Speech Recognition Grammar Compilation in Grammatical Framework

Björn Bringert
Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
bringert@cs.chalmers.se

Abstract

This paper describes how grammar-based language models for speech recognition systems can be generated from Grammatical Framework (GF) grammars. Context-free grammars and finite-state models can be generated in several formats: GSL, SRGS, JSGF, and HTK SLF. In addition, semantic interpretation code can be embedded in the generated context-free grammars. This enables rapid development of portable, multilingual and easily modifiable speech recognition applications.

1 Introduction

Speech recognition grammars are used for guiding speech recognizers in many applications. However, there are a number of problems associated with writing grammars in the low-level, system-specific formats required by speech recognizers. This work addresses these problems by generating speech recognition grammars and semantic interpretation components from grammars written in Grammatical Framework (GF), a high-level, type-theoretical grammar formalism. Compared to existing work on compiling unification grammars, such as Regulus (Rayner et al., 2006), our work uses a type-theoretical grammar formalism with a focus on multilinguality and modular grammar development, and supports multiple speech recognition grammar formalisms, including finite-state models.

We first outline some existing problems in the development and maintenance of speech recognition grammars, and describe how our work attempts to address these problems. In the following two sections we introduce speech recognition grammars and Grammatical Framework. The bulk of the paper then describes how we generate context-free speech recognition grammars, finite-state language models and semantic interpretation code from GF grammars. We conclude by giving references to a number of experimental dialogue systems which already use our grammar compiler for generating speech recognition grammars.

Expressivity Speech recognition grammars are written in simple formalisms which do not have the powerful constructs of high-level grammar formalisms. This makes speech recognition grammar writing labor-intensive and error prone, especially for languages with more inflection and agreement than English.

This is solved by using a high-level grammar formalism with powerful constructs and a grammar library which implements the domain-independent linguistic details.

Duplicated work When speech recognition grammars are written directly in the low-level format required by the speech recognizer, other parts of the system, such as semantic interpretation components, must often be constructed separately.

This duplicated work can be avoided by generating all the components from a single declarative source, such as a GF grammar.

Consistency Because of the lack of abstraction mechanisms and consistency checks, it is difficult

Proceedings of SPEECHGRAM 2007, pages 1–8,

to modify a system which uses hand-written speech recognition grammars. The problem is multiplied when the system is multilingual. The developer has to modify the speech recognition grammar and the semantic interpretation component manually for each language. A simple change may require touching many parts of the grammar, and there are no automatic consistency checks.

The strong typing of the GF language enforces consistency between the semantics and the concrete representation in each language.

Localization With hand-written grammars, it is about as difficult to add support for a new language as it is to write the grammar and semantic interpretation for the first language.

GF's support for multilingual grammars and the common interface implemented by all grammars in the GF resource grammar library makes it easier to translate a grammar to a new language.

Portability A grammar in any given speech recognition grammar format cannot be used with a speech recognizer which uses another format.

In our approach, a GF grammar is used as the canonical representation which the developer works with, and speech recognition grammars in many formats can be generated automatically from this representation.

2 Speech Recognition Grammars

To achieve acceptable accuracy, speech recognition software is guided by a *language model* which defines the language which can be recognized. A language model may also assign different probabilities to different strings in the language. A language model can either be a *statistical language model (SLM)*, such as an *n*-gram model, or a *grammar-based language model*, for example a *context-free grammar (CFG)* or a *finite-state automaton (FSA)*. In this paper, we use the term *speech recognition grammar (SRG)* to refer to all grammar-based language models, including context-free grammars, regular grammars and finite-state automata.

3 Grammatical Framework

Grammatical Framework (GF) (Ranta, 2004) is a grammar formalism based on constructive type theory. In GF, an *abstract syntax* defines a semantic representation. A *concrete syntax* declares how terms in an abstract syntax are *linearized*, that is, how they are mapped to concrete representations. GF grammars can be made multilingual by having multiple concrete syntaxes for a single abstract syntax.

3.1 The Resource Grammar Library

The GF Resource Grammar Library (Ranta et al., 2006) currently implements the morphological and syntactic details of 10 languages. This library is intended to make it possible to write grammars without caring about the linguistic details of particular languages. It is inspired by *library-based software engineering*, where complex functionality is implemented in reusable software libraries with simple interfaces.

