Tree-Structured Models of Multitext: Theory, Design and Experiments

by

Benjamin P. Wellington

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
May, 2007

I. Dan Melamed

Dedication

In memory of my brother Dean Wellington

Acknowledgments

While there are many people I would like to acknowledge, the first is my advisor, I. Dan Melamed. He worked very closely with me over my time at NYU and has always shown a true commitment to the success of his students. He introduced me to his peers in the field, helped me find internships and taught me, with calculated patience, how to become a solid researcher. He has given me a gift that I will always have with me, and I will be forever grateful for that.

My family has always been a driving force for me, and I would not have finished my Ph.D. if I did not have siblings to compete with for who can attend school the longest. (I am not sure that I won, but at least I gave it my all.) My mother was the one to call when things weren't working out just as I planned, or I wasn't sure if I had made some big mistake. She somehow always knows how to make me feel like I did everything right. My dad would give me advice about things he knew absolutely nothing about, acting as if he were the world expert. (He still calls me a "button pusher" when I use computers.) But somehow a lot of his advice made sense. Jon expressed enough pride in what I was doing to fill the Hudson and East Rivers. My parents have always fully supported my decision to pursue a Ph.D. with love and affection. My sister Abby and my brother Brad have also been more supportive than any one could ask. My sister was always someone I could turn to. And it was my brother who got me into computers in the first place. I will always remember the classes I took with him at Courant. James Leslie has always been a source of inspiration and a steady friend here in the big city.

At Bucknell University, I had my first real dose of computer science. My best friend in college, Andrew Colombi, is owed thanks, because he set the benchmark for me to keep up with when it came to science classes. In the computer science department, professors Jerry Mead and Steve Guattery will always stand out as amazing teachers and true role models. In the math department, Tom Cassidy, Pam Gorkin, Karl Voss and Joshua Lansky stick out in my mind as the professors from whom I learned the most. All of these professors helped make Bucknell the perfect place to prepare for my studies at NYU.

I would also like to thank those who coauthored some of the papers that eventually played a

large role in this dissertation. Giorgio Satta was instrumental in formalizing GMTG and Sonjia Waxmonsky did a wonderful job integrating discontinuous charts into the GenPar toolkit, which was fundamental in some of the work within. I also bounced ideas around with Chris Pike, a fellow grad student who contributed efforts to the toolkit. I am also grateful to all of the members of the JHU summer workshop in 2005, who contributed to the GenPar toolkit.

Here at NYU, I would like to thank the members of my thesis committee for their helpful feedback and early morning rise on my defense day: Ralph Grishman, Satoshi Sekine, Mehryar Mohri and Ernest Davis. In addition, Leslie Cerve has gone out of her way many times to make life around the office both more efficient and fun. Ali Argyle set up the infrastructure that I used to do this research. Fellow grad students Elif Tosun and Harper Langston have always been a joy to visit and procrastinate with on the floors above, and they helped me get through the early years here when the scariest thing ahead was an algorithms test. And of course, I would like to thank the present and past members of Proteus, especially Heng Ji, Yusuke Shinyama, Svetlana Stenchikova and Adam Meyers, for their advice and feedback over the years. I would also like to express gratitude to my research "sibling" Joseph Turian, who showed me how to push boundaries, both with his work and with his wardrobe.

Of course, time away from research is as valuable to research as research itself (please don't quote me on that), and nothing at NYU has given me more pleasure than spending time with my young friends in Dangerbox. Throughout the years, when stress levels would peak, I could always look forward to a rehearsal that would often make me laugh until I cried. I want to thank those who trusted me as a coach (Anna Drezen, Andrew Farmer, Spencer Novich and Emily Schmidt), because that turned out to be the teaching experience I valued most during my studies. I also will never forget working on stage with my good friends Andy Lavender, Tim Manley, Erik Nevala-Lee, Nick Packard, Raphaela Weissman and Bennett Williamson. Our work together allowed me to give presentations at conferences armed with the experience of being in front of large audiences in unsure situations. They were also a constant source of support to me throughout the program. Lastly, I will never forget working with Ben Weber to keep the comedy wheels turning and to navigate the mine fields of NYU administrivia (a valuable skill in itself)

and more generally life.

Lastly, I would like to thank Meghan, for her friendship, love, and encouragement, without which I would not have finished this work. Star.

This research was supported by NSF grant #'s 0238406 and 0415933.

Abstract

Statistical machine translation (SMT) systems use empirical models to simulate the act of human translation between language pairs. This dissertation surveys the ability of currently popular syntax-aware SMT systems to model real-world multitext, and shows different types of linguistic phenomena occurring in natural language translation that these popular systems cannot capture. It then proposes a new grammar formalism, Generalized Multitext Grammar (GMTG), and a generalization of Chomsky Normal Form, that allows us to build an efficient SMT system using previously developed parsing techniques. The dissertation addresses many software engineering issues that arise when doing syntax-based SMT using large corpora and lays out a object-oriented design for a translation toolkit. Using the toolkit, we show that a tree-transduction based SMT system, which uses modern machine learning algorithms, outperforms a generative baseline.

Table of Contents

List of Figures

List of Tables

Introduction

Statistical machine translation (SMT) systems use empirical models to simulate the act of human translation between languages. Over the last twenty years, interest has shifted from manually created machine translation systems to statistics-based ones. In the manually created systems, sentences in the source language are analyzed by rules created and compiled by linguists. The rules take into account semantic, lexical and syntactic knowledge about the source sentences in order to predict the correct translation. However, this approach is very time intensive since it requires language experts to form all the rules for each system.

Statistical models gained more interest after a group at IBM (Brown et al., 1988) used French-English translation pairs as a training corpus and tried to estimate parameters of a "noisy channel model" to do translation. However, the translation model was based only on words, meaning that aside from a target language model, the model predicted the translation of each word independently of its **context**, i.e., any information that accompanies that word in the input. Of course, this restriction limits the quality of the translation obtained, and so over the last several years, researchers have turned increasingly to **phrases**, which are sequences of words. Instead of predicting the translation of each word in the input, these phrase-based models predict the translation of each phrase in the input. But as was the case with the word-based models, the phrase-based models do not take contextual information about each phrase into account.

Over the last decade, a significant amount of interest has been shown in syntax-based statistical models (Wu, 1997; Gildea, 2003; Chiang, 2005), which can automatically capture some of the same contextual information that the rule-based systems can, to make translation predictions. For example, Yamada and Knight (2001) created an English-French translation system based on a probabilistic model using pre-existing English parses and part-of-speech taggers. However, when dealing with large numbers of feature types, it becomes very difficult to tell a generative story to explain the model. So in this work we turn to machine learning techniques that can handle arbitrary features, and that do not need to account for them in a generative story.

Melamed and Wang (2005) introduced Statistical Machine Translation by Parsing, a high-

level architecture for a statistical machine translation system that has at its core a single abstract parsing algorithm. The goal of this dissertation is to utilize this architecture in order to motivate, design, and empirically test a novel statistical machine translation system that uses tree-structured models.

Texts that are translations of each other are called **multitexts**. In order to motivate the work, we start by exploring the complexity of real-world multitexts and calculate what percentage of the sentence pairs in these corpora can be explained by previously published models. The prevailing view in the literature has been that there is no need for more expressive translation models because complex translation patterns rarely occur. Contrary to prevailing opinion, we show here the need for a more complex formalism to model multitext.

We propose a new grammar formalism called Generalized Multitext Grammar (GMTG) that can model interesting linguistic and cross-linguistic phenomena, and we explore the formal aspects of the formalism. GMTG, which is a generalization of Context Free Grammar, is both expressive and easy to understand. We also introduce a new normal form for GMTG, called Generalized Chomsky Normal Form, which simplifies the design for SMT-by-parsing systems.

Using the new grammar formalism and its associated normal form, we present the design for a generic toolkit for SMT-by-parsing called GenPar. The toolkit is unique in that it is designed in a highly object oriented manner to allow extensibility and configurability. Throughout its implementation, we experienced some difficulties dealing with large data sets and complex algorithms. We give insight on some of the most important implementation and engineering problems we solved, hopefully shedding light for others building their own SMT systems. This critical area of SMT research has seldom been discussed in the literature.

To score our translation candidates, we turned to the machine learning literature and consider building translation models using perceptron algorithms, support vector machines, maximum entropy models and a recently developed boosted decision tree algorithm (Turian et al., 2006), choosing the latter for it's scalability. To take advantage of recent machine learning advances, we introduce here the idea of performing SMT by learning tree-structures with discriminative models. Since the machine learning itself is not the focus of this dissertation, we designed our software so

that it can interact with any out of the box machine learning toolkit that accepts feature vectors as input. Lastly, we present experiments and results using these new discriminative models.

The dissertation consists of the following chapters. Chapter 1 looks at the complexity of word alignments and the types of models needed to capture certain linguistic phenomena in real-world multitexts. Chapter 2 formally describes Generalized Multitext Grammar, a synchronous grammar formalism that can model some of the complex alignments that other grammars can't. The chapter describes in detail where GMTG falls in a formal grammar hierarchy and proposes a new normal form. In Chapter 3, we lay out the design for performing machine translation using a single core parsing algorithm, adding many details to the work of Melamed and Wang (2005). Chapter 4 describes experiments run using GenPar and the results of those experiments. Appendix A describes the object oriented design of GenPar, a toolkit for SMT by Parsing.

1
Motivation: Coverage

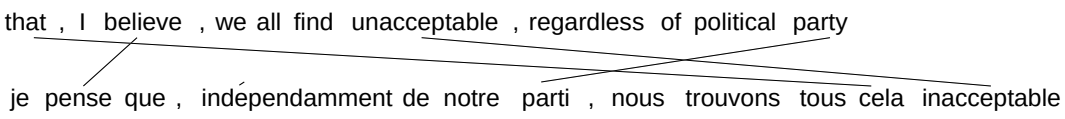

Figure 1.1: Part of a word alignment

Translational equivalence is a mathematical relation that holds between linguistic expressions with the same meaning. The most common explicit representations of this relation are word alignments between sentences that are translations of each other. The complexity of a given word alignment can be measured by the difficulty of decomposing it into its atomic units under certain constraints detailed in Section 1.1. This chapter, an extension of Wellington et al. (2006b), describes a study of the distribution of alignment complexity in a variety of multitexts. It motivates this dissertation showing that currently popular grammar formalisms cannot handle many common linguistic phenomena, and so there is a need for models with greater explanatory power. The study considers word alignments both in isolation and in combination with independently generated parse trees for one or both sentences in each pair. Thus, the study shows the modeling limits of finite-state phrase-based models that use no parse trees (Koehn et al., 2003), tree-to-string models that rely on one parse tree (Yamada and Knight, 2001), and tree-to-tree models that rely on two parse trees (Groves et al., 2004, e.g.).

The word alignments that are the least complex on our measure coincide with those that can be generated by Inversion Transduction Grammar (ITG). Following Wu (1997), the prevailing opinion in the research community has been that more complex patterns of word alignment in real multitexts are rare and mostly attributable to alignment errors. However, the experiments in Section 1.2 show that more complex patterns occur surprisingly often even in manual alignments in relatively simple multitexts. As discussed in Section 1.3, these findings shed new light on why "syntactic" constraints have rarely helped to improve the accuracy of statistical machine translation.

Our study used two kinds of data, each controlling a different confounding variable. First,

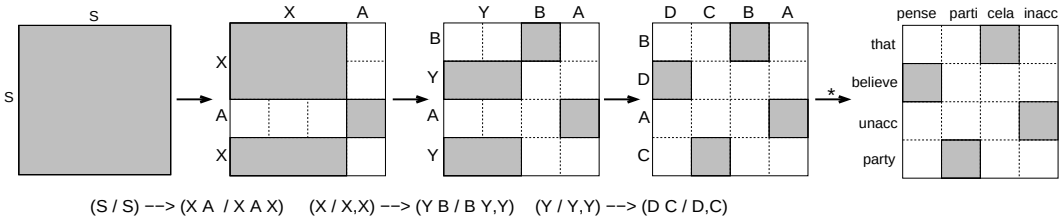

Figure 1.2: Derivation of this French/English word alignment using only binary and nullary productions requires one gap per nonterminal, indicated by commas in the production rules.

Figure 1.3: A Chinese/English word alignment that can't be hierarchically aligned without gaps.

we wanted to study alignments that contained as few errors as possible. So unlike some other studies (Zens and Ney, 2003; Zhang et al., 2006; Wu et al., 2006), we used manually annotated alignments instead of automatically generated ones. The results of our experiments on these data will remain relevant regardless of improvements in technology for automatic word alignment.

Second, we wanted to measure how much of the complexity is not attributable to systematic translation divergences, both in the languages as a whole (SVO vs. SOV), and in specific constructions (English *not* vs. French *ne... pas*). To eliminate this cause of complexity, we used English/English multitexts. We are not aware of any previous studies of word alignments in monolingual multitexts.

Even manually annotated word alignments vary in their reliability. For example, annotators sometimes link many words in one sentence to many words in the other, instead of making the effort to tease apart more fine-grained distinctions. A study of such word alignments might say more about the annotation process than about the translational equivalence relation in the data. The inevitable noise in the data motivated us to focus on lower bounds, complementary to Fox (2002), who wrote that her results "should be looked on as more of an upper bound." (p. 307) As explained in Section 1.2, we modified all unreliable alignments so that they cannot increase the complexity measure. Thus, we arrived at complexity measurements that were underestimates,

but reliably so. It is almost certain that the true complexity of translational equivalence is higher than what we report.

1.1 A Measure of Alignment Complexity

It is easy for a translation model to memorize a training sentence pair as a unit. For example, given a sentence pair like *(he left slowly / slowly he left)* with the correct word alignment, a phrase-based translation model can add a single 3-word biphrase to its phrase table. However, this biphrase would not help the model predict translations of the individual words in it. That's why phrase-based models typically decompose such training examples into their sub-biphrases and remember them too. Decomposing the translational equivalence relations in the training data into smaller units of knowledge can improve a model's ability to generalize. In the limit, to maximize the chances of covering arbitrary new data, a model should decompose the training data into the smallest possible units, and learn from them.[1] For phrase-based models, this stipulation implies phrases of length one. If the model is a synchronous rewriting system, then it should be able to generate every training sentence pair as the yield of a *binary*-branching synchronous derivation tree, where every word-to-word link is generated by a different derivation step. For example, a model that uses production rules could generate the previous example using the following synchronous productions[2]:

$$[(S),(S)] \Rightarrow [(X^1Y^2),(Y^2X^1)] \quad (1.1)$$
$$[(X),(X)] \Rightarrow [(U^1V^2),(U^1V^2)] \quad (1.2)$$
$$[(Y),(Y)] \Rightarrow [(slowly),(slowly)] \quad (1.3)$$
$$[(U),(U)] \Rightarrow [(he),(he)] \quad (1.4)$$
$$[(V),(V)] \Rightarrow [(left),(left)] \quad (1.5)$$

A problem arises when this kind of decomposition is attempted for the alignment pattern

[1] Many popular models learn from larger units at the same time, but the size of the smallest learnable unit is what's important for our purposes.

[2] An explanation of this notation can be found in Section 2.2 on page 24.

multitext	# SPs	min	median	max	95% C.I.
Chinese/English	491	4	24	52	.02
Romanian/English	200	2	19	76	.03
Hindi/English	90	1	10	40	.04
Spanish/English	199	4	23	49	.03
French/English	447	2	15	29	.01
Eng/Eng MTEval	5253	2	26	92	.01
Eng/Eng fiction	6263	2	15	97	.01

Table 1.1: Number of sentence pairs and minimum/median/maximum sentence lengths in each multitext. All failure rates reported later have a 95% confidence interval that is no wider than the value shown for each multitext.

in Figure 1.1. If each link is represented by its own nonterminal, and production rules must be binary-branching, then some of the nonterminals involved in generating this alignment need discontinuities, or **gaps**. Figure 1.2 illustrates how to generate the sentence pair and its word alignment in this manner. The nonterminals X and Y have one discontinuity each. There is no way to generate this example without discontinuous nonterminals. A similar problem would arise with the English/Chinese alignment in Figure 1.3. We discuss gaps in more detail in Chapter 2.

More generally, for any positive integer k, it is possible to construct a word alignment that cannot be generated using binary production rules with fewer than k gaps (Satta and Peserico, 2005). Our study measured the complexity of a given word alignment as the minimum number of gaps needed to generate it under the following constraints:

- each word-to-word link is generated from a separate nonterminal, and
- each step of the derivation generates no more than two different nonterminals.

Our measure of alignment complexity is analogous to what is described as "fan-out" in Chapter 2, which is a property of grammars that has been extensively studied even for monolingual grammars (Rambow and Satta, 1999). As we shall see, for grammars that generate multitexts, fan-out is equal to the maximum number of allowed gaps plus two. The least complex alignments on this measure — those that can be generated with zero gaps, with a fan-out of 2 — are precisely those that can be generated by an Inversion Transduction Grammar (Wu, 1997). For the rest of the chapter, we restrict our attention to binary derivations, except where explicitly noted otherwise.

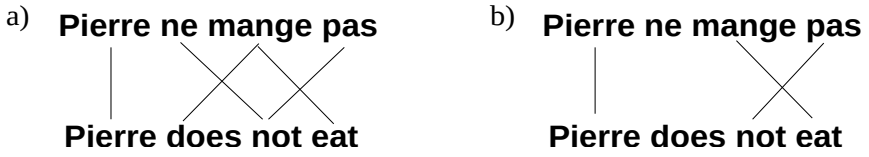

Figure 1.4: (a) A word alignment that should be modeled with discontinuities. (b) A possible word alignment left over after hierarchical alignment, since the algorithm treats links disjunctively.

To measure the number of gaps needed to generate a given word alignment, we used a hierarchical alignment algorithm to infer a binary synchronous parse tree that was consistent with the alignment, using as few gaps as possible. A hierarchical alignment algorithm is a type of synchronous parser, but instead of constraining inferences by the production rules of a grammar, the constraints come from word alignments and possibly other sources. The algorithm begins with word-to-word links as constituents. It then repeatedly composes constituents with other constituents to make larger ones, trying to find a constituent that covers the entire input. Implementation details are described in Chapter 3.

One of the important design choices in this kind of study is how to treat multiple links attached to the same word token. Word aligners, both human and automatic, are often inconsistent about whether they intend such sets of links to be disjunctive or conjunctive. In accordance with our focus on lower bounds, we decided to treat them as disjunctive, to give the hierarchical alignment algorithm more opportunities to use fewer gaps. This design decision is one of the main differences between our study and those of Fox (2002), who treated links to the same word conjunctively.

By treating many-to-one links disjunctively, our measure of complexity ignored a large class of discontinuities. Many types of discontinuous constituents exist in text independently of any translation. Simard et al. (2005) give the example in Figure 1.4(a) with the French negation *ne... pas*. The disparate elements of such constituents would usually be attached to the same word in a translation. However, when our hierarchical aligner saw two words linked to one word, it ignored one of the two links, resulting in the word links in Figure 1.4(b). Our lower bounds would be higher if they accounted for this kind of discontinuity.

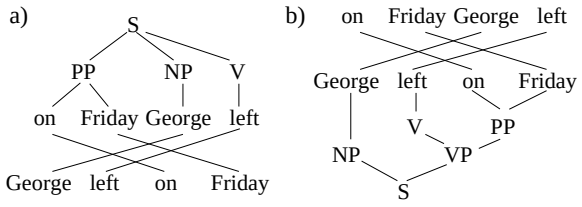

Figure 1.5: (a) With a parse tree constraining the top sentence, a hierarchical alignment is possible without gaps. (b) With a parse tree constraining the bottom sentence, no such alignment exists.

1.2 Experiments

1.2.1 Data

We used two monolingual multitexts and five bilingual multitexts. The Romanian/English and Hindi/English data came from Martin et al. (2005). For Chinese/English and Spanish/English, we used the data from Ayan et al. (2005). Our French/English data were those used by Mihalcea and Pedersen (2003). The monolingual multitext labeled "MTEval" in the tables consists of multiple independent translations from Chinese to English (NIST, 2002). The other monolingual multitext, labeled "fiction," consists of two independent translations from French to English of Jules Verne's novel *20,000 Leagues Under the Sea*, sentence-aligned by Barzilay and McKeown (2001).

From the monolingual multitexts, we removed all sentence pairs where either sentence was longer than 100 words. Table 1.1 gives descriptive statistics for the remaining data. The table also shows the upper bound of the 95% confidence intervals for the coverage rates reported later. The results of experiments on different multitexts are not directly comparable, due to the varying genres and sentence lengths.

1.2.2 Constraining Parse Trees

One of the main independent variables in our experiments was the number of monolingual parse trees used to constrain the hierarchical alignments. To induce models of translational equivalence, some researchers have tried to use such trees to constrain bilingual constituents: The span of

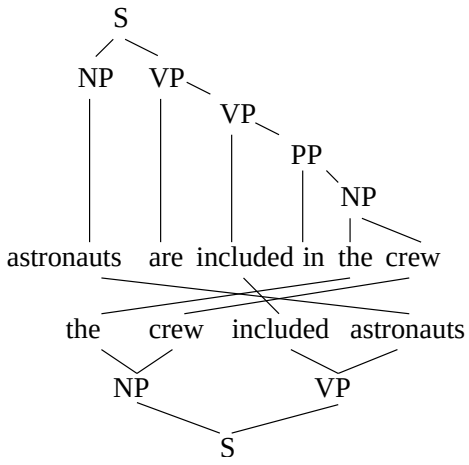

Figure 1.6: A word alignment that cannot be generated without gaps in a manner consistent with both parse trees.

every node in the constraining parse tree must coincide with the relevant monolingual span of some node in the bilingual derivation tree. Not surprisingly, these additional constraints can thwart efforts at hierarchical alignment that might have succeeded otherwise. Figure 1.5(a) shows a word alignment and parse tree that can be hierarchically aligned without gaps. *on* and *Friday* can be composed in both sentences into a constituent without crossing any phrase boundaries in the tree, as can *George* and *left*. These two constituents can then be composed to cover the entire sentence pair. On the other hand, if a constraining tree is applied to the other sentence as shown in Figure 1.5(b), then the word alignment and tree constraint conflict. The restricting factor is that the verb phrase (noted as VP) is composed of the words *left*, *on* and *Friday*. In order for the VP to become a constituent, those words, and what they are aligned to, must be contiguous in both sentences. The words are not contiguous in the top input sentence, so this particular example cannot be hierarchically aligned without gaps.

On the other hand, if a gap is allowed, then the VP can compose as *on Friday . . . left* in the top sentence, where the ellipsis represents a gap. This VP can then compose with the NP to infer a complete synchronous parse tree. Some authors have applied constraining parse trees to both sides of the multitext. The example in Figure 1.6 can be hierarchically aligned using either one of the two constraining trees, but gaps are necessary to align it with both trees.

1.2.3 Methods

We parsed the English side of each bilingual multitext and both sides of each English/English multitext using an off-the-shelf syntactic parser (Bikel, 2004), which was trained on sections 02-21 of the English Penn Treebank (Marcus et al., 1993).

Our bilingual multitexts came with manually annotated word alignments. For the monolingual multitexts, we used an automatic word aligner based on a cognate heuristic and a list of 282 function words compiled by hand. The aligner linked two words to each other only if neither of them was on the function word list and their longest common subsequence ratio (Melamed, 1995) was at least 0.75. Words that were not linked to another word in this manner were linked to nothing ("NULL"). For the purposes of this study, a word aligned to NULL is a non-constraint, because it can always be composed without a gap with some constituent that is adjacent to it on just one side of the multitext. The number of automatically induced non-NULL links was lower than what would be drawn by hand. Fewer word alignments means fewer constraints, so it is likely that the actual complexity of translational equivalence is higher than our lower bounds.

We modified the word alignments in all multitexts to minimize the chances that alignment errors would lead to an over-estimate of alignment complexity. All of the modifications involved adding links to NULL. Due to our disjunctive treatment of conflicting links, the addition of a link to NULL can decrease but cannot increase the complexity of an alignment. For example, if we added the links *(cela, NULL)* and *(NULL, that)* to the alignment in Figure 1.1, the hierarchical alignment algorithm could use them instead of the link between *cela* and *that*. It could thus generate the modified word alignment without using a gap. We added NULL links in two situations. First, if a subset of the links in an alignment formed a many-to-many mapping but did not form a bipartite clique (i.e. every word on one side linked to every word on the other side), then we added links from each of these words to NULL. Second, if n words on one side of the multitext aligned to m words on the other side with $m > n$ then we added NULL links for each of the words on the side with m words.

After modifying the alignments and obtaining monolingual parse trees, we measured the alignment complexity of each multitext using a hierarchical alignment algorithm, as described in

# of gaps allowed →	0/0	0/1 or 1/0
Chinese/English	26 = 5%	0 = 0%
Romanian/English	1 = 0%	0 = 0%
Hindi/English	2 = 2%	0 = 0%
Spanish/English	3 = 2%	0 = 0%
French/English	3 = 1%	0 = 0%

Table 1.2: Alignment failure rates for bilingual multitexts under word alignment constraints only.

# of gaps allowed on non-English side →	0	1	2
Chinese/English	298 = 61%	28 = 6%	0 = 0%
Romanian/English	82 = 41%	6 = 3%	1 = 0%
Hindi/English	33 = 37%	1 = 1%	0 = 0%
Spanish/English	75 = 38%	4 = 2%	0 = 0%
French/English	* 67 = 15%	2 = 0%	0 = 0%

Table 1.3: Alignment failure rates for bilingual multitexts under the constraints of a word alignment and a monolingual parse tree on the English side. The cell marked with a * used data similar to Fox (2002).

Section 1.1. Separate measurements were taken with zero, one, and two constraining parse trees.

Unlike Fox (2002), Galley et al. (2004) and Zhang et al. (2006), we measured failure rates per corpus rather than per sentence pair or per node in a constraining tree. This design was motivated by the observation that if a translation model cannot correctly model a certain word alignment, then it is liable to make incorrect inferences about arbitrary parts of that alignment, not just the particular word links involved in a complex pattern. The failure rates we report represent the fraction of training data that is susceptible to misinterpretation by overconstrained translation models.

1.2.4 Summary Results

Table 1.2 shows the lower bound on alignment failure rates with and without gaps for five languages paired with English. This table represents the case where the only constraints are from word alignments. Wu (2000) called these alignments "highly distorted inside-out alignments, which are extremely rare in correctly translated bitext." He also wrote "we know of no actual

# of gaps →	0/0	0/1	0/2
0 CTs	171 = 3%	0 = 0%	0 = 0%
1 CTs	1792 = 34%	143 = 3%	7 = 0%
2 CTs	3227 = 61%	3227 = 61%	3227 = 61%

Table 1.4: Alignment failure rates in the MTEval multitext, over varying numbers of gaps and constraining trees (CTs).

# of gaps →	0/0	0/1	0/2
0 CTs	23 = 0%	0 = 0%	0 = 0%
1 CTs	655 = 10%	22 = 0%	1 = 0%
2 CTs	1559 = 25%	1559 = 25%	1559 = 25%

Table 1.5: Alignment failure rates in the fiction multitext, over varying numbers of gaps and constraining trees (CTs).

examples in any parallel corpus for languages that do not have free word order" (p. 447). In contrast, we found examples in all multitexts that could not be hierarchically aligned without gaps, including at least 5% of the Chinese/English sentence pairs. Figures 1.1 on page 5 and 1.3 on page 6 are examples taken from our corpora. Allowing constituents with a single gap on one side of the multitext decreased the observed failure rate to zero for all five multitexts.

Table 1.3 shows what happened when we used monolingual parse trees to restrict the compositions on the English side. The failure rates were above 35% for four of the five language pairs, and 61% for Chinese/English! Again, the failure rate fell dramatically when one gap was allowed on the unconstrained (non-English) side of the multitext. Allowing two gaps on the non-English side led to almost complete coverage of these word alignments.

Table 1.3 does not specify the number of gaps allowed on the English side, because varying this parameter never changed the outcome. The only way that a gap on that side could increase coverage is if there was a node in the constraining parse tree that had at least four children whose translations were in one of the complex permutations. The absence of such cases in the data implies that the failure rates would be identical even if we allowed production rules of rank higher than two.

Table 1.4 shows the alignment failure rates for the MTEval multitext. With word alignment constraints only, 3% of the sentence pairs could not be hierarchically aligned without gaps.

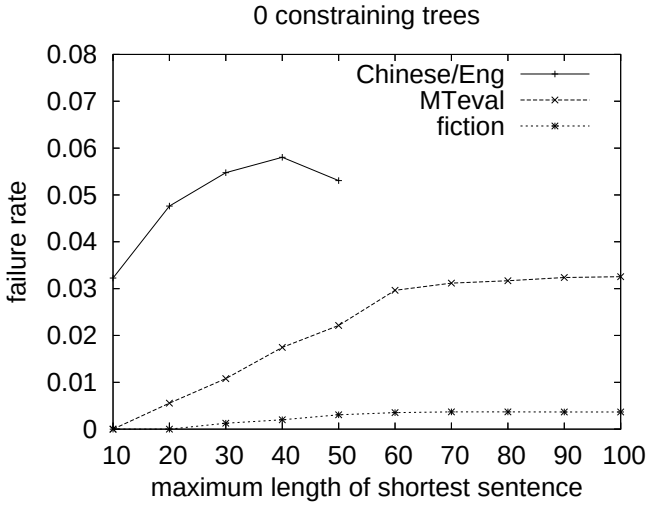

Figure 1.7: Cumulative failure rates for hierarchical alignment without gaps vs. maximum length of shorter sentence with no constraining trees.

