Proceedings of the Grammar Engineering Across Frameworks (GEAF07) Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Contents

1 Editor's Note

The papers in this volume came out of the workshop on Grammar Engineering Across Frameworks held at Stanford University in conjunction with the LSA Linguistics Institute in July 2007. The workshop included a panel discussion on evaluation methodologies and metrics, a regular session, and a demo session.

We would like to thank the Department of Linguistics at Stanford and the LSA Institute for assistance both financial and logistical in putting on the workshop. For logistical support, we are particularly grateful to Vivienne Fong, Anubha Kothari, David Hall and Daria Suk. Additional financial support came from Powerset and CSLI, whom we thank for their sponsorship of the workshop.

Our appreciation also to the program committee, who not only selected the papers to be presented but also provided valuable feedback to the authors: Jason Baldridge, Srinivas Bangalore, John Bateman, Miriam Butt, Aoife Cahill, Stephen Clark, Berthold Crysmann, Steffi Dipper, Dan Flickinger, Ron Kaplan, Montserrat Marimon, Owen Rambow, and Jesse Tseng.

Finally, we would like to acknowledge the workshop participants, many of whom traveled to Stanford from far away. The discussions were lively and productive, and we hope to see more venues for exchange within the grammar engineering community about topics of mutual interest.

DotCCG and VisCCG: Wiki and Programming Paradigms for Improved Grammar Engineering with OpenCCG

Jason Baldridge†, Sudipta Chatterjee‡,
Alexis Palmer†, and Ben Wing‡

†Dept. of Linguistics, ‡Dept. of Computer Science

University of Texas at Austin

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

We present a suite of tools for simplifying the creation and maintenance of grammars for the OpenCCG parsing and realization system. The core of our approach relies on a terse but expressive textual format, DotCCG, for declaring CCG grammars. It supports powerful string expansions that allow grammar developers to eliminate redundancy in the declaration of both morphology and category definitions. Grammars written in this format are converted into the XML utilized by OpenCCG using the `ccg2xml` utility, which –like a programming language compiler– provides information regarding errors in the grammar, including the type of error and the line number on which it occurs. DotCCG grammars can be edited with VisCCG, a graphical interface which provides visualization of various components of the grammar and allows local editing of information in a manner inspired by wikis. We also report on resources developed to facilitate wide use of the OpenCCG tool suite presented in this paper and on recent uses of the tools in both academic research and classroom environments.

1 Introduction

A major challenge of grammar engineering is enabling users with little computer experience to create complex grammars. Many users encounter significant obstacles and easily get frustrated by trivial syntax errors and non-intuitive formats. At the same time, more experienced users can feel needlessly constrained by grammar engineering aids designed for novice users. Such frustrations slow users down and can result in a focus on mechanics more than on the grammar itself.

This paper presents two contributions for improving current practice in grammar engineering. First, it provides a terse but expressive format for declaring Combinatory Categorial Grammars (CCG) (Steedman, 2000; Steedman and Baldridge, To appear) that utilizes ideas from software engineering for reducing redundancy in CCG grammars. The basic idea is general enough to be used with other formalisms. Second, it describes a wiki-inspired editing interface, VisCCG, that supports grammar visualization while allowing users to directly edit plain text grammars.

The core motivation for these developments is to improve the grammar development cycle for OpenCCG (`openccg.sf.net`) (Hockenmaier et al., 2004; Baldridge and Kruijff, 2002; White and Baldridge, 2003), a parsing and realization system that uses CCG, and to provide a model for facilitating grammar development for both novice and expert grammar writers. OpenCCG has long lacked such an environment despite its use in a number of projects. Grammars developed with VisCCG are compiled into OpenCCG's native XML format, much in the same

[†]We would like to thank Emily Bender, Fred Hoyt, Geert-Jan Kruijff, Mark Steedman, Michael White, students in Jason Baldridge's categorial grammar, computational syntax, and computational linguistics courses at UT Austin in 2006/7, and the participants of the GEAF 2007 workshop for valuable feedback. This research was supported by a Liberal Arts Instructional Technology Grant from the University of Texas at Austin.

manner as wiki pages produce HTML. The goal is to create a grammar engineering environment for CCG that is both easy to *learn* to use and easy to use.

We begin by motivating our work in the context of OpenCCG as well as other grammar engineering platforms. In section 4 we then briefly introduce CCG and OpenCCG and some of the problems with OpenCCG's native XML grammar format. Section 5 discusses DotCCG, followed by an extensive discussion of its parameterized macro mechanisms in section 6. Then we present VisCCG and conclude with a brief discussion of uses of our tools and resources for developing OpenCCG grammars.

2 Motivation

A graphical user interface (GUI) was developed for Grok, OpenCCG's predecessor, but development was ceased as the parsing system itself was improved (see Bierner (2001) and Baldridge (2002) for specific reference to Grok). Developing grammars for OpenCCG has since involved working with unwieldy XML specifications. Our work was initiated to address this (rather large) gap in CCG grammar development.[1] Several aspects of our approach are novel and may be useful in the context of work in other formalisms and/or grammar engineering environments.

The schism between computational definitions and the grammar they are supposed to express has been addressed in various ways, with visualization being a common strategy for more intuitive representations of the grammar. One approach is to develop a GUI for editing objects such as trees and feature structures, such as that of the XTAG system (Doran et al., 2000). The XTAG system included a graphical tree-drawing editor which allowed the user to attach features and labels to nodes of a tree. In such systems, grammar developers usually do not work with the underlying code. A high-level approach like that of the XTAG tree editor is friendly for novice users but can be frustratingly restrictive for experienced users.

An alternative is to develop grammars by working with a low-level format and then visualizing them with a separate GUI which *displays* information. For example, the LKB system (Copestake, 2002) provides extensive, highly configurable displays of various components of grammars written in the Type Description Language. The display functionality in the XLE system for grammar development in the Lexical-Functional Grammar framework (Butt et al., 1998) is similarly informative and configurable. In such systems, however, the developer cannot directly edit the grammar using the GUI. Instead, the plain text grammar is edited and then reloaded to view the effect of the modifications in the graphical representation.

An interesting compromise between visualization and low-level specification can be observed with the use of wikis for creating web content. HTML and XML are cumbersome and unintuitive formats; wiki notation as an alternative has en-

[1] Concurrently with our work, Scott Martin and Michael White at Ohio State University developed a complementary tool called `grammardoc` which produces a set of HTML pages for visualizing OpenCCG grammars. Both `grammardoc` and our tools are distributed with the OpenCCG system.

1	`pay **close** attention`	wiki syntax
2	`pay <b>close</b> attention`	HTML syntax
3	pay **close** attention	display

Figure 1: Wiki-style notation as shorthand for HTML

abled lay users to create web content quickly and effectively. For example, in one common wiki syntax, boldfaced text is indicated with double asterisks around the text. This shorthand (Figure 1, line 1) is then converted into HTML (line 2) and displayed as boldfaced text (line 3). Wikis also make it easy to edit small portions of documents while visualizing the rest, and they provide immediate feedback on the visual outcome of edits. DotCCG provides a similar shorthand notation for OpenCCG's XML, and VisCCG provides user-friendly visualization and editing.

Software engineering provides another source of ideas for improving grammar engineering. Most grammar specifications can be viewed as programming languages particularized to natural language, yet grammar platforms typically do not provide much support for error checking and error messages. Our `ccg2xml` utility compiles DotCCG to OpenCCG's XML and supports such checking in the process, while VisCCG provides feedback in real-time (during editing).

Integrated Development Environments (IDEs) for programming languages can be used to improve productivity for many developers. A key property of IDEs is that they are optional – a developer may use a plain text editor to write programs if they wish. We see VisCCG in this light. It is particularly useful for those who are creating their first grammars. In the classroom setting, we observed that users with less experience working with computers tend to stick with editing their grammars using VisCCG, but many others –particularly those with programming experience– switch over to their favorite text editor (e.g. Emacs or Vi) once they understand the DotCCGformat. The latter would still periodically load their grammars in VisCCG. We see this availability of choice as a highly desirable feature of the new tools we have developed for OpenCCG: the DotCCG format, `ccg2xml`, and VisCCG.

3 Combinatory Categorial Grammar

CCG is a lexicalized grammar formalism that has attracted both linguistic and computational interest. It has a universal rule component that drives the combination of categories and their semantics to provide compositional analyses for sentences. Categories may be either atomic elements or (curried) functions which specify the canonical linear direction in which they seek their arguments. Some simplified example lexical entries are given below:

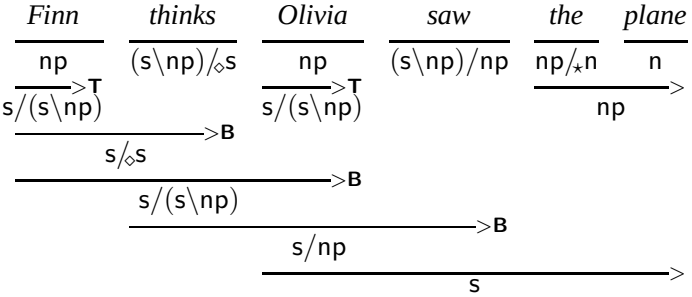
Olivia := np	*the* := $\mathsf{np}/_{\star}\mathsf{n}$
Finn := np	*saw* := $(\mathsf{s}\backslash\mathsf{np})/\mathsf{np}$
plane := n	*thinks* := $(\mathsf{s}\backslash\mathsf{np})/_{\diamond}\mathsf{s}$

The most basic rules are forward ($>$) and backward ($<$) application. CCG also utilizes rules based on the composition (**B**), type-raising (**T**), and substitution (**S**) combinators of combinatory logic. The rules of CCG are:[2]

($>$)	$X/_{\star}Y \quad Y \Rightarrow X$	($<$)	$Y \quad X\backslash_{\star}Y \Rightarrow X$
($>\mathbf{B}$)	$X/_{\diamond}Y \quad Y/_{\diamond}Z \Rightarrow X/_{\diamond}Z$	($<\mathbf{B}$)	$Y\backslash_{\diamond}Z \quad X\backslash_{\diamond}Y \Rightarrow X\backslash_{\diamond}Z$
($>\mathbf{B}_{\times}$)	$X/_{\times}Y \quad Y\backslash_{\times}Z \Rightarrow X\backslash_{\times}Z$	($<\mathbf{B}_{\times}$)	$Y/_{\times}Z \quad X\backslash_{\times}Y \Rightarrow X/_{\times}Z$
($>\mathbf{T}$)	$X \Rightarrow Y/(Y\backslash X)$	($<\mathbf{T}$)	$X \Rightarrow Y\backslash(Y/X)$

Each rule is keyed to a modality; this allows lexical items to selectively utilize some rules but not others. For example, the $/_{\star}$ slash on the category for *the* keeps the composition rules from causing ungrammatical word order permutations within English noun phrases. See Baldridge (2002) and Baldridge and Kruijff (2003) for full explication of the computational and linguistic significance of modalities.

Though the application rules do the majority of the work, the others are crucial for building the non-standard constituents for which categorial grammars are well-known. With these rules and the categories given above, we can provide an incremental derivation for a sentence such as 'Finn thinks Olivia saw the plane':



The constituent s/np derived above for 'Finn thinks Olivia saw' is also used in analyses for relative clauses like 'the plane that [Finn thinks Olivia saw]' and right-node raising sentences like '[Kestrel heard] and [Finn thinks Olivia saw] the plane'.

There has been a great deal of work in computational linguistics using CCG over the past two decades, and there is an even greater degree of activity in recent years. A major development was the creation of CCGbank (Hockenmaier and Steedman, 2007), which has allowed the creation of fast and accurate probabilistic CCG parsers for producing deep dependencies (Hockenmaier, 2003; Bos et al., 2004; Clark and Curran, 2007). CCG has also been used to induce semantic parsers from sentences paired with logical forms (Zettlemoyer and Collins, 2007).

Work with OpenCCG represents another major branch of CCG research. It is used for testing and developing syntactic and semantic analyses (Bierner, 2001; Baldridge, 2002; Kruijff and Baldridge, 2004; Gerstenberger and Wolksa, 2005) and for research into CCG parsing and realization (Hockenmaier et al., 2004; White and Baldridge, 2003; White, 2006b; White et al., 2007). It performs parsing/realization in the systems of a number of projects, many of which are given in Figure 2. Most of these are dialog systems, including natural language interfaces for robots (CoSy, JAST, and INDIGO) and MP3 systems (SAMMIE).

[2]We exclude substitution here for space reasons. An example is $>\mathbf{S}$: $(X/_{\diamond}Y)/_{\diamond}Z \quad Y/_{\diamond}Z \Rightarrow X/_{\diamond}Z$.

Project	References/Website
AdaRTE	Rojas-Barahona (2007) http://www.labmedinfo.org/research/adarte/adarte.htm
COMIC	Foster and White (2005, 2007); Nakatsu and White (2006); White (2006a) http://www.hcrc.ed.ac.uk/comic/
CoSy	Kruijff et al. (2007) http://www.cognitivesystems.org
CrAg	Isard et al. (2006) http://www.hcrc.ed.ac.uk/crag/
DIALOG	Wolska and Kruijff-Korbayová (2004); Benzmüller et al. (2007) http://www.ags.uni-sb.de/~dialog/
FLIGHTS	Moore et al. (2004)
INDIGO	http://www.ics.forth.gr/indigo/
JAST	Rickert et al. (2007) http://www.euprojects-jast.net/
Methodius	Isard (2007) http://www.ltg.ed.ac.uk/methodius/
SAMMIE	Becker et al. (2006) http://www.talk-project.org

Figure 2: Example projects that use OpenCCG for parsing and realization.

4 OpenCCG's XML Format

The underlying native specification format of OpenCCG is XML. Grammatical information is split across six interdependent files, some of which define components that were directly inspired by XTAG (Doran et al., 2000). Each file defines a major component of the grammar, including (a) a structured lexicon containing families of lexical entries, (b) a morphological database pairing words with their stems and morphological features, (c) morphological macros instantiating feature values on lexical entries, (d) a hierarchy of typed features, (e) a set of parameterized CCG rules, and (f) a testbed of sentences used for simple regression testing.

As an example of what is involved in creating lexical entries in OpenCCG, Figure 3 shows a fragment of the XML lexicon, morphology, and typed-feature files for an Ojibwe[3] grammar. This fragment defines a noun family that has a single lexical category, which contains three lexical items: *gaago* 'porcupine', *kwe* 'woman', and *mzinig* 'book'. Each lexical item inflects with four forms: singular proximate, singular obviative, plural proximate, and plural obviative. The inflectional suffixes vary according to the stem. *Gaago* and *kwe* are of animate gender, while *mzinig* is inanimate. A basic feature hierarchy is defined, consisting of person (2nd, 1st, 3rd, non3rd), number (singular, plural), gender (animate, inanimate), and obviation status (proximate, obviative). Note that the majority of the XML for defining the feature hierarchy has been truncated for space reasons.

Developing grammars directly in XML is time-consuming and error prone. XML was designed as a format to standardize communication of data among computers, not for direct editing by humans. Furthermore, OpenCCG's XML format contains many redundancies and interdependencies, leading to errors when a change is made in one place and not propagated elsewhere. For example, the association between the part of speech `N` and the three lexical items is declared in the lexicon file and in multiple places throughout the morphology file. The declarations of multiple inflected forms of the same stem are also highly repetitive and fail to express any generalizations over the forms. Finally, the features attached to

[3]Ojibwe is an Algonquian language of the upper Great Lakes region and southeastern Ontario.

Ojibwe lexicon file

```
<family name=‘‘N’’ pos=‘‘N’’ closed=‘‘true’’>
  <entry name=‘‘Entry-1’’>
    <atomcat type=‘‘n’’>
      <fs id=‘‘1’’>
        <feat attr=‘‘index’’>
          <lf>
            <nomvar name=‘‘X’’/>
          </lf>
        </feat>
      </fs>
      <lf>
        <satop nomvar=‘‘X’’>
          <prop name=‘‘[‘DEFAULT’]’’/>
        </satop>
      </lf>
    </atomcat>
  </entry>
  <member stem=‘‘mzinigna’’/>
  <member stem=‘‘gaago’’/>
  <member stem=‘‘kwe’’/>
</family>
```

Ojibwe morphology file

```
<entry word=‘gaago’ macros=‘@3rd @sg @prox @anim’ pos=‘N’ stem=‘gaago’/>
<entry word=‘gaagon’ macros=‘@3rd @sg @obv @anim’ pos=‘N’ stem=‘gaago’/>
<entry word=‘gaagog’ macros=‘@3rd @pl @prox @anim’ pos=‘N’ stem=‘gaago’/>
<entry word=‘gaagong’ macros=‘@3rd @pl @obv @anim’ pos=‘N’ stem=‘gaago’/>
<entry word=‘mzinigna’ macros=‘@3rd @sg @prox @inan’ pos=‘N’ stem=‘mzinig’/>
<entry word=‘mzinignan’ macros=‘@3rd @sg @obv @inan’ pos=‘N’ stem=‘mzinig’/>
<entry word=‘mzinignag’ macros=‘@3rd @pl @prox @inan’ pos=‘N’ stem=‘mzinig’/>
<entry word=‘mzinignang’ macros=‘@3rd @pl @obv @inan’ pos=‘N’ stem=‘mzinig’/>
<entry word=‘kwe’ macros=‘@3rd @sg @prox @anim’ pos=‘N’ stem=‘kwe’/>
<entry word=‘kwewan’ macros=‘@3rd @sg @obv @anim’ pos=‘N’ stem=‘kwe’/>
<entry word=‘kwen’ macros=‘@3rd @pl @prox @anim’ pos=‘N’ stem=‘kwe’/>
<entry word=‘kwenwan’ macros=‘@3rd @pl @obv @anim’ pos=‘N’ stem=‘kwe’/>

<macro name="@anim">
  <fs id="1" attr="GEND" val="anim"/>
</macro>
<macro name="@inan">
  <fs id="1" attr="GEND" val="inan"/>
</macro>
...
```

Ojibwe typed-feature file

```
  <type name="GEND"/>
  <type name="anim" parents="GEND"/>
  <type name="inan" parents="GEND"/>
  <type name="OBV"/>
  <type name="prox" parents="OBV"/>
  <type name="obv" parents="OBV"/>
...
```

Figure 3: XML specifying an Ojibwe noun family containing three lexical items.

```
feature {
  gend<1>: anim inan;
  pers<1>: 1st 2nd 3rd;
  num<1>: sg pl;
  obv<1>: prox obv;
}

family N {
  entry: n<1>[X]: X(*);
}

def noun(stem, obv-end, pl-end, gend) {
  word stem:N {
    stem: 3rd sg prox gend;
    stem.obv-end: 3rd sg obv gend;
    stem.pl.end: 3rd pl prox gend;
    stem.obv-end.pl-end: 3rd pl obv gend;
  }
}

noun(gaago, n, g, anim)
noun(mzinigna,n, g, inan)
noun(kwe, wan, n, anim)
```

Figure 4: DotCCG equivalent of the Ojibwe XML fragment given in Figure 3.

inflected forms need to be declared both in the morphology and typed-feature files.

5 DotCCG: shorthand for OpenCCG

DotCCG was created to overcome the deficiencies of direct XML input of grammars.[4] It is a human-friendly format which seeks to eliminate redundancy and boost expressiveness while requiring far fewer lines of code than raw XML. It was designed to be concise, flexible, and easy to use, and specifically intended for direct input and editing using a text editor. The grammar is placed in a single `.ccg` file, with declarations in any order and freely grouped or separated. All of the XML required by OpenCCG is generated by passing the `.ccg` file through `ccg2xml`, a program written in Python and implemented using PLY.[5] Handling the dependencies in this way greatly reduces the burden on the grammar developer and increases the grammar's modularity and maintainability. Figure 4 shows the full DotCCG equivalent of the Ojibwe XML fragment.

DotCCG was designed with an emphasis on making the grammar specification language as tolerant and expressive as possible. The general feel of DotCCG syntax is like C, Java, or Perl. However, the syntax is very forgiving on the usage of commas, semicolons, and other terminators and separators. In fact, this punctuation can

[4] An existing solution using XSLT transformations is available (Bozşahin et al., 2006) but requires significant technical expertise.

[5] PLY, available at `http://www.dabeaz.com/ply/`, is a package that provides functionality equivalent to Lex and YACC.

be omitted as long as no syntactic ambiguity will result.[6] This eliminates one of the major stumbling blocks grammar engineers typically face when adjusting to an unfamiliar format. Although DotCCG looks similar to a traditional programming language, the format is intended for use by non-programmers as well as programmers. Its semantics are on a higher level than most programming languages, and it consistently favors expressiveness and ease-of-use over rigid formatting. It is lenient in its handling of commas and other punctuation, and most syntactic elements can be omitted if not needed, with sensible default behavior.

The five sections of DotCCG grammars are described below. Each section is implemented within the `.ccg` file with a series of declarations.

Features — Declaring features allows for simple specification of and reference to features in lexical entries and categories. For example, the Ojibwe grammar fragment shown above creates a simple feature structure with person, number, gender and obviation features. The character in angle brackets following the name of the feature is required by OpenCCG and relates to its mechanism for unifying feature values across lexical categories. Features in DotCCG can also be nested and allow for multiple inheritance.

Words — Word declarations associate lexical items with particular categories and features as well as specifying morphological information. The following are two examples for English, one showing a simple word `the` of family `Det`, and the other showing a pseudo-word `pro1` of family `Pro` and semantic class `animate`, with various surface realizations according to case and number:

(1)
```
word the:Det;
word pro1:Pro(animate) {
  I: 1st sg nom;
  me: 1st sg acc;
  we: 1st pl nom;
  us: 1st pl acc; }
```

Word declarations are commonly placed inside of expansions, as in the `noun` expansion in the Ojibwe fragment. See section 6 for further discussion.

Rules — This section specifies the rules allowed or disallowed in the particular grammar. The CCG rules enabled by default are the forward and backward varieties of application, harmonic composition, and crossed composition. Substitution rules must be invoked explicitly. OpenCCG supports the modalities of Baldridge and Kruijff (2003), so the applicability of the rules is controlled by the use of these modalities on slashes in categories.

Type-raising can be invoked and restricted to particular argument and result categories. For example, the following declaration adds the rule $\mathsf{np} \Rightarrow \mathsf{s\$}/(\mathsf{s\$}\backslash\mathsf{np})$:

(2)
```
typeraise + $: np => s;
```

Type-changing rules can also be added. The following would be one way of implementing pro-drop in a grammar ($\mathsf{s}_{\mathsf{fin}}\backslash\mathsf{np}_{\mathsf{nom}}$ changes to $\mathsf{s}_{\mathsf{fin}}$):

[6] The only situation where separators are required occurs in arguments to textual expansions, which can consist of arbitrary text.

(3)
```
typechange: s[finite]\np[nom] => s[finite] ;
```

Lexicon/Categories — Lexical families consist of one or more category declarations and optional specification of lexical items which are members of that family. For example, in English the lexical family `Det` has just a single category: $\mathtt{np}/_{\diamond}\mathtt{n}$. The family for dative alternation verbs, though, has two possible categories, one for the double object construction and one for the pp-complement construction.

There are two types of intransitive verbs in Ojibwe, those with an animate subject (`VAI`) and those with an inanimate one (`VII`). The category declarations for these two families are shown below.[7] Features are enclosed in square brackets, and the final term, after the second colon, is the semantic representation.

(4)
```
family VAI {
  entry: s<8>[E] | n<1>[anim X]: E:action (* <actor>X:sem-obj); }
family VII {
  entry: s<8>[E] | n<1>[inan X]: E:action (* <actor>X:sem-obj); }
```

Testbed — The testbed contains a list of constructions and the number of parses the grammar is expected to find for each construction. The testbed facility provides for simple regression testing, e.g. whether the expected number of parses are obtained and whether sentences can be reverse realized from their parse results.[8]

(5)
```
testbed {
  wiisniwag gaagog: 1;     ## the porcupines eat
  wiisniwag mzinignan: 0;  ## *the books eat }
```

6 Expansions with DotCCG

6.1 Introduction to expansions

Most grammar engineering systems provide mechanisms to reduce redundancy. These support the expression of various levels of generalization while providing power and flexibility. For example, XLE has macros and parameterized rules, and the LKB uses types to capture lexical and syntactic regularities. DotCCG offers parameterized string-rewrite functions that we call **expansions**.

We chose expansions as our primary abstraction mechanism because they are flexible and easy to use. The definitions directly specify their expansions and mirror what will be inserted and processed when an expansion call is made. The lack of a need to "program" data makes expansions easy to use for non-programmers. Furthermore, expansions can abstract over *any* portion of a text, regardless of whether such a usage was anticipated in the initial design of the grammar. A programmed mechanism, by contrast, either has to impose a uniform structure on all specifications or have separate mechanisms to handle each type of structure.