The resource grammar library is used through GF's facility for *grammar composition*, where the abstract syntax of one grammar is used in the implementation of the concrete syntax of another grammar. Thus, an application grammar writer who uses a resource grammar uses its abstract syntax terms to implement the linearizations in the application grammar.

The resource grammars for the different languages implement a common interface, i.e. they all have a common abstract syntax. This means that grammars which are implemented using resource grammars can be easily localized to other languages. Localization normally consists of translating the application-specific lexical items, and adjusting any linearizations which turn out to be unidiomatic in the language in question. For example, when the GoTGoDiS (Ericsson et al., 2006) application was localized to Finnish, only 3 out of 180 linearization rules had to be changed.

3.2 An Example GF Grammar

Figure 1 contains a small example GF abstract syntax. Figure 2 defines an English concrete syntax for it, using the resource grammar library. We will use this grammar when we show examples of speech recognition grammar generation later.

In the abstract syntax, **cat** judgements introduce syntactic categories, and **fun** judgements declare constructors in those categories. For example, the

```
abstract Food = {
  cat Order; Items; Item; Number; Size;
  fun order : Items → Order;
      and   : Items → Items → Items;
      items : Item → Number → Size → Items;
      pizza, beer  : Item;
      one, two     : Number;
      small, large : Size;
}
```

Figure 1: `Food.gf`: A GF abstract syntax module.

```
concrete FoodEng of Food = open English in {
  flags startcat = Order;
  lincat Order  = Utt; Items  = NP;
         Item   = CN; Number = Det;
         Size   = AP;
  lin order x     = mkUtt x;
      and x y     = mkNP and_Conj x y;
      items x n s = mkNP n (mkCN s x);
      pizza       = mkCN (regN "pizza");
      beer        = mkCN (regN "beer");
      one         = mkDet one_Quant;
      two         = mkDet n2;
      small       = mkAP (regA "small");
      large       = mkAP (regA "large");
}
```

Figure 2: `FoodEng.gf`: English concrete syntax for the abstract syntax in Figure 1.

items constructor makes an Items term from an Item, a Number and a Size. The term *items pizza two small* is an example of a term in this abstract syntax.

In the concrete syntax, a **lincat** judgement declares the type of the concrete terms generated from the abstract syntax terms in a given category. The linearization of each constructor is declared with a **lin** judgement. In the concrete syntax in Figure 2, library functions from the English resource grammar are used for the linearizations, but it is also possible to write concrete syntax terms directly. The linearization of the term *items pizza two small* is $\{s =$ "two small pizzas"$\}$, a record containing a single string field.

By changing the imports and the four lexical items, this grammar can be translated to any other language for which there is a resource grammar. For example, in the German version, we replace (*regN* "beer") with (*reg2N* "Bier" "Biere" *neuter*) and so on. The functions *regN* and *reg2N* implement paradigms for regular English and German nouns, respectively. This replacement can be formalized using GF's *parameterized modules*, which lets one write a common implementation that can be instantiated with the language-specific parts. Note that the application grammar does not deal with details such as agreement, as this is taken care of by the resource grammar.

4 Generating Context-free Grammars

4.1 Algorithm

GF grammars are converted to context-free speech recognition grammars in a number of steps. An overview of the compilation pipeline is show in Figure 3. The figure also includes compilation to finite-state automata, as described in Section 5. Each step of the compilation is described in more detail in the sections below.

Conversion to CFG The GF grammar is first converted into a context-free grammar annotated with functions and profiles, as described by Ljunglöf (2004).

Cycle elimination All directly and indirectly cyclic productions are removed, since they cannot be handled gracefully by the subsequent left-recursion elimination. Such productions do not contribute to the coverage to the grammar, only to the set of possible semantic results.

Bottom-up filtering Productions whose right-hand sides use categories for which there are no productions are removed, since these will never match any input.

Top-down filtering Only productions for categories which can be reached from the start category are kept. This is mainly used to remove parts of the grammar which are unused because of the choice of start category. One example where this is useful is when a speech recognition grammar is generated from a multimodal grammar (Bringert et al., 2005). In this case, the start category is different from the start category used by the parser, in that its linearization only contains the speech component of the in-