Allowing a single gap on one side decreased this failure rate to zero. With a parse tree constraining constituents on one side of the multitext and with no gaps, alignment failure rates rose from 3% to 34%, but allowing a single gap on the side of the multitext that was not constrained by a parse tree brought the failure rate back down to 3%. With two constraining trees the failure rate was 61%, and allowing gaps did not lower it, for the same reasons that allowing gaps on the tree-constrained side made no difference in Table 1.3.

The trends in the fiction multitext (Table 1.5) were similar to those in the MTEval multitext, but the coverage was always higher, for two reasons. First, the median sentence size was lower in the fiction multitext. Second, the MTEval translators were instructed to translate as literally as possible, but the fiction translators paraphrased to make the fiction more interesting. This freedom in word choice reduced the frequency of cognates and thus imposed fewer constraints on the hierarchical alignment, which resulted in looser estimates of the lower bounds. We would expect the opposite effect with hand-aligned data.

To study how sentence length correlates with the complexity of translational equivalence, we took subsets of each multitext while varying the maximum length of the shorter sentence in each pair. The length of the shorter sentence is the upper bound on the number of non-NULL word

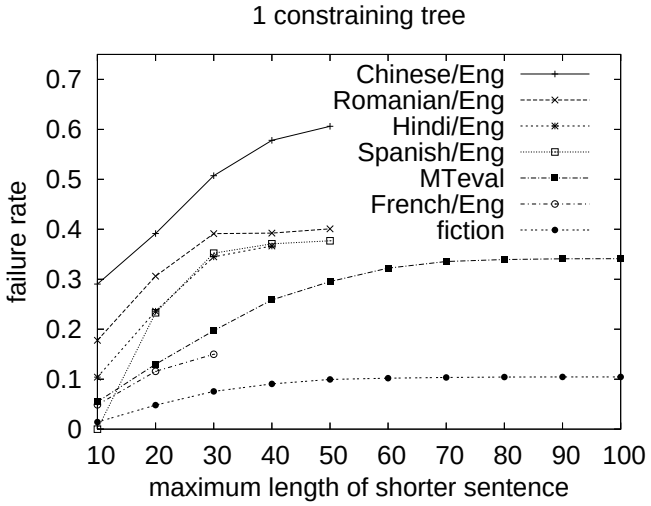

Figure 1.8: Cumulative failure rates for hierarchical alignment without gaps vs. maximum length of shorter sentence with 1 constraining tree.

alignments. Figures 1.7, 1.8 and 1.9 plot the resulting alignment failure rates with and without constraining parse trees. The lines in these graphs are not comparable to each other because of the variety of genres involved.

1.2.5 Detailed Failure Analysis

For certain cross-linguistic constructions, translation from one language to another systematically changes the word order (Dorr, 1994). However, we wanted to control this variable, in order to study the complexity of translational equivalence exhibited by all multitexts, rather than the fraction attributed to divergences between specific language pairs. So our failure analysis focuses on one of the English/English corpora.

We examined by hand 30 random sentence pairs from the MTEval multitext in each of three different categories:

(i) the set of sentence pairs that could not be hierarchically aligned without gaps, even without constraining parse trees;

(ii) the set of sentence pairs that could not be hierarchically aligned without gaps with one

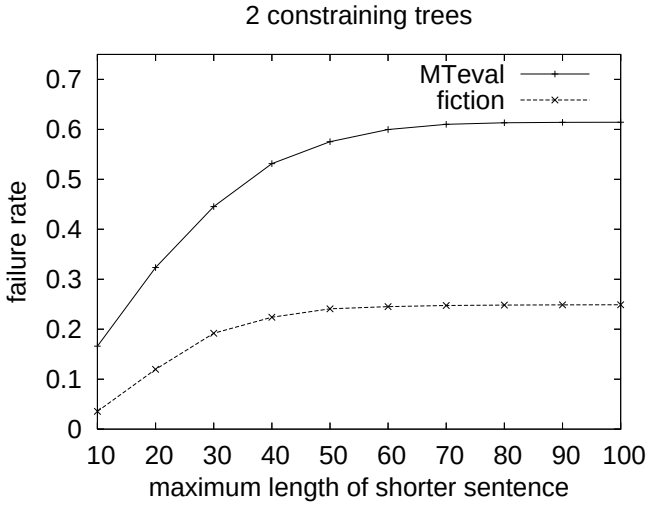

Figure 1.9: Cumulative failure rates for hierarchical alignment without gaps vs. maximum length of shorter sentence with 2 constraining trees.

category →	1	2	3
valid reordering	12	10	5
parser error	n/a	16	25
same word used differently	15	4	0
erroneous cognates	3	0	0
initial failure rate (%)	3.25	31.9	38.4
% false negatives	60±7	66±7	84±3
adjusted failure rate (%)	1.3±.22	11±2.2	6±1.1

Table 1.6: Detailed failure analysis of MTEval multitext. Each column is for a sample size of 30.

constraining parse tree, but that did not fall into category 1;

(iii) the set of sentence pairs that could not be hierarchically aligned without gaps with two constraining parse trees, but that did not fall into category 1 or 2.

Table 1.6 shows the results of this analysis.

In category 1, 60% of the word alignments that could not be hierarchically aligned without gaps were caused by word alignment errors, e.g.:

1a GlaxoSmithKline's second-best selling **drug** may have to face competition.

1b **Drug** maker GlaxoSmithKline may have to face competition on its second best selling product.

The word *drug* appears in both sentences, but for different purposes, so *drug* and *drug* should not have been linked. This sort of error is likely to happen with other word alignment algorithms too because words and their common translations are likely to be linked even if they're not translations in the given sentence. Two errors were caused by words like *targeted* and *started*, which our word alignment algorithm deemed cognates. 12 of the hierarchical alignment failures in this category were true failures, e.g.:

2a **Cheney** denied **yesterday** that the **mission** of his trip was to organize an assault on Iraq, while in **Manama**.

2b **Yesterday** in **Manama**, **Cheney** denied that the **mission** of his trip was to organize an assault on Iraq.

The alignment pattern of the words in bold is the familiar (3,1,4,2) permutation, as in Figure 1.1 on page 5 and 1.3 on page 6. Most of the 12 true failures were due to movement of prepositional phrases. The freedom of movement for such modifiers would be greater in multitexts that involve languages with less rigid word order than English.

The English/English lower bounds are very loose, because the automatic word aligner would not link words that were not cognates. We did not expect to see any failures in hierarchical alignment under such conditions, so the data surprised us. Alignment failure rates on a hand aligned multitext would also be higher. We conclude that the ITG formalism cannot account for the "natural" complexity of translational equivalence, even when translation divergences are factored out.

Of the 30 sentence pairs in category 2, 16 could not be hierarchically aligned due to parser errors and 4 due to faulty word alignments. 10 were due to valid word reordering. In the following example, a co-referring pronoun causes the word alignment to fail with a constraining tree on the second sentence:

3a But **Chretien** appears to have changed his stance after meeting with Bush in Washington last Thursday.

3b But after **Chretien** talked to Bush last Thursday in Washington, **he** seemed to change his original stance.

25 of the 30 sentence pairs in category 3 failed to align due to parser error. 5 examples failed because of valid word reordering. 1 of the 5 reorderings was due to a difference between active voice and passive voice, as in Figure 1.6.

The last row of Table 1.6 takes the various reasons for alignment failure into account. It estimates what the failure rates would be if the monolingual parses and word alignments were perfect, with 95% confidence intervals. These revised rates emphasize the importance of reliable word alignments for this kind of study.

1.3 Discussion

To our knowledge, Example 2 in Section 1.2.5, Figure 1.1 on page 5 and Figure 1.3 on page 6 are the first published examples of sentence pairs from real data that cannot be hierarchically aligned correctly without gaps, even without constraining parse trees. The received wisdom in the literature led us to expect no such examples in bilingual multitexts, let alone in monolingual multitexts. See `http://nlp.cs.nyu.edu/GenPar/ACL06` for more examples.

Perhaps our most surprising results were those involving one constraining parse tree. For Chinese/English, the failure rate was 61%. For one English/English multitext, the failure rate was 34%, even though the translations were very literal and systematic variation between languages could not be the cause. For statistical machine translation (SMT) systems that use monoparses but don't allow discontinuous constituents, these failures represent a major source of noise during training.

These results explain why constraints from independently generated monolingual parse trees have not improved some statistical translation models. For example, Koehn et al. (2003) reported that "requiring constituents to be syntactically motivated does not lead to better constituent pairs, but only fewer constituent pairs, with loss of a good amount of valuable knowledge." This statement is consistent with our findings. However, most of the knowledge loss could be prevented by allowing a gap. With a parse tree constraining constituents on the English side, the coverage failure rate was 61% for the Chinese/English multitext (top row of Table 1.3), but allowing a

gap decreased it to 6%. Zhang and Gildea (2004) found that Yamada and Knight (2001)'s model performed worse than their own alignment, which did not use external syntactic constraints. Yamada and Knight (2001)'s model could explain only the data that would pass the no-gap test in our experiments with one constraining tree (first column of Table 1.3). The conclusions might have been different if Yamada and Knight's model were allowed to use discontinuous constituents. The second row of Table 1.4 suggests that when constraining parse trees are used without gaps, at least 34% of training sentence pairs are likely to introduce noise into the model, even if systematic syntactic differences between languages are factored out. We should not be surprised when such constraints do more harm than good.

To increase the chances that a translation model can explain complex word alignments, some authors have proposed various ways of extending a model's domain of locality. For example, Callison-Burch et al. (2005) have advocated for longer phrases in finite-state phrase-based translation models. We computed the phrase length that would be necessary to cover the words involved in each (3,1,4,2) permutation in the MTEval multitext. Figure 1.10 shows the cumulative percentage of these cases that would be covered by phrases up to a certain length. Only 9 of the 171 cases (5.2%) could be covered by phrases of length 10 or less. Analogous techniques for tree-structured translation models involve either allowing each nonterminal to generate both terminals and other nonterminals (Groves et al., 2004; Chiang, 2005), or, given a constraining parse tree, to "flatten" it (Fox, 2002; Zens and Ney, 2003; Galley et al., 2004). Both of these approaches can increase coverage during training, but, as explained in Section 1.1, they risk losing generalization ability.

Our study suggests that there might be some benefits to an alternative approach using discontinuous constituents, as detailed in the next Chapter. The large reductions in failure rates evident in the second column of Table 1.3 are largely independent of the tightness of our lower bounds. Synchronous parsing with discontinuities is computationally expensive in the worst case, but recently invented data structures make it feasible for typical inputs, as long as the number of gaps allowed per constituent is fixed at a small maximum (Waxmonsky and Melamed, 2006).

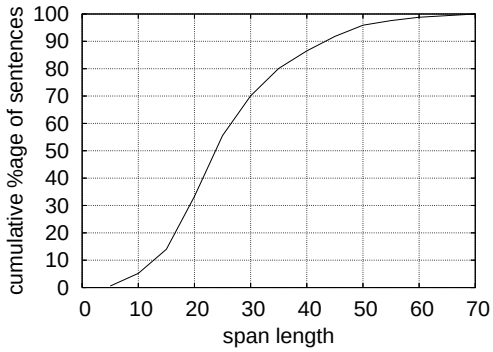

Figure 1.10: Lengths of spans covering words in (3,1,4,2) permutations.

1.4 Conclusions

Our study found surprisingly many examples of translational equivalence that could not be analyzed using binary-branching structures without discontinuities. Allowing a single gap in bilingual phrases or other types of constituents can dramatically reduce the number of such examples. The low coverage rates without gaps might be the main reason why "syntactic" constraints have not increased the accuracy of some well-known SMT systems. The chapter presented evidence of phenomena that can lead to complex patterns of translational equivalence in multitexts of any language pair, and represent the first published examples from real world multitexts of sentences that are not alignable using Inversion Transduction Grammar. We conclude that, regardless of the languages involved, many correct word alignments cannot be generated by any grammar of fan-out 2, including any inversion transduction grammar. In the next chapter, we introduce a new grammar formalism, Generalized Multitext Grammar, which can have an arbitrary number of gaps on each side. Therefore it can explain all the corpora studied in this chapter.

2

Theory: Generalized Multitext Grammars and Generalized Chomsky Normal Form

2.1 Introduction

Transduction grammars have been proposed for the formal description of tuples of texts that are translations of each other, often called parallel texts. As shown theoretically by Melamed (2003) and empirically in Chapter 1, a plausible model of parallel text must be able to express discontinuous constituents. Since linguistic expressions can vanish in translation, a good model must be able to express independent (in addition to synchronous) rewriting. Inversion Transduction Grammar (ITG) (Wu, 1997), Synchronous Context-Free Grammar (SCFG) (Chiang, 2002) and Syntax-Directed Translation Schema (SDTS) (Aho and Ullman, 1969) lack both of these properties. Synchronous Tree Adjoining Grammar (STAG) (Shieber, 1994) lacks the latter and allows only limited discontinuities in each tree.

This chapter, an extension of Melamed et al. (2004), introduces the formal theory behind Generalized Multitext Grammar (GMTG). GMTG satisfies both of the above criteria and offers a way to synchronize Mildly Context-Sensitive Grammar (MCSG). The move to MCSG is motivated by our desire to more perspicuously account for certain syntactic phenomena that cannot be easily captured by context-free grammars, such as clitic climbing, extraposition, and other types of long-distance movement (Becker et al., 1991). On the other hand, MCSGs still observe some restrictions that make the set of strings they generate less expensive to analyze than the strings generated by (properly) context-sensitive grammars.

Ordinary Multitext Grammar (MTG) is a formalism for synchronizing context-free grammars (Melamed, 2003). In MTG, synchronous rewriting is implemented by means of an indexing relation that is maintained over occurrences of nonterminals in a sentential form, using essentially the same machinery as SDTS or SCFG. Unlike SDTS, MTG can extend the dimensionality of the translation relation beyond two, and it can implement independent rewriting by means of partial deletion of syntactic structures.

Our proposal generalizes MTG by moving from component grammars that generate context-free languages to component grammars whose generative power is equivalent to Linear Context-Free Rewriting Systems (LCFRS), a formalism for describing a class of MCSGs. The general-

ization is achieved by allowing context-free productions to rewrite tuples of strings, rather than single strings. Thus, we retain the intuitive top-down definition of synchronous derivation original in SDTS and MTG but not found in LCFRS, while extending the generative power to linear context-free rewriting languages.

This chapter begins with an informal description of GMTG. It continues with a comparison of this formalism's generative capacity to that of a certain well-known non-synchronous formalism. We then propose a synchronous generalization of Chomsky Normal Form, which lays the groundwork for synchronous parsing under GMTG using a CKY-style algorithm (Younger, 1967; Melamed and Wang, 2005). It is this new normal form that allows us to design the efficient SMT system in Chapter 3 and use it in Chapter 4.

2.2 Informal Description and Comparisons

GMTG is a generalization of MTG, which is itself a generalization of CFG to the synchronous case. Here we present MTG in a new notation that shows the relation to CFG more clearly. For example, the following MTG productions can generate the multitext [(*I fed the cat*), (*ya kota kormil*)]:[1]

$$[(\mathrm{S}),(\mathrm{S})] \Rightarrow [(\mathrm{PN}^1\mathrm{VP}^2),(\mathrm{PN}^1\mathrm{VP}^2)] \quad (2.1)$$
$$[(\mathrm{PN}),(\mathrm{PN})] \Rightarrow [(\mathrm{I}),(\mathrm{ya})] \quad (2.2)$$
$$[(\mathrm{VP}),(\mathrm{VP})] \Rightarrow [(\mathrm{V}^1\mathrm{NP}^2),(\mathrm{NP}^2\mathrm{V}^1)] \quad (2.3)$$
$$[(\mathrm{V}),(\mathrm{V})] \Rightarrow [(\mathrm{fed}),(\mathrm{kormil})] \quad (2.4)$$
$$[(\mathrm{NP}),(\mathrm{NP})] \Rightarrow [(\mathrm{D}^1\mathrm{N}^2),(\mathrm{N}^2)] \quad (2.5)$$
$$[(\mathrm{D}),()] \Rightarrow [(\mathrm{the}),()] \quad (2.6)$$
$$[(\mathrm{N}),(\mathrm{N})] \Rightarrow [(\mathrm{cat}),(\mathrm{kota})] \quad (2.7)$$

Each production in this example has two components, the first modeling English and the second (transliterated) Russian. Nonterminals with the same index must be rewritten together (synchronous rewriting). One strength of MTG, and thus also GMTG, is shown in Productions (2.5)

[1] We write production components both side by side and one above another, but each component is always in parentheses.

and (2.6). There is a determiner in English, but not in Russian, so Production (2.5) does not have the nonterminal D in the Russian component and (2.6) applies only to the English component (independent rewriting). Formalisms that do not allow independent rewriting require a corresponding D to appear in the second component on the right-hand side (RHS) of Production (2.5), and this D would eventually generate the empty string. This approach has the disadvantage that it introduces spurious ambiguity about the position of the "empty" nonterminal with respect to the other nonterminals in its component. Spurious ambiguity leads to wasted effort during parsing.

GMTG's implementation of independent rewriting through the empty tuple () serves a very different function from the empty string. Consider the following GMTG:

$$[(S),(S)] \Rightarrow [(a),(\varepsilon)] \tag{2.8}$$
$$[(S),(S)] \Rightarrow [(X^1),(Y^2)] \tag{2.9}$$
$$[(X),()] \Rightarrow [(b),()] \mid [(c),()] \mid [(d),()] \tag{2.10}$$
$$[(),(Y)] \Rightarrow [(),(e)] \mid [(),(f)] \mid [(),(g)] \tag{2.11}$$

Production (2.8) asserts that symbol a vanishes in translation. Its application removes both of the nonterminals on the left-hand side (LHS), pre-empting any other production. In contrast, Production (2.9) explicitly relaxes the synchronization constraint, so that the two components can be rewritten independently. The other six productions make assertions about only one component and are agnostic about the other component. Incidentally, generating the same language with only fully synchronized productions would raise the number of required productions to 11, so independent rewriting also helps to reduce grammar size.

Independent rewriting is also useful for modeling paraphrasing. Take, for example, [(*Tim got a pink slip*), (*Tim got laid off*)]. While the two sentences have the same meaning, the objects of their verb phrases are structured very differently. GMTG can express their relationships as follows:

$$[(\mathrm{S}),(\mathrm{S})] \Rightarrow [(\mathrm{NP}^1\mathrm{VP}^2),(\mathrm{NP}^1\mathrm{VP}^2)] \quad (2.12)$$
$$[(\mathrm{VP}),(\mathrm{VP})] \Rightarrow [(\mathrm{V}^1\mathrm{NP}^2),(\mathrm{V}^1\mathrm{PP}^2)] \quad (2.13)$$
$$[(\mathrm{NP}),(\mathrm{PP})] \Rightarrow [(\mathrm{DT}^1\mathrm{A}^2\mathrm{N}^3),(\mathrm{VB}^4\mathrm{R}^5)] \quad (2.14)$$
$$[(\mathrm{NP}),(\mathrm{NP})] \Rightarrow [(\mathrm{Tim}),(\ \mathrm{Tim})] \quad (2.15)$$
$$[(\mathrm{V}),(\mathrm{V})] \Rightarrow [(\mathrm{got}),(\mathrm{got})] \quad (2.16)$$
$$[(\mathrm{DT}),()] \Rightarrow [(\mathrm{a}),()] \quad (2.17)$$
$$[(\mathrm{A}),()] \Rightarrow [(\mathrm{pink}),()] \quad (2.18)$$
$$[(\mathrm{N}),()] \Rightarrow [(\mathrm{slip}),()] \quad (2.19)$$
$$[(),(\mathrm{VB})] \Rightarrow [(),(\mathrm{laid})] \quad (2.20)$$
$$[(),(\mathrm{R})] \Rightarrow [(),(\mathrm{off})] \quad (2.21)$$

As described by Melamed (2003), MTG requires production components to be contiguous, except after binarization. GMTG removes this restriction. Take, for example, the sentence pair [(*The doctor treats his teeth*), (*El médico le examina los dientes*)] (Dras and Bleam, 2000). The Spanish clitic *le* and the NP *los dientes* should both be paired with the English NP *his teeth*, giving rise to a discontinuous constituent in the Spanish component. A GMTG fragment for this sentence pair is shown below:

$$[(\mathrm{S}),(\mathrm{S})] \Rightarrow [(\mathrm{NP}^1\mathrm{VP}^2),(\mathrm{NP}^1\mathrm{VP}^2)]$$
$$[(\mathrm{VP}),(\mathrm{VP})] \Rightarrow [(\mathrm{V}^1\mathrm{NP}^2),(\mathrm{NP}^2\mathrm{V}^1\mathrm{NP}^2)]$$
$$[(\mathrm{NP}),(\mathrm{NP})] \Rightarrow [(\mathrm{The\ doctor}),(\mathrm{El\ médico})]$$
$$[(\mathrm{V}),(\mathrm{V})] \Rightarrow [(\mathrm{treats}),(\mathrm{examina})]$$
$$[(\mathrm{NP}),(\mathrm{NP},\mathrm{NP})] \Rightarrow [(\mathrm{his\ teeth}),(\mathrm{le},\mathrm{los\ dientes})]$$

Note the discontinuity between *le* and *los dientes*. Such discontinuities are marked by commas on both the LHS and the RHS of the relevant component.

GMTG's flexibility allows it to deal with many complex syntactic phenomena. For example, Becker et al. (1991) point out that TAG does not have the generative capacity to model certain kinds of scrambling in German, when the so-called "co-occurrence constraint" is imposed, requiring the derivational pairing between verbs and their complements. They examine the

English/German sentence fragment [(*... that the detective has promised the client to indict the suspect of the crime*), (*... daß des Verbrechens der Detektiv den Verdächtigen dem Klienten zu überführen versprochen hat*)]. The verbs *versprochen* and *überführen* both have two noun phrases as arguments. In German, these noun phrases can appear to the left of the verbs in any order. The following is a GMTG fragment for the above sentence pair:[2]

$$\begin{bmatrix}(\mathrm{S})\\(\mathrm{S})\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathrm{N}^1_{det} \text{ has promised } \mathrm{N}^2_{client}\hat{\mathrm{S}}^3)\\(\hat{\mathrm{S}}^3\mathrm{N}^1_{Det}\hat{\mathrm{S}}^3\mathrm{N}^2_{Klien}\hat{\mathrm{S}}^3 \text{versprochen hat})\end{bmatrix} \tag{2.22}$$

$$\begin{bmatrix}(\hat{\mathrm{S}})\\(\hat{\mathrm{S}},\hat{\mathrm{S}},\hat{\mathrm{S}})\end{bmatrix} \Rightarrow \begin{bmatrix}(\text{to indict } \mathrm{N}^1_{suspect}\mathrm{N}^2_{crime})\\(\mathrm{N}^2_{Verb},\mathrm{N}^1_{Verd},\ \text{zu überführen})\end{bmatrix} \tag{2.23}$$

The discontinuities allow the noun arguments of *versprochen* to be placed in any order with the noun arguments of *überführen*. Rambow (1995) gives a similar analysis.

2.3 Formal Definitions

Let V_N be a finite set of nonterminal symbols and let $\mathbb{Z}$ be the set of integers.[3] We define $\mathcal{I}(V_N) = \{A^{(t)} \mid A \in V_N,\ t \in \mathbb{Z}\}$.[4] Elements of $\mathcal{I}(V_N)$ will be called **indexed nonterminal symbols**. In what follows we also consider a finite set of terminal symbols V_T, disjoint from V_N, and work with strings in V_I^*, where $V_I = \mathcal{I}(V_N) \cup V_T$. For $\gamma \in V_I^*$, we define $\mathsf{index}(\gamma) = \{t \mid \gamma = \gamma' A^{(t)} \gamma'',\ \gamma', \gamma'' \in V_I^*,\ A^{(t)} \in \mathcal{I}(V_N)\}$, i.e., $\mathsf{index}(\gamma)$ is the set of all indexes that appear in symbols in γ.

An **indexed tuple vector**, or ITV, is a vector of tuples of strings over V_I, having the form

$$\overline{\gamma} \quad = \quad [(\gamma_{11}, \ldots, \gamma_{1q_1}), \ldots, (\gamma_{D1}, \ldots, \gamma_{Dq_D})]$$

where $D \geq 1$, $q_i \geq 0$ and $\gamma_{ij} \in V_I^*$ for $1 \leq i \leq D$, $1 \leq j \leq q_i$. We write $\overline{\gamma}[i]$, $1 \leq i \leq D$, to denote the i-th component of $\overline{\gamma}$ and $\varphi(\overline{\gamma}[i])$ to denote the arity of such a tuple, which is q_i. When

[2] These are only a small subset of the necessary productions. For brevity, the subscripts of the nonterminals indicate their lexical heads, but the terminal productions have been left out.

[3] Any other infinite set of indexes would suit too.

[4] The parentheses around indexes distinguish them from other uses of superscripts in formal language theory. However, we shall omit the parentheses when the context is unambiguous.

$\varphi(\overline{\gamma}[i]) = 0$, $\overline{\gamma}[i]$ is the empty tuple, written (). This should not be confused with (ε), that is the tuple of arity one containing the empty string. A **link** is an ITV having the form

$$\overline{\Gamma} \quad = \quad [(A_{11}^{(t)}, \ldots, A_{1q_1}^{(t)}), \ldots, (A_{D1}^{(t)}, \ldots, A_{Dq_D}^{(t)})],$$

that is, each Γ_{ij} consists of one indexed nonterminal and all of these nonterminals are coindexed. As we shall see, the notion of a link generalizes the notion of nonterminal in context-free grammars: each production rewrites a single link.

Definition 2.1. Let $D \geq 1$ be some integer constant. A **generalized multitext grammar** with D dimensions (D-GMTG for short) is a tuple $G = (V_N, V_T, P, S)$ where V_N, V_T are finite, disjoint sets of nonterminal and terminal symbols, respectively, $S \in V_N$ is the start symbol and P is a finite set of productions.

Each production has the form

$$[(A_{11}^{(t)}, \ldots, A_{1q_1}^{(t)}), \ldots, (A_{D1}^{(t)}, \ldots, A_{Dq_D}^{(t)})] \quad \rightarrow \quad [(\gamma_{11}, \ldots, \gamma_{1q_1}), \ldots, (\gamma_{D1}, \ldots, \gamma_{Dq_D})]$$

where the LHS is a D-dimensional link and the RHS is a D-dimensional ITV. If S appears in $(A_{1i}^{(t)}, \ldots, A_{1q_i}^{(t)})$ for some i, $1 \leq i \leq D$, then $q_i = 1$.

We omit symbol D from D-GMTG whenever it is understood from the context or it is not relevant.

To simplify notation, we omit the unique index appearing on the LHS of productions. We sometimes write productions as

$$\overline{p} \; = \; \begin{bmatrix} (A_{11}, \ldots, A_{1q_1}) \\ \vdots \\ (A_{D1}, \ldots, A_{Dq_D}) \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix}.$$

With some abuse of notation, we also write productions as

$$\overline{p} \quad = \quad [p_1, \ldots, p_D]$$

with $p_i = (A_{i1}, \ldots, A_{iq_i}) \rightarrow (\alpha_{i1}, \ldots, \alpha_{iq_i})$, $1 \leq i \leq D$. Each p_i is called a **production component**. The production component $() \rightarrow ()$ is called the **inactive** production component. All other production components are called **active** and we set $\mathsf{active}(\overline{p}) = \{i \mid q_i > 0\}$. Inactive production components are used to relax synchronous rewriting on some dimensions, that is to implement rewriting on $d < D$ components. When $d = 1$, rewriting is licensed on one component, independently of all the others.

In GMTG, the derives relation is defined over ITVs. GMTG derivation proceeds by synchronous application of all the active components in some production. The indexed nonterminals to be rewritten simultaneously must all have the same index t, and all nonterminals indexed with t in the ITV must be rewritten simultaneously. Some additional notation will help us to define rewriting precisely. A **reindexing** is a bijective function on $\mathbb{Z}$. We extend reindexings to V_I by letting $f(A^{(t)}) = A^{(f(t))}$ for $A^{(t)} \in \mathcal{I}(V_N)$ and $f(a) = a$ for $a \in V_T$. We also extend f to strings in V_I^* as usual, by letting $f(\varepsilon) = \varepsilon$ and $f(X\alpha) = f(X)f(\alpha)$, for each $X \in V_I$ and $\alpha \in V_I^*$. We say that strings $\alpha, \alpha' \in V_I^*$ are **independent** if $\mathsf{index}(\alpha) \cap \mathsf{index}(\alpha') = \varnothing$.

