {
  $mincount = $opt_c;
} else {
  $mincount = 0.0;
}
if (defined($opt_l)) {
  $longout = $opt_l;
} else {
  $longout = 0;
}
if (defined($opt_f)) {
  $rawfreq = $opt_f;
} else {
  $rawfreq = 0;
}
if (defined($opt_r)) {          # don't do blind cutoffs; replace w/ symb
  $replace = $opt_r;
  $newcutoff = 1;
} else {
  $newcutoff = 0;
}

print stderr "mincount = $mincount\n";

# for the replace version
sub replace {
  @args = @_;
  $lowct = $args[0];
```

```
  # get the substitute string ready
  @parts = split(/[ :;\{\}\/,~\(\)\<\>\[\]]+/,$lowct);
  @delims = $lowct =~ /([ :;\{\}\/,~\(\)\<\>\[\]]+)/g;
  @replacers = ( ($replace) x ($#parts+1));
  @combined = map { $replacers[$_], $delims[$_] }
    0 .. ($#replacers > $#delims ? $#replacers : $#delims);
  $r = join("",@combined);
  return $r;
}

while ( <> ) {
  chomp;

  if ( $_ =~ /(.*) : (.*) = (.*)/ ) {
    if (!exists($Cond{$1})) {
      $Cond{$1} = 0.0;
    }
    if (!exists($Genv{$2})) {
      $Genv{$2} = 0.0;
    }
    if (!exists($Val{$1}{$2})) {
      $Val{$1}{$2} = 0.0;
    }
    $Cond{$1} += $3;
    @targs = split(/ +/,$2);
    foreach $t (@targs) {
      $Genv{$t} += $3;
    }
    $Val{$1}{$2} += $3;
  } elsif ( $_ =~ /(.*) : (.*)/ ) {
    $Cond{$1}++;
    @targs = split(/ +/,$2);
    foreach $t (@targs) {
      $Genv{$t}++;
    }
    $Val{$1}{$2}++;
    #print "$1|$2|\n";
  }
}

# don't replace, do blind cutoffs
if ($newcutoff==0) {

  ## Special case for prior distribution:
  foreach $c (keys %Cond) {
    if (not ($c =~ /^[^ ]$/)) {
      next;
    }
    foreach $v (keys %{$Val{$c}}) {
      if ($Val{$c}{$v} < $mincount) {
        $Cond{$c} -= $Val{$c}{$v};
        delete $Val{$c}{$v};
      }
    }
  }

}

# replace whole low counts
else {

  # make a pass to replace low counts
```

```
  if ($mincount>0) {

    foreach $lhs (sort keys %Val) {

      # check on lhs for replaceables
      my @newlwords;
      @lwords = split(/ +/,$lhs);
      my $ctr=0; my $lflag;
      foreach $lword (@lwords) {
        if ( ($Genv{$lword}<$mincount) && ($lword ne replace($lword)) && $ctr!=0) {
          $lflag = 1;
          push( @newlwords, replace($lword) );
        } else {
          push( @newlwords, $lword );
        }
        $ctr++;
      }
      $newlhs = join('␣',@newlwords);

      # check on rhs
      foreach $rhs (sort keys %{$Val{$lhs}}) {
        my @newrwords; my $rflag;
        @rwords = split(/ +/,$rhs);
        foreach $rword (@rwords) {
          if ( $Genv{$rword}<$mincount && $rword ne replace($rword) ) {
            $rflag = 1;
            push( @newrwords, replace($rword) );
            $Genv{$rhs}
          } else {
            push( @newrwords, $rword );
          }
        }
        $newrhs = join('␣',@newrwords);

        # replace rhs targets
        if ($lflag && $rflag) {
          $Val{$newlhs}{$newrhs} += $Val{$lhs}{$rhs};
          $Cond{$newlhs} += $Val{$lhs}{$rhs};
          delete $Val{$lhs}{$rhs};
        } elsif ($rflag) {
          #             print " newrhs $newrhs from $lhs -> $rhs\n";
          $Val{$newlhs}{$newrhs} += $Val{$lhs}{$rhs};
          delete $Val{$lhs}{$rhs};
        } elsif ($lflag) {
          #             print " newlhs $newlhs from $lhs\n";
          $Val{$newlhs}{$rhs} += $Val{$lhs}{$rhs};
          $Cond{$newlhs} += $Val{$lhs}{$rhs};
          delete $Val{$lhs}{$rhs};
        }
      }
    }
  }