[7] The numbers in angle brackets represent the feature structure ID assigned to the category. These are global for the grammar: this is one of the main weaknesses of OpenCCG grammar specification.

[8] The sentences given here are not surface forms but rather idealizations of Ojibwe sentences prior to phonological processes.

Our expansions are quite similar to XLE macros and parameterized rules, but with greater syntactic flexibility, fewer constraints, and increased string manipulation capabilities. The expansions allow DotCCG to handle quite complex morphology without having to interface with external morphological analyzers. Of course, there are many advantages to interfacing with existing tools such as morphological analyzers, and XLE grammars have been successfully interfaced with finite-state analyzers (Kaplan et al., 2004). Along with the flexible syntax, of course, comes a reduced level of control over expansions, for good and for ill. Unlike XLE, for example, no error occurs if not all input arguments appear in the output specified for the expansion. While this may allow a user to write expansions with unexpected consequences, it gives the expansions a broader range of possible functionalities.

A disadvantage to our solution is that expansions are a meta-theoretic construct and as such are not visible in the underlying grammar framework itself. By the time OpenCCG sees the grammar, all expansions have taken place, and there is no record of how the expanded structures were constructed. Thus, it may be hard to debug a problem occurring in a group of deeply nested expansions,[9] and injudicious use of expansions can lead to quite obfuscated code.

A simplified version of an expansion contained in Figure 4 is given in (6). It defines a parameterized expansion named `noun`, with two formal parameters `stem` and `gend`. Calling this expansion with `noun(gaago, anim)` produces the expanded text given in (7).

(6)
```
def noun(stem, gend) {
  word stem:N {
    stem: 3rd sg prox gend;
    stem.n: 3rd sg obv gend;
    stem.g: 3rd pl prox gend;
    stem.ng: 3rd pl obv gend;
}}
noun(gaago, anim)
```

(7)
```
word gaago:N {
     gaago: 3rd sg prox anim;
     gaagon: 3rd sg obv anim;
     gaagog: 3rd pl prox anim;
     gaagong: 3rd pl obv anim;
}
```

Occurrences of formal parameters inside of the expanded text have been replaced with their actual values, and strings separated by a period have been concatenated.

6.2 Nested expansions for complex morphology

Expansions in conjunction with word declarations make it easy to express arbitrarily complicated morphology. They are used extensively in DotCCG grammars. Expansions can be nested inside of each other without restriction, allowing almost any pattern of syncretism to be factored out with little or no repetition.

As an example, a large fragment of Classical Arabic, including all noun, verb, adjective and pronoun morphology and correct handling of resumptive pronouns in relative clauses, was implemented in an 800-line `.ccg` file (about 20% of which is comments). It produces a vocabulary with more than 1100 words. The following portion shows how some of the complexities of present-tense verbs can be handled:

[9] To help alleviate this, `ccg2xml` provides options to debug expansion problems, such as displaying the text after expansion processing.

```
# Arabic verb fragment.  We are omitting a great deal: dual number,
# jussive mood, all past tense forms, doubled verbs, etc.

# All present-tense verbs can be reduced to four forms (five, counting the
# dual), plus prefixes.

def gen-pres(mood, fsing, fsing-fem, fplur-masc, fplur-fem) {
  # A special phonological rule collapses adjacent glottal stops: e.g.
  # _a_kulu -> _aakulu.  We implement using regsub() -- see below.
  _ . regsub('^([aiu])_', '\1\1', fsing): pres, mood, 1st, sg;
  t.fsing: pres, mood, 2nd, m, sg;
  t.fsing-fem: pres, mood, 2nd, f, sg;
  y.fsing: pres, mood, 3rd, m, sg;
  t.fsing: pres, mood, 3rd, f, sg;

  n.fsing: pres, mood, 1st, pl;
  t.fplur-masc: pres, mood, 2nd, m, pl;
  t.fplur-fem: pres, mood, 2nd, f, pl;
  y.fplur-masc: pres, mood, 3rd, m, pl;
  y.fplur-fem: pres, mood, 3rd, f, pl;
}

# Most verbs can be reduced to two stems (one for feminine plural and one
# for all other cases), with a specific set of endings, which vary between
# indicative and subjunctive.

def two-form-pres-indic(formv, formc) {
  gen-pres(indic, formv.u, formv.iina, formv.uuna, formc.na)
}
def two-form-pres-subj(formv, formc) {
  gen-pres(subj, formv.a, formv.ii, formv.uu, formc.na)
}

# The basic Arabic verb conjugations are strong, second-weak, doubled, and
# third-weak.  Strong verbs have one stem, while second-weak and doubled
# (not included here) have two.  Second-weak verbs have many subtypes, so
# we require that each verb give both stems.

def strong-pres(form) {
  two-form-pres-indic(form, form)
  two-form-pres-subj(form, form)
}
def 2nd-weak-pres(formv, formc) {
  two-form-pres-indic(formv, formc)
  two-form-pres-subj(formv, formc)
}

# Third-weak verbs merge stem and endings, and have three subtypes, ending
# in -aa, -ii, or -uu in the base form.

def 3rd-weak-pres-aa(form) {
  gen-pres(indic, form.aa, form.ayna, form.awna, form.ayna)
  gen-pres(subj, form.aa, form.ay, form.aw, form.ayna)
}
def 3rd-weak-pres-ii(form) { ... }       # Omitted to save space
def 3rd-weak-pres-uu(form) { ... }       # Omitted to save space

# Here we provide expansions for the various conjugations. (These are
# appropriate for a full verb paradigm, including both present and past
# tense, but the past-tense expansion has been commented out.) Each lexical
# entry specifies the past-tense stem (which is used to form the verb's
# "dictionary form"), some properties (valency and English translation), a
```

```
# present-tense stem, and any other required info.  Second-weak verbs have
# two stems for each of present and past, while third-weak verbs specify
# the past (ay/aw/ii) and present (ii/uu/aa) subtypes.

def strong-verb(past, props, pres) {
  word past: props {
    strong-pres(pres)
  }
}
def 2nd-weak-verb(pastv, props, pastc, presv, presc) {
  word pastv: props {
    2nd-weak-pres(presv, presc)
  }
}
def 3rd-weak-verb(past_stem, props, past_type, pres_stem, pres_type) {
  word past_stem . past_type: props {
  # Note how we are dynamically constructing the expansion call!
    3rd-weak-pres- . pres_type(pres_stem)
  }
}

# Here we declare the actual verbs.  These are identical to how they appear
# in the full grammar, where each one expands to 52 individual forms.

strong-verb(katab, TransV(pred=write), aktub)
2nd-weak-verb(kaan, TransV(pred=be), kun, akuun, akun)
3rd-weak-verb(_a9T, DitransV(pred=give), ay, u9T, ii)
```

Note that Arabic verbs are formed in a complex fashion involving prefixes, suffixes, and internal stem changes. In general, there are different stems for past and present, and many verbs have two stems in each tense. The endings also vary in complicated ways among different moods and classes. By the judicious use of nested expansions, however, we can reduce each lexical entry down to a very small size, where only the class and underivable stem forms are given. The following table shows the indicative and subjunctive moods generated for the three sample verbs: *kataba* 'write' (strong verb), *kaana* 'be' (2nd-weak verb; note the short vowel in *yakunna*), and *'a9Taa* 'give' (3rd-weak verb).

	kataba.IND	kataba.SBJ	kaana.IND	kaana.SBJ	'a9Taa.IND	'a9Taa.SBJ
1sg	'aktubu	'aktuba	'akuunu	'akuuna	'a9Taa	'a9Taa
2sg.m	taktubu	taktuba	takuunu	takuuna	ta9Taa	ta9Taa
2sg.f	taktubiina	taktubii	takuuniina	takuunii	ta9Tayna	ta9Tay
3sg.m	yaktubu	yaktuba	yakuunu	yakuuna	ya9Taa	ya9Taa
3sg.f	taktubu	taktuba	takuunu	takuuna	ta9Taa	ta9Taa
1pl	naktubu	naktuba	nakuunu	nakuuna	na9Taa	na9Taa
2pl.m	taktubuuna	taktubuu	takuunuuna	takuunuu	ta9Tawna	ta9Taw
2pl.f	taktubna	taktubna	takunna	takunna	ta9Tayna	ta9Tayna
3pl.m	yaktubuuna	yaktubuu	yakuunuuna	yakuunuu	ya9Tawna	ya9Taw
3pl.f	yaktubna	yaktubna	yakunna	yakunna	ya9Tayna	ya9Tayna

6.3 Expansions and built-in functions

Expansions are made even more powerful by three built-in expansion functions, which provide the full power of regular-expression matching and replacement. **`regsub`(PATTERN, REPLACEMENT, TEXT)** returns TEXT, but with all occurrences of PATTERN (a regular expression) replaced with REPLACEMENT (a standard regular expression substitution expression, including backreferences

to captured text). **`ifmatch(PATTERN, TEXT, IF-TRUE, IF-FALSE)`** matches regular expression PATTERN against TEXT, returning IF-TRUE if it matches and IF-FALSE otherwise. **`ifmatch-nocase`** functions similarly, but the matching is case-insensitive.

An example of the usage of these functions is computing English plurals:

(8)
```
def pluralize(Word) {
 ifmatch('^.*[aeiou][oy]\$', Word, Word . s,
  ifmatch('^.*([sxoy]|sh|ch)\$', Word,
    regsub('^(.*)y\$', '\1i', Word) . es,
       Word . s))}
```

This definition handles both *-s* and *-es* endings, including words ending with *-y*. It will correctly map *cat, box, boy, lady* into *cats, boxes, boys, ladies*, respectively.

Expansions in combination with `regsub` can also be used to handle complex cases such as infixation in Tagalog, where verbs can take on a number of different voice affixes that single out a particular participant in an event (Kroeger, 1993). For example, the stem *bili* 'buy' can take the inflected forms *bumili* (actor), *binili* (object), *binilhan* (dative), *ipinambili* (instrumental), *ibinili* (benefactive), and *kabibili* (recent-perfective). The following DotCCG fragment demonstrates this, breaking the stem into two parts to allow for infixation and using `regsub` to handle reduplication in *kabibili* and the deletion of *i* and insertion of *h* in *binilhan*:[10]

(9)
```
def reduplicate (Word) { regsub('^(..)(.*)$', '\1\1\2', Word) }

def regular_verb (St1, St2, LF) {
  St1 . um . St2                                :VerbAV (pred=LF);
  St1 . in . St2                                :VerbOV (pred=LF);
  St1 . in . regsub('^(.*)i$', '\1h', St2) . an :VerbDV (pred=LF);
  ipinam . St1 . St2                            :VerbIV (pred=LF);
  i . St1 . in . St2                            :VerbBV (pred=LF);
  ka . reduplicate(St1 . St2)                   :VerbRP (pred=LF);
}

regular_verb (b, ili, buy);
```

6.4 Expansions for inheritance-like effects

In grammar engineering, inheritance is often used to eliminate redundancy by allowing partial definitions to be used as a base upon which further definitions are built. Inheritance (including defaults) is in fact one of the core aspects of the LKB system (in that it uses the Type Description Language) which allows complex linguistic signs to be built elegantly with a series of incremental declarations using inheritance. Villavicencio (2002) utilizes inheritance in the LKB to create a categorial grammar which defines the transitive verb and sentential complement categories as extensions of the intransitive verb category, ditransitives as extensions of transitives, and so on.

[10] Tagalog verbal morphology in general is of course much more complex than for this one stem, but this shows in principle how such patterns can be captured.

OpenCCG does not provide support for inheritance in general, but the XML format does provide special declarations to allow the inheritance patterns used by Villavicencio (Baldridge, 2002). Interestingly, expansions provide an alternative way to achieve this effect:

(10)
```
def iv_cat (PostSyn, MoreSem) {
  s[E] \ np[X nom] PostSyn: E(* <Subject>X MoreSem)
}
def tv_cat (PreSyn, PostSyn, MoreSem) {
  iv_cat(PreSyn / np[Y acc] PostSyn, <DirectObject>Y MoreSem)
}
family IntransV(V) {
  entry: iv_cat(,);
}
family TransV(V) {
  entry: tv_cat(,,);
}
family DitransV(V) {
  entry: tv_cat(            , / np[Z acc] , <IndirectObject>Z);
  entry: tv_cat(/ pp[Z acc] ,             , <IndirectObject>Z);
}
```

This shows the declaration of a parameterized expansion, `iv_cat`, which defines a category (and its semantics) while leaving variables embedded in it that allow further syntactic and semantic arguments to be added. The `tv_cat` definition in turn builds on `iv_cat`, allowing arguments to be inserted either before or after the direct object. The `DitransV` family makes use of this, providing entries that implement both double-object and PP-shifted forms of a ditransitive verb.

An important aspect of OpenCCG that supports this sort of inheritance in the semantics is the use of hybrid logics (Baldridge and Kruijff, 2002) for representing logical forms as a flattened set of elementary predications.[11]

Expansions provide a very flexible means to generalize not only how words are defined (morphology), but also how categories are constructed. The space savings (in terms of the amount of grammar code which a grammar engineer is confronted with) can be orders of magnitude in size: for example, the 16 DotCCG lines given above translate into 200+ (harder to maintain) lines in OpenCCG's XML.

Of course, constructing words and categories in this way can make it difficult to see exactly what the lexicon looks like directly in DotCCG. VisCCG, described in detail in the next section, is able to display—at various levels of granularity—the resulting lexicon, both the words and the categories that are available, *while* the grammar is being edited for faster development and debugging.

7 VisCCG: wiki-style GUI editing

DotCCG provides a great deal of power to the grammar engineer with or without a GUI. However, for many users, a GUI is still an important means for using a grammar platform effectively, and visualization can help even the advanced developer

[11] Similar representations, e.g. Minimal Recursion Semantics, would work equally well in this regard.

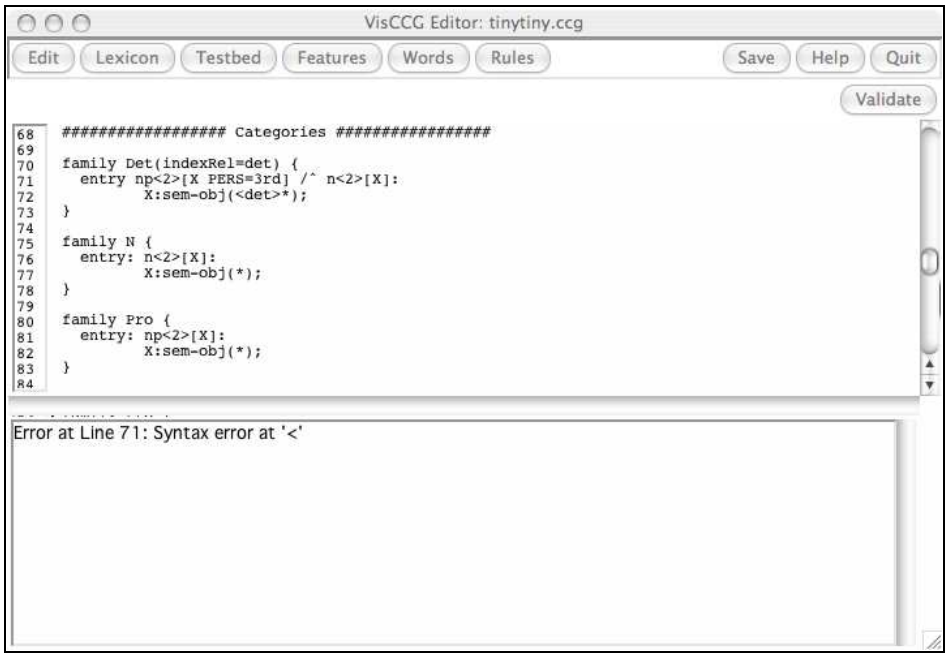


Figure 5: Debugging with CCG

see the structure and definitions of a grammar more effectively. VisCCG takes a wiki-like approach, which enables grammar visualization while never taking the developer too far from the underlying definitions. The goal is to allow new users to begin using the system very quickly without constraining advanced users within the bounds of purely-graphical editing (as opposed to textual editing in conjunction with visualization).

When starting new grammars, it is often useful to iron out nuances of the lexicon, rules or morphology before expanding the grammar significantly. VisCCG allows users to begin with a few essential aspects such as rules and features and then visualize and debug them even without a complete grammar. This adheres to the software engineering paradigm of rapid application development. Individual sections can be edited and visualized independently, enhancing the maintainability of the grammars.

VisCCG allows the user to begin a new grammar with a template that organizes the modules of the grammar. This simplifies bootstrapping of grammar development and also helps maintain a de facto standard for grammars developed using the system – though users are free to deviate from it if they wish. More importantly, as the grammar evolves over time with perhaps multiple people contributing to and refining the grammar, the subsection to be edited is easily localized.

IDEs for programming languages provide detailed debugging information for syntax errors in source code. Similarly, VisCCG identifies syntax errors in the DotCCG source and highlights them for users to fix, as illustrated in figure 5.

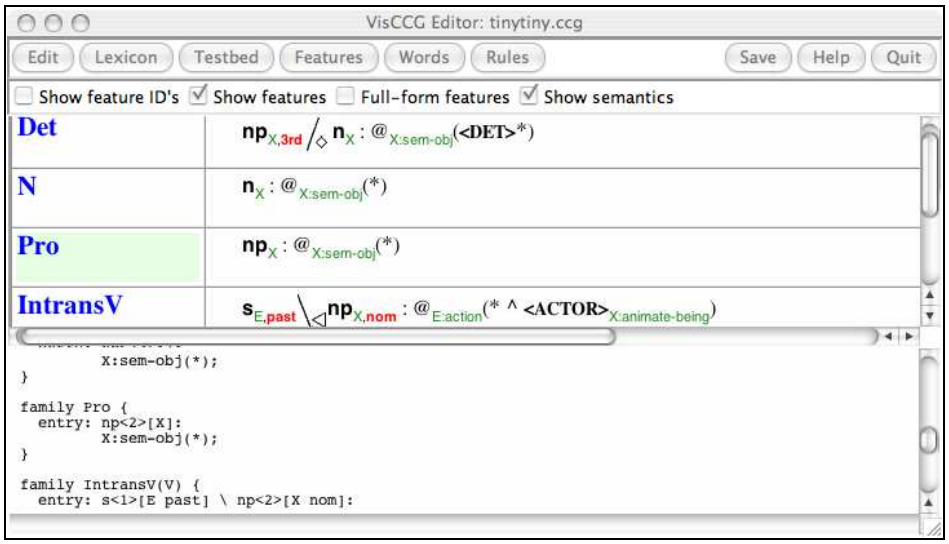


Figure 6: Local editing in Lexicon mode. The *Pro* family has been selected for editing from the graphical display (the top pane); this opens the grammar file for editing at the location which specifies the family (the lower pane).

The line numbers displayed beside the source help localize and isolate individual errors. This capability alone dramatically improves development time, even for experienced developers.

The visualization of a grammar is often very different from what we can express in text. VisCCG enables users to view the grammar at various levels of granularity, allowing the user to spot errors and generalizations easily and without needing to view unrelated information, such as details of features or semantics. As with wikis, VisCCG allows a user to locally edit a small part of the grammar. This is made possible by the terseness of DotCCG, which itself is made possible by the fact that CCG categories can be concisely specified in a linear format. VisCCG additionally allows editing to occur while the user continues to view the graphical representation of the grammar. This feature allows seamless editing of one category definition in the 'Lexicon' tab while other categories are visualized at the desired granularity. Also, the results of such an edit are immediately visible, allowing the user to try out various features before saving changes. An example of editing the 'Pro' family is illustrated in Figure 6.

VisCCG has many different modes of visualization. The initial screen is a basic editor that allows the user to develop their grammar from scratch. The 'Testbed' tab also the user to input new test sentences, and the 'Feature' tab provides a straightforward means of editing the feature hierarchy. The 'Words' tab lists all available lexical items as well as their various inflected forms. This is especially useful for checking the output of expansions, and in particular expansions which produce words based on stems and morphological regularities. This rich set of capabilities

enables the user to update the grammar with a tight editing and visualization cycle. These capabilities also ease the process of grammar development by allowing the user to focus on particular sections, while being able to switch back to any other view easily.

8 Uses of and resources for DotCCG and VisCCG

VisCCG has been used so far in both graduate and undergraduate classes to teach both CCG and grammar engineering. Even students with little computational background were able to use the tools effectively with just a single lab session. Previous courses that used the XML format proved it to be frustrating for students, and required many sessions for them to use at all (and certainly not master). This experience was in fact the genesis of DotCCG.

For teaching purposes and to facilitate wider use of VisCCG, we have developed a wiki[12] which focuses on the various computational and linguistic resources available for learning to use and for using the system. These resources include tutorials, links to software download sites, and access to a number of grammars which have been developed using VisCCG. Among these are small (in many cases tiny) grammars for Tagalog, Ojibwe, French, and Hungarian, as well as some small-domain English grammars. Though no truly broad-coverage grammar has been developed with our new tools to date, they are already being used to develop grammars used in some of the projects listed in Figure 2, including AdaRTE, INDIGO, and Methodius.

We see a number of interesting directions for development of the tools discussed in this paper. In addition to refining the presentation of the various components of the grammar, it would be extremely useful to be able to run the OpenCCG parser from inside VisCCG. It would also be interesting to expand the grammar initialization process to include something like the customization questionnaire used in the Grammar Matrix (Bender and Flickinger, 2005).

9 Conclusion

We have presented an overview and motivation of our work on a set of tools for improving grammar engineering for OpenCCG. The approach is two-pronged in that it improves textual representations of CCG grammars via the DotCCG format and it allows the information in such grammars to be visualized with VisCCG. VisCCG furthermore supports wiki-style editing that enables users to edit small sections of the grammar while visualizing the rest and to see the results of their edits immediately. However, the use of VisCCG for editing is optional – DotCCG grammars can be edited with any plain-text editor as well. The simplicity, flexibility and power

[12] `http://comp.ling.utexas.edu/wiki/doku.php/openccg`

of DotCCG and the optional availability of VisCCG is crucial for supporting the needs of both new and advanced users.

References

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

Baldridge, Jason and Kruijff, Geert-Jan. 2003. Multi-Modal Combinatory Categorial Grammar. In *Proceedings of EACL*, Budapest, Hungary.

Baldridge, Jason and Kruijff, Geert-Jan M. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proceedings of ACL*.

Becker, Tilman, Blaylock, Nate, Gerstenberger, Ciprian, Kruijff-Korbayov, Ivana, Korthauer, Andreas, Pinkal, Manfred, Pitz, Michael, Poller, Peter and Schehl, Jan. 2006. Natural and intuitive multimodal dialogue for in-car applications: The SAMMIE system. In *Proceedings of the ECAI Sub-Conference on Prestigious Applications of Intelligent Systems (PAIS 2006)*, Riva del Garda, Italy.

Bender, Emily M. and Flickinger, Dan. 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.

Benzmüller, Christoph, Horacek, Helmut, Kruijff-Korbayova, Ivana, Pinkal, Manfred, Siekmann, Jörg and Wolska, Magdalena. 2007. Natural Language Dialog with a Tutor System for Mathematical Proofs. In Ruqian Lu, Jörg Siekmann and Carsten Ullrich (eds.), *Cognitive Systems*, volume 4429 of *LNAI*, Springer.

Bierner, Gann. 2001. *Alternative Phrases: Theoretical Analysis and Practical Applications*. Ph. D.thesis, Division of Informatics, University of Edinburgh.

Bos, Johan, Clark, Stephen, Steedman, Mark, Curran, James R. and Hockenmaier, Julia. 2004. Wide-Coverage Semantic Representations from a CCG Parser. In *Proceedings of COLING-04*, pages 1240–1246.

Bozşahin, Cem, Kruijff, Geert-Jan M. and White, Michael. 2006. Specifying Grammars for OpenCCG: A Rough Guide. http://openccg.sf.net/.

Butt, Miriam, King, Tracy Holloway, Niño, María-Eugenia and Segond, Frédérique. 1998. *A Grammar Writer's Cookbook*. Stanford, CA: CSLI.

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

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

Doran, Christine, Hockey, Beth Ann, Sarkar, Anoop, Srinivas, B. and Xia, Fei. 2000. Evolution of the XTAG System. In Anne Abeillé and Owen Rambo (eds.), *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*, pages 371–404, Stanford, CA: CSLI Publishing.

Foster, Mary Ellen and White, Michael. 2005. Assessing the impact of adaptive generation in the COMIC multimodal dialogue system. In *Proceedings of the IJCAI 2005 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Edinburgh.