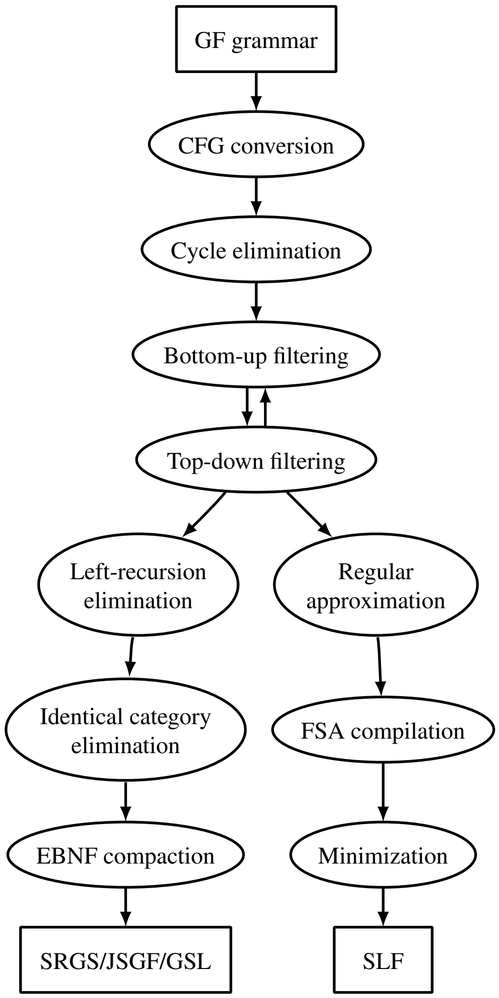

Figure 3: Grammar compilation pipeline.

put. Top-down filtering then has the effect of excluding the non-speech modalities from the speech recognition grammar.

The bottom-up and top-down filtering steps are iterated until a fixed point is reached, since both these steps may produce new filtering opportunities.

Left-recursion elimination All direct and indirect left-recursion is removed using the LC_{LR} transform described by Moore (2000). We have modified the LC_{LR} transform to avoid adding productions which use a category $A{-}X$ when there are no productions for $A{-}X$.

Identical category elimination In this step, the categories are grouped into equivalence classes by their right-hand sides and semantic annotations. The categories $A_1 \ldots A_n$ in each class are replaced by a single category $A_1{+}\ldots{+}A_n$ throughout the grammar, discarding any duplicate productions. This has the effect of replacing all categories which have identical sets of productions with a single category. Concrete syntax parameters which do not affect inflection is one source of such redundancy; the LC_{LR} transform is another.

EBNF compaction The resulting context-free grammar is compacted into an *Extended Backus-Naur Form (EBNF)* representation. This reduces the size and improves the readability of the final grammar. The compaction is done by, for each category, grouping all the productions which have the same semantic interpretation, and the same sequence of non-terminals on their right-hand sides, ignoring any terminals. The productions in each group are merged into one EBNF production, where the terminal sequences between the non-terminals are converted to regular expressions which are the unions of the original terminal sequences. These regular expressions are then minimized.

Conversion to output format The resulting non-left-recursive grammar is converted to SRGS, JSGF or Nuance GSL format.

A fragment of a SRGS ABNF grammar generated from the GF grammar in Figure 2 is shown below. The left-recursive *and* rule was removed from the grammar before compilation, as the left-recursion elimination step makes it difficult to read the generated grammar. The fragment shown here is for the singular part of the *items* rule.

```
$FE1 = $FE6 $FE9 $FE4;
$FE6 = one;
$FE9 = large | small;
$FE4 = beer  | pizza;
```

The corresponding fragment generated from the German version of the grammar is more complex, since the numeral and the adjective must agree with the gender of the noun.

```
$FG1  = $FG10 $FG13 $FG6 | $FG9 $FG12 $FG4;
$FG9  = eine;     $FG10 = ein;
$FG12 = große  | kleine;
$FG13 = großes | kleines;
$FG4  = Pizza;    $FG6 = Bier;
```

4.2 Discussion

The generated grammar is an overgenerating approximation of the original GF grammar. This is inevitable, since the GF formalism is stronger than context-free grammars, for example through its support for reduplication. GF's support for dependently typed and higher-order abstract syntax is also not yet carried over to the generated speech recognition grammars. This could be handled in a subsequent semantic interpretation step. However, that requires that the speech recognizer considers multiple hypotheses, since some may be discarded by the semantic interpretation. Currently, if the abstract syntax types are only dependent on finite types, the grammar can be expanded to remove the dependencies. This appears to be sufficient for many realistic applications.

In some cases, empty productions in the generated grammar could cause problems for the cycle and left-recursion elimination, though we have yet to encounter this in practice. Empty productions can be removed by transforming the grammar, though this has not yet been implemented.

