Practical Issues in Compiling Typed Unification Grammars for Speech Recognition

John Dowding Beth Ann Hockey
RIACS RIALIST Group
NASA Ames Research Center
Moffett Field, CA 94035
jdowding@riacs.edu
bahockey@riacs.edu

Jean Mark Gawron
Dept. of Linguistics
San Diego State University
San Diego, CA
gawron@mail.sdsu.edu

Christopher Culy
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
culy@ai.sri.com

Abstract

Current alternatives for language modeling are statistical techniques based on large amounts of training data, and hand-crafted context-free or finite-state grammars that are difficult to build and maintain. One way to address the problems of the grammar-based approach is to compile recognition grammars from grammars written in a more expressive formalism. While theoretically straight-forward, the compilation process can exceed memory and time bounds, and might not always result in accurate and efficient speech recognition. We will describe and evaluate two approaches to this compilation problem. We will also describe and evaluate additional techniques to reduce the structural ambiguity of the language model.

1 Introduction

Language models to constrain speech recognition are a crucial component of interactive spoken language systems. The more varied the language that must be recognized, the more critical good language modeling becomes. Research in language modeling has heavily favored statistical approaches (Cohen 1995, Ward 1995, Hu et al. 1996, Iyer and Ostendorf 1997, Bellegarda 1999, Stolcke and Shriberg 1996) while hand-coded finite-state or context-free language models dominate the commercial sector (Nuance 2001, SpeechWorks 2001, TellMe 2001, BeVocal 2001, HeyAnita 2001, W3C 2001). The difference revolves around the availability of data. Research systems can achieve impressive performance using statistical language models trained on large amounts of domain-targeted data, but for many domains sufficient data is not available. Data may be unavailable because the domain has not been explored before, the relevant data may be confidential, or the system may be designed to do new functions for which there is no human-human analog interaction. The statistical approach is unworkable in such cases for both the commercial developers and for some research systems (Moore et al. 1997, Rayner et al. 2000, Lemon et al. 2001, Gauthron and Colineau 1999). Even in cases for which there is no impediment to collecting data, the expense and time required to collect a corpus can be prohibitive. The existence of the ATIS database (Dahl et al. 1994) is no doubt a factor in the popularity of the travel domain among the research community for exactly this reason.

A major problem with grammar-based finite-state or context-free language models is that they can be tedious to build and difficult to maintain, as they can become quite large very quickly as the scope of the grammar increases. One way to address this problem is to write the grammar in a more expressive formalism and generate an approximation of this grammar in the format needed by the recognizer. This approach has been used in several systems, CommandTalk (Moore et al. 1997), RIALIST PSA simulator (Rayner et al. 2000), WITAS (Lemon et al.

2001), and SETHIVoice (Gauthron and Colineau 1999). While theoretically straight-forward, this approach is more demanding in practice, as each of the compilation stages contains the potential for a combinatorial explosion that will exceed memory and time bounds. There is also no guarantee that the resulting language model will lead to accurate and efficient speech recognition. We will be interested in this paper in *sound approximations* (Pereira and Wright 1991) in which the language accepted by the approximation is a superset of language accepted by the original grammar. While we conceed that alternative techniques that are not sound (Black 1989, (Johnson 1998, Rayner and Carter 1996) may still be useful for many purposes, we prefer sound approximations because there is no chance that the correct hypothesis will be eliminated. Thus, further processing techniques (for instance, N-best search) will still have an opportunity to find the optimal solution.

We will describe and evaluate two compilation approaches to approximating a typed unification grammar with a context-free grammar. We will also describe and evaluate additional techniques to reduce the size and structural ambiguity of the language model.