Foster, Mary Ellen and White, Michael. 2007. Avoiding repetition in generated text. In *Proceedings of ENLG*, Schloss Dagstuhl.

Gerstenberger, Ciprian-Virgil and Wolksa, Magdalena. 2005. Introducing Topological Field Information into CCG. In *Proceedings of the 10th ESSLLI Student Session*, pages 62–74, Edinburgh, UK.

Hockenmaier, Julia. 2003. Parsing with Generative Models of Predicate-Argument Structure. In *Proceedings of ACL*.

Hockenmaier, Julia, Bierner, Gann and Baldridge, Jason. 2004. Extending the coverage of a CCG System. *Research in Language and Computation* 2, 165–208.

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

Isard, Amy. 2007. Choosing the Best Comparison Under the Circumstances. In *Proceedings of the International Workshop on Personalization Enhanced Access to Cultural Heritage (PATCH07)*, Corfu, Greece.

Isard, Amy, Brockmann, Carsten and Oberlander, Jon. 2006. Individuality and Alignment in Generated Dialogues. In *Proceedings of INLG-06*, pages 22–29.

Kaplan, R. M., Maxwell, J. T., King, T. H. and Crouch, R. S. 2004. Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of Combining Shallow and Deep Processing for NLP, ESSLLI 2004*.

Kroeger, Paul. 1993. *Phrase Structure and Grammatical Relations in Tagalog*. Stanford: CSLI Publications.

Kruijff, Geert-Jan and Baldridge, Jason. 2004. Generalizing Dimensionality in Combinatory Categorial Grammar. In *Proceedings of COLING-04*.

Kruijff, Geert-Jan M., Zender, Hendrik, Jensfelt, Patric and Christensen, Henrik I. 2007. Situated Dialogue and Spatial Organization: What, Where... and Why? *International Journal of Advanced Robotic Systems* 4(2).

Moore, Johanna D., Foster, Mary Ellen, Lemon, Oliver and White, Michael. 2004. Generating tailored, comparative descriptions in spoken dialogue. In *Proceedings of FLAIRS 2004*, Miami Beach.

Nakatsu, Crystal and White, Michael. 2006. Learning to Say It Well: Reranking Realizations by Predicted Synthesis Quality. In *Proceedings of COLING-ACL 2006*.

Rickert, Markus, Foster, Mary Ellen, Giuliani, Manuel, By, Tomas, Panin, Giorgio and Knoll, Alois. 2007. Integrating language, vision and action for human robot dialog systems. In *Proceedings of HCI International 2007*, Beijing.

Rojas-Barahona, Lina M. 2007. Adapting Combinatory Categorial Grammars in a Framework for Health Care Dialogue Systems. In *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue (DECALOG 2007)*, pages 187–188.

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press/Bradford Books.

Steedman, Mark and Baldridge, Jason. To appear. Combinatory Categorial Grammar. In Robert Boersley and Kersti Börjars (eds.), *Nontransformational Syntax: A Guide to Current Models*, Blackwell.

Villavicencio, Aline. 2002. *The Acquisition of a Unification-Based Generalised Categorial Grammar*. Ph. D.thesis, University of Cambridge.

White, Michael. 2006a. CCG Chart Realization from Disjunctive Inputs. In *Proceedings of INLG-06*.

White, Michael. 2006b. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation* 4(1), 39–75.

White, Michael and Baldridge, Jason. 2003. Adapting Chart Realization to CCG. In *Proceedings of ENLG*.

White, Michael, Rajkumar, Rajakrishnan and Martin, Scott. 2007. Towards Broad Coverage Surface Realization with CCG. In *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*, Copenhagen.

Wolska, Magdalena and Kruijff-Korbayová, Ivana. 2004. Analysis of Mixed Natural and Symbolic Input in Mathematical Dialogs. In *Proceedings of ACL*, pages 25–32.

Zettlemoyer, Luke and Collins, Michael. 2007. Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In *Proceedings of EMNLP-CoNLL 2007*.

Combining Research and Pedagogy in the Development of a Crosslinguistic Grammar Resource

Emily M. Bender
University of Washington

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes how a graduate course in multilingual grammar engineering has been used to inform the development of the LinGO Grammar Matrix. When the course was first taught (in 2004), the Grammar Matrix consisted only of the cross-linguistic core grammar. Over time, the lab instructions for the students in the course sparked the development of extensions to the Matrix providing 'libraries' of analyses of crosslinguistically variable phenomena. At the same time, the students' course work has provided valuable feedback both in error checking of the core grammar and refinement of the libraries. Based on the experience of teaching this class for four years, I suggest that grammar engineering courses present a rich opportunity for the combination of pedagogy and research. Involving even beginning grammar engineers in on-going investigations can be rewarding for all involved.

1 Introduction

This paper is an exploration of how student work can be harnessed to further current research goals in a way that also benefits the students, and how individual faculty or researchers can efficiently combine both teaching and research roles. In particular, I present some reflections on these ideas from the vantage point of research on the LinGO Grammar Matrix (Bender et al., 2002) and the University of Washington's Linguistics 567: "Knowledge Engineering for Natural Language Processing". In Section 2, I briefly present the Grammar Matrix. Section 3 describes how the course is structured. Section 4 gives examples of how student work has provided valuable feedback to the Grammar Matrix. Section 5 outlines possible future directions for the course.

2 The Grammar Matrix

This section briefly introduces the Grammar Matrix, situating it in its theoretical context, and describing the contents of the core grammar and the phenomenon specific libraries.

2.1 Theoretical context

The LinGO Grammar Matrix (Bender et al., 2002; Flickinger and Bender, 2003; Bender and Flickinger, 2005; Drellishak and Bender, 2005) is a starter-kit designed to facilitate the rapid development of broad-coverage precision grammars. In addition, it is intended to promote consistency in semantic representations such that

[†]The Grammar Matrix project is a collaborative effort, and I would like to acknowledge the contributions of Scott Drellishak, Chris Evans, Dan Flickinger, Stephan Oepen, Kelly O'Hara, and Laurie Poulson, as well as the students in Linguistics 471 and 567 in 2004-2007. The original inspiration for the course came from Petter Haugereid. The Grammar Matrix project is supported by NSF grant BCS-0644097 and a gift to the Turing Center from the Utilika Foundation.

broader applications using a Matrix-derived or Matrix-compatible grammar can easily be adapted to additional languages by switching in different Matrix-derived or Matrix-compatible grammars. The Grammar Matrix has also emerged as an interesting platform for exploring generalizations and the range of variation across languages in syntax and the syntax-semantics interface. In this respect, it represents a kind of computational linguistic typology.

The Grammar Matrix is couched within the Head-driven Phrase Structure Grammar framework (HPSG; Pollard and Sag 1994), and uses Minimal Recursion Semantics (MRS; Copestake et al. 2005) for the semantic representations. It is implemented in tdl, a typed-feature formalism interpreted by the LKB grammar development environment (Copestake, 2002) and the PET parser (Callmeier, 2000). The Grammar Matrix is developed within the context of the DELPH-IN consortium (www.delph-in.net), as part of a larger constellation of grammars and associated software.

More broadly, the Grammar Matrix is an instance of multilingual grammar engineering. In this sense, it is similar in spirit to the ParGram project (Butt et al., 2002; King et al., 2005), the MetaGrammar project (Kinyon et al., 2006), KPML (Bateman et al., 2005), Grammix (Müller, 2007) and OpenCCG (Baldridge et al., 2007). Among approaches to multilingual grammar engineering (Bender et al., 2005), the Grammar Matrix's distinguishing characteristics include the deployment of a shared core grammar for crosslinguistically consistent constraints and a series of libraries modeling varying linguistic properties.

2.2 Core grammar

The core grammar consists of types and constraints which are meant to be crosslinguistically useful. The core grammar always, in fact, represents a set of working hypotheses about language universals within the general framework of the Grammar Matrix. As constraints are found to be incorrect for languages analyzed using the Matrix, they are removed from the core grammar. Types found to be irrelevant for particular languages are also intended to be moved out to the libraries, though in practice this process is slower, as the presence of irrelevant types in a grammar does not affect the analyses assigned to strings by the grammar, provided those types are not used in any rule or lexical item instances.

The core grammar focuses on the following six aspects of a grammar: (i) the basic feature geometry, (ii) basic construction types (e.g., head-subject phrases, head-adjunct phrases, and coordination phrases), (iii) semantic composition, or the way in which the semantic representations of phrases are computed on the basis of the semantic representations of their daughters and the contribution of the rule licensing the phrase, (iv) basic lexical types, including linking types associating syntactic with semantic arguments, (v) basic types for lexical rules, and (vi) collateral files for interaction with the parsing, generation and grammar development software (the LKB and PET).

2.3 Libraries and customization

The core grammar itself has already proven useful, jump-starting the development of several grammars including the NorSource grammar of Norwegian (Hellan and Haugereid, 2003), the Spanish Resource Grammar (Marimon et al., 2007) and the Modern Greek Resource Grammar (Kordoni and Neu, 2005). If the goal is the reuse of grammar code across languages, however, restricting the Matrix to those types and constraints which are valid across all languages is quite limiting: In other words, it seems likely that the analysis (or implementation) of, say, verb-final word order used in Japanese ought to also be applicable to another verb-final language, such as Malayalam.[1] This is the motivation behind the development of phenomenon-specific libraries which provide analyses of different variations on the same phenomenon (e.g., major constituent word order, AND-coordination, etc.) (Bender and Flickinger, 2005; Drellishak and Bender, 2005). These analyses are accessed through a website[2] which presents the user with a typological questionnaire and outputs a working grammar combining the Matrix core grammar with information from the libraries on the basis of the user's answers to the questionnaire.

The statement above about cross-linguistic applicability of particular analyses is a hypothesis to be tested: Given the interconnectedness of analyses of disparate phenomena within a grammar, it is not *a priori* obvious that one and the same analysis of a given phenomenon (e.g., verb-final order) will integrate properly with the required analyses of all the rest of the phenomena in two different languages. In fact, in developing the libraries to date, we have found them to be non-modular in several respects. It is not possible, for example, to fully specify the head-complement rule which is output by the basic word order module without also knowing whether adpositions, complementizers, and auxiliaries (if present) precede or follow their complements. Nonetheless, it is interesting to work towards universal coverage in the libraries while attempting to properly account for their interactions. It is in this way that the development of the Matrix becomes an exercise in computational linguistic typology.

The Matrix libraries are a type of parameterization of linguistic variation, and in that sense, this approach is similar to the Principles and Parameters approach (P&P) (Chomsky, 1981, *inter alia*). However, where P&P work typically tries to derive multiple disparate surface phenomena from each parameter, the Matrix libraries target one phenomenon at a time. Another important difference is that the Matrix is a grammar engineering project, producing grammar fragments which can be run against test suites to validate the interaction of the analyses (Oepen and Flickinger, 1998; Bender, 2006; Bender et al., 2007).

The current libraries address major constituent order, strategies for expressing sentential negation and (matrix) yes-no questions, a handful of lexical prop-

[1] Asher and Kumari (1997) give SOV as the basic order in Malayalam, but also state that there is a good deal of freedom of order of constituents even in unmarked sentences.

[2] http://www.delph-in.net/matrix/customize/matrix.cgi

erties (optionality of determiners, NP v. PP arguments of verbs, intransitive and transitive argument frames) (Bender and Flickinger, 2005) and AND-coordination (Drellishak and Bender, 2005). The coordination library in particular is based on a thorough typological study of the phenomenon in question (Drellishak, 2004). Current work is targeting case, verb-argument agreement in person, number and gender, tense and aspect, argument optionality, and demonstratives. In addition, we are developing general mechanisms for handling lexical rules and the interactions between them through the customization interface.

The general methodology for constructing libraries begins with a survey of the typological and syntactic research literature to map out the typological domain. Then we create analyses for each variant and construct questions to elicit information needed to decide between the variants from the linguist-user. The next step is to create the software to select and output analyses on the basis of the linguist-user's answers, while accounting for interactions with other existing libraries. Finally, we also create test items and filters for the regression testing system (Poulson, 2006; Bender et al., 2007) to validate the new functionality, check for regressions in previously covered territory, and document the new functionality for future regression testing purposes.

2.4 Goals of the project

This section has outlined the current state of the Grammar Matrix project. Our long-term goals for this project are: (i) to increase the gain of the jump-start, i.e., the size of the initial grammar fragments provided by the customization system, (ii) to facilitate the deployment of NLP technology such as grammar checkers, machine translation systems and computer assisted language learning software for low-density languages, (iii) to integrate the Grammar Matrix with other technologies for language documentation and to foster collaboration between field linguists and grammar engineers (Bender et al., 2004), and (iv) to further develop the research field of computational linguistic typology.

In the next sections, I describe the class which has been the driving force behind much of the development of the Grammar Matrix over the past four years, and the ways in which student work in the course provides feedback which is folded back into the Grammar Matrix.

3 Course overview

The pedagogical goals of the course are (i) to give students hands-on experience in the development of substantial linguistic resources for NLP; (ii) to illustrate the importance of test suite creation in the development and evaluation of such resources; and (iii) to explore the nature of linguistic hypothesis testing given the interconnectedness of subsystems within grammars. The students are typically graduate

students in computational linguistics,[3] though graduate students and advanced undergraduates in general linguistics and computer science also attend. All have taken an introductory theoretical HPSG syntax course as a prerequisite.

3.1 Course outline

The course is organized around weekly lab assignments. The course meetings are divided into lectures covering background material and discussion sessions which are driven by student questions and which typically involve interactive work with the grammar development environment (the LKB; Copestake, 2002).

In the first week, the students get to know the LKB by extending a grammar for a small fragment of English. They also choose the language they will be working with for the rest of the quarter and find reference grammars. Each student must choose a different language which has not been studied before in the class. In four years, we have covered 42 languages, from American Sign Language to Zulu, representing the language families Indo-European (15 languages), Afro-Asiatic (3), Niger-Congo (3), Altaic (2), Austronesian (2), Dravidian (2), Na-Dene (2), Sino-Tibetan (2), Uralic (2), Eskimo-Aleut (1), and Uto-Aztecan (1).[4] In addition, the languages covered include two creoles, three languages the Ethnologue classifies as isolates or quasi-isolates (Basque, Japanese, and Korean), one signed language (ASL), and one invented language (Esperanto). Students typically end up working with languages they have not studied before.[5]

In the second and third weeks, the students create test suites for their languages covering the phenomena to be analyzed in the class. In the process of creating these test suites, they become familiar with their reference grammars and in some cases seek out native speaker consultants to ask for acceptability judgments. The test suites include both positive and negative examples, with the latter typically outnumbering the former. Students are encouraged to use a restricted vocabulary, to illustrate each phenomenon with sentences that are as simple as possible, and to include examples illustrating the interaction of multiple phenomena (e.g., negative questions).

In the remaining seven weeks, students incrementally extend grammars for fragments of their languages. They begin by customizing and downloading a copy of the Matrix through the Matrix customization web page. If the libraries do not yet include an analysis of the appropriate variation on some phenomenon, the starter grammars will be correspondingly smaller (omitting coordination, say, or sentential negation).

In subsequent labs, students add case and agreement (as appropriate for their

[3] Particularly in the professional MA program in computational linguistics, for which this course serves as an elective, see http://compling.washington.edu.

[4] These counts are based on the Ethnologue's classifications of the languages in question (Gordon, 2005).

[5] Artificial languages are generally not allowed. The criterion is that the language must have or have had native speakers, which Esperanto does (Gordon, 2005).

languages), move from a full-form lexicon to one incorporating lexical rules, and add the rules and types needed to treat argument optionality (*pro*-drop), demonstrative adjectives or determiners, (other) adjectives and adverbs, embedded declaratives and polar interrogatives, and expressions of ability (e.g., modals). In this, the students are guided by lab instructions describing the phenomena to be analyzed, enumerating known variations on those phenomena, and suggesting Matrix-compatible analyses for each variation. When the lab instructions do not anticipate a variation that turns up in one of the languages, the student working on that language and I work together to produce an appropriate analysis, which the student tests by implementing it.

The lab requirements crucially include a write-up explaining how the phenomena treated that week manifest in the student's language, detailing the analyses that the student developed, and describing any difficulties that the student encountered. These write-ups are critical for communication between the students and the instructor in the on-going development of the grammar, and for the incorporation of feedback from the course grammars into the Grammar Matrix itself.

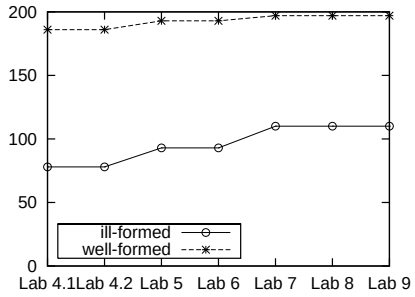
As a grand finale, we use the LOGON open-source translation software (Oepen et al., 2004a; Bond et al., 2005) to put the grammars together into an NxN machine translation system. The coverage is necessarily limited, but the students are always excited to see their grammars 'talking' to each other, and it serves as a motivating end point for the grammar development.

3.2 Test suites and grammar evolution

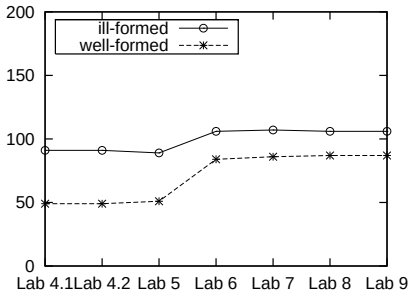
The students track the progress of their grammars using the [incr tsdb()] competence and performance profiling system (Oepen, 2002). [incr tsdb()] allows the students to compare the coverage, ambiguity, and overgeneration of their grammars over their test suite across different stages in the grammar's development, and to discover which test items have different analyses across test runs. Students are encouraged to use this not only for overall benchmarking but also to explore the consequences of particular changes to the grammar in the process of grammar development.

While the bulk of the test suite development is done at the beginning of the quarter, the test suites continue to evolve over time for a variety of reasons: In some cases, students change their transcription system or opt for a more morphophonologically abstract representation, and end up editing their test suites consistently. In other cases, in the course of grammar development, students discover and correct errors in their test suites. A third possible reason for test suite evolution is the addition of further examples to test interactions and corner cases unnoticed in the original test suite development.

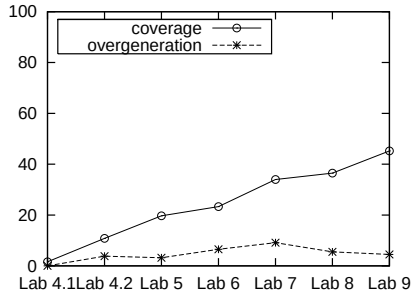
Figure 1 illustrates the evolution of the test suites on the one hand and coverage and ambiguity over those test suites on the other for two grammars: a Hebrew grammar developed in 2006 by Margalit Zabludowski and a Zulu grammar devel-



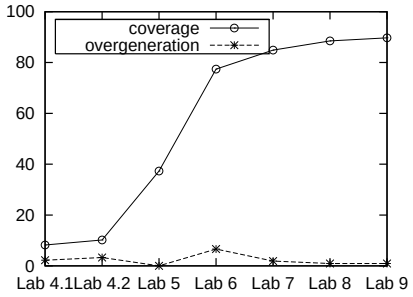
Hebrew test suite evolution



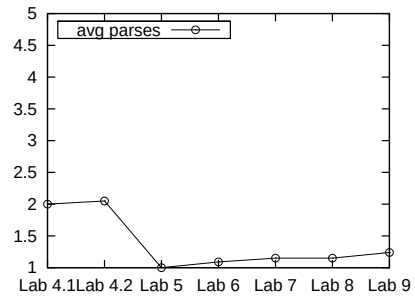
Zulu test suite evolution



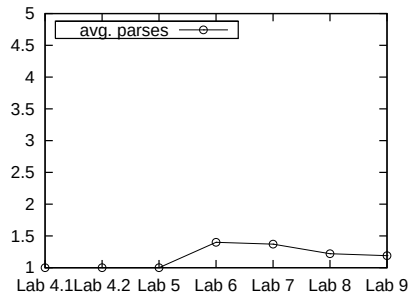
Hebrew coverage



Zulu coverage



Hebrew ambiguity



Zulu ambiguity

Figure 1: Test suite and grammar evolution for two languages

oped in 2007 by Kelly O'Hara. Each point on the graph represents one week's lab.[6] As can be seen from the graphs, the grammar writers were able to substantially increase their coverage over grammatical sentences without overgenerating.[7] The ambiguity rates reflect typical progress as well: Some changes to the grammars introduce spurious ambiguity, which is corrected in later versions. At the same time, there is a gradual increase in real ambiguity as the grammars expand to cover more phenomena.

3.3 Challenges and benefits

The course described here is tremendously challenging for both the students and the instructor, but also has benefits in proportion to the challenges. For the students, the challenges largely come from the fact that they are asked to master quite a bit of material quickly and in parallel: The software we use (the LKB, [incr tsdb()], emacs), the Grammar Matrix itself (including its assumptions and general structure), and the language they are working with (which is often completely new to them). For students coming from a primarily linguistic background, there are also engineering skills to master, such as the debugging process. In addition, the assignments are generally open-ended. In my experience, the students have typically taken this open-endedness as a challenge to extend their grammars further and stretch for broader coverage and higher precision. The flip side of these challenges are the benefits: Because the course is intimately connected with a larger research project, the students are participating in knowledge creation, which they tend to find extremely motivating. At the same time, having all of their effort in the course directed to a single, original, course project gives them a potential spring board for further research as well as valuable experience to point to when they are on the job market

From the instructor's point of view, the first challenge is again the open-endedness of the assignments. This makes student evaluation difficult, as the students submit highly diverse work, both in terms of the specific aspects of their languages they work on and in terms of the distance they go with the assignments. A second challenge is dealing with many languages at once. Here, I found that giving specific instructions for lab write-ups guiding students to provide the relevant background information about the analyses they are implementing has been very important. In fact, the ability to deal with many languages at once is one of the main benefits from my point of view as well, especially dealing with the same or similar phenomena in many languages. In order to answer a question about a particular analysis (e.g., of case) in a Matrix-derived grammar, I need to review the relevant aspects of the Matrix. Not only is it easier to answer a question about case in another language while all the information is fresh in my mind, it is also very useful to get a compar-

[6] The first entry for Lab 4 represents the grammar fragment as downloaded from the customization system. The second represents the addition of some of the missing vocabulary.

[7] The higher coverage of the Zulu grammar as compared to the Hebrew mainly reflects the fact that the Hebrew test suite staked out a more ambitious fragment of the language.

ative view on case (or any other phenomenon) by considering data and especially difficulties in analysis from multiple languages all at once. More generally, the students' work provides feedback to the Matrix which allows for error detection, library development, and library refinement, as discussed in the next section.

4 Feedback to the Matrix

The Grammar Matrix project faces sizeable hurdles in evaluation and validation (Poulson, 2006). The Matrix core grammar itself cannot be directly tested, as it is not a grammar of any particular language, and cannot parse or generate any strings. The Matrix customization system and libraries can be validated through autogeneration of test suites for abstract language types (Bender et al., 2007), but this is only validation, and not evaluation. The test suite generation system allows us to verify that the libraries have the behavior we intend them to, and that they interact in reasonable ways. It does not provide a means of testing the linguistic correctness of the system, i.e., its typological predictions. Are the libraries complete? Do the libraries interact in ways that predict the facts of actual human languages? This section will describe how student work in the class can help with detecting errors in the core grammar (Section 4.1) and with detecting lacunae in existing libraries (Section 4.3). In addition, the course played an important role in the development of the initial set of libraries (Section 4.2).

4.1 Error detection

The Grammar Matrix core was originally derived from the English Resource Grammar (ERG; version of November 2001), by taking ERG and removing all types or constraints that appeared to be English-specific. This process was informed by comparison with the JACY grammar of Japanese (Siegel and Bender, 2002). Nonetheless, this process was expected to be somewhat error-prone: English-specific constraints could easily be missed, and constraints which are in fact general could also be removed. In the intervening six years, the core grammar has grown and been refined. We have added linking and lexical rule types, which were not modeled directly on the analogous types in the ERG (though the ERG was a reference point), and we have explored new analyses of phenomena such as the marking of illocutionary force and the licensing and interpretation of dropped arguments.

The core grammar is abstract in the sense that it cannot in itself parse or generate any sentences. That is, while the Matrix core grammar has many of the essential ingredient for actual rules and lexical entries, in includes no fully specified rules. Among the types that define the phrase structure rules, for example, there is an abstract head-complement phrase type which describes the syntactic and semantic effects of combining a head with a complement it is seeking. This type does not specify the order of its head and non-head daughters. In any particular grammar it will be cross-classified with either the head-final or the head-initial type (or both,

instantiated by separate rule instances).