For some grammars, the initial CFG generation can generate a very large number of productions. While the resulting speech recognition grammars are of a reasonable size, the large intermediate grammars can cause memory problems. Further optimization is needed to address this problem.

5 Finite-State Models

5.1 Algorithm

Some speech recognition systems use finite-state automata rather than context-free grammars as language models. GF grammars can be compiled to finite-state automata using the procedure shown in Figure 3. The initial part of the compilation to a finite-state model is shared with the context-free SRG compilation, and is described in Section 4.

Regular approximation The context-free grammar is approximated with a regular grammar, using the algorithm described by Mohri and Nederhof (2001).

Compilation to finite-state automata The regular grammar is transformed into a set of *non-deterministic finite automata (NFA)* using a modified version of the *make_fa* algorithm described by Nederhof (2000). For realistic grammars, applying the original *make_fa* algorithm to the whole grammar generates a very large automaton, since a copy of the sub-automaton corresponding to a given category is made for every use of the category.

Instead, one automaton is generated for each category in the regular grammar. All categories which are not in the same mutually recursive set as the category for which the automaton is generated are treated as terminal symbols. This results in a set of automata with edges labeled with either terminal symbols or the names of other automata.

If desired, the set of automata can be converted into a single automaton by substituting each category-labeled edge with a copy of the corresponding automaton. Note that this always terminates, since the sub-automata do not have edges labeled with the categories from the same mutually recursive set.

Minimization Each of the automata is turned into a minimal *deterministic finite automaton (DFA)* by using Brzozowski's (1962) algorithm, which minimizes the automaton by performing two determinizations and reversals.

Conversion to output format The resulting finite automaton can be output in HTK *Standard Lattice Format (SLF)*. SLF supports sub-lattices, which allows us to convert our set of automata directly into a set of lattices. Since SLF uses labeled nodes, rather than labeled edges, we move the labels to the nodes. This is done by first introducing a new labeled node for each edge, and then eliminating all internal unlabeled nodes. Figure 4 shows the SLF model generated from the example grammar. For clarity, the sub-lattices have been inlined.

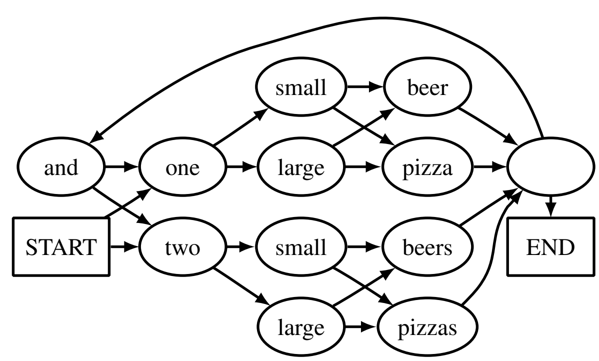

Figure 4: SLF model generated from the grammar in Figure 2.

5.2 Discussion

Finite-state models are even more restrictive than context-free grammars. This problem is handled by approximating the context-free grammar with an overgenerating finite-state automaton. This may lead to failure in a subsequent parsing step, which, as in the context-free case, is acceptable if the recognizer can return all hypotheses.

6 Semantic Interpretation

Semantic interpretation can be done as a separate parsing step after speech recognition, or it can be done with semantic information embedded in the speech recognition grammar. The latter approach resembles the *semantic actions* used by parser generators for programming languages. One formalism for semantic interpretation is the proposed *Semantic Interpretation for Speech Recognition (SISR)* standard. SISR tags are pieces of ECMAScript code embedded in the speech recognition grammar.

6.1 Algorithm

The GF system can include SISR tags when generating speech recognitions grammars in SRGS and JSGF format. The SISR tags are generated from the semantic information in the annotated CFG (Ljunglöf, 2004). The result of the semantic interpretation is an abstract syntax term.

The left-recursion elimination step makes it somewhat challenging to produce correct abstract syntax trees. We have extended Moore's (2000) LC_{LR} transform to preserve the semantic interpretation. The LC_{LR} transform introduces new categories of the form $A{-}X$ where X is a proper left corner of a category A. The new category $A{-}X$ can be understood as "the category A, but missing an initial X". Thus the semantic interpretation for a production in $A{-}X$ is the semantic interpretation for the original A-production, abstracted (in the λ-calculus sense) over the semantic interpretation of the missing X. Conversely, where-ever a category $A{-}X$ is used, its result is applied to the interpretation of the occurrence of X.