Definition 2.2. Let $G = (V_N, V_T, P, S)$ be a D-GMTG and let $\overline{p} = [p_1, \ldots, p_D] \in P$ with $p_i = (A_{i1}, \ldots, A_{iq_i}) \rightarrow (\alpha_{i1}, \ldots, \alpha_{iq_i})$, $1 \leq i \leq D$. Let also $\overline{\gamma}$ and $\overline{\delta}$ be two ITVs with $\overline{\gamma}[i] = (\gamma_{i1}, \ldots, \gamma_{iq_i})$ and $\overline{\delta}[i] = (\delta_{i1}, \ldots, \delta_{iq_i})$, $1 \leq i \leq D$. Assume that α is some concatenation of all α_{ij} and that γ is some concatenation of all γ_{ij}, $1 \leq i \leq D$, $1 \leq j \leq q_i$, and let f be some reindexing such that strings $f(\alpha)$ and γ are independent. The **derives relation** $\overline{\gamma} \Rightarrow_G^{\overline{p}} \overline{\delta}$ holds whenever there exists an index $t \in \mathbb{Z}$ such that the following two conditions are satisfied:

(i) for each $i \in \mathsf{active}(\overline{p})$ we have

$$\gamma_{i1} \cdots \gamma_{iq_i} \quad = \quad \gamma'_{i0} A_{i1}^{(t)} \gamma'_{i1} A_{i2}^{(t)} \cdots \gamma'_{iq_i-1} A_{iq_i}^{(t)} \gamma'_{iq_i}$$

with $\gamma'_{ij} \in V_I^*$, $0 \leq j \leq q_i$, and $t \notin \mathsf{index}(\gamma'_{i0}\gamma'_{i1} \cdots \gamma'_{iq_i})$, and each δ_{ij} is obtained from γ_{ij} by replacing each $A_{ij'}^{(t)}$ with $f(\alpha_{ij'})$;

(ii) for each i with $1 \leq i \leq D$ and $i \notin \mathsf{active}(\overline{p})$, we have $t \notin \mathsf{index}(\gamma_{i1} \cdots \gamma_{iq_i})$ and $\overline{\gamma}[i] = \overline{\delta}[i]$.

Condition (i) in Definition 2.2 imposes that, for each production component p_i of $\overline{p}$ that

is not the inactive production component, we must have that q_i exactly equals the number of occurrences of index t within $(\gamma_{i1}, \ldots, \gamma_{iq_i})$. Furthermore, condition (ii) imposes that there are no occurrences of index t within $(\gamma_{i1}, \ldots, \gamma_{iq_i})$, in case p_i is the inactive production component. If one of these conditions is violated for some i, then production $\overline{p}$ cannot be used to rewrite $\overline{\gamma}$.

We use relation $\Rightarrow_G$ defined as the union of all $\Rightarrow_G^{\overline{p}}$ for $\overline{p} \in P$. We also use the reflexive and transitive closure of $\Rightarrow_G$, written $\Rightarrow_G^*$, to represent derivations.

We can now introduce the notion of generated language (or generated relation). A **start link** is a link where at least one component is the tuple $(S^{(t)})$, S the start symbol, and the rest of the components are (). There may be no more than $2^D - 1$ types of start links in the LHS of a given D-GMTG. A **terminal ITV** is an ITV where each component has either the form (w) with $w \in V_T^*$, or the form (). The **language** generated by a D-GMTG G is defined as

$$L(G) \quad = \quad \{\overline{\gamma} \mid \overline{\$} \Rightarrow_G^* \overline{\gamma},\ \overline{\$} \text{ a start link},\ \overline{\gamma} \text{ a terminal ITV}\}. \tag{2.24}$$

Each terminal ITV in $L(G)$ is called a **multitext**. For every D-GMTG G, $L(G)$ can be partitioned into $2^D - 1$ subsets, each containing multitexts derived from a different start link. These subsets are disjoint, since every non-empty tuple of a start link is eventually rewritten as a tuple containing a single string, of length greater than or equal to zero.[5]

A **start production** is a production whose LHS is a start link. A GMTG writer can choose the combinations of components in which the grammar can generate, by including start productions with the desired combinations of active components. Some example start productions are:

$$[(S), (S)] \quad \Rightarrow \quad [(X^1), (Y^1)] \tag{2.25}$$
$$[(S), (S)] \quad \Rightarrow \quad [(X^1), (Y^2)] \tag{2.26}$$
$$[(S), ()] \quad \Rightarrow \quad [(X^1), ()] \tag{2.27}$$
$$[(), (S)] \quad \Rightarrow \quad [(), (Y^1)] \tag{2.28}$$

[5] We are assuming that there are no useless nonterminals.

If a grammar contains no start productions with a certain combination of active components, then the corresponding subset of $L(G)$ will be empty.

A GMTG G that can generate multitexts with inactive components can model relations of different dimensionalities. This capability enables a synchronous grammar to govern lower-dimensional sublanguages/translations. For example, an English/Italian GMTG can include Production (2.9), an English CFG, and an Italian CFG. A single GMTG can then govern both translingual and monolingual information in applications. Furthermore, this capability simplifies the normalization procedure described in Section 2.4. Otherwise, this procedure would require exceptions to be made when eliminating ϵ's from start productions.

We close this section with the definition of two parameters associated with GMTGs, that play an important role in this dissertation.

Definition 2.3. Let $G = (V_N, V_T, P, S)$ be a D-GMTG and let $\overline{p} = [p_1, \ldots, p_D]$ be a production in P with $p_i = (A_{i1}, \ldots, A_{iq_i}) \rightarrow (\alpha_{i1}, \ldots, \alpha_{iq_i})$, $1 \leq i \leq D$. The **rank** of $\overline{p}$ and G are, respectively, $\rho(\overline{p}) = |\mathsf{index}(\alpha_{11} \cdots \alpha_{1q_1} \alpha_{21} \cdots \alpha_{Dq_D})|$ and $\rho(G) = \max_{\overline{p} \in P} \rho(\overline{p})$. The **fan-out** of p_i, $\overline{p}$ and G are, respectively, $\varphi(p_i) = q_i$, $\varphi(\overline{p}) = \sum_{i=1}^{D} \varphi(p_i)$ and $\varphi(G) = \max_{\overline{p} \in P} \varphi(\overline{p})$.

For example, the rank of production (2.23) is two and its fan-out is four. This also gives us a more formal understanding of the relationship between fan-out and gaps alluded to in Chapter 1.

Throughout the rest of this dissertation, we denote the class of grammars in GMTG with a fan-out of f as GMTG(f) .

2.4 Motivation for Generalized Chomsky Normal Form

Certain kinds of text analysis require a grammar to be cast in a convenient normal form. The prototypical example for CFG is Chomsky Normal Form (CNF), which is required for CKY-style parsing (Aho and Ullman, 1972).

A CFG is in **Chomsky Normal Form** (CNF) if it has no useless nonterminals or useless terminals and if every production is in one of two forms:

(i) A **nonterminal production** has a rank of two and no terminals or ε's on the RHS.

(ii) A **terminal production** is of the form $A \Rightarrow a$, where $A \in V_N$ and $a \in V_T$.

We introduce here a generalization of CNF for GMTG. A D-GMTG is in **Generalized Chomsky Normal Form** (GCNF) if it has no useless links or useless terminals and if every production is in one of two forms:

(i) A **nonterminal production** has a rank of two and no terminals or ε's on the RHS.

(ii) A **terminal production** has exactly one component of the form $A \Rightarrow a$, where $A \in V_N$ and $a \in V_T$. The other components are inactive.

Before proceeding, it is important to understand how GCNF allows us to build an SMT system which is both efficient and conceptually simple. In order to fully understand that, we must first review what a logic is and how it can be described.

Logics

Melamed and Wang (2005) describe how a parser's **logic** determines the parser's possible states and transitions. Understanding the description and inner workings of the Logic will help explian why GCNF is useful for SMT by parsing, so we describe their view of Logics here. The specification of a parsing logic has three parts:

- a set of term type signatures,
- a set of inference rule type signatures,
- a set of axiom terms.

Terms are the building blocks of inference rules. **Items** are terms that represent partial parses. Terms that represent grammatical constraints such as production rules are sometimes called **grammar terms**. When the parser runs, the term and inference rule types are instantiated and their variables are assigned values. The state of a parser can be uniquely specified by the values of all possible terms.

In the parser's initial state, all terms have a particular default value, such as "zero probability." **Axioms** are term instances (not types) that are assigned non-default values during the parser's

Term Types	
terminal items	$\langle i, t \rangle$
nonterminal items	$[X; (i, j)]$
terminating productions	$X \Rightarrow t$
nonterminating productions	$X \Rightarrow Y\ Z$
Axioms	
input words	$\langle i, w_i \rangle$ for $1 \leq i \leq n$
grammar terms	as given by the grammar
Inference Rule Types	
Scan	$\frac{\langle i,t \rangle\ ,\ X \Rightarrow t}{[X;(i-1,i)]}$
Compose	$\frac{[Y;(i,j)]\ ,\ [Z;(j,k)]\ ,\ X \Rightarrow Y\ Z}{[X;(i,k)]}$

Table 2.1: Logic D1C. w_i are input words; X, Y and Z are nonterminal labels; t is a terminal; i and j are word boundary positions; n is the length of the input.

initialization procedure. The most common kinds of axioms come from the grammar and from the input. Typically, each input word gives rise to an axiom. If the grammar involves production rules, then each production rule becomes an axiom too. As a parser runs, it can change the values of terms from their initial value to some other value.

Inference rules describe the parser's possible transitions from one state to another. We shall express inference rules as sequents: $\frac{x_1,\ldots,x_k}{y}$ means that the value of the **consequent** y depends on the values of the **antecedents** $x_1, \ldots, x_k$. For example, if we are dealing with probabilities, then the probability of the consequent might be defined as the product of the probabilities of the antecedents. Every change in term value corresponds to the invocation of an inference rule where that term is a consequent.

Melamed and Wang (2005) describe Logic D1C, shown in Table 2.1. This is a logic for parsing under context-free grammars (CFGs) in Chomsky Normal Form. This logic has four term types. Two term types represent production rules in the grammar. The other two term types are items.

Each of the logic's terminal items relates a terminal symbol to a word position. Each of the logic's nonterminal items relates a nonterminal symbol to a span. Each span consists of boundaries i and j, which range over positions between and around the words in the input. The position to the left of the first word is zero, and the position to the right of the jth word is j. Thus, $0 \leq i < j \leq n$, where n is the length of the input.

Parser D1C is any inference procedure based on Logic D1C. For every run, Parser D1C is initialized with axioms that represent its grammar's production rules and the words in its input. It then commences to fire inferences. A *Scan* inference can fire for the ith word w_i if that word appears on the right-hand side (RHS) of a terminating production in the grammar. If a word appears on the RHS of multiple productions (with different left-hand sides), then multiple *Scan* inferences can fire for that word. The span of each item inferred by a *Scan* inference always has the form $(i-1, i)$ because such items always span one word, so the distance between the item's boundaries is always one.

Parser D1C spends most of its time composing pairs of items into larger items. It can *Compose* two items whenever they satisfy both of the following constraints:

- *Immediate Dominance (ID).* Their labels must match the nonterminals on the RHS of a nonterminating production rule in the grammar.

- *Linear Precedence (LP).* The order of the items' spans over the input must match the order of their labels in the antecedent production rule.

If two spans overlap, then their order is undefined, and they cannot *Compose*. Thus, the LP constraint ensures that no part of the input is covered more than once, so that every partial parse is a tree (rather than a more general graph). The reason items store their spans is to help the parser to enforce the LP constraint.

It is Chomsky Normal Form that allows Logic D1C to exist in this form. When a CFG is not in Chomsky Normal Form, it can have any number of nonterminals or terminals on the RHS of a production. The CKY algorithm requires that the grammar have terminal productions with *one terminal* on the RHS.

To explore why this restriction is useful, let us first suppose that the CKY algorithm restricted the number of terminals on the RHS to two instead of one. Instead of having n input word axioms like the ones in Table 2.1, where n is the length of the input, the parser would have to load $2n$ axioms, one for each word and then one for each adjacent set of words. Allowing three terminals on the RHS leads to $3n$ axioms. And if the number of terminals on the RHS were unbounded, the algorithm would require $O(n^2)$ axioms. Increasing the number of axioms also requires a generalization of the Scan inference so that it can handle arbitrary numbers of terminals on the RHS. So, as the number of terminals on the RHS goes up, the number of axioms in the algorithm goes rapidly up and the complexity of the explanation increases, since there are more special cases.

The CKY algorithm for CFG also requires that there be no ε in any production. This restriction also leads to efficiency and simplification of the algorithm. If ε were allowed on the RHS of productions, then the axioms would have to load into the chart all of the nonterminals that yield an ε, which is inefficient. In addition, these items would have no span, so they could compose with any other nonterminal at any time, possibly leading to even more inefficiencies.

So, the single terminal restriction and ε restriction are both important aspects of CNF that make CKY possible.

2.5 Algorithms for Converting a GMTG to GCNF

The algorithm to convert a GMTG G_0 to a new grammar G_{GCNF} in GCNF has several steps detailed below.

(i) isolate terminals

(ii) binarize productions

(iii) remove ε's

(iv) eliminate unit productions

(v) eliminate useless links and terminals

Parts of the GCNF conversion algorithm and some of the proofs presented here are generalizations to the multidimensional case with discontinuities of the proofs presented in Hopcroft et al. (2001) for the transformation of a CFG to CNF. The ordering of these steps is important, as some steps can restore conditions that others eliminate. In the CNF procedure in Hopcroft et al. (2001), the terminal isolation and binarization steps come last, but the alternative order above is more efficient because it reduces the number of productions that can be created during ε-elimination. This is a result of the exponential nature of ε-elimination step, shown later. In the worst case, if a grammar has a fan-out of f, a rank of r and n productions, the ε-elimination procedure can create $(2^r f) * n$ new productions. Guaranteeing that the $rank <= 2$ can have a huge effect on the efficiency of this step when dealing with syncrhonous grammars. Higher ranks pose less of a problem when dealing with CFGs since f is always one.

2.5.1 Step 1: Terminal Isolation

The RHS of a GMTG production can mix terminals and nonterminals, which is not allowed in GCNF. In addition, GMTG allows more than one terminal on the RHS. We would like to arrange it so that terminals always appear in productions without any other symbols on the RHS (including the symbol ε). Our goal is to identify all the terminals in the productions where they don't appear as the only symbol, and replace each of those terminals with a nonterminal that eventually yields it. We do this by replacing the set of terminals on the RHS of each of these non-isolated productions with a single link, and then adding a new production $\overline{p}_1$ with that link as its LHS. On the RHS of $\overline{p}_1$, there is a new unique nonterminal for each terminal being replaced. Each nonterminal has a unique index. Lastly, for each nonterminal N in component d on the RHS of $\overline{p}_1$, there will be a new production with N on the LHS and the terminal it is associated with on the RHS in component d. All the other components are inactive. This process is repeated until all the non-isolated terminals have been replaced with nonterminals. During this process, the rank of any production with non-isolated terminals is increased by one, and therefore the rank of the grammar could be increased by one.

Lemma 2.1. *If the grammar G_2 is constructed from G_1 by isolating terminals using the above construction, then $L(G_2) = L(G_1)$.*

Proof. The construction that converts G_1 to a grammar with no non-isolated terminals G_2, does so in such a way that each production of G_1 can be simulated by one or more productions of G_2. Conversely, the introduced nonterminals of G_2 each have only one production, so they can only be used in the manner intended. More formally, we prove that there sentential form of terminals $\overline{w}$ is in $L(G_1)$ iff $\overline{w}$ is in $L(G_2)$.

(Only-if) If $\overline{w}$ has a derivation in G_1, it is easy to replace each production used by a sequence of productions of G_2. That is, one step in the derivation in G_1 becomes one or more steps in G_2. If any item on the RHS of that production is a non-isolated terminal, we know G_2 has a version of the link with a nonterminal that yields it. We conclude that there is a derivation of $\overline{w}$ in G_2, so $\overline{w}$ is in $L(G_2)$.

(If) Suppose $\overline{w}$ is in $L(G_2)$. Then there is a parse tree in G_2 with $\overline{\$}$ at the root which yields $\overline{w}$. We convert this tree into a parse tree of G_1 that also has a root $\overline{\$}$ and yields $\overline{w}$.

The terminal separation step in the GCNF construction introduced other links that derive terminals. We can identify those in the current parse, and replace them with the terminal they yield. Now every interior node of the parse tree forms a production of G_1. Since $\overline{w}$ is the yield of a parse tree in G_1, we conclude that $\overline{w}$ is in $L(G_1)$ ∎

Example: Consider the example grammar $G_1 = (V_{N1}, V_{T1}, P_1, S)$, where P_1 contains the following productions:

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1D^2B^1)\\(D^2B^1)\end{bmatrix} \tag{2.29}$$

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(bC^1,D^2)\\(C^1D^2b)\end{bmatrix} \tag{2.30}$$

$$\begin{bmatrix}(C)\\(C)\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\(c)\end{bmatrix} \tag{2.31}$$

$$\begin{bmatrix}(D)\\(D)\end{bmatrix} \Rightarrow \begin{bmatrix}(\varepsilon)\\(d)\end{bmatrix} \tag{2.32}$$

We want to convert it to a new grammar, $G_2 = (V_{N2}, V_{T2}, P_2, S)$. We are interested in altering

any productions that mix terminals and nonterminals, or that have more than one terminal on the RHS. Three productions have non-isolated terminals in G_1, (2.30), (2.31) and (2.32). First we examine (2.30). We create a new link $[(B_1), (B_1)]$ and replace the terminals with it. We create two new nonterminals, B_2 and B_3, one for each of the terminals that must be replaced, and add them to V_{N2}. We then add the following productions to P_2:

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_2^1)\\(B_3^2)\end{bmatrix} \tag{2.33}$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \tag{2.34}$$

$$\begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \tag{2.35}$$

We alter production (2.30) by replacing the terminals with a new link that yields them, shown in bold:

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathbf{B_1^3}C^1, D^2)\\(C^1D^2\mathbf{B_1^3})\end{bmatrix} \tag{2.36}$$

We do the same thing to production (2.31). We create three new productions:

$$\begin{bmatrix}(C_1)\\(C_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_2^1)\\(C_3^1)\end{bmatrix} \tag{2.37}$$

$$\begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \tag{2.38}$$

$$\begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \tag{2.39}$$

and alter (2.31) to look like:

$$\begin{bmatrix}(C)\\(C)\end{bmatrix} \Rightarrow \begin{bmatrix}C_1^1\\C_1^1\end{bmatrix} \tag{2.40}$$

Lastly we do the same to (2.32). So our new set of productions P_2 is seen in Figure 2.1.

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1D^2B^1)\\(D^2B^1)\end{bmatrix} \quad (2.41)$$

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_1^3C_1^1,D^2)\\(C_1^1D^2B_1^3)\end{bmatrix} \quad (2.42)$$

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_2^1)\\(B_3^2)\end{bmatrix} \quad (2.43)$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \quad (2.44)$$

$$\begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \quad (2.45)$$

$$\begin{bmatrix}C\\C\end{bmatrix} \Rightarrow \begin{bmatrix}(C_1^1)\\(C_1^1)\end{bmatrix} \quad (2.46)$$

$$\begin{bmatrix}(C_1)\\(C_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_2^1)\\(C_3^2)\end{bmatrix} \quad (2.47)$$

$$\begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \quad (2.48)$$

$$\begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \quad (2.49)$$

$$\begin{bmatrix}(D)\\(D)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_1^1)\\(D_1^1)\end{bmatrix} \quad (2.50)$$

$$\begin{bmatrix}(D_1)\\(D_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_2^1)\\(D_3^2)\end{bmatrix} \quad (2.51)$$

$$\begin{bmatrix}(D_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(\varepsilon)\\()\end{bmatrix} \quad (2.52)$$

$$\begin{bmatrix}()\\(D_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.53)$$

Figure 2.1: Our sample grammar after isolating terminals.

2.5.2 Step 2: Binarization

The second step of converting to GCNF is a 'binarization' of the productions, reducing the rank of the grammar to two. In the case of a CFG, binarization can easily be carried out: two adjacent nonterminals are replaced with a single nonterminal that yields them. In contrast, it might be impossible to transform a D-GMTG having rank r and fan-out f into an equivalent D-GMTG having rank 2 and the same fan-out. As a general statement, for every fan-out $f \geq 2$ and rank $r \geq 4$, there are some index orderings that can be represented by a D-GMTG of rank r and fan-out f but not by any D-GMTG of rank $r-1$ and fan-out f. This statement follows from results presented in (Rambow and Satta, 1999). The distinguishing characteristic of the index orderings mentioned above is apparent in Figure 2.2, which shows a production in a grammar with fan-out two, and a graph that illustrates which nonterminals are coindexed. No two nonterminals are adjacent in more than one string, so replacing any two nonterminals with a single nonterminal causes a discontinuity. Increasing the fan-out of the grammar allows a single nonterminal to yield two nonadjacent nonterminals in a single string. ITG (Wu, 1997) cannot express such structures in a binarized form since it does not allow discontinuous constituents. This example is similar to the examples given earlier in Figure 1.1 on page 5 and Figure 1.3 on page 6.

(a) $\begin{bmatrix}(\mathrm{S})\\(\mathrm{S})\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathrm{N}^1_{\mathrm{Pat}} & \mathrm{V}^2_{\mathrm{went}} & \mathrm{P}^3_{\mathrm{home}} & \mathrm{A}^4_{\mathrm{early}})\\(\mathrm{P}^3_{\mathrm{damoy}} & \mathrm{N}^1_{\mathrm{Pat}} & \mathrm{A}^4_{\mathrm{rano}} & \mathrm{V}^2_{\mathrm{pashol}})\end{bmatrix}$

(b)

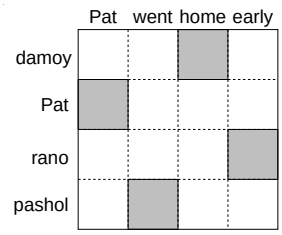

Figure 2.2: (a) A production that requires an increased fan-out to binarize. (b) A graphical representation of the production.

To binarize, we arbitrarily split each nonterminal production $\overline{p}_0$ of rank r greater than 2 into two nonterminal productions $\overline{p}_1$ and $\overline{p}_2$ of rank strictly smaller than r, but possibly with a higher fan-out than $\overline{p}_0$. This recursive application of the algorithm to productions of rank greater than two will reduce the rank of the grammar to two. Below, we say that two occurrences of nonterminals are **neighbors** if they are located at adjacent positions in some string. We binarize by the following algorithm:

(i) Arbitrarily select n links from the RHS of $\overline{p}_0$, where $2 \leq n \leq r-1$. Call these links the **s-links** of $\overline{p}_0$. Select also all the maximal length strings in the RHS of production components of $\overline{p}_0$, that are formed by neighbor occurrences from the s-links only. Call β_{ij} the j-th such string from left to right in the RHS of the i-th production component of $\overline{p}_0$.

(ii) Create a new production $\overline{p}_1$ by replacing all the β_{ij} in $\overline{p}_0$ with fresh indexed nonterminals $B^{(t)}_{ij}$.

(iii) Create a new production $\overline{p}_2$ of the form

$$\begin{bmatrix}(B_{11},\ldots,B_{1q_1})\\ \vdots \\ (B_{D1},\ldots,B_{Dq_D})\end{bmatrix} \Rightarrow \begin{bmatrix}(\beta_{11},\ldots,\beta_{1q_1})\\ \vdots \\ (\beta_{D1},\ldots,\beta_{Dq_D})\end{bmatrix}. \tag{2.54}$$

For example, binarization of the production in Figure 2.2 requires that we introduce a new

production of fan-out three. The binarized productions are as follows:

$$\begin{bmatrix}(\mathrm{S})\\(\mathrm{S})\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathrm{N}^1_{\mathrm{Pat}}\mathrm{VP}^2)\\(\mathrm{VP}^2\mathrm{N}^1_{\mathrm{Pat}}\mathrm{VP}^2)\end{bmatrix} \tag{2.55}$$

$$\begin{bmatrix}(\mathrm{VP})\\(\mathrm{VP},\mathrm{VP})\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathrm{V}^1\mathrm{A}^2_{\mathrm{early}})\\(\mathrm{V}^1,\mathrm{A}^2_{\mathrm{rano}}\mathrm{V}^1)\end{bmatrix} \tag{2.56}$$

$$\begin{bmatrix}(\mathrm{V})\\(\mathrm{V},\mathrm{V})\end{bmatrix} \Rightarrow \begin{bmatrix}(\mathrm{V}^1_{\mathrm{went}}\mathrm{P}^2_{\mathrm{home}})\\(\mathrm{P}^2_{\mathrm{damoy}},\mathrm{V}^1_{\mathrm{pashol}})\end{bmatrix} \tag{2.57}$$

Increasing the fan-out can be necessary even for binarizing a 1-GMTG production, as in:

$$[(\mathrm{B,B})] \Rightarrow [(\mathrm{N}^1\mathrm{V}^2\mathrm{P}^3\mathrm{A}^4,\mathrm{P}^3\mathrm{N}^1\mathrm{A}^4\mathrm{V}^2)] \tag{2.58}$$

Lemma 2.2. *If the grammar G_3 is constructed from G_2 by binarizing productions using the above construction, then $L(G_3) = L(G_2)$.*

Proof. The construction that converts G_2 to a grammar G_3 of rank smaller than or equal to 2 does so in such a way that each production of G_2 can be simulated by one or more productions of G_3. Conversely, the introduced nonterminals of G_3 each have only one production, so they can only be used in the manner intended. More formally, we prove that the sentential form of terminals $\overline{w}$ is in $L(G_2)$ iff $\overline{w}$ is in $L(G_3)$.

(Only-if) If $\overline{w}$ has a derivation in G_2, it is easy to replace each production used by a sequence of productions of G_3. That is, one step in the derivation in G_2 becomes one or more steps in G_3. If the RHS has more than two links in a single component, we know there are instead a sequence of links that yield them, as that is how G_3 was constructed. We conclude that there is a derivation of $\overline{w}$ in G_3, so $\overline{w}$ is in $L(G_3)$.

(If) Suppose $\overline{w}$ is in $L(G_3)$. Then there is a parse tree in G_3 with $\overline{\$}$ at the root which yields $\overline{w}$. We convert this tree into a parse tree of G_2 that also has a root $\overline{\$}$ and yields $\overline{w}$.

We "undo" the binarization step of the GCNF construction. That is, suppose there is a node labeled with the link $\overline{\Gamma}$, with two children, the links $\overline{\Psi}$ and $\overline{\Lambda}$, where $\overline{\Lambda}$ is one of the links

introduced in the binarization. Then this portion of the parse tree has under it all the links that it replaced. We can alter the derivation tree by replacing $\overline{\Lambda}$ with all those links, and leaving what the links yield untouched. Now every interior node of the parse tree forms a production of G_2. Since $\overline{w}$ is the yield of a parse tree in G_2, we conclude that $\overline{w}$ is in $L(G_2)$. ■

Example: We run this algorithm on G_2. G_2 has one production that has rank greater than 2. We examine (2.42) and apply our algorithm to it.

$$\overline{p}_0 = \begin{bmatrix} (B, B) \\ (B) \end{bmatrix} \Rightarrow \begin{bmatrix} (B_1^3\{C^1, D^2\}) \\ (\{C^1 D^2\} B_1^3) \end{bmatrix}$$

(i) We chose the links marked with indices 1 and 2 to be our s-links, and mark the neighbors with brackets.

(ii) Create new production $\overline{p}_1$ with fresh indexed nonterminal H^1:

$$\begin{bmatrix} (B, B) \\ (B) \end{bmatrix} \Rightarrow \begin{bmatrix} (B_1^3 H^1, H^1) \\ (H^1 B_1^3) \end{bmatrix} \tag{2.59}$$

(iii) Create new production $\overline{p}_2$:

$$\begin{bmatrix} (H, H) \\ (H) \end{bmatrix} \Rightarrow \begin{bmatrix} (C^1, D^2) \\ (C^1 D^2) \end{bmatrix} \tag{2.60}$$

Our production $\overline{p}_2$ now has a rank of two, so we can stop the binarization algorithm. The complete production set of G_3 is seen in Figure 2.3.

2.5.3 Step 3: ε Elimination

We now introduce an algorithm that eliminates the ε symbols from the RHS of productions of GMTGs. We denote a sentential form with only ε in every component as $\overline{\varepsilon}$. An **ε-component** is a production component where at least one string in some tuple in the RHS is ε. An **ε-production** is any production that has an ε-component. Grammars in GCNF cannot have any

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1D^2B^1)\\(D^2B^1)\end{bmatrix} \quad (2.61) \qquad \begin{bmatrix}(C_1)\\(C_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_2^1)\\(C_3^2)\end{bmatrix} \quad (2.68)$$

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_1^1H^2,H^2)\\(H^2B_1^1)\end{bmatrix} \quad (2.62) \qquad \begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \quad (2.69)$$

$$\begin{bmatrix}(H,H)\\(H)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_1^1,D^2)\\(C_1^1D^2)\end{bmatrix} \quad (2.63) \qquad \begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \quad (2.70)$$

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_2^1)\\(B_3^2)\end{bmatrix} \quad (2.64) \qquad \begin{bmatrix}(D)\\(D)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_1^1)\\(D_1^1)\end{bmatrix} \quad (2.71)$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \quad (2.65) \qquad \begin{bmatrix}(D_1)\\(D_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_2^1)\\(D_3^2)\end{bmatrix} \quad (2.72)$$

$$\begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \quad (2.66) \qquad \begin{bmatrix}(D_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(\varepsilon)\\()\end{bmatrix} \quad (2.73)$$

$$\begin{bmatrix}C\\C\end{bmatrix} \Rightarrow \begin{bmatrix}(C_1^1)\\(C_1^1)\end{bmatrix} \quad (2.67) \qquad \begin{bmatrix}()\\(D_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.74)$$

Figure 2.3: Our sample grammar after binarization.