Unfortunately, abstract does not necessarily mean simple, and even these relatively underspecified types bear constraints which relate to many different analyses. For example, the Matrix provides a feature MC (for 'main clause') which records whether a constituent displays phenomena restricted to root (or alternatively subordinate clauses). For example, in English, subject-auxiliary inversion is limited to root clauses, and in German (and similar V2 languages) the major constituent order differs between the clause types. A constituent which is [MC +] is restricted to main clauses, while a [MC −] constituent is restricted to subordinate clauses. A constituent which is underspecified for MC can appear in either. Following the ERG, we provide a third possible (non-underspecified) value *na* ('not applicable') for non-clausal constituents. Analyses which make use of this feature rely on its being specified for every constituent, and thus all of the phrase structure rules must inherit a specification for the feature from some supertype.

This is just one example of the way in which individual types bear constraints which relate to multiple different linguistic phenomena, even in the abstract core grammar. Since the core grammar was derived from a broad-coverage grammar and since it aims to support the development of similarly broad-coverage grammars, it is a complex object. Thus in general, even with perfect knowledge of linguistic typology, it would not be possible to examine the core grammar directly and find all of the errors it contains. It is only in applying the Matrix to particular languages that we can hope to find out where the current working hypotheses are incorrect.

In addition to removing constraints which turned out not to be universal, the analysis of new languages sometimes leads to the addition of types to the Matrix core. The most interesting example from the past year is the introduction of ternary rules for certain constructions. The ternary rules types were added to handle negation in Hausa (for the grammar developed by Kelsey Hutchins), which is marked by two particles, one on either side of the clause:[8]

(1) Hausa (hau)
bàà rashìn nāmàà zâi kashè mùtûm ba
NEG lack.of meat FUT kill person NEG
'It is not that lack of meat will kill a person.' (Newman, 2000, 363)

The analysis we developed for such examples involves a ternary rule which requires a finite clause as its middle daughter and specific lexical items (the left and right negation markers) as its left and right daughters, as illustrated in Figure 2. The middle daughter is the syntactic head, but the construction itself functions as the semantic head.

Previously, Matrix-derived grammars (following the ERG) had handled all constructions with unary or binary phrases, using recursing binary phrases to model

[8]For improved automatic discovery of this document, all examples are labeled with the name of the language represented, followed by the ISO 639-3 language code, in parentheses.

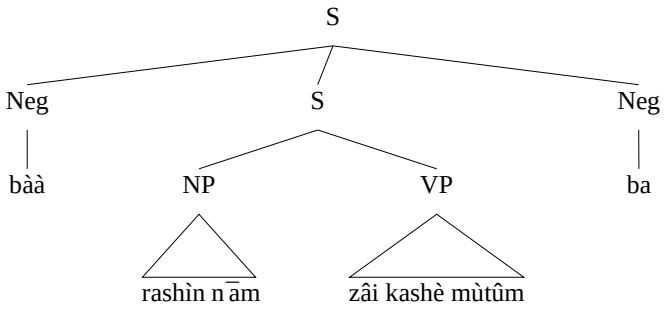


Figure 2: Schematic tree for Hausa example (1), 'It is not that lack of meat will kill a person.'

variable arity in, for example, head-complement constructions with multiple complements for a single head and coordination of more than two coordinands. This construction in Hausa is different, however, in that it requires specifically three daughters. While it would certainly have been possible to attach one of the two markers lower than the other, such an analysis would have required diacritic features in order to require the second marker (and ensure that nothing else attached in between).

The ternary rule types were immediately useful for another grammar being developed at the same time by Sarah Churng for American Sign Language (ASL). In ASL, many grammatical features are signaled through 'non-manual markers' (NMMs) (Baker and Padden, 1978), typically facial expressions. These NMMs extend over whole constituents, and in fact constitute a separate parallel channel to the signal. For the purposes of building a Matrix-derived grammar, we developed a transliteration system which indicates the NMMs through left and right brackets.

(2) American Sign Language (ase)
JOHN ⟨ne BUY HOUSE ne⟩
John nm-neg buy house nm-neg
'John is not buying a house.'

These left and right pairs (for negation and yes-no questions) were then parsed by the same kind of ternary-branching rules required for Hausa.[9]

Error detection in the Grammar Matrix depends on the deployment of the Matrix in grammars for many languages, but building such grammars is time-consuming, even with the jump-start provided by the Matrix. Feedback does come

[9]The types for ternary rules are another case where there is tension between the strict interpretation of the core grammar as universally valid types and constraints and other classifications of the types. It is quite likely that there are languages for which we will never need ternary rules, and yet since these types do much the same work as the basic types for unary and binary rules, they are currently stored as part of the core grammar.

On the other hand, these types might turn out to be useful in many languages when we consider punctuation.

in from research groups around the world using the Grammar Matrix, but the communication is uneven in such cases: users may simply decide to edit the Matrix core grammar without reporting back to the Matrix developers. In the context of the class, on the other hand, I get to explore as many languages as there are students, with extensive information on how each grammar was able to use the Matrix, and where the Matrix needed to be modified.

4.2 Library development

The initial set of libraries (Bender and Flickinger, 2005) also grew directly out of the course. In the first two years of teaching this course (in 2004 and 2005), I developed and expanded an initial set of lab instructions, covering basic word order, case, agreement, modification, the expression of ability, sentential negation, argument optionality, and matrix and embedded statements and yes-no questions. In some cases, these directions were fairly open-ended. In others, by the end of the second time teaching the course, they were specific enough that it was clear that there was only a little work left to make them so precise a machine could follow them.

With the initial libraries and customization system in place, the Grammar Matrix can provide a greater jump-start (provided appropriate options are available within the libraries for the language in question). This means that students can explore their languages in greater detail within the 10-week course. In the most recent course, in addition to the phenomena from years one and two, we have been looking into the marking of discourse status (in particular definiteness and demonstratives) and coordination, with the latter supported by and providing feedback to the Matrix coordination library (Drellishak and Bender, 2005).

4.3 Library refinement

Just as the class provides a chance to find errors and lacunae in the Matrix core grammar, it also provides crucial feedback to the libraries. As the negation and yes-no question libraries were not based on thorough studies of the typological literature, it is not surprising that we have already turned up cases that were not covered. For negation, this includes the circumfixal negation described above for Hausa and ASL. For the question library, Wendy Bannister's work on Malayalam turned up what is likely a common strategy: question marking via inflection on the main verb, as illustrated in (3) (Asher and Kumari, 1997, 8):

(3) Malayalam (mal)

a. Avan vannu
   He come.past
   'He came.'

b. Avan vann-oo
   He come.past-Q
   'Did he come?'

In addition, the French grammar developed by Fabiola Henri and Gwendoline Fox[10] turned up another problem with the question library: Unlike the negation library, the question library only allowed for one kind of question marking per language. In fact, however, French is representative in allowing multiple strategies: a sentence-initial question marker (*est-ce que*) and subject-verb inversion, as illustrated in (4).

(4) French (fra)

a. Est-ce qu' il 3SG.NOM.MASC est be.3SG parti? leave.PAST-PART.
   Q
   'Has he left?

b. Est-il parti?
   be.3SG-3SG.NOM.MASC leave.PAST-PART.
   'Has he left?

Even with the coordination library, which was based from its inception on a typological study, the course grammars have turned up an unhandled case: The coordination library currently provides for multiple coordination strategies within a single language, each with its own coordination mark, but any given strategy will use only one mark. Michelle Neves's work on Indonesian showed this to be inadequate, as Indonesian can mix the coordinators *serta* and *dan* in the same coordinate structure, typically using *dan* for all but the last coordinand pair of coordinands, which are joined instead by *serta* (Sneddon, 1996, 339-340):[11]

(5) Indonesian (ind)
   Ini untuk hiasan dinding dan meja serta kursi
   DEM. for decoration wall CONJ table CONJ chair
   'These are decorations for walls, tables and chairs.'

The course grammars also provide interesting information on the interaction between libraries. For example, one of the course grammars for 2007 (developed by Ryan Georgi) was for Modern Standard Arabic (MSA). In MSA, word order interacts with agreement: Both VSO and SVO are possible. In VSO word order, the verb agrees with the subject in person, number, and gender, whereas in SVO word order, there is only agreement in person and gender (Soltan, 2006, 240).

[10] Students in a similar course at the LSA Institute, Stanford 2007, taught by Stephan Oepen, Dan Flickinger, and myself

[11] Sentence (5) is constructed on the basis of the Sneddon, but does not appear in this form in the book.

(6) Modern Standard Arabic (arb)

a. ʔal-ʔawlaad-u qaraʔ-uu d-dars-a
the-boys-NOM read-3.PL.MASC the-lesson-ACC
'The boys read the lesson.'

b.*ʔal-ʔawlaad-u qaraʔ-a d-dars-a
the-boysNOM read-3.SG.MASC the-lesson-ACC
'The boys read the lesson.'

c. qaraʔ-a l-ʔawlaad-u d-dars-a
read-3.SG.MASC the-boysNOM the-lesson-ACC
'The boys read the lesson.'

c.*qaraʔ-uu l-ʔawlaad-u d-dars-a
read-3.PL.MASC the-boysNOM the-lesson-ACC
'The boys read the lesson.'

The word order library does not yet allow for variable word order of this kind. Verb-final (i.e., variation between SOV and OSV), verb-initial (variation between VSO and VOS), and free word order (of major constituents) are allowed, but not yet variation between VSO and SVO. It is not clear how soon the customization system will achieve the level of complexity required to specify an MSA-style system through the customization interface. Nonetheless, as we extend the word order library and begin to develop a library for agreement as well, MSA provides an interesting case to work towards.

4.4 Summary

This section has described how student work based on the Grammar Matrix in the grammar engineering course has contributed to error detection in the core grammar as well as to the development and refinement of the libraries. To a certain extent, any context in which the Matrix is applied to new languages will have similar benefits. However, there are some ways in which the classroom context is particularly helpful, compared to, for example, feedback from other research groups using the Grammar Matrix. The first is the degree of detail that is available. The students turn in multiple versions of their grammars, along with write-ups of the linguistic data analyzed and the analyses themselves. The grading work for the course thus doubles as information gathering for the Matrix project. The second major benefit of the classroom context is the coordinated, focused attention of many participants (students and instructor) on the same phenomena at the same time. It is much easier to integrate new information about different languages in this format, than when the information comes in a less coordinated fashion. Finally, as noted above, the pedagogical work of developing the lab instructions fed directly into the research/engineering work of developing the libraries. At the same time, it should be noted that the course grammars alone are not sufficient to test the Grammar Matrix. In particular, they remain small grammars, with only 10 weeks of

development time. To learn how the various proposed analyses scale as grammars reach both interesting coverage and interesting ambiguity, the Matrix needs to be embedded in grammars undergoing sustained development.[12]

5 Future Directions

As the jump-start provided by the Grammar Matrix grows, the course grammars should attain greater complexity, even within the same 10-week time period. Initially, this will simply mean grammars which cover more phenomena. Beyond a certain level of complexity, however, I anticipate bigger changes. Within a few years, it should be feasible to have the students collect small corpora, and then process those corpora with their grammars. This will quickly turn up additional phenomena to work on (cf. Baldwin et al., 2005).[13]

In addition, as the grammars gain complexity they will also display more ambiguity. This makes it interesting to explore the creation of treebanks in the Redwoods style (Oepen et al., 2004b), where grammar engineers select among the trees proposed by the grammar on the basis of minimal discriminants. These treebanks can be used to train parse selection models (Toutanova et al., 2002). They will also represent small but interesting resources for low-density languages.

Finally, a recent project at the Turing Center at the University of Washington[14] has been piloting a many-to-many machine translation system based on the LOGON machine translation infrastructure (Oepen et al., 2004a; Bond et al., 2005) and using nine grammars from the first four years of the grammar engineering class.[15] The current system has only a toy vocabulary but works across an interesting range of grammatical phenomena. As this MT system grows more robust, it will be interesting to explore adding the course grammars to it as they are produced.

6 Conclusion

The Grammar Matrix project is well-suited to harnessing student work. It needs input from many languages, and relatively basic input is still quite valuable: the first grammatical phenomena one might try to account for in a language (e.g., word order, valence patterns, case, agreement) are currently under development in the Grammar Matrix. As the customization system grows, each new grammar will still provide useful information: either the libraries will handle the variants found in a grammar and the student can explore additional phenomena at the boundaries of

[12] Thanks to an anonymous reviewer for highlighting this point.

[13] Note that in order to make this practical, however, we will need to handle standard orthography as well as morphophonology. Currently, in order to focus on morphosyntax, I advise students to abstract away from these, working with a transliteration system and assuming a morphophonological preprocessor.

[14] http://turing.cs.washington.edu

[15] The grammars are for Armenian, Esperanto, Farsi, Finnish, Hausa, Hebrew, Icelandic, Italian and Zulu. In addition, we have a purpose-built grammar for English with similar coverage.

Matrix development, or the language will turn up a new variant to be incorporated into one of the libraries, or both.

There is no denying that the course is very intense for both the students and the instructor. To the extent that it is intense, it is also rewarding, certainly for me and I believe also for the students. I find that most of the additional work I put into teaching this course (above the commitment required for other courses) is effectively research effort, in that it feeds back into the Grammar Matrix project.

I also believe that there should be similar opportunities to integrate student course work into sustained projects elsewhere within the field of grammar engineering, as we need detailed attention to many separate linguistic phenomena. The problem in many cases is to find ways to lower the barriers to entry, such that the student projects become tractable, while also maintaining checks on the quality of the data or analyses added to the system.

References

Asher, Ronald E. and Kumari, T.C. 1997. *Malayalam*. London: Routledge.

Baker, Charlotte and Padden, Carol. 1978. Focusing on the Non-manual Components of American Sign Language. In P. Siple (ed.), *Understanding Language Through Sign Language Research*, pages 27–57, New York: Academic Press.

Baldridge, Jason, Chatterjee, Sudipta, Palmer, Alexis and Wing, Ben. 2007. DotCCG and VisCCG: Wiki and Programming Paradigms for Improved Grammar Engineering with OpenCCG. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF 2007 Workshp*, Stanford, CA: CSLI.

Baldwin, Timothy, Beavers, John, Bender, Emily M., Flickinger, Dan, Kim, Ara and Oepen, Stephan. 2005. Beauty and the Beast: What running a broad-coverage precision grammar over the BNC taught us about the grammar — and the corpus. In Stephan Kepser and Marga Reis (eds.), *Linguistic Evidence: Empirical, Theoretical, and Computational Perspectives*, Berlin: Mouton de Gruyter.

Bateman, John A., Kruijff-Korbayová, Ivana and Kruijff, Geert-Jan. 2005. Multilingual Resource Sharing Across Both Related and Unrelated Languages: An Implemented, Open-Source Framework for Practical Natural Language Generation. *Research on Language and Computation, Special Issue on Shared Representations in Multilingual Grammar Engineering* 3(2), 191–219.

Bender, Emily M. 2006. Grammar Engineering for Linguistic Hypothesis Testing. In *Proceedings of Texas Linguistic Society X*, Stanford: CSLI Publications.

Bender, Emily M. and Flickinger, Dan. 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 M., Flickinger, Dan, Fouvry, Frederik and Siegel, Melanie. 2005. Introduction. *Research on Language and Computation, Special Issue on Shared Representations in Multilingual Grammar Engineering* 3(2).

Bender, Emily M., Flickinger, Dan, Good, Jeff and Sag, Ivan A. 2004. Montage: Leveraging Advances in Grammar Engineering, Linguistic Ontologies, and Mark-up for the Documentation of Underdescribed Languages. In *Proceedings of the Workshop on First Steps for Language Documentation of Minority Languages: Computational Linguistic Tools for Morphology, Lexicon and Corpus Compilation, LREC 2004*, Lisbon, Portugal.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Bender, Emily M., Poulson, Laurie, Drellishak, Scott and Evans, Chris. 2007. Validation and Regression Testing for a Cross-linguistic Grammar Resource. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 136–143, Prague, Czech Republic: Association for Computational Linguistics.

Bond, Francis, Oepen, Stephan, Siegel, Melanie, Copestake, Ann and Flickinger, Dan. 2005. Open Source Machine Translation with DELPH-IN. In *Open-Source Machine Translation: Workshop at MT Summit X*, pages 15–22, Phuket.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 1–7.

Callmeier, Ulrich. 2000. PET — A Platform for Experimentation with Efficient HPSG Processing Techniques. *Natural Language Engineering* 6 (1) (Special Issue on Efficient Processing with HPSG), 99 – 108.

Chomsky, Noam. 1981. *Lectures on Government and Binding*. Dordrecht: Foris.

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

Copestake, Ann, Flickinger, Dan, Pollard, Carl and Sag, Ivan A. 2005. Minimal Recursion Semantics: An Introduction. *Research on Language & Computation* 3(2–3), 281–332.

Drellishak, Scott. 2004. A Survey of Coordination in the World's Languages, MA thesis, University of Washington.

Drellishak, Scott and Bender, Emily M. 2005. A Coordination Module for a Crosslinguistic Grammar Resource. In Stefan Müller (ed.), *The Proceedings of the 12th International Conference on Head-Driven Phrase Structure Grammar, Department of Informatics, University of Lisbon*, pages 108–128, Stanford: CSLI Publications.

Flickinger, Dan and Bender, Emily M. 2003. Compositional Semantics in a Multilingual Grammar Resource. In Emily M. Bender, Dan Flickinger, Frederik Fouvry and Melanie Siegel (eds.), *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Devel, ESSLLI 2003opment*, pages 33–42, Vienna, Austria.

Gordon, Raymond G., Jr. (ed.). 2005. *Ethnologue: Languages of the World*. Dallas, TX: SIL International, fifteenth edition, online version: http://www.ethnologue.com.

Hellan, Lars and Haugereid, Petter. 2003. NorSource: An Exercise in Matrix Grammar-Building Design. In Emily M. Bender, Dan Flickinger, Freerik Fouvry and Melanie Siegel (eds.), *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development, ESSLLI 2003*, pages 41–48, Vienna, Austria.

King, Tracy Holloway, Forst, Martin, Kuhn, Jonas and Butt, Miriam. 2005. The Feature Space in Parallel Grammar Writing. *Research on Language and Computation, Special Issue on Shared Representations in Multilingual Grammar Engineering* 3(2), 139–163.

Kinyon, Alexandra, Rambow, Owen, Scheffler, Tatjana, Yoon, SinWon and Joshi, Aravind K. 2006. The Metagrammar Goes Multilingual: A Cross-Linguistic Look at the V2-Phenomenon. In *In Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8)*, Sydney, Australia.

Kordoni, Valia and Neu, Julia. 2005. Deep Analysis of Modern Greek. In Keh-Yih Su, Jun'ichi Tsujii and Jong-Hyeok Lee (eds.), *Lecture Notes in Computer Science*, volume 3248, pages 674–683, Berlin: Springer-Verlag.

Marimon, Montserrat, Bel, Núria and Seghezzi, Natalia. 2007. Test-suite Construction for a Spanish Grammar. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF 2007 Workshop*, Stanford, CA: CSLI Publications.

Müller, Stefan. 2007. The Grammix CD-ROM: A Software Collection for Developing Typed Feature Structure Grammars. In Tracy Holloway King and Emily M.

Bender (eds.), *Proceedings of the GEAF 2007 Workshop*, Stanford, CA: CSLI Publications.

Newman, Paul. 2000. *The Hausa Language: an Encyclopedic Reference Grammar*. New Haven: Yale University Press.

Oepen, Stephan. 2002. *Competence and Performance Profiling for Constraint-based Grammars: A New Methodology, Toolkit, and Applications*. Ph. D.thesis, Universität des Saarlandes.

Oepen, Stephan, Dyvik, Helge, Lønning, Jan Tore, Velldal, Erik, Beermann, Dorothee, Carroll, John, Flickinger, Dan, Hellan, Lars, Johannessen, Janne Bondi, Meurer, Paul, Nordgård, Torbjørn and Rosén, Victoria. 2004a. Som å kapp-ete med trollet? Towards MRS-Based Norwegian – English Machine Translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*.

Oepen, Stephan, Flickinger, Daniel, Toutanova, Kristina and Manning, Christopher D. 2004b. LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. *Journal of Research on Language and Computation* 2(4), 575 – 596.

Oepen, Stephan and Flickinger, Daniel P. 1998. Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language* 12 (4) (Special Issue on Evaluation), 411 – 436.

Pollard, Carl and Sag, Ivan A. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics, Chicago, IL and Stanford, CA: The University of Chicago Press and CSLI Publications.

Poulson, Laurie. 2006. Evaluating a cross-linguistic grammar model: Methodology and test-suite resource development, MA thesis, University of Washington.

Siegel, Melanie and Bender, Emily M. 2002. Efficient Deep Processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Sneddon, James Neil. 1996. *Indonesian: A Comprehensive Grammar*. London and New York: Routledge.

Soltan, Usama. 2006. Standard Arabic subject-verb agreement asymmetry revisited in an Agree-based minimalist syntax. In Cedric Boeckx (ed.), *Agreement Systems*, Amsterdam: John Benjamins.

Toutanova, Kristina, Manning, Chris and Oepen, Stephan. 2002. Parse Ranking for a Rich HPSG Grammar. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories (TLT2002)*, Sozopol, Bulgaria.

PARC's Bridge and Question Answering System

Daniel G. Bobrow, Bob Cheslow, Cleo Condoravdi, Lauri Karttunen, Tracy Holloway King, Rowan Nairn, Valeria de Paiva, Charlotte Price, and Annie Zaenen

PARC

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes the Bridge system, a system designed to robustly map from natural language sentences to abstract knowledge representations. The system runs on PARC's parser, generator, and ordered rewrite platform XLE. The Bridge system has been extended to include a type of light inference, based on an entailment and contradiction detection algorithm which also runs on XLE. The paper then describes a search and question answering application, Asker, which uses the Bridge system to create a semantic index of text passages and which allows a user to query the index in natural language.

1 Introduction

Bridge is a PARC system that robustly maps natural language sentences into a logical abstract knowledge representation language (AKR). Using this mapping, we have built an application, Asker, that supports high-precision question-answering of natural language queries from large document collections (e.g., the Wikipedia, newswire, financial reports). For example, if a collection includes the sentence *The reporter failed to discover that three young men were killed in the attack on Ryad.*, then the system could answer the query *Did anyone die in the attack on Ryad?* with YES (perhaps indicating who died) and highlight the phrase in the document in the collection that contains this information.

The basic system components and their connection is shown in the diagrams in Figures 1–4. Natural language text is mapped into a first level of abstract knowledge representation (AKR0) (see section 2). Text passages are then passed through an expansion step to produce a representation with addtional inferrable facts (P-AKR). In contrast, queries are passed through a simplification step to produce a representation with fewer facts (Q-AKR), a smaller kernel from which the rest can be inferred. Asker uses the expanded passage to compute index terms that capture semantic roles in the representation (section 4.1). To retrieve potential answer passages from the collection, index terms from the query representation identify stored texts with corresponding semantic structure (section 4.2); as a backoff, texts are retrieved that share expanded, normalized keywords with the query. Entailment and contradiction detection (ECD) can be performed to determine subsumption relations between the passage and question and hence provide an answer (section 3). ECD can be used separately to check whether a given passage text entails or contradicts a given query/hypothesis text.

[†]This work was sponsored in part by DTO. Approved for Public Release; distribution unlimited. We thank the audience of GEAF for providing extensive feedback on the QA demo of the system. We also thank all of the people who have worked on the system over time. Ron Kaplan and Dick Crouch were central members of the team, and helped define the framework of the Bridge/Asker system. Dick Crouch was a major designer and implementor of key components. John T. Maxwell III is a major designer and implementor of the core XLE system. We also want to thank the interns and postdocs who contributed: Tina Bögel, Hannah Copperman, Liz Coppock, Olya Gurevich, Anubha Kothari, Xiaofei Lu, Johannes Neubarth, Matt Paden, Karl Pichotta, and Kiyoko Uchiyama.

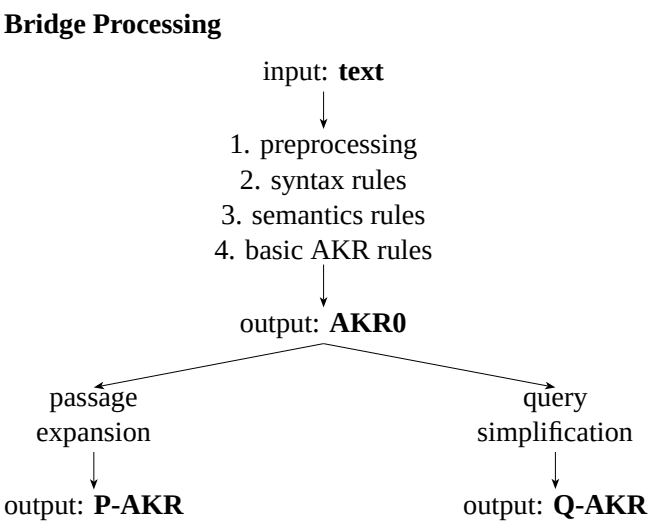


Figure 1: Syntactic Lexical-Function Grammar (LFG) rules and semantic and KR-specific ordered rewrite rules produce a basic knowledge representation for passage and query texts. Passages expand inferences based on linguistic properties. Queries are simplified to their core meaning to remove unnecessary structure.