6.2 Discussion

As discussed in Section 4.2, the semantic interpretation code could be used to implement the non-context-free features of GF, but this is not yet done.

The slot-filling mechanism in the GSL format could also be used to build semantic representations, by returning program code which can then be executed. The UNIANCE grammar compiler (Bos, 2002) uses that approach.

7 Related Work

7.1 Unification Grammar Compilation

Compilation of unification grammars to speech recognition grammars is well described in the literature (Moore, 1999; Dowding et al., 2001). Regulus (Rayner et al., 2006) is perhaps the most ambitious such system. Like GF, Regulus uses a general grammar for each language, which is specialized to a domain-specific one. Ljunglöf (Ljunglöf, 2007b) relates GF and Regulus by showing how to convert GF grammars to Regulus grammars. We carry compositional semantic interpretation through left-recursion elimination using the same idea as the UNIANCE grammar compiler (Bos, 2002), though our version handles both direct and indirect left-recursion.

The main difference between our work and the existing compilers is that we work with type-theoretical grammars rather than unification grammars. While the existing work focuses on GSL as the output language, we also support a number of other formats, including finite-state models. By using the GF resource grammars, speech recognition language models can be produced for more languages than with previous systems. One shortcoming of our system is that it does not yet have support for weighted grammars.

7.2 Generating SLMs from GF Grammars

Jonson (2006) has shown that in addition to generating grammar-based language models, GF can be used to build statistical language models (SLMs). It was found that compared to our grammar-based approach, use of generated SLMs improved the recognition performance for out-of-grammar utterances significantly.

8 Results

Speech recognition grammars generated from GF grammars have already been used in a number of research dialogue systems.

GOTTIS (Bringert et al., 2005; Ericsson et al., 2006), an experimental multimodal and multilingual dialogue system for public transportation queries, uses GF grammars for parsing multimodal input. For speech recognition, it uses GSL grammars generated from the speech modality part of the GF grammars.

DJ-GoDiS, GoDiS-deLUX, and GoTGoDiS (Ericsson et al., 2006) are three applications which use GF grammars for speech recognition and parsing together with the GoDiS implementation of issue-based dialogue management (Larsson, 2002). GoTGoDiS has been translated to 7 languages using the GF resource grammar library, with each new translation taking less than one day (Ericsson et al., 2006).

The DICO (Villing and Larsson, 2006) dialogue system for trucks has recently been modified to use GF grammars for speech recognition and parsing (Ljunglöf, 2007a).

DUDE (Lemon and Liu, 2006) and its extension REALL-DUDE (Lemon et al., 2006b) are environments where non-experts can develop dialogue systems based on Business Process Models describing the applications. From keywords, prompts and answer sets defined by the developer, the system generates a GF grammar. This grammar is used for parsing input, and for generating a language model in SLF or GSL format.

The Voice Programming system by Georgila and Lemon (Georgila and Lemon, 2006; Lemon et al., 2006a) uses an SLF language model generated from a GF grammar.

Perera and Ranta (2007) have studied how GF grammars can be used for localization of dialogue systems. A GF grammar was developed and localized to 4 other languages in significantly less time than an equivalent GSL grammar. They also found the GSL grammar generated by GF to be much smaller than the hand-written GSL grammar.

9 Conclusions

We have shown how GF grammars can be compiled to several common speech recognition grammar formats. This has helped decrease development time, improve modifiability, aid localization and enable portability in a number of experimental dialogue systems.

Several systems developed in the TALK and DICO projects use the same GF grammars for speech recognition, parsing and multimodal fusion (Ericsson et al., 2006). Using the same grammar for multiple system components **reduces development and modification costs**, and makes it easier to **maintain consistency** within the system.

The feasibility of **rapid localization** of dialogue systems which use GF grammars has been demonstrated in the GoTGoDiS (Ericsson et al., 2006) system, and in experiments by Perera and Ranta (2007).

Using speech recognition grammars generated by GF makes it easy to **support different speech recognizers**. For example, by using the GF grammar compiler, the DUDE (Lemon and Liu, 2006) system can support both the ATK and Nuance recognizers.

Implementations of the methods described in this paper are freely available as part of the GF distribution[1].