}



foreach $c (sort keys %Cond) {
  if ( $Cond{$c} >= $mincount ) {
    foreach $v (sort keys %{$Val{$c}}) {
      #print STDERR "count = $Cond{$c}\n";
```

```
      if ($Cond{$c} == 0.0) {
        print STDERR "Divide by zero encountered with:\n cond=$c\n";
        last;
      }
      print "$c : $v = ";
      my $valtoprint = $rawfreq ? $Val{$c}{$v} : $Val{$c}{$v}/$Cond{$c};
      if (defined($opt_l)) {
        printf( "%.12f\n", $valtoprint);
      } elsif (!defined($opt_f)) {
        printf( "%.8f\n", $valtoprint);
      } else {
        print "$valtoprint\n";
      }
    }
  }
}
```

B.10 scripts/calc-cfp-hhmm.py

```
import re
import sys
#from copy import deepcopy
from model import Model, CondModel

########################################
#
#  main
#
########################################

########## read rule counts

# init relevant models
Pc = CondModel('Pc')
Pw = CondModel('Pw')
P  = Model('P')
W  = Model('W')
Ch0_giv_BL_D_Ch = CondModel('C0_L')
Ch0_giv_BR_D_Ch = CondModel('C0_R')
Ch1_giv_BL_D_Ch = CondModel('C1_L')
Ch1_giv_BR_D_Ch = CondModel('C1_R')
Ch0_Ch1_giv_BL_D_Ch = CondModel('CC_L')
Ch0_Ch1_giv_BR_D_Ch = CondModel('CC_R')

# read in CC model and obtain relevant models
for s in sys.stdin:
    m = re.search('P[gc] (.*)\^.,. : (.*) = (.*)',s)
    if m is not None:
        Pc[m.group(1)][m.group(2)] = float(m.group(3))
    m = re.search('Pw (.*) : (.*) = (.*)',s)
    if m is not None:
        Pw[m.group(1)][m.group(2)] = float(m.group(3))
        P[m.group(2)] += float(m.group(3))
        #W[m.group(1)] += float(m.group(3))
    W.read(s)
    s = re.sub('^Cr : (.*\^[Ll],1) =','CC_REST^R,0 : \\1_REST^R,0 =',s)
```

```
    s = re.sub('␣—␣—␣','␣—^.,.␣—^.,.␣',s)
    m = re.search('CC␣(.*)\^(.),(.)␣:␣(.*)\^.,.␣(.*)\^.,.␣=␣(.*)',s)
    if m is not None:
        (ch, b,sd, ch0, ch1, ct) = m.groups()
        d = int(sd)
        if b=='L' or b=='l':
            Ch0_giv_BL_D_Ch[d,ch][ch0] += float(ct)
            Ch1_giv_BL_D_Ch[d,ch][ch1] += float(ct)
            Ch0_Ch1_giv_BL_D_Ch[d,ch][ch0,ch1] += float(ct)
        else:
            Ch0_giv_BR_D_Ch[d+1,ch][ch0] += float(ct)
            Ch1_giv_BR_D_Ch[d,ch][ch1] += float(ct)
            Ch0_Ch1_giv_BR_D_Ch[d,ch][ch0,ch1] += float(ct)

# normalize models
Ch0_giv_BL_D_Ch.normalize()
Ch0_giv_BR_D_Ch.normalize()
Ch1_giv_BL_D_Ch.normalize()
Ch1_giv_BR_D_Ch.normalize()
Ch0_Ch1_giv_BL_D_Ch.normalize()
Ch0_Ch1_giv_BR_D_Ch.normalize()
Pc.normalize()
Pw.normalize()
P.normalize()
W.normalize()

Pc.write()
Pw.write()
P.write()
W.write()

Pc.clear()
Pw.clear()
P.clear()
W.clear()

sys.stderr.write('1\n')