2 Typed Unification Grammars

Typed Unification Grammars (TUG), like HPSG (Pollard and Sag 1994) and Gemini (Dowding et al. 1993) are a more expressive formalism in which to write formal grammars[1]. As opposed to atomic nonterminal symbols in a CFG, each nonterminal in a TUG is a complex feature structure (Shieber 1986) where features with values can be attached. For example, the rule:

s[] → np:[num=N] vp:[num=N]

can be considered a shorthand for 2 context free rules (assuming just two values for number):

s → np_singular vp_singular
s → np_plural vp_plural

[1]This paper specifically concerns grammars written in the Gemini formalism. However, the basic issues involved in compiling typed unification grammars to context-free grammars remain the same across formalisms.

This expressiveness allows us to write grammars with a small number of rules (from dozens to a few hundred) that correspond to grammars with large numbers of CF rules. Note that the approximation need not incorporate all of the features from the original grammar in order to provide a sound approximation. In particular, in order to derive a finite CF grammar, we will need to consider only those features that have a finite number of possible values, or at least consider only finitely many of the possible values for infinitely valued features. We can use the technique of *restriction* (Shieber 1985) to remove these features from our feature structures. Removing these features may give us a more permissive language model, but it will still be a sound approximation.

The experimental results reported in this paper are based on a grammar under development at RIACS for a spoken dialogue interface to a semi-autonomous robot, the Personal Satellite Assistant (PSA). We consider this grammar to be medium-sized, with 61 grammar rules and 424 lexical entries. While this may sound small, if the grammar were expanded by instantiating variables in all legal permutations, it would contain over 6.8^{25} context-free rules.

3 The Compilation Process

We will be studying the compilation process to convert typed unification grammars expressed in Gemini notation into language models for use with the Nuance speech recognizer (Nuance, 2001). We are using Nuance in part because it supports context-free language models, which is not yet industry standard.[2] Figure 1 illustrates the stages of processing: a typed unification grammar is first compiled to a context-free grammar. This is in turn converted into a grammar in Nuance's *Grammar Specification Language* (GSL), which is a form of context-free grammar in a BNF-like notation, with one rule defining each nonterminal, and allowing alternation and Kleene closure on the right-hand-side. Critically, the GSL must not contain any left-recursion, which must be eliminated before the GSL representation is produced.

[2]The standard is moving in the direction of context-free language models, as can be seen in the draft standard for Speech Recognition Grammars being developed by the World Wide Web Consortium (W3C 2001).

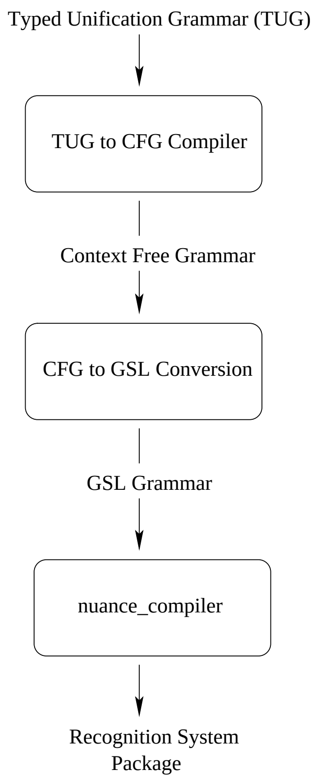

Figure 1: Compilation Process

```
1  $T_0 := \emptyset$;
2  for each $l \in \mathcal{L}$
3    $l := L(l)$;
4    $T_0 := T_0 \cup_{\sqsubseteq} \{l\}$;
5  $T := Iterate(\mathcal{R}, R, T_0)$;

6  $Iterate(\mathcal{R}, R, T_i)$
7    local $T_{i+1}$;
8    $T_{i+1} := T_i$;
9    for each $= r \in \mathcal{R}$
10     for each $t \in FillDaughters(r, T_i)$
11     $t := R(t)$;
12     $T_{i+1} := T_{i+1} \cup_{\sqsubseteq} \{t\}$;
13   if $T_i = T_{i+1}$
14     then return $T_i$
15     else return $Iterate(\mathcal{R}, R, T_{i+1})$
```

Table 1: Construction of the fixed-point

The GSL representation is then compiled into a Nuance package with the `nuance_compiler`. This package is the input to the speech recognizer. In our experience, each of the compilation stages, as well as speech recognition itself, has the potential to lead to a combinatorial explosion that exceeds practical memory or time bounds.

