Coling 2008

22nd International Conference on Computational Linguistics

Proceedings of the workshop on Grammar Engineering Across Frameworks

Workshop chairs:
Stephen Clark and Tracy Holloway King

24 August 2008
Manchester, UK

Order copies of this and other Coling proceedings from:

Association for Computational Linguistics (ACL)
209 N. Eighth Street
Stroudsburg, PA 18360
USA
Tel: +1-570-476-8006
Fax: +1-570-476-0860
`acl@aclweb.org`

ISBN 978-1-905593-54-5

Design by Chimney Design, Brighton, UK
Production and manufacture by One Digital, Brighton, UK

Introduction

The GEAF workshop aims to bring together grammar engineers from different frameworks to compare research and methodologies, particularly around the themes of evaluation, modularity, maintainability, relevance to theoretical and computational linguistics, and applications of "deep" grammars to real-world domains and NLP tasks.

Recent years have seen the development of techniques and resources to support robust, deep grammatical analysis of natural language in real-world domains and applications. The demands of these types of tasks have resulted in significant advances in areas such as parser efficiency, hybrid statistical/symbolic approaches to disambiguation, and the acquisition of large-scale lexicons. The effective acquisition, development, maintenance and enhancement of grammars is a central issue in such efforts, and the size and complexity of realistic grammars makes these tasks extremely challenging; indeed, these tasks are often tackled in ways that have much in common with software engineering. This workshop aims to bring together grammar engineers from different frameworks — for example LFG, HPSG, TAG, CCG, dependency grammar — to compare their research and methodologies.

We would like to thank the program committee Jason Baldridge, Emily Bender, Miriam Butt, Aoife Cahill, John Carroll, Ann Copestake, Berthold Crysmann, Mary Dalrymple, Stefanie Dipper, Dan Flickinger, Josef van Genabith, Ron Kaplan, Montserrat Marimon, Yusuke Miyao, Owen Rambow, and Jesse Tseng for their detailed reviews of all of the submitted papers, and Prof. Jun'ichi Tsujii for agreeing to give the invited talk.

In addition, we would like to thank Mark Stevenson for helping with general COLING workshop organization and Roger Evans for helping create these proceedings.

Invited Speaker:

Jun'ichi Tsujii, Univerity of Tokyo and University of Manchester

Table of Contents

Conference Programme

Sunday, August 24, 2008

9:00–9:15 Opening Remarks

9:15–10:30 Invited Talk by Jun'ichi Tsujii

10:30–11:00 Break

11:00–11:30 *TuLiPA: Towards a Multi-Formalism Parsing Environment for Grammar Engineering*
Laura Kallmeyer, Timm Lichte, Wolfgang Maier, Yannick Parmentier, Johannes Dellert and Kilian Evang

11:30–12:00 *Making Speech Look Like Text in the Regulus Development Environment*
Elisabeth Kron, Manny Rayner, Marianne Santaholma, Pierrette Bouillon and Agnes Lisowska

12:00–12:30 *A More Precise Analysis of Punctuation for Broad-Coverage Surface Realization with CCG*
Michael White and Rajakrishnan Rajkumar

12:30-14:00 Lunch

14:00–14:30 *Multilingual Grammar Resources in Multilingual Application Development*
Marianne Santaholma

14:30–15:00 *Speeding up LFG Parsing Using C-Structure Pruning*
Aoife Cahill, John T. Maxwell III, Paul Meurer, Christian Rohrer and Victoria Rosén

15:00–15:30 *From Grammar-Independent Construction Enumeration to Lexical Types in Computational Grammars*
Lars Hellan

15:30–16:30 Demo Session with break

TuLiPA: Towards a Multi-Formalism Parsing Environment for Grammar Engineering (Laura Kallmeyer, Timm Lichte, Wolfgang Maier, Yannick Parmentier, Johannes Dellert and Kilian Evang)

Natural Language Entailment Using Implicit Information: An XLE Implementation (Daniel G. Bobrow, Bob Cheslow, Cleo Condoravdi, Lauri Karttunen, Tracy Holloway King, Charlotte Price and Annie Zaenen)

The Regulus Development Environment (Elisabeth Kron, Manny Rayner, Pierrette Bouillon, Marianne Santaholma and Agnes Lisowska)

Sunday, August 24, 2008 (continued)

TuLiPA: Towards a Multi-Formalism Parsing Environment for Grammar Engineering

Laura Kallmeyer
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
lk@sfs.uni-tuebingen.de

Timm Lichte
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
timm.lichte@uni-tuebingen.de

Wolfgang Maier
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
wo.maier@uni-tuebingen.de

Yannick Parmentier
CNRS - LORIA
Nancy Université
F-54506, Vandœuvre, France
parmenti@loria.fr

Johannes Dellert
SFB 441 - SfS
Universität Tübingen
D-72074, Tübingen, Germany

Kilian Evang
SFB 441 - SfS
Universität Tübingen
D-72074, Tübingen, Germany

{jdellert,kevang}@sfs.uni-tuebingen.de

Abstract

In this paper, we present an open-source parsing environment (Tübingen Linguistic Parsing Architecture, TuLiPA) which uses Range Concatenation Grammar (RCG) as a pivot formalism, thus opening the way to the parsing of several mildly context-sensitive formalisms. This environment currently supports tree-based grammars (namely Tree-Adjoining Grammars (TAG) and Multi-Component Tree-Adjoining Grammars with Tree Tuples (TT-MCTAG)) and allows computation not only of syntactic structures, but also of the corresponding semantic representations. It is used for the development of a tree-based grammar for German.

1 Introduction

Grammars and lexicons represent important linguistic resources for many NLP applications, among which one may cite dialog systems, automatic summarization or machine translation. Developing such resources is known to be a complex task that needs useful tools such as parsers and generators (Erbach, 1992).

Furthermore, there is a lack of a common framework allowing for multi-formalism grammar engineering. Thus, many formalisms have been proposed to model natural language, each coming with specific implementations. Having a common framework would facilitate the comparison between formalisms (e.g., in terms of parsing complexity in practice), and would allow for a better sharing of resources (e.g., having a common lexicon, from which different features would be extracted depending on the target formalism).

In this context, we present a parsing environment relying on a general architecture that can be used for parsing with mildly context-sensitive (MCS) formalisms[1] (Joshi, 1987). Its underlying idea is to use Range Concatenation Grammar (RCG) as a pivot formalism, for RCG has been shown to strictly include MCS languages while being parsable in polynomial time (Boullier, 2000).

Currently, this architecture supports tree-based grammars (Tree-Adjoining Grammars and Multi-Component Tree-Adjoining Grammars with Tree Tuples (Lichte, 2007)). More precisely, tree-based grammars are first converted into equivalent RCGs, which are then used for parsing. The result of RCG parsing is finally interpreted to extract a derivation structure for the input grammar, as well as to perform additional processings (e.g., semantic calculus, extraction of dependency views).

The paper is structured as follows. In section 2, we present the architecture of the TuLiPA parsing environment and show how the use of RCG as a pivot formalism makes it easier to design a modular system that can be extended to support several dimensions (syntax, semantics) and/or formalisms. In section 3, we give some desiderata for grammar engineering and present TuLiPA's current state

[1] A formalism is said to be mildly context sensitive (MCS) iff (i) it generates limited cross-serial dependencies, (ii) it is polynomially parsable, and (iii) the string languages generated by the formalism have the constant growth property (e.g., $\{a^{2^n} | n \geq 0\}$ does not have this property). Examples of MCS formalisms include Tree-Adjoining Grammars, Combinatory Categorial Grammars and Linear Indexed Grammars.

with respect to these. In section 4, we compare this system with existing approaches for parsing and more generally for grammar engineering. Finally, in section 5, we conclude by presenting future work.

2 Range Concatenation Grammar as a pivot formalism

The main idea underlying TuLiPA is to use RCG as a pivot formalism for RCG has appealing formal properties (e.g., a generative capacity lying beyond Linear Context Free Rewriting Systems and a polynomial parsing complexity) and there exist efficient algorithms, for RCG parsing (Boullier, 2000) and for grammar transformation into RCG (Boullier, 1998; Boullier, 1999).

Parsing with TuLiPA is thus a 3-step process:

1. The input tree-based grammar is converted into an RCG (using the algorithm of Kallmeyer and Parmentier (2008) when dealing with TT-MCTAG).
2. The resulting RCG is used for parsing the input string using an extension of the parsing algorithm of Boullier (2000).
3. The RCG derivation structure is interpreted to extract the derivation and derived trees with respect to the input grammar.

The use of RCG as a pivot formalism, and thus of an RCG parser as a core component of the system, leads to a modular architecture. In turns, this makes TuLiPA more easily extensible, either in terms of functionalities, or in terms of formalisms.

2.1 Adding functionalities to the parsing environment

As an illustration of TuLiPA's extensibility, one may consider two extensions applied to the system recently.

First, a semantic calculus using the syntax/semantics interface for TAG proposed by Gardent and Kallmeyer (2003) has been added. This interface associates each tree with flat semantic formulas. The arguments of these formulas are unification variables, which are co-indexed with features labelling the nodes of the syntactic tree. During classical TAG derivation, trees are combined, triggering unifications of the feature structures labelling nodes. As a result of these unifications, the arguments of the semantic formulas are unified (see Fig. 1).

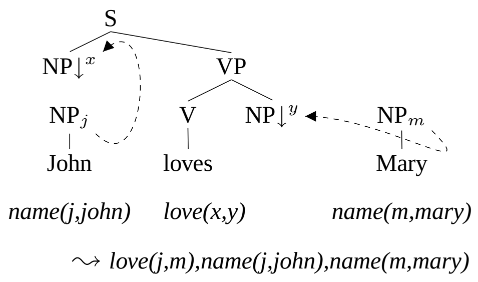


Figure 1: Semantic calculus in Feature-Based TAG.

In our system, the semantic support has been integrated by (i) extending the internal tree objects to include semantic formulas (the RCG-conversion is kept unchanged), and (ii) extending the construction of the derived tree (step 3) so that during the interpretation of the RCG derivation in terms of tree combinations, the semantic formulas are carried and updated with respect to the feature unifications performed.

Secondly, let us consider lexical disambiguation. Because of the high redundancy lying within lexicalized formalisms such as lexicalized TAG, it is common to consider tree schemata having a frontier node marked for *anchoring* (i.e., lexicalization). At parsing time, the tree schemata are anchored according to the input string. This anchoring selects a subgrammar supposed to cover the input string. Unfortunately, this subgrammar may contain many trees that either do not lead to a parse or for which we know *a priori* that they cannot be combined within the same derivation (so we should not predict a derivation from one of these trees to another during parsing). As a result, the parser could have poor performance because of the many derivation paths that have to be explored. Bonfante et al. (2004) proposed to polarize the structures of the grammar, and to apply an automaton-based filtering of the compatible structures. The idea is the following. One compute polarities representing the needs/resources brought by a given tree (or tree tuple for TT-MCTAG). A substitution or foot node with category NP reflects a need for an NP (written NP-). In the same way, an NP root node reflects a resource of type NP (written NP+). Then you build an automaton whose edges correspond to trees, and states to polarities brought by trees along the path. The automaton is then traversed to extract all paths leading to a final state with a neutral polarity for each category and +1 for the axiom (see Fig. 2, the state

7 is the only valid state and {proper., trans., det., noun.} the only compatible set of trees).

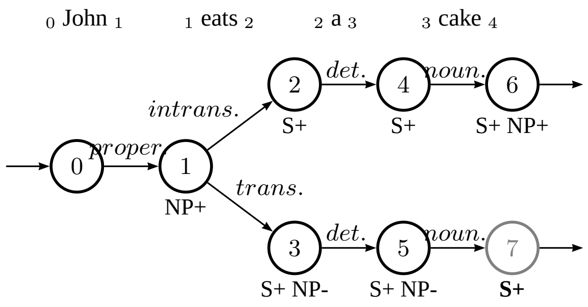


Figure 2: Polarity-based lexical disambiguation.

In our context, this polarity filtering has been added before step 1, leaving untouched the core RCG conversion and parsing steps. The idea is to compute the sets of compatible trees (or tree tuples for TT-MCTAG) and to convert these sets separately. Indeed the RCG has to encode only valid adjunctions/substitutions. Thanks to this automaton-based "clustering" of the compatible tree (or tree tuples), we avoid predicting incompatible derivations. Note that the time saved by using a polarity-based filter is not negligible, especially when parsing long sentences.[2]

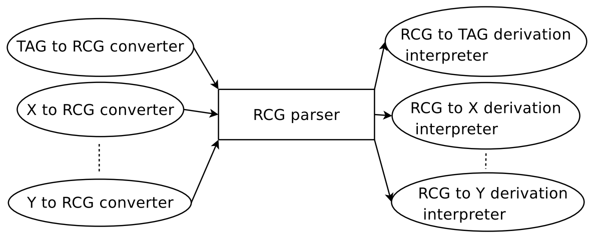
2.2 Adding formalisms to the parsing environment

Of course, the two extensions introduced in the previous section may have been added to other modular architectures as well. The main gain brought by RCG is the possibility to parse not only tree-based grammars, but other formalisms provided they can be encoded into RCG. In our system, only TAG and TT-MCTAG have been considered so far. Nonetheless, Boullier (1998) and Søgaard (2007) have defined transformations into RCG for other mildly context-sensitive formalisms.[3]

To sum up, the idea would be to keep the core RCG parser, and to extend TuLiPA with a specific conversion module for each targeted formalism. On top of these conversion modules, one should also provide interpretation modules allowing to decode the RCG derivation forest in terms of the input formalism (see Fig. 3).

[2]An evaluation of the gain brought by this technique when using Interaction Grammar is given by Bonfante et al. (2004).

[3]These include Multi-Component Tree-Adjoining Grammar, Linear Indexed Grammar, Head Grammar, Coupled Context Free Grammar, Right Linear Unification Grammar and Synchronous Unification Grammar.



TAG to RCG converter
X to RCG converter
Y to RCG converter
RCG parser
RCG to TAG derivation interpreter
RCG to X derivation interpreter
RCG to Y derivation interpreter

Figure 3: Towards a multi-formalism parsing environment.

An important point remains to be discussed. It concerns the role of lexicalization with respect to the formalism used. Indeed, the tree-based grammar formalisms currently supported (TAG and TT-MCTAG) both share the same lexicalization process (i.e., tree *anchoring*). Thus the lexicon format is common to these formalisms. As we will see below, it corresponds to a 2-layer lexicon made of inflected forms and lemma respectively, the latter selecting specific grammatical structures. When parsing other formalisms, it is still unclear whether one can use the same lexicon format, and if not what kind of general lexicon management module should be added to the parser (in particular to deal with morphology).

3 Towards a complete grammar engineering environment

So far, we have seen how to use a generic parsing architecture relying on RCG to parse different formalisms. In this section, we adopt a broader view and enumerate some requirements for a linguistic resource development environment. We also see to what extent these requirements are fulfilled (or partially fulfilled) within the TuLiPA system.

3.1 Grammar engineering with TuLiPA

As advocated by Erbach (1992), grammar engineering needs *"tools for testing the grammar with respect to consistency, coverage, overgeneration and accuracy"*. These characteristics may be taken into account by different interacting software. Thus, consistency can be checked by a semi-automatic grammar production device, such as the XMG system of Duchier et al. (2004). Overgeneration is mainly checked by a generator (or by a parser with adequate test suites), and coverage and accuracy by a parser. In our case, the TuLiPA system provides an entry point for using a grammar production system (and a lexicon conversion tool introduced below), while including a parser. Note that TuLiPA does not include any generator, nonetheless it uses the same lexicon format as the GenI surface realizer for TAG[4].

TuLiPA's input grammar is designed using XMG, which is a *metagrammar* compiler for tree-based formalisms. In other terms, the linguist defines a factorized description of the grammar (the so-called metagrammar) in the XMG language. Briefly, an XMG metagrammar consists of (i) elementary tree fragments represented as tree description logic formulas, and (ii) conjunctive and disjunctive combinations of these tree fragments to describe actual TAG tree schemata.[5] This metagrammar is then compiled by the XMG system to produce a tree grammar in an XML format. Note that the resulting grammar contains tree schemata (i.e., unlexicalized trees). To lexicalize these, the linguist defines a lexicon mapping words with corresponding sets of trees. Following XTAG (2001), this lexicon is a 2-layer lexicon made of morphological and lemma specifications. The motivation of this 2-layer format is (i) to express linguistic generalizations at the lexicon level, and (ii) to allow the parser to only select a subgrammar according to a given sentence, thus reducing parsing complexity. TuLiPA comes with a lexicon conversion tool (namely lexConverter) allowing to write a lexicon in a user-friendly text format and to convert it into XML. An example of an entry of such a lexicon is given in Fig. 4.

The morphological specification consists of a word, the corresponding lemma and morphological features. The main pieces of information contained in the lemma specification are the ∗ENTRY field, which refers to the lemma, the ∗CAT field referring to the syntactic category of the anchor node, the ∗SEM field containing some semantic information allowing for semantic instantiation, the ∗FAM field, which contains the name of the tree family to be anchored, the ∗FILTERS field which consists of a feature structure constraining by unification the trees of a given family that can be anchored by the given lemma (used for instance for non-passivable verbs), the ∗EQUATIONS field allowing for the definition of equations targeting named nodes of the trees, and the ∗COANCHORS field, which allows for the specification of co-anchors (such as *by* in the verb *to come by*).

[4]`http://trac.loria.fr/~geni`

[5]See (Crabbé, 2005) for a presentation on how to use the XMG formalism for describing a core TAG for French.

Morphological specification:

```
vergisst    vergessen    [pos=v,num=sg,per=3]
```

Lemma specification:

```
∗ENTRY: vergessen
∗CAT: v
∗SEM: BinaryRel[pred=vergessen]
∗ACC: 1
∗FAM: Vnp2
∗FILTERS: []
∗EX:
∗EQUATIONS:
NParg1 → cas = nom
NParg2 → cas = acc
∗COANCHORS:
```

Figure 4: Morphological and lemma specification of *vergisst*.

From these XML resources, TuLiPA parses a string, corresponding either to a sentence or a constituent (noun phrase, prepositional phrase, *etc.*), and computes several output pieces of information, namely (for TAG and TT-MCTAG): derivation/derived trees, semantic representations (computed from underspecified representations using the utool software[6], or dependency views of the derivation trees (using the DTool software[7]).

3.2 Grammar debugging

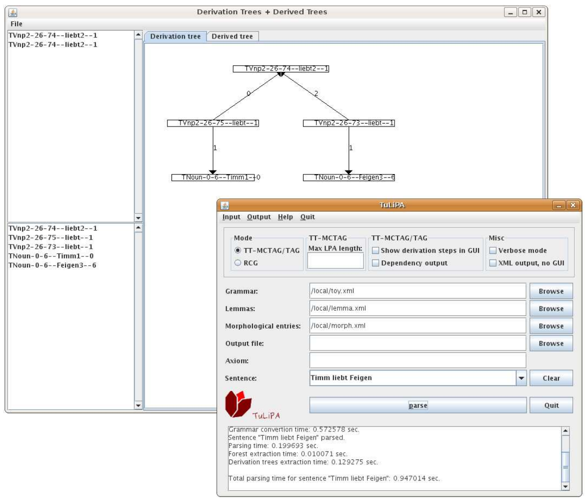
The engineering process introduced in the preceding section belongs to a development cycle, where one first designs a grammar and corresponding lexicons using XMG, then checks these with the parser, fixes them, parses again, and so on.

To facilitate grammar debugging, TuLiPA includes both a verbose and a robust mode allowing respectively to (i) produce a log of the RCG-conversion, RCG-parsing and RCG-derivation interpretation, and (ii) display mismatching features leading to incomplete derivations. More precisely, in robust mode, the parser displays derivations step by step, highlighting feature unification failures.

TuLiPA's options can be activated via an intuitive Graphical User Interface (see Fig. 5).

[6]See `http://www.coli.uni-saarland.de/projects/chorus/utool/`, with courtesy of Alexander Koller.

[7]With courtesy of Marco Kuhlmann.



Mode: [✓ TT-MCTAG/TAG] RCG
TT-MCTAG Max LPA length:
TT-MCTAG/TAG: Show derivation steps in GUI; Dependency output
Misc: Verbose mode; XML output, no GUI

Grammar: /local/toy.xml
Lemmas: /local/lemma.xml
Morphological entries: /local/morph.xml
Output file:
Axiom:
Sentence: Timm liebt Feigen

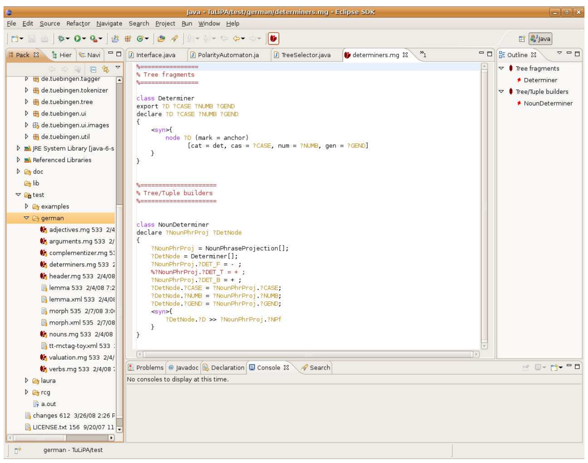
```
Grammar convertion time: 0.572578 sec.
Sentence "Timm liebt Feigen" parsed.
Parsing time: 0.199693 sec.
Forest extraction time: 0.010071 sec.
Derivation trees extraction time: 0.129275 sec.

Total parsing time for sentence "Timm liebt Feigen": 0.947014 sec.
```

Figure 5: TuLiPA's Graphical User Interface.

3.3 Towards a functional common interface

Unfortunately, as mentioned above, the linguist has to move back-and-forth from the grammar/lexicon descriptions to the parser, i.e., each time the parser reports grammar errors, the linguist fixes these and then recomputes the XML files and then parses again. To avoid this tedious task of resources re-compilation, we started developing an Eclipse[8] plug-in for the TuLiPA system. Thus, the linguist will be able to manage all these resources, and to call the parser, the metagrammar compiler, and the lexConverter from a common interface (see Fig. 6).



```
%================
% Tree fragments
%================

class Determiner
export ?D ?CASE ?NUMB ?GEND
declare ?D ?CASE ?NUMB ?GEND
{
    <syn>{
        node ?D (mark = anchor)
              [cat = det, cas = ?CASE, num = ?NUMB, gen = ?GEND]
    }
}


%====================
% Tree/Tuple builders
%====================


class NounDeterminer
declare ?NounPhrProj ?DetNode
{
    ?NounPhrProj = NounPhraseProjection[];
    ?DetNode = Determiner[];
    ?NounPhrProj.?DET_F = - ;
    %?NounPhrProj.?DET_T = + ;
    ?NounPhrProj.?DET_B = + ;
    ?DetNode.?CASE = ?NounPhrProj.?CASE;
    ?DetNode.?NUMB = ?NounPhrProj.?NUMB;
    ?DetNode.?GEND = ?NounPhrProj.?GEND;
    <syn>{
        ?DetNode.?D >> ?NounPhrProj.?NPf
    }
}
```

No consoles to display at this time.

Figure 6: TuLiPA's eclipse plug-in.

The motivation for this plug-in comes from the observation that designing electronic grammars is a task comparable to designing source code. A powerful grammar engineering environment should thus come with development facilities such as precise debugging information, syntax highlighting, *etc*. Using the Eclipse open-source development platform allows for reusing several components inherited from the software development community, such as plug-ins for version control, editors coupled with explorers, *etc*.

Eventually, one point worth considering in the context of grammar development concerns data encoding. To our knowledge, only few environments provide support for UTF-8 encoding, thus guarantying the coverage of a wide set of charsets and languages. In TuLiPA, we added an UTF-8 support (in the lexConverter), thus allowing to design a TAG for Korean (work in progress).

3.4 Usability of the TuLiPA system

As mentioned above, the TuLiPA system is made of several interacting components, that one currently has to install separately. Nonetheless, much attention has been paid to make this installation process as easy as possible and compatible with all major platforms.[9]

XMG and lexConverter can be installed by compiling their sources (using a *make* command). TuLiPA is developed in Java and released as an executable jar. No compilation is needed for it, the only requirement is the Gecode/GecodeJ library[10] (available as a binary package for many platforms). Finally, the TuLiPA eclipse plug-in can be installed easily from eclipse itself. All these tools are released under Free software licenses (either GNU GPL or Eclipse Public License).

This environment is being used (i) at the University of Tübingen, in the context of the development of a TT-MCTAG for German describing both syntax and semantics, and (ii) at LORIA Nancy, in the development of an XTAG-based metagrammar for English. The German grammar, called GerTT (for German Tree Tuples), is released under a LGPL license for Linguistic Resources[11] and is presented in (Kallmeyer et al., 2008). The test-suite currently used to check the grammar is hand-crafted. A more systematic evaluation of the grammar is in preparation, using the Test Suite for Natural Language Processing (Lehmann et al., 1996).

[8]See `http://www.eclipse.org`

[9]See `http://sourcesup.cru.fr/tulipa`.

[10]See `http://www.gecode.org/gecodej`.

[11]See `http://infolingu.univ-mlv.fr/DonneesLinguistiques/Lexiques-Grammaires/lgpllr.html`

4 Comparison with existing approaches

4.1 Engineering environments for tree-based grammar formalisms

To our knowledge, there is currently no available parsing environment for multi-component TAG.

Existing grammar engineering environments for TAG include the DyALog system[12] described in Villemonte de la Clergerie (2005). DyALog is a compiler for a logic programming language using tabulation and dynamic programming techniques. This compiler has been used to implement efficient parsing algorithms for several formalisms, including TAG and RCG. Unfortunately, it does not include any built-in GUI and requires a good knowledge of the GNU build tools to compile parsers. This makes it relatively difficult to use. DyALog's main quality lies in its efficiency in terms of parsing time and its capacity to handle very large resources. Unlike TuLiPA, it does not compute semantic representations.

The closest approach to TuLiPA corresponds to the SemTAG system[13], which extends TAG parsers compiled with DyALog with a semantic calculus module (Gardent and Parmentier, 2007). Unlike TuLiPA, this system only supports TAG, and does not provide any graphical output allowing to easily check the result of parsing.

Note that, for grammar designers mainly interested in TAG, SemTAG and TuLiPA can be seen as complementary tools. Indeed, one may use TuLiPA to develop the grammar and check specific syntactic structures thanks to its intuitive parsing environment. Once the grammar is stable, one may use SemTAG in batch processing to parse corpuses and build semantic representations using large grammars. This combination of these 2 systems is made easier by the fact that both use the same input formats (a metagrammar in the XMG language and a text-based lexicon). This approach is the one being adopted for the development of a French TAG equipped with semantics.

For Interaction Grammar (Perrier, 2000), there exists an engineering environment gathering the XMG metagrammar compiler and an eLEtrOstatic PARser (LEOPAR).[14] This environment is being used to develop an Interaction Grammar for French. TuLiPA's lexical disambiguation module reuses techniques introduced by LEOPAR. Unlike TuLiPA, LEOPAR does not currently support semantic information.

[12]See `http://dyalog.gforge.inria.fr`

[13]See `http://trac.loria.fr/~semconst`

[14]See `http://www.loria.fr/equipes/calligramme/leopar/`

4.2 Engineering environments for other grammar formalisms

For other formalisms, there exist state-of-the-art grammar engineering environments that have been used for many years to design large deep grammars for several languages.

For Lexical Functional Grammar, one may cite the Xerox Linguistic Environment (XLE).[15] For Head-driven Phrase Structure Grammar, the main available systems are the Linguistic Knowledge Base (LKB)[16] and the TRALE system.[17] For Combinatory Categorial Grammar, one may cite the OpenCCG library[18] and the C&C parser.[19]

These environments have been used to develop broad-coverage resources equipped with semantics and include both a generator and a parser. Unlike TuLiPA, they represent advanced projects, that have been used for dialog and machine translation applications. They are mainly tailored for a specific formalism.[20]

5 Future work

In this section, we give some prospective views concerning engineering environments in general, and TuLiPA in particular. We first distinguish between 2 main usages of grammar engineering environments, namely a pedagogical usage and an application-oriented usage, and finally give some comments about multi-formalism.

5.1 Pedagogical usage

Developing grammars in a pedagogical context needs facilities allowing for inspection of the structures of the grammar, step-by-step parsing (or generation), along with an intuitive interface. The idea is to abstract away from technical aspects related to implementation (intermediate data structures, optimizations, etc.).