ε-productions. We need to show that ε-productions are not essential in any GMTG, i.e. given a GMTG G_3 with ε-productions, we can convert it to a weakly equivalent grammar G_4 without ε-productions.We must be careful here. If an ε exists in a string in a production of G_3, we will instead have an empty component in some corresponding production of G_4. Thus the two grammars are equivalent modulo the ε-components. Consider a production $\overline{p}$ of the form

$$\overline{p} \;=\; \begin{bmatrix}(A_{11},\ldots,A_{1q_1})\\ \vdots \\ (A_{D1},\ldots,A_{Dq_D})\end{bmatrix} \;\rightarrow\; \begin{bmatrix}(\alpha_{11},\ldots,\alpha_{1q_1})\\ \vdots \\ (\alpha_{D1},\ldots,\alpha_{Dq_D})\end{bmatrix}. \quad (2.75)$$

We define an **addresses** of $\overline{p}$ as an integer pair, (d,q), with $1 \le d \le D$ and $1 \le q \le q_d$. Here d represents a production component, and q marks some string in the RHS of that component.

Our strategy is to start by discovering which links, and in them which symbols, are nullable, that is, can derive the empty string ε. Once these links are found, we will eliminate the productions with nullable links on their RHS, and replace them with one or more productions that yield the same terminal ITVs. Consider a link $\overline{\Gamma}$ and assume that $\overline{\Gamma} \stackrel{*}{\Rightarrow} \overline{\gamma}$, where $\overline{\gamma} = [(\gamma_{11},\ldots,\gamma_{1q_1}),\ldots,(\gamma_{D1},\ldots,\gamma_{Dq_D})]$ and at least one string γ_{dq} is ε. Then we say that the link $\overline{\Gamma}$ is **nullable** at address (d,q). If every string γ_{dq} is ε, we call the link **fully nullable**.

Let $G_3 = (V_{N3}, V_{T3}, P_3, S)$ be a GMTG. We can find all the nullable links in G_3, and the corresponding addresses, by the following iterative algorithm.

BASIS: Let $\overline{p}$ be a production in P_3 having the form in (2.75), and assume $\alpha_{dq} = \varepsilon$ for some d and q with $1 \leq d \leq D$ and $1 \leq q \leq q_d$. Then by definition the link on the LHS of $\overline{p}$ is nullable at address (d, q).

INDUCTION: Let $\overline{p}$ be a production in P_3 having the form in (2.75). Assume that for some d and q the following conditions hold:

- there are no terminal symbols in string α_{dq}; and
- every nonterminal in α_{dq} is the q'-th symbol in the d'-th tuple of some link $\overline{\Gamma}$, and $\overline{\Gamma}$ is nullable at address (d', q').

Then the link on the LHS of $\overline{p}$ is nullable at address (d, q). ■

We have shown how to detect all nullable links and their associated addresses. Now we give the construction of a grammar without ε-productions. Let $G_3 = (V_{N3}, V_{T3}, P_3, S)$ be a GMTG. First, we determine all nullable links in G_3 and the associated addresses. We then construct a new grammar $G_4 = (V_{N4}, V_{T4}, P_4, S)$, with $V_{T4} = V_{T3}$, where sets P_4 and V_{N4} are determined as described below.

Let $\overline{\Gamma} = [(A_{11}^{(t)}, \ldots, A_{1q_1}^{(t)}), \ldots, (A_{D1}^{(t)}, \ldots, A_{Dq_D}^{(t)})]$ be a link that is nullable at n distinguished addresses. The construction creates 2^n versions of the link. There is one **version** for each of the possible combinations of the nonterminals $A_{qd}^{(t)}$ present or absent, where (q, d) is an address at which the link is nullable. The version of the link with all strings present is its original version. Each non-original version of the link gets fresh nonterminals (except for start links), so that each link is unique in the grammar. For each production $\overline{p}$, we identify the nullable links on the RHS and replace them with each combination of the non-original versions found earlier. Each of these new productions is called a **version** of the original. If a string in $\overline{p}$'s RHS is left empty during this process, that string is removed from the RHS and the fan-out of the production component is reduced by one unit. As a consequence, the link on the LHS is replaced with its appropriate non-original version. There are two exceptions to the replacements. One is that, after the replacement, $\overline{p}$ consists of all empty strings in its RHS. In this case the production is

not added to P_4. The other is that if $\overline{p}$ is a start production, and all the strings on its RHS become empty after the replacement, then that RHS is replaced with ε.

Lemma 2.3. *If the grammar G_4 is constructed from G_3 by the above method for eliminating ε-productions, then $L(G_4) = L(G_3) - \overline{\varepsilon}$*

Proof. We must show that given a sentential form $\overline{w}$ of terminals with $\overline{w} \neq \overline{\varepsilon}$, $\overline{w} \in L(G_4)$ iff $\overline{w} \in L(G_3)$. We shall prove a generalization of this: Given a link $\overline{\Gamma}_4$, $\overline{\Gamma}_4 \stackrel{*}{\Rightarrow}_{G_4} \overline{w}$ if and only if $\overline{\Gamma}_3 \stackrel{*}{\Rightarrow}_{G_3} \overline{w}$ where $\overline{\Gamma}_3$ is a version of $\overline{\Gamma}_4$ created by the procedure, $\overline{w}$ is an ITV containing only terminals and $\overline{w} \neq \overline{\varepsilon}$.

(Only-if) Suppose $\overline{\Gamma}_4 \stackrel{*}{\Rightarrow}_{G_4} \overline{w}$. Then surely $\overline{w} \neq \overline{\varepsilon}$, because G_4 has no ε-productions. We must show by induction on the length of the derivation that $\overline{\Gamma}_3 \stackrel{*}{\Rightarrow}_{G_3} \overline{w}$, where $\overline{\Gamma}_4$ is a version of $\overline{\Gamma}_3$.

BASIS1: Zero Step. Then $\overline{\Gamma}_4 = \overline{w}$. By definition, since $\overline{\Gamma}_4$ is a version of the link $\overline{\Gamma}_3$, $\overline{\Gamma}_3 = \overline{w}$.

BASIS2: One step. Then there is a production $\overline{\Gamma}_4 \rightarrow \overline{w}$ in G_4. The construction of G_4 tells us that there is some version of that production $\overline{\Gamma}_3 \rightarrow \overline{\beta}$ of G_3, such that $\overline{\beta}$ is $\overline{w}$, with zero or more fully nullable links interspersed, and $\overline{\Gamma}_4$ is a version of $\overline{\Gamma}_3$. Applying productions to those fully nullable links removes them and we are left with $\overline{w}$. So in G_3, $\overline{\Gamma}_3 \Rightarrow \overline{\beta} \stackrel{*}{\Rightarrow}_{G_3} \overline{w}$, where the steps after the first, if any, derive ε from whatever links are remaining on the RHS.

INDUCTION: Suppose that the derivation takes $n > 1$ steps. Then the derivation looks like

$$\Gamma_4 = \begin{bmatrix} (\gamma_{11}, \ldots, \gamma_{1q_1}) \\ \vdots \\ (\gamma_{D1}, \ldots, \gamma_{Dq_D}) \end{bmatrix} \Rightarrow_{G_4} \begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} \stackrel{*}{\Rightarrow}_{G_4} \overline{w} \tag{2.76}$$

The first production used must have come from a production

$$\Gamma_3 = \begin{bmatrix} (\delta_{11}, \ldots, \delta_{1r_1}) \\ \vdots \\ (\delta_{D1}, \ldots, \delta_{Dr_D}) \end{bmatrix} \rightarrow \begin{bmatrix} (\beta_{11}, \ldots, \beta_{1r_1}) \\ \vdots \\ (\beta_{D1}, \ldots, \beta_{Dr_D}) \end{bmatrix} \tag{2.77}$$

where the β's are the α's in order with zero or more additional nullable variables interspersed. Also, we can break $\overline{w}$ down:

$$\overline{w} = \begin{bmatrix} (w_{11}, \ldots, w_{1q_1}) \\ \vdots \\ (w_{D1}, \ldots, w_{Dq_D}) \end{bmatrix} \tag{2.78}$$

where α_{ij} will evantually yield w_{ij} in the derivation process as part of a link.. If α_{ij} is a terminal, then $w_{ij} = \alpha_{ij}$. If it is a nonterminal, then it is part of some link. Through derivation of that link, α_{ij} will yield w_{ij} in G_4 in fewer than n steps, as will any other nonterminal in the link. By the inductive hypothesis, we know that α_{ij} will yield w_{ij} in G_3 in fewer than n steps.

Now we can construct a corresponding derivation in G_3 as follows:

$$\Gamma_3 = \begin{bmatrix} (\delta_{11}, \ldots, \delta_{1r_1}) \\ \vdots \\ (\delta_{D1}, \ldots, \delta_{Dr_D}) \end{bmatrix} \tag{2.79}$$

$$\Rightarrow_{G_3} \begin{bmatrix} (\beta_{11}, \ldots, \beta_{1r_1}) \\ \vdots \\ (\beta_{D1}, \ldots, \beta_{Dr_D}) \end{bmatrix} \tag{2.80}$$

$$\overset{*}{\Rightarrow}_{G_3} \begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} \tag{2.81}$$

$$\overset{*}{\Rightarrow}_{G_3} \begin{bmatrix} (w_{11}, \ldots, w_{1q_1}) \\ \vdots \\ (w_{D1}, \ldots, w_{Dq_D}) \end{bmatrix} \tag{2.82}$$

$$= \overline{w} \tag{2.83}$$

The first step is the application of the production 2.77 above that we know exists in G_3. The next group of steps represetns the derivation of ε from each of the β_{kl}'s that is not one of the α_{ij}'s. The final group of steps represents the derivation of the w_{ij}'s from the α_{ij}'s, which we know exist from the inductive hypothesis.

(If) Suppose $\overline{\Gamma}_3 \overset{*}{\Rightarrow}_{G_3} \overline{w}$, and $\overline{w}$ contains no ε's. We must show by induction on the length of the derivation that $\overline{\Gamma}_4 \overset{*}{\Rightarrow}_{G_4} \overline{w}$, where $\overline{\Gamma}_4$ is a version of $\overline{\Gamma}_3$.

BASIS: One step. Then $\overline{\Gamma}_3 \rightarrow \overline{w}$ is a production of G_3. Since the production contains no ε's, it is also a production in G_4, so $\overline{\Gamma}_4 \stackrel{*}{\Rightarrow}_{G_4} \overline{w}$.

INDUCTION: Suppose the derivation takes $n > 1$ steps. Then the derivation looks like

$$\begin{aligned} \Gamma_3 &= \begin{bmatrix} (\delta_{11}, \ldots, \delta_{1r_1}) \\ \vdots \\ (\delta_{D1}, \ldots, \delta_{Dr_D}) \end{bmatrix} && (2.84) \\ &\Rightarrow_{G_3} \begin{bmatrix} (\beta_{11}, \ldots, \beta_{1r_1}) \\ \vdots \\ (\beta_{D1}, \ldots, \beta_{Dr_D}) \end{bmatrix} && (2.85) \\ &\stackrel{*}{\Rightarrow}_{G_3} \overline{w}. && (2.86) \end{aligned}$$

We can break $\overline{w}$ down:

$$\overline{w} = \begin{bmatrix} (w_{11}, \ldots, w_{1r_1}) \\ \vdots \\ (w_{D1}, \ldots, w_{Dr_D}) \end{bmatrix} \tag{2.87}$$

such that each nonterminal β_{ij} will yield w_{ij}. Let

$$\begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} \tag{2.88}$$

be those of the βij's such that $wij \neq \varepsilon$. We must have at least one non nullable variable since $\overline{w} \neq \varepsilon$. Thus,

$$\begin{aligned} \Gamma_4 &= \begin{bmatrix} (\gamma_{11}, \ldots, \gamma_{1q_1}) \\ \vdots \\ (\gamma_{D1}, \ldots, \gamma_{Dq_D}) \end{bmatrix} && (2.89) \\ &\rightarrow \begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} && (2.90) \end{aligned}$$

is a production of G_4.

We claim that

$$\begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} \overset{*}{\Rightarrow}_{G_3} \overline{w} \tag{2.91}$$

since the only β_{ij}'s that are not present among the α's were used to derive ε, and thus do not contribute to the derivation of $\overline{w}$. Since each of the links derive their terminals in less than n steps, we may apply the inductive hypothesis and conclude that, if $w_{ij} \neq \varepsilon$, then β_{ij} will derive w_{ij} within G_4. Thus,

$$\Gamma_4 = \begin{bmatrix} (\gamma_{11}, \ldots, \gamma_{1q_1}) \\ \vdots \\ (\gamma_{D1}, \ldots, \gamma_{Dq_D}) \end{bmatrix} \tag{2.92}$$

$$\Rightarrow_{G_4} \begin{bmatrix} (\alpha_{11}, \ldots, \alpha_{1q_1}) \\ \vdots \\ (\alpha_{D1}, \ldots, \alpha_{Dq_D}) \end{bmatrix} \tag{2.93}$$

$$\overset{*}{\Rightarrow}_{G_4} \begin{bmatrix} (w_{11}, \ldots, w_{1q_1}) \\ \vdots \\ (w_{D1}, \ldots, w_{Dq_D}) \end{bmatrix} \tag{2.94}$$

$$= \overline{w} \tag{2.95}$$

Now we complete the proof as follows. We know $\overline{w}$ is in $L(G_4)$ iff $\overline{\$} \overset{*}{\Rightarrow}_{G_4} \overline{w}$. Letting $\overline{\$}$ be $\overline{\Gamma}_3$ and $\overline{\$}$ be $\overline{\Gamma}_4$ above, we know that $\overline{w}$ is in $L(G_4)$ iff $\overline{\$} \overset{*}{\Rightarrow}_{G_3} \overline{w}$ and $\overline{w} \neq \overline{\varepsilon}$. That is, $\overline{w}$ is in $L(G_4)$ iff $\overline{w}$ is in $L(G_3)$ and $\overline{w} \neq \overline{\varepsilon}$. ■

Example: Let us find all the nullable links in the example grammar $G_3 = (V_{N3}, V_{T3}, P_3, S)$. We display each nullable link in Table 2.2, along with the address it is nullable at and all possible versions of the link. Unique subscripts are assigned in the right column of the table to maintain the expressivity of the grammar.

Now let us construct the productions of grammar $G_4 = (V_{N4}, V_{T4}, P_4, S)$. First, we consider production (2.61 on page 43). This production contains two of our nullable links on the RHS, each containing one nullable string. So, we have four combinations of these two nullable strings

Link	Address	Versions
$\begin{bmatrix}(D_2)\\()\end{bmatrix}$	(1,1)	NA
$\begin{bmatrix}(D_1)\\(D_1)\end{bmatrix}$	(1,1)	$\begin{bmatrix}()\\(D_4)\end{bmatrix}$
$\begin{bmatrix}(D)\\(D)\end{bmatrix}$	(1,1)	$\begin{bmatrix}()\\(D_5)\end{bmatrix}$
$\begin{bmatrix}(H,H)\\(H)\end{bmatrix}$	(1,2)	$\begin{bmatrix}(H_6)\\(H_6)\end{bmatrix}$
$\begin{bmatrix}(B,B)\\(B)\end{bmatrix}$	(1,2)	$\begin{bmatrix}(B_7)\\(B_7)\end{bmatrix}$

Table 2.2: Nullable links

being present and absent. We are left with the following productions, with the nullable strings in bold:

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1\mathbf{D^2B^1})\\(D^2B^1)\end{bmatrix} \tag{2.96}$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_7^1\mathbf{D^2})\\(D^2B_7^{(1))}\end{bmatrix} \tag{2.97}$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1\mathbf{B^1})\\(D_5^2B^1)\end{bmatrix} \tag{2.98}$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_7^1)\\(D_5^2B_7^1)\end{bmatrix} \tag{2.99}$$

Production (2.63) contains a nullable link as well, with one nullable string. So there are two versions of the production, one with the nullable string present, and one with it absent:

$$\begin{bmatrix}(H,H)\\(H)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1,\mathbf{D^2})\\(C^1D^2)\end{bmatrix} \;\Big|\; \begin{bmatrix}(C^1)\\(C^1D_5^2)\end{bmatrix} \tag{2.100}$$

The same thing is applied to Production (2.62). Production (2.73) has a component with only an ε in it, so the component is replaced with the inactive component. This removes the production completely from the grammar. Productions (2.71) and (2.72) are also altered. The productions of P_4 are seen in Figure 2.4.

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1D^2B^1)\\(D^2B^1)\end{bmatrix} \quad (2.101)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_7^1D^2)\\(D^2B_7^1)\end{bmatrix} \quad (2.102)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1B^1)\\(D_5^2B^1)\end{bmatrix} \quad (2.103)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_7^1)\\(D_5^2B_7^1)\end{bmatrix} \quad (2.104)$$

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_1^2H^1,H^1)\\(H^1B_1^2)\end{bmatrix} \quad (2.105)$$

$$\begin{bmatrix}(B_7)\\(B_7)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_1^2H_6^1)\\(H_6^1B_1^2)\end{bmatrix} \quad (2.106)$$

$$\begin{bmatrix}(H,H)\\(H)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1,D^2)\\(C^1D^2)\end{bmatrix} \quad (2.107)$$

$$\begin{bmatrix}(H_6)\\(H_6)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1)\\(C^1D_5^2)\end{bmatrix} \quad (2.108)$$

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_2^1)\\(B_3^2)\end{bmatrix} \quad (2.109)$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \quad (2.110)$$

$$\begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \quad (2.111)$$

$$\begin{bmatrix}C\\C\end{bmatrix} \Rightarrow \begin{bmatrix}(C_1^1)\\(C_1^1)\end{bmatrix} \quad (2.112)$$

$$\begin{bmatrix}(C_1)\\(C_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_2^1)\\(C_3^2)\end{bmatrix} \quad (2.113)$$

$$\begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \quad (2.114)$$

$$\begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \quad (2.115)$$

$$\begin{bmatrix}(D)\\(D)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_1^1)\\(D_1^1)\end{bmatrix} \quad (2.116)$$

$$\begin{bmatrix}()\\(D_5)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(D_4)\end{bmatrix} \quad (2.117)$$

$$\begin{bmatrix}(D_1)\\(D_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(D_2^1)\\(D_3^1)\end{bmatrix} \quad (2.118)$$

$$\begin{bmatrix}()\\(D_4)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(D_3^1)\end{bmatrix} \quad (2.119)$$

$$\begin{bmatrix}()\\(D_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.120)$$

Figure 2.4: Our sample grammar after removing ε's.

2.5.4 Step 4: Elimination of Unit Productions

A unit production is a production of rank one with no terminals. This means that the RHS is composed of a single link. We write such a unit production in short hand as $\overline{\Gamma} \Rightarrow \overline{\Lambda}$, where both $\overline{\Gamma}$ and $\overline{\Lambda}$ are links such that $\overline{\Gamma}$ is the LHS of the unit production and $\overline{\Lambda}$ constitutes the RHS.

Given a grammar $G_4 = (V_{N4}, V_{T4}, P_4, S)$, we find all those **unit pairs** $(\overline{\Gamma}, \overline{\Lambda})$ such that $\overline{\Gamma} \stackrel{*}{\Rightarrow} \overline{\Lambda}$ using a series of unit productions only. Once we have determined all such unit pairs, we replace any sequence of derivation steps in which $\overline{\Gamma} \Rightarrow \overline{\Lambda_1} \Rightarrow \overline{\Lambda_2} \Rightarrow \ldots \Rightarrow \overline{\Lambda} \Rightarrow \overline{\psi}$ by the production that uses the non-unit production $\overline{\Gamma} \Rightarrow \overline{\psi}$, where $\overline{\psi}$ is a non-unit ITV. Lastly we remove all unit productions from the grammar. The resulting production set is P_5 in $G_5 = (V_{N5}, V_{T5}, P_5, S)$.

Here is the inductive construction of all unit pairs $(\overline{\Gamma}, \overline{\Lambda})$, such that $\overline{\Gamma} \stackrel{*}{\Rightarrow} \overline{\Lambda}$ using only unit productions.

BASIS: $(\overline{\Gamma}, \overline{\Gamma})$ is a unit pair for any link $\overline{\Gamma}$. That is, $\overline{\Gamma} \stackrel{*}{\Rightarrow} \overline{\Gamma}$ in zero steps.

INDUCTION: Suppose we have determined that $(\overline{\Gamma}, \overline{\Lambda})$ is a unit pair and $\overline{\Lambda} \Rightarrow \overline{\Psi}$ is a unit production, where $\overline{\Psi}$ is a link. Then $(\overline{\Gamma}, \overline{\Psi})$ is a unit pair. ■

Once we have all unit pairs, we take the first member of a pair as the head, and all the non-unit bodies for the second member of the pair as the production bodies.

Lemma 2.4. *If the grammar G_5 is constructed from G_4 by the above construction for eliminating unit-productions, then $L(G_5) = L(G_4)$*

Proof. We shall show that $\overline{w}$ is in $L(G_5)$ iff it is in $L(G_4)$

(IF) Suppose $\overline{w}$ is in $L(G_5)$. Then $\overline{\$} \stackrel{*}{\Rightarrow}_{G_4} w$. Since every production of G_5 is equivalent to a sequence of zero or more unit productions of G_4 followed by a non-unit production of G_4, we know $\overline{\Gamma} \Rightarrow_{G_5} \overline{\beta}$ implies $\overline{\Gamma} \stackrel{*}{\Rightarrow}_{G_4} \overline{\beta}$. That is, every step of a derivation in G_5 can be replaced by one or more derivation steps in G_4. If we put these sequences of steps together, we conclude that $\overline{\$} \stackrel{*}{\Rightarrow}_{G_4} \overline{w}$ and so $\overline{w}$ is in $L(G_4)$.

(Only-if) Suppose now that $\overline{w}$ is in $L(G_4)$. Then we know that $\overline{w}$ has a leftmost derivation, i.e. $\overline{\$} \Rightarrow_{lm} \overline{w}$. Whenever a unit production is used in a leftmost derivation, the link that constitutes the RHS becomes the leftmost link, and so it is replaced immediately. Thus, the

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1D^2B^1)\\(D^2B^1)\end{bmatrix} \quad (2.121)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1_7D^2)\\(D^2B^1_7)\end{bmatrix} \quad (2.122)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1B^1)\\(D^2_5B^1)\end{bmatrix} \quad (2.123)$$

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1_7)\\(D^2_5B^1_7)\end{bmatrix} \quad (2.124)$$

$$\begin{bmatrix}(B,B)\\(B)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^2_1H^1,H^1)\\(H^1B^2_1)\end{bmatrix} \quad (2.125)$$

$$\begin{bmatrix}(B_7)\\(B_7)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^2_1H^1_6)\\(H^1_6B^2_1)\end{bmatrix} \quad (2.126)$$

$$\begin{bmatrix}(H,H)\\(H)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1,D^2)\\(C^1D^2)\end{bmatrix} \quad (2.127)$$

$$\begin{bmatrix}(H_6)\\(H_6)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1)\\(C^1D^2_5)\end{bmatrix} \quad (2.128)$$

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B^1_2)\\(B^2_3)\end{bmatrix} \quad (2.129)$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \quad (2.130)$$

$$\begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \quad (2.131)$$

$$\begin{bmatrix}(C)\\(C)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1_2)\\(C^2_3)\end{bmatrix} \quad (2.132)$$

$$\begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \quad (2.133)$$

$$\begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \quad (2.134)$$

$$\begin{bmatrix}()\\(D_5)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.135)$$

$$\begin{bmatrix}()\\(D_4)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.136)$$

$$\begin{bmatrix}()\\(D_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.137)$$

Figure 2.5: Our sample grammar after removing unit productions.

leftmost derivation in grammar G_4 can be broken down into a sequence of steps in which zero or more unit productions are followed by a non-unit production. Any non-unit production that is not preceded by a unit production is a "step" by itself. Each of these steps can be performed by one production of G_5, because the construction of G_5 created exactly the productions that reflect zero or more unit productions followed by a non-unit production. Thus $\overline{\$} \stackrel{*}{\Rightarrow}_{G_5} \overline{w}$, and so $\overline{w}$ is in $L(G_5)$. ∎

Example: We apply this process to our grammar $G_4 = (V_{N4}, V_{T4}, P_4, S)$ to get a new grammar $G_5 = (V_{N5}, V_{T5}, P_5, S)$ where $V_{T5} = V_{T4}$ and $V_{N5} = V_{N4}$. Productions (2.112), (2.116), (2.117), (2.118) and (2.119) are the unit productions in G_4. Following the algorithm leads to many new productions, as well as the removal of the five unit productions. Our new set of productions, P_5, is seen in Figure 2.5.

2.5.5 Step 5: Elimination of Useless Productions and Symbols

Let X be either a link or a terminal. We say X is **useful** for a GMTG G if there is some derivation of the form

$$\overline{\$} \stackrel{*}{\Rightarrow} \overline{\alpha} \stackrel{*}{\Rightarrow} \overline{w} \tag{2.138}$$

where X is contained in the sentential form $\overline{\alpha}$, and where $\overline{w}$ is a sentential form containing only terminals and ()'s. If X is not useful, we say it is **useless**. Evidently, omitting useless links and terminals from our grammar will not change the language generated, so we eliminate them.

For X to be useful, it must have both of the following properties:

(i) X is **generating** if it is a terminal or if it is a link such that $X \stackrel{*}{\Rightarrow} \overline{w}$ for some ITV $\overline{w}$ which contains only terminals in its nonempty components.

(ii) X is **reachable** if there is a derivation

$$\overline{\$} \stackrel{*}{\Rightarrow} \overline{\alpha} \tag{2.139}$$

such that X is contained in the tuple-vector $\overline{\alpha}$.

An X that is useful will be both generating and reachable. If we eliminate all non-generating links, and then eliminate all non-reachable links and terminals, we shall have only useful links and terminals left.

To compute the generating links and terminals of G, we perform the following induction.

BASIS: Every terminal is generating by definition.

INDUCTION: Suppose there is a production, $\overline{\Gamma} \Rightarrow \overline{\alpha}$, where every symbol of $\overline{\alpha}$ is known to be either a terminal or part of a generating link. Then the link $\overline{\Gamma}$ is generating. ■

To compute the reachable links and terminals of G, we perform the following induction.

BASIS: $\overline{\$}$, the special start link, is reachable by definition.

INDUCTION: Suppose we have discovered that some link $\overline{\Gamma}$ is reachable. Then for all productions with $\overline{\Gamma}$ as their LHS, all the links and terminals on the RHS are reachable. ■

Once identified, the following algorithm removes useless symbols from the grammar $G_5 = (V_{N5}, V_{T5}, P_5, S)$:

(i) First, eliminate non-generating links, and all productions that have a non-generating link in them. We call the remaining grammar $G'_5 = (V'_{N5}, V'_{T5}, P'_5, S)$.

(ii) Second, eliminate all terminals and links that are not reachable, and eliminate all productions with these terminals and links. The remaining grammar is $G_6 = (V_{N6}, V_{T6}, P_6, S_prime)$.

Lemma 2.5. *Applying the above process to a GMTG G_5 will create a new GMTG, G_6, with no useless links or terminals.*

Proof. Suppose X is a link or terminal that remains. If X is a terminal, it is generating by definition and so it's in G'_5. If X is a link, we know that $X \stackrel{*}{\Rightarrow}_{G_5} \overline{w}$ for some sentential form $\overline{w}$. Moreover, every link and terminal used in the derivation of $\overline{w}$ is generating. Thus $X \stackrel{*}{\Rightarrow}_{G'_5} \overline{w}$

Since X was not eliminated in the second step, we also know that there is a sentential form $\overline{\alpha}$ such that $\overline{\$} \stackrel{*}{\Rightarrow}_{G'_5} \overline{\alpha}$, where X is in $\overline{\alpha}$. Further, every link and terminal used in this derivation is reachable, so $\overline{\$} \stackrel{*}{\Rightarrow}_{G_6} \overline{\alpha}$.

We know that every link and terminal in $\overline{\alpha}$ is reachable, and we also know all these links and terminals are in G'_5. The derivation of some terminal string, say $\overline{\alpha} \stackrel{*}{\Rightarrow}_{G'_5} \overline{v}$, where $\overline{v}$ contains $\overline{w}$, involves only links and terminals that are reachable from $\overline{\$}$, because they are reached by symbols in $\overline{\alpha}$. Thus, this derivation is also a derivation of G_6; that is, $\overline{\$} \stackrel{*}{\Rightarrow}_{G_6} \overline{\alpha} \stackrel{*}{\Rightarrow}_{G_6} \overline{v}$, where X is in $\overline{\alpha}$ and $\overline{w}$ is in $\overline{v}$. We conclude that X is useful in G_6. Since X is an arbitrary link or terminal of G_6, we conclude that G_6 has no useless links or terminals. ∎

Lemma 2.6. *If the GMTG G_6 is constructed from G_5 by the above construction for eliminating useless nonterminals and links, then $L(G_6) = L(G_5)$*

Proof. First we will show $L(G_6) \subseteq L(G_5)$. Since we have only eliminated links and symbols from G_5 to make G_6, it follows that this is true.