We will now describe implementations of the first stage, generating a context-free grammar from a typed unification grammar, by two different algorithms, one defined by Kiefer and Krieger (2000) and one by Moore and Gawron, described in Moore (1998) The critical difficulty for both of these approaches is how to select the set of derived nonterminals that will appear in the final CFG.

3.1 Kiefer&Krieger's Algorithm

The algorithm of Kiefer&Krieger (K&K) divides this compilation step into two phases: first, the set of context-free nonterminals is determined by iterating a bottom-up search until a least fixed-point is reached; second, this least fixed-point is used to instantiate the set of context-free productions.

The computation of the fixed-point T, described in Table 1, proceeds as follows. First, T_0 is constructed by finding the most-general set of feature structures that occur in the lexicon $\mathcal{L}$ (lines 1-4). Each feature structure has the lexical restrictor L applied to it before being added to T_0 (line 3) with the $\cup_{\sqsubseteq}$ operator. This operator maintains the set T_0 of most-general feature structures. A new feature structure is added to the set only when it is not subsumed by any current members of the set, and any current members that are subsumed by the new member are removed as the new element is added. The computation of T proceeds with the call to *Iterate* (line 6), which adds new feature structures that can be derived bottom-up. Each call to *Iterate* generates a new set T_{i+1}, including T_i as its base (line 8). It then adds new feature structures to T_{i+1} by instantiating every grammar rule r in $\mathcal{R}$, the set of grammar rules. The first step in the instantiation is to unify every combination of daughters with all possible feature structures from T_i (*FillDaughters*, line 10). The rule restrictor is applied to each resulting feature structure (line 11) before it is added to T_{i+1} using the $\cup_{\sqsubseteq}$ operator (line 12), similar to the lexical case. If after checking all rule applications bottom up, no new feature structures have been added to T_{i+1} (line 13), then the least fixed-point had been found, and the process terminates. Otherwise, *It-*

erate is called recursively. See Kiefer and Krieger (2000) for proof that this terminates, and finds the appropriate fixed-point.

Having computed the least fixed-point T, the next step is to compute the set of corresponding CF productions. For each *r* in $\mathcal{R}$, of the form $t \rightarrow t_1 \ldots t_n$, instantiate the daughters $t_1 \ldots t_n$ using all combinations of unifiable feature structures from T. Context-free productions $\hat{t} \rightarrow t_1 \ldots t_n$ will be added, where $\hat{t} \in T$ and $\hat{t}$ $subsumes$ t.[3]

3.2 Moore and Gawron's Algorithm

While K&K uses subsumption to generate the set of most-general nonterminals, the algorithm of Moore and Gawron (M&G), described in Moore (1998) attempts to propagate features values both bottom-up and top-down through the grammar to generate a set of nonterminals that contains no variables. Also unlike K&K, the production of the CF rules and associated nonterminals is interleaved. The process consists of a preprocessing stage to eliminate singleton variables, a bottom-up propagation stage, and a top-down propagation stage.

The preprocessing stage rewrites the grammar to eliminate singleton variables. This step effective replaces singleton variables with a new unique atomic symbol 'ANY'. The feature structure for each lexical item and grammar rule is rewritten such that singleton variables are unified with a special value 'ANY', and every non-singleton variable expression is embedded in a val() term. After this transformation, singleton variables will not unify with non-singleton variable expressions, only with other singletons. Additional rules are then introduced to deal with the singleton variable cases. For each daughter in a grammar rule in which a singleton variable appears, new lexical items and grammar rules are introduced which unify with that daughter in the original grammar. As an example, consider the grammar fragment:

vp:[num=N] $\rightarrow$ v:[num=N] np:[]
np:[num=N] $\rightarrow$ det:[num=N] n:[num=N]
np:[num=pl] $\rightarrow$ n:[num=pl]