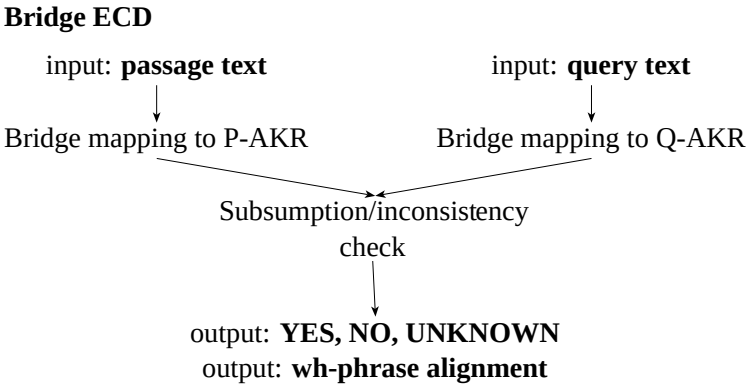


Figure 2: Expanded passage representations are compared using subsumption with simplified query representations to determine if the passage entails the query.

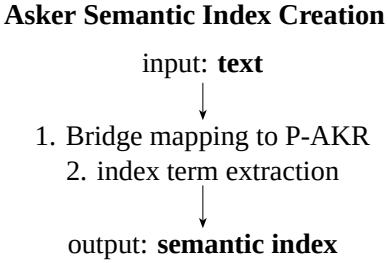


Figure 3: Index terms for each passage reflect the semantic roles of terms in a sentence.

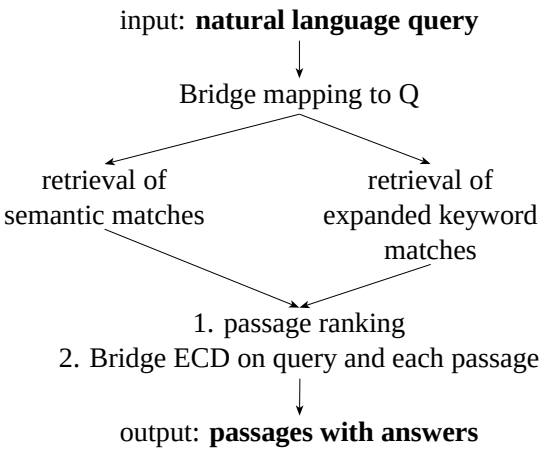


Figure 4: Use of index terms in the query supports more precise retrieval of relevant sentences. Keywords, expanded with WordNet synonym sets (synsets) and hypernyms, provide a backoff for recall.

The mapping from syntactic structures to (linguistic) semantics and then abstract knowledge representations (AKR) runs on the XLE platform (Maxwell and Kaplan, 1996; Crouch et al., 2007) and is described in Crouch and King (2006) and Crouch (2005). The logic of the representations has been described in Bobrow et al. (2005) and de Paiva et al. (2007). The linguistic, semantic rationale for the use of concepts in AKR was originally described in Condoravdi et al. (2001, 2003). Components of the system have been described in Crouch and King (2005), Gurevich et al. (2005), and Nairn et al. (2006). An earlier application to a collection of copier repair tips written by Xerox technicians is described in Crouch et al. (2002) and Everett et al. (2002). The more recent application to question-answering in

the framework of the PASCAL-organized[1] competition Recognizing Textual Entailment (RTE) is described in Bobrow et al. (2007).

In this paper, we first describe the AKR language that our system uses (section 2). AKR is designed to meet two constraints that are somewhat in tension: a natural representation of language constructs on the one hand and a straightforward computation of direct inferential relations between two texts on the other. Our entailment and contradiction detection algorithm (section 3) implements this inference procedure between two possibly ambiguous texts without the need for disambiguation. Finally, we discuss the structure of the Asker repository which indexes sentences on the basis of their AKR representation in a large scale database (over 10^6 documents) and allows real-time semantic retrieval from this index (section 4).

2 Abstract Knowledge Representation (AKR)

We start our discussion of AKR representations with the sentence *John Smith discovered that three men died.* The full AKR is as in (1).

(1) **Conceptual Structure:**
subconcept(discover:2, [detect-1, ..., identify-5])
role(Theme, discover:2, ctx(die:5))
role(Agent, discover:2, Smith:1)
subconcept(Smith:1, [male-2])
alias(Smith:1, [John, Smith, John_Smith])
role(cardinality_restriction, Smith:1, sg)
subconcept(die:5, [die-1, die-2, ..., die-11])
role(Theme, die:5, man:4)
subconcept(man:4, [man-1, ..., world-8])
role(cardinality_restriction, man:4, 3)
Contextual Structure:
context(t)
context(ctx(die:5))
top_context(t)
context_lifting_relation(veridical, t, ctx(die:5))
context_relation(t, ctx(die:5), crel(Theme, discover:2))
instantiable(Smith:1, t)
instantiable(discover:2, t)
instantiable(die:5, ctx(die:5))
instantiable(man:4, ctx(die:5))
Temporal Structure:
temporalRel(startsAfterEndingOf, Now, discover:2)
temporalRel(startsAfterEndingOf, Now, die:5)

[1] See the PASCAL website: www.pascal-network.org

The representation for this sentence has two contexts: the top context t, specifying what the author of the sentence is committed to as the true state of the world by virtue of uttering the sentence; and ctx(die:5), specifying what was discovered by John Smith, which is the proposition that three men died.

The verb *discover* carries a presupposition that what is described as being discovered is true according to the author of the sentence; that is, one can only be said to discover true facts. This is part of lexical knowledge and is captured in this example by context_lifting_relation(veridical, t, ctx(die:5)). Because of this veridical relation, in the expansion to P-AKR, the clauses:

(2) instantiable(die:5, t)
instantiable(man:4, t)

are added to the contextual structure. These instantiability statements capture existence commitments in our representation. As a result, the system will answer YES to the passage-query pair *John discovered that three men died. Did three men die?* In the top context t, we also have the instatiability claims:

(3) instantiable(Smith:1, t)
instantiable(discover:2, t)

Within the context of what was discovered by John Smith we have two concepts, the dying event die:5, and the concept man:4. For each of these, the representation has a subconcept expression. These expressions encode WordNet's representation of the verb *die* (a list of 11 synsets, corresponding to the 11 verb senses for *die* differentiated in WordNet) and the noun *man* (a list of 8 synsets):

(4) subconcept(die:5, [die-1, die-2, die-3, ..., die-11])
subconcept(man:4, [man-1, serviceman-1, ..., world-8])

We are using WordNet (Fellbaum, 1998) as a surrogate for the taxonomic part of an ontology because it is the largest available resource for mapping English words into an (approximate) abstraction hierarchy through WordNet's hypernyms. We have patched WordNet in places where omissions and extra entries became problems for the system. Since VerbNet, whose use is described below, links to WordNet, we have also made these two resources more consistent.

To capture the fact that the number of dying men is three, the representation includes a cardinality restriction on the concept man:4.[2] The dying event is related to its undergoer participants via role(Theme, die:5, man:4). In the top context we have two more concepts, the concept for John Smith and the discovering event discover:2, a subconcept of WordNet's synsets for the verb *discover*.

[2] Our representations deal with quantifiers in general through a combination of instantiability statements, contexts and cardinality restriction clauses.

While WordNet knows about some words used as names,[3] it does not list every man named John in history, nor does it list every masculine name. The English morphology associated with the system's syntactic grammar knows that *John* is a man's name, and the semantics uses this information to create a subconcept structure based on WordNet: subconcept(Smith:1, [male-2]). The name itself is captured in an alias fact. Incorporated into the system is a theory of when two aliases can refer to the same individual. So *John Smith* can be mentioned later as *John*, *Smith*, or *John Smith*. These three possiblities are included in the alias fact. Given a passage-query pair like *John Smith arrived and John Bowler left. Did Bowler leave?* the system will answer YES. Moreover, to the passage-query pair *John Smith arrived and John Bowler left. Did John leave?* the system will answer YES: [John Bowler], since at least one of the people named *John* in the passage did leave.

Finally, the concept discover:2 is restricted to have Smith:1 as its agent role (role(Agent, discover:2, Smith:1)) and the context specifying what John discovered as its theme role (role(Theme, discover:2, ctx(die:5))).

The temporal relations capture the relative time ordering of the events described with respect to the time of utterance or writing of the sentence. Now (the time of utterance) is after the discovering, and the dying, as represented by:

(5) temporalRel(startsAfterEndingOf, Now, discover:2)
temporalRel(startsAfterEndingOf, Now, die:5)

As indicated by this example, AKR representations can express the content of beliefs, possible states of the world, counterfactuals, etc.

2.1 Existence and Restrictions

Terms like die:5 and man:4 do not refer to individuals, but to concepts (or types). When the AKR makes reference to a subconcept man:4 of the kind [man-1, serviceman-1, man-3, . . ., world-8] restricted to be a kind of man that died, the AKR does not make a commitment that there are any instances of this subconcept in the world being described by the author of a sentence. For example, the sentence *John imagined that three men died.*, has in the AKR an embedded context representing what is being imagined. Because this embedded context is not veridical with respect to the top context, there is no commitment (by the author or in the representation) about there actually being any dead men.

The instantiable assertions represent the existence of the kinds of objects described. In the top-level context t, there is a commitment to an instance of a male individual with the name *John Smith* and of a discover event discover:2 made by him. While the three men and the dying event occur in the context of what was discovered by John Smith, they become instantiable at the top context because *discover* with a *that* complement is marked as a factive verb (Nairn et al., 2006).

[3]For example, WordNet has synsets for the evangelist John and the English King John who signed the Magna Carta. There are also entries for the common noun *john*.

Compared to traditional first order logic with complex quantifiers, AKR separates the descriptions of types of events and objects (in the conceptual block) from the commitments to existence (in the contextual block). The conceptual block includes subconcept assertions, role restrictions and cardinality constraints. The contextual block includes (un)instantiability of these concepts in contexts, and relations between contexts, including context-lifting rules similar in spirit to those in McCarthy's context logic (McCarthy, 1993). The use of contexts to capture a collection of statements true in a context and the use of contexts as arguments (reifying the collection of statements) makes AKR technically not first order, but the reasoning in the system preserves many first order properties locally.

2.2 Lexical resources

Mapping to AKR and textual inference depend crucially on words and ontological relations between the concepts they map to. We have integrated a number of existing lexical resources into a Unified Lexicon (UL) (Crouch and King, 2005), adding new annotations to classes of words to support desired inferences. The basic size of the UL is shown in (6).

(6) **Unified Lexicon: Part of Speech of Entries**

POS	Number of Entries
verbs	42,675
nouns	14,293
adjectives	8,537
deverbal adjectives	1,291
adverbs	13

Note that many words have no UL entry because their behavior in the mapping to AKR is predictable from their syntactic structure (e.g., most nouns, adjectives, and adverbs). In addition, adjectives and nouns that are predictably derived from verbs (e.g, *the hopping frog*, *the defeated champion*, *the writing of the book*) do not need entries in the UL to trigger the appropriate mapping rules.

2.2.1 Basic Concept and Role Lookup

The mapping rules and the UL use WordNet synsets and hypernyms. The system maps the words recognized by WordNet into the associated synsets directly via the WordNet API; a copy of WordNet is not included in the UL. Words not in WordNet are mapped to, generally singleton, synsets based on information from the XLE morphology and syntax (e.g., the treatment of person names discussed above). Initially all synsets for a given word are retrieved; this list is then trimmed to a subset of the WordNet concepts if additional information is available, for example from VerbNet or from the context of the text. Noun-noun compounds (e.g., *theme park*) and adjective-noun compounds (e.g., *high school*) known to WordNet are assigned the appropriate WordNet synsets.

VerbNet (Kipper et al., 2000) is used to map from syntactic predicate-argument structures to event structures with named roles, occasionally simplified by collapsing certain role distinctions. These role and event structures have been heuristically augmented to cover all of the verb-subcategorization frame pairs in the XLE syntactic lexicon (e.g., the role assignments from verbs known to VerbNet can be used to provide roles for other verbs in their WordNet synset with the same subcategorization frames). This results in significant expansion of the coverage of VerbNet: of the ∼42,000 verb entries in the UL, ∼25,000 are not directly from VerbNet. Examples of the VerbNet roles can be seen in the AKRs in examples such as (1).

2.2.2 Lexical Marking for Rule Triggering

In addition to these basic resources, the UL incorporates information about lexical items that is needed to trigger mapping rules that affect the contextual facts, especially those involving relations between contexts (Nairn et al., 2006). These lexical classes are shown in (7).

(7) **Unified Lexicon: Lexical Marking for Rule Triggering**

Lexical Class	Number	Example
factives	230	John discovered that Mary left.
implicatives	192	John managed to leave.
propositional attitude	762	John abhors that Mary left.
neutral	33	John sought a unicorn.[4]
temporal relation	721	John longs to leave.
temporal: forward shift	301	John authorized Mary to leave.
temporal: simultaneous	70	John attempted to leave.
sentential adverbs	13	Obviously John left.

For example, the factivity of the verb *discover* when used with a *that* complement is marked. This marking indicates that *discover*'s Theme context argument is veridical with respect to its immediately higher context, enabling the lifting of instantiability from the lower context to the higher one, as described in (1).

2.2.3 Lexical Marking for Normalization

Lexical resources are also used in the normalization of representations. Relevant lexical classes are shown in (8). A canonical example of this type of normalization is the mapping of eventive nominal expressions into equivalent verbal counterparts (e.g., *Rome's destruction of Carthage* is mapped to the same representation as *Rome destroyed Carthage*.) (Gurevich et al., 2005). The UL contains related noun-verb

[4]This marking is meant for intensional predicates with respect to an argument position, distinguishing between *seek* and *find*, for instance. It results in having no instantiability assertion, capturing an existential commitment, for the term corresponding to the relevant argument of the predicate (in the case of *seek* the direct object). By default there is an instantiability assertion for every argument of a predicate in the context of predication.

pairs which are used by the rules to map nouns and their associated phrases into their verbal, eventive counterparts with appropriate arguments. These entries not only include the pairings (e.g. *destruction-destroy*, *employer-employ*) but also classification information. Some of this information involves the mapping of arguments; for example, agentive nominals like *employer* refer to the agent of the event, while *-ee* nominals like *employee* refer to the patient. Other information involves the degree of lexicalization; this determines whether the mapping to the eventive representation is obligatory or optional. These rules, in conjunction with the lexical class information, capture ambiguity in the language; for example, *Rome's destruction* can mean either that Rome is the patient of the destroying event or the agent.

(8) **Unified Lexicon: Lexical Marking for Normalization**

Lexical Class	Number	Example
deverbal nouns	5,484	Rome's destruction of Carthage
become adjective	51	The child sickened.
become more adjective	121	John darkened the room.
pertainyms	289	Japanese children
conditional verb	29	John wasted the chance to leave.
ability nouns	11	John had the choice to leave.
asset nouns	15	John had the money to leave.
bravery nouns	16	John had the strength to leave.
chance nouns	19	John had the chance to leave.
effort nouns	13	John took the trouble to leave.
certainty adjectives	3	John is sure to leave.
consider verb	4	John considered the boy foolish.

The mapping of texts to AKR involves changes of representation to aid inference. Among these are the representation of linguistic paraphrases and idioms which fall into classes that are lexicalized appropriately. For example, the "become adjective" verbs like *redden* are rewritten to an AKR similar to that of *become red*. Phrases such as *take a turn for the worse* are mapped to the representation for *worsen*. An additional, related large class of items are light verbs such as *take*, where the meaning of the verb phrase depends on an argument of the verb. Some examples of light verb use include *take a flight* and *use a hammer* that can be transformed into *fly* and *hammer*. Some verbs are marked as conditionally implicative because they form implicative constructions with a particular class of nouns. For example, *have the foresight to X* is semantically the same type as *manage to X*. As the best representation for the output of these rules is still being explored, there are only a few lexicalizations for each class currently implemented.

As mentioned above, many noun-noun compounds are known to WordNet and hence are given the appropriate WordNet synset. However, many such compounds, especially the less-lexicalized ones, are not in WordNet. The AKR mapping rules define noun-noun relations based on the meaning of the head noun and the meaning of its modifier, where the meanings are (upper level) WordNet synsets. For exam-

ple, a food solid modifying a type of tableware (e.g, *meat plate*) creates a *for* relation. These rules allow multiple mappings to reflect the multiple readings of many noun-noun compounds (e.g., *a wood box* can mean either a box made of wood or a box for holding wood).

Not all normalization is triggered by lexical classes that are encoded in the UL: the structure of the representations is often sufficient to determine how to map them into AKR. Our general approach is to capture the similar content of alternative linguistic expressions by normalizing their AKR to a common representation. This normalization occurs at many levels. For example, the syntax abstracts away from word order and localizes dependencies (e.g. in *John wants to leave.*, John is localized as the subject of both *want* and *leave*), the semantics canonicalizes passives to actives (*The cake was eaten by John.* becomes *John ate the cake.*)[5] and negative quantifiers on subjects (*No boy left.* introduces a sentential negation similar to *not*). Lexically-based inferences provide further information. One significant type of such inferences is associated with verbs of change, such verbs of change of location (e.g. from *John left Athens.* one concludes that John was in Athens before the departure and was not there at least for a while afterwards). The information about pre- and post-conditions of events described by verbs of change such as *leave* is productively extracted from the VerbNet event structure into the UL and then used by the mapping rules.

2.2.4 Lexical Marking for Expansion of Representation

Some mappings expand the representation instead of, or in addition to, normalizing it. Most of these mappings expand just the passages and not the queries. Sample lexical classes of this type are shown in (9).

(9) **Unified Lexicon: Lexical Marking for Expansion of Representation**

Lexical Class	Number	Example
lethal cause verbs	29	John strangled his victim.
symmetric nouns	2	John is Mary's partner.

Such expansions are sometimes specific enough that they are done exclusively in the rules and are not currently in the UL. For example, in a text, *buy* is inferred from *sell*, with appropriate role substitutions, and vice versa. As a result, a query about a buying event can match against a passage described in the text as a selling event. These are done as relatively constrained lexical classes in order to correctly map the arguments of one event to those of the other (e.g. *win-lose* maps its surface arguments differently from *buy-sell*).[6] Family relations such as *husband-wife*

[5]The choice to have the active-passive correspondence dealt with in the mapping component rather than the UL reduces the size of the UL. The active-passive correspondence could, alternatively, be encoded in the UL by matching every transitive verb entry with an entry for its passive counterpart, thus substantially increasing the size of the UL.

[6]With appropriate, more complex lexical markings, such correspondences could be encoded in the UL. Mapping rules would then be used to generate terms and role restrictions for the member of the pair not explicit in the input sentence.

are also expanded in the passages to allow them to match with queries using the converse relation.

A related aspect of our approach is to make information in the structure of certain phrases explicit. For example, date expressions (e.g., *May 1, 2007*) and location expressions (e.g., *Boise, Idaho*) are decomposed into subfacts that allow basic inferencing in conjunction with the rest of the representation. For example, making explicit that Boise is in Idaho, not just part of the name of the place, makes it possible to conclude from the fact that John lives in Boise, Idaho, that John lives in Idaho.

As seen by the wide range of examples in this section, lexical resources are a vital component of the Bridge system. The system incorporates existing resources, such as VerbNet, as well as resources created especially for the system. Each set of resources is used by the AKR mapping rules to create appropriate representations of natural language texts. The efficacy of these resources and their implementation is demonstrated by the ability of the system to use the resulting representations in applications such as the Asker search and question answering system.

2.3 Ambiguity Management

A hallmark of our computational approach to syntax, semantics, and knowledge mapping has been the ability to manage ambiguity by combining alternative interpretations into a single packed structure that can be further processed without the typically exponential cost of unpacking (Maxwell and Kaplan, 1991). For the traditional example of *John saw a girl with a telescope*, the packed representation compactly represents two interpretations: one where the seeing was done with a telescope and the alternative where the girl was carrying a telescope. In the packed representation, the common elements of both interpretations are represented only once, and only the alternative connections need to be expressed. The packed AKR representation is shown in (10). The alternate connections are shown in the lines labeled A1 and A2.

(10) **Choice Space:**
xor(A1, A2) iff 1
Conceptual Structure:
subconcept(see:2, [see-1, ..., interpret-1])
A1: role(prep(with), see:2, telescope:9)
role(Stimulus, see:2, girl:6)
role(Experiencer, see:2, John:1)
subconcept(John:1, [male-2])
alias(John:1, [John])
role(cardinality_restriction, John:1, sg)
subconcept(girl:6, [girl-1, ..., girl-5])
A2: role(prep(with), girl:6, telescope:9)

role(cardinality_restriction, girl:6, sg)
subconcept(telescope:9, [telescope-1])
role(cardinality_restriction, telescope:9, sg)

Contextual Structure:

context(t)
top_context(t)
instantiable(John:1, t)
instantiable(girl:6, t)
instantiable(see:2, t)
instantiable(telescope:9, t)

Temporal Structure:

temporalRel(startsAfterEndingOf, Now, see:2)

The two distinct readings are labeled by A1 and A2, which are a disjoint partition of the top level choice 1 (xor(A1, A2) iff 1). In reading A1, the seeing concept is further restricted to be a seeing with a telescope, whereas in A2, the girl is restricted to be a girl with a telescope.

The mapping from text to AKR via the syntactic and semantic representations and the entailment and contradiction detection take advantage of the same ambiguity management system, thereby gaining full efficiency by never unpacking.

Each level of representation provides possible sources of additional ambiguity. Sometimes it is useful to choose a subset of the interpretations for efficiency reasons or to interface with non-ambiguity-enabled modules and applications. Stochastic models are used to order the interpretations by probability in the XLE system (Riezler et al., 2002). In addition, rule-based optimality marks allow low probability interpretations through only if there is no more optimal interpretation available (Frank et al., 2001). This mechanism is used, for example, to apply VerbNet's sortal restrictions on roles so that the subconcept associated with a verb's arguments can be further constrained, thereby increasing precision and decreasing ambiguity. The optimality mechanism treats these sortal restrictions as soft constraints. If in an ambiguous, packed representation one solution satisfies the sortal restrictions and one does not, only the one that satisfies them appears in the final representation. However, if all the solutions violate the sortal restrictions, the ones which violate the fewest restrictions are used. The combination of efficient processing of packed ambiguous structures with stochastic and rule-based methods for selecting among these representations supports practical, robust analysis of natural language texts.

3 Entailment and Contradiction Detection (ECD)

So far we have described how the Bridge system produces AKR logical forms. These are used for light reasoning, that we call entailment and contradiction detection. It follows the form of the "textual inference" challenge problems that have been part of the PASCAL initiative. The task of the challenge is: given two sentences, P (for passage or premise) and Q (for query or question), determine whether P provides an

intuitive answer for Q as judged by a competent user of the language without any special knowledge. Thus the goal is to decide whether Q follows from P plus some background knowledge, according to the intuitions of an intelligent human reader. This decision is supposed to be based simply on the language involved, factoring out world knowledge, but this distinction is difficult to characterize precisely and has become the topic of much current research.

We have developed a collection of algorithms for efficiently detecting entailment and contradiction relations holding between AKRs for queries and AKRs for candidate answer texts. We have taken a very strict approach, not including plausible inferences. Thus we deal only with a circumscribed set of textual inferences, but ones that must be handled by any system aiming for the larger task. Our approach is to expand the passage texts by using the linguistic inference patterns described earlier. The system tests entailment and contradiction through a subsumption process described below. Some special case reasoners support identification of named objects, comparison of specificity of WordNet synsets, and compatibility of cardinality restrictions. We call our strict form of textual inference "precision-focused textual inference"; our approach and results are described in Bobrow et al. (2007).

As a simple example consider how we conclude from *John saw a happy girl.* that *A child was seen.* The representations are shown in (11) and (12) respectively.

(11) John saw a happy girl.