Next we show $L(G_5) \subseteq L(G_6)$. We must prove that if $\overline{w}$ is in $L(G_5)$, then $\overline{w}$ is in $L(G_6)$. If $\overline{w}$ is in $L(G_5)$, then $\overline{\$} \stackrel{*}{\Rightarrow}_{G_5} \overline{w}$. Each symbol and link in this derivation is evidently both reachable and generating, so it is also a derivation of G_6. That is $\overline{\$} \stackrel{*}{\Rightarrow}_{G_6} \overline{w}$, and thus $\overline{w}$ is in $L(G_6)$. ∎

Example: The previous steps have created many non-generating links. We remove all productions with non-generating links and are left with $G_6 = (V_{N6}, V_{T6}, P_6, S)$, where P_6 is as follows:

$$\begin{bmatrix}(S)\\(S)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_7^1)\\(D_5^2 B_7^1)\end{bmatrix} \quad (2.140) \qquad \begin{bmatrix}()\\(B_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(b)\end{bmatrix} \quad (2.145)$$

$$\begin{bmatrix}(B_7)\\(B_7)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_1^2 H_6^1)\\(H_6^1 B_1^2)\end{bmatrix} \quad (2.141) \qquad \begin{bmatrix}(C)\\(C)\end{bmatrix} \Rightarrow \begin{bmatrix}(C_2^1)\\(C_3^2)\end{bmatrix} \quad (2.146)$$

$$\begin{bmatrix}(H_6)\\(H_6)\end{bmatrix} \Rightarrow \begin{bmatrix}(C^1)\\(C^1 D_5^2)\end{bmatrix} \quad (2.142) \qquad \begin{bmatrix}(C_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(c)\\()\end{bmatrix} \quad (2.147)$$

$$\begin{bmatrix}(B_1)\\(B_1)\end{bmatrix} \Rightarrow \begin{bmatrix}(B_2^3)\\(B_3^4)\end{bmatrix} \quad (2.143) \qquad \begin{bmatrix}()\\(C_3)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(c)\end{bmatrix} \quad (2.148)$$

$$\begin{bmatrix}(B_2)\\()\end{bmatrix} \Rightarrow \begin{bmatrix}(b)\\()\end{bmatrix} \quad (2.144) \qquad \begin{bmatrix}()\\(D_5)\end{bmatrix} \Rightarrow \begin{bmatrix}()\\(d)\end{bmatrix} \quad (2.149)$$

Theorem 2.4. *For each GMTG G_0 there exists a GMTG G_{GCNF} in GCNF generating the same set of multitexts as G but with each (ε) component in a multitext replaced by $()$.*

Proof. We have already shown that each step in the process to convert a GMTG to GCNF does not alter the language of the grammar, so we have shown that $L(G_0) = L(G_{GCNF})$. All that is left to show is that at the end of all the steps, the grammar is still in GCNF. To do this, we show that no step reinstates and condition that was eliminated in a prior ste:

(i) **isolate terminals:** There are no prior steps.

(ii) **binarize productions:** Binarizing productions of rank greater than 2 will not affect terminal productions since they are of rank 0.

(iii) **remove ε's:** Removing ε's does not affect terminal productions since they have no ε's and it can't increase the rank of any production, since nonterminals are being removed or altered, but never added, so all the productions still have a rank $<= 2$.

(iv) **eliminate unit productions:** Eliminating productions cannot alter the RHS of any production, and so it can not effect prior steps.

(v) **eliminate useless links and terminals:** Eliminating productions cannot alter the RHS of any production, and so it can not effect prior steps.

Thus, after performing the five steps above, our new grammar meets all of the conditions for a grammar in GCNF and $L(G_0) = L(G_{GCNF})$. ∎

2.6 Generalized Chomsky Normal Form vs. Wu's 2-normal Form

GCNF is not the first normal form for a synchronous grammar formalism. Wu (1997) proposes a 2-normal form that retains the binarization of nonterminals described. Unlike the "isolate terminals" step in the GCNF conversion, 2-normal form allows *two* terminals on the RHS of each production, one in each of the two components. This leads to the same efficiency and simplicity problems just described for CFG, but to an even greater degree. Suppose we wanted to multiparse an input multitext with a source sentence of length n and a target sentence of length m. The logic for 2-normal form would need an axiom for each pair of terminals found, one from the source-side and one from the target-side, leading to $n * m$ different axioms to get started. In contrast, a grammar in GCNF can start off with only $m + n$ axioms, since each terminal would match one axiom. By limiting the axioms to a single terminal on the RHS, GCNF retains the efficiency that motivated the CKY algorithm. Of course, this change pushes the combination of terminals from different components to the composition stage. However, doing so allows the grammar to score the new item during composition, and filter out the compositions that are unlikely by assigning them lower scores. Scoring the items as axioms would also require more inference rule types in the logic.

Another difference between GCNF and Wu's 2-normal form is that ε-elimination step allows ε's on the RHS of any production in 2-normal form, as long as one component is not an ε. This leads to axioms which include each terminal paired with ε, which also increases the number of

axioms to get started. Similarly, the nonterminals that yield ε's require a position, leading to spurious ambiguity when parsing.

So, just as CNF allowed the straight-forward logic description in Table 2.1, GCNF allowed Melamed and Wang (2005) to describe a new Logic meant for 2D parsing, called Logic-C, that is a direct generalization of the Logic just discussed for monolingual parsing. The logic based on GCNF is in Table 2.3. **PAV** stands for permutation array vector, and it contains the order of the nonterminals in each item. A **d-span** is a discontinuous span.

2.7 Conclusions

Generalized Multitext Grammar is a convenient and intuitive and sufficiently expressive model of parallel text. In this chapter, we formally defined GMTG and presented some properties of the grammar. We also proposed a synchronous generalization of Chomsky Normal Form, laying the foundation for synchronous CKY parsing under GMTG.

In the next chapter we describe the design of a new toolkit for generalized parsing that is capable of working with GMTG to do SMT. The simplicity of the design is made possible by GCNF.

Term Types	
terminal items	$\langle d, i, t \rangle$
nonterminal items	$\left[X_D^1; \sigma_D^1\right]$
terminating productions	$\begin{matrix} \varnothing_{d-1}^1 \\ X \\ \varnothing_D^{d+1} \end{matrix} \Rightarrow \begin{matrix} \varnothing_{d-1}^1 \\ t \\ \varnothing_D^{d+1} \end{matrix}$ for $1 \leq d \leq D$
nonterminating productions	$X_D^1 \Rightarrow\!\bowtie [\pi_D^1](Y_D^1\ Z_D^1)$
Axioms	
input words	$\langle d, i_d, w_{d,i_d} \rangle$ for $1 \leq d \leq D, 1 \leq i_d \leq n_d$
grammar terms	as given by the grammar
Inference Rule Types	
Scan component d, $1 \leq d \leq D$	$\dfrac{\langle d,i,t \rangle \ , \ \begin{matrix} \varnothing_{d-1}^1 \\ X \\ \varnothing_D^{d+1} \end{matrix} \Rightarrow \begin{matrix} \varnothing_{d-1}^1 \\ t \\ \varnothing_D^{d+1} \end{matrix}}{\begin{bmatrix} c\varnothing_{d-1}^1 & c()_{d-1}^1 \\ X & ;(i-1,i) \\ \varnothing_D^{d+1} & ()_D^{d+1} \end{bmatrix}}$
Compose	$\dfrac{\left[Y_D^1; \tau_D^1\right] \ , \ \left[Z_D^1; \sigma_D^1\right] \ , \ X_D^1 \Rightarrow\!\bowtie [\tau_D^1 \wr \sigma_D^1](Y_D^1\ Z_D^1)}{[X_D^1; \tau_D^1 + \sigma_D^1]}$

Table 2.3: Logic C: D is the dimensionality of the grammar and d ranges over dimensions; n_d is the length of the input in dimension d; i_d ranges over word positions in dimension $d, 1 \leq i_d \leq n_d$; w_{d,i_d} are input words; X, Y and Z are nonterminal symbols; t is a terminal symbol; π is a PAV; σ and τ are d-spans.

3

Design: Statistical Machine Translation By Parsing

In natural language processing, a parser is an algorithm for inferring linguistic structure. To facilitate generalization of ordinary parsers to algorithms necessary for SMT, Melamed and Wang (2005) recast them in terms of an abstract parsing algorithm with five functional parameters: a grammar, a logic, a semiring, a search strategy, and a termination condition. They then express, at a high level, all the algorithms necessary for SMT by generalizing two of those parameters — the grammar and the logic. The goal of their work was to reduce the conceptual complexity of SMT-systems driven by tree-structured translation models to facilitate research in the area.

Implementing the theory into a functioning SMT system has resulted in an open source toolkit called "GenPar", which stands for Generalized Parser. A first version was released in early 2005, and it was the first open source toolkit for machine translation by parsing. The toolkit was further developed by the "Statistical Machine Translation by Parsing" Team at the 2005 Johns Hopkins University Language Engineering Workshop. After the workshop, we continued working on the toolkit to produce some of the results in this dissertation.

While many details of the GenPar toolkit are presented in Burbank et al. (2005), this chapter seeks to reintroduce the main attributes of GenPar and present the latest research since the JHU workshop. It also addresses some software engineering concerns that one runs into when using large data sets. Later, in Chapter 4, we describe experiments using the implementation details in this chapter, and report on our findings. In Appendix A, updated software design documentation for the toolkit is presented which further details its object oriented approach to SMT.

3.1 An Introduction to GenPar

GenPar is a software toolkit for generalized parsing with an object oriented design. The design is based on the architecture laid out by Melamed and Wang (2005). The design has two goals:

(i) *flexibility*: The toolkit should support many parser variants, and should be easily configurable at run time.

(ii) *extensibility*: It should be easy to add new features and new functionality.

To satisfy the above requirements, we decompose the parser into different components and let the Parser class act as a "central controller". Each type of component corresponds to an object class family. The parser refers to each component via a pointer to the abstract type of the component. The concrete component type is instantiated during the parser's construction. A better decomposition of the parser helps the parser to support more variations and to be easily configured on demand. It also makes the parser more easily extensible. To support a new concrete component, one can merely add a new class to define the new component.

The parser implements the generic inference algorithm in Figure 3.1. Almost all parsing algorithms ever published can be viewed as special cases of this generic algorithm. The input of the algorithm is a logic, a grammar, a search strategy, a semiring, a termination condition, and a tuple of sentences to parse. The output is a semiring value, such as a Viterbi parse tree.[1] The main loop proceeds as follows. If the agenda is empty, indicating that the parsing has just started, the logic generates the first batch of inferences to initialize the agenda. These inferences are based on the logic's axioms discussed in Chapter 2. In subsequent iterations of the main loop, the parser first pops an item from the agenda called the "trigger", and then passes that item to the logic. Taking this trigger, the logic decides whether the item should be pruned with the help of a pruning strategy. If the item survives, new items are generated according to the constraints of the grammar. These items are then pushed onto the agenda. If the termination test is satisfied, then the loop terminates.

To support the generic parsing algorithm, a parser can be decomposed into the following major abstract components.

- **Logic**: a set of inference type signatures, a set of item type signatures and a set of axioms.
- **Grammar** : an evaluator of parse structures.
- **Semiring** : specifying what type of values the parser computes, and the relevant operators for computing them.[2]
- **Search Strategy**: how the inferences on the agenda are prioritized.

[1] Under other semirings, we might return a parse forest, or just a score.
[2] Only the Viterbi-derivation semiring is currently implemented.

Input: `tuple of sentences`, `Grammar`, `Logic`, `Comparator`, `TerminationCondition`

```
1: Agenda(Comparator).clear()
2: Item e = NULL;
3: repeat
4:   if not Agenda.empty() then
5:     Item e ← Agenda.pop()
6:   end if
7:   set⟨Item⟩ I ← Logic.expand(e, Grammar)
8:   for Item i ∈ I do
9:     Agenda.push(i)
10:  end for
11: until TerminationTest is satisfied
```

Output: a `semiring` value, such as a `parse tree`

Figure 3.1: Generic parsing algorithm, a special case of the abstract parsing algorithm presented by Melamed and Wang (2005).

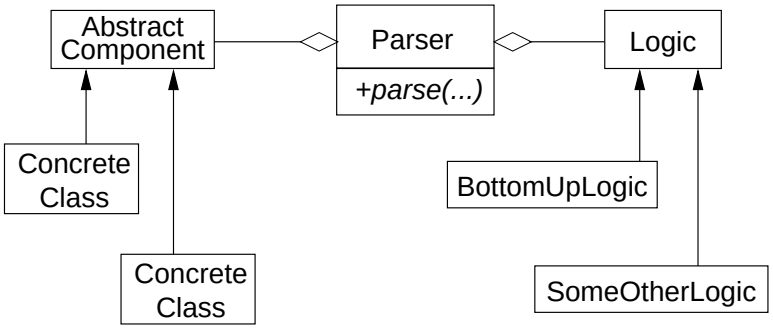

Figure 3.2: The Parser is the central component of the architecture.

- **Termination Condition**: a function that decides when to stop parsing.

There can be many varieties of each component, and each of the components has its own subcomponents. For example, the Logic generates Items for the Chart. Each of these components and subcomponents are described in the following sections. Table 3.1 shows the main classes involved in the toolkit.

The relationship between the parser and its components is one of aggregation. The implementation of the parser is realized by calling abstract operations in component base classes. Figure 3.2 shows how different abstract components make up a Parser. The types of the con-

class family	role
Parser	central controller; implements the abstract parsing algorithm
Logic	the logic (axioms and inference templates) used in the deduction
Grammar	an evaluator of parse structures
Chart	container of items already derived
Agenda	contains new chart items that have not been expanded upon
Item	represents a partial parse
Inference	inference rule in the logic
MTree	multitree, used to represent parse trees
Production	production rule (specific to WMTGs)
SentenceTuple	contains the input (generalization of ordinary sentence)
Comparator	compares two items in Agenda

Table 3.1: Prominent class families in the GenPar design

crete components are specified via the Parser constructor's arguments.[3] In the figure, there is a member pointer to Logic which will refer to the concrete BottomUpLogic or to some other Logic at runtime. There are also pointers to other abstract base classes.

The UML diagram in Figure 3.3 shows the top-level design of the parser. At the center of the diagram is the Parser, and around the Parser are its components. Among the five major parser components, the **grammar** is realized by the Grammar class family. The **logic** corresponds to the Logic class family. The **semiring** is fixed and currently corresponds to no classes. The Viterbi-derivation semiring is currently hardwired into the code, though this should not be difficult to generalize, if it's ever deemed necessary. The termination condition is implemented as the TerminationTester class family. Finally, the **search strategy** is implemented by the SearchStrategy class family.

3.1.1 The GenPar Philosophy

One of the foundations of the scientific method is that results should be replicable. Unfortunately, the majority of SMT systems in the literature have not been released to the general public to use, and their published descriptions are often less than complete. This leads to a great deal of difficulty when trying to compare two systems, slowing down overall scientific progress considerably. We tried to address these problems in GenPar in two ways.

[3] We pass the configuration information in a config file – see Appendix A.

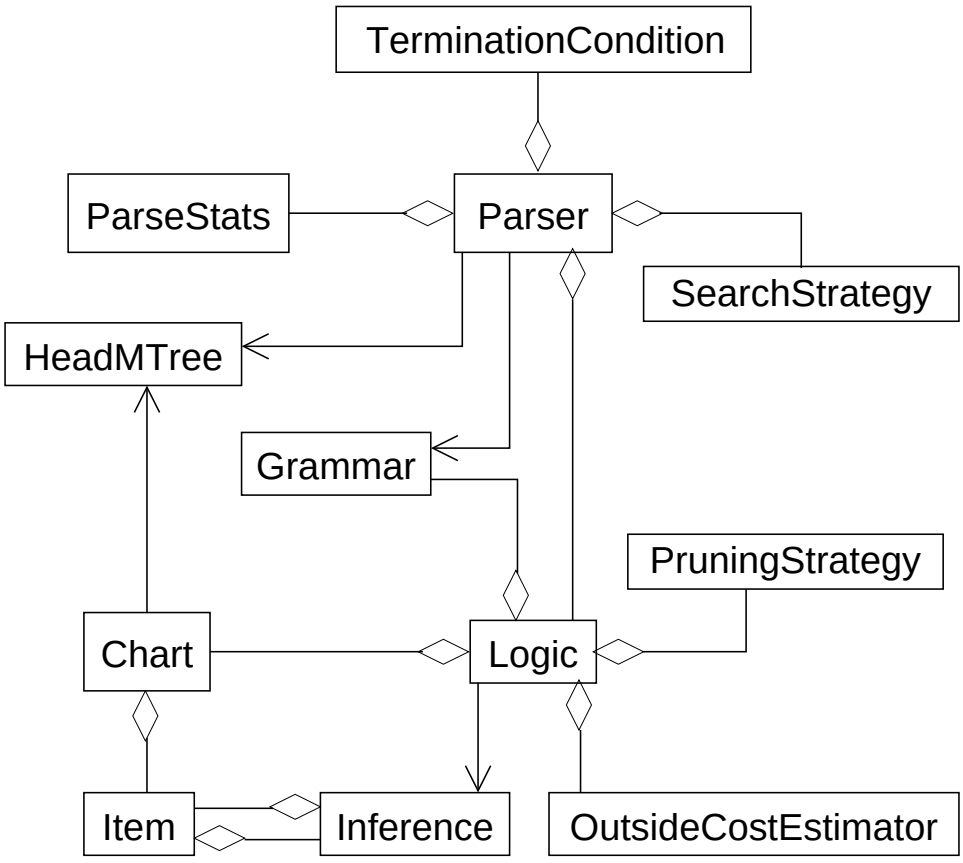

Figure 3.3: Top-level design of the generalized parser

Checkpointing

If a user of GenPar publishes a result obtained using GenPar, the user can mark the version of the code and release a "sandbox"[4] of the experiments, allowing anyone to replicate all the experiments described.

Controlled Experiments

We strived to make it easy to change just one variable in the GenPar system. So if one wanted to compare two different grammar formalisms within GenPar, they would build a new grammar and set the Grammar pointer in the parser to an instance of their own grammar, leaving all other portions of the code untouched. Pruning values, optimizations, etc. will then be uniform through all experiments, allowing scientists to truly run controlled experiments. Without this ability, controlled experiments are nearly impossible, and a lack of controlled experiments makes it difficult to compare what variable is affecting the results.

3.2 Review of SMT-by-Parsing

Melamed and Wang (2005) present a data-flow diagram for a hypothetical SMT by Parsing system, seen in Figure 3.4. Understanding the components of the diagram is useful for understanding the details put forth in this chapter, so we reprint the diagram and its description here for the reader's convenience:

T1. Induce a word-to-word translation model. Use Logic A (Melamed, 2003) with enough goal items spanning one word from each component to cover the input. Alternatively, publicly available WFST-based tools can be used (Och and Ney, 2003).

T2. Induce PCFG(s) from monolingual treebank(s), e.g. by computing the relative frequencies of productions.

T3. Hierarchically align the training multitext, e.g. using Logic C and the derivation-forest semiring. Constraining PCFG(s) and a word-to-word translation model can be used to

[4]There is more about sandboxes in Appendix A.

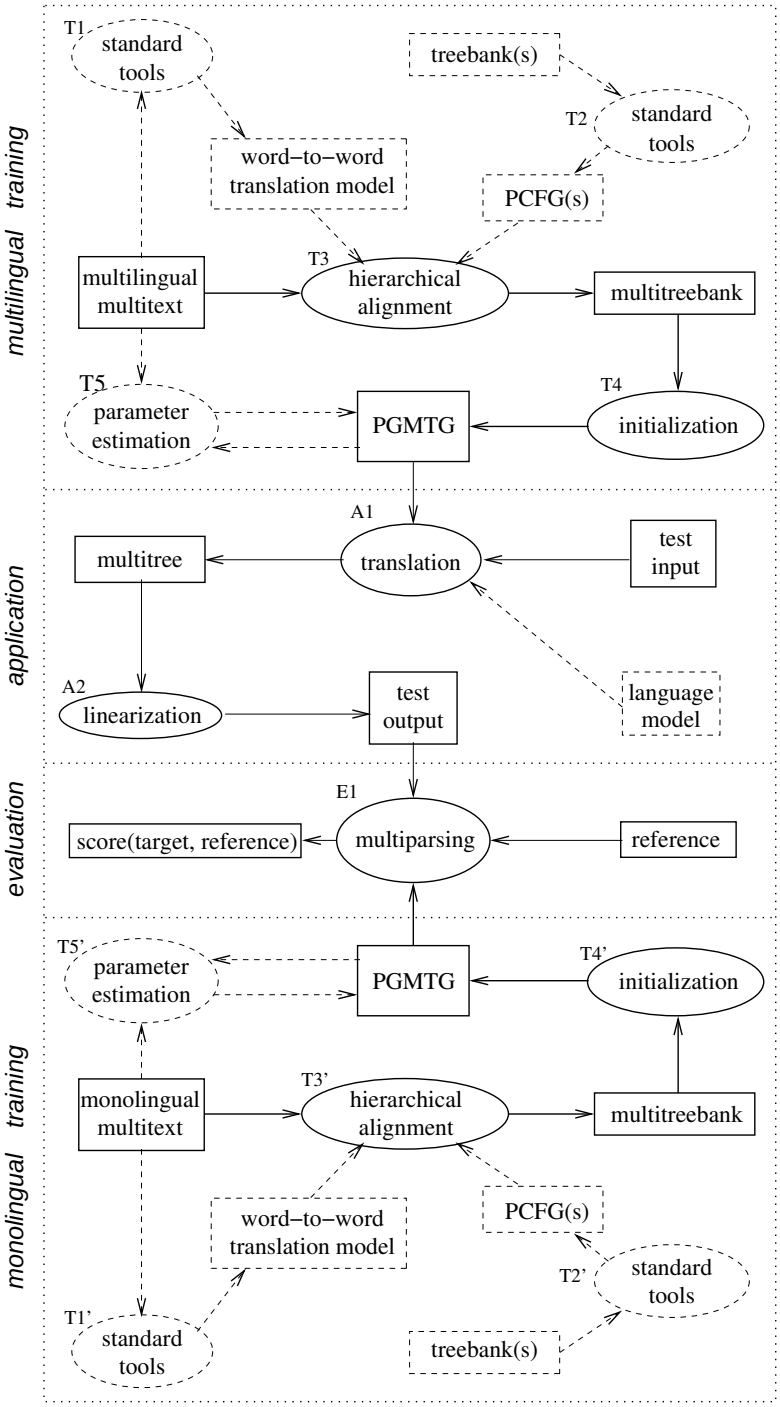

Figure 3.4: Data-flow diagram for a rudimentary system for SMT by parsing. Boxes are data; ovals are processes; arcs are flows; dashed flows and data are recommended but optional.

imitate a PGMTG. Other approximations can be used if these knowledge sources are not available or if other relevant knowledge sources are available.

T4. Induce an initial PGMTG from the multitreebank, e.g. by computing the relative frequencies of productions.

T5. Re-estimate the PGMTG parameters using Logic CR, starting with the initial PGMTG. Ideally, use the inside semiring, but if that's too expensive, then use Viterbi-n-best. In addition to the usual goal item, the termination condition involves the reverse item corresponding to each of the input axioms.

T1'-T5' Same as T1-T5, but starting with monolingual multitext. The identity relation can be used for T1' as a short-cut.

A1. Use the PGMTG to infer the most probable multitree covering the input multitext. Use Logic CT under the Viterbi-derivation semiring. If a target language model is available, use Logic CTM (Melamed, 2004a).

A2. Linearize the output yield of the multitree.

E1. For each component of the test output, multiparse the multitext consisting of this component and the corresponding reference translation, using Logic C under the inside semiring and the monolingual PGMTG.

We focus for the rest of this chapter on important implementation details for processes T3, T5, T3', T5', and A1, which are generalizations of parsing.

3.3 Grammars

A grammar is something that assigns values to partial analyses of linguistic expressions, usually in the form of a partial parse. The term conjures up images of production rules for most people, like those of a Context Free Grammar, but that is just one type of grammar. One can make a

grammar which allows any partial parse, nothing at all, or anything in between. So while the logic guides the way items are formed, it is the grammar which assigns values to each new item.

In Chapter 2, we introduced GMTG. As pointed out in that chapter, GMTG(2) has the same generative capacity as ITG, and the SMT system we use to run experiments in Chapter 4 uses GMTG(2). So it is important to point out the main differences between the GMTG(2) grammars we build and Wu's ITG. First, as we explained in Chapter 2 , GMTG's GCNF allows us to keep the logic simple and leads to efficient "scans" analogous to those found in CKY parsing for CFGs . Second, we differ from Wu's approach described above in that we vary how each item is scored when it is inserted in the chart. We can create a variety of different grammars by changing the scoring mechanism, while leaving the parsing algorithm and logic untouched, as long as each grammar is in GCNF. Good object oriented design allows us to plug in whatever scoring mechanism we would like. This is useful both from a scientific point of view, because it allows us to change only one variable during experiments, and also from an engineering point of view, since we can do software development on the scoring mechanism that is orthogonal to the rest of the SMT system.

In the following sections, we introduce several grammar types which are in the class of GMTG, each meant for performing a different task in the SMT-by-parsing framework.

3.3.1 Grammars for Hierarchical Alignment

The task of hierarchical alignment was discussed in Chapter 1, and is shown by T3 in Figure 3.4, but until this point, the implementation details indicating why our hierarchical alignment procedure is novel have been left out.

Prior work has been done on hierarchical alignment to create synchronous production rules of arbitrary rank (Galley et al., 2004). Some other work has explored binarization of synchronous context free grammars once they have been formed. For example, Zhang et al. (2006) present a procedure to binarize a Synchronous Context Free Grammar (Chiang, 2002), if it is binarizable, in $O(n^2)$. However, our approach is novel for several reasons. First, it is the first to combine binarization and hierarchical alignment into a single parsing procedure, flexibly allowing mono-

lingual parse trees on the source, target, neither or both sides. This flexibility stems from the object oriented design just discussed. The design allows us to use the same logic and binarization for hierarchical alignment as we use for translation later. More significantly, this is the first hierarchical alignment and parsing procedure that can output multitrees whose fan-out is greater than two. In other words, it is the first hierarchical alignment algorithm that allows gaps, or discontinuous constituents.

The procedure takes an input multitext, word alignments, and any monolingual parse trees (which are not necessarily binarized) and outputs a binary multitree that does not violate the constraints of any monolingual parse tree and has as low a fan-out as possible. All this is done with the same bottom up generalized parsing algorithm.

More specifically, to do binarization while performing hierarchical alignment, we create a grammar that assigns infinite cost to productions that violate the constraints of the input monolingual parse trees. To do this, we add to each item a pointer to a node in each monolingual parse tree that it is constrained by, starting with the leaves of the trees during the axiom creation. These leaves are given spans as well, indicating which terminals they cover in their respective components. So each item has a span indicating what terminals it covers, and a monolingual span in a constrained component, indicating what span the node in the monolingual tree that the item points to covers. Given an item i with a monolingual tree pointer pointing to a node n, we say the "acting parent" of i is the first node along the path from n to the root of the monolingual tree with a span that is larger than that of i in the constrained component. Then two items can compose iff they have the same acting parent. If this is the case, the new item will have a pointer to that acting parent and the process can continue. If the composition involves two items, each having span in a separate component, then the word translation score of the heads are taken into account, and are used as the score of the composition. When we are done parsing with this grammar, we are left with a binarized multitree which does not violate any of the monolingual tree inputs. So by changing only the scoring mechanism of the grammar, and adding a pointer to each item, we are able to do synchronous binarization using the same parsing framework that we later use for translation.

3.3.2 Grammars for Parameter Estimation and Translation

Maximum Likelihood GMTG

One way to score a production is to calculate the probability of the RHS given the LHS. This is a straightforward maximum likelihood calculation and it is generalized directly from the PCFG literature (Booth and Thompson, 1973). We are able to build a probabilistic GMTG by keeping counts of the LHS and RHS of each production and then creating a conditional distribution. However, if the LHS and RHS of each of our production rules includes lexical heads, this can immediately lead to poor estimates due to sparseness in our data. To combat data sparseness, we can break each production down and tell a generative story by making a series of independence assumptions. Details of this approach can be seen in Melamed and Wang (2005). We used these maximum likelihood counts as the baseline for the experiments in Chapter 4.

Tree-Transduction Grammars

To do tree transduction, i.e. the inference of one tree from another (Comon et al., 1997), we use a very similar mechanism as we do for hierarchical alignment. A pointer is kept to a node in the source tree. Any composition that violates the structure of a monolingual source tree by crossing a phrase boundary is given a cost of infinity. This means that as we parse a multitext, we are stepping through the source tree as a transducer would do and creating a grammar for tree transduction within the GenPar framework.

Feature Grammars

Given an item or multitree node, FeatureGrammars generate all of the appropriate features by using a FeatureGenerator. The FeatureGenerator is responsible for reading a configuration file to determine which features types to generate, and correctly extracting those particular features from the item or multitree node. Once the features are extracted, they are either learned by some discriminative model that is being used during training or a score is assigned to the item during testing. Productions are a special case of this because an entire production can be coded as a single feature.

Feature Grammars have allowed us to build the first fully discriminative end-to-end tree-

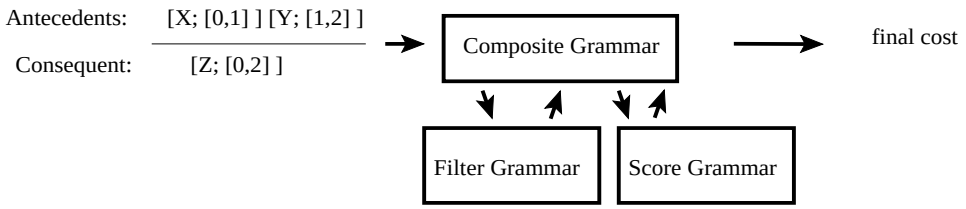

Figure 3.5: Information flow through a composite grammar.

structured SMT system. The machine learning algorithm is given feature vectors from which to predict tree-structures, node-by-node. During test time a trained classifier is given a feature vector and asked to predict the score of a particular partial tree-structure. By treating the machine learning algorithm as a black box, we are able to pick the appropriate algorithm for the task.