Here, the np object of vp is underspecified for num (as English does not generally require number agreement between the verb and its object), so it will be a singleton variable. So, the following rules will be generated:

vp:[num=val(N)] $\rightarrow$
    v:[num=val(N)] np:[num='ANY']
np:[num=val(N)] $\rightarrow$
    det:[num=val(N)] n:[num=val(N)]
np:[num=val(pl)] $\rightarrow$ n:[num=val(pl)]
np:[num='ANY'] $\rightarrow$
    det:[num=val(N)] n:[num=val(N)]
np:[num='ANY'] $\rightarrow$ n:[num=val(pl)]

After preprocessing, any variables remaining in the bodies of grammar rules will be shared variables. Singleton variable elimination by itself is very effective at shrinking the size of the CF grammar space, reducing the size of the rule space for the PSA grammar from $6.8x10^{25}$ rules to $4.2x10^{16}$ rules.

The bottom-up stage starts from this grammar, and derives a new grammar by propagating feature values up from the lexicon. The process acts like a chart parser, except that indicies are not kept. When a rule transitions from an active edge to an inactive edge, a new rule with those feature instantiations is recorded. As a side-effect of this compilation, ϵ-productions are eliminated.

Top-down processing fires last, and performs a recursive-descent walk of the grammar starting atthe start symbol Σ, generating a new grammar that propagates features downward through the grammar. A side-effect of this computation is that useless-productions (rules not reachable from Σ) are removed. It might still be possible that after top-down propagation there would still be variables present in the grammar. For example, if the grammar allows sentences like "the deer walked", which are ambiguous for number, then there will be a rule in the grammar that contains a shared variable for the number feature. To address this, as top-down propagation is progressing, all remaining variables are identified and unified with

[3]There is a minor bug in K&K where they state that the result t will always be in T and $t \rightarrow t_1 \ldots t_n$ will be a CF production in the approximation, but this may not be true if t was removed from T by $\cup_{\sqsubseteq}$. Instead, the subsuming nonterminal $\hat{t}$ should be the new mother.

a special value ’ALL’. Since each nonterminal is now ground, it is trivial to assign each nonterminal a unique atomic symbol, and rewrite the grammar as a CFG.

3.3 Comparison

Table 2 contains a summary of some key statistics generated using both techniques. The recognition results were obtained on a test set of 250 utterances. Recognition accuracy is measured in word error rate, and recognition speed is measured in multiples of real time (RT), the length of the utterance compared with the length of the CPU time required for the recognition result[4]. The size of the resulting language model is measured in terms of the number of nonterminals in the grammar, and the size of the Nuance node array, a binary representation of the recursive transition network it uses to search the grammar. Ambiguity counts the average number of parses per sentence that were allowed by the CF grammar. As can be readily seen, the compilation time for the K&K algorithm is dramatically lower than the M&G algorithm, while producing a similarly lower recognition performance, measured in both word error rate and recognition speed.

Given that the two techniques generate grammars of roughly similar sizes, the difference in performance is striking. We believe that the use of the $\cup_{\sqsubseteq}$ in K&K is partially responsible. Consider a grammar that contains a lexical item like “deer” that is underspecified for number, and will contain a singleton variable. This will lead to a nonterminal feature structure for noun phrase that is also underspecified for number, which will be more general than any noun phrase feature structures that are marked for number. The $\cup_{\sqsubseteq}$ operator will remove those noun phrases as being less general, effectively removing the number agreement constraint between subject and verb from the context-free approximation. The use of $\cup_{\sqsubseteq}$ allows a single grammar rule or lexical item to have non-local effects on the approximation. As seen in Table 2, the grammar derived from the K&K algorithm is much more ambiguous than the grammar derived the M&G algorithm, and, as is further elaborated

[4]All timing results presented in this paper were executed on a Sun Ultra 60 workstation, running at 330MHz with 1.5 GB physical memory and an additional 1GB swap.