Conceptual Structure:

subconcept(happy:12, [happy-1, felicitous-2, glad-2, happy-4])
subconcept(see:6, [see-1, understand-2, witness-2, , see-23])
role(Stimulus, see:6, girl:18)
role(Experiencer, see:6, John:1)
subconcept(John:1, [male-2])
alias(John:1, [John])
role(cardinality_restriction, John:1, sg)
subconcept(girl:18, [girl-1, female_child-1, . . . girl-5])
role(cardinality_restriction, girl:18, sg)
role(subsective, girl:18, happy:12)

Contextual Structure:

context(t)
top_context(t)
instantiable(John:1, t)
instantiable(girl:18, t)
instantiable(see:6, t)

Temporal Structure:

temporalRel(startsAfterEndingOf, Now, see:6)

(12) A child was seen.

Conceptual Structure:

subconcept(see:13, [see-1, understand-2, witness-2, . . . see-23])

role(Stimulus, see:13, child:3)
subconcept(child:3, [child-1, child-2, ... child-4])
role(cardinality_restriction, child:3, sg)

Contextual Structure:

context(t)
top_context(t)
instantiable(see:13, t)
instantiable(child:3, t)

Temporal Structure:

temporalRel(startsAfterEndingOf, Now, see:13)

ECD works on texts that have been analyzed into AKRs. Passage AKRs are expanded to encode linguistically based inferences (none in (11)). The AKR for concept and context denoting terms are aligned across the passage and question representations, and rules defining a calculus of entailment and contradiction are applied.

Before determining specificity relations between terms in the premise and conclusion AKRs, it is necessary to align these terms: alignments are not always obvious. They are computed by a heuristic algorithm that considers all plausible alignments where there is sufficient conceptual overlap between terms. This may result in multiple possible alignments with different likelihood scores. Term alignments with wh-terms (*who*, *what*, *where*, etc.) provide the answers to wh-questions when an entailment is detected. In the above example, the two seeing events are aligned, as are the skolems for girl:18 and child:3.

We check each possible term alignment to see if there is an entailment or contradiction between the premise and conclusion representations. The technique detects an entailment or contradiction if any interpretation of a premise entails or contradicts any interpretation of the conclusion.

The detection mechanism is implemented using XLE's packed rewrite system. The core idea behind using the rewrite system is that if the premise representation entails part of the conclusion representation, then that part of the conclusion can be deleted (i.e. rewritten to nil). A conclusion is entailed if all of its component parts have been removed. Hence, if there is a choice in which all of the conclusion representation has been removed, then there is some interpretation of the premise and the conclusion under which the premise entails the conclusion. Contradictions are detected via rules that add a contradiction flag whenever there is a choice of premise and conclusion interpretations such that parts of the representations conflict.

As a preliminary to deleting entailed conclusion facts or flagging contradictions, rules are first applied to make explicit the subsumption and specificity relations holding between concept terms in the premise and conclusion.

The next set of rules explores the consequences of these specificity relations on instantiability claims. For an upward monotone environment, instantiability of a specific concept entails instantiability of a more general concept and uninstantiability of a general concept entails uninstantiability of a more specific concept. For downward monotone environments, the relations are reversed. This captures the

pattern that if *A little girl hopped.* then we know that *A girl hopped.*, since *little girl* is more specific than *girl*. From *Girls hopped.* we cannot infer that *Little girls hopped.*, as it is possible that all the hopping girls are big girls, but from *All girls hopped.* we can infer that *All little girls hopped.*, as the quantifier creates a specificity reversing situation.

To return to our example, it is determined from WordNet that a girl is a kind of child. A happy girl (girl with the role subsective happy) is yet more specific. Hence the seeing event in the passage is more specific than that in the hypothesis and hence instantiable(see:6, t) entails instantiable(see:13, t). Instantiability statements in t are existence statements, and the existence of an instance of a more specific concept implies the existence of its generalizations (if there is a happy girl, there is a girl, which means there is a child, and similarly for *see*).

The ECD algorithm separates the task of structure alignment from the task of detecting logical relations between the representations. This separation makes the method more robust than many graph alignment and matching approaches (Braz et al., 2006) and is applicable to packed representations without full graph matching. This implements a verifiable calculus of entailment and contradiction, which in theory corresponds (closely) to Natural Logic entailment (van Benthem, 1986; MacCartney and Manning, 2007). The differences reside in the introductions of contexts and of packed representations. We believe that the ECD algorithm combines the best of the inference-based and graph-matching approaches. Term alignment is robust to variations in the input structures and the absence of precisely formulated axioms. The entailment calculus rules can be sensitive to non-local aspects of structure and thus deal with more global constraints on entailment or contradiction. In addition, since the approach is ambiguity-enabled, the system can detect whether any one of the possible interpretations of the putative answer answers any one of the possible interpretations of the question.

Given this ability to determine entailment and contradiction between a passage and a query, the Asker system builds up a semantic index of AKRs for passages and then at run-time produces AKRs for queries. These query AKRs are used to retrieve possible answer passages and then ECD can be applied to provide answers to the original query. This process is described in the next section.

4 Indexing and Retrieval

Asker is a search and question answering system. In order to retrieve relevant passages and documents from a large corpus, a specialized search index is constructed that encodes the information from the AKR for each sentence. Typical keyword search indices map words (or their stems) to their document occurrences, along with word offset information and other metadata. The Asker semantic index contains this information, but also maps each word's synonyms and hypernyms to the passages containing them, along with their semantic roles and relations. The index scales to retrieve semantically related passages from very large corpora (millions of docu-

ments from a single server) in a second or less. The results correspond to the semantic structure of the query, enabling much higher precision than free text searches. The results of the semantic search can be evaluated by the ECD algorithms to test for entailment or contradiction and hence answer the query.

4.1 Indexing

Each document in the corpus is broken into sentences, and the AKR for each sentence is fed into the indexer (see Fig. 3). The lexical identifiers (literal strings or identifiers from linguistic resources) for each word in the AKR are combined with information about their semantic function in the passage to create a set of index term strings. These strings are then associated with occurrence information, which records the relationship of the identifier to the actual word (i.e., alias, synonym, or hypernym and the number of levels in the ontology from the word to the hypernym), along with the document, sentence, and predication containing the word, indicators for montonicity, byte positions of the words in the sentence, and other information.

For example, the AKR for the sentence *Ramazi knows Legrande.* contains the semantic roles Agent and Theme, describing the knowing relation between Ramazi and Legrande. These semantic relations are encoded into index terms by combining the term with the role and its position in the relation, e.g. know:Agent:1, Ramazi:Agent:2 and know:Theme:1 and Legrande:Theme:2. These index terms are associated in the search index with the information about how they occur in the passage.

By looking up know:Agent:1, the system will find all occurrences of any agent knowing anything, and Ramazi:Agent:2 will retrieve occurrences where Ramazi is the agent of some event. By taking the intersection of these occurrence lists, the system finds passages where Ramazi knows something. Likewise, the system finds the intersection of occurrences where the Theme involves knowing Legrande. The occurrence information specifies the predication containing these relations, so the system can find those passages containing references to Ramazi knowing Legrande.

The actual index terms are generated using WordNet concept IDs and alias information, rather than the string. So in this example the term *know* is associated with a number of WordNet IDs (synonyms and hypernyms of the term), and each of these IDs is stored separately in the index. Thus, rather than know:Agent:1, the actual terms stored would be 587430:Agent:1, 588355:Agent:1, 588050:Agent:1, etc. The passages associated with these index terms will be retrieved for any term in a query mapping to the same Wordnet concept.

Finally, this information is inverted to enable efficient lookup of occurrences by index term. The index format is designed to store all of this information for each index term in a highly compressed encoding and to permit lookup with little or no degradation of performance as the corpus grows. The occurrence lists (known as postings) are arranged to take advantage of regularities in the occurrence data, using variable-length integers and delta-encoding for compression (as well as bitvectors for the most frequently occurring terms) and the data is localized using skip-lists

to enable efficient disk reads. Each sentence is associated with its containing document, and an arbitrary amount of metadata can be stored for each document.

4.2 Retrieval

At query time (see Fig. 1), the semantic search module receives an AKR for the natural language query. A set of index terms is generated from the query AKR in the same manner used for indexing of passage AKRs, only with simplification of the facts instead of augmentation. The postings for the index terms are retrieved, and the data is processed to find documents containing the appropriately tagged terms occurring in predications that correspond to those in query.

The system attempts to match each semantic fact in the query with each result passage, checking to see that the terms align in corresponding predications. For example, for a query like *Does Ramazi know Legrande?*, the results would include the passage *Legrand is known by Ramazi.*, but it would not include *Ramazi knows Hassan, Hussein knows Legrande, but no one knows the cell leader.*, where both Ramazi and Legrande play roles in a knowing relationship, but not to each other.

This strategy results in high-precision retrieval. As a back-off strategy, the system uses extended key word and key word search techniques. Extended key word search takes advantage of the stemming of word forms and the mapping into WordNet concepts and, for proper nouns, alias facts in order to increase recall of standard key word search. The results of these key word searches are presented separately from the full retrieval results and are not input to ECD.

The retrieval process does not test for strict entailment and contradiction, however. For example, a query of *Did Cheney go to Baghdad?* Might return the passage *Cheney was believed to have gone to Baghdad.*, even though it is not entailed by the query. To check for entailment and contradiction, the results of indexed search can be filtered through the ECD component (section 3) to eliminate false positives for answering the question.

5 Discussion and Conclusions

The Bridge system is a complexly engineered combination of linguistic formalisms based on theoretical criteria. It provides the basis for applications, including entailment and contradiction detection (ECD) and semantic retrieval (Asker). Because the system is under development by a significant number of people working in parallel, it requires a support environment to ensure that changes improve the system in the intended directions, without losing efficiency or accuracy. Some tools supporting this system development are described in Chatzichrisafis et al. (2007).

The architecture provides a layered set of transformations of English text to an abstract knowledge representation. The LFG-based syntactic parsing system produces a dependency structure. The semantics module produces a flattened representation that normalizes these functional dependency structures. It maps grammat-

ical functions into semantic roles and further normalizes the syntactic dependencies, e.g., transforming deverbal nouns and adjectives into their underlying verbal form. Criteria for the syntactic and semantic representations include capturing linguistic generalizations and parallelisms cross-linguistically (Butt et al., 1999, 2002).

The mapping rules for knowledge representation produce descriptions of the concepts under discussion, and a contextual structure that captures the nested structure of contexts. They also specify for each concept whether it is instantiable in the relevant contexts or not. Passage expansion rules add linguistically supported inferences to the representation, to make the import of the sentence explicit in the representation. The criteria for the AKR include natural representation of the distinct meanings of a text, the ability to be transformed into an (extended) first order form for use by other logical reasoners, and support for applications, especially Asker.

The architecture is supported by a collection of linguistic resources, some of which were developed specifically for this system. The broad-coverage English grammar, morphology, and lexicon were developed over many years for a range of applications. The semantics module uses WordNet for its linguistic taxonomic ontology and VerbNet as a resource for transforming grammatical roles into semantic roles. These resources have been extended using syntactic resources from the XLE grammar to produce a Unified Lexicon (UL). In addition, the UL includes lexical markings needed to support normalization, paraphrase, lexical inference, and structural inference. Classes of words that support specific extensions to the initial AKR are lexicalized in the UL. For example, we have identified and categorized over 300 verbs that support pre-suppositional and implicative inference.

Our question answering architecture exploits the AKR representation of sentences. The use of AKR structural components as index terms has significantly improved precision of retrieval from our semantically indexed repository. ECD can be used as a mechanism to answer questions, not just retrieve relevant passages.

References

Bobrow, Daniel G., Condoravdi, Cleo, Crouch, Richard, Kaplan, Ron, Karttunen, Lauri, King, Tracy Holloway, de Paiva, Valeria and Zaenen, Annie. 2005. A Basic Logic for Textual Inference. In *Proceedings of the AAAI Workshop on Inference for Textual Question Answering.*

Bobrow, Daniel G., Condoravdi, Cleo, de Paiva, Valeria, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan, Price, Charlotte and Zaenen, Annie. 2007. Precision-focused Textual Inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing.*

Braz, Rodrigo, Girju, Roxana, Punyakanok, Vasin, Roth, Dan and Sammons, Mark. 2006. An Inference Model for Semantic Entailment in Natural Language. *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Ob-*

ject Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop pages 261–286.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7.

Butt, Miriam, King, Tracy Holloway, Niño, María-Eugenia and Segond, Frédérique. 1999. *A Grammar Writer's Cookbook.* CSLI Publications.

Chatzichrisafis, Nikos, Crouch, Dick, King, Tracy Holloway, Nairn, Rowan, Rayner, Manny and Santaholma, Marianne. 2007. Regression Testing For Grammar-Based Systems. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF 2007 Workshop*, CSLI Publications.

Condoravdi, Cleo, Crouch, Dick, Stolle, Rienhard, de Paiva, Valeria and Bobrow, Daniel G. 2003. Entailment, Intensionality and Text Understanding. In *Proceedings Human Language Technology Conference, Workshop on Text Meaning.*

Condoravdi, Cleo, Crouch, Richard, van der Berg, Martin, Everett, John O., Stolle, Reinhard, de Paiva, Valeria and Bobrow, Daniel G. 2001. Preventing Existence. In *Proceedings of the Conference on Formal Ontologies in Information Systems.*

Crouch, Dick, Condoravdi, Cleo, Stolle, Reinhard, King, Tracy Holloway, de Paiva, Valeria, Everett, John O. and Bobrow, Daniel G. 2002. Scalability of redundancy detection in focused document collections. In *Proceedings First International Workshop on Scalable Natural Language Understanding.*

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John and Newman, Paula. 2007. XLE Documentation, www2.parc.com/isl/groups/nltt/xle/doc/.

Crouch, Dick and King, Tracy Holloway. 2006. Semantics via F-Structure Rewriting. In *Proceedings of LFG06*, pages 145–165, CSLI On-line publications.

Crouch, Richard. 2005. Packed Rewriting for Mapping Semantics to KR. In *Proceedings Sixth International Workshop on Computational Semantics.*

Crouch, Richard and King, Tracy Holloway. 2005. Unifying Lexical Resources. In *Proceedings of the Interdisciplinary Workshop on the Identification and Representation of Verb Features and Verb Classes.*

de Paiva, Valeria, Bobrow, Daniel G., Condoravdi, Cleo, Crouch, Richard, Kaplan, Ron, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan and Zaenen, Annie. 2007. Textual inference logic: Take two. In *Proceedings of the Workshop on Contexts and Ontologies: Representation and Reasoning, Workshop associated with the 6th International Conference on Modeling and Using Context.*

Everett, John O., Bobrow, Daniel G., Stolle, Reinhard, Crouch, Dick, Condoravdi, Cleo, de Paiva, Valeria, van den Berg, Martin and Polanyi, Livia. 2002. Making ontologies work for resolving redundancies across documents. In *Communications of the ACM*, volume 45, pages 55–60.

Fellbaum, Christiane (ed.). 1998. *WordNet: An Electronic Lexical Database.* The MIT Press.

Frank, Anette, King, Tracy Holloway, Kuhn, Jonas and Maxwell, John T. 2001. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In Peter Sells (ed.), *Formal and Empirical Issues in Optimality Theoretic Syntax*, pages 367–397, CSLI Publications.

Gurevich, Olga, Crouch, Richard, King, Tracy Holloway and de Paiva, Valeria. 2005. Deverbals in Knowledge Representation. In *Proceedings of FLAIRS'06*.

Kipper, Karin, Dang, Hoa Trang and Palmer, Martha. 2000. Class-based Construction of a Verb Lexicon. In *AAAI-2000 17th National Conference on Artificial Intelligence.*

MacCartney, Bill and Manning, Christopher D. 2007. Natural Logic for Textual Inference. In *Proceedings of the ACL 2007 Workshop on Textual Entailment and Paraphrasing.*

Maxwell, John and Kaplan, Ron. 1991. A Method for Disjunctive Constraint Satisfaction. In Masaru Tomita (ed.), *Current Issues in Parsing Technologies*, pages 173–190, Kluwer.

Maxwell, John and Kaplan, Ron. 1996. An Efficient Parser for LFG. In *Proceedings of the First LFG Conference*, CSLI Publications.

McCarthy, John. 1993. Notes on Formalizing Context. In *Proceedings of the 13th Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 555–560.

Nairn, Rowan, Condoravdi, Cleo and Karttunen, Lauri. 2006. Computing Relative Polarity for Textual Inference. In *Proceedings of ICoS-5 (Inference in Computational Semantics).*

Riezler, Stefan, King, Tracy Holloway, Kaplan, Ronald M., Crouch, Richard, Maxwell, John T. and Johnson, Mark. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL02*.

van Benthem, Johan. 1986. *Essays in Logical Semantics.* Reidel.

Accommodating Language Variation in Deep Processing

António Branco and Francisco Costa
University of Lisbon

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

We present an approach to handle variation in deep linguistic processing. It allows a grammar to be parameterized as to what language variants it accepts and also to detect the variant of the input. We also report on the evaluation of this approach by having the system detect the dialect of input sentences extracted from corpora of two different dialects.

1 Introduction

This paper proposes a design strategy for deep language processing grammars to appropriately handle language variants of a given language.

In the benefit of generalization and grammar writing economy, it is desirable that a grammar can handle language variants — that is variants which share most grammatical structures and lexicon — in order to avoid the multiplication of individual grammars, motivated by inessential differences.

The design presented here allows a grammar to be restricted as to what language variant it is tuned to, but also to detect the variant a given input pertains to. Evaluation of this design is also reported.

We assume the HPSG framework (Pollard and Sag, 1994) and a grammar that handles two close variants of the same language, European and Brazilian Portuguese. These assumptions are merely instrumental, and the results obtained can be easily extended to other languages and variants, and to other grammatical frameworks for deep linguistic processing.

The HPSG setup for handling variation and the experiments themselves were carried out with a computational HPSG for Portuguese. It is being developed in the LKB (Copestake, 2002) on top of the Grammar Matrix (Bender et al., 2002), and it uses MRS for semantic description (Copestake et al., 2001). This grammar is part of the DELPH-IN Consortium (http://www.delph-in.net).[1]

2 Handling variation

We propose an approach that allows flexibility with respect to variation in the same language and also permits a grammar to be tuned to a particular variant. It relies on the use of a feature VARIANT to model variation. This feature is appropriate for all signs, and its value declared to be of type *variant*. Given the working language variants assumed here for the sake of the evaluation experiment, its possible values are the ones presented in Figure 1.

This attribute is constrained to take the appropriate value in lexical items and constructions specific to one of the two varieties. For example, a hypothetical lexical entry for the lexical item *autocarro* (*bus*, exclusive to European Portuguese) would include the constraint that the attribute VARIANT has the value *ep-variant*,

[1] At the time of the experiments reported here, the grammar was of modest size, resulting from a year and a half of development.

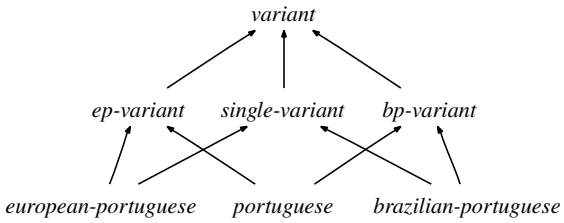


Figure 1: Type hierarchy under *variant*.

and the corresponding Brazilian Portuguese entry for *ônibus* would constrain the same feature to bear the value *bp-variant*. Items that are common to both European Portuguese and Brazilian Portuguese are left underspecified with respect to this feature. They do not have to be constrained with [VARIANT *variant*] because this constraint is defined in the type *sign*, from which all lexical types inherit.

Figure 2 shows examples of these cases, with simplified feature structures. The only two types that are used to mark signs are *ep-variant* and *bp-variant*. The remaining types presented in Figure 1 are used to constrain grammar behavior, as explained below.

$$\begin{bmatrix} \text{STEM} \left\langle \text{“autocarro”} \right\rangle \\ \text{VARIANT } \textit{ep-variant} \end{bmatrix} \quad \begin{bmatrix} \text{STEM} \left\langle \text{“ônibus”} \right\rangle \\ \text{VARIANT } \textit{bp-variant} \end{bmatrix} \quad \begin{bmatrix} \text{STEM} \left\langle \text{“carro”} \right\rangle \\ \text{VARIANT } \textit{variant} \end{bmatrix}$$

Figure 2: Constraints on lexical items. Example of an European Portuguese item (*autocarro — bus*), a Brazilian Portuguese item (*ônibus — bus*) and an item common to both varieties (*carro — car*).

Lexical items are not the only elements that can have marked values in the VARIANT feature. Lexical and syntax rules can have them, too. Such constraints model constructions that markedly pertain to one of the dialects. Section 4 presents a small examination of these differences.

The feature VARIANT is structure-shared among all signs comprised in a full parse tree. This is achieved by having all lexical or syntactic rules unify their VARIANT feature with the VARIANT feature of their daughters (Figure 3).

Since this feature is shared among all signs, it will be visible everywhere, including the root node. It is possible to constrain the feature VARIANT in the root condition of the grammar so that the grammar works in a variant-“consistent” fashion: this feature just has to be constrained to be of type *single-variant* (in root nodes) and the grammar will accept either European Portuguese or Brazilian Portuguese. Furthermore, in the unnatural condition where the input string bears marked properties of both variants (e.g. from lexical items and syntax rules), that string will receive no analysis: the feature VARIANT will have the value *portuguese* in this case (the greatest lower bounds for *ep-variant* and the other *bp-variant*), and there is no unifier for *portuguese* and *single-variant*.

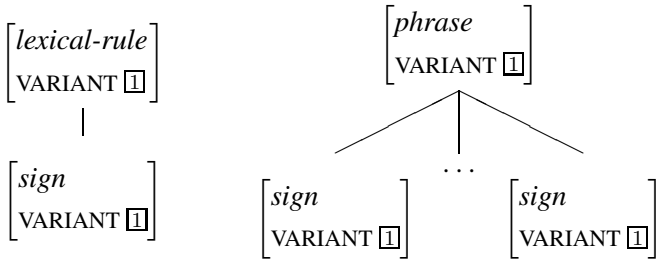


Figure 3: Constraints on rules. Lexical and syntax rules identify the VARIANT feature of the mother with the VARIANT features of all the daughters.

Figure 4 shows an example of this situation, where the marked Brazilian item *dezesseis* (sixteen) co-occurs with the marked European item *autocarros* (buses). This is specially useful in generation, where one may be interested in generating all relevant sentences in either European Portuguese or Brazilian Portuguese, but one does not want to generate sentences with phrases like the one in this example.

If this feature is constrained to be of type *european-portuguese* in the root node, the grammar will not accept any sentence with features of Brazilian Portuguese, since these will be marked to have a VARIANT of type *bp-variant*, which is incompatible with *european-portuguese* (there is no unifier for these two types according to the hierarchy in Figure 1). It is also possible to have the grammar reject European Portuguese sentences in detriment of Brazilian Portuguese ones (by using type *brazilian-portuguese*) or to ignore variation completely by assigning to VARIANT the *variant* value, thus not constraining the VARIANT feature in the start symbol.

The mechanism presented here has the following properties:

- Increased coverage and flexibility. The grammar can handle input from all variants under consideration if the VARIANT feature is constrained with a general type.
- Parameterization. The grammar can be tuned to a relevant dialect by constraining the feature VARIANT with a specific type. This is welcome in parsing, but specially desirable in generation, where the grammar can be configured to generate only in a given selected variant.
- Consistency. If VARIANT is constrained to be *single-variant*, the grammar can deal with all variants, but it will reject sentences with mixed characteristics.

The ability to parse more variants means more coverage, which generally increases ambiguity. The last two properties above are ways to control this kind of ambiguity. If the input string contains an element that can only be found in variety v_1 and that input string yields ambiguity in a different stretch but only in varieties

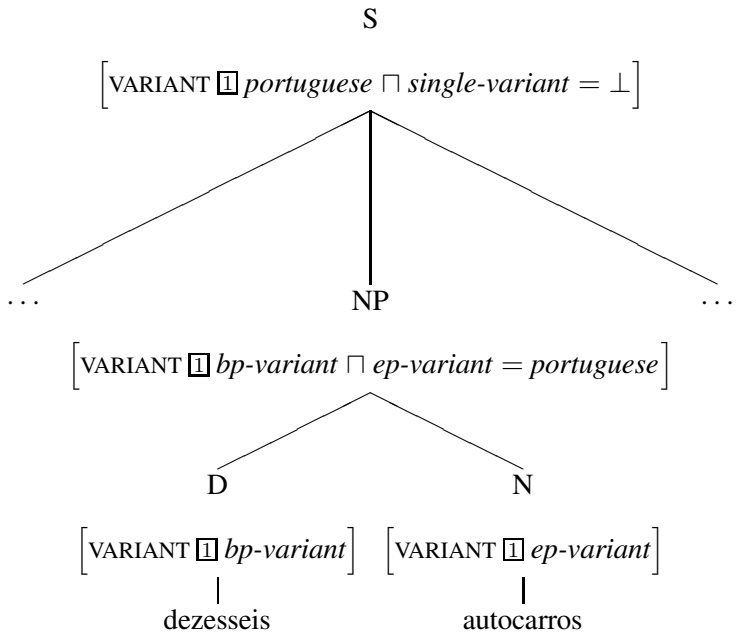


Figure 4: Example of an inconsistency. The noun phrase *dezesseis autocarros* (*sixteen buses*) is inconsistent. It should be either *dezesseis ônibus* (Brazilian Portuguese) or *dezasseis autocarros* (European Portuguese). The constraint on the VARIANT of the root node (to be *single-variant*) rejects the structure.

v_k other than v_1, this ambiguity will not give rise to multiple analyses if the grammar is constrained to accept strings with marked elements of at most one variety.