Composite Grammars

Recall that grammars are meant to apply scores to partial parses. Applying an idea from design patterns (Gamma et al., 1994), a composite grammar is a grammar composed of two or more subgrammars, each which contributes to the score of a partial parse. When a function is called on the composite, it decides which of its subgrammars it will call that function on, and then it combines those results and returns them. Composite Grammars are powerful because they allow us to mix and match grammars that do different things to work together.

Take for example the grammar shown in Figure 3.5. Here the parsing algorithm asks the grammar for the score of a newly created item. The Composite grammar first asks if it should be filtered or not based on one of its subgrammars. The subgrammar might check if it violates a tree constraint or not. If it fails the filter, the item is given an infinite cost. If it passes this filter, the item is then passed to the other grammar for an actual score. The second grammar may break the item down into features and use a classifier to score it.

This is how our tree transduction grammar described above works. First, a check is done to make sure the new item is not violating the input parse tree. Then the item is scored using whatever grammar is chosen as the "Score Grammar" in the figure. To run the controlled

experiments in Chapter 4, we simply test two different score grammars while leaving the rest of the setup identical.

3.4 Pruning

Although pruning is commonly used in parsing, there has not been very much discussion about its uses for multiparsing, even though it plays an important role in keeping the algorithms tractable.

3.4.1 Item Index

When deciding whether to prune a candidate item in the chart, it is usually compared with other items in the chart. Typically, in standard monolingual chart parsing, items with the same span are compared to one another as candidates for pruning. This idea has been extended by others in syntax aware SMT literature, but the standard approach has been to only compare items with the same source span as candidates for pruning (e.g. Chiang, 2005).

However, there are many ways to choose which items to compare the candidate to and this is an interesting area of research in itself. To facilitate research in this direction, we have abstracted out the selection of comparable items to an ItemIndex. Items in the same index are compared to one another but any indexing scheme can be used. So, for the standard case discussed above, each item is indexed by its source span. However, during some procedures, like the hierarchical alignment experiments discussed in Chapter 1, it is desirable to index items by their spans on both the source and target components. During translation, one might index an item on its source span and the length of the target string it yields.

3.4.2 Pruning Strategy

Once it is decided which items a candidate is going to be compared to during pruning, the actual pruning strategy comes into play. Like ItemIndexes, there are many ways to customize pruning strategies. We commonly use a combination of two strategies, easily put together with a composite design pattern. The first is the Absolute Count strategy, which says that only the

best c items are kept in each index. This means if a candidate item's index already has c items in it, each with a higher score (or lower cost), the item is pruned. Another strategy is the Relative strategy which says that an item in index i is pruned if its score is less than $t * max(i)$ where t is some threshold and $max(i)$ returns the max score in index i.

To do the alignments in Chapter 1, we used an AbsoluteCount pruning strategy with a count of 1, and an index which indexed both spans. The idea is that in those experiments, once a particular span was in the chart, there was no need to find other multitrees that cover the same span, since we were dealing with a boolean semiring; either the sentence could be hierarchically aligned or not. Aggressive pruning allowed us to parse with a GMTG(4), a task which has never been done before.

3.5 Search strategies

A parsing logic specifies how terms can be inferred, but it does not specify the order of inferences. When a parser needs an inference to evaluate, it consults its **search strategy**. Goodman (1998) required one particular search strategy for his abstract parsing algorithm, which depended on a topological sort of all possible terms. Melamed (2004b) used inference rules to specify a partial order on the computations of term values, although he allowed the order to be determined on the fly. Here we leave all ordering decisions to the search strategy, which may or may not consult the logic. A variety of search strategies are in common use. For example, the CKY algorithm (Kasami, 1965; Younger, 1967) always infers smaller items before larger ones. Alternatively, given term costs such as negative log-probabilities, we can run the parser as a uniform-cost search, inferring less costly consequents before more costly ones. If we are interested in just the single best parse or the n-best parses, then A* strategies of varying sophistication can be employed to speed up the search (Klein and Manning, 2003). The benefit of a separate search strategy is the usual benefit of abstraction: analyses of logics unobscured by search strategies are applicable to a larger class of algorithms. In Chapter 1, we used a longest first strategy, which always tried to expand the biggest item first. This made the search for a full parse significantly faster on average.

3.6 Termination Conditions

Different termination conditions are appropriate for different parsing applications. Most applications involve a goal item, such as an item that spans the input and is labeled with the start symbol of the grammar. Then the termination condition is that no further inference can change the semiring value of the goal item. Some applications involve multiple goal items. There the termination condition must hold conjunctively for all goal items.

In practice, termination conditions often cannot be expressed solely in terms of goal items and their values. For example, Earley parsing logic (Goodman, 1998, Section 2.1.1) might be used to compute the probability of a string under the inside semiring and a PCFG that is not in CNF. If the PCFG has cycles of unary productions (like $\{A \rightarrow B, B \rightarrow A\}$), then the parser will not terminate, because it will be computing an infinite sum. A typical implementation limits the computing resources that a run of the parser can expend. So, in addition to goal items, the termination condition might test the elapsed time, the size of allocated memory, or the number of inferences fired, possibly as a function of the input size.

3.7 Conclusion

We have presented implementation details for the GenPar toolkit. The software represents the first open source toolkit for performing SMT using tree-structured models. It is both extensible and flexible, and facilitates adhering to the scientific method by providing check pointing. We also presented a novel hierarchical alignment algorithm that is the first to align bitexts that require grammars with a fan-out of greater than two. Lastly, we presented details on several grammars for SMT that are in the class of GMTG.

Additional design details, including an explanation of the main classes used to perform SMT, can be found in Appendix A.

4

Application: A Discriminative System for MT by Parsing

4.1 Introduction

Discriminative training methods have recently led to significant advances in the state of the art of SMT. Another promising trend is the incorporation of syntactic information into SMT systems. Generally, combining these trends is difficult for reasons of system complexity and computational complexity.

This chapter, an extension of Wellington et al. (2006a), presents a syntax-aware SMT system whose every component is trained discriminatively. Our main innovation is an approach to discriminative learning that is computationally efficient enough for large SMT systems, yet whose accuracy on translation sub-tasks is near the state of the art. The core of our system is described in Section 4.2, and evaluated in Section 4.3.

Several studies have explored discriminative training for machine translation, following the method of Och and Ney (2002). Several authors have even applied this method to syntax-aware systems (Chiang, 2005; Quirk et al., 2005). However, Och's method applies discriminative training only to a handful of meta-parameters that are used to combine information from a variety of sub-models. The sub-models may or may not be optimized for the same objective as the meta-parameters. On a task like machine translation of common languages, where training data is abundant, a more elegant and more accurate system might be obtained by training all parts of the system discriminatively (Ng and Jordan, 2002). Foster (2000) built such a system, but could not fully exploit the benefits of discriminative training for lack of a suitable regularization scheme.

4.2 Training Method

4.2.1 The Training Set

Predicting a target tree given a source tree is equivalent to predicting a multitree that is consistent with the source tree. Our method for training tree transducers was to train an inference engine to predict multitrees. Our method for predicting multitrees is to make a sequence of multiclass

classification decisions called **inferences**. Each inference predicts an unstructured part of the eventual multitree. Thus, to train a model for predicting multitrees, it is sufficient to train it to predict correct inferences.

To generate training examples for predicting correct inferences, each training multitree t is decomposed into a partially-ordered set ("poset") of inferences, which jointly predict t and no other multitree. Next, a completely ordered sequence of correct inferences is chosen randomly, among all complete orderings that are consistent with the poset. All the inferences that can possibly follow each prefix of the correct sequence become part of the training set. All but one of the inferences generated for each prefix will be incorrect. An advantage of this method of generating training examples is that it does not require a working inference engine and can be run prior to any training. A disadvantage of this approach is that it does not teach the model to recover from mistakes.

Our approach to training multiclass classifiers is to decompose each *multiclass* training inference into a set of training examples of *binary* classification decisions, all but one of which are negative examples. For example, one binary decision might be about whether a given source word should translate to a given target word. Another might be whether two constituents should switch order in translation. After the decomposition from multiclass to binary, the training set I consists of training examples i, where each i is a tuple $\langle X(i), y(i) \rangle$. $X(i)$ is a feature vector describing i, with each element in $\{0, 1\}$. We will use $X_f(i)$ to refer to the element of $X(i)$ that pertains to feature f. $y(i) = +1$ if i is correct, and $y(i) = -1$ if not.

4.2.2 Objective Function

The discriminative toolkit we use is that of Turian et al. (2006). The training algorithm induces a hypothesis $h_\Theta(i)$, which is a real-valued example scoring function, parameterized by a real vector Θ. Θ has one entry for each possible feature f. In the present work, h_Θ is a linear model:

$$h_\Theta(i) = \Theta \cdot X(i) = \sum_f \Theta_f \cdot X_f(i) \tag{4.1}$$

	sent.	English words		French words	
	pairs	types	tokens	types	tokens
training1	10	11	210	14	232
training2	100	29	2100	38	2300
tuning	1	3.5	21	4.2	23
devel	1	3.5	21	4.1	23
test	1	3.5	21	4.1	23

Table 4.1: Data sizes in 000's.

The sign of $h_\Theta(i)$ predicts the y-value of i and the magnitude gives the confidence in this prediction.

Turian and Melamed's choice of objective R_Θ was motivated by Ng (2004), who suggested that, given a learning setting where the number of irrelevant features is exponential in the number of training examples, we can still learn effectively by minimizing the ℓ_1-regularized log-loss.

4.3 Experiments

4.3.1 Data

The data for our experiments came from the English and French components of the EuroParl corpus (Koehn, 2005). From this corpus, we extracted sentence pairs where both sentences had between 5 and 40 words, and where the ratio of their lengths was no more than 2:1. We then extracted disjoint training, tuning, development, and test sets. The tuning, development, and test sets were 1000 sentence pairs each. For some experiments we used 10,000 sentence pairs of training data; for others we used 100,000. Descriptive statistics for these corpora are in Table 4.1.

We parsed the English half of the training, tuning, development, and test multitexts using Dan Bikel's parser (Bikel, 2004), which was trained on the Penn treebank (Marcus et al., 1993). On each of our two training sets, we induced word alignments using the default configuration of GIZA++ (Och and Ney, 2003). The training set word alignments and English parse trees were fed into the default French-English hierarchical alignment algorithm distributed with the GenPar system (Burbank et al., 2005), to produce binarized tree alignments.

4.3.2 Word Transduction

Our first set of experiments evaluated our approach on the task of translating individual words from English to French. The input was a single English word, which we'll call the "focus" word, along with a vector of features (described below). The output was a single French word, possibly NULL. The proposed translation was compared to a "gold standard" translation.

The gold-standard word pairs that we used for this task were extracted from the tree alignments described above. Thus, the gold standard was a set of GIZA++ Viterbi word alignments filtered by a tree cohesion constraint. Regardless of whether they are created manually or automatically, word alignments are known to be highly unreliable. This property of the data imposed a very low artificial ceiling on all of our results, but it did not significantly interfere with our goal of controlled experiments to compare learning methods. To keep our measurements consistent across different training data sizes, the word alignments used for testing were the ones induced by GIZA++ when trained on the larger training set. The number of trials was equal to the number of source words for which GIZA++ predicts an alignment.

In contrast to Vickrey et al. (2005), we did not allow multi-word "phrases" as possible translations, because we do not yet understand how methods for compiling such phrases interact with our discriminative training methods and our objective function. In future work, we intend to explore discriminative phrase induction techniques. In the present study, phrases might have raised our absolute scores, but they would have confounded our understanding of the results. Our experiment design also differs from Vickrey et al. (2005) in that we trained classifiers for *all* words in the training data.[1] There were 161K word predictions in the smaller (10,000 sentence pairs) training set, 1866K in the larger training set, 17.8K predictions in the tuning set, 14.2K predictions in the development set, and 17.5K predictions in the test set. The number of predictions in the tuning set varied, depending on which training set was used to induce the GIZA++ model. If the smaller training set was used, then there were 14.4K predictions in the tuning set; when the larger training set was used, there were 17.8K predictions.

Using the smaller training set, and guessing the most frequent translation of each source word

[1] David Vickrey (p.c.) informed us that Vickrey et al. (2005) omitted punctuation and function words, which are the most difficult in this task.

achieves a baseline accuracy of 47.54% on the development set. With this baseline, we compared three methods for training word transducers on the word alignments described above. The first was the method described in Section 4.2. The second was a method similar to Vickrey et al. (2005), but using a maximum entropy toolkit (Daumé, 2004). Both of these methods use logistic regression. Their main difference is that the second method used ℓ_2 regularization, but the first method used ℓ_1. The third method was LaSVM (Bordes et al., 2005), an online SVM algorithm designed for large datasets.

For each training method, we experimented with several kinds of features, which we call "window," "co-occurrence," and "dependency." Window features included source words and part-of-speech (POS) tags within a 2-word window around the focus word, along with their relative positions (from -2 to +2). Co-occurrence features included all words and POS tags from the whole source sentence, without position information. Dependency features were compiled from the automatically generated English parse trees. The dependency features of each focus word were:

- the label of its maximal projection (i.e. the highest node that has the focus word as its lexical head, which might be a leaf, in which case that label is a POS tag),
- the label and lexical head of the parent of the maximal projection
- the label and lexical head of all dependents of the maximal projection
- all the labels of all head-children (recursively) of the maximal projection

The window features were present in all experimental conditions. The presence/absence of co-occurrence and dependency features yielded 4 "configurations."

Using each of these configurations, each training method produced a confidence-rating binary classifier for each translation of each English word seen in the training data. In all cases, the test procedure was to choose the French word predicted with the highest confidence. All methods, including the baseline, predicted NULL for source words that were not seen in training data.

Table 4.2 shows the size and accuracy of all three methods on the development set, after training on 10,000 sentence pairs, for each of the four configurations. The best configurations of the two logistic regression methods far exceed the baseline, but otherwise they were statistically

	W	W+D	W+C	W+D+C
10,000 training sentences — baseline = 47.54				
ℓ_2	54.09	54.33	52.36	52.88
ℓ_1	53.96	54.13	53.29	53.75
LaSVM	53.38	51.93	49.13	50.71
pruned ℓ_2	47.37	46.01	46.68	45.01
ℓ_1 size	54.1K	41.7K	37.8K	38.7K
ℓ_2 size	1.67M	2.51M	5.63M	6.47M
pruned ℓ_2 size	54.1K	41.7K	37.8K	38.7K
100,000 training sentences — baseline = 51.94				
ℓ_1	62.00	62.42	61.98	62.40
ℓ_1 size	736K	703K	316K	322K

Table 4.2: Percent accuracy on the development set and sizes of word-to-word classifiers trained on 10K or 100K sentence pairs. The feature sets used were (W)indow , (D)ependency, and (C)o-occurrence. ℓ_1 size is the number of compound feature types.

10K w/ ℓ_1 reg'n, dependency	54.64
100K w/ ℓ_1 reg'n, dependency	62.88

Table 4.3: Percent accuracy of word-to-word classifiers on the test set.

indistinguishable. The accuracy of LaSVM was similar to the regression methods when using only the window features, but it was significantly worse with the larger feature sets.

More interesting were the differences in model sizes. The ℓ_2-regularized models were bigger than the ℓ_1-regularized models by two orders of magnitude. The ℓ_2-regularized models grew in size to accommodate each new feature type. In contrast, the ℓ_1-regularized models *decreased* in size when given more useful features, without significantly losing accuracy. This trend was even stronger on the larger training set, where more of the features were more reliable.

The size of models produced by LaSVM grew linearly with the number of examples, because for source words like "the," about 90% of the examples became support vectors. This behavior makes it infeasible to scale up LaSVM to significantly larger data sets, because it would need to compare each new example to all support vectors, resulting in near-quadratic run-time complexity.

To scale up to 100,000 sentence pairs of training data with only the window features, classifiers could not fit in the memory of our computers. To make them fit, we could set all but the heaviest feature weights to zero. We tried this on 10,000 sentence pairs of training data. The number of

features allowed to remain active in each ℓ_2 classifier was the number of active features in the ℓ_1 classifier. Table 4.2 shows the accuracy of these "pruned" ℓ_2-regularized classifiers on the development set, when trained on 10,000 sentence pairs. With the playing field leveled, the ℓ_1 classifiers were far more effective.

In preliminary experiments, we also tried perceptron-style updates, as suggested by Tillmann and Zhang (2005). However, for reasons given by Tewari and Bartlett (2005), the high-entropy decisions involved in our structured prediction setting often prevented convergence to useful classifiers. Likewise, C. Tillmann informed us (p.c.) that, to ensure convergence, he had to choose features very carefully even for his finite-state SMT system.

Regularization schemes that don't produce sparse representations seem unsuitable for problems on the scale of machine translation. We would expect similar problems with the perceptron-style training regime proposed by Tillmann and Zhang (2005), unless the number of different features was limited. For this reason, we used only ℓ_1 regularized log-loss for the rest of our experiments. Table 4.2 shows the accuracy and model size of the ℓ_1-regularized classifier on the development set, when trained on 100,000 sentence pairs, using each of the 4 configurations. Our classifier far exceeded the baseline. The test set results for the best models (window + dependency features) were quite close to those on the development set: 54.64% with the smaller training set, and 62.88% with the larger.

4.3.3 Bag Transduction

The word-to-word translation task is a good starting point, but any conclusions that we might draw from it are inherently biased by the algorithm used to map source words to target words in the test data. Our next set of experiments was on a task with more external validity – predict a translation for *each* source word in the test data, regardless of whether GIZA++ predicted an alignment for it. The difficulty with this task, of course, is that we have no deterministic word alignment to use as a gold standard. Our solution was to pool the word translations in each source sentence and compare them to the bag of words in the target sentence. We still predicted exactly one translation per source word, and that translation could be NULL. Thus, the number

of target words predicted for each source sentence was less than or equal to the number of words in that source sentence. The evaluation measures for this experiment were precision, recall, and F-measure, with respect to the bag of words in the test target sentence.

We compared the 4 configurations of our ℓ_1-regularized classifiers on this task to the most-frequent-translation baseline. We also evaluated a mixture model, where a classifier for each source word was chosen from the best one of the 4 configurations, based on that configuration's accuracy on that source word in the tuning data. As an additional gauge of external validity, we performed the same task using the best publicly available machine translation system (Koehn et al., 2003). This comparison was enlightening but necessarily unfair. As mentioned above, our long-term goal is to build a system whose every component is discriminatively trained to optimize the objective function. We did not want to confuse our study with heuristic methods, so we avoided "phrase" induction, word class induction, non-discriminatively trained target language models, etc. On the other hand, modern SMT systems are designed for use with such information sources, and cannot be fairly evaluated without them. So, we ran Pharaoh in two configurations. The first used the default system configuration, with a target language model trained on the target half of the training data. The second allowed Pharaoh to use its phrase tables but without a target language model. This second configuration allowed us to compare the accuracy of our classifiers to Pharaoh specifically on the subtask of SMT for which they were designed.

The results are in Table 4.4. The table shows that our method far exceeds the baseline. Since we predict only one French target word per English source word, the recall of our bag transducer was severely handicapped by the tendency of French sentences to be longer than their English equivalents. This handicap is reflected in the 1-to-1 upper bound shown in the table. With a language model, Pharaoh's recall exceeded that of our best model by slightly less than this 13.7% handicap. However, we were surprised to discover that the bag transducer's precision was significantly higher than Pharaoh's when they compete on a level playing field (without a language model). Table 4.5 shows the accuracy of the best models on the test set, where the numbers closely follow those on the development set.

This result suggests that it might not be necessary to induce "phrases" on the source side[2].

[2]Quirk and Menezes (2006) offer additional evidence for this hypothesis.

	P	R	F
training on 10,000 sentence pairs			
baseline	48.09	41.26	44.42
window only	53.93	42.81	47.73
dependency	53.97	42.72	47.69
co-occurrence	53.31	42.45	47.26
co-oc + dep	53.58	42.67	47.50
mixture model	54.05	42.95	47.87
training on 100,000 sentence pairs			
baseline	51.91	40.79	45.68
window only	58.79	44.26	50.50
dependency	59.12	44.57	50.82
co-occurrence	59.06	44.36	50.67
co-oc + dep	59.19	44.60	50.87
mixture model	59.03	44.55	50.78
Pharaoh w/o LM	32.20	54.62	40.51
Pharaoh with LM	56.20	57.49	56.84
1-to-1 upper bound	100.00	86.31	92.65

Table 4.4: (P)recision, (R)ecall and (F)-measure for bag transduction of the development set. The discriminative transducers were trained with ℓ_1 regularization.

After all, the main benefits of phrases on the source side are in capturing lexical context and local word reordering patterns. Our bag transducers capture lexical context in their feature vectors. Word order is irrelevant for bag transduction (but see the next section). The only advantage of phrases on this task is in proposing more words on the target side, which eliminates the 1-to-1 upper bound on recall.

	P	R	F
training on 10,000 sentence pairs			
dependency	54.36	42.75	47.86
mixture model	54.27	42.81	47.86
training on 100,000 sentence pairs			
co-oc + dep	59.49	44.19	50.71
mixture model	59.62	44.38	50.88
Pharaoh w/o LM	32.55	54.62	40.80
Pharaoh with LM	57.01	57.84	57.45

Table 4.5: (P)recision, (R)ecall and (F)-measure of bag transducers on the test set.

4.3.4 Tree Transduction

We experimented with a simplistic tree transducer, which involves only two types of inference. The first type transduces leaves; the second type transduces internal nodes. The transduction of leaves is exactly the word-to-word translation task described in Section 4.3.2. Leaves that are transduced to NULL are deterministically erased. Internal nodes are transduced merely by permuting the order of their children, where one of the possible permutations it to retain the original order. This transducer is grossly inadequate for modeling real multitext (Galley et al., 2004): It cannot account for many kinds of noise and for many real translingual phenomena, such as head-switching and discontinuous constituents, which are important for accurate SMT. It cannot even capture common phrasal translations such as *there is / il y a*. However, it is sufficient for controlled comparison of learning methods. The learning method will be the same when we use more sophisticated tree transducers. Another advantage of this experimental design is that it uses minimal linguistic cleverness and is likely to apply to many language pairs, in contrast to other studies of constituent/dependent reordering that are more language-specific (Collins et al., 2005; Xia and McCord, 2004).

To reduce data sparseness, each internal node with more than two children was binarized, so that the multiclass permutation classification for the original node was reduced to a sequence of binary classifications. This reduction is different from the usual multiclass reduction to binary: In addition to making the classifier binary instead of multiclass, the reduction decomposes the label so that some parts of it can be predicted before others. For example, without this reduction, a node with children $\langle A, B, C\rangle$ can be transduced to any of 6 possible permutations, requiring a 6-class classifier. After binarization, the same 6 possible permutations can be obtained by first permuting $\langle A, B\rangle$, and then permuting the result with C, or by first permuting $\langle B, C\rangle$ and then permuting the result with A. This reduction eliminates some of the possible permutations for nodes with four or more children, as discussed in Chapter 1.

Our monolingual parser indicated which node is the head-child of each internal node. Some additional permutations were filtered out using this information: Two sibling nodes that were *not* the head-children of their parent were not allowed to participate in a permutation until

one of them was permuted with the head-child sibling. Thus, if C was the head-child in the previous example, then $\langle A, B\rangle$ could not be permuted first; $\langle B, C\rangle$ had to be permuted first, before permuting with A.

To make the tree transducer deterministic, we add a procedure that searches for the tree with minimum total loss, among all possible output trees:

$$\widehat{T} = \arg\min_{T \in \mathcal{T}} \sum_{i \in T} l(i) \tag{4.2}$$

where $\mathcal{T}$ is the set of all possible output trees, $i \in T$ are the inferences used to build the tree T and $l(i)$ is the loss associated with inference i. We compared two models of inference loss — one generative and one discriminative.

The generative model was based on a top-down tree transducer (Comon et al., 1997) that stochastically generates the target tree given the source tree. The generative process starts by generating the target root given the source root. It then proceeds top-down, generating every target node conditioned on its parent and on the corresponding node in the source tree. Let π be the function that maps every node to its parent, and let η be the function that maps every target node to its corresponding source. If we view the target tree as consisting of nodes n with n_0 being the root node, then the probability of the target tree T is

$$\Pr(T) = \Pr(n_0|\eta(n_0)) \cdot \prod_{n \neq n_0 \in T} \Pr(n|\pi(n), \eta(n))$$

For the generative model, the loss of an inference i is the negative logarithm of the probability of the node $n(i)$ that it infers:

$$l(i) = -\log \Pr[n(i)|\pi(n(i)), \eta(n(i))]. \tag{4.3}$$

We estimated the parameters of this transducer using the Viterbi approximation of the inside-outside algorithm described by Graehl and Knight (2004). Following (Zhang and Gildea, 2005), we lexicalized the nodes so that their probabilities capture bilexical dependencies.

source parser	transduction model	exponent 1			exponent 2		
		P	R	F	P	R	F
generative	generative	51.29	38.30	43.85	22.62	16.90	19.35
generative	discriminative	59.89	39.53	47.62	26.94	17.78	21.42
discriminative	generative	50.51	37.76	43.21	22.04	16.47	18.85
discriminative	discriminative	**62.36**	39.06	**48.04**	**28.02**	17.55	**21.59**
	Pharaoh (w/o LM)	32.19	**54.62**	40.51	12.37	**20.99**	15.57

Table 4.6: (P)recision, (R)ecall, and (F)-measure of transducers using 100,000 sentence pairs of training data.

The discriminative model was trained using the method in Section 4.2, with $l(i) = \sigma_\Theta(i)$. A separate classifier was induced for each possible translation of each source word seen in training data, to evaluate candidate transductions of leaf nodes. Additional classifiers were induced to confidence-rate candidate permutations of sibling nodes. Recall that each permutation involved a head-child node and one of its siblings. Since our input trees were lexicalized, it was easy to determine the lexical head of both the head-child and the other node participating in each permutation. Features were then compiled separately for each of these words according to the "window" and "dependency" feature types described in Section 4.3.2. Since the tree was transduced bottom-up, the word-to-word translation of the lexical head of any node was already known by the time it participated in a permutation. So, in addition to dependents on the source side, there were also features to encode their translations. The final kind of feature used to predict permutations was whole synchronous context-free production rules, in bilexical, monolexical, and unlexicalized forms. These kinds of feature combinations are very difficult to model in the traditional generative framework. Our hypothesis was that the discriminative approach would be more accurate, because its evaluation of each inference could take into account a great variety of information in the tree, including its entire yield (string), not just the information in nearby nodes. In principle, our step-wise approach to structured inference is also more flexible than the approach of Taskar et al. (2004), because we are not limited to objective functions that can be decomposed along subsets of the relevant features.

For both models, the search for the optimal tree was organized by an agenda, as is typically done for tree inference algorithms. For efficiency, we used a chart, and pruned items whose score

was less than 10^{-3} times the score of the best item in the same chart cell. We also pruned items from cells whenever the number of items in the same cell exceeded 40. Our entire tree transduction algorithm can be viewed as translation by parsing, discussed in Chapter 3 where the source side of the output bi-tree was constrained by the input (source) tree.

We compared the generative and discriminative models by reading out the string encoded in their predicted trees, and comparing that string to the target sentence in the test corpus. In pilot experiments we used the BLEU measure commonly used for such comparisons (Papineni et al., 2002). To our surprise, BLEU reported unbelievably high accuracy for our discriminative transducer, exceeding the accuracy of Pharaoh even with a language model. Subsequently, we discovered that BLEU was incorrectly inflating our scores by internally re-tokenizing our French output. This behavior, together with the growing evidence against using BLEU for syntax-aware SMT (Callison-Burch et al., 2006), convinced us to use the more transparent precision, recall, and F-measure, as computed by GTM (Turian et al., 2003). With the exponent set to 1.0, the F-measure is the unigram overlap ratio, except it avoids double-counting. With a higher exponent, the F-measure accounts for overlap of all n-grams (i.e. for all values of n), again without double-counting.

During testing, we compared two kinds of input parse trees for each kind of tree transducer. The first kind was generated by the parser of Bikel (2004). The second kind was generated by the parser of Turian and Melamed (2006), which was trained in a purely discriminative manner using the method of Section 4.2. Table 4.6 shows the results. The discriminatively-trained transducer far outperformed the generatively trained transducer on both precision and recall. In addition, the discriminatively-trained transducer performed better when it started with parse trees from a purely discriminative parser. To our knowledge, these are the first reported results for a syntax-driven SMT system that makes no use of generative models.

4.4 Conclusions

We have presented a method for training a syntax-aware statistical machine translation system in a fully discriminative manner. This work is the first to do SMT by using discriminative

models to predict target tree-structure node by node, extracting features from each node in the multitree. The system outperforms a generative baseline, despite not using the standard trick of bootstrapping from a generative model. We have not yet added all the standard information sources that are necessary for a state-of-the-art MT system, but the scalability of our system suggests that we have overcome the main obstacle for doing so.