	K&K	M&G
Compilation Time	11 min.	192 min.
Nonterminals	2,284	1,837
Node Array Size	224KB	204KB
Word Error Rate	25.05%	11.91%
Recognition Time	13.8xRT	1.7xRT
Ambiguity	15.4	1.9

Table 2: Comparison Results

in Section 4, we believe that the amount of ambiguity can be a significant factor in recognition performance.

On the other hand, attention must be paid to the amount of time and memory required by the Moore algorithm. On a medium-sized grammar, this compilation step took over 3 hours, and was close to exceeding the memory capacity of our computer, with a process size of over 1GB. The approximation is only valuable if we can succeed in computing it. Finally, it should also be noted that M&G’s algorithm removes ϵ-productions and useless-productions, while we had to add a separate postprocessing stage to K&K’s algorithm to get comparable results.

For future work we plan to explore possible integrations of these two algorithms. One possibility is to include the singleton-elimination process as an early stage in the K&K algorithm. This is a relatively fast step, but may lead to a significant increase in the size of the grammar. Another possibility is to embed a variant of the K&K algorithm, and its clean separation of generating nonterminals from generating CF productions, in place of the bottom-up processing stage in M&G’s algorithm.

4 Reducing Structural Ambiguity

It has been observed (Bratt and Stolcke 1999) that a potential difficulty with using linguistically-motivated grammars as language models is that ambiguity in the grammar will lead to multiple paths in the language model for the same recognition hypothesis. In a standard beam-search architecture, depending on the level of ambiguity, this may tend to fill the beam with multiple hypotheses for the same word sequence, and force other good hypotheses out of the beam, potentially increasing word error rate. This observation appears to be supported in practice. The original form of the PSA grammar allows an average of 1.4 parses per sentence, and while both the K&K and M&G algorigthm increase the level of ambiguity, the K&K algorithm increases much more dramatically.

We are investigating techniques to transform a CFG into one weakly equivalent but with less ambiguity. While it is not possible in general to remove all ambiguity (Hopcroft and Ullman 1979) we hope that reducing the amount of ambiguity in the resulting grammar will result in improved recognition performance.

4.1 Grammar Compactor

The first technique is actually a combination of three related transformations:

- *Duplicate Nonterminal Elimination* – If two nonterminals A and B have exactly the same set of productions

  $$A \rightarrow \beta_1 | \ldots | \beta_n$$
  $$B \rightarrow \beta_1 | \ldots | \beta_n$$

  then remove the productions for B, and rewrite B as A everywhere it occurs in the grammar.

- *Unit Rule Elimination* – If there is only one production for a nonterminal A, and it has a single daughter on its right-hand side

  $$A \rightarrow \beta, |\beta| = 1$$

  then remove the production for A, and rewrite A as β everywhere it occurs in the grammar.

- *Duplicate Production Elimination* – If a nonterminal A has two productions that are identical

  $$A \rightarrow \beta_1 | \ldots | \beta_n, \beta_i = \beta_j, i \neq j$$

  then remove the production for β_i.

These transformations are applied repeatedly until they can no longer be applied. Each of these transformations may introduce opportunities for the others to apply, so the process needs to be order insensitive. This technique can be applied after the traditional reduction techniques of ϵ-elimination, cycle-elimination, and left-recursion elimination, since they don't introduce any new ϵ-productions or any new left-recursion.

Although these transformations seem rather specialized, they were surprisingly effective at reducing the size of the grammar. For the K&K algorithm, the number of grammar rules was reduced from 3,246 to 2,893, a reduction of 9.2%, and for the M&G algorithm, the number of rules was reduced from 4,758 to 1,837, a reduction of 61%. While these transforms do reduce the size of the grammar, and modestly reduce the level of ambiguity from 1.96 to 1.92, they did not initially appear to improve recognition performance. However, that was with the nuance parameter `-node_array_optimization_level` set to the default value `FULL`. When set to the value `MIN`, the compacted grammar was approximately 60% faster, and about 9% reduction in the word error rate, suggesting that the `nuance_compiler` is performing a similar form of compaction during node array optimization.