This can be illustrated with a simple example. The preposition *a* (to, at) is homonymous with a form of the definite article. In European Portuguese, in many contexts definite articles are obligatory before possessives, but in Brazilian Portuguese they are optional in these cases. In Brazilian Portuguese the string *a minha opinião* is ambiguous between the reading corresponding to *my opinion* and *to my opinion*, because of the lexical ambiguity of *a*. The interaction with pro-drop and various word-order possibilities multiplies possible parses as this and similar phrases can be the subject, the direct object, the indirect object or a PP adjunct. But in European Portuguese this string will not be ambiguous between an NP and a PP in contexts where the article is obligatory. In these contexts, only the reading corresponding to *my opinion* will be available.

In general, we can know whether a string is European Portuguese or Brazilian Portuguese if a marked item or construction occurs. Consider a similar example, but where the noun is specific to European Portuguese: for instance *a minha ideia* (*my idea*, the Brazilian Portuguese spelling of the word is *idéia*). If the root node is constrained to have a VARIANT of type *single-variant*, the PP reading is rejected (even when we do not know the specific variant of the input in advance), since the PP analysis is only available in Brazilian Portuguese where the noun is spelled differently. That PP will have a VARIANT of type *portuguese*, which does not unify with *single-variant* in the root node, as was seen before. Figure 5 depicts the corresponding computations.

Variant Detection

With this grammar design it is also possible to use the grammar to detect to which variety the input happens to belong. This is done by parsing that input and placing no constraint on the feature VARIANT of root nodes, and then reading the value of attribute VARIANT from the resulting feature structure: values *ep-variant* and *bp-variant* result from parsing text with properties specific to European Portuguese or Brazilian Portuguese respectively; the value *variant* or *single-variant* (depending on the constraint on the root node) indicates that no marked elements were detected and the text can be from both variants.

Also in this case where the language variant of the input is detected by the grammar, the desired variant-"consistent" behavior of the grammar is enforced if the feature VARIANT is set to *single-variant*. The examples in Figure 5 also illustrate this functioning: the constraint on the feature VARIANT of the marked item *ideia* is propagated throughout the syntactic structure.

Evaluation

It is important to gain insight on the quality of the performance of this method. This is addressed in the next sections. The question we want to find an answer

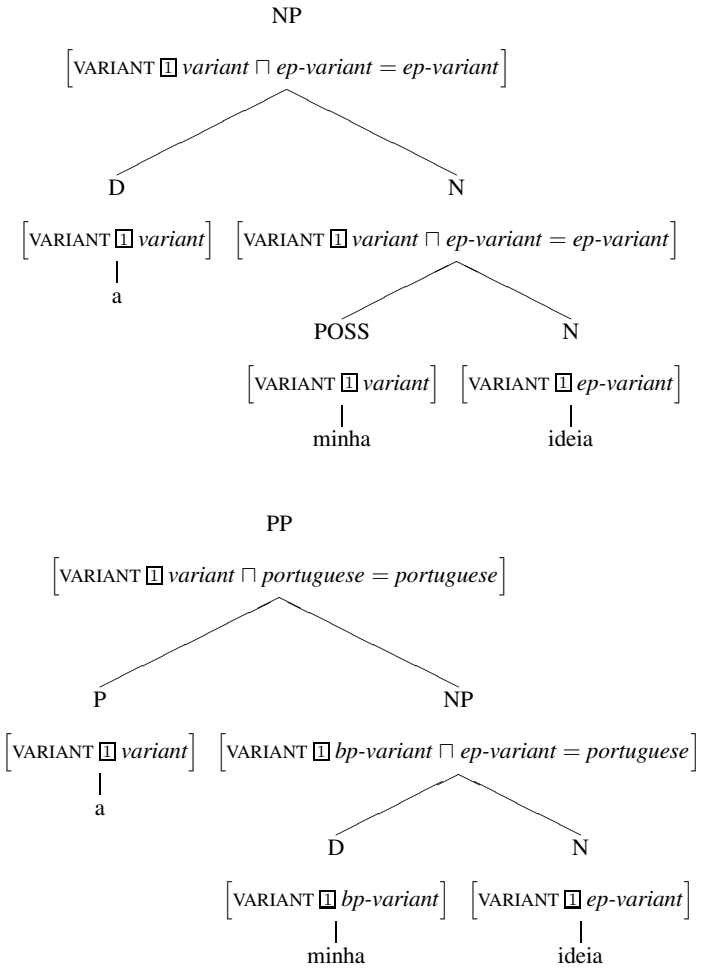


Figure 5: Example of ambiguity specific to one variety. When such ambiguous forms co-occur with items of another variety, they can be resolved by constraining the start symbol with a VARIANT feature of type *single-variant*.

to is: how appropriate is this design for the handling of variation? A simple way to evaluate this design is to parse sentences whose original dialect is known and check whether the grammar can consistently detect the right dialect, by reading off the value of the feature VARIANT in the feature structure for the sentence.

3 Data

To evaluate the approach to accommodate variation presented above, two corpora of newspaper text were used, CETEMPublico (204M tokens) and CETENFolha (32M tokens). The first contains text from a Portuguese newspaper, and the latter from a Brazilian one. These corpora are only minimally annotated (paragraph and sentence boundaries, *inter alia*), but are very large.

Some preprocessing was carried out: XML-like tags, such as the <s> and </s> tags marking sentence boundaries, were removed and each individual sentence was put on a single line. Some heuristics were also employed to remove loose lines (parts of lists, etc.) so that only lines ending in ., ! and ? and containing more than 5 tokens (whitespace delimited) were considered. Other character sequences that were judged irrelevant and potentially misguiding for the purpose at hand were normalized: URLs were replaced by the sequence URL, e-mail addresses by MAIL, hours and dates by HORA and DATA, etc. Names at the beginning of lines indicating speaker (in an interview, for instance) were removed, since they are frequent and the grammar used is not intended to parse name plus sentence strings.

From each of the two corpora, 90K lines were selected, with the smallest length sentences. Of the resulting 90K+90K, 26% were shown to be fully parsable by the grammar and set apart. From these 26%, $1800 + 1800$ sentences were randomly chosen.

If a sentence is found in the European corpus, one can be sure that it is possible in European Portuguese, but one does not know if it is Brazilian Portuguese, too. The same is true of any sentences in the American corpus — these can also be sentences of European Portuguese in case they only contain lexical items and structures that are common to both variants.

In order to address this, a native speaker of European Portuguese was asked to manually decide from sentences found in the American corpus whether they were markedly Brazilian Portuguese. Conversely, a Brazilian informant detected markedly European Portuguese sentences from the European corpus. Thus a three-way classification is obtained: every sentence was classified as being markedly Brazilian Portuguese, European Portuguese or common to both variants.

As a result, 5KB of text (140 sentences) from each one of the three classes were selected for testing, and another 5KB (also around 140 sentences each) for training (development).

Many more sentences were classified as possible in both dialects than as sentences specific to either one. We only kept a subset of the sentences judged to

be common, in order to have a uniform distribution of the three classes in the data. 16% of the sentences in the European corpus were considered impossible in Brazilian Portuguese, and 21% of the sentences in the American corpus were judged exclusive to Brazilian Portuguese. Overall, 81% of the text was common to both varieties. Since a single marked item or construction in a sentence causes it to be classified as marked, we see that a very large part of the language variants overlap (very likely more than 81%).

4 Differences Between European Portuguese and Brazilian Portuguese Found in the Training Corpora

We proceed to an analysis of the training data resulting from the manual classification described in Section 3. A brief typology of the markedly Brazilian elements found in the American training corpus is presented. We also present the relative frequency of these phenomena based on the same data. We do not present the marked items found in the European corpus, because, being native speakers of European Portuguese, we could not always determine the reason why the Brazilian informant marked sentences as specific to European Portuguese.[2]

0. Differences due to lack of orthographic harmonization (33.3%)
   (a) Phonetic or phonological differences reflected in spelling (9.3%)
   e.g. BP *irônico* vs. EP *irónico* (*ironic*)
   (b) Pure spelling differences, no phonemic difference (24%)
   e.g. BP *ação* vs. EP *acção* (*action*)
1. Lexical differences (26.9% of all differences found)
   (a) Different form, same meaning (22.5%)
   e.g. BP *time* vs. EP *equipa* (*team*)
   (b) Same form, different meaning (4.4%)
   e.g. *policial*: BP *police officer*, EP *criminal novel*

[2] Although we were able to extract a large amount of information from the European Portuguese training data as well, by checking possible candidates in dictionaries and web searches, we cannot quantify the different phenomena at stake precisely, as in some cases a decision could not be made. We should have asked the informants to paraphrase the marked sentences in a way that sounded acceptable to them, so that we could have detected the markedly European items and constructions consistently.

2. Syntactic differences (39.7%)

   (a) Co-occurrence of definite articles and prenominal possessives (12.2%)
   BP: Meu pai cuida de tudo.
   my father takes care of everything
   EP paraphrase: O meu pai cuida de tudo.
   the my father takes care of everything
   My father takes care of everything.

   (b) Different subcategorization frames (9.8%)
   Progressive auxiliary *estar* selects for a gerund in Brazilian Portuguese, and preposition *a* plus infinitive in European Portuguese (5.4%)
   BP: O gravador está funcionando?
   the tape recorder is working.GER
   EP paraphrase: O gravador está a funcionar?
   the tape recorder is PREP work.INF
   Is the tape recorder working?

   (c) Clitic placement (6.4%)
   BP: Tommy se apaixona por Betsy.
   Tommy CLITIC falls in love for Betsy
   EP paraphrase: Tommy apaixona-se por Betsy.
   Tommy falls in love CLITIC for Betsy
   Tommy falls in love with Betsy.

   (d) Bare NPs headed by singular count nouns (5.4%)
   BP: Médico também é ser humano.
   doctor also is being human
   EP paraphrase: Um médico também é um ser humano.
   a doctor also is a being human
   A doctor is a human being, too.

   (e) Different subcategorization frame and different word sense (1.9%) e.g. BP *fato* (*fact*, with a sentential complement) vs. EP *fato* (*suit*, no complements)

   (f) Co-occurrence of prenominal *todo* and definite articles (0.9%)
   BP: Todo mundo aqui gosta deles.
   all world here likes of them
   EP paraphrase: Todo o mundo aqui gosta deles.
   all the world here likes of them
   Everyone here likes them.

   (g) Contractions of prepositions and articles (0.9%)
   BP: Eles estão em uma creche da cidade.
   they are in a kindergarten of the city
   EP paraphrase: Eles estão numa creche da cidade.
   they are in a kindergarten of the city.
   They are in one of the city's kindergartens.

(h) Matrix wh-questions without subject-verb inversion or *é que* (0.9%)
BP: O que ele veio fazer aqui?
what he came to do here?
EP paraphrase: O que é que ele veio fazer aqui?
what is that he came to do here
What did he come here for?

(i) Postverbal sentential negation (0.5%)
BP: Mas, felizmente, isso não existe não, bonitinha .
but fortunately that not exists not foxy
EP paraphrase: Mas, felizmente, isso não existe, bonitinha .
But fortunately that not exists foxy
But fortunately that doesn't exist, foxy.

(j) other (0.5%)
BP: Enquanto isso, de dia, trabalhava de alfaiate.
while that at day I worked as tailor
EP paraphrase: Enquanto isso, de dia, trabalhava como alfaiate.
while that at day I worked as tailor
Meanwhile, I worked as a tailor during the day.

Figure 6 presents a pie chart of these differences.

One third of the differences found would be avoided if the orthographies were harmonized (0). Differences that are reflected in spelling can be modeled by the grammar via multiple lexical entries, with constraints on the feature VARIANT reflecting the variety in which the lexical item with that spelling is used. In some cases, a different solution would be preferable. When the difference is systematic (e.g. the European Portuguese sequence *ón* always corresponds to a Brazilian Portuguese sequence *ôn*, with an example in (0a)), it would be better to have a lexical rule that affects only spelling and the VARIANT feature producing one variant from the other.[3]

Orthographic differences, which account for 33.3% of all differences appear in 47.9% of the sentences (in the American training corpus). This means that, by simply looking at lexical items, almost 50% recall could be obtained on these data, assuming perfect lexical coverage.

Some differences cannot be detected by the grammar. This is the case of (1b), which would require word sense disambiguation. When word sense differences are accompanied by different syntax, they can be detected by the grammar (2e) in limited circumstances (e.g., in that example, the difference is detected only if the complement is expressed). This places the upper bound for recall for Brazilian Portuguese between 95.6% and 93.7%, judging by these frequencies.

Interestingly, 40% of the differences are syntactic. These cases are not expected to be difficult to detect by a grammar, but it may be difficult to take advantage of them with shallower methods. Consider the example of clitic placement, illustrated

[3]This was not implemented, because string manipulation is limited in the LKB.

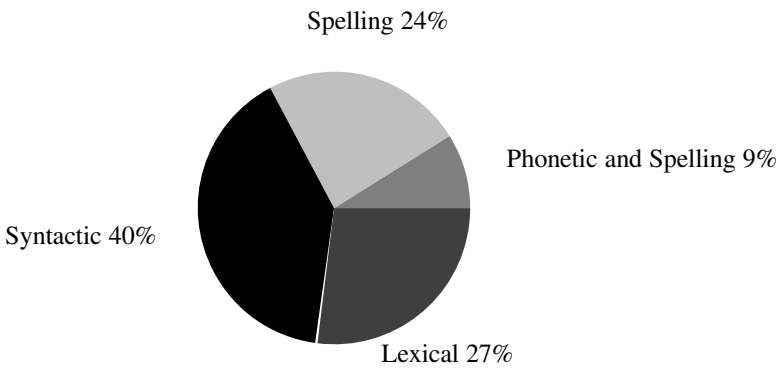


Figure 6: Breakdown by type of the differences detected in the Brazilian Portuguese training corpus.

in (2c). It is not a simple matter of clitics preceding the verb in Brazilian Portuguese and following it in European Portuguese, because they can also precede the verb in European Portuguese depending on the syntactic context (e.g. in finite subordinate clauses they must do so). Therefore, syntactic information is crucial to detect some of the differences found.

Another interesting example is the co-occurrence of definite articles and possessives (2a). Recall from one of the examples in Section 2 that the feminine singular form of the definite article, *a*, is homonymous with a preposition. Syntactic context can disambiguate this situation in several circumstances (e.g. after a preposition that does not introduce an infinitival clause it can only be an article; as an article it cannot introduce an NP headed by a noun that is masculine or plural, etc.).

5 Grammar Preparation

The evaluation experiments were carried out with a computational HPSG for Portuguese developed with the LKB platform (Copestake, 2002) that uses MRS for semantic representation (Copestake et al., 2001). At the time of the experiments reported here, this grammar was of modest size. In terms of linguistic phenomena, it covered basic declarative sentences and basic phrase structure of all categories, with a fully detailed account of the structure of NPs. It contained 42 syntax rules, 37 lexical rules (mostly inflectional) and a total of 2988 types, with 417 types for lexical entries. There were 2630 hand-built lexical entries, mostly nouns, with 1000 entries. It was coupled with a POS tagger for Portuguese, with 97% accuracy (Branco and Silva, 2004; Silva, 2007).

In terms of the sources of variant specificity, this grammar was specifically designed to handle the co-occurrence of prenominal possessives and determiners and most of the syntactic constructions related to clitic-verb order. As revealed by the study of the training corpus, these constructions underlie almost 20% of marked

sentences, and they are the bulk of the syntactic differences.

We present a simplified description of how word-order of complement clitics was controlled by the grammar at the time of the experiments. Basically, several binary versions of Head-Complement rules are used. In the feature structure for these rules there is a boolean feature PROCLISIS indicating whether proclisis (clitics before the verb) or enclisis/mesoclisis (clitics after or in the middle of the verb) is expected according to European Portuguese.[4] The value for this feature is determined by other elements in a sentence. An example: since in finite subordinate clauses proclisis is enforced, complementizers select for a complement with a PROCLISIS feature with the value $+$ (the start symbol is constrained with the value $-$ for this feature, because the unmarked order in matrix clauses is enclisis). There is a Head-Complement construction that ignores this feature and projects a non clitic complement.

The nature of clitics is represented by a feature WEIGHT: clitics have the value *clitic* for this feature, other syntactic constituents have the value *non-clitic*, and there is no unifier for these two types. The value of WEIGHT is lexically specified and always *non-clitic* for phrases.[5] The Head-Complement schema that projects non-clitics has constraints like:

$$\begin{bmatrix} \text{HEAD-DTR } \boxed{1} \\ \text{NON-HEAD-DTR } \boxed{2}\left[\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{WEIGHT } \textit{non-clitic}\right] \\ \text{ARGS } \left\langle \boxed{1}, \boxed{2} \right\rangle \end{bmatrix}$$

The feature ARGS has as its value the list of daughters of a syntactic rule. The order of the elements in this list correlates with word order. The actual value of ARGS is determined by general types in the Matrix (*head-initial* and *head-final*), from which specific syntactic rules inherit, but we present the constraints on ARGS here instead of the relevant supertypes, in order for the word-order patterns in these rules to be visible.

There is a Head-Complement rule that projects a clitic to the left of the verb in proclisis contexts:

[4]The choice between enclisis and mesoclisis depends only on verbal tense and mood and is not relevant for our purposes. The opposition is between proclisis contexts and non proclisis contexts.

[5]The feature WEIGHT is reminiscent of the same feature in Abeillé and Godard (2003), but here different values are used. An equivalent treatment would be to enrich the type hierarchy under *synsem*, so that the distinction between clitics and non clitics is represented via subtypes of *synsem*, as in Miller and Sag (1997). Contrary to much HPSG work on Romance clitics, we chose to have them combine with verbs in syntax rather than in morphology for practical reasons that relate to orthography: the resulting string includes a space whenever the clitic precedes the verb. When clitics follow the verb, a hyphen is used instead, which is removed in a preprocessing step.

$$\left[\begin{array}{l}\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \left[\begin{array}{l}\text{HEAD } \textit{verb} \\ \text{PROCLISIS } + \end{array}\right] \\ \text{HEAD-DTR } \boxed{1} \\ \text{NON-HEAD-DTR } \boxed{2}\left[\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{WEIGHT } \textit{clitic}\right] \\ \text{ARGS} \left\langle \boxed{2}, \boxed{1} \right\rangle \end{array}\right]$$

In order to account for variation with respect to clitic placement, there are thus two versions of Head-Complement rules for clitics in enclisis contexts that are marked with respect to the VARIANT feature and resort to the feature PROCLISIS:

$$\left[\begin{array}{l}\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \left[\begin{array}{l}\text{HEAD } \textit{verb} \\ \text{PROCLISIS } - \end{array}\right] \\ \text{HEAD-DTR } \boxed{1} \\ \text{NON-HEAD-DTR } \boxed{2}\left[\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{WEIGHT } \textit{clitic}\right] \\ \text{ARGS} \left\langle \boxed{1}, \boxed{2} \right\rangle \\ \text{VARIANT } \textit{ep-variant} \end{array}\right]$$

$$\left[\begin{array}{l}\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \left[\begin{array}{l}\text{HEAD } \textit{verb} \\ \text{PROCLISIS } - \end{array}\right] \\ \text{HEAD-DTR } \boxed{1} \\ \text{NON-HEAD-DTR } \boxed{2}\left[\text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{WEIGHT } \textit{clitic}\right] \\ \text{ARGS} \left\langle \boxed{2}, \boxed{1} \right\rangle \\ \text{VARIANT } \textit{bp-variant} \end{array}\right]$$

Turning now to the issue of prenominal possessives, in order to parse items that are not preceded by articles in Brazilian Portuguese, we just added determiner versions of possessives that have a marked VARIANT feature, with the value *bp-variant* (see Figure 5 above).

Finally, the lexicon contained lexical items specific to European Portuguese and specific to Brazilian Portuguese. They were taken from the Portuguese Wiktionary (http://pt.wiktionary.org), where this information is available. Namely, the Portuguese Wiktionary contains the categories "Portuguese spelling" ("grafia portuguesa") and "Brazilian spelling" ("grafia brasileira"), associated with items with specific spellings, and it is possible to list all the items in these categories. Leaving aside items judged to be very infrequent (e.g. *aniónico* / *aniônico* — *anionic*), around 740 marked lexical items were coded. Lexical items that are variant specific that were found in the training corpora (80 more) were also entered in the lexicon.

Known class	*Predicted class*			
	EP	**BP**	**Common**	Recall
EP	53	1	86	0.38
BP	6	61	73	0.44
Common	14	1	125	0.89
Precision	0.73	0.97	0.44	

Table 1: Confusion matrix for variant detection.

6 Results

The results obtained are presented in Table 1. When the grammar produced multiple analyses for a given sentence, that sentence was classified as markedly European, respectively Brazilian, Portuguese if all the parses produced VARIANT with type *ep-variant*, respectively *bp-variant*. In all other cases, the sentence would be classified as common to both. Every sentence in the test data was classified, and the figure of 0.57 was obtained as overall precision and recall.

The results in Table 1 concern the test corpus, of which all sentences are parsable. Hence, actual recall over a naturally occurring text is expected to be lower, given the development status of the grammar used in the experiment. Using the estimate that only 26% of the input sentences receive a parse by the grammar that was employed in these experiments (see Section 3), the actual figure for recall would lie near 0.15 (= 0.57 x 0.26).

Good recall was achieved for Common (89%), which means that the system erroneously commits to one of the variants only 11% of the time.

In contrast, recall for European Portuguese and Brazilian Portuguese was very low (38% and 44% respectively). What has the most negative impact on the recall values for European Portuguese and Brazilian Portuguese is a very high number of European Portuguese and Brazilian Portuguese test items being classified as "Common" (61% of all European Portuguese test sentences and 52% of all Brazilian Portuguese test sentences), because no marked item or construction was found. We believe that this is a consequence of a lack of lexical coverage (see Section 7) of items that are specific to one of the dialects and may also be a consequence of using only two syntactic cues (regarding clitics and possessives). Therefore, improving lexical coverage and taking advantage of more syntactic differences between the two variants should improve recall in this respect. These errors are also responsible for the low precision for the Common class (44%).

Very good precision was obtained for Brazilian Portuguese (97%): the cues used to classify a sentence as Brazilian Portuguese thus seem to be very robust (proclisis in contexts where European Portuguese shows enclisis, absence of definite articles preceding prenominal possessives, marked lexical items).

Precision for European Portuguese was lower (73%). As can be seen from Table 1, most of these errors originate from the system classifying as European Portuguese sentences that the gold standard says are common to both variants.

This situation arises because enclisis is correlated with European Portuguese by the grammar, but this correlation is not very strong in the test sentences (more about this in Section 7).

7 Error Analysis

Limited lexical coverage is responsible for a large proportion of errors: at least 40% of the cases of sentences incorrectly classified were due to lexical items specific to one of the two variants that were not in the lexicon. We used a POS-tagger to guess the category of unknown words, so problems of lexical coverage often did not have an impact on parse coverage. However, the POS-tagger cannot guess whether a word is specific to Brazilian Portuguese or European Portuguese, so these items were underspecified with respect to their VARIANT feature.

Many of these missing lexical items are interesting or challenging. Some involve derivation. The adverbs *tranqüilamente* (Brazilian Portuguese) and *tranquilamente* (European Portuguese) — *calmly* — were not in the lexicon, although their adjectival bases were (Brazilian *tranqüilo*, European *tranquilo* — *calm*). In some cases the morphological process involved seems less productive: Brazilian *gringolândia* (*a place filled with foreigners*) from Brazilian *gringo* (*foreigner*). There is also a case of a noun derived from an acronym, with the acronym showing up in the derived form with a phonetic spelling: *peemedebista* (a member of the Brazilian political party PMDB). Some other missing lexical entries involve multi-word expressions or idioms: European *de jeito* (*of acceptable quality*, literally *of skill*); European *a cores* vs. Brazilian *em cores* (*in color*, using different prepositions).