5
Conclusion

We have presented motivation for, the theory behind, and experiments from a statistical machine translation system that uses a single abstract parsing algorithm and tree structured models to do hierarchical alignment, training and translation.

Our motivating study found surprisingly many examples of translational equivalence that could not be analyzed using binary-branching structures without discontinuities. Allowing a single gap in bilingual phrases or other types of constituents can dramatically reduce the number of unalignable sentences in a multitext. We presented evidence of phenomena that can lead to complex patterns of translational equivalence in multitexts of any language pair, and presented the first published examples from real multitext of sentences that are not hierarchically alignable using any Inversion Transduction Grammar.

We then presented a new grammar formalism, Generalized Multitext Grammar, which is an intuitive sufficiently expressive model of parallel text. We formally defined GMTG and presented some interesting properties of the grammar, including a new normal form called GCNF. This allowed us to build the first freely available toolkit that does SMT-by-parsing, called "GenPar". We presented implementation and design details about GenPar here, and illustrated the research benefits of an object oriented design. The toolkit is the first software able to do parsing with grammars of fan-out greater than 2, and to calculate the fan-out needed to hierarchically align multitexts.

Lastly, we presented a method for training our new syntax-aware statistical machine translation system in a fully discriminative manner. The system outperformed a generative baseline, despite not using the standard trick of bootstrapping from a generative model. It is the first fully discriminative end-to-end tree-structured SMT system, made possible by recent advances in machine learning, an object oriented software design, and a new machine learning approach to SMT which breaks down tree transductions into a series of classification decisions.

References

A. Aho and J. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*.

A. Aho and J. Ullman. 1972. *The Theory of Parsing, Translation and Compiling, Vol. 1.* Prentice-Hall, Englewood Cliffs, N.J.

Necip Fazil Ayan, Bonnie J. Dorr, and Christof Monz. 2005. Alignment link projection using transformation-based learning. In *Proceedings of the conference on HLT and EMNLP*, Vancouver, BC. Association for Computational Linguistics.

Regina Barzilay and Kathleen R. McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, Toulouse.

T. Becker, A. Joshi, and O. Rambow. 1991. Long-distance scrambling and tree adjoining grammars. In *Proceedings of the 5th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, Berlin, Germany.

Dan Bikel. 2004. A distributional analysis of a lexicalized statistical parsing model. In *Proceedings of the 9th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Barcelona, Spain.

T.L. Booth and R.A. Thompson. 1973. Applying probability measures to abstract languages. *IEEE Transaction on Computers*.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. 2005. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, Robert L. Mercer, and Paul Roossin. 1988. A statistical approach to language translation. In *Proceedings of the International Conference on Computational Linguistics (COLING) 1988*, Budapest, Hungary.

Andrea Burbank, Marine Carpuat, Stephen Clark, Markus Dreyer, Pamela Fox, Declan Groves, Keith Hall, Mary Hearne, I. Dan Melamed, Yihai Shen, Andy Way, Ben Wellington, and Dekai Wu. 2005. Final report on statistical machine translation by parsing. Technical report, Johns Hopkins University Center for Speech and Language Processing. `http://www.clsp.jhu.edu/ws2005/groups/statistical/report.html`.

Chris Callison-Burch, Colin Bannard, and Josh Scroeder. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, Michigan.

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

David Chiang. 2002. Putting some weakly context-free formalisms in order. In *Proceedings of the Sixth International Workshop on TAG and Related Formalisms (TAG+).*

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, Michigan.

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, Michigan.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. 1997. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata`. released October 1, 2002.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*.

Hal Daumé. 2004. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://pub.hal3.name#daume04cg-bfgs`, implementation available at `http://hal3.name/megam/`.

Bonnie Dorr. 1994. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics 20:4*.

M. Dras and T. Bleam. 2000. How problematic are clitics for S-TAG translation? In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, France.

George Foster. 2000. A maximum entropy / minimum divergence translation model. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, Hong Kong.

Heidi Fox. 2002. Phrasal cohesion and statistical machine translation. In *Proceedings of the 7th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, PA.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Boston, MA.

Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson. 1994. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison Wesley Longman, Inc.

Daniel Gildea. 2003. Loosely tree-based alignment for machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Sapporo, Japan.

Joshua Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, Harvard University.

Jonathan Graehl and Kevin Knight. 2004. Training tree transducers. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Boston, MA.

Declan Groves, Mary Hearne, and Andy Way. 2004. Robust sub-sentential alignment of phrase-structure trees. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING) 2004*, Geneva, Switzerland, August.

J. Hopcroft, R. Motwani, and J. Ullman. 2001. *Introduction to Automota Theory, Languages and Computation*. Addison-Wesley, USA.

Tadao Kasami. 1965. An efficient recognition and syntax algorithm for context-free languages. Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Edmonton, AB.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Edmonton, AB.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of MT Summit X*, Phuket, Thailand. International Association for Machine Translation.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*.

Joel Martin, Rada Mihalcea, and Ted Pedersen. 2005. Word alignments for languages with scarce resources. In *The Association of Computational Linguistics (ACL) Workshop on Building and Using Parallel Text*, Ann Arbor, MI.

I. Dan Melamed and Wei Wang. 2005. Generalized parsers for machine translation. Technical Report 05-001, Proteus Project, New York University. `http://nlp.cs.nyu.edu/pubs/`.

I. Dan Melamed, Giorgio Satta, and Benjamin Wellington. 2004. Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain.

I. Dan Melamed. 1995. Automatic evaluation and uniform filter cascades for inducing N-best translation lexicons. In *Proceedings of the 3rd ACL Workshop on Very Large Corpora (WVLC)*, Cambridge, MA.

I. Dan Melamed. 2003. Multitext grammars and synchronous parsers. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Edmonton, AB.

I. Dan Melamed. 2004a. Algorithms for syntax-aware statistical machine translation. In *Proceedings of the 10th Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, Baltimore, MD.

I. Dan Melamed. 2004b. Statistical machine translation by parsing. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain.

Rada Mihalcea and Ted Pedersen. 2003. An evaluation exercise for word alignment. In *HLT-NAACL 2003 Wordshop: Building and Using Parallel Texts*, Edmonton.

Andrew Y. Ng and Michael Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of the 2002 Neural Information Processing Systems (NIPS)*.

Andrew Y. Ng. 2004. Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. In *ICML*.

NIST. 2002. MT evaluation data (LDC2002E53). `http://projects.ldc.upenn.edu/TIDES/mt2003.html`.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia.

Chris Quirk and Arul Menezes. 2006. Do we need phrases? Challenging the conventional wisdom in statistical machine translation. In *Proceedings of the 11th European Chapter of the Association for Computational Linguistics (EACL)*, Trento, Italy.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal SMT. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, MI.

O. Rambow and G. Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*.

O. Rambow. 1995. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Giorgio Satta and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of the conference on HLT and EMNLP*, Vancouver, BC. Association for Computational Linguistics.

S. Shieber. 1994. Restricting the weak-generative capactiy of synchronous tree-adjoining grammars. *Computational Intelligence*.

Michel Simard, Nicola Cancedda, Bruno Cavestro, Marc Dymetman, Eric Guassier, Cyril Goutte, and Kenji Yamada. 2005. Translating with non-contiguous phrases. In *Proceedings of the conference on HLT and EMNLP*, Vancouver, BC. Association for Computational Linguistics.

B. Taskar, Dan Klein, Michael Collins, Daphne Koller, and Chris Manning. 2004. Max-margin parsing. In *Proceedings of the 9th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Barcelona, Spain.

Ambuj Tewari and Peter L. Bartlett. 2005. On the consistency of multiclass classification methods. In *Proceedings of the 18th Conference on Computational Learning Theory (COLT)*.

Christoph Tillmann and Tong Zhang. 2005. A localized prediction model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, Michigan.

Joseph Turian and I. Dan Melamed. 2006. Advances in discriminative parsing. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, Sydney, Australia.

Joseph Turian, Luke Shen, and I. Dan Melamed. 2003. Evaluation of machine translation and its evaluation. In *Proceedings of MT Summit IX*, New Orleans, LA.

Joseph Turian, Benjamin Wellington, and I. Dan Melamed. 2006. Scalable discriminative learning for natural language parsing and translation. In *Proceedings of the 2006 Neural Information Processing Systems (NIPS)*, Vancouver, BC.

David Vickrey, Luke Biewald, Marc Teyssier, and Daphne Koller. 2005. Word-sense disambiguation for machine translation. In *Proceedings of the conference on HLT and EMNLP*, Vancouver, BC. Association for Computational Linguistics.

Sonjia Waxmonsky and I. Dan Melamed. 2006. A dynamic data structure for parsing with discontinuous constituents. Technical Report 06-001, Proteus Project, New York University. `http://nlp.cs.nyu.edu/pubs`.

Benjamin Wellington, Joseph Turian, Chris Pike, and I. Dan Melamed. 2006a. Scalable purely-discriminative training for word and tree transducers. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA)*, Boston, MA.

Benjamin Wellington, Sonjia Waxmonsky, and I. Dan Melamed. 2006b. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, Sydney, Australia.

Dekai Wu, Marine Carpuat, and Yihai Shen. 2006. Inversion transduction grammar coverage of arabic-english word alignment for tree-structured statistical machine translation. In *Proceedings of the IEEE/ACL Workshop on Spoken Language Technology*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*.

Dekai Wu. 2000. Alignment. In Robert Dale, Hermann Moisl, and Harold Somers, editors, *Handbook of Natural Language Processing*. Marcel Dekker.

Fei Xia and Michael McCord. 2004. Improving a statistical MT system with automatically learned rewrite patterns. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING) 2004*, Geneva, Switzerland.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, Toulouse.

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*.

Richard Zens and Hermann Ney. 2003. A comparative study on reordering constraints in statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Sapporo, Japan.

Hao Zhang and Daniel Gildea. 2004. Syntax-based alignment: Supervised or unsupervised? In *Proceedings of the 20th International Conference on Computational Linguistics (COLING) 2004*, Geneva, Switzerland.

Hao Zhang and Daniel Gildea. 2005. Stochastic lexicalized inversion transduction grammar for alignment. In *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, Ann Arbor, MI.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the International Conference on Computational Linguistics (COLING) 2006*, Sydney, Australia.

A

GenPar, A Toolkit for Statistical Machine Translation by Parsing

A.1 Classes

A.1.1 Parser

The Parser class is the heart of the toolkit. Every parser is a "mediator" of different parser components (Gamma et al., 1994). A Parser contains pointers to several different objects: A Logic which creates inferences, a Grammar which evaluates the partial parse structures, and an Agenda of items which have not been expanded upon. A Parser also keeps track of the diagnostic info of the parse in ParseStats. The collaboration diagram for this class is shown in Figure A.1. The Parser runs the generic chart parsing algorithm in Figure 3.1, and references each of these components when needed during this algorithm.

Many applications need an object that behaves like a parser, in the sense of taking sentence tuples as input and producing multitrees as output. However, sometimes the correct parses are already available. In that case, the DummyParser can be used to read them in and output them. Both Parser and DummyParser are subclasses of AbstractParser, which is the object type that most applications should pass around. E.g., the ParserBuilder constructs one of the two concrete types of parser and returns an AbstractParser pointer.

A.1.2 Grammar

A Grammar is, essentially, an evaluator of parse structures. It decides what parse structures can be inferred and assigns scores to different allowable structures. While traditional grammars in formal language theory have "productions," grammars in the generalized parser do not necessarily need productions, as long as the grammar has some way of assigning scores[1] to partial parses.

The Grammar class family is shown in Figure A.2. The large number of classes in this class family follows from GenPar's philosophy that everything revolves around a very small set of inference algorithms, which are parameterized by various kinds of constraints. The different grammars represent different ways of expressing and combining constraints. The base class is

[1] These scores are semiring values. This means that the scores can be boolean.

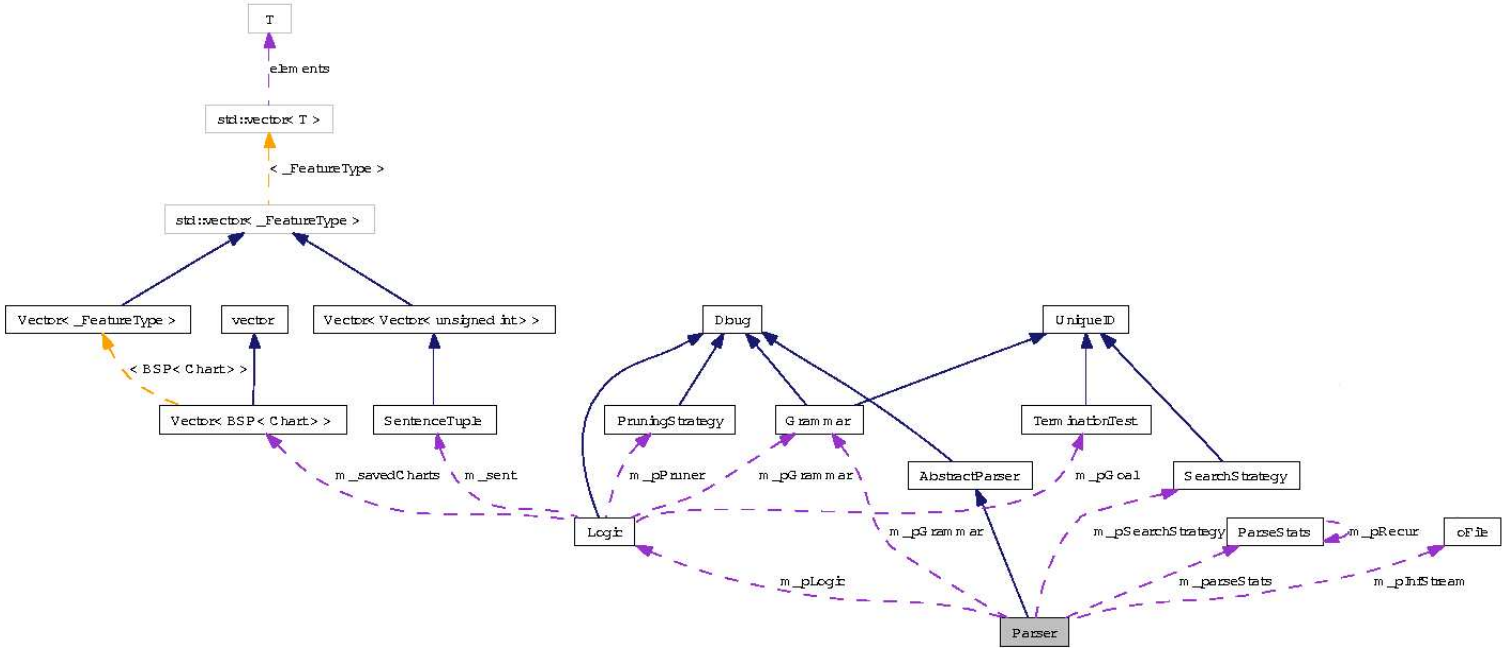

Figure A.1: Parser Collaboration Diagram

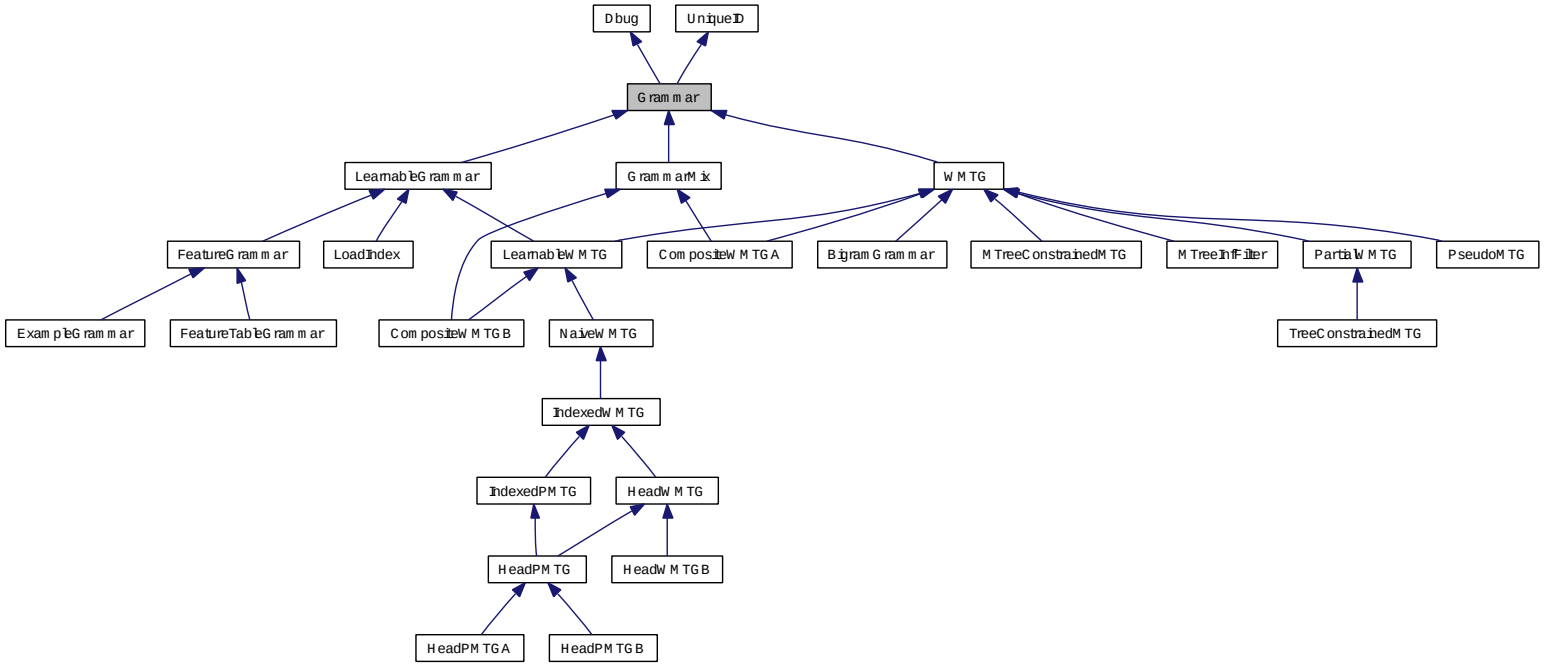

Figure A.2: Grammar class family

called Grammar. Since the base class should be sufficiently general to encompass all imaginable grammars, it contains only some utility methods.

Most of the grammar development to date has focused on (Generalized) Multitext Grammars (Melamed et al., 2004). WMTG is an abstract class for Weighted Multitext Grammars. Its signature includes methods for querying the standard components of an MTG with weights, as well as for conversion into a Generalized Chomsky Normal Form (GCNF). It also supports query methods that are necessary for bottom-up parsing.

The longest chain of WMTG subclasses includes grammars that use traditional production rules and/or their finer-grained decompositions. NaiveWMTG is a concrete class that includes a Histogram of Productions. Such a simple structure cannot be queried efficiently, so the IndexedWMTG subclass adds a production rule index. The IndexedPMTG class adds a normalization method to make the grammar probabilistic.

Statistics over production rules are too crude to generalize well to unseen data. To improve generalization, we have designed generative processes that decompose the generation of each production rule into a series of smaller events. The generative processes are stochastic, and they can assign a probability to any parse structure. We therefore call them Probabilistic Multitext Grammars (PMTGs). All the currently implemented PMTGs include information about head-children, so the class that serves as the abstract interface to such grammars is called HeadPMTG. It presents just a couple of methods for manipulating the events of an arbitrary generative process. The specific PMTGs that are currently implemented are HeadPMTGA and HeadPMTGB. They differ only in whether the precedence arrays in each component of a Production are generated independently of each other. HeadPMTGB is based on the generative process described in Burbank et al. (2005). The non-probabilistic HeadWMTG classes have the same event distributions as their probabilistic counterparts, but they do no normalization.

A subset of the subclasses of WMTG have no production rules to constrain what antecedent items can compose. At the moment, this subset of grammar classes is used mainly for hierarchical alignment. PseudoMTG is a grammar which consists of two or more sub-grammars, each applying constraints to one of the input components. PseudoMTG combines the constraints imposed on

each component into constraints over all components. It also contains a word-to-word model. The implemented sub-grammars that can be used in PseudoMTG are TreeConstrainedMTG, which encodes a parse tree as a constraint, and PartialMTG, which allow all item compositions, i.e. its set of constraints is empty. PseudoMTGs offer the flexibility to decide what constraints you want to apply in each dimension.[2] This means you can apply syntactic constraints from a monoparser on all, some, or none of the dimensions of the hierarchical aligner. For example, to apply a tree constraint in only one component, you might perform alignment using a PseudoMTG, with a TreeConstrainedMTG in one component and an PartialMTG in the other.

The Grammar base class has three sub-interfaces. One is LearnableGrammar, which has generic methods required for parameter estimation. Another is GrammarMix, an abstract base class for grammar mixtures. It contains only some utility methods for constructing grammar mixtures. Its concrete subclasses combine different grammars using the composite design pattern (Gamma et al., 1994). The third is WMTG. Both of the currently implemented composites also implement the WMTG interface. In both cases, only one of the grammars in the mixture is used to generate possible consequents for inferences. In CompositeWMTGA, both of the composed grammars then assign a score to the inferences, and these scores are summed. This kind of composite can be used to mix in a target language model for translation, by way of a BigramGrammar. CompositeWMTGB ignores the score that the first grammar in the mixture assigns to the inference, and just uses the score assigned by the second grammar. This kind of composite is convenient for mixing in non-generative models — see below.

Several of the GenPar grammars are designed to support non-generative models that evaluate inferences using arbitrary features. Key among these grammars are FeatureGrammar, Examples Grammar and FeatureTableGrammar. FeatureGrammar has methods for decomposing an inference into a bag of features using a FeatureGenerator (see below). The grammar stores weights associated with each feature type. When asked to score an inference, it just sums up

[2] "Dimension" is the same as "component," and refers to one of the texts in a multitext, one of the components of a multiparsing logic, one of the languages generated by a transduction grammar, etc.. In the latter case, we intend the formal language theory sense of "language" — a transduction grammar that generates pairs of strings that are both in the same *natural* language is still considered to range over two *formal* languages. This ambiguity of the word "language" is why we prefer "dimension" or "component." See Melamed and Wang (2005) for details.

the weights of its features. ExampleGrammar takes individual examples and stores them, while FeatureTableGrammar breaks each example down into features and stores their weights.

The FeatureGenerator has methods for generating a wide variety of feature types, which can be turned on and off using a configuration file. New feature types are easy to implement. The features pertaining to an inference can reflect any available information about the input or the state of the parse. In GenPar, all aspects of the input and parse state can be represented by tree-structures. The FeatureGenerator class is templatized on a tree type, so that its feature generation code can be reused and/or customized for the various kinds of trees used in GenPar — see Section A.1.14.

A.1.3 Logic

A Logic can be seen as a factory of items. Each concrete Logic subclass is characterized by a different set of inference types. For example, the inference types involved in traditional bottom-up parsing are Compose and Scan; those involved in Earley parsing[3] are Scan, Predict, and Complete.

Figure A.3 shows the Logic class hierarchy. BottomUpTranslationLogic corresponds to LogicCT in (Melamed and Wang, 2005), which uses Scan, Load and Compose inferences. It is a subclass of BottomUpLogic, presented in that paper as LogicC. FasterBottomUpTranslationLogic is a variant that uses a word-to-word translation model to constrain Load inferences. BottomUpExtensionLogic is a Logic which accounts for source and target "extensions" explicitly in the logic itself. The Extension Logics are useful for modeling target sentences that are longer than the source.

Figure A.4 shows the collaboration diagram of Logic. The main member variables in class Logic are a Grammar, a PruningStrategy, a Termination Test, one or more Charts, and the SentenceTuple representing the current input. Charts are in Logic because, formally speaking, Charts are just a bookkeeping device for efficiency, and no other parser component needs to know about them. The PruningStrategy is in Logic because pruning strategies can be viewed

[3] not implemented yet

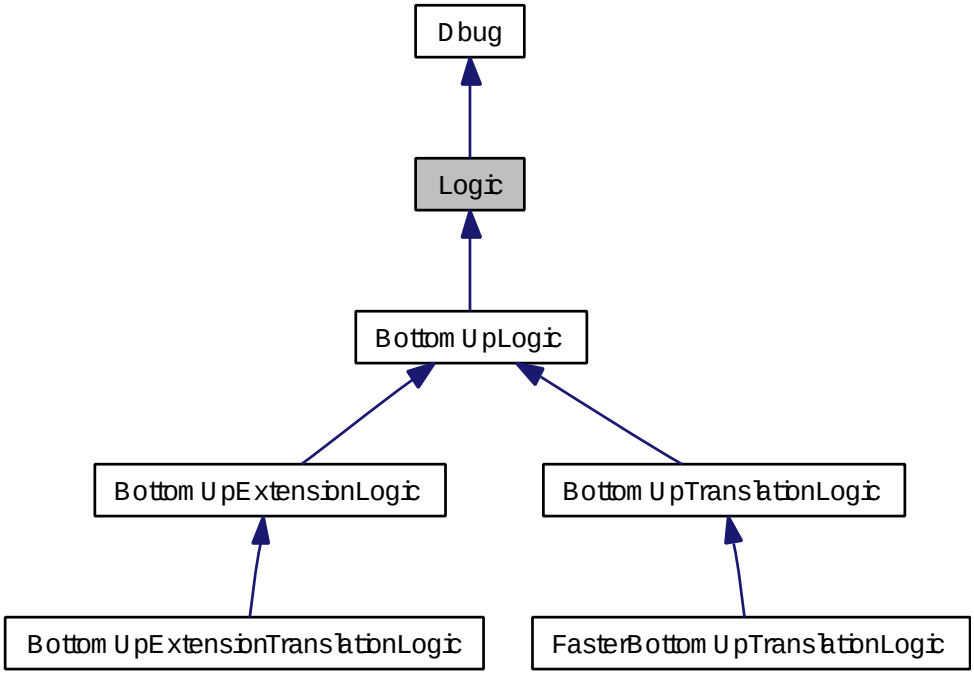

Figure A.3: The Logic class hierarchy.

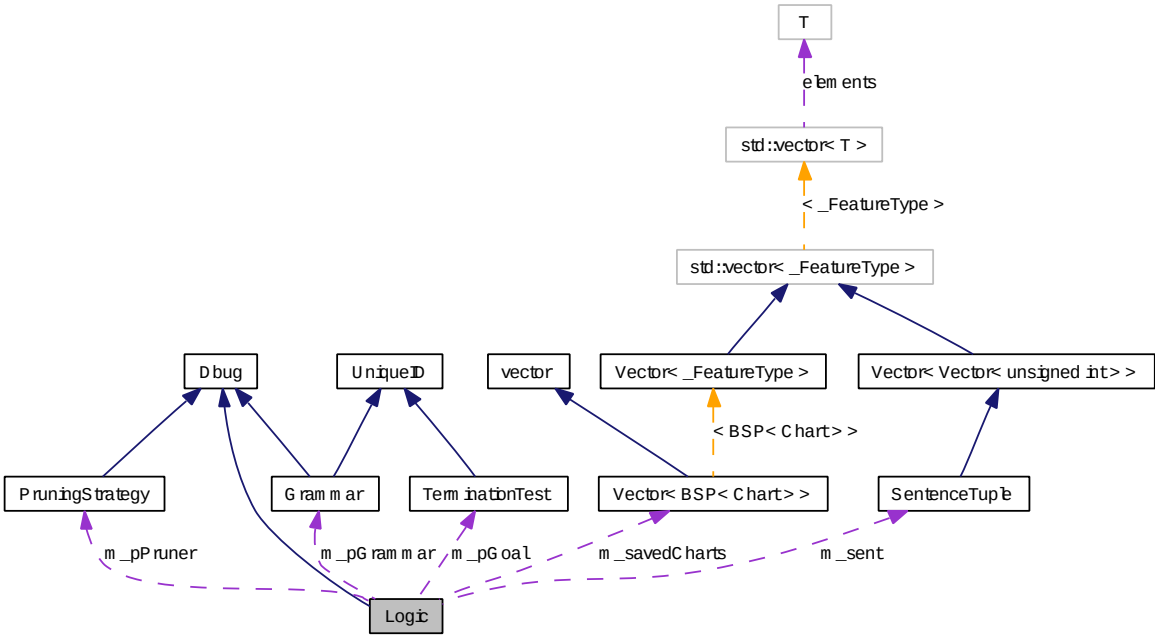

Figure A.4: Logic collaboration diagram.

as side conditions of inferences, and the Logic is responsible for creating new Items by firing Inferences. A TerminationTest is often part of the Parser's termination condition. This part of it is implemented in a class contained in the Logic. The logic also contains a pointer to the Grammar. New logics can be added by providing any necessary new types of items, inferences, grammars, and charts. However, most new logics will be able to reuse many of the classes built for previous logics, as we have done with our translation logics.

Figure A.5 shows a typical implementation of the Logic. Everything above the dotted line are abstract base classes, and everything below the line are concrete classes; the Logic acts as a mediator of all these parts. Note that it is the Logic's responsibility to construct the concrete Chart, OutsideCostEstimator, PruningStrategy and TerminationTest. The Logic constructor takes in parameters indicating which concrete classes to instantiate.