[15]See `http://www2.parc.com/isl/groups/nltt/xle/`

[16]See `http://wiki.delph-in.net/moin`

[17]See `http://milca.sfs.uni-tuebingen.de/A4/Course/trale/`

[18]See `http://openccg.sourceforge.net/`

[19]See `http://svn.ask.it.usyd.edu.au/trac/candc/wiki`

[20]Nonetheless, Beavers (2002) encoded a CCG in the LKB's Type Description Language.

The question whether to provide graphical or text-based editors can be discussed. As advocated by Baldridge et al. (2007), a low-level text-based specification can offer more flexibility and bring less frustration to the grammar designer, especially when such a specification can be graphically interpreted. This is the approach chosen by XMG, where the grammar is defined via an (advanced or not) editor such as gedit or emacs. Within TuLiPA, we chose to go further by using the Eclipse platform. Currently, it allows for displaying a summary of the content of a metagrammar or lexicon on a side panel, while editing these on a middle panel. These two panels are linked via a jump functionality. The next steps concern (i) the plugging of a graphical viewer to display the (meta)grammar structures independently from a given parse, and (ii) the extension of the eclipse plug-in so that one can easily consistently modify entries of the metagrammar or lexicon (especially when these are split over several files).

5.2 Application-oriented usage

When dealing with applications, one may demand more from the grammar engineering environment, especially in terms of efficiency and robustness (support for larger resources, partial parsing, etc.).

Efficiency needs optimizations in the parsing engine making it possible to support grammars containing several thousands of structures. One interesting question concerns the compilation of a grammar either off-line or on-line. In DyALog's approach, the grammar is compiled off-line into a logical automaton encoding all possible derivations. This off-line compilation can take some minutes with a TAG having 6000 trees, but the resulting parser can parse sentences within a second.

In TuLiPA's approach, the grammar is compiled into an RCG on-line. While giving satisfactory results on reduced resources[21], it may lead to troubles when scaling up. This is especially true for TAG (the TT-MCTAG formalism is by definition a factorized formalism compared with TAG). In the future, it would be useful to look for a way to precompile a TAG into an RCG off-line, thus saving the conversion time.

Another important feature of grammar engineering environments consists of its debugging functionalities. Among these, one may cite unit and integration testing. It would be useful to extend the TuLiPA system to provide a module for generating test-suites for a given grammar. The idea would be to record the coverage and analyses of a grammar at a given time. Once the grammar is further developed, these snapshots would allow for regression testing.

[21]For a TT-MCTAG counting about 300 sets of trees and an and-crafted lexicon made of about 300 of words, a 10-word sentence is parsed (and a semantic representation computed) within seconds.

5.3 About multi-formalism

We already mentioned that TuLiPA was opening a way towards multi-formalism by relying on an RCG core. It is worth noticing that the XMG system was also designed to be further extensible. Indeed, a metagrammar in XMG corresponds to the combination of elementary structures. One may think of designing a library of such structures, these would be dependent on the target grammar formalism. The combinations may represent general linguistic concepts and would be shared by different grammar implementations, following ideas presented by Bender et al. (2005).

6 Conclusion

In this paper, we have presented a multi-formalism parsing architecture using RCG as a pivot formalism to parse mildly context-sensitive formalisms (currently TAG and TT-MCTAG). This system has been designed to facilitate grammar development by providing user-friendly interfaces, along with several functionalities (e.g., dependency extraction, derivation/derived tree display and semantic calculus). It is currently used for developing a core grammar for German.

At the moment, we are working on the extension of this architecture to include a fully functional Eclipse plug-in. Other current tasks concern optimizations to support large scale parsing and the extension of the syntactic and semantic coverage of the German grammar under development.

In a near future, we plan to evaluate the parser and the German grammar (parsing time, correction of syntactic and semantic outputs) with respect to a standard test-suite such as the TSNLP (Lehmann et al., 1996).

Acknowledgments

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) and the Deutscher Akademischer Austausch Dienst (DAAD, grant

A/06/71039). We are grateful to three anonymous reviewers for valuable comments on this work.

References

Baldridge, Jason, Sudipta Chatterjee, Alexis Palmer, and Ben Wing. 2007. DotCCG and VisCCG: Wiki and programming paradigms for improved grammar engineering with OpenCCG. In King, Tracy Holloway and Emily M. Bender, editors, *Proceedings of the GEAF07 workshop*, pages 5–25, Stanford, CA. CSLI.

Beavers, John. 2002. Documentation: A CCG Implementation for the LKB. LinGO Working Paper No. 2002-08, CSLI, Stanford University, Stanford, CA.

Bender, Emily, Dan Flickinger, Frederik Fouvry, and Melanie Siegel. 2005. Shared representation in multilingual grammar engineering. *Research on Language & Computation*, 3(2):131–138.

Bonfante, Guillaume, Bruno Guillaume, and Guy Perrier. 2004. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of the International Conference on Computational Linguistics (CoLing 2004)*, pages 303–309, Geneva, Switzerland.

Boullier, Pierre. 1998. Proposal for a natural language processing syntactic backbone. Rapport de Recherche 3342, INRIA.

Boullier, Pierre. 1999. On TAG and Multicomponent TAG Parsing. Rapport de Recherche 3668, INRIA.

Boullier, Pierre. 2000. Range concatenation grammars. In *Proceedings of the International Workshop on Parsing Technologies (IWPT 2000)*, pages 53–64, Trento, Italy.

Crabbé, Benoit. 2005. Grammatical development with XMG. In *Proceedings of the conference on Logical Aspects of Computational Linguistics 2005 (LACL 05)*, pages 84–100, Bordeaux, France.

Duchier, Denys, Joseph Le Roux, and Yannick Parmentier. 2004. The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In *Proceedings of the 2nd International Mozart/Oz Conference (MOZ'2004)*, pages 175–187, Charleroi, Belgium.

Erbach, Gregor. 1992. Tools for grammar engineering. In *3rd Conference on Applied Natural Language Processing*, pages 243–244, Trento, Italy.

Gardent, Claire and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *Proceedings of the Conference of the European chapter of the Association for Computational Linguistics (EACL 2003)*, pages 123–130, Budapest, Hungary.

Gardent, Claire and Yannick Parmentier. 2007. Semtag: a platform for specifying tree adjoining grammars and performing tag-based semantic construction. In *Proceedings of the International Conference of the Association for Computational Linguistics (ACL 2007), Companion Volume Proceedings of the Demo and Poster Sessions*, pages 13–16, Prague, Czech Republic.

Joshi, Aravind K. 1987. An introduction to Tree Adjoining Grammars. In Manaster-Ramer, A., editor, *Mathematics of Language*, pages 87–114. John Benjamins, Amsterdam.

Kallmeyer, Laura and Yannick Parmentier. 2008. On the relation between Multicomponent Tree Adjoining Grammars with Tree Tuples (TT-MCTAG) and Range Concatenation Grammars (RCG). In *Proceedings of the 2nd International Conference on Language and Automata Theories and Applications (LATA 2008)*, pages 277–288, Tarragona, Spain.

Kallmeyer, Laura, Timm Lichte, Wolfgang Maier, Yannick Parmentier, and Johannes Dellert. 2008. Developping an MCTAG for German with an RCG-based Parser. In *Proceedings of the Language, Resource and Evaluation Conference (LREC 2008)*, Marrakech, Morocco.

Lehmann, Sabine, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, Veronika Lux, Judith Klein, Kirsten Falkedal, Frederik Fouvry, Dominique Estival, Eva Dauphin, Hervé Compagnion, Judith Baur, Lorna Balkan, and Doug Arnold. 1996. TSNLP — Test Suites for Natural Language Processing. In *Proceedings of the International Conference on Computational Linguistics (Coling 1996)*, volume 2, pages 711–716, Copenhagen, Denmark.

Lichte, Timm. 2007. An MCTAG with tuples for coherent constructions in German. In *Proceedings of the 12th Conference on Formal Grammar*, Dublin, Ireland.

Perrier, Guy. 2000. Interaction grammars. In *Proceedings of the International Conference on Computational Linguistics (CoLing 2000)*, pages 600–606, Saarbruecken, Germany.

Søgaard, Anders. 2007. *Complexity, expressivity and logic of linguistic theories*. Ph.D. thesis, University of Copenhagen, Copenhagen, Denmark.

Villemonte de la Clergerie, Éric. 2005. DyALog: a tabular logic programming based environment for NLP. In *Proceedings of the workshop on Constraint Satisfaction for Language Processing (CSLP 2005)*, pages 18–33, Barcelona, Spain.

XTAG-Research-Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania. Available at `http://www.cis.upenn.edu/~xtag/gramrelease.html`.

Making Speech Look Like Text in the Regulus Development Environment

Elisabeth Kron
3 St Margarets Road, Cambridge CB3 0LT, England
elisabethkron@yahoo.co.uk

Manny Rayner, Marianne Santaholma, Pierrette Bouillon, Agnes Lisowska
University of Geneva, TIM/ISSCO, 40 bvd du Pont-d'Arve
CH-1211 Geneva 4, Switzerland
Emmanuel.Rayner@issco.unige.ch
Marianne.Santaholma@eti.unige.ch
Pierrette.Bouillon@issco.unige.ch
Agnes.Lisowska@issco.unige.ch

Abstract

We present an overview of the development environment for Regulus, an Open Source platform for construction of grammar-based speech-enabled systems, focussing on recent work whose goal has been to introduce uniformity between text and speech views of Regulus-based applications. We argue the advantages of being able to switch quickly between text and speech modalities in interactive and offline testing, and describe how the new functionalities enable rapid prototyping of spoken dialogue systems and speech translators.

1 Introduction

Sex is not love, as Madonna points out at the beginning of her 1992 book *Sex,* and love is not sex. None the less, even people who agree with Madonna often find it convenient to pretend that these two concepts are synonymous, or at least closely related. Similarly, although text is not speech, and speech is not text, it is often convenient to pretend that they are both just different aspects of the same thing.

In this paper, we will explore the similarities and differences between text and speech, in the concrete setting of Regulus, a development environment for grammar based spoken dialogue systems. Our basic goal will be to make text and speech processing as similar as possible from the point of view of the developer. Specifically, we arrange things so that the developer is able to develop her system using a text view; she will write text-based rules, and initially test the system using text input and output. At any point, she will be able to switch to a speech view, compiling the text-based processing rules into corresponding speech-based versions, and test the resulting speech-based system using speech input and output.

Paradoxically, the reason why it is so important to be able to switch seamlessly between text and speech viewpoints is that text and speech are in fact *not* the same. For example, a pervasive problem in speech recognition is that of easily confusable pairs of words. This type of problem is often apparent after just a few minutes when running the system in speech mode (the recogniser keeps recognising one word as the other), but is invisible in text mode. More subtly, some grammar problems can be obvious in text mode, but hard to see in speech mode. For instance, articles like "the" and "a" are short, and usually pronounced unstressed, which means that recognisers can be reasonably forgiving about whether or not to hypothesise them when they are required or not required by the recognition grammar. In text mode, it will immediately be clear if the grammar requires an article in a given NP context: incorrect variants will fail to parse. In speech mode, the symptoms are far less obvious, and typically amount to no more than a degradation in recognition performance.

The rest of the paper is structured as follows. Sections 2 and 3 provide background on the Regulus platform and development cycle respectively. Section 4 describes speech and text support in the interactive development environment, and 5 describes how the framework simplifies the task of switching between modalities in regression testing. Section 6 concludes.

2 The Regulus platform

The Regulus platform is a comprehensive toolkit for developing grammar-based speech-enabled systems that can be run on the commercially available Nuance recognition environment. The platform has been developed by an Open Source consortium, the main partners of which have been NASA Ames Research Center and Geneva University, and is freely available for download from the SourceForge website[1]. In terms of ideas (though not code), Regulus is a descendent of SRI International's CLE and Gemini platforms (Alshawi, 1992; Dowding et al., 1993); other related systems are LKB (Copestake, 2002), XLE (Crouch et al., 2008) and UNIANCE (Bos, 2002).

Regulus has already been used to build several large applications. Prominent examples are Geneva University's MedSLT medical speech translator (Bouillon et al., 2005), NASA's Clarissa procedure browser (Rayner et al., 2005) and Ford Research's experimental SDS in-car spoken dialogue system, which was awarded first prize at the 2007 Ford internal demo fair. Regulus is described at length in (Rayner et al., 2006), the first half of which consists of an extended tutorial introduction. The release includes a command-line development environment, extensive online documentation, and several example applications.

The core functionality offered by Regulus is compilation of typed unification grammars into parsers, generators, and Nuance-formatted CFG language models, and hence also into Nuance recognition packages. These recognition packages produced by Regulus can be invoked through the Regulus SpeechServer ("Regserver"), which provides an interface to the underlying Nuance recognition engine. The value added by the Regserver is to provide a view of the recognition process based on the Regulus unification grammar framework. In particular, recognition results, originally produced in the Nuance recognition platform's internal format, are reformatted into the semantic notation used by the Regulus grammar formalism.

There is extensive support within the Regulus toolkit for development of both speech translation and spoken dialogue applications. Spoken dialogue applications (Rayner et al., 2006, Chapter 5) use a rule-based side-effect free state update model similar in spirit to that described in (Larsson and Traum, 2000). Very briefly, there are three types of rules: state update rules, input management rules, and output management rules. State update rules take as input the current state, and a "dialogue move"; they produce as output a new state, and an "abstract action". Dialogue moves are abstract representations of system inputs; these inputs can either be logical forms produced by the grammar, or non-speech inputs (for example, mouse-clicks in a GUI). Similarly, abstract actions are, as the name suggests, abstract representations of the concrete actions the dialogue system will perform, for example speaking or updating a visual display. Input management rules map system inputs to dialogue moves; output management rules map abstract actions to system outputs.

Speech translation applications are also rule-based, using an interlingua model (Rayner et al., 2006, Chapter 6). The developer writes a second grammar for the target language, using Regulus tools to compile it into a generator; mappings from source representation to interlingua, and from interlingua to target representation, are defined by sets of translation rules. The interlingua itself is specified using a third Regulus grammar (Bouillon et al., 2008).

To summarise, the core of a Regulus application consists of several different linguistically oriented rule-sets, some of which can be interpreted in either a text or a speech modality, and all of which need to interact correctly together. In the next section, we describe how this determines the nature of the Regulus development cycle.

3 The Regulus development cycle

Small unification grammars can be compiled directly into executable forms. The central idea of Regulus, however, is to base as much of the development work as possible on large, domain-independent, linguistically motivated resource grammars. A resource grammar for English is available from the Regulus website; similar grammars for several other languages have been developed under the MedSLT project at Geneva University, and can be downloaded from the MedSLT SourceForge website[2]. Regulus contains

[1]`http://sourceforge.net/projects/regulus/`

[2]`http://sourceforge.net/projects/medslt`

an extensive set of tools that permit specialised domain-specific grammars to be extracted from the larger resource grammars, using example-based methods driven by small corpora (Rayner et al., 2006, Chapter 7). At the beginning of a project, these corpora can consist of just a few dozen examples; for a mature application, they will typically have grown to something between a few hundred and a couple of thousand sentences. Specialised grammars can be compiled by Regulus into efficient recognisers and generators.

As should be apparent from the preceding description, the Regulus architecture is designed to empower linguists to the maximum possible extent, in terms of increasing their ability directly to build speech enabled systems; the greater part of the core development teams in the large Regulus projects mentioned in Section 1 have indeed come from linguistics backgrounds. Experience with Regulus has however shown that linguists are not quite as autonomous as they are meant to be, and in particular are reluctant to work directly with the speech view of the application. There are several reasons.

First, non-toy Regulus projects require a range of competences, including both software engineering and linguistics. In practice, linguist rule-writers have not been able to test their rules in the speech view without writing glue code, scripts, and other infrastructure required to tie together the various generated components. These are not necessarily things that they want to spend their time doing. The consequence can easily be that the linguists end up working exclusively in the text view, and over-refine the text versions of the rule-sets. From a project management viewpoint, this results in bad prioritisation decisions, since there are more pressing issues to address in the speech view.

A second reason why linguist rule-writers have been unhappy working in the speech view is the lack of reproducibility associated with speech input. One can type "John loves Mary" into a text-processing system any number of times, and expect to get the same result. It is much less reasonable to expect to get the same result each time if one *says* "John loves Mary" to a speech recogniser. Often, anomalous results occur, but cannot be debugged in a systematic fashion, leading to general frustration. The result, once again, is that linguists have preferred to stick with the text view, where they feel at home.

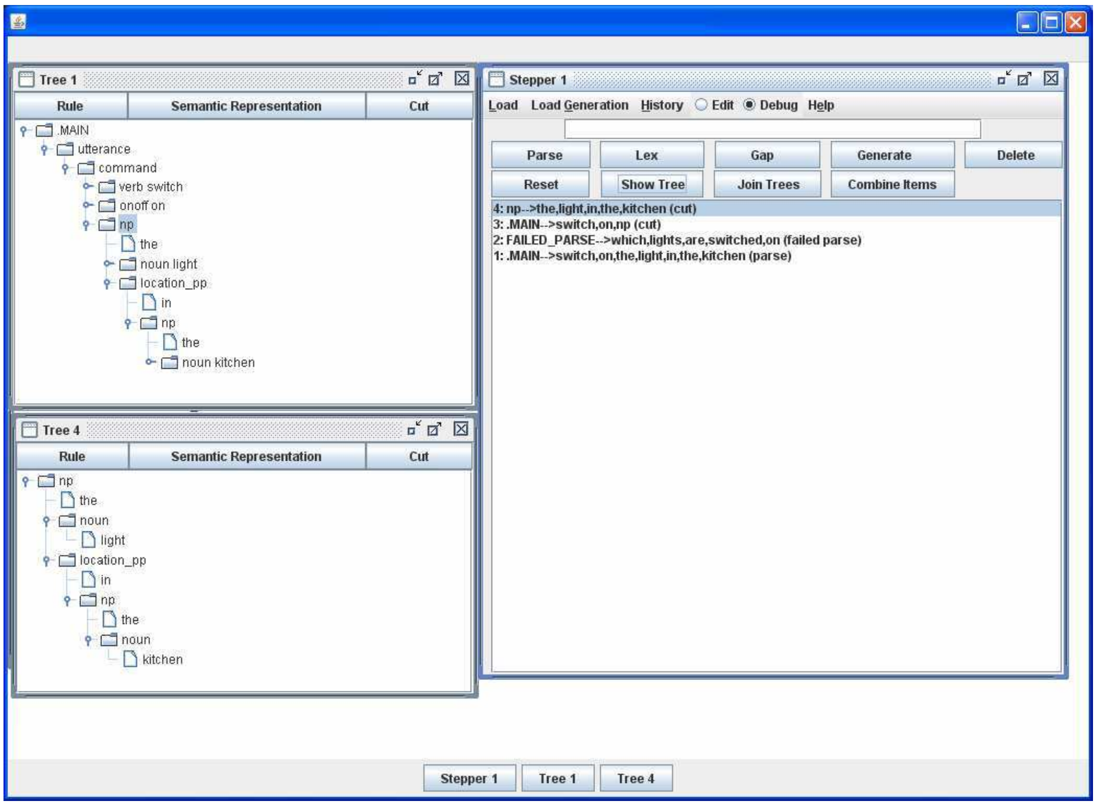
Yet another reason why rule-writers tend to limit themselves to the text view is simply the large number of top-level commands and intermediate compilation results. The current Regulus command-line environment includes over 110 different commands, and compilation from the initial resource grammar to the final Nuance recognition package involves creating a sequence of five compilation steps, each of which requires the output created by the preceding one. This makes it difficult for novice users to get their bearings, and increases their cognitive load. Additionally, once the commands for the text view have been mastered, there is a certain temptation to consider that these are enough, since the text and speech views can reasonably be perceived as fairly similar.

In the next two sections, we describe an enhanced development environment for Regulus, which addresses the key problems we have just sketched. From the point of view of the linguist rule-writer, we want speech-based development to feel more like text-based development.

4 Speech and text in the online development environment

The Regulus GUI (Kron et al., 2007) is intended as a complete redesign of the development environment, which simultaneously attacks all of the central issues. Commands are organised in a structured set of functionality-based windows, each of which has an appropriate set of drop-down menus. Following normal GUI design practice (Dix et al., 1998, Chapters 3 and 4); (Jacko and Sears, 2003, Chapter 13), only currently meaningful commands are executable in each menu, with the others shown greyed out.

Both compile-time and run-time speech-related functionality can be invoked directly from the command menus, with no need for external scripts, Makefiles or glue code. Focussing for the moment on the specific case of developing a speech translation application, the rule-writer will initially write and debug her rules in text mode. She will be able to manipulate grammar rules and derivation trees using the Stepper window (Figure 1; cf. also (Kron et al., 2007)), and load and test translation rules in the Translate window (Figure 2). As soon as the grammar is consistent, it can at any point be compiled into a Nuance recognition package using the command menus. The resulting recogniser, together with other speech resources (license man-

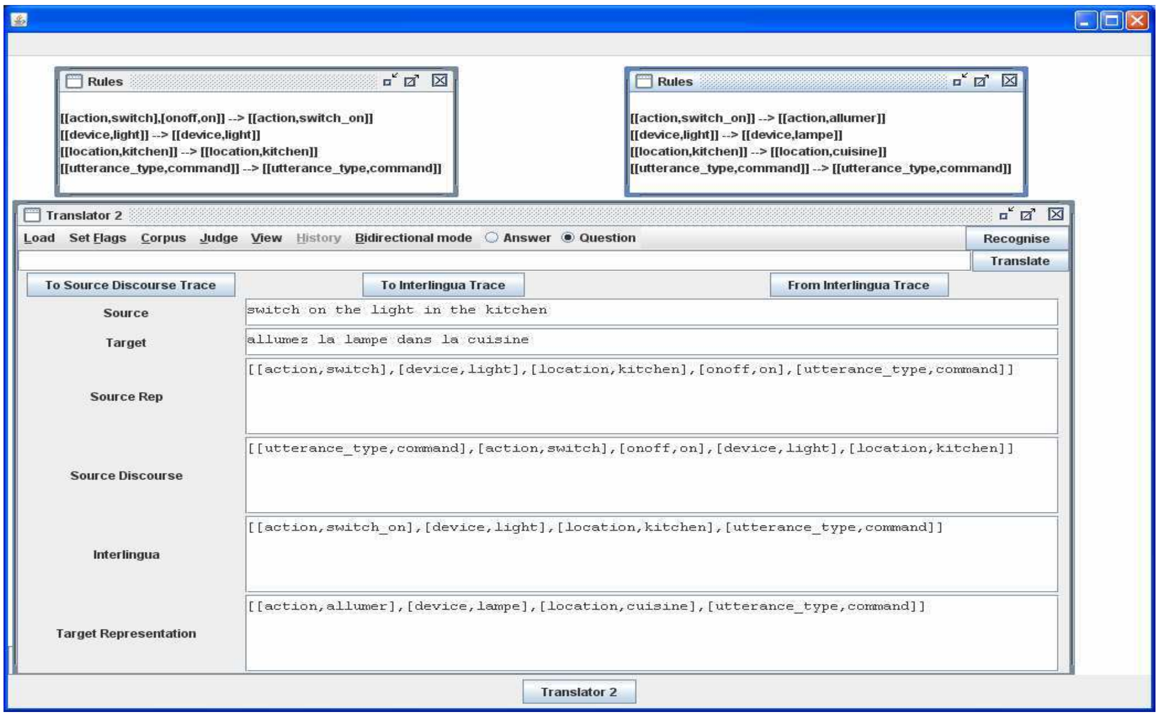


```
.MAIN
  utterance
    command
      verb switch
      onoff on
      np
        the
        noun light
        location_pp
          in
          np
            the
            noun kitchen
```

```
4: np-->the,light,in,the,kitchen (cut)
3: .MAIN-->switch,on,np (cut)
2: FAILED_PARSE-->which,lights,are,switched,on (failed parse)
1: .MAIN-->switch,on,the,light,in,the,kitchen (parse)
```

```
np
  the
  noun
    light
  location_pp
    in
    np
      the
      noun
        kitchen
```

Figure 1: Using the Stepper window to browse trees in the Toy1 grammar from (Rayner et al., 2006, Chapter 4). The upper left window shows the analysis tree for “switch on the light in the kitchen”; the lower left window shows one of the subtrees created by cutting the first tree at the higher NP node. Cut subtrees can be recombined for debugging purposes (Kron et al., 2007).



```
[[action,switch],[onoff,on]] --> [[action,switch_on]]
[[device,light]] --> [[device,light]]
[[location,kitchen]] --> [[location,kitchen]]
[[utterance_type,command]] --> [[utterance_type,command]]
```

```
[[action,switch_on]] --> [[action,allumer]]
[[device,light]] --> [[device,lampe]]
[[location,kitchen]] --> [[location,cuisine]]
[[utterance_type,command]] --> [[utterance_type,command]]
```

Source	switch on the light in the kitchen
Target	allumez la lampe dans la cuisine
Source Rep	[[action,switch],[device,light],[location,kitchen],[onoff,on],[utterance_type,command]]
Source Discourse	[[utterance_type,command],[action,switch],[onoff,on],[device,light],[location,kitchen]]
Interlingua	[[action,switch_on],[device,light],[location,kitchen],[utterance_type,command]]
Target Representation	[[action,allumer],[device,lampe],[location,cuisine],[utterance_type,command]]

Figure 2: Using the Translate window to test the toy English → French translation application from (Rayner et al., 2006, Chapter 6). The to- and from-interlingua rules used in the example are shown in the two pop-up windows at the top of the figure.

```
regulus_config(regulus_grammar,
               [toy1_grammars(toy1_declarations),
                toy1_grammars(toy1_rules),
                toy1_grammars(toy1_lexicon)]).
regulus_config(top_level_cat, '.MAIN').
regulus_config(nuance_grammar, toy1_runtime(recogniser)).

regulus_config(to_interlingua_rules,
               toy1_prolog('eng_to_interlingua.pl')).
regulus_config(from_interlingua_rules,
               toy1_prolog('interlingua_to_fre.pl')).
regulus_config(generation_rules, toy1_runtime('generator.pl')).

regulus_config(nuance_language_pack,
               'English.America').
regulus_config(nuance_compile_params, ['-auto_pron', '-dont_flatten']).
regulus_config(translation_rec_params,
               [package=toy1_runtime(recogniser), grammar='.MAIN']).
regulus_config(tts_command,
    'vocalizer -num_channels 1 -voice juliedeschamps -voices_from_disk').
```

Figure 3: Config file for a toy English → French speech translation application, showing items relevant to the speech view. Some declarations have been omitted for expositional reasons.

ager, TTS engine etc), can then be started using a single menu command.

In accordance with the usual Regulus design philosophy of declaring all the resources associated with a given application in its config file, the speech resources are also specified here. Figure 3 shows part of the config file for a toy translation application, in particular listing all the declarations relevant to the speech view. If we needed to change the speech resources, this would be done just by modifying the last four lines. For example, the config file as shown specifies construction of a recogniser using acoustic models appropriate to American English. We could change this to British English by replacing the entry

```
regulus_config(nuance_language_pack,
               'English.America').
```

with

```
regulus_config(nuance_language_pack,
               'English.UK').
```

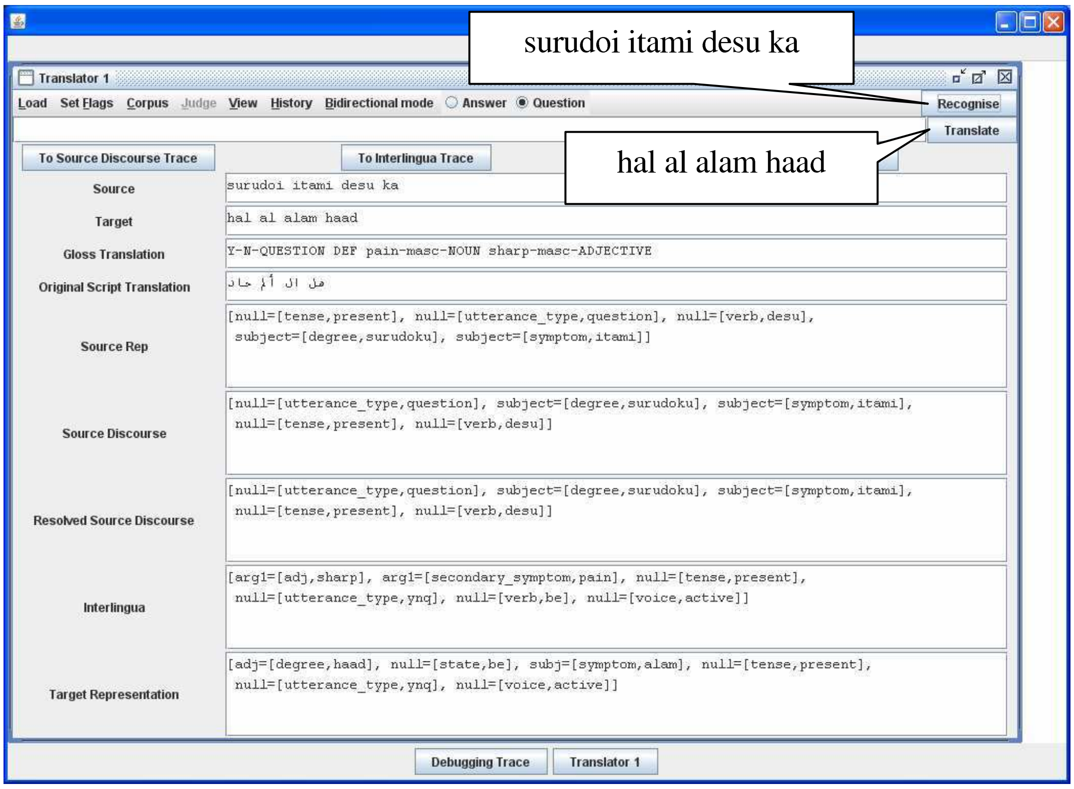
When the speech resources have been loaded, the Translate window can take input equally easily in text or speech mode; the Translate button processes written text from the input pane, while the Recognise button asks for spoken input. In each case, the input is passed through the same processing stages of source-to-interlingua and interlingua-to-target translation, followed by target-language generation. If a TTS engine or a set of recorded target language wavfiles is specified, they are used to realise the final result in spoken form (Figure 4).