4.2 Immediate Recursion Detection

Another technique to reduce ambiguity was motivated by a desire to reduce the amount of prepositional phrase attachment ambiguity in our grammar. This technique detects when a Kleene closure will be introduced into the final form of the grammar, and takes advantage of this to remove ambiguity. Consider this grammar fragment:

NP → NP PP
VP → V NP PP

The first rule tells us that an NP can be followed by an arbitrary number of PPs, and that the PP following the NP in the second rule will be ambiguous. In addition, any nonterminal that has an NP as its rightmost daughter can also be followed by an arbitrary number of PPs, so we can detect ambiguity following those nonterminals as well. We define a predicate *follows* as:
A *follows* B iff
B → B A or
B → β C and A *follows* C

Now, the *follows* relation can be used to reduce ambiguity by modifying other productions where a B is followed by an A:

$X \rightarrow \beta_1 \ldots \beta_i \beta_{i+1} \ldots \beta_n$
where β_{i+1} *follows* β_i and $\beta_1 \neq X$
can be rewritten as
$A \rightarrow \beta_1 \ldots \beta_i \beta_{i+2} \ldots \beta_n$

There is an exactly analogous transformation involving immediate right-recursion and a similar predicate *preceeds.* These transformation produce almost the same language, but can modify it by possibly allowing constructions that were not allowed in the original grammar. In our case, the initial grammar fragment above would require that at least one PP be generated within the scope of the VP, but after the transformation that is no longer required. So, while these transformations are not exact, they are still sound aproximations, as the resulting language is a superset of the original language.

Unfortunately, we have had mixed results with applying these transformations. In earlier versions of our implementation, applying these transformations succeeded in improving the recognition speed up to 20%, while having some modest improvements in word error rate. But, as we improved other aspects of the compilation process, notably the grammar compaction techniques and the left-recursion elimination technique, those improvements disappeared, and the transformations actually made things worse. The problem appears to be that both transformations can introduce cycles, and the right-recursive case can introduce left-recursion even in cases where cycles are not introduced. When the introduced cycles and left-recursions are later removed, the size of the grammar is increased, which can lead to poorer recognition performance. In the earlier implementations, cycles were fortuitously avoided, probably due to the fact that there were more unique nonterminals overall. We expect that these transformations may be effective for some grammars, but not others. We plan to continue to explore refinements to these techiques to prevent them from applying in cases where cycles or left-recursion may be introduced.

5 Left Recursion Elimination

We have used two left-recursion elimination techniques, the traditional one based on Paull's algorithm, as reported by Hopcroft and Ullman (1979), and one described by Moore (2000)[5], based on a technique described by Johnson (1998). Our experience concurs with Moore that the left-corner transform he describes produces a more compact left-recursion free grammar than that of Paull's algorithm. For the K&K approximation, we were unable to get any grammar to compile through to a working language model using Paull's algorithm (the models built with Paull's algorithm caused the recognizer to exceed memory bounds), and only succeeded with Moore's left-recursion elimination technique.

6 Conclusions

We have presented descriptions of two algorithms for approximating typed unification grammars with context-free grammars, and evaluated their performance during speech recognition. Initial results show that high levels of ambiguity coorelate with poor recognition performance, and that size of the resuling language model does not appear to directly coorelate with recognition performance. We have developed new techniques for further reducing the size and amount of ambiguity in these context-free grammars, but have so far met with mixed results.

References

J. Bellegarda. Context scope selection in multi-span statistical language modeling. In *Proceedings of the 6th European Conference on Speech Communication and Technology (EuroSpeech99)*, pages 2163–2166, 1999.

BeVocal. http://www.bevocal.com, 2001. As of 31 January 2001.

A. Black. Finite state machines from feature grammars. In *International Workshop on Parsing Technologies*, pages 277–285, 1989.