The key method provided by Logic is `expand`(trigger), which generates a batch of new items:

```
expand(Item trigger){
```

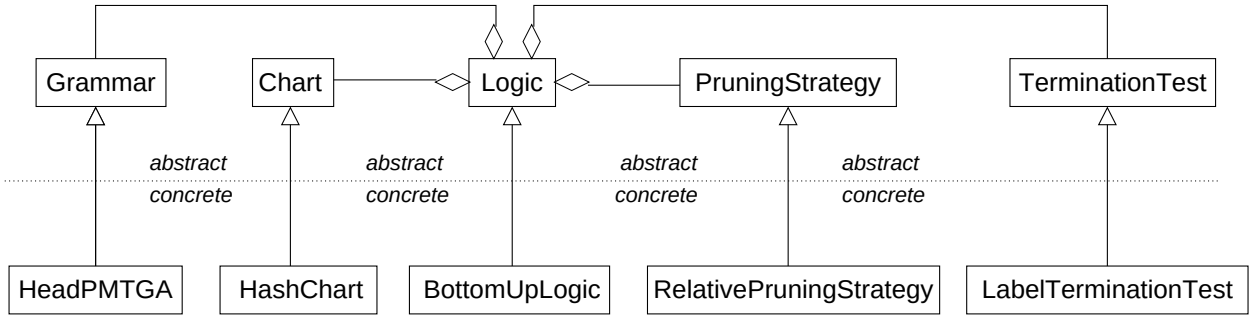

Figure A.5: Abstract components of Logic with some example instantiations.

```
    #Get "sibling" item from trigger
    Item matcher = chart.matchItem(trigger);
    #Get "parent" from trigger and sibling
    set ⟨ Item ⟩ I = grammar.getPossibleCons(trigger,matcher);
    return I;
}
```

In BottomUpLogic's implementation of `expand`, a NULL pointer to the trigger indicates the beginning of parsing. In this case, `expand` will generate the first batch of inferences from the axioms, so that the agenda can be initialized by them. For example, if the logic is BottomUpTranslationLogic, the first iteration creates Scan and Load inferences.

A.1.4 Item

Different logics make use of different types of items. There is currently only one immediate subclass of Item called Hedge. Hedges corresponds to what is called an edge in 1D CKY parsing; the name "hedge" is short for hyper-edge. A Hedge is composed of a nonterminal link, a d-span vector, and heir information. In addition, all items maintain a pointer to the inference that they were the consequent of, which is necessary for reconstructing a parse. A subclass of Hedge,

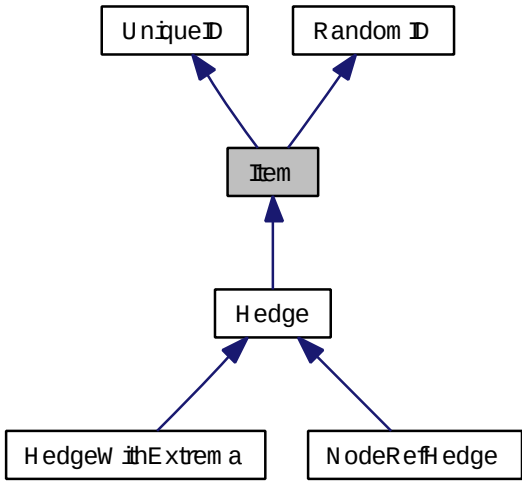

Figure A.6: Item class family

NodeRefHedge, also contains a pointer to a node in a PositionTree, used during hierarchical alignment. Figure A.6 shows this class family.

A.1.5 Chart

The Chart is a container that holds and indexes items that have already been inferred. When a trigger item comes off the agenda, the chart is consulted to see which items satisfy the linear precedence constraints for composition. The trigger is then inserted into the Chart.

The abstract base class Chart is just an interface. It does not even know the type of items that will be stored. It provides an `insert` method which takes an Item to be inserted in the chart, and an `exportParses` method which returns a vector of parse trees with their scores. If a full parse was not found, the chart returns a "maximum cover" which is as few parse trees as possible that cover the entire span.

There are two kinds of concrete Charts, HashChart and GPDiscChart. The former uses hash tables; the latter uses range trees. The range tree chart can be used for parsing with discontinuities (Waxmonsky and Melamed, 2006). The HashChart is limited to finding items that are immediately adjacent to the trigger, but it is asymptotically faster for this limited form of indexing.

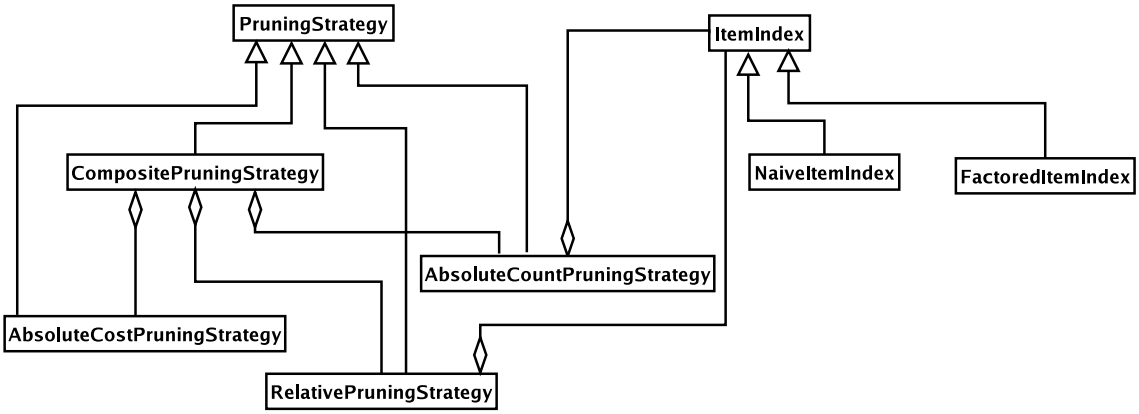

Figure A.7: PruningStrategy class family

A.1.6 Inference

An Inference deduces an item based on items that have already been derived. A Scan or Load has only the axioms as antecedents. Compose inferences are made from two smaller Hedges, in addition to the usual grammar term. HeadCompose inferences also keep track of which antecedent is the heir in each dimension. A CSHeadCompose inference is "Context-Sensitive" and includes features to represent information about the context.

A.1.7 PruningStrategy

GenPar supports several pruning strategies, illustrated in Figure A.7. Constructing the RelativePruningStrategy requires specifying the pruning factor, which decides the size of the pruning beam. The AbsoluteCostPruningStrategy involves two scores: one serves as the threshold used to prune inferences whose consequent f-cost is larger, the other serves as the increment from the previous pruning threshold. This is useful for multi-pass parsing. The AbsoluteCountPruning strategy sets a maximum number of inferences which can exist over a given d-span vector of the input. It is also possible to use a composite of these pruning strategies, with thresholds set for each.

As shown in Figure A.7, the RelativePruningStrategy class contains an ItemIndex. This index

maps from item spans to pairs of items and their sentinel. The sentinel is a variable specified by the PruningStrategy used. RelativePruningStrategy specifies the sentinel to be the minimum cost item in the span, and AbsoluteCountPruningStrategy specifies the sentinel to be the number of items in the span. The PruningStrategy uses this sentinel in deciding whether to prune an Item.

Items located by the same key belong to the same beam, the size of which is decided by the pruning factor stored in RelativePruningStrategy. The AbsoluteCountPruningStrategy uses an index of items like that used in RelativePruningStrategy. It then prunes all but the k lowest cost inferences over a given span. The AbsoluteCostPruningStrategy does not require such an ItemIndex, because it is able to decide whether to prune an item using just the cost of the item and the absolute threshold stored inside AbsoluteCostPruningStrategy. GenPar offers two kinds of ItemIndexes. The NaiveItemIndex has a separate cell for each possible d-span vector. However, this index can use $O(n^4)$ cells. To reduce memory consumption, the FactoredItemIndex implements a separate beam for each d-span, i.e. separately on each dimension.

A.1.8 TerminationTest

The TerminationTest determines when the parsing should terminate. The TerminationTest class family is designed as a flexible utility for this purpose. For example, the parsing could continue until a specific item is derived, or until any item with a given label is derived, or until some set of items is derived. Other tests might be for a span, a cost threshold, a time limit, or a combination of these. Figure A.8 shows the TerminationTest class family. In this family, all tests excluding ConjunctCompositeTerminationTest and DisjunctCompositeTerminationTest are primitive termination tests. The ConjunctCompositeTerminationTest is used to compose primitive or composite tests into a larger test. Only when all the subtests are satisfied will the parsing terminate. DisjunctCompositeTerminationTest also composes subtests, but parsing terminates if any subtest is satisfied. Based on this class family, we can specify a large variety of termination tests.

The TerminationTest class family provides two main methods: `add` and `achieved`. The former

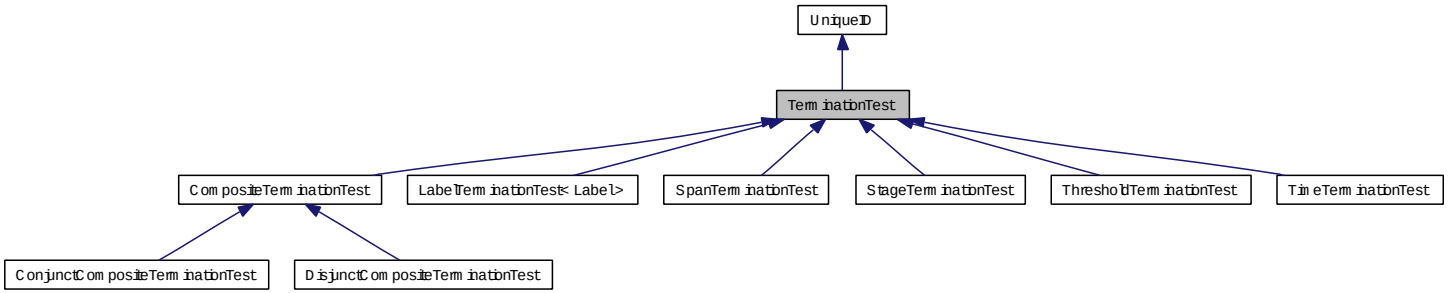

Figure A.8: TerminationTest class family.

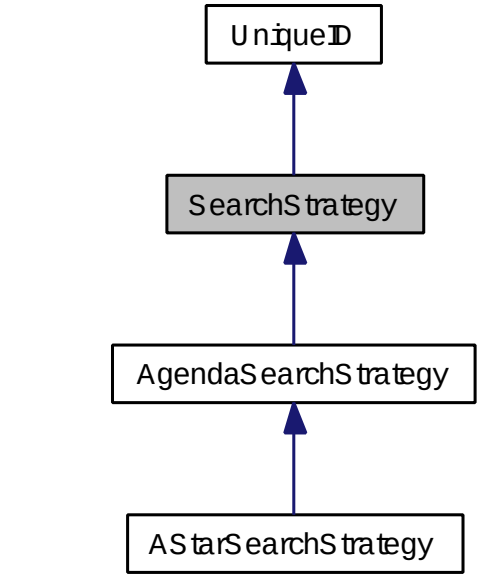

Figure A.9: SearchStrategy class family.

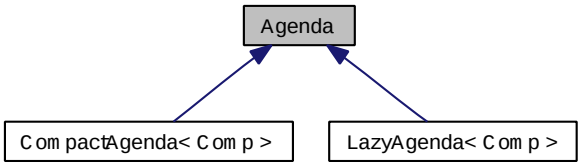

Figure A.10: Agenda class family.

adds a subtest into the current termination test, and returns an ID for the just added test. The latter checks whether the test has been satisfied by the item that has just been derived.

A.1.9 SearchStrategy

The SearchStrategy governs the order in which the Parser processes Items. The base SearchStrategy class is abstract. There are two concrete implementations. AgendaSearchStrategy uses an Agenda and Comparator to schedule items. AStarSearchStrategy uses an OutsideCostEstimator to do best first parsing. This is implemented using the Agenda with a BestFirstComparator.

A BestFirstComparator compares two items based on the scores (i.e., f-cost); a Smallest-

SizeFirstComparator compares two items according to their spans, as in ordinary CKY parsing. Agenda is implemented using a priority queue templatized over a concrete Comparator class.

The difference between the LazyAgenda and the CompactAgenda is that the latter never duplicates items of the same signature. However, this space efficiency comes at the price of some time efficiency – the time necessary to detect and remove duplicates. Both LazyAgenda and CompactAgenda are template classes with Comparator type being their template parameter.

A.1.10 OutsideCostEstimator

This class family realizes different estimators for the outside costs of items proposed by the logic. The base class OutsideCostEstimator specifies the public interface of this class family. We currently are not using OutsideCostEstimators, but somebody might eventually want to, so some of the basic classes exist.

LengthBasedEstimator calculates the estimate by multiplying the lowest cost item in the grammar by the length of the unparsed input sentence. In practice this estimator has no benefit to parsing performance. There is almost always and element in a grammar with a cost of 0 so this estimator is no more effective than not using an estimator.

DimSplitEstimator works on the basis that one dimensional parsing is much faster than multi-parsing. This estimator does monolingual parsing on each dimension in the input sentence. Estimates for each input dimension are combined to form a final estimate.

OptimalEstimator gives a theoretical upper bound on parsing performance. This estimator parses the input sentence using agenda parsing. It then uses the complete parse tree to make estimates. Since this estimator knows all the intermediate costs of the final parse tree this is the optimal estimate.

A.1.11 Terminal and SynCat

Terminals and SynCats are both just numbers. A Terminal represents a "word" in the input language, and every Terminal is associated with its corresponding word in the terminal vocabulary.

SynCats represents "syntactic categories", which are also indexed in a dictionary. Both of these types are implemented using typedefs.

A.1.12 Nonterminals

Nonterminals can be seen as the smallest unit in a single dimension that is being rewritten. In a non-lexicalized grammar, a Nonterminal contains simply a SynCat, but in some grammars, nonterminals have more information than just the syntactic category. For example, the code base uses LexNTs, which are Lexicalized Nonterminals. LexNTs are a subclass of Nonterminal, and so they inherit a syntactic category, but they also have a Terminal member, which represents the lexical head of the Nonterminal.

A.1.13 NTLinks and TLinks

Terminals and Nonterminals are grouped into links called TLinks (terminal links) and NTLinks (nonterminal links). Terminal links are a D-dimensional vector of Terminals, one in each component. NTLinks are a D-dimensional vector of Nonterminal pointers, which allow for polymorphism. Care needs to be taken when using NTLinks since they are vectors of pointers. When first initialized with just a size parameter, the pointers in an NTLink are not yet initialized, since it is unclear which kind of Nonterminal pointer the user wants to put into the NTLink. The NTLink copy constructor and operator= copy the values of the Nonterminals pointed to, not the pointers, using the `clone()` operation on each member nonterminal. `clone()` is a virtual method which, when called on a object, returns a new instance of that object.

A.1.14 Trees

GenPar uses several different classes to represent trees: MTree (multitree), HeadMTree (headed multitree, a subclass of MTree that adds an HeirRoleVector), PositionTree, and HedgeTree. All trees are containers of tree nodes. Depending on the type of tree, each node can contain such features as the Precedence Array Vector (PAV), the HeirVector, and the constituent label assigned to the node. There are quite a few types of features for the parse tree to maintain. To avoid

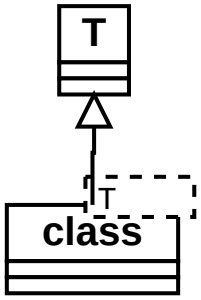

Figure A.11: Static decorator pattern

a potentially huge parse tree class that maintains all the features, we prefer that the parse tree type be a composition of several simpler parse tree types, each manipulating only one type of feature. We use a design technique known as the "static decorator pattern". A static decorator is a template class, and at the same time, derives from the template type (see Figure A.11). It is intended to decorate other objects of the template type by adding more methods to or overriding the methods in the decorated object being inherited. Both the decorating and decorated classes share the same data, and there is no extra data inside the decorator. All methods of both the decorating and decorated classes operate only on the shared data. The decorator is initialized by copying the pointer to the data inside the decorated object. The reason we call it "static" is that the decoration is achieved through template and inheritance, and thus is a "compile time structure". To apply the static pattern to the design of the parse tree, we first design some simple trees, such as a PAVTree that maintains only the PAV information, and an HeirTree that maintains only the heir information. Then a more complex tree can be obtained according to the static pattern by templatizing the PAVTree over the HeirTree, and the HeirTree over the base tree container type. Compared to the standard decorator pattern (Gamma et al., 1994), the static decorator pattern doesn't require the methods of each class (base or subclass) to be exactly the same, saving much code if each class has a large number of methods.

It is worth mentioning how to implement the methods involving several features (e.g., PAV and label) in a simple tree. In a simple tree, we don't know the type of the superclass (it is only a template parameter), nor could we know the types that the tuple is templatized over (they

are also template parameters). But most methods involving multiple features need to know the types of the class and data, and must make sure they are of certain type(s). In this case, we add a **test** to the very beginning of the method implementation. A test consists of some casting statements, which checks the types of the base class and data type at compile time.

A PositionTree is implemented in much the same way as a HeadMTree, but instead of containing PAVs in each node, it contains DSVs. PAVs are directly obtainable from these DSVs, but not vice versa. PositionTrees are used as the constraining tree in TreeConstrainedMTG and as the input tree in FTreeTransducer.

GenPar's third type of tree is a HedgeTree. This is not an instantiatable tree class, but rather just a facade pattern (Gamma et al., 1994) for treating the antecedent/consequent relationship among hedges as a tree. It is used by the FeatureGenerator during translation, to generate features from Items that have already been inferred before they are exported into a HeadMTree.

The file formats used for multitrees are described in the MTV documentation.

A.1.15 Production

The Production class family represents the production rules of MTGs. The Production base class is concrete, and consists of an LHS, which is an NTLink, a PAV, which gives the linear ordering of the children in each component and a RHS, which is a dequeue of RHSElems. Each RHSElem is a boost::variant of NTLinks and TLinks. This means that it can contain either an NTLink or a TLink. We use variants to avoid a common superclass for TLinks and NTLinks, which might cost more memory. The = operator is used on a variant to set it to any of its associated types. Later, when you want to obtain the link, you simply check which type it is first. See boost.org for more information on the syntax of boost::variants. There is a HeadProduction subclass that also stores an HeirRoleVector, which keeps track of which child is the heir in each dimension. The parallel structure between Productions and Compose inferences is no accident. The latter infer the former.

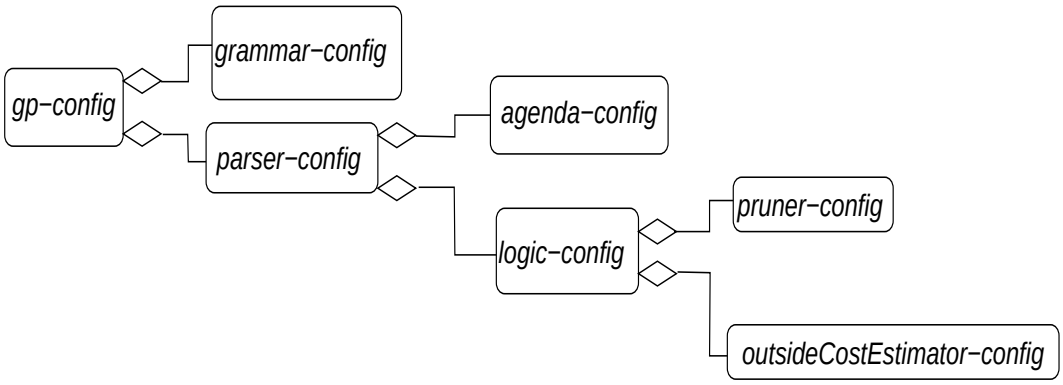

Figure A.12: The config files of the Generalized Parser.

A.2 Data Encapsulation via Nested Configuration Files

We generally define the behavior of class objects via the arguments of the class constructor. However, our design has many classes composed of other classes and so the number of parameters being passed around can get large. To solve this problem, we turn to configuration files to construct class objects instead of using constructor arguments.

Every class with configurable options has a separate configuration file. The class itself defines which option names and types it knows about and reads the option values from the configuration file whose name is typically passed as the only constructor argument. The options reader (from the boost::program_options library, see below) accepts only options that the class specifically defines. This way we ensure data encapsulation: Just as classes themselves are small encapsulated units containing only the data and behavior they need locally, so is every configuration file a single unit containing only the values that the specific class needs.

Figure A.12 shows the config files used by the `gp` program and their relationships. Each node represents a config file, lines connecting nodes represent the containment relation. The relationships between config files are almost isomorphic to that in the overall design of the generalized parser (figure 3.1). The `gp` program takes as parameters the names of the config files of the Parser and the Grammar, as well as the options specifying, for example, the sentence tuple file. The Parser config file not only contains the values of member variables of the Parser class itself,

but also the names of the config files of class Agenda and class Logic. The Logic configuration file then contains the config file names of OutsideCostEstimator and PruningStrategy.

Each config file is of the following format:

```
\# The Config File for Agenda class family.
\# the type of agenda
AgendaType = LAZY
\# The search strategy used by the agenda
SearchStrategy = BESTFIRST
```

Comments start with symbol "#". An option name and its value are separated by symbol " = ".

The option names and types that a class defines are inherited by its subclasses. For example, the class NaiveWMTG defines an option `FanOut`, and so its subclasses HeadPMTG, HeadPMTGA, HeadPMTGB know about and accept such an option in their respective configuration files.

Suppose you implement MySubClass, a subclass of BaseClass. Your class has to take a config file name as a constructor argument and pass it on to the superclass in the member initialization list:

```
MySubClass(const string& configFile) : BaseClass(configFile){
  // ...
}
```

The superclass will read the config file and provide the values through its member functions. If it does not provide the values you need you will have to parse the config file yourself, which means you need a boost::program_options::options_description object that contains the names and types of the options expected in the config file. Fortunately, the superclass defines those options descriptions for you. You just have to construct the options_description object and pass it to the superclass in the initialization list. Many GenPar classes take a smart pointer (boost::shared_ptr) to an options_description as an optional constructor argument. After the superclass is initialized

your options_descriptions object will automatically be filled with the expected options, e.g. the string option `TreebankGrammar`. You are ready to parse the config file:

```
shared_ptr<options_description> optionsDescription;
MySubClass(const string& configFile) : BaseClass(configFile,
                                                 optionsDescription){
  string optTreebankGrammar;
  variables_map vm;
  ifstream ifs(configFile.c_str());
  store(parse_config_file(ifs, *optionsDescription), vm);
  notify(vm);
  if(vm.count("TreebankGrammar"))
    optTreebankGrammar = vm["TreebankGrammar"].as<string>();
  // ...
}
```

This way you can get all the options the user has specified in the config file. But this might still not be enough. Now you want to add your own options description. You want the config file to also contain an integer option `MaxValue`. Remember that the same config file will be read by the superclass and by MySubClass. Therefore, they both have to know that such an option might occur. You have to add `MaxValue` before you construct the superclass:

```
shared_ptr<options_description> optionsDescription;
MySubClass(const string& configFile)
  : BaseClass(configFile,
              addOptionsDescriptionsTo(optionsDescription)){
  int optMaxValue;
  variables_map vm;
  ifstream ifs(configFile.c_str());
  store(parse_config_file(ifs, *optionsDescription), vm);
```

```
  notify(vm);
  if(vm.count("MaxValue"))
    optMaxValue = vm["MaxValue"].as<int>();
  // ...
}
shared_ptr<options_description>
MySubClass::addOptionDescriptionsTo(shared_ptr<options_description> optionsDesc){
  optionsDesc->add_options()
    ("MaxValue", value<int>()->default_value("99999"), "Defines the max value");
  return optionsDesc;
}
```

That is all you have to know if you want to write your subclass with its own options handling in GenPar. If you want others to be able to again subclass your class you must make its constructor take a smart pointer to an options_description as optional argument:

```
  MySubClass(const string& configFile,
     shared_ptr<options_description> priorOptions =
       shared_ptr<options_description>(new options_description()));
```

This will make sure that others can also pass in an optional options_description when they construct your class as their base class, the same way we did it with BaseClass in the above example. See NaiveWMTG.cpp and IndexedWMTG.cpp for examples of config file handling in our code base. For more information on the boost::program_options library see `http://www.boost.org/doc/html/program_options.html`.

A.2.1 "Builder" classes

The "builder" classes (Gamma et al., 1994), also known as "virtual constructors," are responsible for instantiating concrete classes based on the config files that they receive. These builder classes include the AgendaBuilder, GrammarBuilder, LogicBuilder, OutsideCostEstimatorBuilder and

PrunerBuilder. For example, the GrammarBuilder is given a config file that contains what type of concrete Grammar needs to be instantiated by the Parser, for example a HeadPMTGA. The GrammarBuilder parses the file, and instantiates this concrete Grammar and then passes it back to the Parser. The Parser then maintains a pointer to this Grammar and whenever it is accessed, it is in fact the concrete class being accessed. This means that the Parser need not know what kind of Grammar it is pointing to. Another benefit is that the Grammar construction code, which is quite involved, need not be implemented more than once. All of the builder classes follow the singleton design pattern (Gamma et al., 1994).

A.3 Machine Learning in GenPar

GenPar can do machine learning for itself, or let an external package do it and just use the results. Internally, GenPar implements two learning methods using grammar gradients: EM (Viterbi approximation) and the tree perceptron. GenPar also has hooks for interfacing with independent software that can estimate models over arbitrary (non-independent) features. One such software package is the Boosted Decision Trees (BDT) package by Joseph Turian and Dan Melamed, which is included in GenPar's BundledSoftware/ area.

A.3.1 Tree Perceptron

The tree perceptron algorithm assumes a pre-existing gold-standard multitreebank (even if it's produced automatically by hierarchical alignment). It alternates between two stages. In the prediction stage, it takes one side of the training data as input and uses its latest model to predict a multitree that has the training sentence pair in its leaves. In the update stage, it compares its predicted multitree with the gold-standard, computes a gradient from the former to the latter, and updates its model using this gradient. Optionally, after the update, it can replace the gold-standard with its latest prediction. These stages alternate until some convergence criterion is reached.

We call it a tree perceptron algorithm because the model updates are based on differences between trees (i.e. multitrees). What model parameters get updated depends on the model

(i.e. grammar), which is configurable. For example, if the model is a NaiveWMTG, then the parameters are the weights associated with production rules. If the model is more fine-grained, then weights can be associated with smaller steps of a generative process, or with any other kind of feature, as along as there is a way to compute these parameters from a given multitree. A variety of models can be estimated in this manner, as long as they assign scores to multitrees, and can be updated incrementally.

The tree perceptron's prediction step is implemented by an ordinary invocation of the Parser in translation mode, i.e. where the grammar/model is synchronous, but there is only one input sentence. The implementation of the update step uses mainly two classes: UniformPerceptronUpdateRule, which computes differences between multitrees, and the GradientMStep, which updates a model based on these differences. The UniformPerceptronUpdateRule is so named because the magnitudes of the positive and negative feature gradients are balanced to sum to zero (Crammer and Singer, 2003). There are also several kinds of termination testers, in the ConvergenceTester class family. By default, the tree perceptron does not replace the gold-standard tree with the latest predicted tree. This option is mainly for the EM variant.

A.3.2 EM

The Viterbi approximation to EM for probabilistic synchronous grammars can be viewed as a special case of the tree perceptron algorithm described above. If we predict the empty tree on every iteration, then we arrive at the usual Viterbi EStep. If we then use the option to replace the correct tree with the latest prediction on every iteration, we arrive at the usual MStep (modulo normalization). The EStep and MStep classes implement these choices.

A.3.3 Interface to Boosted Decision Trees (BDT)

The Boosted Decision Trees (BDT) software package estimates log-linear models over incrementally extracted compound features. BDT can also use its own models to perform multiclass classification. Wellington et al. (2006a) describe how these capabilities can be used to perform machine translation. BDT does not care what kind of features are involved, as long as they are

binary and have unique names, like "the word to the left is *today*". GenPar interfaces with BDT in two ways. First, GenPar's `trees2grammar` program can generate the features and classes that BDT uses to induce its models from training data. Second, GenPar's FeatureWMTG instantiates a BDT classifier, generates features for it during inference, and asks the classifier to evaluate candidate inferences using its models over those features. The interface to BDT is quite simple, so it should be possible to substitute a different piece of modeling software quite easily.

A.4 Examples of implemented applications

We describe the three main programs that have been implemented using the GenPar classes. There are many others in the `GenPar/tools/` directory of the toolkit.

A.4.1 The program `gp`

`gp` acts as a front end to the Parser class. The front end's main job is to handle configuration and I/O. The parser has numerous configurable parameters, some of which are described in the GenPar User Guide. `gp` is used in several places in the typical SMT pipeline, including modules for **hierarchical alignment**, ordinary **multiparsing**, and **translation**. See Melamed and Wang (2005) for a description of how all of these algorithms are special cases of a single abstract parsing algorithm (which is what the Parser implements).

A.4.2 The model initializer: `trees2grammar`

Many of GenPar's grammars can be initialized from a multitreebank using the `trees2grammar` program. After instantiating one of the grammar classes, the program streams multitrees from the treebank, and accumulates the relevant features of each tree into the grammar's containers. These features can be production rules, more fine-grained events for a more detailed generative process, or arbitrary features of the tree for log-linear models.

A.4.3 The learning front-end: `treeLearn`

The `treeLearn` program acts as a front end to the tree perceptron and EM algorithms described in Section A.3. The front end's main jobs are to handle configuration and I/O, as well as to manage iterations and convergence criteria.