Every spoken utterance submitted to recognition is logged as a SPHERE-headed wavfile, in a time-stamped directory started at the beginning of the current session; this directory also contains a meta-data file, which associates each recorded wavfile with the recognition result it produced. The Translate window's History menu is constructed using the meta-data file, and allows the user to select any recorded utterance, and re-run it through the system as though it were a new speech input. The consequence is that speech input becomes just as reproducible as text, with corresponding gains for interactive debugging in speech mode.

5 Speech and text in regression testing

In earlier versions of the Regulus development environment (Rayner et al., 2006, §6.6), regression testing in speech mode was all based on Nuance's `batchrec` utility, which permits offline recognition of a set of recorded wavfiles. A test suite for spoken regression testing consequently consisted of a list of wavfiles. These were first passed through `batchrec`; outputs were then post-processed into Regulus form, and finally passed through Regulus speech understanding modules, such as translation or dialogue management.

As Regulus applications grow in complexity, this model has become increasingly inadequate, since system input is very frequently not just a list of monolingual speech events. In a multimodal dialogue system, input can consist of either speech or screen events (text/mouse-clicks); context is generally important, and the events have to be processed in the order in which they occurred. Dialogue systems which control real or simulated robots, like the Wheelchair application of (Hockey



Source	surudoi itami desu ka
Target	hal al alam haad
Gloss Translation	Y-N-QUESTION DEF pain-masc-NOUN sharp-masc-ADJECTIVE
Original Script Translation	هل ال ألم حاد
Source Rep	[null=[tense,present], null=[utterance_type,question], null=[verb,desu], subject=[degree,surudoku], subject=[symptom,itami]]
Source Discourse	[null=[utterance_type,question], subject=[degree,surudoku], subject=[symptom,itami], null=[tense,present], null=[verb,desu]]
Resolved Source Discourse	[null=[utterance_type,question], subject=[degree,surudoku], subject=[symptom,itami], null=[tense,present], null=[verb,desu]]
Interlingua	[arg1=[adj,sharp], arg1=[secondary_symptom,pain], null=[tense,present], null=[utterance_type,ynq], null=[verb,be], null=[voice,active]]
Target Representation	[adj=[degree,haad], null=[state,be], subj=[symptom,alam], null=[tense,present], null=[utterance_type,ynq], null=[voice,active]]

Figure 4: Speech to speech translation from the GUI, using a Japanese to Arabic translator built from MedSLT components (Bouillon et al., 2008). The user presses the Recognise button (top right), speaks in Japanese, and receives a spoken translation in Arabic together with screen display of various processing results. The application is defined by a config file which combines a Japanese recogniser and analysis grammar, Japanese to Interlingua and Interlingua to Arabic translation rules, an Arabic generation grammar, and recorded Arabic wavfiles used to construct a spoken result.

and Miller, 2007) will also receive asynchronous inputs from the robot control and monitoring process; once again, all inputs have to be processed in the appropriate temporal order. A third example is contextual bidirectional speech translation (Bouillon et al., 2007). Here, the problem is slightly different — we have only speech inputs, but they are for two different languages. The basic issue, however, remains the same, since inputs have to be processed in the right order to maintain the correct context at each point.

With examples like these in mind, we have also effected a complete redesign of the Regulus environment's regression testing facilities. A test suite is now allowed to consist of a list of items of any type — text, wavfile, or non-speech input — in any order. Instead of trying to fit processing into the constraints imposed by the `batchrec` utility, offline processing now starts up speech resources in the same way as the interactive environment, and submits each item for appropriate processing in the order in which it occurs. By adhering to the principle that text and speech should be treated uniformly, we arrive at a framework which is simpler, less error-prone (the underlying code is less fragile) and above all much more flexible.

6 Summary and conclusions

The new functionality offered by the redesigned Regulus top-level is not strikingly deep. In the context of any given application, it could all have been duplicated by reasonably simple scripts, which linked together existing Regulus components. Indeed, much of this new functionality is implemented using code derived precisely from such scripts. Our observation, however, has been that few developers have actually taken the time to write these scripts, and that when they have been developed inside one project they have usually not migrated to other ones. One of the things we have done, essentially, is to generalise previously *ad hoc* application-dependent functionality, and make it part of the top-level development environment. The other main achievements of the new Regulus top-level are to organise the existing functionality in a more systematic way, so that it is easier to find commands, and to package it all as a normal-looking Swing-based GUI.

Although none of these items sound dramatic, they make a large difference to the platform's overall usability, and to the development cycle it supports. In effect, the Regulus top-level becomes a generic speech-enabled application, into which developers can plug their grammars, rule-sets and derived components. Applications can be tested in the speech view much earlier, giving a correspondingly better chance of catching bad design decisions before they become entrenched. The mechanisms used to enable this functionality do not depend on any special properties of Regulus, and could readily be implemented in other grammar-based development platforms, such as Gemini and UNIANCE, which support compilation of feature grammars into grammar-based language models.

At risk of stating the obvious, it is also worth pointing out that many users, particularly younger ones who have grown up using Windows and Mac environments, expect as a matter of course that development platforms will be GUI-based rather than command-line. Addressing this issue, and simplifying the transition between text- and speech-based, views has the pleasant consequence of improving Regulus as a vehicle for introducing linguistics students to speech technology. An initial Regulus-based course at the University of Santa Cruz, focussing on spoken dialogue systems, is described in (Hockey and Christian, 2008); a similar one, but oriented towards speech translation and using the new top-level described here, is currently under way at the University of Geneva. We expect to present this in detail in a later paper.

References

Alshawi, H., editor. 1992. *The Core Language Engine*. MIT Press, Cambridge, Massachusetts.

Bos, J. 2002. Compilation of unification grammars with compositional semantics to speech recognition packages. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Bouillon, P., M. Rayner, N. Chatzichrisafis, B.A. Hockey, M. Santaholma, M. Starlander, Y. Nakao, K. Kanzaki, and H. Isahara. 2005. A generic multilingual open source platform for limited-domain medical speech translation. In *Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT)*, pages 50–58, Budapest, Hungary.

Bouillon, P., G. Flores, M. Starlander, N. Chatzichrisafis, M. Santaholma, N. Tsourakis, M. Rayner, and B.A. Hockey. 2007. A bidirectional grammar-based medical speech translator. In *Proceedings of the ACL Workshop on Grammar-based Approaches to Spoken Language Processing*, pages 41–48, Prague, Czech Republic.

Bouillon, P., S. Halimi, Y. Nakao, K. Kanzaki, H. Isahara, N. Tsourakis, M. Starlander, B.A. Hockey, and M. Rayner. 2008. Developing non-european translation pairs in a medium-vocabulary medical speech translation system. In *Proceedings of LREC 2008*, Marrakesh, Morocco.

Copestake, A. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Press, Chicago.

Crouch, R., M. Dalrymple, R. Kaplan, T. King, J. Maxwell, and P. Newman, 2008. *XLE Documentation*. http://www2.parc.com/isl/groups/nltt/xle/doc. As of 29 Apr 2008.

Dix, A., J.E. Finlay, G.D. Abowd, and R. Beale, editors. 1998. *Human Computer Interaction. Second ed.* Prentice Hall, England.

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

Hockey, B.A. and G. Christian. 2008. Zero to spoken dialogue system in one quarter: Teaching computational linguistics to linguists using regulus. In *Proceedings of the Third ACL Workshop on Teaching Computational Linguistics (TeachCL-08)*, Columbus, OH.

Hockey, B.A. and D. Miller. 2007. A demonstration of a conversationally guided smart wheelchair. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 243–244, Denver, CO.

Jacko, J.A. and A. Sears, editors. 2003. *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications*. Lawerence Erlbaum Associates, Mahwah, New Jersey.

Kron, E., M. Rayner, P. Bouillon, and M. Santaholma. 2007. A development environment for building grammar-based speech-enabled applications. In *Proceedings of the ACL Workshop on Grammar-based Approaches to Spoken Language Processing*, pages 49–52, Prague, Czech Republic.

Larsson, S. and D. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340.

Rayner, M., B.A. Hockey, J.M. Renders, N. Chatzichrisafis, and K. Farrell. 2005. A voice enabled procedure browser for the international space station. In *Proceedings of the 43rd*

Annual Meeting of the Association for Computational Linguistics (interactive poster and demo track), Ann Arbor, MI.

Rayner, M., B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.

A More Precise Analysis of Punctuation for Broad-Coverage Surface Realization with CCG

Michael White and **Rajakrishnan Rajkumar**
Department of Linguistics
The Ohio State University
Columbus, OH, USA
{mwhite,raja}@ling.osu.edu

Abstract

This paper describes a more precise analysis of punctuation for a bi-directional, broad coverage English grammar extracted from the CCGbank (Hockenmaier and Steedman, 2007). We discuss various approaches which have been proposed in the literature to constrain overgeneration with punctuation, and illustrate how aspects of Briscoe's (1994) influential approach, which relies on syntactic features to constrain the appearance of balanced and unbalanced commas and dashes to appropriate sentential contexts, is unattractive for CCG. As an interim solution to constrain overgeneration, we propose a rule-based filter which bars illicit sequences of punctuation and cases of improperly unbalanced apposition. Using the OpenCCG toolkit, we demonstrate that our punctuation-augmented grammar yields substantial increases in surface realization coverage and quality, helping to achieve state-of-the-art BLEU scores.

1 Introduction

In his pioneering monograph, Nunberg (1990) argues that punctuation is a systematic module of the grammar of written text and is governed by principles and constraints like other sub-systems such as syntax or phonology. Since then, others including Briscoe (1994) and Doran (1998) have explored ways of including rules and representations for punctuation marks in broad coverage grammars. In computational systems, punctuation provides disambiguation cues which can help parsers arrive at the correct parse. From a natural language generation standpoint, text without punctuation can be difficult to comprehend, or even misleading.

In this paper, we describe a more precise analysis of punctuation for a bi-directional, broad coverage English grammar extracted from the CCGbank (Hockenmaier and Steedman, 2007). In contrast to previous work, which has been primarily oriented towards parsing, our goal has been to develop an analysis of punctuation that is well suited for both parsing and surface realization. In addition, while Briscoe and Doran have simply included punctuation rules in their manually written grammars, our approach has been to revise the CCGbank itself with punctuation categories and more precise linguistic analyses, and then to extract a grammar from the enhanced corpus.

In developing our analysis, we illustrate how aspects of Briscoe's (1994) approach, which relies on syntactic features to constrain the appearance of balanced and unbalanced commas and dashes to appropriate sentential contexts, is unattractive for CCG, with its more flexible handling of word order. Consequently, as an interim solution, we have chosen to identify and filter undesirable configurations when scoring alternative realizations. We also point to other ways in which punctuation constraints could be incorporated into the grammar, for exploration in future work.

Using the OpenCCG toolkit, we demonstrate that our punctuation-enhanced grammar yields substantial increases in surface realization quality, helping to achieve state-of-the-art BLEU scores. We use non-blind testing to evaluate the efficacy of the grammar, and blind-testing to evaluate its performance on unseen data. The baseline models are (1) a grammar which has lexicalized punctuation categories only for conjunction and apposition, and (2) one which has punctuation categories corresponding to the existing treatment of punctuation in the corpus. Non-blind testing results shown a nearly 9-point increase in BLEU scores compared to the best baseline model using oracle n-grams, as well as a 40% increase in exact matches. Blind testing results show a more than 5.5-point increase in BLEU scores, contributing to an all-sentences score of 0.7323 on Section 23 with over 96% coverage.

2 Background

CCG (Steedman, 2000) is a unification-based categorial grammar formalism which is defined almost entirely in terms of lexical entries that encode sub-categorization information as well as syntactic feature information (e.g. number and agreement). Complementing function application as the standard means of combining a head with its argument, type-raising and composition support transparent analyses for a wide range of phenomena, including right-node raising and long distance dependencies. Semantic composition happens in parallel with syntactic composition, which makes it attractive for generation.

OpenCCG is a parsing/generation library which works by combining lexical categories for words using CCG rules and multi-modal extensions on rules (Baldridge, 2002) to produce derivations. Surface realization is the process by which logical forms are transduced to strings. OpenCCG uses a hybrid symbolic-statistical chart realizer (White, 2006) which takes logical forms as input and produces sentences by using CCG combinators to combine signs. Alternative realizations are ranked using integrated n-gram scoring.

To illustrate the input to OpenCCG, consider the semantic dependency graph in Figure 1. In the graph, each node has a lexical predication (e.g. **make.03**) and a set of semantic features (e.g. ⟨NUM⟩sg); nodes are connected via dependency relations (e.g. ⟨ARG0⟩). Internally, such graphs are represented using Hybrid Logic Dependency Semantics (HLDS), a dependency-based approach to representing linguistic meaning (Baldridge and Kruijff, 2002). In HLDS, each semantic head (corresponding to a node in the graph) is associated with a nominal that identifies its discourse referent, and relations between heads and their dependents are modeled as modal relations.

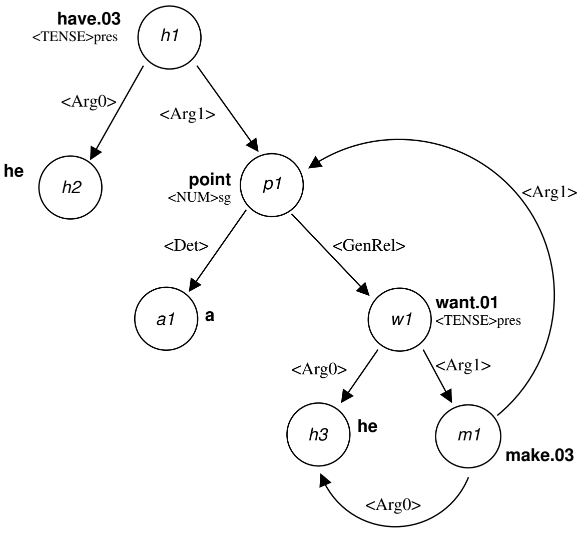


Figure 1: Semantic dependency graph from the CCGbank for *He has a point he wants to make [. . .]*

3 The need for an OpenCCG analysis of punctuation

The linguistic analysis aims to make a broad coverage OpenCCG grammar extracted from the CCGbank (White et al., 2007) more precise by adding lexicalized punctuation categories to deal with constructions involving punctuation. The original CCGbank corpus does not have lexical categories for punctuation; instead, punctuation marks carry categories derived from their part of speech tags and form part of a binary rule. It is assumed that there are no dependencies between words and punctuation marks and that the result of punctuation rules is the same as the non-punctuation category. OpenCCG does not support non-combinatory binary rules, as they can be replaced by equivalent lexicalized categories with application-only slashes. For example, a binary rule of the form $, \ s \Rightarrow s$ can be replaced by the equivalent category $s_{\langle 1 \rangle}/_{\star}s_{\langle 1 \rangle}$ for the comma. In fact, this would work reasonably well for parsing, but is inadequate for generation. To illustrate, consider (1):

(1) Despite recent declines in yields, investors continue to pour cash into money funds. (wsj_0004.10)

A comma category like the one shown above would end up overgenerating, as sentences and sentential complements would be generated with a comma preceding them. Also, the result of the above function application rule could act as its own argument, producing a string of commas. More generally, binary rules miss out on many linguistic generalizations, such as the presence of mandatory balancing marks in sentence-medial comma or dash adjuncts.

The literature discusses various means to address the issue of overgeneration: absorption rules (Nunberg, 1990), syntactic features (Doran, 1998) and (Briscoe, 1994) and semantic features (White, 2006). Section 5 explains these approaches in detail, and considers a possible system of syntactic features for a multi-modal CCG grammar implementation. We show how such a system is inadequate to constrain all possible cases of overgeneration, motivating our decision to employ semantic features in our bi-directional grammar.

4 Integrating an analysis of punctuation into the grammar

As our starting point, we used an XML representation of an enhanced version of the CCGbank with Propbank roles projected onto it (Boxwell and White, 2008). Contexts and constructions in which punctuation marks occur were isolated and the corpus was then restructured by inserting new categories and modified derivations using XSL transforms. In many cases this also involved modifying the gold standard derivations substantially and adding semantic representations to syntactic categories using logical form templates. Currently, the algorithm succeeds in creating logical forms for 98.01% of the sentences in the development section (Sect. 00) of the converted CCGbank, and 96.46% of the sentences in the test section (Sect. 23). Of these, 92.10% of the development LFs are semantic dependency graphs with a single root, while 92.12% of the test LFs have a single root. The remaining cases, with multiple roots, are missing one or more dependencies required to form a fully connected graph. These missing dependencies usually reflect inadequacies in the current logical form templates. In Section 00, 89 punctuation categories were created (66 commas, 14 dashes and 3 each for the rest) out of 54 classes of binary rules (37 comma, 8 dash, 3 apiece of colon, parenthesis and dots). Three high frequency comma categories are explained below.

4.1 Sentential Adjuncts

The comma in example (1) has been analysed as selecting a sentential modifier to its left, *Despite recent declines in yields*, to result in a sentential modifier which then selects the rest of the sentence. This results in the following lexical category and semantics for the comma category:

(2) $, \vdash \mathsf{s}_{\langle 1\rangle ind=X1, mod=M} / \mathsf{s}_{\langle 1\rangle} \backslash_{\star} (\mathsf{s}_{\langle 1\rangle} / \mathsf{s}_{\langle 1\rangle})$
$: @_M(\langle \text{EMPH-INTRO}\rangle +)$

Syntactic categories and their semantics are linked by index variables in the feature structures of categories. Index variables for semantic heads (e.g. $X1$) are conventionally named X plus the number of the feature structure. To support modifier modifiers, as in (2), semantic heads of modifiers are also made available through a modifier index feature, with a variable conventionally named M.[1] Here, the effect of combining the comma with the phrase headed by *despite* is to add the $\langle$EMPH-INTRO$\rangle+$ feature to the *despite*-phrase's semantics. Following (Bayraktar et al., 1998), this feature indicates that the comma has the discourse function of emphasizing an introductory clause or phrase. During realization, the feature triggers the look-up of the category in (2), and prevents the re-application of the category to its own output (as the feature should only be realized once).

The category in (2) illustrates our approach, which is to assign to every punctuation mark (other than balancing marks) a category whose LF includes a feature or relation which represents its discourse semantic function in broad-brush terms such as emphasis, elaboration and apposition.

4.2 Verbs of reported speech

In (3), the comma which follows *Neverthless* and sets off the phrase headed by *said* has the category in (4):

(3) Nevertheless, said Brenda Malizia Negus, editor of Money Fund Report, yields may blip up again before they blip down because of recent rises in short-term interest rates. (wsj_0004.8)

(4) $, \vdash \mathsf{s}_{\langle 2\rangle} / \mathsf{s}_{\langle 2\rangle} /_{\star} \mathsf{punct}[,] /_{\star} (\mathsf{s}_{\langle 1\rangle dcl} \backslash \mathsf{s}_{\langle 2\rangle dcl})$
$: @_{X2}(\langle \text{ElabRel}\rangle \wedge X1)$

[1] A limited form of default unification is used in the implementation to keep multiple modifiers from conflicting. As the names of index variables are entirely predictable, they are suppressed in the remainder of the paper.

In the genre of newswire text, this construction occurs frequently with verbs of reported speech. The CCGbank derivation of (3) assigns the category $\mathsf{s}_{\langle 1\rangle dcl}\backslash\mathsf{s}_{\langle 2\rangle dcl}$ to the phrase headed by *said*, the same category that is used when the phrase follows the missing sentential complement. The comma category in (4) selects for this category and a balancing comma and then converts it to a pre-sentential modifier, $\mathsf{s}_{\langle 2\rangle}/\mathsf{s}_{\langle 2\rangle}$. Semantically, an elaboration relation is added between the main clause and the reported speech phrase.

Category (4) overgenerates to some extent in that it will allow a comma at the beginning of the sentence. To prevent this, an alternative would be to make the comma explicitly select for lexical material to its left (in this case for the category of *Nevertheless*). Another possibility would be to follow Doran (1998) in analyzing the above construction by using the verb itself to select for the comma. However, since our method involves changing the gold standard derivations, and since making the verb select extra commas or having the comma select leftward material would entail substantial further changes to the derivations, we have opted to go with (4), balancing adequacy and convenience.

4.3 NP appositives

Neither the Penn Tree Bank nor the CCGbank distinguishes between NP appositives and NP conjunctions. We wrote a set of simple heuristic rules to enforce this distinction, which is vital to generation. Appositives can occur sentence medially or finally. The conventions of writing mandate that sentence medial appositives should be balanced—i.e., the appositive NP should be surrounded by commas or dashes on both sides—while sentence final appositives should be unbalanced—i.e., they should only have one preceding comma or dash. The categories and semantics for unbalanced and balanced appositive commas are, respectively:

(5) a. $, \vdash \mathsf{np}_{\langle 1\rangle}\backslash\mathsf{np}_{\langle 1\rangle}/_{\star}\mathsf{np}_{\langle 3\rangle}$
$: @_{X1}(\langle \textsc{ApposRel}\rangle \wedge X3)$

b. $, \vdash \mathsf{np}_{\langle 1\rangle}\backslash\mathsf{np}_{\langle 1\rangle}/_{\star}\mathsf{punct}[,]/_{\star}\mathsf{np}_{\langle 3\rangle}$
$: @_{X1}(\langle \textsc{ApposRel}\rangle \wedge X3)$

Here, the unbalanced appositive has a category where the comma selects as argument the appositive NP and converts it to a nominal modifier. For balanced appositives, the comma selects the appositive NP and the balancing comma to form a nominal modifier (examples are given in the next section).

5 Constraining overgeneration in bi-directional grammars

A complex issue that arises in the design of bi-directional grammars is ensuring the proper presentation of punctuation. Among other things, this involves the task of ensuring the correct realization of commas introducing noun phrase appositives—in our case, choosing when to use (5a) vs. (5b). In this section, we consider and ultimately reject a solution that follows Briscoe (1994) in using syntactic features. As an alternative, interim solution, we then describe a rule-based filter which bars illicit punctuation sequences and improperly unbalanced apposition. The paradigm below helps illustrate the issues:

(6) John, CEO of ABC, loves Mary.

(7) * John, CEO of ABC loves Mary.

(8) Mary loves John, CEO of ABC.

(9) * Mary loves John, CEO of ABC,.

(10) Mary loves John, CEO of ABC, madly.

(11) * Mary loves John, CEO of ABC madly.

5.1 Absorption vs. syntactic features

Nunberg (1990) argues that text adjuncts introduced by punctuation marks have an underlying representation where these adjuncts have marks on either side. They attain their surface form when a set of presentation rules are applied. This approach ensures that all sentence medial cases like (6) and (10) above are generated correctly, while unacceptable examples (7) and (11) would not be generated at all. Example (8) would at first be generated as (9): to deal with such sentences, where two points happen to coincide, Nunberg posits an implicit point which is absorbed by the adjacent point. Absorption occurs according to the "strength" of the two points. Strength is determined according to the Point Absorption Hierarchy, which ranks commas lower than dashes, semi-colons, colons and periods. As White (1995) observes, from a generation-only perspective, it makes sense to generate text adjuncts which are always balanced and post-process the output to delete lower ranked points, as absorption uses relatively simple rules that operate independently of the hierarchy of the constituents. However, using this approach for parsing would involve a preprocessing step which inserts commas into possible edges of possible constituents, as described in (Forst and Kaplan, 2006). To avoid this considerable complication, Briscoe (1994) has argued for developing declarative approaches involving syntactic features, with no deletions or insertions of punctuation marks.

5.2 Features for punctuation in CCG?

Unfortunately, the feature-based approach appears to be inadequate for dealing with the class of examples presented above in CCG. This approach involves the incorporation of syntactic features for punctuation into atomic categories so that certain combinations are blocked. To ensure proper appositive balancing sentence finally, the rightmost element in the sentence should transmit a relevant feature to the clause level, which the sentence-final period can then check for the presence of right-edge punctuation. Possible categories for a transitive verb and the full stop appear below:

(12) *loves* $\vdash$ $\mathsf{s}_{\langle 1\rangle bal=BAL,end=PE}\backslash\mathsf{np}_{\langle 2\rangle bal=+}/\mathsf{np}_{\langle 3\rangle bal=BAL,end=PE}$

(13) . $\vdash$ $\mathsf{sent}\backslash_{\star}\mathsf{s}_{end=nil}$

Here the feature variables *BAL* and *PE* of the rightmost argument of the verb would unify with the corresponding result category feature values to realize the main clauses of (8) and (9) with the following feature values:

(14) Mary loves John, CEO of ABC $\vdash$ $\mathsf{s}_{\langle 1\rangle bal=-,end=nil}$

(15) Mary loves John, CEO of ABC, $\vdash$ $\mathsf{s}_{\langle 1\rangle bal=+,end=comma}$

Thus, in (15), the sentence-final period would not combine with $\mathsf{s}_{\langle 1\rangle bal=+,end=comma}$ and the derivation would be blocked.[2]

5.2.1 Issue: Extraction cases

The solution sketched above is not adequate to deal with extraction involving ditransitive verbs in cases like (16) and (17):

[2]It is worth noting than an n-gram scorer would highly disprefer example (9), as a comma period sequence would not be attested in the training data. However, an n-gram model cannot be relied upon to eliminate examples like (11), which would likely be favored as they are shorter than their balanced counterparts.

(16) Mary loves a book that John gave Bill, his brother.

(17) * Mary loves a book that John gave Bill, his brother,.

As Figure 2 shows, an unacceptable case like (17) is not blocked. Even when the sentence final NP is balanced, the *end=comma* value is not propagated to the root level. This is because the *end* feature for the relative clause should depend on the first (indirect) object of *gave*, rather than the second (direct) object as in a full ditransitive clause. A possible solution would be to introduce more features which record the presence of punctuation in the leftward and rightward arguments of complex categories; this would be rather baroque, however.

5.2.2 Issue: Crossing composition

Another issue is how crossing composition, used with adverbs in heavy NP shift contructions, interacts with appositives, as in the following examples:

(18) Mary loves madly John, CEO of ABC.

(19) * Mary loves madly John, CEO of ABC,.

For examples (10) and (11), which do not involve crossing composition, the category for the adverb should be the one in (20):

(20) *madly* $\vdash$ $\mathsf{s}_{\langle 1\rangle end=nil}\backslash\mathsf{np}_{\langle 2\rangle}\backslash(\mathsf{s}_{\langle 1\rangle bal=+}\backslash\mathsf{np}_{\langle 2\rangle})$

Here the *bal*=+ feature on the argument of the adverb *madly* ensures that the direct object of the verb is balanced, as in (10); otherwise, the derivation fails, as in (11). Irrespective of the value of the *end* feature of the argument, the result of the adverb has the feature *end=nil* as the post-modifier is lexical material which occurs after the VP. With crossing composition, however, category (20) would licence an erroneous derivation for example (19), as the *end=nil* feature on the result of the adverb category would prevent the percolation of the *end* feature at the edge of the phrase to the clausal root, as Figure 3 shows.