H. Bratt and A. Stolcke. private communication, 1999.

[5]There is a minor bug in the description of Moore's algorithm that occurs in his paper, that the set of "retained nonterminals" needs to be extended to include any nonterminals that occur either in the non-initial daughter of a left-recursive nonterminal, or in any daughter of a non-left-recursive nonterminal. Thanks to Robert Moore for providing the solution to this bug. This bug applies only to the description of his algorithm, not to the implementation on which the empirical results reported is based. Please see Moore (2000) for more details.

M. Cohen, Z. Rivlin, and H. Bratt. Speech recognition in the ATIS domain using multiple knowledge sources. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 257–260, 1995.

D. Dahl, M. Bates, M. Brown, K. Hunicke-Smith, D. Pallet, C. Pao, A. Rudnicky, and E. Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the ARPA Human Language Technology Workshop*, Princeton, NJ, March 1994.

J. Dowding, M. Gawron, D. Appelt, L. Cherny, R. Moore, and D. Moran. Gemini: A natural language system for spoken language understanding. In *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics*, 1993.

O. Gauthron and N. Colineau. SETHIVoice: Cgf control by speech-recognition/interpretation. In *I/ITSEC '99 (Interservice/Industry Training, Simulation and Education Conference), Synthetic Solutions for the 21st Century*, Orlando, FL, 1999.

HeyAnita. http://www.heyanita.com, 2001. As of 31 January 2001.

J. Hu, W. Turin, and M.K. Brown. Language modeling with stochastic automata. In *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP)*, pages 406–413, 1996.

J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

R. Iyer and M. Ostendorf. Transforming out-of-domain estimates to improve in-domain language models. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EuroSpeech97)*, pages 1975–1978, 1997.

M. Johnson. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*, pages 619–623, 1998.

B. Kiefer and H. Krieger. A context-free approximation of head-driven phrase structure grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies*, pages 135–146, 2000.

Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. A multi-modal dialogue system for human-robot conversation. In *Proceedings of North American Association for Computational Linguistics (NAACL 2001)*, 2001.

R. Moore, J. Dowding, H. Bratt, J. Gawron, Y. Gorfu, and A. Cheyer. CommandTalk: A spoken-language interface for battlefield simulations. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 1–7, 1997.

R. Moore. Using natural language knowledge sources in speech recognition. In *Proceedings of the NATO Advanced Studies Institute*, 1998.

R. Moore. Removing left-recusion from context-free grammars. In *Proceedings of 1st Meeting of the North Americal Chapter of the Associations for Computational Linguistics*, pages 249–255, 2000.

Nuance. http://www.nuance.com, 1999. As of 1 April 1999.

C. Pollard and I. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

F. Pereira and R. Wright. Finite-state approximation of phrase structure grammars. In *Proceedings of the 29th Annual Meeting of the Assocations for Computational Linguistics*, pages 246–255, 1991.

M. Rayner and D.M. Carter. Fast parsing using pruning and grammar specialization. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 223–230, Santa Cruz, California, 1996.

M. Rayner, B.A. Hockey, and F. James. A compact architecture for dialogue management based on scripts and meta-outputs. In *Proceedings of Applied Natural Language Processing (ANLP)*, 2000.

S. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Assocations for Computational Linguistics*, pages 145–152, 1985.

S. Shieber. *An Introduction to Unification-based Approaches to Grammar*. CLSI Lecture Notes no. 4. Center for the Study of Language and Information, 1986. (distributed by the University of Chicago Press).

SpeechWorks. http://www.speechworks.com, 2001. As of 31 January 2001.

A. Stolcke and E. Shriberg. Statistical language modeling for speech disfluencies. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 405–408, 1996.

TellMe. http://www.tellme.com, 2001. As of 31 January 2001.

World Wide Web Consortium (W3C). *Speech Recognition Grammar Specification for the W3C Speech Interface Framework*. http://www.w3.org/TR/speech-grammar, 2001. As of 3 January 2001.

W. Ward and S. Issar. The cmu atis system. In *Spoken Language System Technology Workshop*, pages 249–251, 1995.