########## obtain intermediate models

# define iteration constant
K=20

# obtain expected counts for unbounded left descendants
Chi_giv_D_Ch_zero = Ch0_giv_BR_D_Ch
Chi_giv_D_Ch_curr = Chi_giv_D_Ch_zero
Chi_giv_D_Ch_star = CondModel('Crl*')
# add zero iteration to star model
for d,ch in Chi_giv_D_Ch_zero:
    for chi in Chi_giv_D_Ch_zero[d,ch]:
        Chi_giv_D_Ch_star[d,ch][chi] = Chi_giv_D_Ch_zero[d,ch][chi]
# add subsequent iterations to star model
for k in range(1,K+1):
    sys.stderr.write('k='+str(k)+'/'+str(K)+'\n')
    Chi_giv_D_Ch_prev = Chi_giv_D_Ch_curr
    Chi_giv_D_Ch_curr = CondModel('Crl*_k')
    for d,ch in Chi_giv_D_Ch_zero:
        for chi in Chi_giv_D_Ch_prev[d,ch]:
            for chi0 in Ch0_giv_BL_D_Ch[d,chi]:
                pr = Chi_giv_D_Ch_prev[d,ch][chi] * Ch0_giv_BL_D_Ch[d,chi][chi0]
                Chi_giv_D_Ch_curr[d,ch][chi0] += pr
```

```
                Chi_giv_D_Ch_star[d,ch][chi0] += pr

sys.stderr.write('2\n')


########## obtain hhmm models

# obtain expansion model
Ce = CondModel('Ce')
for d,ch in Chi_giv_D_Ch_star:
    pr = Ch0_giv_BR_D_Ch[d,ch].get('-')
    if pr > 0.0:
        Ce[d,ch][ch] += pr
    for chi in Chi_giv_D_Ch_star[d,ch]:
        pr = Chi_giv_D_Ch_star[d,ch][chi] * Ch0_giv_BL_D_Ch[d,chi].get('-')
        if pr > 0.0:
            Ce[d,ch][chi] += pr

Ce.normalize()
Ce.write()
Ce.clear()

sys.stderr.write('3\n')


# obtain reduction model
Fr = CondModel('F')
for d,ch in Chi_giv_D_Ch_star:
    pr = Ch0_giv_BR_D_Ch[d,ch].get('-')
    if pr > 0.0:
        Fr[d,ch,'_',ch]['1,_'] += pr
    for chi in Chi_giv_D_Ch_star[d,ch]:
        prL = Ch0_giv_BL_D_Ch[d,chi].get('-')
        pr = Chi_giv_D_Ch_zero[d,ch][chi]
        if pr > 0.0:
            Fr[d,ch,chi,'_']['1,'+chi] += pr
            if prL > 0.0:
                Fr[d,ch,'_',chi]['1,'+chi] += pr * prL
        pr = (Chi_giv_D_Ch_star[d,ch][chi] - Chi_giv_D_Ch_zero[d,ch][chi])
        if pr > 0.0:
            Fr[d,ch,chi,'_']['0,'+chi] += pr
            if prL > 0.0:
                Fr[d,ch,'_',chi]['0,'+chi] += pr * prL

Fr.normalize()
Fr.write()
Fr.clear()

sys.stderr.write('4\n')


# obtain active transition models
Chi_giv_D_Ch_Chi0    = CondModel('Ctaa')
Chi1_giv_D_Chi_Chi0 = CondModel('Ctaw')
for d,ch in Chi_giv_D_Ch_star:
    for chi in Chi_giv_D_Ch_star[d,ch]:
        for chi0,chi1 in Ch0_Ch1_giv_BL_D_Ch[d,chi]:
            if chi0 != '-':
                pr = Chi_giv_D_Ch_star[d,ch][chi] * Ch0_Ch1_giv_BL_D_Ch[d,chi][chi0,chi1]
                if pr > 0.0:
                    Chi_giv_D_Ch_Chi0[d,ch,chi0][chi]      += pr
                    Chi1_giv_D_Chi_Chi0[d,chi,chi0][chi1] += pr
```

```
Chi_giv_D_Ch_Chi0.normalize()
Chi_giv_D_Ch_Chi0.write()
Chi1_giv_D_Chi_Chi0.normalize()
Chi1_giv_D_Chi_Chi0.write()
Chi_giv_D_Ch_Chi0.clear()
Chi1_giv_D_Chi_Chi0.clear()

sys.stderr.write('5\n')


# obtain awaited transition models
Chim1_giv_D_Chim_Chim0 = CondModel('Ctww')
for d,chim in Ch0_Ch1_giv_BR_D_Ch:
    if d>0:
        for chim0,chim1 in Ch0_Ch1_giv_BR_D_Ch[d,chim]:
            if chim0 != '-':
                pr = Ch0_Ch1_giv_BR_D_Ch[d,chim][chim0,chim1]
                if pr > 0.0:
                    Chim1_giv_D_Chim_Chim0[d,chim,chim0][chim1] += pr

Chim1_giv_D_Chim_Chim0.normalize()
Chim1_giv_D_Chim_Chim0.write()
```