To block such derivations, one might consider giving the adverb another category for use with crossing composition:

(21) *madly* $\vdash$ $\mathsf{s}_{\langle 1\rangle}\backslash\mathsf{np}_{\langle 2\rangle}\backslash_{\times}(\mathsf{s}_{\langle 1\rangle}\backslash\mathsf{np}_{\langle 2\rangle})$

The use of the non-associative, permutative modality $\times$ on the main slash allows the crossing composition rule to be applied, and feature inheritance

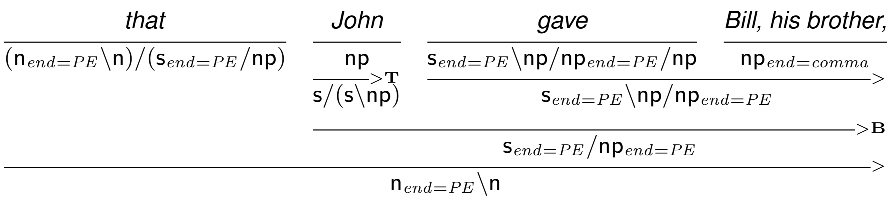


Figure 2: Object extraction

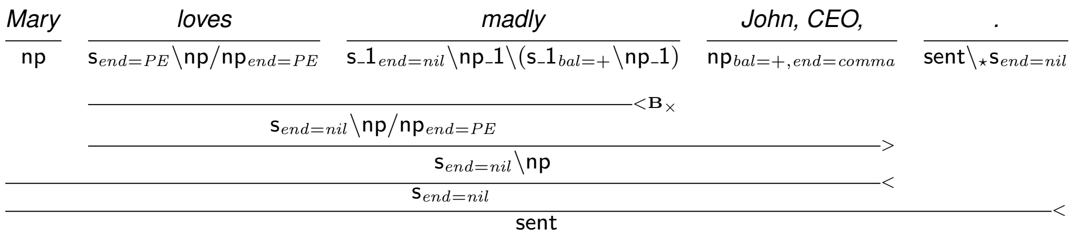

Mary
loves
madly
John, CEO,
.
np
$s_{end=PE}\backslash np/np_{end=PE}$
$s\_1_{end=nil}\backslash np\_1\backslash(s\_1_{bal=+}\backslash np\_1)$
$np_{bal=+,end=comma}$
$sent\backslash_{\star}s_{end=nil}$
$<B_{\times}$
$s_{end=nil}\backslash np/np_{end=PE}$
>
$s_{end=nil}\backslash np$
<
$s_{end=nil}$
<
sent

Figure 3: Crossing composition

ensures that the *end* feature from the verb *loves* is also copied over. Thus, in example (19), the punctuation at the edge of the phrase would be percolated to the clausal root, where the sentence-final period would block the derivation. However, in the slash modality inheritance hierarchy proposed by Baldridge (2002), the $\times$ modality inherits the properties of function application. Consequently, this category could also lead to the erroneous derivation of example (11). In such a derivation, category (21) will not require the direct object to have a balanced appositive; meanwhile, the *end=nil* feature on the direct object will propagate to the clausal root, where it will happily combine with the category for the full stop. Finally, having two distinct categories for the adverb would offset the advantage of multi-modal categorial grammar in dealing with word order variation, where it is possible to use one category in situations where otherwise several categories would be required.

5.3 A rule-based filter to constrain overgeneration

For the reasons discussed in the preceding section, we decided not to use syntactic features to constrain overgeneration. Instead, we have employed semantic features in the logical form together with a rule-based filter, as an interim solution. During realization, the generated output is examined and fragments where two marks appear in a row are eliminated. Additionally, to handle improperly unbalanced punctuation, we modified the result categories of unbalanced appositive commas and dashes to include a feature marking unbalanced punctuation, as follows:

(22) $, \vdash np_{\langle 1\rangle unbal=comma}\backslash_{\star}np_{\langle 1\rangle}/_{\star}np_{\langle 2\rangle}$

Then, during realization, a filter on derivations looks for categories such as $np_{unbal=comma}$, and checks to make sure this NP is followed by a another punctuation mark in the string. We report on the effects of the filter in our results section.

6 Evaluation

We extracted a grammar from the restructured corpus and created testbeds of logical forms under the following conditions:

1. Baseline 1: A CCGbank version which has no lexicalized categories corresponding to any of the punctuation marks except sentence final marks and commas which conjoin elements or introduce NP appositives. Conjunction and apposition are frequent in the corpus and if excluded, logical forms for many sentences are not produced, weakening the baseline considerably.

2. Baseline 2: A CCGbank version where all punctuation marks (except conjunction/apposition commas and sentence-final marks, which have proper categories) have lexicalized MMCCG categories with no semantics, corresponding to binary rules in the original CCGbank.

3. The CCGbank augmented with punctuation categories.

Testing was done under four conditions:

1. Non-blind testing with oracle n-gram scoring. This condition tests the grammar most directly, as it avoids the issue of lexical smoothing and keeps the combinatorial search manageable. A grammar extracted from the development section (Section 00) of the CCG-bank was applied to the LF testbed of that section, using oracle n-gram scoring (along with FLMs, see next) to generate the sentences back. For each logical form, the generated output sentence was compared with the actual gold standard sentence corresponding to that logical form.

2. Blind testing with factored language models (FLM) and lexical smoothing, following (White et al., 2007). Blind testing naturally provides a more realistic test of performance on unseen data. Here logical forms of Sections 00 and 23 were created using grammars of those sections respectively and then a grammar was extracted from the standard training sections (02-21). This grammar was used to generate from the LFs of the development and test sections; for space reasons, we only report the results on the test section.

3. Blind testing with hypertagging. Hypertagging (Espinosa et al., 2008) is supertagging for surface realization; it improves realizer speed and coverage with large grammars by predicting lexical category assignments with a maximum entropy model.

4. The punctuation-enhanced grammars were tested in the three conditions above with and without the balanced punctuation filter.

7 Results

Non-blind testing results in Table 1 indicate that both exact match figures as well BLEU scores increase substantially in comparison to the baselines when a punctuation augmented grammar is used. The difference is especially notable when oracle n-gram scoring is used. The punctuation filter improves performance as exact matches increase by 1.66% and BLEU scores also show a slight increase. Complete realizations are slightly worse for the augmented grammar than Baseline 1, but the coverage of the baseline grammar is lower.

Table 1: Non-blind testing on Section 00 (Grammar coverage: Baseline 1, 95.8%; Baseline 2, 95.03%; Punct grammar, 98.0%)

N-grams	Grammar	Exact	Complete	BLEU
Oracle	Baseline 1	35.8%	86.2%	0.8613
	Baseline 2	39.10%	53.58%	0.8053
	Punct	75.9%	85.3%	0.9503
FLM w/o filter	Baseline 1	17.7%	83.0%	0.7293
	Baseline 2	5.72%	4.18%	0.4470
	Punct	29.7%	80.6%	0.7984
FLM w/ filt.	Punct	31.3%	80.6%	0.8062

Table 2: Blind testing on Section 23 with FLM (Grammar coverage: Baseline 1, 94.8%; Baseline 2, 95.06%; Punct grammar, 96.5%)

Hyp., Filt.	Grammar	Exact	Complete	BLEU
no, w/o	Baseline 1	11.1%	46.4%	0.6297
	Baseline 2	2.97%	3.97%	0.3104
	Punct	18.0%	43.2%	0.6815
no, w/	Punct	19.3%	43.3%	0.6868
yes, w/o	Punct	20.4%	61.5%	0.7270
yes, w/	Punct	21.6%	61.5%	0.7323

Blind testing results shown in Table 2 also demonstrate that the augmented grammar does better than the baseline in terms of BLEU scores and exact matches, with the hypertagger further boosting BLEU scores and the number of complete realizations. The use of the filter yields a further 1.2–1.3% increase in exact match figures as well as a half a BLEU point improvement; a planned collection of human judgments may reveal that these improvements are more meaningful than the scores would indicate.

Baseline 2, which models all punctuation, performs very badly with FLM scoring though it does better than the minimal punctuation Baseline 1 with oracle scoring. The main reason for this is that, without any semantic or syntactic features to constrain punctuation categories, they tend to reapply to their own output, clogging up the chart. This results in a low number of complete realizations as well as exact matches.

While direct comparisons cannot really be made across grammar frameworks, as inputs vary in their semantic depth and specificity, we observe that our all-sentences BLEU score of 0.7323 exceeds that of Hogan et al. (2007), who report a top score of 0.6882 including special treatment of multi-word units (though their coverage is near 100%). Nakanishi et al. (2005) and Langkilde-

Geary (2002) report scores several points higher, though the former is limited to sentences of length 20 or less, and the latter's coverage is much lower.

8 Conclusion

We have shown that incorporating a more precise analysis of punctuation into a broad-coverage reversible grammar extracted from the CCGbank yields substantial increases in the number of exact matches and BLEU scores when performing surface realization with OpenCCG, contributing to state-of-the-art results. Our discussion has also highlighted the inadequacy of using syntactic features to control punctuation placement in CCG, leading us to develop a filter to ensure appropriately balanced commas and dashes. In future work, we plan to investigate a more satisfactory grammatical treatment involving constraints in independent orthographic derivations, perhaps along the lines of the autonomous prosodic derivations which Steedman and Prevost (1994) discuss. An evaluation of parsing side performance is also planned.

Acknowledgments

We thank the anonymous reviewers, Detmar Meurers and the Clippers and Synners groups at OSU for helpful comments and discussion.

References

Baldridge, Jason and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proc. ACL-02*.

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Bayraktar, Murat, Bilge Say, and Varol Akman. 1998. An Analysis of English Punctuation: The Special Case of Comma. *International Journal of Corpus Linguistics*, 3(1):33–58.

Boxwell, Stephen and Michael White. 2008. Projecting Propbank roles onto the CCGbank. In *Proc. LREC-08*. To appear.

Briscoe, Ted. 1994. Parsing (with) punctuation. Technical report, Xerox, Grenoble, France.

Doran, Christine. 1998. *Incorporating Punctuation into the Sentence Grammar: A Lexicalized Tree Adjoining Grammar Perspective*. Ph.D. thesis, University of Pennsylvania.

Espinosa, Dominic, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proc. ACL-08:HLT*. To appear.

Forst, Martin and Ronald M. Kaplan. 2006. The importance of precise tokenizing for deep grammars. In *Proc. LREC-06*.

Hockenmaier, Julia and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Hogan, Deirdre, Conor Cafferkey, Aoife Cahill, and Josef van Genabith. 2007. Exploiting multi-word units in history-based probabilistic generation. In *Proc. EMNLP-CoNLL-07*.

Langkilde-Geary, Irene. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proc. INLG-02*.

Nakanishi, Hiroko, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic methods for disambiguation of an HPSG-based chart generator. In *Proc. IWPT-05*.

Nunberg, Geoffrey. 1990. *The Linguistics of Punctuation*. CSLI Publications, Stanford, CA.

Steedman, Mark and S. Prevost. 1994. Specifying intonation from context for speech synthesis. *Speech Communication*, 15(1–2):139–153.

Steedman, Mark. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

White, Michael, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with CCG. In *Proc. of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*.

White, Michael. 1995. Presenting punctuation. In *Proceedings of the Fifth European Workshop on Natural Language Generation*, pages 107–125.

White, Michael. 2006. Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation*, 4(1):39–75.

Multilingual Grammar Resources in Multilingual Application Development

Marianne Santaholma
Geneva University, ETI/TIM/ISSCO
40, bvd du Pont-d'Arve
1211 Geneva 4, Switzerland
Marianne.Santaholma@eti.unige.ch

Abstract

Grammar development makes up a large part of the multilingual rule-based application development cycle. One way to decrease the required grammar development efforts is to base the systems on multilingual grammar resources. This paper presents a detailed description of a parametrization mechanism used for building multilingual grammar rules. We show how these rules, which had originally been designed and developed for typologically different languages (English, Japanese and Finnish) are applied to a new language (Greek). The developed shared grammar system has been implemented for a domain specific speech-to-speech translation application. A majority of these rules (54%) are shared amongst the four languages, 75% of the rules are shared for at least two languages. The main benefit of the described approach is shorter development cycles for new system languages.

1 Introduction

Most of grammar based applications are built on monolingual grammars. However, it is not unusual that the same application is deployed for more than one language. For these types of systems the monolingual grammar approach is clearly not the best choice, since similar grammar rules are written several times, which increases overall development efforts and makes maintenance laborious.

One way to decrease these efforts is to share already developed linguistic resources between system languages. Common approaches for sharing information include grammar adaptation and grammar sharing. Grammar adaptation is the technique of modifying an already existing grammar to cover a new language as implemented among others by Alshawi et al. 1992; Kim et al. 2003; and Santaholma, 2005.

In grammar sharing, existing grammar rules are directly shared between languages rather than just being recycled as they are in grammar adaptation. Compared to both the monolingual grammar approach and the grammar adaptation approach, grammar sharing reduces the amount of code that needs to be written as the central rules are written only once. This automatically leads to coherence between language descriptions for different languages, which improves grammar maintainability, and eliminates the duplication effort that otherwise occurs when monolingual grammars are used.

Multilingual grammars can share resources between languages in various ways. Ranta (2007) has developed an abstract syntax that defines a common semantic representation in a multilingual grammar.

Another type of approach is implemented in the LinGO Grammar Matrix project (Bender et al. 2005; Bender, 2007). The Grammar Matrix consists of a core grammar that contains the types and constraints that are regarded as cross-linguistically useful. This core is further linked to phenomenon-specific libraries. These consist of rule repertories based on typological categories. The necessary modules are put together like building blocks according to language characteristics to form the final grammar of a language.

The work described in this paper implements a grammar sharing approach that is based on language-independent parameterized rules that are complemented with necessary language-specific rules and features. These shared rules have been implemented for MedSLT, a multilingual spoken language translation system (Bouillon et al., 2005). All of the central language processing components of MedSLT, including the speech recognizer, parser and generator, are derived from handcrafted general grammars of a language. The biggest effort in adding a new language to the existing spoken language translation framework is the grammar development cycle. As more languages are added to the existing spoken language translation framework, the necessity for multilingual grammar rules grows.

(Bouillon et al., 2006) first developed shared MedSLT grammar rules for the Romance languages French, Spanish and Catalan. Compared to the monolingual grammar system, the shared grammar-based system facilitated application development without degrading the performance of any of its components (speech recognition, translation) on these languages.

We took this approach further and implemented parameterized grammar rules for typologically different languages - English, Finnish and Japanese. Experiments have shown that these shared rules perform equally well on all three languages (Santaholma, 2007). As these grammars had been developed in parallel, it was not clear how flexible the parameterized grammar approach would be for new a language, which was not included in the original development process. We thus extended the grammar to cover Modern Greek as a new language. The paper describes the methodology of adding this new language and evaluates the parametrization mechanism.

The rest of the paper is structured as follows. Section 2 describes the Regulus toolkit (Rayner et al., 2006) and MedSLT, which form the development environment and application framework on which this work is based. Section 3 describes the parameterized multilingual grammar and parametrization mechanism. Section 4 summarizes techniques used to adding Modern Greek to the shared grammar system. Section 5 concludes.

2 Regulus Development environment and MedSLT application

2.1 Regulus features

The Regulus grammar framework has been designed for spoken language grammars, and thus differs from popular grammar frameworks like Lexical Functional Grammar (LFG) (Bresnan and Kaplan, 1985) and Head-driven Phrase Structure Grammar (HSPG) (Pollard and Sag, 1994). Regulus grammars are written with a feature-grammar formalism that can easily be compiled into context free grammars (CFG). These are required for compilation of grammar-based language models used in speech recognition. Characteristic for Regulus grammars are finite valued features and exclusion of complex feature-value structures. Consequently the resulting rule-sets are perhaps more repetitive and less elegant than the equivalent LFGs or HPSGs. This design, however, enables compilation of non-trivial feature-grammars to CFG.

Another Regulus feature that enables CFG compilation is grammar specialization that reduces the extent of the grammar. Grammar specialization is performed by explanation-based learning (EBL) [1]. Multilingual grammar development can profit from grammar specialization in various ways. The general grammar of a language can be specialized to specific domains based on domain specific information[2]. Thus the specialization serves as a way to limit the ambiguities typical for general grammars. Furthermore, the procedure is used to specialize the grammars for different tasks. Ideally a grammar should recognize variant forms but generate only one. This variation can be controlled by specializing the Regulus grammars for these tasks. Finally the multilingual Regulus grammar can be specialized for specific languages by automatically removing the unnecessary rules.

2.2 MedSLT

Most large-scale machine translation systems are currently based on statistical language processing. MedSLT, however, has been implemented with linguistically motivated grammars mainly for the following reasons: (1) necessary data for inducing the grammars and training the statistical language

[1]The method is described in detail in (Rayner et al., 2006), Chapter 10.

[2]These include domain specific corpus, lexica and operationality criteria that control the granularity of specialized grammar. Details provided by (Rayner et al., 2006).

models were not available for the required domain and languages. (2) the medical domain demands accurate translation performance, which can be more easily guaranteed with rule based systems.

MedSLT is an unidirectional[3] speech-to-speech translation system that has been designed to help in emergency situations where a common language between the physician and the patient does not exist. In addition to influencing the system architecture, this communication goal also significantly influences system coverage, and consequently the grammars. The typical utterances MedSLT translates consist of physician's questions about the intensity, location, duration and quality of pain, factors that increase and decrease the pain, therapeutic processes and the family history of the patient. These include yes-no questions like "Does it hurt in the morning?", "Is the pain stubbing?" and "Do you have fever when you have the headaches?". Other frequent type of questions include wh-questions followed by elliptical utterance, like "Where is the pain?", "In the front of the head?", "On both sides of the head?". Currently MedSLT translates between Arabic, Catalan, English, Finnish, French, Japanese, and Spanish. The translation is interlingua based.

The following sections describe the implementation of the shared parameterized grammar rules for this specific application using the Regulus platform.

3 Parameterized grammar rules

The parameterized grammar rules assemble the common foundations of linguistic phenomena in different languages. The framework for the language-independent rules presented here was developed and tested with English, Japanese and Finnish. These languages represent different types of languages and hence express the same linguistic phenomena in different ways. Consequently they provided a good starting point for framework design.

The Regulus multilingual grammar is modular and organized hierarchically. Parameterized rules are stored in the "language-independent core" module. This is the most generic level and as such is shared between all languages. The "lower levels" include the language-family specific modules and the language-specific modules. The modules for related languages decrease redundancy as related languages commonly share characteristics at least to some extent. [4] The information in this modular structure is inherited top-down from the most generic to language specific.

The first language to which we applied the parameterized rules and which had not been part of the original shared grammar framework development is Modern Greek.

In the following we first describe the parameterized grammar rules. Then we focus on how these rules are applied for Greek.

3.1 Coverage

The parameterized grammar currently covers basic linguistic phenomena by focusing on the structures required to process MedSLT system coverage. The current coverage is summarized in Table 1.

Phenomena	Construction
Sentence types	declarative, yn-question, wh-question, ellipsis subordinate when clause
Tense	present, past(imperfect)
Voice	active, passive
Aspect	continuous, present perfect, past perfect
Verb subcategorization	transitive, intransitive, predicative (be+adj), existential (there+be+np)
Determiners	article, number, quantifier
Adpositional modifiers	prepositional, postpositional
Adverbial modifiers	verb and sentence modifying adverbs, comparison adverbs
Pronouns	personal, possessive, dummy pronouns
Adjective modifiers	predicative, attributive, comparison

Table 1: Linguistic phenomena covered by the shared grammar.

The general difficulty of spoken language for grammar development is frequent ungrammatical

[3]Bidirectional MedSLT exists currently for English-Spanish language pair. Details are provided in (Bouillon et al., 2007).

[4]However, as identical constructions and features also exist in unrelated languages the advantage of language family modules is finally not so significant.

and non-standard use of language. This includes for example incorrect use of case inflections in Finnish and missing particles in spoken Japanese.

3.2 Parametrization - abstracting away from language specific details

The parametrization aims to generalize the cross-linguistic variation in grammar rules. English yes-no questions require an auxiliary and inverted word order, in Finnish yes-no questions the subject-verb inversion is combined with a certain form of the main verb; in Finnish noun heads and the modifying adjective agree in case and number, in Greek they additionally agree in gender, and so forth. The way of expressing the same linguistic phenomena or constructions varies from one language to another. Hence, shared grammar rules need to abstract away from these kinds of details.

The multilingual Regulus rules are parameterized using macro declarations. Macros are a standard tool in many development environments. In Regulus grammars they are extensively used to catch generalizations in the rules, and in particular in lexica. In multilingual grammar rules the macros serve as “links” towards language-specific information.

The shared rules have a language-neutral surface representation where macros invoke the required language-specific information. The macro reference of a language-independent rule is replaced by the information contained in the macro definition. The context of the macro reference determines how the macro definition combines with other parts of the description. The mechanism is similar to LFG ‘templates’, which encode linguistic generalizations in a language description (Dalrymple et al., 2004).

The macro mechanism itself is rather simple. The crucial is that the macros are defined in a transparent and coherent way. Otherwise the grammar developer will spend more time learning to how to use the parameterized rule set than she would spend to develop a new grammar from scratch. When the macros are well defined, sharing the rules for a new language is just a matter of defining the language-specific macro definitions.

In the following we present some concrete examples of how cross-linguistic variation can be parameterized in a multilingual Regulus grammar using macros.

3.2.1 Parameterizing features

The following example shows how we parameterize the previously mentioned agreement features required in different languages. In Regulus grammars, like in other constraint-based grammars, this fine-grained information is encoded in feature-value pairs. We encode a basic declarative sentence rule (`s`) that consists of a noun phrase (`np`) and a verb phrase (`vp`):

```
s:[sem=concat(Np, Vp)] -->
  np:[sem=Np, sem_np_type=T,
  @noun_head_features(Head)],
  vp:[sem=Vp, subj_sem_np_type=T,
  @verb_head_features(Head)].
```

In Finnish sentences the subject and the main verb agree in person and number. Japanese doesn't make use of these agreement features in this context. Consequently, the common rules have to express the agreement in a parameterized way. For this reason in the `np` we introduce a macro called `noun_head_features(Head)` and in the `vp` the macro `verb_head_features(Head)`. [5] These macro declarations unify but don't say anything explicit about the unifying features themselves at this common level. The macros thus “neutralize” the language-specific variation and only point further down to language-specific information.

In Finnish, the `noun_head_features` and `verb_head_features` macros invoke the language specific features ‘number’ and ‘person’:

```
macro
(noun_head_features([P, N]),
[person=P, number=N]).

macro
(verb_head_features(([P, N]),
[person=P, number=N]).
```

The macro references are replaced by these features in the final Finnish declarative sentence rule which takes the form:

```
s:[sem=concat(Np, Vp)] -->
  np:[sem=Np, sem_np_type=T,
  person=P, number=N],
  vp:[sem=Vp, subj_sem_np_type=T,
  person=P, number=N].
```

[5]The Regulus macro declaration is preceded by '@'.

As Japanese does not apply either 'number' or 'person' features the macro definition consists of an empty value:

```
macro(noun_head_features([]),
[]).
```

The final Japanese sentence rule takes after the macro replacement the form:

```
s:[sem=concat(Np, Vp)] -->
   np:[sem=Np, sem_np_type=T],
   vp:[sem=Vp, subj_sem_np_type=T].
```

Similarly we can parameterize the value of a specific feature. A `vp` could include a `verb_form` feature that in English could take as its value "gerundive" and in Finnish "infinite" in that particular context. We can parameterize the `vp` rule with a macro `vform` which invokes the language-specific macro definition and replaces it with the corresponding language-specific feature-value pairs:

```
vp:[sem=concat(Aux, Vp)] -->
  aux:[sem=Aux,@aux_features(Head)],
  vp:[sem=Vp,
   @vform(Vform),
   @verb_head_features(Head)].
```

The English macro definition would be:

```
macro(vform(Vform),
[verb_form=gerund,
 verb_form=Vform]).
```

The Finnish equivalent:

```
macro(vform(Vform),
[verb_form=finite,
 verb_form=Vform]).
```

Macros can furthermore refer to other macro definitions and in this way represent inclusion relations between different features. This forms a multilevel macro hierarchy. The macro `noun_head_features(Head)` included in `np` rule (1) could contain a macro `arg` (2), that would further be defined by (3):

```
1)
np:[sem=Np, sem_np_type=SemType,
   @noun_head_features(Head)].

2)
macro(noun_head_features([Agr,Case]),
   [@agr(Agr), case=Case]).

3)
macro(agr([Case, Number]),
   [case=Case, number=Number]).
```

3.2.2 Parameterizing the constituent order

The constituent order is defined by concatenation of linguistic categories in the wanted order (`vp:[sem=concat(Verb, NP)]`). This order can, similarly to features, also be parameterized by using macros. We show here as an example of how the order of a transitive main verb (`verb`) and direct object (`np`) is parameterized in a verb phrase:

```
vp:[sem=concat(Verb, NP)] -->
verb:[sem=Verb, subcat=transitive,
      obj_sem_np_type=ObjType],
np:[sem=NP, sem_np_type=ObjType]).
```

In English the direct object follows the verb, whereas in Japanese it precedes the verb. The order of these constituents can be parameterized by introducing into the rule a macro that in the example rule is represented by '`verb_transitive_np`':

```
vp:[sem=concat(Verb, NP)] -->
@verb_transitive_np(
verb:[sem=Verb, subcat=transitive,
       obj_sem_np_type=ObjType],
np:[sem=NP, sem_np_type=ObjType]).
```

This macro invokes the language-specific rules that define the order of the semantic values of categories required in a specific language. The semantic value of the category verb is `sem=Verb` and of noun `sem=Noun`. Consequently the English-specific macro definition would be:

```
macro(verb_transitive_np
(Verb, Noun),(Verb, Noun)).
```

This rule specifies that when there is a semantic value '`Verb`' followed by a semantic value '`Noun`' these should be processed in the order '`Verb`', '`Noun`'. The order of constituents remains unchanged.

The equivalent Japanese macro definition would be:

```
macro(verb_transitive_np
(Verb, Noun),(Noun, Verb)).
```

Contrary to the English rule this rule specifies that when there is a semantic value '`Verb`' followed by a semantic value '`Noun`' these should be processed in the order '`Noun`', '`Verb`'. This changes the order the of constituents. Details of Regulus semantic processing are available in Rayner et al., 2006.

3.2.3 Ignoring rules/features and using empty values

There exist several ways to ignore rules and features or to introduce empty values in Regulus grammars. These have proven practical in rule parametrization. In the following we present some frequent examples.

Features that are irrelevant for a particular language (in a particular context) can take 'empty' (`[]`) as their value. This can be encoded in several ways.

- Macro takes an empty value. This is encoded by '`[]`'

  ```
  Example:
  macro(noun_head_features([]),
  []).
  ```

- Feature takes an empty value. This is encoded by '_':

  ```
  Example:
  macro(premod_case(Case),
  [case=_]).
  ```

Rules that are applied to only one language are organized in the language-specific modules. However most of the rules are necessary for two or more languages. The rules that are used for groups of specific languages can be 'tagged' using macro declarations. For example a rule or feature that is valid for English and Japanese could be simply tagged with an identifier macro 'eng_jap':

```
@eng_jap
('rule_body_here').
```

The English and Japanese rules would call the rule body by macro definition:

```
macro(eng_jap(Body), (Body).
```

The Finnish language-specific macro definition would call an empty category that we call here 'dummy_cat' and the rule would be ignored:

```
macro(eng_jap(Body),
(dummy_cat:[] --> dummy)).
```

Specialization of a grammar for a specific language and into domain-specific form checks which rules are necessary for processing the domain specific-coverage in that particular language. Consequently empty features of the general grammar are automatically ignored and the language processing remains efficient.

4 Processing Modern Greek with shared parameterized grammar rules

Cross-linguistic comparison shows that the Greek that belongs to the Indo-European language family does not only share some features with English but also with Japanese and Finnish. Common with English is, for example, the use of prepositions and articles, and with Finnish and Japanese the pro-drop.

The development of Greek grammar coverage equivalent to those of English, Japanese and Finnish coverage in MedSLT took about two weeks. For most part only the language-specific macro definitions needed to be specified. Five new rules were developed from scratch. The most significant part of the development consisted of building the Greek lexicon and verifying that the analyses produced by the shared grammar rules were correct.

In the following we summarize Greek-specific rules, features and macros.

4.1 Greek rules and features

In general, Greek word order is flexible, especially in spoken language. All permutations of ordering of subject, object, and verb can be found, though the language shows a preference for Subject-Verb-Object ordering in neutral contexts. New parametrized *constituent orders* were the most significant additions to the multilingual grammar. These are listed below.