Acknowledgments

Aarne Ranta, Peter Ljunglöf, Rebecca Jonson, David Hjelm, Ann-Charlotte Forslund, Håkan Burden, Xingkun Liu, Oliver Lemon, and the anonymous referees have contributed valuable comments on the grammar compiler implementation and/or this article. We would like to thank Nuance Communications, Inc., OptimSys, s.r.o., and Opera Software ASA for software licenses and technical support. The code in this paper has been typeset using `lhs2TeX`, with help from Andres Löh. This work has been partly funded by the EU TALK project, IST-507802.

References

Johan Bos. 2002. Compilation of unification grammars with compositional semantics to speech recognition packages. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Björn Bringert, Robin Cooper, Peter Ljunglöf, and Aarne Ranta. 2005. Multimodal Dialogue System Grammars. In *Proceedings of DIALOR'05, Ninth Workshop on the Semantics and Pragmatics of Dialogue*, pages 53–60.

[1] `http://www.cs.chalmers.se/~aarne/GF/`

Janusz A. Brzozowski. 1962. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, Volume 12 of MRI Symposia Series, pages 529–561. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y.

John Dowding, Beth A. Hockey, Jean M. Gawron, and Christopher Culy. 2001. Practical issues in compiling typed unification grammars for speech recognition. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 164–171, Morristown, NJ, USA. Association for Computational Linguistics.

Stina Ericsson, Gabriel Amores, Björn Bringert, Håkan Burden, Ann C. Forslund, David Hjelm, Rebecca Jonson, Staffan Larsson, Peter Ljunglöf, Pilar Manchón, David Milward, Guillermo Pérez, and Mikael Sandin. 2006. Software illustrating a unified approach to multimodality and multilinguality in the in-home domain. Technical Report 1.6, TALK Project.

Kallirroi Georgila and Oliver Lemon. 2006. Programming by Voice: enhancing adaptivity and robustness of spoken dialogue systems. In *BRANDIAL'06, Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue*, pages 199–200.

Rebecca Jonson. 2006. Generating Statistical Language Models from Interpretation Grammars in Dialogue Systems. In *Proceedings of EACL'06*.

Staffan Larsson. 2002. *Issue-based Dialogue Management*. Ph.D. thesis, Göteborg University.

Oliver Lemon and Xingkun Liu. 2006. DUDE: a Dialogue and Understanding Development Environment, mapping Business Process Models to Information State Update dialogue systems. In *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics*.

Oliver Lemon, Kallirroi Georgila, David Milward, and Tommy Herbert. 2006a. Programming Devices and Services. Technical Report 2.3, TALK Project.

Oliver Lemon, Xingkun Liu, Daniel Shapiro, and Carl Tollander. 2006b. Hierarchical Reinforcement Learning of Dialogue Policies in a development environment for dialogue systems: REALL-DUDE. In *BRANDIAL'06, Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue*, pages 185–186, September.

Peter Ljunglöf. 2004. *Expressivity and Complexity of the Grammatical Framework*. Ph.D. thesis, Göteborg University, Göteborg, Sweden.

Peter Ljunglöf. 2007a. Personal communication, March.

Peter Ljunglöf. 2007b. Converting Grammatical Framework to Regulus.

Mehryar Mohri and Mark J. Nederhof. 2001. Regular Approximation of Context-Free Grammars through Transformation. In Jean C. Junqua and Gertjan van Noord, editors, *Robustness in Language and Speech Technology*, pages 153–163. Kluwer Academic Publishers, Dordrecht.

Robert C. Moore. 1999. Using Natural-Language Knowledge Sources in Speech Recognition. In K. M. Ponting, editor, *Computational Models of Speech Pattern Processing*, pages 304–327. Springer.

Robert C. Moore. 2000. Removing left recursion from context-free grammars. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 249–255, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mark J. Nederhof. 2000. Regular Approximation of CFLs: A Grammatical View. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and other Parsing Technologies*, pages 221–241. Kluwer Academic Publishers.

Nadine Perera and Aarne Ranta. 2007. An Experiment in Dialogue System Localization with the GF Resource Grammar Library.

Aarne Ranta, Ali El Dada, and Janna Khegai. 2006. The GF Resource Grammar Library, June.

Aarne Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189, March.

Manny Rayner, Beth A. Hockey, and Pierrette Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Publications, Ventura Hall, Stanford University, Stanford, CA 94305, USA, July.

Jessica Villing and Staffan Larsson. 2006. Dico: A Multimodal Menu-based In-vehicle Dialogue System. In *BRANDIAL'06, Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue*, pages 187–188.