1. Yes-no questions, which are a central part of the MedSLT application's coverage, can be expressed by both direct and indirect constituent order in Greek. As these are both common in spoken language, the Japanese question rule (direct constituent order + question particle 'ka') was parameterized for Greek.

2. The order of possessive pronoun and head noun required parametrization. Until now the shared grammar contained only the order where a head noun is preceded by the possessive. In Greek the opposite order is used, with the possessive following the head noun. The existing rule was parameterized by a new macro.

3. Similar parameterization was performed for verb phrases including an indirect object. The Greek constituent order is reversed relative to English order. That is, the pronoun goes before the verb. A new macro was introduced to parameterize the rule.

One main area of difference compared to English/Finnish/Japanese, is in the placement of weak pronouns, generally referred to as 'clitics'. Their position in Greek is relative to the verb. In standard language they are placed before finite verbs and after infinite verbs. Thus these weak pronouns can occur in sentence-initial position. New rules were developed to process these clitics as well as the Greek genitive post-modifier structure.

Greek could mainly use the existing grammar *features*. The difference, compared to the original three languages, was in the extensive use of the 'gender' feature (possible values: feminine, masculine and neuter). For example, Greek articles agree with the head noun in gender, number, and case. Furthermore, prepositions agree with the following nouns in gender, number and case.

4.2 Summary of multilingual rules

Table 2 summarizes current use of the multilingual rules. The grammar includes a total of 80 rules for English, Finnish, Japanese and Greek. 54% of the rules are shared between all four languages and 75% of the rules are shared between two or more languages. Not everything can be parameterized, and some language-specific rules are necessary. The language-specific rules cover 25% of all rules.

Languages	N. of rules	% of total
Eng + Fin + Jap + Gre	43	54%
Eng + Fin + Jap	0	
Eng + Fin + Gre	4	
Eng + Jap + Gre	0	
Fin + Jap + Gre	6	
TOTAL	10	12.5%
Fin + Jap	3	
Eng + Fin	1	
Eng + Jap	1	
Gre + Eng	1	
Gre + Jap	1	
Gre + Fin	0	
TOTAL	7	8.75%
Eng	9	
Fin	0	
Jap	6	
Gre	5	
TOTAL	20	25%
TOTAL	80	100%

Table 2: Grammar rules in total

5 Conclusions

We have described a shared grammar approach for multilingual application development. The described approach is based on parametrization of Regulus grammar rules using macros. We have shown that these parameterized rules can with comparably little effort be used for a new system language in a multilingual limited-domain application. A majority of rules were shared between all implemented languages and 75% of rules by at least two languages. The deployment of a new language was mainly based on already existing rules.

The shared grammar approach promotes consistency across all system languages, effectively increasing maintainability.

Acknowledgement

I would like to thank Pierrette Bouillon and Manny Rayner for their advise, and Agnes Lisowska and Nikos Chatzichrisafis for their suggestions and English corrections.

References

Bender, Emily and Dan Flickinger. 2005. *Rapid Prototyping of Scalable Grammars: Towards Modularity in Extensions to a Language-Independent Core*. In: Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos), Jeju Island, Korea.

Bender, Emily. 2007. *Combining Research and Pedagogy in the Development of a Crosslinguistic Grammar Resource*. In: Proceedings of the workshop Grammar Engineering across Frameworks 2007, Stanford University.

Bouillon, Pierrette, Manny Rayner, Nikos Chatzichrisafis, Beth Ann Hockey, Marianne Santaholma, Marianne Starlander, Yukie Nakao, Kyoko Kanzaki, Hitoshi Isahara. 2005. *A generic multilingual open source platform for limited-domain medical speech translation*. In: Proceedings of the 10th Conference of the European Association for Machine Translation, EAMT, Budapest, Hungary.

Bouillon, Pierrette, Manny Rayner, Bruna Novellas Vall, Marianne Starlander, Marianne Santaholma, Nikos Chatzichrisafis. 2007. Une grammaire partage multi-tache pour le traitement de la parole : application aux langues romanes. *TAL (Traitement Automatique des Langues), Volume 47, 2006/3*.

Bouillon, Pierrette, Glenn Flores, Marianne Starlander, Nikos Chatzichrisafis, Marianne Santaholma, Nikos Tsourakis, Manny Rayner, Beth Ann Hockey. 2007. *A Bidirectional Grammar-Based Medical Speech Translator*. In: Proceedings of workshop on Grammar-based approaches to spoken language processing, ACL 2007, June 29, Prague, Czech Republic.

Bresnan, Joan and Ronald Kaplan. 1985. *The mental representation of grammatical relations*. MIT Press, Cambridge, MA.

Butt, Miriam , Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. *The Parallel Grammar Project*. In: Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation.

Dalrymple, Mary, Ron Kaplan, and Tracy Holloway King. 2004. *Linguistics Generalizations over Descriptions*. In M. Butt and T.H. King (ed.) Proceedings of the LFG04 Conference.

Kim, Roger, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, Hiroshi Masuichi, Tomoko Ohkuma. 2003. *Language Multilingual Grammar Development via Grammar Porting*. In: Proceedings of the ESSLLI Workshop on Ideas and Strategies for Multilingual Grammar Development, Vienna, Austria.

Pollard, Carl and Ivan Sag. 1994. *Head Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Ranta, Aarne. 2007. Modular Grammar Engineering in GF. *Research on Language and Computation, Volume 5, 2/2007, 133–158*.

Rayner, Manny, Beth Ann Hockey, Pierrette Bouillon. 2006. *Regulus-Putting linguistics into speech recognition*. CSLI publications, California, USA.

Santaholma, Marianne. 2005. *Linguistic representation of Finnish in a limited domain speech-to-speech translation system*. In: Proceedings of the 10th Conference on European Association of Machine Translation, Budapest, Hungary.

Santaholma, Marianne. 2007. *Grammar sharing techniques for rule-based multilingual NLP systems*. In: Proceedings of NODALIDA 2007, the 16th Nordic Conference of Computational Linguistics, Tartu, Estonia.

Speeding up LFG Parsing Using C-Structure Pruning

Aoife Cahill‡ John T. Maxwell III† Paul Meurer§ Christian Rohrer‡ Victoria Rosén¶

‡IMS, University of Stuttgart, Germany, {cahillae, rohrer}@ims.uni-stuttgart.de
†Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, maxwell@parc.com
§Unifob Aksis, Bergen, Norway, paul.meurer@aksis.uib.no
¶Unifob Aksis and University of Bergen, Norway, victoria@uib.no

Abstract

In this paper we present a method for greatly reducing parse times in LFG parsing, while at the same time maintaining parse accuracy. We evaluate the methodology on data from English, German and Norwegian and show that the same patterns hold across languages. We achieve a speedup of 67% on the English data and 49% on the German data. On a small amount of data for Norwegian, we achieve a speedup of 40%, although with more training data we expect this figure to increase.

1 Introduction

Efficient parsing of large amounts of natural language is extremely important for any real-world application. The XLE Parsing System is a large-scale, hand-crafted, deep, unification-based system that processes raw text and produces both constituent structures (phrase structure trees) and feature structures (dependency attribute-value matrices). A typical breakdown of parsing time of XLE components with the English grammar is Morphology (1.6%), Chart (5.8%) and Unifier (92.6%). It is clear that the major bottleneck in processing is in unification. Cahill et al. (2007) carried out a preliminary experiment to test the theory that if fewer c-structures were passed to the unifier, overall parsing times would improve, while the accuracy of parsing would remain stable. Their experiments used state-of-the-art probabilistic treebank-based parsers to automatically mark certain constituents on the input sentences, limiting the number of c-structures the XLE parsing system would build. They achieved an 18% speedup in parse times, while maintaining the accuracy of the output f-structures. The experiments presented in Cahill et al. (2007) used the XLE system as a black box and did not make any changes to it. However, the results were encouraging enough for a c-structure pruning mechanism to be fully integrated into the XLE system.

The paper is structured as follows: we present the pruning model that has been integrated into the XLE system (Section 2), and how it can be applied successfully to more than one language. We present experiments for English (Section 3), German (Section 4) and Norwegian (Section 5) showing that for both German and English, a significant improvement in speed is achieved, while the quality of the f-structures remains stable. For Norwegian a speedup is also achieved, but more training data is required to sustain the accuracy of the f-structures. In Section 7 we present an error analysis on the German data. We then relate the work presented in this paper to similar efficient parsing strategies (Section 8) before concluding in Section 9.

2 XLE and the C-Structure Pruning Mechanism

The XLE system is designed to deal with large amounts of data in a robust manner. There are several mechanisms which facilitate this, including fragmenting and skimming. Fragmenting is called when the grammar is unable to provide a complete parse for the input sentence, and a fragment analysis of largest possible chunks is built. Skimming is called when too much time or memory has been used by XLE. Any constituents that have not been fully processed are "skimmed", which means that the amount of work carried out in processing the constituent is limited. This guarantees that XLE will finish processing the sentence in polynomial time.

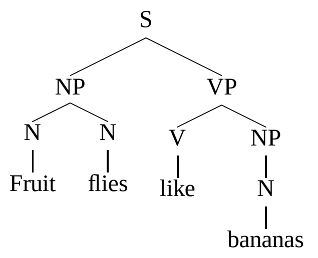
XLE uses a chart-based mechanism for building parses, and has been complemented with a c-structure pruning mechanism to speed up parsing time. During pruning, subtrees at a particular cell in the chart are pruned if their probabilities are not higher than a certain threshold. The chart pruner uses a simple stochastic CFG model. The probability of a tree is the product of the probabilities of each of the rules used to form the tree, including the rules that lead to lexical items (such as N → dog). The probability of a rule is basically the number of times that that particular form of the rule occurs in the training data divided by the number of times the rule's category occurs in the training data, plus a smoothing term. This is similar to the pruning described in Charniak and Johnson (2005) where edges in a coarse-grained parse forest are pruned to allow full evaluation with fine-grained categories.

The pruner prunes at the level of individual constituents in the chart. It calculates the probabilities of each of the subtrees of a constituent and compares them. The probability of each subtree is compared with the best subtree probability for that constituent. If a subtree's probability is lower than the best probability by a given factor, then the subtree is pruned. In practice, the threshold is the natural logarithm of the factor used. So a value of 5 means that a subtree will be pruned if its probability is about a factor of 150 less than the best probability.

If two different subtrees have different numbers of morphemes under them, then the probability model is biased towards the subtree that has fewer morphemes (since there are fewer probabilities multiplied together). XLE counteracts this by normalizing the probabilities based on the difference in length.

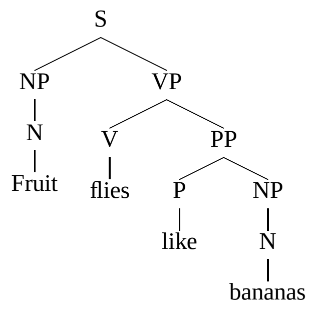
To illustrate how this works, we give the following example. The string *Fruit flies like bananas* has two different analyses. Figures 1 and 2 give their analyses along with hypothetical probabilities for each rule.

These two analyses come together at the S constituent that spans the whole sentence. The probability of the first analysis is 8.4375E-14. The probability of the second analysis is 4.21875E-12. This means that the probability of the second analysis is 50 times higher than the probability of the first analysis. If the threshold is less than the natural logarithm of 50 (about 3.9), then the subtree of the first analysis will be pruned from the S constituent.



S	→	NP VP	0.5000
NP	→	N N	0.1500
N	→	Fruit	0.0010
N	→	flies	0.0015
VP	→	V NP	0.2000
V	→	like	0.0050
NP	→	N	0.5000
N	→	bananas	0.0015
			8.4375E-14

Figure 1: Analysis (1) for the string *Fruit flies like bananas* with hypothetical probabilities



S	→	NP VP	0.5000
NP	→	N	0.5000
N	→	Fruit	0.0010
V	→	flies	0.0025
VP	→	V PP	0.1000
P	→	like	0.0500
PP	→	P NP	0.9000
NP	→	bananas	0.0015
			4.21875E-12

Figure 2: Analysis (2) for the string *Fruit flies like bananas* with hypothetical probabilities

3 Experiments on English

We carried out a number of parsing experiments to test the effect of c-structure pruning, both in terms of time and accuracy. We trained the c-structure pruning algorithm on the standard sections of Penn Treebank Wall Street Journal Text (Marcus et al., 1994). The training data consists of the original WSJ strings, marked up with some of the Penn

Treebank constituent information. We marked up NPs and SBARs as well as adjective and verbal POS categories. This is meant to guide the training process, so that it does learn from parses that are not compatible with the original treebank analysis. We evaluated against the PARC 700 Dependency Bank (King et al., 2003), splitting it into 140 sentences as development data and the remaining unseen 560 for final testing (as in Kaplan et al. (2004)). We experimented with different values of the pruning cutoff on the development set; the results are given in Table 1.

The results show that the lower the cutoff value, the quicker the sentences can be parsed. Using a cutoff of 4, the development sentences can be parsed in 100 CPU seconds, while with a cutoff of 10, the same experiment takes 182 seconds. With no cutoff, the experiment takes 288 CPU seconds. However, this increase in speed comes at a price. The number of fragment parses increases, i.e. there are more sentences that fail to be analyzed with a complete spanning parse. With no pruning, the number of fragment parses is 23, while with the most aggressive pruning factor of 4, there are 39 fragment parses. There are also many more skimmed sentences with no c-structure pruning, which impacts negatively on the results. The oracle f-score with no pruning is 83.07, but with pruning (at all thresholds) the oracle f-score is higher. This is due to less skimming when pruning is activated, since the more subtrees that are pruned, the less likely the XLE system is to run over the time or memory limits needed to trigger skimming.

Having established that a cutoff of 5 performs best on the development data, we carried out the final evaluation on the 560-sentence test set using this cutoff. The results are given in Table 2. There is a 67% speedup in parsing the 560 sentences, and the most probable f-score increases significantly from 79.93 to 82.83. The oracle f-score also increases, while there is a decrease in the random f-score. This shows that we are throwing away good solutions during pruning, but that overall the results improve. Part of this again is due to the fact that with no pruning, skimming is triggered much more often. With a pruning factor of 5, there are no skimmed sentences. There is also one sentence that timed out with no pruning, which also lowers the most probable and oracle f-scores.

Pruning Level	None	5
Total Time	1204	392
Most Probable F-Score	79.93	82.83
Oracle F-Score	84.75	87.79
Random F-Score	75.47	74.31
# Fragment Parses	96	91
# Time Outs	1	0
# Skimmed Sents	33	0

Table 2: Results of c-structure pruning experiments on English test data

4 Experiments on German

We carried out a similar set of experiments on German data to test whether the methodology described above ported to a language other than English. In the case of German, the typical time of XLE components is: Morphology (22.5%), Chart (3.5%) and Unifier (74%). As training data we used the TIGER corpus (Brants et al., 2002). Setting aside 2000 sentences for development and testing, we used the remaining 48,474 sentences as training data. In order to create the partially bracketed input required for training, we converted the original TIGER graphs into Penn-style trees with empty nodes and retained bracketed constituents of the type NP, S, PN and AP. The training data was parsed by the German ParGram LFG (Rohrer and Forst, 2006). This resulted in 25,677 full parses, 21,279 fragmented parses and 1,518 parse failures.[1] There are 52,959 features in the final pruning model.

To establish the optimal pruning settings for German, we split the 2,000 saved sentences into 371 development sentences and 1495 test sentences for final evaluation. We evaluated against the TiGer Dependency Bank (Forst et al., 2004) (TiGerDB), a dependency-based gold standard for German parsers that encodes grammatical relations similar to, though more fine-grained than, the ones in the TIGER Treebank as well as morphosyntactic features. We experimented with the same pruning levels as in the English experiments. The results are given in Table 3.

The results on the development set show a similar trend to the English results. A cutoff of 4 results in the fastest system, however at the expense

[1]The reason there are more fragment parses than, for example, the results reported in Rohrer and Forst (2006) is that the bracketed input constrains the parser to only return parses compatible with the bracketed input. If there is no solution compatible with the brackets, then a fragment parse is returned.

Pruning Level	None	4	5	6	7	8	9	10
Oracle F-Score	83.07	84.50	85.47	**85.75**	85.57	85.57	85.02	84.10
Time (CPU seconds)	288	**100**	109	123	132	151	156	182
# Time Outs	0	0		0	0	0	0	0
# Fragments	**23**	39	36	31	29	27	27	24
# Skimmed Sents	8	0	0	1	1	1	1	3

Table 1: Results of c-structure pruning experiments on English development data

Pruning Level	None	4	5	6	7	8	9	10
Oracle F-Score	83.69	83.45	**84.02**	82.86	82.82	82.95	83.03	82.81
Time (CPU seconds)	1313	**331**	465	871	962	1151	1168	1163
# Time Outs	6	**0**	**0**	5	5	5	5	6
# Fragments	**65**	104	93	81	74	73	73	68

Table 3: Results of c-structure pruning experiments on German development data

Pruning Level	None	5
Total Time	3300	1655
Most Probable F-Score	82.63	82.73
Oracle F-Score	84.96	84.79
Random F-Score	73.58	73.72
# Fragment Parses	324	381
# Time Outs	2	2

Table 4: Results of c-structure pruning experiments on German test data

of accuracy. A cutoff of 5 seems to provide the best tradeoff between time and accuracy. Again, most of the gain in oracle f-score is due to fewer timeouts, rather than improved f-structures. In the German development set, a cutoff of 5 leads to a speedup of over 64% and a small increase in oracle f-score of 0.33 points. Therefore, for the final evaluation on the unseen test-set, we choose a cutoff of 5. The results are given in Table 4. We achieve a speedup of 49% and a non-significant increase in most probable f-score of 0.094. The time spent by the system on morphology is much higher for German than for English. If we only take the unification stage of the process into account, the German experiments show a speedup of 65.5%.

5 Experiments on Norwegian

As there is no treebank currently available for Norwegian, we were unable to train the c-structure pruning mechanism for Norwegian in the same way as was done for English and German. There is, however, some LFG-parsed data that has been completely disambiguated using the techniques described in Rosén et al. (2006). In total there are 937 sentences from various text genres including Norwegian hiking guides, Sophie's World and the Norwegian Wikipedia. We also use this disambiguated data as a gold standard for evaluation. The typical time of XLE components with the Norwegian grammar is: Morphology (1.6%), Chart (11.2%) and Unifier (87.2%).

From the disambiguated text, we can automatically extract partially bracketed sentences as input to the c-structure pruning training method. We can also extract sentences for training that are partially disambiguated, but these cannot be used as part of the test data. To do this, we extract the bracketed string for each solution. If all the solutions produce the same bracketed string, then this is added to the training data. This results in an average of 4556 features. As the data set is small, we do not split it into development, training and test sections as was done for English and German. Instead we carry out a 10-fold cross validation over the entire set. The results for each pruning level are given in Table 5.

The results in Table 5 show that the pattern that held for English and German does not quite hold for Norwegian. While, as expected, the time taken to parse the test set is greatly reduced when using c-structure pruning, there is also a negative impact on the quality of the f-structures. One reason for this is that there are now sentences that could previously be parsed, and that now no longer can be parsed, even with a fragment grammar.[2] With c-structure pruning, the number of fragment parses increases for all thresholds, apart from 10. It is also difficult to compare the Norwegian experiment to the English and German, since the gold standard is constrained to only consist of sentences that can be parsed by the grammar. Theoretically the oracle f-score for the experiment with no prun-

[2]With an extended fragment grammar, this would not happen.

Pruning Level	None	4	5	6	7	8	9	10
Oracle F-Score	**98.76**	94.45	95.60	96.40	96.90	97.52	98.00	98.33
Time (CPU seconds)	218.8	**106.2**	107.4	109.3	112	116.2	124	130.7
# Time Outs	0	0	0	0	0	0	0	0
# Parse Failures	**0.2**	5.7	3.9	2	3.2	4.2	4.6	4.2
# Fragments	1.3	7.7	6.5	4.7	2.8	1.8	1.5	**1.2**

Table 5: Results of c-structure pruning 10-fold cross validation experiments on Norwegian data

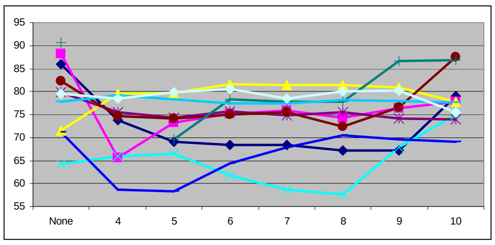


Figure 3: The lower-bound results for each of the 10 cross validation runs across the thresholds

ing should be 100. The slight drop is due to a slightly different morphological analyzer used in the final experiments that treats compound nouns differently. A threshold of 10 gives the best results, with a speedup of 40% and a drop in f-score of 0.43 points. It is difficult to choose the "best" threshold, as the amount of training data is probably not enough to get an accurate picture of the data. For example, Figure 3 shows the lower-bound results for each of the 10 runs. It is difficult to see a clear pattern for all the runs, indicating that the amount of training data is probably not enough for a reliable experiment.

6 Size of Training Data Corpus

The size of the Norwegian training corpus is considerably smaller than the training corpora for English or German, so the question remains how much training data we need in order for the c-structure pruning to deliver reliable results. In order to establish a rough estimate for the size of training corpus required, we carried out an experiment on the German TIGER training corpus.

We randomly divided the TIGER training corpus into sets of 500 sentences. We plot the learning curve of the c-structure pruning mechanism in Figure 4, examining the effect of increasing the size of the training corpus on the oracle f-score on the development set of 371 sentences. The curve shows that, for the German data, the highest oracle f-score of 84.98 was achieved with a training corpus of 32,000 sentences. Although the curve fluctuates, the general trend is that the more training data, the better the oracle f-score.[3]

7 Error Analysis

Given that we are removing some subtrees during parsing, it can sometimes happen that the desired analysis gets pruned. We will take German as an example, and look at some of these cases.

7.1 Separable particles vs pronominal adverbs

The word *dagegen* ("against it") can be a separable prefix (VPART) or a pronominal adverb (PADV). The verb *protestieren* ("to protest") does not take *dagegen* as separable prefix. The verb *stimmen* ("to agree") however does. If we parse the sentence in (1) with the verb *protestieren* and activate pruning, we do not get a complete parse. If we parse the same sentence with *stimmen* as in (2) we do get a complete parse. If we replace *dagegen* by *dafür*, which in the current version of the German LFG can only be a pronominal adverb, the sentence in (3) gets a parse. We also notice that if we parse a sentence, as in (4), where *dagegen* occurs in a position where our grammar does not allow separable prefixes to occur, we get a complete parse for the sentence. These examples show that the pruning mechanism has learned to prune the separable prefix reading of words that can be both separable prefixes and pronominal adverbs.

(1)	Sie	protestieren	dagegen.
	they	protest	against-it
	'They protest against it.'		

(2)	Sie	stimmen	dagegen.
	they	vote	against-it
	'They vote against it.'		

[3]Unexpectedly, the curve begins to decline after 32,000 sentences. However, the differences in f-score are not statistically significant (using the approximate randomization test). Running the same experiment with a different random seed results in a similarly shaped graph, but any decline in f-score when training on more data was not statistically significant at the 99% level.

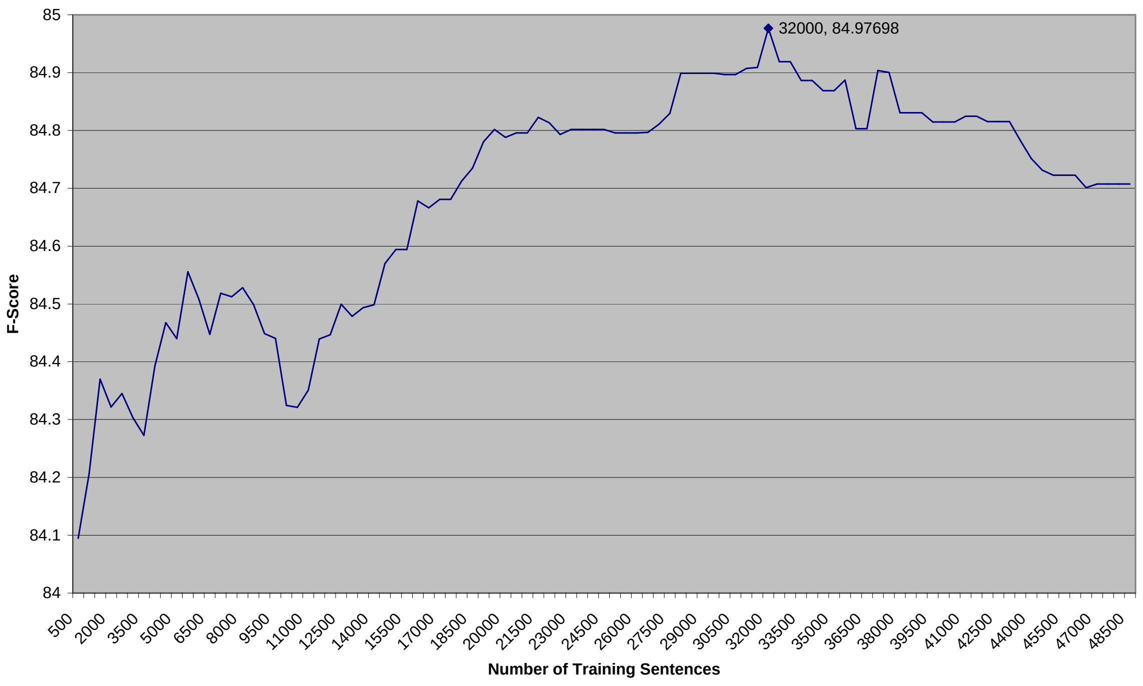


Figure 4: The effect of increasing the size of the training data on the oracle f-score

(3)

Er	protestiert	dafür.
he	protests	for-it

‘He protests in favour of it.’

(4)

Dagegen	protestiert	er.
against-it	protests	he

‘Against it, he protests.’

7.2 Derived nominal vs non-derived nominal

The word *Morden* can be the dative plural of the noun *Mord* (“murder”) or the nominalized form of the verb *morden* (“to murder”). With c-structure pruning activated (at level 5), the nominalized reading, as in (6), gets pruned, whereas the dative plural reading is not (5). At pruning level 6, both readings are assigned a full parse. We see similar pruning of nominalized readings as in (7). If we look in more detail at the raw counts for related subtrees gathered from the training data, we see that the common noun reading for *Morden* occurs 156 times, while the nominalized reading only occurs three times. With more training data, the c-structure pruning mechanism could possibly learn when to prune correctly in such cases.

(5)

Er	redet	von	Morden.
he	speaks	of	murders

‘He speaks of murders.’

(6)

Das	Morden	will	nicht	enden.
the	murdering	wants	not	end

‘The murdering does not want to end.’

(7)

Das	Arbeiten	endet.
the	working	ends

‘The operation ends.’

7.3 Personal pronouns which also function as determiners

There are a number of words in German that can function both as personal pronouns and determiners. If we take, for example, the word *ihr*, which can mean “her”, “their”, “to-her”, “you-pl” etc., the reading as a determiner gets pruned as well as some occurrences as a pronoun. In example (8), we get a complete parse for the sentence with the dative pronoun reading of *ihr*. However, in example (9), the determiner reading is pruned and we fail to get a complete parse. In example (10), we also fail to get a complete parse, but in example (11), we do get a complete parse. There is a parameter we can set that sets a confidence value in certain tags. So, for example, we set the confidence value of INFL-F_BASE[det] (the tag given to the determiner reading of personal pronouns) to be 0.5, which says that we are 50% confident that the tag INFL-F_BASE[det] is correct. This results in examples 8, 9 and 11 receiving a complete parse, with the pruning threshold set to 5.

(8) Er gibt es ihr.
he gives it her
'He gives it to her.'

(9) Ihr Auto fährt.
her/their car drives
'Her/Their car drives.

(10) Ihr kommt.
you(pl) come
'You come.'

(11) Er vertraut ihr.
he trusts her
'He trusts her.'

7.4 Coordination of Proper Nouns

Training the German c-structure pruning mechanism on the TIGER treebank resulted in a peculiar phenomenon when parsing coordinated proper nouns. If we parse four coordinated proper nouns with c-structure pruning activated as in (12), we get a complete parse. However, as soon as we add a fifth proper noun as in (13), we get a fragment parse. This is only the case with proper nouns, since the sentence in (14) which coordinates common nouns gets a complete parse. Interestingly, if we coordinate *n* proper nouns plus one common noun, we also get a complete parse. The reason for this is that proper noun coordination is less common than common noun coordination in our training set.

(12) Hans, Fritz, Emil und Maria singen.
'Hans, Fritz, Emil and Maria sing.'

(13) Hans, Fritz, Emil, Walter und Maria singen.
'Hans, Fritz, Emil, Walter and Maria sing.'

(14) Hunde, Katzen, Esel, Pferde und Affen kommen.
'Dogs, cats, donkeys, horses and apes come.'

(15) Hans, Fritz, Emil, Walter, Maria und Kinder singen.
'Hans, Fritz, Emil, Walter, Maria and children sing.'

We ran a further experiment to test what effect adding targeted training data had on c-structure pruning. We automatically extracted a specialized corpus of 31,845 sentences from the Huge German Corpus. This corpus is a collection of 200 million words of newspaper and other text. The sentences we extracted all contained examples of proper noun coordination and had been automatically chunked. Training on this sub-corpus as well as the original TIGER training data did have the desired effect of now parsing example (13) with c-structure pruning activated.

8 Related Work

Ninomiya et al. (2005) investigate beam thresholding based on the local width to improve the speed of a probabilistic HPSG parser. In each cell of a CYK chart, the method keeps only a portion of the edges which have higher figure of merits compared to the other edges in the same cell. In particular, each cell keeps the edges whose figure of merit is greater than α_{max} - δ, where α_{max} is the highest figure of merit among the edges in the chart. The term "beam thresholding" is a little confusing, since a beam search is not necessary – instead, the CYK chart is pruned directly. For this reason, we prefer the term "chart pruning" instead.

Clark and Curran (2007) describe the use of a supertagger with a CCG parser. A supertagger is like a tagger but with subcategorization information included. Chart pruners and supertaggers are conceptually complementary, since chart pruners prune edges with the same span and the same category, whereas supertaggers prune (lexical) edges with the same span and different categories. Ninomiya et al. (2005) showed that combining a chunk parser with beam thresholding produced better results than either technique alone. So adding a supertagger should improve the results described in this paper.

Zhang et al. (2007) describe a technique to selectively unpack an HPSG parse forest to apply maximum entropy features and get the n-best parses. XLE already does something similar when it applies maximum entropy features to get the n-best feature structures after having obtained a packed representation of all of the valid feature structures. The current paper shows that pruning the c-structure chart before doing (packed) unification speeds up the process of getting a packed representation of all the valid feature structures (except the ones that may have been pruned).

9 Conclusions

In this paper we have presented a c-structure pruning mechanism which has been integrated into the XLE LFG parsing system. By pruning the number of c-structures built in the chart, the next stage of processing, the unifier, has considerably less work to do. This results in a speedup of 67% for English, 49% for German and 40% for Norwegian. The amount of training data for Norwegian was much less than that for English or German, therefore further work is required to fully investigate the effect of c-structure pruning. However, the results, even from the small training data, were encouraging and show the same general patterns as English and German. We showed that for the German training data, 32,000 sentences was the optimal number in order to achieve the highest oracle f-score. There remains some work to be done in tuning the parameters for the c-structure pruning, as our error analysis shows. Of course, with statistical methods one can never be guaranteed that the correct parse will be produced; however we can adjust the parameters to account for known problems. We have shown that the c-structure pruning mechanism described is an efficient way of reducing parse times, while maintaining the accuracy of the overall system.

Acknowledgements

The work presented in this paper was supported by the COINS project as part of the linguistic Collaborative Research Centre (SFB 732) at the University of Stuttgart and by the Norwegian Research Council through the LOGON and TREPIL projects.

References

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria.

Cahill, Aoife, Tracy Holloway King, and John T. Maxwell III. 2007. Pruning the Search Space of a Hand-Crafted Parsing System with a Probabilistic Parser. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 65–72, Prague, Czech Republic, June. Association for Computational Linguistics.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Clark, Stephen and James R. Curran. 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

Forst, Martin, Núria Bertomeu, Berthold Crysmann, Frederik Fouvry, Silvia Hansen-Schirra, and Valia Kordoni. 2004. Towards a dependency-based gold standard for German parsers – The TiGer Dependency Bank. In *Proceedings of the COLING Workshop on Linguistically Interpreted Corpora (LINC '04)*, Geneva, Switzerland.

Kaplan, Ronald M., John T. Maxwell, Tracy H. King, and Richard Crouch. 2004. Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of the ESSLLI 2004 Workshop on Combining Shallow and Deep Processing for NLP*, Nancy, France.

King, Tracy Holloway, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC '03)*, Budapest, Hungary.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ninomiya, Takashi, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Efficacy of Beam Thresholding, Unification Filtering and Hybrid Parsing in Probabilistic HPSG Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 103–114, Vancouver, British Columbia, October. Association for Computational Linguistics.

Rohrer, Christian and Martin Forst. 2006. Improving Coverage and Parsing Quality of a Large-scale LFG for German. In *Proceedings of the Language Resources and Evaluation Conference (LREC-2006)*, Genoa, Italy.

Rosén, Victoria, Paul Meurer, and Koenraad de Smedt. 2006. Towards a Toolkit Linking Treebanking and Grammar Development. In Hajic, Jan and Joakim Nivre, editors, *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories*, pages 55–66, December.

Zhang, Yi, Stephan Oepen, and John Carroll. 2007. Efficiency in Unification-Based N-Best Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 48–59, Prague, Czech Republic, June. Association for Computational Linguistics.

From Grammar-Independent Construction Enumeration to Lexical Types in Computational Grammars

Lars Hellan
NTNU
N-7491 Trondheim
Norway
lars.hellan@hf.ntnu.no

Abstract

The paper presents a code for enumerating verb-construction templates, from which lexical type inventories of computational grammars can be derived, and test suites can be systematically developed. The templates also serve for descriptive and typological research. The code is string-based, with divisions into slots providing modularity and flexibility of specification.

1 Introduction

This paper presents a code for enumerating verb-construction templates. The code is string-based, with divisions into slots providing modularity and flexibility of specification. The templates provide slots for specifying, relative to a construction

- part of speech (POS) of the head
- grammatical relations exposed
- valence bound items
- thematic roles expressed
- situation type
- aspect (Aktionsart)
- part of speech of valence-bound items.

(These parameters altogether cover what is commonly referred to as 'argument structure' relative to the main predicate.) The code is outlined in sections 2-5, and 8.

From the verb construction templates, lexical type inventories of computational grammars can be derived (section 6). The design offers a systematic way of organizing test suites, and herewith improved means of defining intra- and cross-framework reference points of coverage and depth of analysis. The template code also lends itself for descriptive and typological research (section 7).

The design is not geared to any particular framework of computational grammar or linguistics. Examples will be offered relative to HPSG- and LFG- grammars, and the actual conversions from templates to lexical types so far developed relate to HPSG grammars using the LKB platform (cf. (Copestake 2002)), based on the 'HPSG Grammar Matrix' design ((Bender et al. 2002)). Our exposition will be based on the design as it relates to the LKB-grammar *NorSource* (cf. (Beermann and Hellan 2004)) and a Verb-Construction enumeration for Norwegian.

The enterprise here presented has lines going back at least to the mid and late 80ies, both regarding test suite development (e.g., (Flickinger et al. 1987), (Lehmann et al. 1996)) and argument frame inventories ((Hellan et al. 1889)).

2 Code for Template Enumeration

By a *template* for a verb construction we understand a standardized way of exposing selected features of the construction. Exposed features are *classificatory* features, and in this respect, a template may be regarded as a *type*.

A system for enumerating templates should be designed such that they are, internal to a given language, complete and transparent, and across languages, comparable both in templates shared and in templates distinct. Technologically they should be as low level as possible, and platform independent, and be equally accessible to practising field linguists as to NLP researchers in computational settings. With such desiderata in mind, we develop a code residing simply in *strings* of symbols with a minimum of internal syntax.

The basic structural parts of such strings are referred to as *slots*. In the slot specification, the following conventions are observed:

* Slots are interconnected by '-' (hyphen).
* Distinct items inside a slot are interconnected by '_' (underline).
* An item label containing neither '-' nor '_' is an uninterrupted string of letters. Internal composition is indicated by alternation between small and capital letters.

The format can be applied also to non-verbal constructions, but we here focus exclusively on verbal ones. These have a template structure with *five* slots:

Slot 1: POS of the head, and diathesis information.
Slot 2: Valency, or transitivity specification (e.g., `intr`, `tr`, `ditr`,...).
Slot 3: Dependents' specification (syntactic and referential properties of arguments).
Slot 4: Participant roles.
Slot 5: Situation type (written in SMALL CAPS).

Slots 1 and 2 are always filled, the others need not be. A slot not specified is not displayed as empty; however, the sets of labels defined for the various slots are disjoint. Likewise, no labels are distinguished in terms of capital letter vs. not. (1) illustrates the composition, for a type instantiated by the clause *Mary throws the ball*:

(1) v-tr-obDir-suAg_obEjct-EJECTION

Slot 1 here says the head is Verb, slot 2 says that the construction is transitive, slot 3 says that the object har a directional function, slot 4 says that the thematic roles are 'agent', expressed by the subject, and 'ejected', expressed by the object, and slot 5 says that the situation type is one characterizable as 'ejection'.

We start with a survey of the labels used for slot 2, valence information. First, for the use of the notions `intr`, `tr`, `ditr`, the following definitions apply. By a *direct syntactic argument* of a verb, we understand a nominal constituent syntactically related to the verb as *subject-of*, *direct object-of*, or *indirect object-of*, and any clausal constituent with either of these functions. (This *in*cludes expletive subjects and objects, and *ex*cludes clausal constituents in extraposed position; it also excludes any NP or clause governed by a preposition or another relation-item.) An *intransitive* construction is then one with only SUBJECT as a direct syntactic argument, a *transitive* construction has a SUBJECT and one OBJECT as direct syntactic arguments, and a *ditransitive* construction has a SUBJECT and two OBJECTs as direct syntactic arguments. Any valence-bound item other than those now described is represented by an extension of the above transitivity-strings, for instance, in the strings `intrObl` and `trObl`, `Obl` means 'oblique', that is, in addition to the number of arguments represented by `intr/tr`, there is a PP with 'selected' status.

The valence slot includes information as to referential status of the arguments. We say that a direct syntactic argument is *standardly linked* when it has referential content and has a semantic argument function relative to the verb. This *ex*cludes expletive subjects and expletive objects, as well as 'raised' full NPs. The following substrings in slot 2 indicate the presence of items that are *not standardly linked*:
`Impers` ('impersonal'), `Presentational`, `Epon` ('extraposition'), `Nrf` ('non-referential'), `Rais` ('item raised, i.e., non-argument'), `Nrg` ('non-argument' - used in slot 3)).

Specifications are sought to be non-redundant. For instance, the string `intrEpon` occurring in slot 2 entails that there is an expletive subject, and when used for a langauge like English, there is no need to say elsewhere that the subject is expletive. Since what is redundant relative to one language may not be so relative to another, questions of language-parametrized interpretation of code may arise; however, we do not have a basis yet for suggesting whether and how this would be accommodated.

3 Valency labels

The slot for valency, slot 2, has around 45 possible specifications relevant to Norwegian, and we state those in full, to give an impression of what may be the expected scope of this slot; an example illustrates each type:

`intr` = intransitive, i.e., with only SUBJECT as direct syntactic argument.
`intrImpers` = impersonal intransitive, i.e., SUBJECT is an expletive not linked to any other item in the clause. (Ex.: *det regner* 'it rains')
`intrImpersPrtcl` = impersonal intransitive with an aspectual particle. (Ex.: *det klarner opp* 'it clears up')
`intrImpersObl` = impersonal intransitive with an oblique argument. (Ex.: *det synger i fjellene*

'it sings in the mountains')

`intrPresentational` = intransitive with a presentational structure, i.e., an expletive subject and an indefinite NP.

`intrDirPresentational` = `intrPresentational` where the presented NP has a directional function. (Ex.: *det springer en mann* 'there runs a man')

`intrPresentationalLoc` = `intrPresentational` with a locative constituent. (Ex.: *det sitter en mann i stolen* 'there sits a man in the chair')

`intrDir` = intransitive where the subject has a directional function. (Ex.: *gutten løper* 'the boy runs')

`intrAdv` = intransitive with an obligatory adverb. (Ex.: *han fungerer godt* 'he functions well')

`intrPrtcl` = intransitive with an aspectual particle. (Ex.: *regnet varer ved* 'the rain lasts')

`intrObl` = intransitive with an oblique argument. (Ex.: *jeg snakker om Ola* 'I talk about Ola')

`intrOblRais` = intransitive with an oblique argument from which an NP has been 'raised'. (Ex.: *han later til å komme* 'he appears [to] to come')

`intrScpr` = intransitive with a secondary predicate (Small Clause predicate). (Ex.: *gutten virker syk* 'the boy seems sick')

`intrEpon` = intransitive with an 'extraposed' clause. (Ex.: *det hender at han kommer* 'it happens that he comes')

`intrPrtclEpon` = intransitive with an 'extraposed' clause and an advparticle. (Ex.: *det hører med at han kommer* Mock Eng: "it hears along that he comes")

`intrOblEpon` = intransitive with an 'extraposed' clause and an oblique argument. (Ex.: *det haster med å rydde* Mock Eng: "it urges with to tiden up")

`intrPrtclOblEpon` = intransitive with an 'extraposed' clause, an oblique argument, and an advparticle. (Ex.: *det ser ut til at han kommer* Mock Eng: "it looks out to that he comes")

`intrPrtclOblRais` = intransitive with an oblique argument from which an NP has been 'raised', and an advparticle. (Ex.: *han ser ut til å komme* Mock Eng: "he looks out to to come")

`intrImplicobjObl` = intransitive with implicit object, followed by PP (Ex.: *han sølte på seg* 'he spilled on himself')

`tr` = transitive, i.e., with SUBJECT and one OBJECT.

`trDir` = transitive, where the subject is understood in a directional function. (Ex.: *Kari når toppen* 'Kari reaches the top')

`trPrtcl` = transitive with an advparticle. (Ex.: *Kari fant ut svaret* 'Kari found out the answer')

`trPresentational` = presentational structure with an NP preceding the 'presented' NP. (Ex.: *det venter ham en ulykke* 'there awaits him an accident')

`trObl` = transitive with an oblique. (Ex.: *jeg sparker Ola i baken* 'I kick Ola in the butt')

`trEponSu` = transitive with an extraposed clause correlated with the subject, and an argument object. (Ex.: *det bekymrer meg at han kommer* 'it worries me that he comes')

`trEponOb` = transitive with an extraposed clause correlated with the object, and an argument subject.. (Ex.: *vi muliggjorde det at han fikk innreisetillatelse* 'we possible-made it that he got entrance visa')

`trScpr` = transitive with a secondary predicate (Small Clause predicate). (Ex.: *han sparket ballen flat* 'he kicked the ball flat')

`trNrf` = transitive whose object is non-referential. (Ex.: *Kari skammer seg* "Kari shamess herself" - 'Kari is ashamed')

`ditr` = ditransitive, i.e., with SUBJECT and two OBJECTs (here referred to by the traditional terms 'indirect' ('iob') and 'direct' object, when distinction is necessary).

`ditrObl` = ditransitive with oblique. (Ex.: *jeg kaster Ola kakestykker i ansiktet* "I throw Ola cakes in the face" - 'I throw cakes in the face of Ola')

`ditrNrf` = ditransitive whose indirect object is non-referential. (Ex.: *han foresetter seg å komme* 'he [foresetter] himself to come)

`copAdj` = predicative copular construction with adjectival predicative. (Ex.: *han er snill* 'he is kind'). (Similarly: `copN`. (Ex.: *han er bonde* 'he is peasant'); `copPP` (Ex.: *forestillingen var under enhver kritikk* 'the performmance was below critique'); `copPredprtcl` (Ex.: *Ola er som en gud* 'Ola is like a god'))

`copIdN` = identity copular construction with nominal predicative. (Ex.: *dette er mannen* 'this is the man'.) (Similarly: `copIdAbsinf` (Ex.: *oppgaven er å spise silden* 'the task is to eat the herring'.); `copIdDECL` (Ex.: *problemet er at han spiser silden* 'the problem is that he eats the herring'.); `copIdYN` (Ex.: *problemet er om han spiser silden* 'the problem is whether he eats the herring'.); `copIdWH` (Ex.: *spørsmålet er hvem som spiser silden* 'the question is who eats the herring'.))

`copEponAdj` = predicative copular construction with adjectival predicative and the 'logical subject' extraposed. (Ex.: *det er uvisst hvem som kommer* 'it is uncertain who that comes'.) Similarly: `copEponN` (Ex.: *det er et spørsmål hvem som kommer* 'it is a question who comes'.); `copEponPP` (Ex.: *det er hinsides diskusjon at han kommer* 'it is beyond discussion that he comes'.); `copEponPredprtcl`

(Ex.: *det var som bestilt at han tapte igjen* 'it was like booked that he lost again'.))

4 Dependents' labels

The slot for dependents, slot 3, has around 40 labels relevant for Norwegian. Each is built up with a first part indicating the grammatical function of the item specified (`su, ob, iob, obl, sc, epon`), and the remainder providing the specification, possibly also with some internal structure. The following lists most of them:

`suExpl` = subject is an expletive.

`suDECL` = subject is a declarative clause. (Similarly: `suYN` = subject is a yes-no-interrogative clause; `suWH` = subject is a wh-interrogative clause; `suAbsinf` = subject is an infinitival clause with non-controlled interpretation.)

`suNrg` = subject is a non-argument.

`obDir` = object is understood in a directional capacity.

`obRefl` = object is a reflexive.

`obReflExpl` = object is an expletive reflexive.

`obDECL` = object is a declarative clause. (Similarly: `obYN`, `obWH`, `obAbsInf`)

`obEqInf` = object is an infinitive equi-controlled by the only available possible controller.

`obEqSuInf` = object is an infinitive equi-controlled by subject.

`obEqIobInf` = object is an infinitive equi-controlled by indirect object.

`obEqSuBareinf` = object is a bare infinitive equi-controlled by subject.

`obEqIobBareinf` = object is a bare infinitive equi-controlled by indirect object.

`iobRefl` = indirect object is a reflexive.

`iobReflExpl` = indirect object is an expletive reflexive.

`oblEqSuInf` = the governee of the oblique is an infinitive equi-controlled by subject.

`oblEqObInf` = the governee of the oblique is an infinitive equi-controlled by object.

`oblRaisInf` = the governee of the oblique is an infinitive which is raising-controlled by the subject.

`oblRefl` = the governee of the oblique is a reflexive.

`oblDECL` = the governee of the oblique is a declarative clause. (Similarly: `oblYN`, `oblWH`, `oblAbsinf`)

`oblPRTOFsubj` = the referent of the governee of the oblique is interpreted as part-of the referent of the subject. (Ex.: *jeg fryser på ryggen* 'I freeze on the back' - I'm cold on my back')

`oblPRTOFobj` = the referent of the governee of the oblique is interpreted as part-of the referent of the object. . (Ex.: *jeg sparker Ola i baken* 'I kick Ola in the butt')

`oblPRTOFiobj` = the referent of the governee of the oblique is interpreted as part-of the referent of the indirect object. (Ex.: *jeg kaster Ola kakestykker i ansiktet* "I throw Ola cakes in the face" - 'I throw cakes in the face of Ola')

`oblEponAbsinf` = extraposed is a non-controlled infinitive occurring as governee of an oblique.

`oblEponDECL` = extraposed is a declarative clause occurring as governee of an oblique.

`scSuNrg` = the secondary predicate is predicated of a non-argument subject (i.e., a subject not serving as semantic argument of the matrix verb - i.e., a 'raising to subject' subject).

`scObNrg` = the secondary predicate is predicated of a non-argument object (i.e., an object not serving as semantic argument of the matrix verb - i.e., a 'raising to object' object).

`scAdj` = the secondary predicate is headed by an adjective. (Similarly: `scN`, `scPP`, `scPredprtcl`, `scInf`, `scBareinf`)

`eponDECL` = extraposed is a declarative clause. (Similarly: `eponYN`, `eponWH`, `eponCOND`, `eponEqInf`, `eponAbsinf`)

We illustrate with a use of the 'small clause' specification `scSuNrg`. One of the construction types it serves to qualify is exemplified by

han synes syk 'he seems sick',

which is a raising construction of the logical form 'seem (he sick)', where the subject *han* does not have a semantic argument function relative to the verb. The label specifying this type is

`v-intrScpr-scSuNrg_scAdj`

where `intrScpr` states that the only constituents syntactically present are a subject and a secondary predicate, `scSuNrg` states that the subject lacks semantic argument status relative to the verb, and `scAdj` states that the secondary predicate is headed by an adjective. The circumstance that the latter two specifications concern dependents rather than over-all valence, is marked by an underscore interrelating them.

A transitive counterpart to this type is exemplified by *han synes meg syk* 'he seems me sick', of the logical form 'seem-to-me (he sick)', where the subject *han* still does not have a semantic argument function relative to the verb. The label specifying this type is

`v-trScpr-scSuNrg_scAdj`

where `trScpr` states that the constituents syntactically present are a subject, an object and a secondary predicate, and the other specifications serve as in the previous example.

With utilization of the slot 2 and slot 3 determinants, around 200 templates have been defined for Norwegian (these can be viewed at the site typecraft.org (research/research projects).

Deciding what is to go in slot 2 and what in slot 3 is in most cases starightforward, but not always. For instance, it will be noted that in the copula valence labels entered at the end of the list in section 3, specifications like '`YN`'. '`DECL`' etc are used which are otherwise used mainly in dependents' specifications. For one thing, in a case where an adverb or a PP is obligatory, and there is no relational 'super-term' available for specifying its presence, one will refer to the constituent by head property directly, as in `intrAdv`. In the case of the copulas, one might have entered '`YN`' etc tied to a grammatical relation 'identifier' for the identity copula, and 'predicative' for predicative copula, giving, e.g., `v-copPred-predAdj` instead of `v-copAdj` for the predicative adjectival copula construction, and `v-copID-idN` instead of `v-copIdN` for the identity construction. Here it is essentially length of labels, and sparsity concerns concerning grammatical relations notions, which have counted in favor of the latter options - either option is in principle possible.

Conversely, instead of writing '`trScpr-scSuNrg_scAdj`' in the example discussed, one could have written '`trScprAdj-scSuNrg`', or '`trScpr-scSuNrgAdj`' - against the former is a wish to generally have POS of dependents being stated in the dependents' slot, and against the latter counts the circumstance that the secondary predicate specifications are in general rather complex already; this point will be further illustrated in section 8 below.

5 Thematic roles and situation types

In specifying semantic roles and situation types, classifications available are less robust than they are for the factors covered above, and for that reason, the notational system will not insist that they should be provided. Closest to practical consensus are core semantic role labels such as 'agent', 'patient' and the like, and aspectual specifications; much less established is a set of situation types covering the full range of constructions in a language. In this section we do not provide any tentative list of values to be used, but comment only on how they are expressed.

As exemplified in (1), each semantic role label is built up with a first part indicating the grammatical function of the item specified, and the remainder providing the specification - thus, `suAg`, `obEjct`.. Unlike the case with dependents' labels, the remaining part has no internal structure.

Situation types may in principle cover anything between Aktionsart and detailed situational specification, like in a FrameNet label (cf. http://framenet.icsi.berkeley.edu/). In the system currently implemented, the level of specification is somewhere between these two: Sitaution type labels can be decomposed into standard aspectual notions (like those proposed in Smith 1991, 1997) and specifications uniquely identifying each type. An example is the possible situation label CAUSATION_WITH_CAUSINGEVENT, which means "causation where the cause is itself an event and its event type is linguistically identified", and which implies certain aspectual notions, such as 'dynamic' and 'telic'.

We illustrate the full specification of the example *han synes meg syk* 'he seems me sick' discussed above, which is:

(2)
`v-trScpr-scSuNrg_scAdj-obCog_scThSit-PROPOSITIONALATTITUDE`

'`obCog`' here means that the object expresses a 'cognizer', and '`scThSit`' that the secondary predication expresses a 'situational theme'. It will be noted that, consistent with the non-argument status of the subject, there is no thematic role tied to the subject.

With utilization of the slot 4 and slot 5 determinants, around 280 templates are currently defined for Norwegian.

Slots 3 and 4 are both 'constituent oriented', and may provide specifications of one and the same item. For instance, in (2) all of `scSuNrg`, `scAdj` (slot 3), and `scThSit` (slot 4) define the secondary predicate. In principle it would be possible to draw these different specifications together into a unitary, but more complex, specification. This was done, e.g., in the TROLL system (cf. (Hellan et al. 1989)), where arguments were specified as triples of (i) head's POS, (ii) grammatical function, and (iii) thematic role (including possible non-argument status). Among possible advantages of the current system are that it better profiles 'global' properties of the construction, that it better displays the profile of participant roles, when entered, and makes omission of them practically more easy. Cf. (Lehmann et al. 1996) for further discussion.

6 From Templates to Grammars

The information encoded in the first three slots attains the same depth of argument structure description as is modeled in standard Matrix-HPSG grammars, and approximately as in standard LFG-Pargram grammars (cf. (Butt et al. 1999)). Argument structure being what is generally encoded in *lexical entries for verbs* in such grammars, we now address how the template system can be used as lexical types or macros.

Minimally, templates could be imported as 'en bloc' type- or macro labels into computational grammars. However, the hyphenation and underscore structure of the templates suggest more modular strategies, as we will now show for a typed feature structure design.

For instance, for the template in (2) - `v-trScpr-scSuNrg_scAdj-obCog_scThSit-PROPOSITIONALATTITUDE` one could see this as equivalent to a unification of syntactic types representing, resp., 'verb-headed', 'transitive with a secondary predicate', 'secondary predicate predicated of raised subject', and 'secondary predicate headed by an adjective', and the semantic types 'cognizer, as role of object', and 'situational theme', as role of secondary predicate. In the tdl notation used in LKB grammars, this would suggest (3) as one of its type definitions (ignoring the situation type label for now):

(3)
v-trScpr-scSuNrg_scAdj-obCog_scThSit :=
v & trScpr & scSuNrg & scAdj & obCog & scThSit.

Here, the type in line 1 is defined as the unification of the types inter-connected with '&'. Mechanically speaking, in going from template to grammatical type, one simply replaces each hyphen or underline in the template label by a type unification symbol. As individual types (as is customary, mention of such types is done with italics) *v*, *trScpr*, *scSuNrg*, *scAdj, obCog* and *scThSit* will all be at 'sign' level. That is: when, in an LKB-like grammar, these types are to unify with each other, they must belong to a common supertype, and given that what they are composing together is the type of a verb lexeme, this is, in a Matrix-type grammar, an instance of the type *sign*. For instance, the type definition for *scAdj*, relative to NorSource, is (with PREDIC being an attribute introducing secondary predicates, and QVAL introducing grammatical relations in a non-list fashion, à la LFG):

(4) scAdj := sign &
[SYNSEM | LOCAL | CAT | QVAL | PREDIC | LOCAL | CAT | HEAD adj].

In what part of the over-all grammar will these types be introduced? A first question is if 'construction' is a type of entity to be assumed among the building blocks of the grammar. In standard HPSG and LFG design, the tendency is to project construction types into the inventory of lexical types, so that verb-construction types enter the grammar through the subcategorization frames associated with verbs. On this view, a definition like (3) will be in an inventory of *lexical type definitions*.

How do lexical items, in this case verbs, relate to these types? If we consider the more standard design in HPSG and LFG grammars, where a verb has as many entries as there are construction frames in which it can enter, most verbs can enter more than one constructional environment.[1] Thus, in the typical case, a verb will be associated with a subset of the totality of types derivable from the template collection, and thus have entries each one defined by one of these types.

Some points of useful flexibility in this mapping may be noted, illustrated with the choice of head in secondary predicate constructions (cf. (4)): in constructions like those discussed, eligible such heads are in principle adjectives, adverbs, prepositions, predicative particles and infinitivals. For a given verb, the full range of options need not be open, hence in defining the general verb type corresponding to the template `v-trScpr-scSuNrg_scAdj-obCog_scThSit` one may want to leave the sc-head open, and rather have a way of appending that information for each individual verb. By separating out the

[1] We here ignore possible designs which might, for each verb, represent it with one single entry, and account for its many frames of occurrence either through a network of lexical rules, or through underspecifying each entry to yield, for each verb, exactly the range of environments it can occur in.

relevant part (*_scAdj, _scAdv...,)*, and defining *v-trScpr-sSubNrg_scAdj*, *v-trScpr-scSuNrg_scAdv*, etc. as subtypes of *v-trScpr-sSubNrg*, one can in an LKB grammar enter each verb in the lexicon with the appropriate last part provided (and leave them out when the verb actually can combine with all options). In such an approach one has to define all constellations in the relevant type file, the gain lies in the flexibility one has in the lexical entry specifications. The same advantages apply with regard to specification of semantic roles.

7 Uses of the template inventories

A first possible usage of a template inventory is that one can employ a set of example sentences illustrating the various templates as a *test suite* for the grammar. Given the systematic design of the template list, one is assured to have a systematic test suite in the respects covered by the templates.

A second benefit of the design is as a basis for building cross-linguistically matching test-suites, to the extent that templates coincide cross-linguistically.

For linguistic typology, once one has template lists developed for many languages, comparison and systematization of differences can be facilitated.

For linguistic description and grammar creation, having established template lists for related languages may enhance efficiency, in providing 'check-list' starting and reference points.

All of these points will presuppose that one can reach a commonly approved standard of notation. (In principle, with different types of notation, but a one-to-one correlation between notations, similar effects may be gained, although there is then an extra step of identifying correlations.)

Currently, such a combined initiative of notation development and typological investigation is being pursued for a group of Ghanaian languages in consonance with the Norwegian system; cf. (Dakubu, 2008). (For both systems, full template and example lists can be viewed at the site typecraft.org mentioned above.)

As still another enterprise connected to the present template inventory may be mentioned a partial *ontology of verb construction types* developed with the LKB platform (in principle exportable also to OWL), representing all of the templates in the Norwegian inventory and some more. For a partial description, see (Hellan 2007).

Relative to the present system, a *verb class* can be identified as a set of verbs which are accommodated by the same set of construction types. (This notion of 'verb class' is related to that employed in (Levin 1993), which is based on *alternations* between construction types. An alternation, such as the *'spray-load* alternation', can be viewed as a *pair* of construction types in which a number of verbs can participate, typically with rather similar semantics, highlighting – by a 'minimal pair' technique - semantic properties of the constructions chosen.)

8 More complex types

In its current version, the system does not include 'derived' constructions, of which in Norwegian *passive* constructions would be the main instance. As a prerequisite for a notational system for derivation, systems will first be made for selected Bantu and Ethio-Semitic languages (representing future development)

Possibly also of a derivational nature, but here treated as basic patterns, are 'Secondary predicate' constructions, a few of which were discussed above. To indicate where the Norwegian label inventory probably reaches its peak of complexity, we give a brief resymé of the parameters involved in these constructions, and the more complex labels employed.

The secondary predicate (henceforth: secpred) can relate to the main predicate either as the content of a propositional attitude or perception, or as concurring in time, or as the causee of a causation. In the latter case, either an event is portrayed as the cause (indicated by the substring `....Cse`), or an entity. In the former case, the causing event can have from zero to two participants, and when one or two, one can be implicit. What can never be implicit is the entity of which the secpred is predicated: it may occur as subject or object, and in either case either realizing this grammatical function by itself (in which case the function is 'non-argument'), or sharing it with a participant of the causing event (in which case the function has 'argument' status). The following slot 3 labels serve to encode the various possibilities:

`scObArgConcurr` (*he drank the coffee warm*) `scObNrgRes` (*he made me sick*): Of the causing event, only the participant denoted by the subject is specified.

`scSuArgCse` (*kaffen koker bort* 'the coffee boils away'): The matrix verb (together with its argument subject) expresses part of the description of the causing event.

`scObArgCse` (*han sparket ballen flat* 'he kicked the ball flat'): The secondary predicate is predicated of an argument object, and the matrix verb (together with its object) expresses part of the causing event.

`scSuNrgCse` (*landsbyen snør ned* 'the village snows down'): The secondary predicate is predicated of a non-argument subject, and the matrix verb expresses part of the causing event.

`scObNrgCse` (*han sang rommet tomt* 'he sang the room empty'): The secondary predicate is predicated of a non-argument object, and the matrix verb (together with its subject) expresses part of the causing event.

In dealing with typologically different languages, it is not a priori given what constructional template options may present themselves (see Dakubu op.cit. for discussion of some Volta Basin languages). Whatever these additional types may be, in designing labels, one probably should not exceed the complexity of the labels just presented.

9 Conclusion

With an encoding of a construction type's argument structure and semantics which is probably representative of what one may want to expose, each template in the system presented here is by itself as compressed as can be, which gives the template structure some interest by itself. However, it is through the totality of templates, and through the design by which they can be easily enumerated, compared and computed, that the system presented may be a contribution to grammar engineering and language typology alike. While the system reflects such ambitions, it is still in an initial state of deployment both in grammar engineering and typology, and its potential value will reside in the extent to which it will be used, and receive feedback for usability.

References

Beermann, Dorothee and Lars Hellan. 2004. A treatment of directionals in two implemented HPSG grammars. In St. Müller (ed) *Proceedings of the HPSG04 Conference,* CSLI Publications */http://csli-publications.stanford.edu/*

Bender, Emily M., Dan Flickinger, and Stephan Oepen. 2002. The Grammar Matrix: An open-source starter kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of the Workshop on Grammar Engineering and Evaluation*, Coling 2002, Taipei.

Butt, Miriam, Tracy Holloway King, Maria-Eugenia Nini and Frederique Segond. 1999. *A Grammar-writer's Cookbook*. Stanford: CSLI Publications.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars.* CSLI Publications, Stanford.

Dakubu, Mary E. K. 2008. The Construction label project: a tool for typological study. Presented at West African Languages Congress (WALC), Winneba, July 2008.

Flickinger, Daniel, John Nerbonne, Ivan A. Sag, and Thomas Wassow. 1987. Toward Evaluation of NLP Systems. Technical report. Hewlett-Packard Laboratories. Distributed at the 24th Annual Meeting of the Association for Computational Linguistics (ACL).

Hellan, Lars. 2007. On 'Deep Evaluation' for Individual Computational Grammars and for Cross-Framework Comparison. In: T.H. King and E. M. Bender (eds) *Proceedings of the GEAF 2007 Workshop*. CSLI Studies in Computational Linguistics ONLINE. CSLI Publications. *http://csli-publications.stanford.edu/*

Hellan, Lars., Lars Johnsen and Anneliese Pitz. 1989. TROLL. Ms., NTNU

Lehmann, Sabine., S. Oepen, S. Regier-Prost, K. Netter, V. Lux, J. Klein, K. Falkedal, F. Fouvry, D. Estival, E. Dauphin, H. Compagnion ,J. Baur, L. Balkan, D. Arnold. 1996. Test Suites for Natural Language Processing. *Proceedings of COLING* 16, p. 711-16.

Levin, Beth. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press.

Smith, Carlota. 1991, 1997. *The Parameter of Aspect.* Kluwer Publishers, Dordrecht.

Designing Testsuites for Grammar-based Systems in Applications

Valeria de Paiva
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304 USA
valeria.paiva@gmail.com

Tracy Holloway King
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304 USA
thking@parc.com

Abstract

In complex grammar-based systems, even small changes may have an unforeseeable impact on overall system performance. Regression testing of the system and its components becomes crucial for the grammar engineers developing the system. As part of this regression testing, the testsuites themselves must be designed to accurately assess coverage and progress and to help rapidly identify problems. We describe a system of passage-query pairs divided into three types of phenomenon-based testsuites (sanity, query, basic correct). These allow for rapid development and for specific coverage assessment. In addition, real-world testsuites allow for overall performance and coverage assessment. These testsuites are used in conjunction with the more traditional representation-based regression testsuites used by grammar engineers.

1 Introduction

In complex grammar-based systems, even small changes may have an unforeseeable impact on overall system performance.[1] Systematic regression testing helps grammar engineers to track progress, and to recognize and correct shortcomings in linguistic rule sets. It is also an essential tool for assessing overall system status in terms of task and runtime performance.

As discussed in (Chatzichrisafis et al., 2007), regression testing for grammar-based systems involves two phases. The first includes systematic testing of the grammar rule sets during their development. This is the part of regression testing that grammar engineers are generally most familiar with. The second phase involves the deployment of the grammar in a system and the regression testing of the grammar as a part of the whole system. This allows the grammar engineer to see whether changes have any effect on the system, positive or negative. In addition, the results of regression testing in the system allow a level of abstraction away from the details of the grammar output, which can ease maintenance of the regression testsuites so that the grammar engineers do not need to change the gold standard annotation every time an intermediate level of representation changes.

In this paper, we focus on the design of testsuites for grammar-based systems, using a question-answering system as a model. In particular, we are interested in what types of testsuites allow for rapid development and efficient debugging.

1.1 The Question-Answering System

To anchor the discussion, we focus on regression testsuites designed for a grammar-based question-answering system (Bobrow et al., 2007). The Bridge system uses the XLE (Crouch et al., 2008) parser to produce syntactic structures and then the XLE ordered rewrite system to produce linguistic semantics (Crouch and King, 2006) and abstract knowledge representations. Abstract knowledge representations for passages and queries are processed by an entailment and contradiction detection system which determines whether the query is

[1]We would like to thank Rowan Nairn for his design and implementation of the regression platform that runs these testsuites. We would also like to thank the PARC Natural Language Theory and Technology group for their work with these testsuites and their comments on this paper.

entailed, contradicted, or neither by the passage.

Entailment and contradiction detection between passages and queries is a task well suited to regression testing. There are generally only two or three possible answers given a passage and a query: entails, contradicts or neither (or in the looser case: relevant or irrelevant). *Wh*-questions (section 5.1) receive a YES answer if an alignment is found between the *wh*-word in the query and an appropriate part of the passage representation; in this case, the proposed alignment is returned as well as the YES answer. This is particularly important for *who* and *what* questions where more than one entity in the passage might align with the *wh*-word.

From the standpoint of regression testing, two important aspects of the question-answering application are:

(1) The correct answer for a given pair is independent of the representations used by the system and even of which system is used.

(2) The passage-query pairs with answers can be constructed by someone who does not know the details of the system.

The first aspect means that even drastic changes in representation will not result in having to update the regression suites. This contrasts sharply with regressions run against representative output which require either that the gold standard be updated or that the mapping from the output to that standard be updated. The second aspect means that externally developed testsuites (e.g. FraCaS (Cooper et al., 1996), Pascal RTE (Sekine et al., 2007)) can easily be incorporated into the regression testing and that grammar engineers can rapidly add new testsuites, even if they do not have experience with the internal structure of the system. These aspects also mean that such passage-query pairs can be used for cross-system comparisons of coverage (Bos, 2008).

1.2 Testsuite Types

In the regression testsuites designed for the question-answering system, the passage-query pair testsuites are divided into two main types: those that focus on single phenomena (section 2) and those that use real-world passages (section 3). The phenomenon-based testsuites allow the grammar engineer to track the behavior of the system with respect to a given construction, such as implicativity, noun-noun compounds, temporal expressions, or comparatives. In contrast, the real-world passages allow the grammar engineer to see how the system will behave when applied to real data, including data which the system will encounter in applications. Such sentences tend to stress the system in terms of basic performance (e.g. efficiency and memory requirements for processing of long sentences) and in terms of interactions of different phenomena (e.g. coordination ambiguity interacting with implicativity).

In addition to the passage-query pairs, the system includes regression over representations at several levels of analysis (section 4). These are limited in number, focus only on core phenomena, and are not gold standard representations but instead the best structure of the ones produced. These are used to detect whether unintentional changes were introduced to the representations (e.g. new features were accidentally created).

2 Phenomenon Sets

Real-world sentences involve analysis of multiple interacting phenomena. Longer sentences tend to have more diverse sets of phenomena and hence a higher chance of containing a construction that the system does not handle well. This can lead to frustration for grammar engineers trying to track progress; fixing a major piece of the system can have little or no effect on a testsuite of real-world examples. To alleviate this frustration, we have extensive sets of hand-crafted test examples that are focused as much as possible on single phenomenon. These include externally developed testsuites such as the FraCaS (Cooper et al., 1996) and HP testsuites (Nerbonne et al., 1988). Focused testsuites are also good for quickly diagnosing problems. If all the broken examples are in the deverbal testsuite, for example, it gives grammar engineers a good idea of where to look for bugs.

The majority of the testsuites are organized by syntactic and semantic phenomena and are designed to test all known variants of that phenomenon (see (Cohen et al., 2008) on the need to use testsuites designed to test system coverage as well as real-world corpora). For the question-answering system, these include topics such as anaphora, appositives, copulars, negation, deverbal nouns and adjectives, implicatives and factives, temporals, cardinality and quantifiers, comparatives, possessives, context introducing nouns, and pertainyms. These categories align with many of those cited by (Bos, 2008) in his discussion of semantic parser coverage. Some example passage-query pairs for deverbal nouns are shown in (3).

(3) a. **P:** Ed's abdication of the throne was welcome.
Q: Ed abdicated the throne.
A: YES

b. **P:** Ed's abdication was welcome.
Q: Ed abdicated.
A: YES

c. **P:** Ed is an abdicator.
Q: Ed abdicated.
A: YES

Each of the phenomena has three sets of testsuites associated with it. Sanity sets (section 2.1) match a passage against itself. The motivation behind this is that a passage should generally entail itself and that if the system cannot capture this entailment, something is wrong. Query sets (section 2.2) match the passage against query versions of the passage. The simplest form of this is to have a polarity question formed from the passage. More complex versions involve negative polarity questions, questions with different adjuncts or argument structures, and questions with synonyms or antonyms. Basic correct sets (section 2.3) are selected passage-query pairs in which the system is known to obtain the correct answer for the correct reason. The idea behind these sets is that they can be run immediately by the grammar engineer after making any changes and the results should be 100% correct: any mistakes indicates a problem introduced by the grammar engineer's changes.

2.1 Sanity Sets

The entailment and contradiction detection part of the system is tested in isolation by matching queries against themselves. Some example sanity pairs from the copula testsuite are shown in (4).

(4) a. **P:** A boy is tall.
Q: A boy is tall.
A: YES

b. **P:** A girl was the hero.
Q: A girl was the hero.
A: YES

c. **P:** The boy is in the garden.
Q: The boy is in the garden.
A: YES

d. **P:** The boy is not in the garden.
Q: The boy is not in the garden.
A: YES

Note that queries in the question-answering system do not have to be syntactically interrogative. This allows the sanity pairs to be processed by the same mechanism that processes passage-query pairs with syntactically interrogative queries.

The sanity check testsuites are largely composed of simple, hand-crafted examples of all the syntactic and semantic patterns that the system is known to cover. This minimal check ensures that at least identical representations trigger an entailment.

2.2 Query Sets

The query sets form the bulk of the regression sets. The query sets comprise passages of the types found in the sanity sets, but with more complex queries. The simplest form of these is to form the polarity question from the passage, as in (5). More complex queries can be formed by switching the polarity from the passage to the query, as in (6).

(5) a. **P:** A boy is tall.
Q: Is a boy tall?
A: YES

b. **P:** A girl was the hero.
Q: Was a girl the hero?
A: YES

(6) **P:** The boy is not in the garden.
Q: Is the boy in the garden?
A: NO

To form more complex pairs, adjuncts and argument structure can be altered from the passage to the query. These have to be checked carefully to ensure that the correct answer is coded for the pair since entailment relations are highly sensitive to such changes. Some examples are shown in (7). Alternations such as those in (7c) are crucial for testing implicativity, which plays a key role in question answering.

(7) a. **P:** An older man hopped.
Q: A man hopped.
A: YES

b. **P:** John broke the box.
Q: The box broke.
A: YES

c. **P:** Ed admitted that Mary arrived.
Q: Mary arrived.
A: YES

A similar type of alteration of the query is to substitute synonyms for items in the passage, as in (8). This is currently done less systematically in the testsuites but helps determine lexical coverage.

(8) a. **P:** Some governments ignore historical facts.
Q: Some governments ignore the facts of history.
A: YES

b. **P:** The boys bought some candy.
Q: The boys purchased some candy.
A: YES

In addition to the testsuites created by the question-answering system developers, the query sets include externally developed pairs, such as those created for FraCaS (Cooper et al., 1996). These testsuites also involve handcrafted passage-query pairs, but the fact that they were developed outside of the system helps to detect gaps in system coverage. In addition, some of the FraCaS pairs involve multi-sentence passages. Since the sentences in these passages are very short, they are appropriate for inclusion in the phenomenon-based testsuites. Some externally developed testsuites such as the HP testsuite (Nerbonne et al., 1988) do not involve passage-query pairs but the same techniques used by the grammar engineers to create the sanity and the query sets are applied to these testsuites as well.

2.3 Basic Correct Sets

A subset of the query sets described above are used to form a core set of basic correct testsuites. These testsuites contain passage-query pairs that the developers have determined the system is answering correctly for the correct reason.

Since these testsuites are run each time the grammar engineer makes a change to the system before checking the changes into the version control repository, it is essential that the basic correct testsuites can be run quickly. Each pair is processed rapidly because the query sets are composed of simple passages that focus on a given phenomenon. In addition, only one or two representatives of any given construction is included in the basic correct set; that is, the sanity sets and query sets may contain many pairs testing copular constructions with adjectival complements, but only a small subset of these are included in the basic correct set. In the question-answering system, ∼375 passage-query pairs are in the basic correct sets; it takes less than six minutes to run the full set on standard machines. In addition, since the basic correct sets are divided by phenomena, developers can first run those testsuites which relate directly to the phenomena they have been working on.

Examining the basic correct sets gives an overview of the expected base coverage of the system. In addition, since all of the pairs are working for the correct reason when they are added to the basic correct set, any breakage is a sign that an error has been introduced into the system. It is important to fix these immediately so that grammar engineers working on other parts of the system can use the basic correct sets to assess the impact of their changes on the system.

3 Real-world Sets

The ultimate goal of the system is to work on real-world texts used in the application. So, tests of those texts are important for assessing progress on naturally occurring data. These testsuites are created by extracting sentences from the corpora expected to be used in the run-time system, e.g. newspaper text or the Wikipedia.[2] Queries are then created by hand for these sentences. Once the system is being used by non-developers, queries posed by those users can be incorporated into the testsuites to ensure that the real-world sets have an appropriate range of queries. Currently, the system uses a combination of hand-crafted queries and queries from the RTE data which were hand-crafted, but not by the question-answering system developers. Some examples are shown in (9).

(9) a. **P:** The interest of the automotive industry increases and the first amplifier project, a four-channel output module for the German car manufacturer, Porsche, is finished.
Q: Porsche is a German car manufacturer.
A: YES

b. **P:** The Royal Navy servicemen being held captive by Iran are expected to be freed to-

[2]If the application involves corpora containing ungrammatical input (e.g. email messages), it is important to include both real-world and phenomenon sets for such data.

day.
Q: British servicemen detained
A: YES

c. **P:** "I guess you have to expect this in a growing community," said Mardelle Kean, who lives across the street from John Joseph Famalaro, charged in the death of Denise A. Huber, who was 23 when she disappeared in 1991.
Q: John J. Famalaro is accused of having killed Denise A. Huber.
A: YES

These real-world passages are not generally useful for debugging during the development cycle. However, they serve to track progress over time, to see where remaining gaps may be, and to provide an indication of system performance in applications. For example, the passage-query pairs can be roughly divided as to those using just linguistic meaning, those using logical reasoning, those requiring plausible reasoning, and finally those requiring world knowledge. Although the boundaries between these are not always clear (Sekine et al., 2007), having a rough division helps in guiding development.

4 Regression on Representations

There has been significant work on regression testing of a system's output representations (Nerbonne et al., 1988; Cooper et al., 1996; Lehmann et al., 1996; Oepen et al., 1998; Oepen et al., 2002): designing of the testsuites, running and maintaining them, and tracking the results over time. As mentioned in the previous discussion, for a complex system such as a question-answering system, having regression testing that depends on the performance of the system rather than on details of the representations has significant advantages for development because the regression testsuites do not have to be redone whenever there is a change to the system and because the gold standard items (i.e., the passage-query pairs with answers) can be created by those less familiar with the details of the system.

However, having a small but representative set of banked representations at each major level of system output has proven useful for detecting unintended changes that may not immediately disturb the passage-query pairs.[3] This is especially the case with the sanity sets and the most basic query sets: with these the query is identical to or very closely resembles the passage so that changes to the representation on the passage side will also be in the representation on the query side and hence may not be detected as erroneous by the entailment and contradiction detection.

For the question-answering system, ∼1200 sentences covering basic syntactic and semantic types form a testsuite for representations. The best representation currently produced by the system is stored for the syntax, the linguistic semantics, and the abstract knowledge representation levels. To allow for greater stability over time and less sensitivity to minor feature changes in the rule sets, it is possible to bank only the most important features in the representations may, e.g. the core predicate-argument structure. The banked representations are then compared with the output of the system after any changes are made. Any differences are examined to see whether they are intentional. If they were intended, then new representations need to be banked for the ones that have changed (see (Rosén et al., 2005) for ways to speed up this process by use of discrimants). If the differences were not intended, then the developer knows which constructions were affected by their changes and can more easily determine where in the system the error might have been introduced.

5 Discussion and Conclusions

The testsuites discussed above are continually under development. We believe that the basic ideas behind these testsuites should be applicable to other grammar-based systems used in applications. The passage-query pairs are most applicable to question-answering and search/retrieval systems, but aspects of the approach can apply to other systems.

Some issues that remain for the testsuites discussed above are extending the use of *wh*-questions in passage-query pairs, the division between development and test sets, and the incorporation of context into the testing.

5.1 *Wh*-questions

The testsuites as described have not yet been systematically extended to *wh*-questions. The query

[3]In addition to running regression tests against representations, the syntax, semantics, and abstract knowledge representation have type declarations (Crouch and King, 2008) which help to detect malformed representations.

sets can be easily extended to involve some substitution of *wh*-phrases for arguments and adjuncts in the passage, as in (10).

(10) a. **P:** John broke the box.
Q: Who broke the box?

b. **P:** John broke the box.
Q: What did John break?

c. **P:** John broke the box.
Q: What broke?

d. **P:** John broke the box.
Q: What did John do?

e. **P:** We went to John's party last night.
Q: Who went to John's party?

There is a long-standing issue as to how to evaluate responses to *wh*-questions (see (Voorhees and Tice, 2000a; Voorhees and Tice, 2000b) and the TREC question-answering task web pages for discussion and data). For example, in (10a) most people would agree that the answer should be *John*, although there may be less agreement as to whether *John broke the box.* is an appropriate answer. In (10b) and (10c) there is an issue as to whether the answer should be *box* or *the box* and how to assess partial answers. This becomes more of an issue as the passages become more complicated, e.g. with heavily modified nominals that serve as potential answers. While for (10d) the passage is a good answer to the question, for (10e) presumably the answer should be a list of names, not simply "we". Obtaining such lists and deciding how complete and appropriate they are is challenging. Since most question-answering systems are not constrained to polarity questions, it is important to assess performance on *wh*-questions as the system develops. Other, even more complicated questions, for example *how to* questions are also currently out of the scope of our testsuites.

5.2 Development vs. Testing

For development and evaluation of systems, testsuites are usually divided into development sets, which the system developers examine in detail, and test sets, which represent data unseen by the developers.[4] To a limited extent, the real-world sets serve as a form of test set since they reflect the performance of the system on real data and are often not examined in detail for why any given pair fails to parse. However, the testsuites described above are all treated as development sets. There are no reserved phenomenon-based testsuites for blind testing of the system's performance on each phenomenon, although there are real-world testsuites reserved as test sets.

If a given testsuite was created all at once, a random sampling of it could be held out as a test set. However, since there are often only a few pairs per construction or lexical item, it is unclear whether this approach would give a fair view of system coverage. In addition, for rule-based systems such as the syntax and semantics used in the question-answering system, the pairs are often constructed based on the rules and lexicons as they were being developed. As such, they more closely match the coverage of the system than if it were possible to randomly select such pairs from external sources.

As a system is used in an application, a test set of unseen, application-specific data becomes increasingly necessary. Such sets can be created from the use of the application: for example, queries and returned answers with judgments as to correctness can provide seeds for test sets, as well as for extending the phenomenon-based and real-world development testsuites.

5.3 Context

The real sentences that a question-answering system would use to answer questions appear in a larger textual and metadata context. This context provides information as to the resolution of pronouns, temporal expressions such as *today* and *this morning*, ellipsis, etc. The passage-query pairs in the testsuites do not accurately reflect how well the system handles the integration of context. Small two sentence passages can be used to, for example, test anaphora resolution, as shown in (11).

(11) **P:** Mary hopped. Then, she skipped.
Q: Did Mary skip?
A: YES

Even in this isolated example, the answer can be construed as being UNKNOWN since it is possible, although unlikely, that *she* resolves to some other entity. This type of problem is pervasive in using

[4]The usual division is between training, development, and test sets, with the training set generally being much larger than the development and test sets. For rule based systems, the training/development distinction is often irrelevant, and so a distinction is made between those sets used in the development of the system and those unseen sets used to test and evaluate the system's performance.

simple passage-query pairs for system regression testing.

A further issue with testing phenomena linked to context, such as anaphora resolution, is that they are usually very complex and can result in significant ambiguity. When used on real-world texts, efficiency can be a serious issue which this type of more isolated testing does not systematically explore. As a result of this, the anaphora testsuites must be more carefully constructed to take advantage of isolated, simpler pairs when possible but to also contain progressively more complicated examples that eventually become real-world pairs.

5.4 Summary Conclusions

In complex grammar-based systems, even small changes may have an unforeseeable impact on system performance. Regression testing of the system and its components becomes crucial for the grammar engineers developing the system.

A key part of regression testing is the testsuites themselves, which must be designed to accurately assess coverage and progress and to help to rapidly identify problems. For broad-coverage grammars, such as those used in open domain applications like consumer search and question answering, testsuite design is particularly important to ensure adequate coverage of basic linguistic (e.g. syntactic and semantic) phenomena as well as application specific phenomena (e.g. interpretation of markup, incorporation of metadata).

We described a system of passage-query pairs divided into three types of phenomenon-based testsuites (sanity, query, basic correct). These allow for rapid development and specific coverage assessment. In addition, real-world testsuites allow for overall performance and coverage assessment. More work is needed to find a systematic way to provide "stepping stones" in terms of complexity between phenomenon-based and real-world testsuites.

These testsuites are used in conjunction with the more traditional representation-based regression testsuites used by grammar engineers. These representation-based testsuites use the same phenomenon-based approach in order to assess coverage and pinpoint problems as efficiently as possible.

References

Bobrow, Daniel G., Bob Cheslow, Cleo Condoravdi, Lauri Karttunen, Tracy Holloway King, Rowan Nairn, Valeria de Paiva, Charlotte Price, and Annie Zaenen. 2007. PARC's bridge and question answering system. In King, Tracy Holloway and Emily M. Bender, editors, *Grammar Engineering Across Frameworks*, pages 46–66. CSLI Publications.

Bos, Johan. 2008. Let's not argue about semantics. In *Proceedings of LREC*.

Chatzichrisafis, Nikos, Dick Crouch, Tracy Holloway King, Rowan Nairn, Manny Rayner, and Marianne Santaholma. 2007. Regression testing for grammar-based systems. In King, Tracy Holloway and Emily M. Bender, editors, *Proceedings of the Grammar Engineering Across Frameworks (GEAF07) Workshop*, pages 128–143. CSLI Publications.

Cohen, K. Bretonnel, William A. Baumgartner Jr., and Lawrence Hunter. 2008. Software testing and the naturally occurring data assumption in natural language processing. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 23–30. Association for Computational Linguistics.

Cooper, Robin, Dick Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, and Steve Pulman. 1996. Using the framework. FraCas: A Framework for Computational Semantics (LRE 62-051).

Crouch, Dick and Tracy Holloway King. 2006. Semantics via f-structure rewriting. In Butt, Miriam and Tracy Holloway King, editors, *LFG06 Proceedings*, pages 145–165. CSLI Publications.

Crouch, Dick and Tracy Holloway King. 2008. Type-checking in formally non-typed systems. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 3–4. Association for Computational Linguistics.

Crouch, Dick, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. 2008. XLE documentation. http://www2.parc.com/isl/groups/nltt/xle/doc/.

Lehmann, Sabine, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, Veronika Lux, Judith Klein, Kirsten Falkedal, Frederik Fouvry, Dominique Estival, Eva Dauphin, Hervé Compagnion, Judith Baur, Lorna Balkan, and Doug Arnold. 1996. TSNLP — Test Suites for Natural Language Processing. In *Proceedings of COLING 1996*.

Nerbonne, John, Dan Flickinger, and Tom Wasow. 1988. The HP Labs natural language evaluation tool. In *Proceedings of the Workshop on Evaluation of Natural Language Processing Systems*.

Oepen, Stephan, Klaus Netter, and Judith Klein. 1998. TSNLP — Test Suites for Natural Language Processing. In Nerbonne, John, editor, *Linguistic Databases*, pages 13–36. CSLI.

Oepen, Stephan, Dan Flickinger, Kristina Toutanova, and Chris D. Manning. 2002. LinGO Redwoods. a rich and dynamic treebank for HPSG. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories*, pages 139–149.

Rosén, Victoria, Koenraad de Smedt, Helge Dyvik, and Paul Meurer. 2005. TREPIL: Developing methods and tools for multilevel treebank construction. In *Proceedings of The Fourth Workshop on Treebanks and Linguistic Theories*.

Sekine, Satoshi, Kentaro Inui, Ido Dagan, Bill Dolan, Danilo Giampiccolo, and Bernardo Magnini, editors. 2007. *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. Association for Computational Linguistics, Prague, June.

Voorhees, Ellen and Dawn Tice. 2000a. Building a question answering test collection. In *Proceedings of SIGIR-2000*, pages 200–207.

Voorhees, Ellen and Dawn Tice. 2000b. The TREC-8 question answering track evaluation. In *Proceedings 8th Text REtrieval Conference (TREC-8)*, pages 83–105.

Towards Domain-Independent Deep Linguistic Processing: Ensuring Portability and Re-Usability of Lexicalised Grammars

Kostadin Cholakov[†], **Valia Kordoni**[†‡], **Yi Zhang**[†‡]
† Department of Computational Linguistics, Saarland University, Germany
‡ LT-Lab, DFKI GmbH, Germany
{kostadin,kordoni,yzhang}@coli.uni-sb.de

Abstract

In this paper we illustrate and underline the importance of making detailed linguistic information a central part of the process of automatic acquisition of large-scale lexicons as a means for enhancing robustness and at the same time ensuring maintainability and re-usability of *deep* lexicalised grammars. Using the error mining techniques proposed in (van Noord, 2004) we show very convincingly that the main hindrance to portability of *deep* lexicalised grammars to domains other than the ones originally developed in, as well as to robustness of systems using such grammars is low lexical coverage. To this effect, we develop linguistically-driven methods that use detailed morphosyntactic information to automatically enhance the performance of *deep* lexicalised grammars maintaining at the same time their usually already achieved high linguistic quality.

1 Introduction

We focus on enhancing robustness and ensuring maintainability and re-usability for a large-scale *deep* grammar of German (GG; (Crysmann, 2003)), developed in the framework of Head-driven Phrase Structure Grammar (HPSG). Specifically, we show that the incorporation of detailed linguistic information into the process of automatic extension of the lexicon of such a language resource enhances its performance and provides linguistically sound and more informative predictions which bring a bigger benefit for the grammar when employed in practical real-life applications.

In recent years, various techniques and resources have been developed in order to improve robustness of deep grammars for real-life applications in various domains. Nevertheless, *low coverage* of such grammars remains the main hindrance to their employment in open domain natural language processing. (Baldwin et al., 2004), as well as (van Noord, 2004) and (Zhang and Kordoni, 2006) have clearly shown that the majority of parsing failures with large-scale deep grammars are caused by missing or wrong entries in the lexicons accompanying grammars like the aforementioned ones. Based on these findings, it has become clear that it is crucial to explore and develop efficient methods for *automated (Deep) Lexical Acquisition* (henceforward (D)LA), the process of automatically recovering missing entries in the lexicons of deep grammars.

Recently, various high-quality DLA approaches have been proposed. (Baldwin, 2005), as well as (Zhang and Kordoni, 2006), (van de Cruys, 2006) and (Nicholson et al., 2008) describe efficient methods towards the task of lexicon acquisition for large-scale deep grammars for English, Dutch and German. They treat DLA as a classification task and make use of various robust and efficient machine learning techniques to perform the acquisition process.

However, it is our claim that to achieve better and more practically useful results, apart from good learning algorithms, we also need to *incorporate* into the learning process fine-grained linguistic information which deep grammars inherently include and provide for. As we clearly show in the following, it is **not** sufficient to only develop and use good and complicated classification algorithms. We must look at the detailed linguistic information that is already included and provided for by the grammar itself and try to capture and make as much use of it as possible, for this is the information we aim at learning when performing DLA.

In this way, the learning process is facilitated and at the same time it is as much as possible ensured that its outcome be linguistically more informative and, thus, practically more useful.

We use the GG deep grammar for the work we present in this paper because German is a language with rich morphology and free word order, which exhibits a range of interesting linguistic phenomena, a fair number of which are already analysed in the GG. Thus, the grammar is a valuable linguistic resource since it provides linguistically sound and detailed analyses of these phenomena. Apart from the interesting syntactic structures, though, the lexical entries in the lexicon of the aforementioned grammar also exhibit a rich and complicated structure and contain various important linguistic constraints. Based on our claim above, in this paper we show how the information these constraints provide can be captured and used in *linguistically-motivated* DLA methods which we propose here. We then apply our approach on real-life data and observe the impact it has on the the grammar coverage and its practical application. In this way we try to prove our assumption that the linguistic information we incorporate into our DLA methods is vital for the good performance of the acquisition process and for the maintainability and re-usability of the grammar, as well for its successful practical application.

The remainder of the paper is organised as follows. In Section 2 we show that low (lexical) coverage is a serious issue for the GG when employed for open domain natural language processing. Section 3 presents the types in the lexical architecture of the GG that are considered to be relevant for the purposes of our experiments. Section 4 describes the extensive linguistic analysis we perform in order to deal with the linguistic information these types provide and presents the target type inventory for our DLA methods. Section 5 reports on statistical approaches towards automatic DLA and shows the importance of a good and linguistically-motivated feature selection. Section 6 illustrates the practical usage of the proposed DLA methods and their impact on grammar coverage. Section 7 concludes the paper.

2 Coverage Test with the GG

We start off adopting the automated error mining method described in (van Noord, 2004) for identification of the major type of errors in the GG. As an HPSG grammar, the GG is based on typed feature structures. The GG types are strictly defined within a type hierarchy. The GG also contains constructional and lexical rules and a lexicon with its entries belonging to lexical types which are themselves defined again within the type hierarchy. The grammar originates from (Müller and Kasper, 2000), but continued to improve after the end of the Verbmobil project (Wahlster, 2000) and it currently consists of 5K types, 115 rules and the lexicon contains approximately 35K entries. These entries belong to 386 distinct lexical types.

In the experiments we report here two corpora of different kind and size have been used. The first one has been extracted from the Frankfurter Rundschau newspaper and contains about 614K sentences that have between 5 and 20 tokens. The second corpus is a subset of the German part of the Wacky project (Kilgarriff and Grefenstette, 2003). The Wacky project aims at the creation of large corpora for different languages, including German, from various web sources, such as online newspapers and magazines, legal texts, internet fora, university and science web sites, etc. The German part, named deWaC (Web as Corpus), contains about 93M sentences and 1.65 billion tokens. The subset used in our experiments is extracted by randomly selecting 2.57M sentences that have between 4 and 30 tokens. These corpora have been chosen because it is interesting to observe the grammar performance on a relatively balanced newspaper corpus that does not include so many long sentences and sophisticated linguistic constructions and to compare it with the performance of the grammar on a random open domain text corpus.

The sentences are fed into the PET HPSG parser (Callmeier, 2000) with the GG loaded. The parser has been configured with a maximum edge number limit of 100K and it is running in the *best-only* mode so that it does not exhaustively find all possible parses. The result of each sentence is marked as one of the following four cases:

- *P* means at least one parse is found for the sentence;
- *L* means the parser halted after the morphological analysis and was not able to construct any lexical item for the input token;
- *N* means that the parser exhausted the searching and was not able to parse the sentence;

- *E* means the parser reached the maximum edge number limit and was still not able to find a parse.

Table 1 shows the results of the experiments with the two corpora. From these results it can be seen that the GG has full lexical span for only a small portion of the sentences– about 25% and 10% for the Frankfurter Rundschau and the deWaC corpora, respectively. The output of the error mining confirms our assumption that missing lexical entries are the main problem when it comes to robust performance of the GG and illustrates the need for efficient DLA methods.

	FR		deWaC	
Result	#Sentences	%	#Sentences	%
P	62,768	10.22%	109,498	4.3%
L	464,112	75.55%	2,328,490	90.5%
N	87,415	14.23%	134,917	5.2%
E	3	–	14	–
Total:	614,298	100%	2,572,919	100%

Table 1: Parsing results with the GG and the test corpora

3 Atomic Lexical Types

Before describing the proposed DLA algorithm, we should define what exactly is being learnt. Most of the so called *deep* grammars are strongly lexicalised. As mentioned in the previous section, the GG employs a type inheritance system and its lexicon has a flat structure with each lexical entry mapped onto one type in the inheritance hierarchy. Normally, the types assigned to the lexical entries are maximal on the type hierarchy, i.e., they do not have any subtypes. They provide the most specific information available for this branch of the hierarchy. These maximal types which the lexical entries are mapped onto are called *atomic lexical types*. Thus, in our experiment setup, we can define the lexicon of the grammar as being a one-to-one mapping from word stems to atomic lexical types. It is this mapping which must be automatically learnt (guessed) by the different DLA methods.

We are interested in learning open-class words, i.e., nouns, adjectives, verbs and adverbs. We assume that the close-class words are already in the lexicon or the grammar can handle them through various lexical rules and they are not crucial for the grammar performance in real life applications. Thus, for the purposes of our experiments, we consider only the open-class lexical types. Moreover, we propose an *inventory of open-class lexical types with sufficient type and token frequency*. The **type frequency** of a given lexical type is defined as the number of lexical entries in the lexicon of the grammar that belong to this type and the **token frequency** is the number of words in some corpus that belong to this type.

We use sentences from the Verbmobil corpus which have been treebanked with the GG in order to determine the token frequency and to map the lexemes to their correct entries in the lexicon for the purposes of the experiment. This set contains 11K sentences and about 73K tokens; this gives an average of 6.8 words per sentence. The sentences are taken from spoken dialogues. Hence, they are not long and most of them do not exhibit interesting linguistic properties which is a clear drawback but currently there is no other annotated data compatible with the GG.

We used a type frequency threshold of 10 entries in the lexicon and a token frequency threshold of 3 occurrences in the treebanked sentences to form a list of relevant open-class lexical types. The resulting list contains 38 atomic lexical types with a total of 32,687 lexical entries.

4 Incorporation of Linguistic Features

However, in the case of the GG this type inventory is not a sufficient solution. As already mentioned, in the lexicon of the grammar much of the relevant linguistic information is encoded not in the type definition itself but in the form of constraints in the feature structures of the various types. Moreover, given that German has a rich morphology, a given attribute may have many different values among lexical entries of the same type and it is crucial for the DLA process to capture all the different combinations. That is why we expand the identified 38 atomic lexical type definitions by including the values of various features into them.

By doing this, we are trying to facilitate the DLA process because, in that way, it can ‘learn’ to differentiate not only the various lexical types but also significant morphosyntactic differences among entries that belong to the same lexical type. That gives the DLA methods access to much more linguistic information and they are able to apply more linguistically fine-tuned classification criteria when deciding which lexical type the unknown word must be assigned to. Furthermore, we ensure that the learning process deliver linguistically

Feature	Values	Meaning
SUBJOPT (subject options)	+	in some cases the article for the noun can be omitted
	-	the noun always goes with an article
	+	raising verb
	-	non-raising verb
KEYAGR (key agreement)	–	case-number-gender information for nouns
	c-s-n	underspecified-singular-neutral
	c-p-g	underspecified-plural-underspecified
	...	...
(O)COMPAGR ((oblique)	a-n-g, d-n-g, etc.	case-number-gender information
complement	–	for (oblique) verb complements
agreement	–	case-number-gender of the modified noun (for adjectives)
(O)COMPTOPT ((oblique)	–	verbs can take a different number of complements
complement	+	the respective (oblique) complement is present
options	-	the respective (oblique) complement is absent
KEYFORM	–	the auxiliary verb used for the formation of perfect tense
	haben	the auxiliary verb is ‘haben’
	sein	the auxiliary verb is ‘sein’

Table 2: Relevant features used for type expansion

plausible, precise and more practically useful results. The more the captured and used linguistic information is, the better and more useful the DLA results will be.

However, we have to avoid creating data sparse problems. We do so by making the assumption that not every feature could really contribute to the classification process and by filtering out these features that we consider *irrelevant* for the enhancement of the DLA task. Naturally, the question which features are to be considered relevant arises. After performing an extensive linguistic analysis, we have decided to take the features shown in Table 2 into account.

We have thoroughly analysed each of these features and selected them on the basis of their linguistic meaning and their significance and contribution to the DLA process. The SUBJOPT feature can be used to differentiate among nouns that have a similar morphosyntactic behaviour but differ only in the usage of articles; 4 out of the considered 9 noun atomic lexical types do not define this feature. Furthermore, using this feature, we can also refine our classification within a single atomic lexical type. For example, the entry ‘*adresse-n*’ (address) of the type ‘count-noun-le’[1] has ‘-’ for the SUBJOPT value, whereas the value for the entry ‘*anbindung-n*’ (connection) of the same type is ‘+’:

(1) a. Das hotel hat gute
det.NEUT.NOM hotel have.3PER.SG good
Anbindung an die öffentlichen
connection to det.PL.ACC public
Verkehrsmittel.
transportation means
‘The hotel has a good connection to public transportation.’

b. **Die** *Anbindung* an Rom mit
det.FEM.NOM connection to Rome with
dem Zug ist gut.
det.MASC.DAT train be.3PER.SG good
‘The train connection to Rome is good.’

The distinction between raising and non-raising verbs that this feature expresses is also an important contribution to the classification process.

The case-number-gender data the KEYAGR and (O)COMPAGR features provide allows for a better usage of morphosyntactic information for the purposes of DLA. Based on this data, the classification method is able to capture words with similar morphosyntactic behaviour and give various indications for their syntactic nature; for instance, if the word is a subject, direct or indirect object. This is especially relevant and useful for languages with rich morphology and relatively free word order such as German. The same is also valid for the (O)COMPOPT and KEYFORM features– they allow the DLA method to successfully learn and classify verbs with similar syntactic properties.

The values of the features are just attached to the old type name to form a new type definition. In this way, we ‘promote’ them and these features are now part of the type hierarchy of the grammar which makes them accessible for the DLA process since this operates on the type level. For example, the original type of the entry for the noun ‘abenteuer’ (adventure):

```
abenteuer-n := count-noun-le &
 [ [ --SUBJOPT -,
```

[1]count noun lexeme; all lexical entries in the lexicon end with *le* which stands for lexeme.

```
        KEYAGR c-n-n,
        KEYREL "_abenteuer_n_rel",
        KEYSORT situation,
        MCLASS nclass-2_-u_-e ] ].
```

will become *abenteuer-n := count-noun-le_-_c-n-n* when we incorporate the values of the features SUBJOPT and KEYAGR into the original type definition. The new expanded type inventory is shown in Table 3.

	Original lexicon	Expanded lexicon
Number of lexical types	386	485
Atomic lexical types	38	137
-nouns	9	72
-verbs	19	53
-adjectives	3	5
-adverbs	7	7

Table 3: Expanded atomic lexical types

The features we have ignored do not contribute to the learning process and are likely to create sparse data problems. The (O)COMPFORM ((oblique) complement form) features which denote dependent to verbs prepositions are not considered to be relevant. An example of OCOMPFORM is the lexical entry 'begründen_*mit*-v' (justify with) where the feature has the preposition '*mit*' (with) as its value. Though for German prepositions can be considered as case markers, the DLA has already a reliable access to case information through the (O)COMPAGR features. Moreover, a given dependent preposition is distributed across many types and it does not indicate clearly which type the respective verb belongs to.

The same is valid for the feature VCOPMFORM (verb complement form) that denotes the separable particle (if present) of the verb in question. An example of this feature is the lexical entry '*ab*decken-v' (to cover) where VCOMPFORM has the separable particle '*ab*' as its value. However, treating such discontinuous verb-particle combinations as a lexical unit could help for the acquisition of subcategorizational frames. For example, ***an****hören* (to listen to someone/something) takes an accusative NP as argument, ***zu****hören* (to listen to) takes a dative NP and ***auf****hören* (to stop, to terminate) takes an infinitival complement. Thus, ignoring VCOMPFORM could be a hindrance for the acquisition of some verb types[2].

We have also tried to incorporate some sort of semantic information into the expanded atomic lexical type definitions by also attaching the KEYSORT semantic feature to them. KEYSORT defines a certain situation semantics category ('anything', 'action_sit', 'mental_sit') which the lexical entry belongs to. However, this has caused again a sparse data problem because the semantic classification is too specific and, thus, the number of possible classes is too large. Moreover, semantic classification is done based on completely different criteria and it cannot be directly linked to the morphosyntactic features. That is why we have finally excluded this feature, as well.

Armed with this elaborate target type inventory, we now proceed with the DLA experiments for the GG.

5 DLA Experiments with the GG

For our DLA experiments, we adopted the Maximum Entropy based model described in (Zhang and Kordoni, 2006), which has been applied to the ERG (Copestake and Flickinger, 2000), a wide-coverage HPSG grammar for English. For the proposed prediction model, the probability of a lexical type *t* given an unknown word and its context *c* is:

$$(2) \qquad p(t|c) = \frac{exp(\sum_i \Theta_i f_i(t,c))}{\sum_{t' \in T} exp(\sum_i \Theta_i f_i(t',c))}$$

where $f_i(t,c)$ may encode arbitrary characteristics of the context and Θ_i is a weighting factor estimated on a training corpus. Our experiments have been performed with the feature set shown in Table 4.

Features
the prefix of the unknown word (length is less or equal 4)
the suffix of the unknown word (length is less or equal 4)
the 2 words before and after the unknown word
the 2 types before and after the unknown word

Table 4: Features for the DLA experiment

We have also experimented with prefix and suffix lengths up to 3. To evaluate the contribution of various features and the overall precision of the ME-based unknown word prediction model, we have done a 10-fold cross validation on the Verbmobil treebanked data. For each fold, words that do not occur in the training partition are assumed to be unknown and are temporarily removed from the lexicon.

For comparison, we have also built a baseline model that always assigns a majority type to each

[2]We thank the anonymous reviewer who pointed this out for us.

unknown word according to its POS tag. Specifically, we tag the input sentence with a small POS tagset. It is then mapped to a most popular lexical type for that POS. Table 5 shows the relevant mappings.

POS	Majority lexical type
noun	count-noun-le_- _c-n-f
verb	trans-nerg-str-verb-le_haben-auxf
adj	adj-non-prd-le
adv	intersect-adv-le

Table 5: POS tags to lexical types mapping

Again for comparison, we have built another simple baseline model using the *TnT* POS tagger (Brants, 2000). TnT is a general-purpose HMM-based trigram tagger. We have trained the tagging models with all the lexical types as the tagset. The tagger tags the whole sentence but only the output tags for the unknown words are taken to generate lexical entries and to be considered for the evaluation. The precisions of the different prediction models are given in Table 6.

The baseline achieves a precision of about 38% and the POS tagger outperforms it by nearly 10%. These results can be explained by the nature of the Verbmobil data. The vast majority of the adjectives and the adverbs in the sentences belong to the majority types shown in Table 5 and, thus, the baseline model assigns the correct lexical types to almost every adjective and adverb, which brings up the overall precision. The short sentence length facilitates the tagger extremely, for TnT, as an HMM-based tagger, makes predictions based on the whole sentence. The longer the sentences are, the more challenging the tagging task for TnT is. The results of these models clearly show that the task of unknown word type prediction for deep grammars is non-trivial.

Our ME-based models give the best results in terms of precision. However, verbs and adverbs remain extremely difficult for classification. The simple morphological features we use in the ME model are not good enough for making good predictions for verbs. Morphology cannot capture such purely syntactic features as subcategorizational frames, for example.

While the errors for verbs are pretty random, there is one major type of wrong predictions for adverbs. Most of them are correctly predicted as such but they receive the majority type for adverbs, namely '*intersect-adv-le*'. Since most of the adverbs in the Verbmobil data we are using belong to the majority adverb type, the predictor is biased towards assigning it to the unknown words which have been identified as adverbs.

The results in the top half of the Table 6 show that morphological features are already very good for predicting adjectives. In contrast with adverbs, adjectives occur in pretty limited number of contexts. Moreover, when dealing with morphologically rich languages such as German, adjectives are typically marked by specific affixes corresponding to a specific case-number-gender combination. Since we have incorporated this kind of linguistic information into our target lexical type definitions, this significantly helps the prediction process based on morphological features.

Surprisingly, nouns seem to be hard to learn. Apparently, the vast majority of the wrong predictions have been made for nouns that belong to the expanded variants of the lexical type '*count-noun-le*' which is also the most common non-expanded lexical type for nouns in the original lexicon. Many nouns have been assigned the right lexical type except for the gender:

(3) *Betrieb* (business, company, enterprise)
prediction: *count-noun-le_-_c-n-**n***
correct type: *count-noun-le_-_c-n-**m***

According to the strict *exact-match* evaluate measure we use, such cases are considered to be errors because the predicted lexical type does not match the type of the lexical entry in the lexicon.

The low numbers for verbs and adverbs show clearly that we also need to incorporate some sort of syntactic information into the prediction model. We adopt the method described in (Zhang and Kordoni, 2006) where the disambiguation model of the parser is used for this purpose. We also believe that the kind of detailed morphosyntactic information which the learning process now has access to would facilitate the disambiguation model because the input to the model is linguistically more fine-grained. In another DLA experiment we let PET use the top 3 predictions provided by the lexical type predictor in order to generate sentence analyses. Then we use the disambiguation model, trained on the Verbmobil data, to choose the best one of these analyses and the corresponding lexical entry is taken to be the final result of the prediction process.

As shown in the last line of Table 6, we achieve an increase of 19% which means that in many cases the correct lexical type has been ranked sec-

Model	Precision	Nouns	Adjectives	Verbs	Adverbs
Baseline	37.89%	27.03%	62.69%	33.57%	67.14%
TnT	47.53%	53.76%	74.52%	26.94%	32.68%
ME(affix length=3)	51.2%	48.25%	75.41%	44.06%	44.13%
ME(affix length=4)	54.63%	53.55%	76.79%	47.10%	43.55%
ME + disamb.	73.54%	75%	88.24%	65.98%	65.90%

Table 6: Precision of unknown word type predictors

ond or third by the predictor. This proves that the expanded lexical types improve also the performance of the disambiguation model and allow for its successful application for the purposes of DLA. It also shows, once again, the importance of the morphology in the case of the GG and proves the rightness of our decision to expand the type definitions with detailed linguistic information.[3]

6 Practical Application

Since our main claim in this paper is that for good and *practically useful* DLA, which at the same time may facilitate robustness and ensure maintainability and re-usability of *deep* lexicalised grammars, we do not only need good machine learning algorithms but also classification and feature selection that are based on an extensive linguistic analysis, we apply our DLA methods to real test data. We believe that due to our expanded lexical type definitions, we provide much more linguistically accurate predictions. With this type of predictions, we anticipate a bigger improvement of the grammar coverage and accuracy for the prediction process delivers much more *linguistically relevant* information which facilitates parsing with the GG.

We have conducted experiments with PET and the two corpora we have used for the error mining to determine whether we can improve coverage by using our DLA method to predict the types of unknown words online. We have trained the predictor on the whole set of treebanked sentences and extracted a subset of 50K sentences from each corpus. Since lexical types are not available for these sentences, we have used POS tags instead as features for our prediction model. Coverage is measured as the number of sentences that received at least one parse and accuracy is measured as the number of sentences that received a *correct* analysis. The results are shown in Table 7.

The coverage for FR improves with more than 12% and the accuracy number remains almost the same. Thus, with our linguistically-oriented DLA method, we have managed to increase parsing coverage and at the same time to preserve the high accuracy of the grammar. It is also interesting to note the increase in coverage for the deWaC corpus. It is about 10%, and given the fact that deWaC is an open and unbalanced corpus, this is a clear improvement. However, we do not measure accuracy on the deWaC corpus because many sentences are not well formed and the corpus itself contains much ‘noise’. Still, these results show that the incorporation of detailed linguistic information in the prediction process contributed to the parser performance and the robustness of the grammar without harming the quality of the delivered analyses.

[3]Another reason for this high result is the short average length of the treebanked sentences which facilitates the disambiguation model of the parser.

Parsed Corpus	Coverage	Accuracy
FR with the vanilla version GG	8.89%	85%
FR with the GG + DLA	21.08%	83%
deWaC with the vanilla version GG	7.46%	–
deWaC with the GG + DLA	16.95%	–

Table 7: Coverage results

7 Conclusion

In this paper, we have tackled from a more linguistically-oriented point of view the lexicon acquisition problem for a large-scale deep grammar for German, developed in HPSG. We have shown clearly that missing lexical entries are the main cause for parsing failures and, thus, illustrated the importance of increasing the lexical coverage of the grammar. The target type inventory for the learning process has been developed in a linguistically motivated way in an attempt to capture significant morphosyntactic information and, thus, achieve a better performance and more practically useful results.

With the proposed DLA approach and our elaborate target type inventory we have achieved nearly 75% precision and this way we have illustrated the importance of fine-grained linguistic information for the lexical prediction process. In the end, we have shown that with our linguistically motivated DLA methods, the parsing coverage of the aforementioned deep grammar improves significantly while its linguistic quality remains intact.

The conclusion, therefore, is that it is vital to be able to capture linguistic information and successfully incorporate it in DLA processes, for it facilitates deep grammars and makes processing with them much more robust for applications. At the same time, the almost self-evident portability to new domains and the re-usability of the grammar for open domain natural language processing is significantly enhanced.

The DLA method we propose can be used as an external module that can help the grammar be ported and operate on different domains. Thus, specifically in the case of HPSG, DLA can also be seen as a way for achieving more modularity in the grammar. Moreover, in a future research, the proposed kind of DLA might also be used in order to facilitate the division and transition from a *core deep grammar* with a *core lexicon* towards *subgrammars* with *domain specific lexicons/lexical constraints* in a linguistically motivated way. The use of both these divisions naturally leads to a highly modular structure of the grammar and the system using the grammar, which at the same time helps in controlling its complexity.

Our linguistically motivated approach provides fine-grained results that can be used in a number of different ways. It is a valuable linguistic tool and it is up to the grammar developer to choose how to use the many opportunities it provides.

References

Baldwin, Timothy, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. 2004. Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the Fourth Internation Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal.

Baldwin, Timothy. 2005. Bootstrapping deep lexical resources: Resources for courses. In *Proceedings of the ACL-SIGLEX 2005 Workshop on Deep Lexical Acquisition*, pages 67–76, Ann Arbor, USA.

Brants, Thorsten. 2000. TnT- a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing ANLP-2000*, Seattle, WA, USA.

Callmeier, Ulrich. 2000. PET- a platform for experimentation with efficient HPSG processing techniques. In *Journal of Natural Language Engineering*, volume 6(1), pages 99–108.

Copestake, Ann and Dan Flickinger. 2000. An open-sourse grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC 2000)*, Athens, Greece.

Crysmann, Berthold. 2003. On the efficient implementation of German verb placement in HPSG. In *Proceedings of RANLP 2003*, pages 112–116, Borovets, Bulgaria.

Kilgarriff, Adam and G Grefenstette. 2003. Introduction to the special issue on the web as corpus. *Computational Linguistics*, 29:333–347.

Müller, Stephan and Walter Kasper. 2000. HPSG analysis of German. In Wahlster, Wolfgang, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 238–253. Springer-Verlag.

Nicholson, Jeremy, Valia Kordoni, Yi Zhang, Timothy Baldwin, and Rebecca Dridan. 2008. Evaluating and extending the coverage of HPSG grammars. In *In proceedings of LREC*, Marrakesh, Marocco.

van de Cruys, Tim. 2006. Automatically extending the lexicon for parsing. In Huitink, Janneke and Sophia Katrenko, editors, *Proceedings of the Student Session of the European Summer School in Logic, Language and Information (ESSLLI)*, pages 180–191, Malaga, Spain.

van Noord, Gertjan. 2004. Error mining for wide coverage grammar engineering. In *Proceedings of the 42nd Meeting of the Assiciation for Computational Linguistics (ACL'04), Main Volume*, pages 446–453, Barcelona, Spain.

Wahlster, Wolfgang, editor. 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Artificial Intelligence. Springer.

Zhang, Yi and Valia Kordoni. 2006. Automated deep lexical acquisition for robust open text processing. In *Proceedings of the Fifth International Conference on Language Resourses and Evaluation (LREC 2006)*, Genoa, Italy.

Author Index