In some cases the differences are difficult to detect via dictionaries, as they involve only grammatical features. One example is the noun *ioga* (*yoga*), which is feminine in Brazilian Portuguese and masculine in European Portuguese. Also, some differences in spelling only show up in inflected forms (not in the lemma): European *europeia(s)* vs. Brazilian *européia(s)* — *European*, feminine singular (plural), the lemma being *europeu* in both dialects.

It is worth noting that 20 sentences (14 with the class Common and 6 with the class Brazilian Portuguese) were misclassified by the grammar as European Portuguese. 70% of these errors (11/14 for the Common class and 3/6 for the Brazilian Portuguese class) are due to clitic placement according to European syntax. The point here is that clitic placement according to European syntax appears in Brazilian newspaper text as well. In fact, three sentences in the Brazilian Portuguese class presented enclisis (and also characteristics specific to Brazilian Portuguese) and were misclassified as European Portuguese by the grammar for this reason and because the Brazilian Portuguese characteristics were not detected. 11 sentences in the Common class also presented enclisis, and were misclassified by the grammar as European Portuguese because of this. Some of these sentences came from the American corpus, and some from the European one. The justification we find for

enclisis appearing in the Common class (in sentences from the European corpus) is that, since enclisis is possible in Brazilian newspaper text, it is not considered markedly European when it is seen in European newspaper text, so the Brazilian informants did not classify sentences with enclisis as markedly European. This means that there is some interference of genre in these results. While proclisis in contexts where enclisis is expected in European Portuguese is a so good indicator of Brazilian Portuguese text, enclisis in European enclisis contexts is not a good indicator of European Portuguese, as it can also be found in Brazilian Portuguese text.

The remaining sentences misclassified as European Portuguese are due to misspellings in Brazilian text that unexpectedly conform to European orthography. In Brazil a diaeresis is used on *u* (*ü*) when it follows *q* or *g*, precedes *e* or *i* and is pronounced. The errors were due to spellings like *aguentar* (*to bear*) and *tranquilo* (*calm*), instead of *agüentar* and *tranqüilo*.

A very small number of errors (<1%) was due to the lack of case sensitivity in the LKB (month names are capitalized in European Portuguese and not capitalized in Brazilian Portuguese) and word sense differences.

8 Related Work

There is a considerable amount of literature on grammar specialization and grammar porting (Kim et al., 2003).

With the architecture presented here, it is still possible to specialize a grammar to one of the dialects. In fact this can be done automatically by traversing the source files with the lexical entries and the syntactic/morphological rules and eliminating those that are marked to be specific to all but the desired dialect. This can be done for efficiency reasons. If one wants to parse or generate in a specific variant and this elimination is not performed, the constructions and lexical items specific to all others will only be ruled out when the root node is reached. Therefore, it can be much more efficient to eliminate them in the source files altogether. On the other hand, our experiments showed a large amount of overlap between the two dialects under consideration, so we expect that items that are specific to only one of them should not be frequent in practice. Therefore, the added cost of considering both dialects at run time may not be too detrimental as far as efficiency is concerned, but we have not measured the impact of this.

Søgaard and Haugereid (2005) present a proposal similar to ours. They seek to model variation within Scandinavian languages, by resorting to a LANGUAGE feature. Stymne (2006) goes even farther and uses a LANG feature in a grammar for two rather different languages: English and Swedish.

9 Conclusions

In this paper we presented an architecture to model language variation with typed-feature formalisms. The design that was proposed here can allow for parameterization of a grammar to parse or generate only in a given dialect, or parse input consistently only in one dialect even when the language variant of the input is unknown beforehand. At the same time, consistency of analysis can be enforced, and ambiguity controlled. Moreover, this approach also allows the grammar to function as a dialect classifier, as it can be used to detect the language variant at stake.

We proceeded to evaluate this design, using a grammar for Portuguese that accommodates both European Portuguese and Brazilian Portuguese. Our results are promising, and the grammar achieved very high precision in some cases (97% precision when classifying the input as belonging to Brazilian Portuguese). When the grammar classified the input as European Portuguese, it was right 73% of the time, which is another encouraging result. 89% of the sentences that displayed no dialectal characteristics were also correctly classified as common to European Portuguese and Brazilian Portuguese.

In other cases, the results can be improved. Many European Portuguese characteristics were not recognized (resulting in 38% recall for European Portuguese), and neither were several Brazilian Portuguese characteristics (with 44% recall for Brazilian Portuguese). This means that large improvements can be obtained by extending the grammar with more dialect specific lexical items and constructions. In addition, from the several sources of variant specificity, the grammar used here was prepared to cope only with grammatical constructs that are responsible for at most 20% of them. Also the lexicon, that included a little more than 800 variant-distinctive items, can be largely improved.

There are some interesting challenges, too. We came across the classical problems of lexical coverage, like multi-word expressions and new words.

Some differences between variants are not absolute in practical scenarios. An example of this that affected our results is the spelling oscillations between *u* and *ü* after *q* and *g* in Brazilian Portuguese.

Also, textual genre seemed to affect the results, as Brazilian newspaper text presents some syntactic properties of European Portuguese, like clitic word order.

Besides, there are problems beyond a grammar's capacity, like word sense distinctions. Although word sense differences were frequent in the training data (present in 6.3% of all marked Brazilian Portuguese lexical items found), they turned out to be negligible in the errors found in the test data.

These are issues over which more acute insight will be gained in future work, which will seek to improve the contributions put forth in the present paper.

Given the 97% precision achieved for the Brazilian Portuguese class (with a somewhat lower precision for the European Portuguese class, of 73%), we think that our results are the proof-of-concept that an informed approach can produce very good results in this task, using the architecture we presented.

Summing up, a major contribution of the present paper is a design strategy

for type-feature grammars that allows them to be appropriately set to the specific variant of a given input. Concomitantly, this design allows the grammars to identify the variety used in the input.

References

Abeillé, Anne and Godard, Danièle. 2003. The Syntactic Flexibility of French Degree Adverbs. In Stefan Müller (ed.), *Proceedings of the HPSG-2003 Conference, Michigan State University, East Lansing*, pages 26–46, Stanford: CSLI Publications.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Procedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Branco, António and Silva, João. 2004. Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for Portuguese. In Maria Teresa Lino, Maria Francisca Xavier, Fátima Ferreira, Rute Costa and Raquel Silva (eds.), *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC2004)*, pages 507–510, Paris: ELRA.

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

Copestake, Ann, Flickinger, Dan, Pollard, Carl and Sag, Ivan A. 2001. Minimal Recursion Semantics: An Introduction. *Language and Computation* 3, 1–47.

Kim, Roger, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Masuichi, Hiroshi and Ohkuma, Tomoko. 2003. In Emily Bender, Dan Flickinger, Frederik Fouvry and Melanie Siegel (eds.), *Proceedings of ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*, pages 49–56, Vienna, Austria.

Miller, Phillip H. and Sag, Ivan A. 1997. French Clitic Movement without Clitics or Movement. *Natural Language and Linguistic Theory* 15(3), 573–639.

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

Silva, João Ricardo. 2007. *Shallow Processing of Portuguese: From Sentence Chunking to Nominal Lemmatization*. MSc Dissertation, Faculdade de Ciências da Universidade de Lisboa, Lisbon, Portugal.

Søgaard, Anders and Haugereid, Petter. 2005. Implementing Dialectal Variation in Typed Feature Structure Grammars. Unpublished Manuscript.

Stymne, Sara. 2006. *Swedish-English Verb Frame Divergences in a Bilingual Head-driven Phrase Structure Grammar for Machine Translation*. MSc Dissertation, Linköpings Universitet.

Challenges in Interpreting Spoken Military Commands and Tutoring Session Responses

Elizabeth Owen Bratt, Karl Schultz, and Stanley Peters

Stanford University CSLI

Proceedings of the GEAF 2007 Workshop
Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

Various challenges have emerged over several years of grammar engineering for the spoken dialogue interface to the Navy damage control simulator DC-Train and the Spoken Conversational Tutor SCoT-DC, which reviews DC-Train performance. The systems use two methods for finding interpretations for student utterances from the recognized string. First, a Gemini grammar interprets full strings into a complex, structured logical form. A successful Gemini logical form is the preferred interpretation. Next, a robust Nuance natural language grammar looks for any interpretable phrases in the utterances which Gemini could not interpret, and uses heuristics to determine the best set of slots and values. Voice-Enabled DC-Train and SCoT-DC face challenges due to speech recognition errors, disfluent speech, underspecified responses requiring dialogue context for disambiguation, students' varying levels of familiarity and skill at using the preferred military terminology, and the need for coordination with the dialogue manager's strategies for clarification, confirmation and modeling of student uncertainty.

1 Introduction

We examine various types of challenges faced during the grammar development for the spoken language interfaces to the DC-Train Navy damage control simulator (Bulitko & Wilkins 1999) and to the Spoken Conversational Tutor (SCoT-DC) (Schultz et al. 2003), which reviews a student's DC-Train performance. First, we describe the task of damage control and the types of utterances which require coverage. Next, we present an overview of the spoken natural language architecture and systems used. Finally, we review various categories of spoken input that required special strategies, constraints or decisions during grammar engineering.

2 Spoken Commands and Discussion in the Damage Control Domain

Voice-enabled DC-Train (VE-DCT) provides a student the opportunity to play the role of the Damage Control Assistant (DCA) on a DDG-51 destroyer. As DCA, the student is responsible for receiving messages from others on the ship and making decisions about which personnel should take which actions to combat adverse events like fires, flooding, and smoke in any of the 484 compartments on the ship.

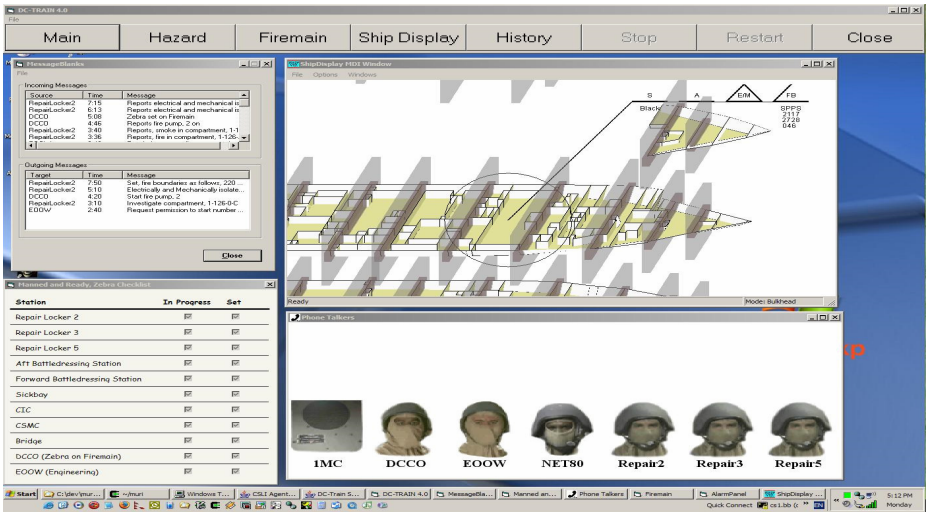


Figure 1. DC-Train Information Windows

The simpler forms of speech to VE-DCT include brief acknowledgements of incoming messages (*affirmative* or *DCA aye*) and brief cancellations of incorrect commands (*cancel that* or *negative*).

In their full form, commands to personnel always involve identification of the addressee. Full commands also identify either the speaker (*DCA*) or the method of communication to the addressee (e.g. *NET80*, for broadcast throughout the ship). Next, full commands contain the desired action and any required parameters, such as a boundary (e.g. *primary aft 97*), a compartment number (e.g. *2-126-2-C*) or a compartment name (e.g. *Combat System Equipment Room Number 2*). This results in commands such as *NET80 to Repair Two electrically and mechanically isolate compartment 1-126-0-C* and *Repair Two, DCA, set smoke boundaries primary forward 42 primary aft 78 secondary forward 18 secondary aft 97 above 1 below 3.* Requests for permission (*EOOW, DCA, request permission to start fire pump number two*) and informative communication (*NET80 to CO, all stations are manned and ready, zebra is set*) are similar in form to commands.

VE-DCT also allows the student to omit the addressee (along with the speaker or method of communication) and any parameters of a command, as long as the student fills in the required parameters in response to system queries. This multi-turn method of issuing commands takes place as seen in Figure 2.

Student: set smoke boundaries
VE-DCT: DCA interrogative for repair team and boundary bulkheads
Student: repair three
VE-DCT: DCA interrogative for boundary bulkheads
Student: primary forward 42, primary aft 78
VE-DCT: DCA interrogative for secondary boundaries
Student: secondary forward 18, secondary aft 97
VE-DCT: To Repair Locker 3, set smoke boundaries as follows: 97, 78, 42, 18, Aye

Figure 2. Interrogative Dialogue in VE-DCT

In SCoT, the student reviews events from the VE-DCT session with an automated tutor, discussing in particular areas where the student did not take required actions in the correct order, the correct selection of repair team by region of the ship, the correct selection of bulkheads to set boundaries for preventing the spread of fire, smoke or flooding, and how to prioritize actions by the type and location of compartments affected.

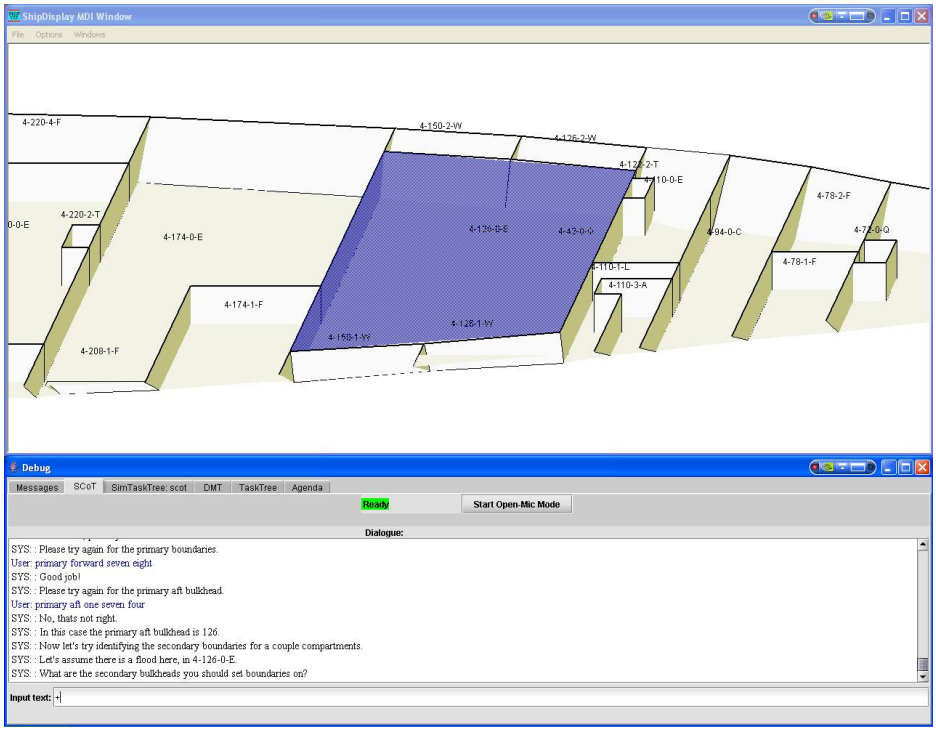


Figure 3. SCoT Display

Since SCoT takes the initiative in leading the tutoring session, the student's utterances are generally responses to tutoring questions, such as

Assuming you have a fire report, there are 2 other things you should have done before ordering fire fighting. Lets begin with the first 1. What is it? The student responses can be very brief (e.g. *investigate*) or take a longer form (e.g. *I guess I should um set fire boundaries first*.) SCoT also permits gestural input in response to some questions, such as clicking on a compartment or circling it on the ship display. SCoT does not have any tutorial discussions in which it would be natural for the student to speak and click or circle compartments at the same time, so natural language interpretation has not had to support gestural constraints, though this capability would be supported by the CSLI dialogue manager architecture (Lemon et al. 2001, Schultz et al. 2003).

In addition to the questions to which students often answer with a short, simple verb or noun phrase, SCoT also asks more open-ended questions, requesting definitions for terms (e.g. *First of all can you tell me what primary boundaries are?)* and reasons for actions (e.g. *Why is it necessary to isolate when you have a report of fire?*). Answers to these questions generally are syntactically more complex, such as *First two bulkheads around the crisis*, *To see if it's a false fire*, and *Prevent smoke from spreading to other compartments*.

VE-DCT and SCoT have been used in a number of experiments and data collections, which have given results for the experimental conditions studied, but also on the range of possible user input and ways it can support student modeling (Jones, Bratt & Schultz 2007).

Theme	# of Subjects	Type of Subject	Year	Results
Different Tutoring Topics at Different Times	30	Stanford students	2004	Pon-Barry et al. 2004, Peters et al. 2004
Natural Language-based Tutoring Strategies	40	Stanford students	2004	Pon-Barry et al. 2006
Multimodality and Active/Passive Tutoring	210	U.S. Naval Academy midshipmen	2005	Bratt et al. 2005
Human Coaching with DC-Train and SCoT	5	Stanford students	2005	Bratt et al. 2005
Human Coaching with DC-Train and SCoT	10	Surface Warfare Officers' School students	2006	No paper yet

Figure 4. Experiments with DC-Train and SCoT

3 Architecture of the Natural Language Interface

After the spoken input into VE-DCT and SCoT is transformed into a string of words by Nuance speech recognition, the natural language understanding takes place in one of two components, as shown in Figure 5.

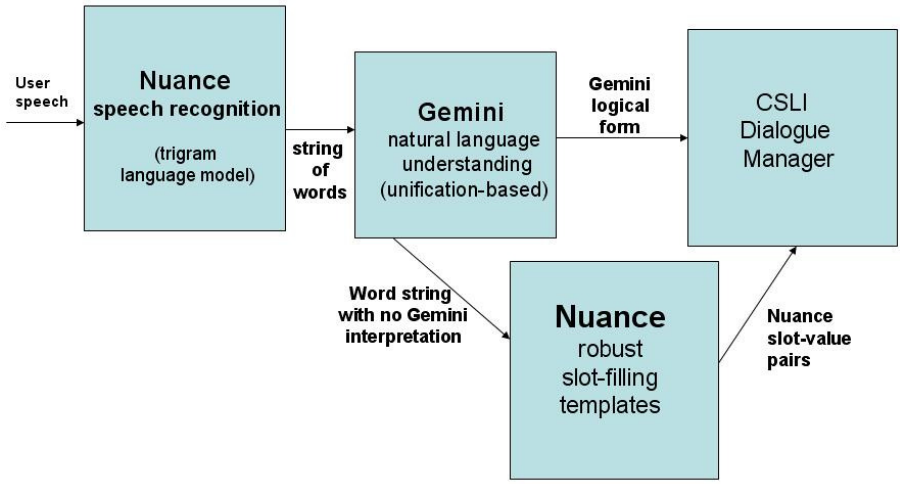


Figure 5. Paths of Interpretation

If the string is well-formed in our Gemini unification-based grammar (Dowding et al. 1993), then it receives a Gemini logical form, from which the Dialogue Manager will extract relevant information. If Gemini cannot interpret the string, perhaps because it has disfluencies or previously unencountered phrasings, then a robust slot-filling Nuance grammar will look for the maximum number of words it can interpret with the minimum number of grammar rules, and return slot-value pairs for its best interpretation.

Our model for these two forms of interpretation is that Gemini is intended to be a linguistically motivated grammar, which uses phrases like NP (noun phrase) and VP (verb phrase), and builds a logical form (LF) capable of representing embedding and other complex relationships. The Nuance slots are intended as a fallback mechanism, which capture partial meaning that is helpful when we cannot understand the entire utterance. The Nuance slots are mainly a flat representation, though certain items, such as boundaries do involve nesting of a single level of structure (rank, direction and frame number for each boundary). Because the Nuance slots involve filling slots from phrases, in sentences which did not parse in Gemini, the emphasis is on

very local interpretation. This means that very particular, idiosyncratic patterns are easy to include in Nuance, since there is less chance of their affecting rules elsewhere, than in a more interconnected, richer system like Gemini. The Nuance slots are close to the domain representation used by the dialogue manager, so they allow for a quick development cycle, with little effort spent compared to the effort needed for any new complexities in a Gemini logical form.

One characteristic of the Nuance slots which can severely limit their utility for certain kinds of discussions is that they permit only one instance of a slot to be filled per sentence. If there are multiple items of the same kind in the same utterance, Nuance natural language rules must provide separate slots for each item so that all of them can be interpreted individually. Without this provision, the information in the slot would be overwritten by each new phrase that matched the Nuance rule, and the only slot information provided to the dialogue manager would be that of the final eligible phrase. We encountered this situation with boundaries, and we defined slots for the usual number of up to four boundaries within an utterance to account for it. But we do at times run into problems with actions, since usually there is only one per utterance, and if there are two, the later one will overwrite the information from the earlier one. Another area where Nuance's behavior of overwriting later slots gives us trouble is when students issue commands with conjunctions, such as *Set fire and smoke boundaries*, *Fight fire in compartments 1-174-01-L and 3-116-1-T*, or *Set flooding boundaries and electrically and mechanically isolate compartment 3-310-2-L.* Students rarely use conjunctions, so this is not a frequent problem, but it has happened at times. The main place students use conjunctions is for pairs of related boundaries, such as both primary boundaries or the above and below decks for vertical boundaries, and our robust interpretation treats these conjunctions the same as the list of boundaries we expect.

The two layers of interpretation for robustness and confidence, i.e., Gemini first, then Nuance, have served our system fairly well. In recent error analysis, we have considered the possible utility of an additional layer of less reliable Nuance slots, so that a complete Gemini LF would be preferred, then the Nuance slot-values which seem fairly reliable, then if there are none of these, we could use the less reliable Nuance slots to start a clarification dialogue with the user, rather than simply asking the user to repeat the utterance.

Our current model of interpreting Nuance slots requires an "action" slot to be filled, indicating what kind of command is intended, and allows the parameters to be filled in later. However, if a set of parameters slots are filled, but the action is not, we might consider clarifying with the user if a particular action was intended. One reason for our current requirement of an "action" slot is that many parameters can involve numbers as values, and many numbers are short words (e.g. *two* or *eight*) which can result from misrecognitions. Thus it is very easy to have a misrecognized sentence which

appears to have parameter values of short numbers, when this is not actually the case. If we had less reliable, back-off Nuance slots, we might have those slots only filled by numbers with identifiers such as *frame number*, *pump number*, etc., as opposed to the numbers which occur in a more standard, full command.

The perspective of what the dialogue manager will do with an interpretation is important to keep in mind. If the dialogue manager requires an "action" slot to be able to clarify a partially understood command, it is particularly important to make the interpretation rules yielding action slots robust, such as making sure they work if there are filled pauses in likely places, or other possible word variations.

Our Gemini grammar currently includes 183 grammar rules, 949 one-word lexical entries, and 2991 multi-word lexical entries. The vocabulary includes 51 action verbs (some synonymous), 36 lexical items for ship personnel, 396 compartment names, 2152 frame numbers (for compartments, bulkheads, valves, etc.), and 23 synonyms for *yes*. In earlier versions of our system, we compiled the Gemini grammar into a language model for Nuance speech recognition (Moore 1998). This kept our coverage for speech recognition and natural language understanding tightly synchronized, and allowed development for the natural language understanding to automatically produce speech recognition results; however, as our system coverage grew, we experienced various problems which led us to abandon this approach. Specifically, compiling the Gemini grammar into a Nuance grammar took a long time, as long as a day, which made it difficult to address bugs or test out alternate options rapidly. Also, as the grammar grew more complex, speech recognition began to get significantly slower than real time, which made it less practical for a dialogue system. Finally, certain Gemini grammars would produce Nuance grammar files that would fail either at Nuance compile time or Nuance run time without explanation.

Currently, we train a trigram model on a corpus of utterances from our past experiments, using Nuance's standard tools to create a probabilistic finite state grammar. The entire process is much faster, and can be completed in under 15 minutes, so we have a more responsive development cycle. Using trigram recognition has improved our speech performance by making us more robust to unanticipated phrasings and out of vocabulary words. We currently train a DC-Train language model on a corpus with 11,551 utterances, containing 390 unique words, and a SCoT language model on a corpus with 19,123 utterances, containing 1780 unique words. The SCoT corpus has more unique words than the DCT corpus because in addition to the words for the basic commands to the simulator, SCoT involves discussion of why to take certain actions, definitions for terms, and discussion of which kinds of compartments should be prioritized over others.

3.2 Tools to Support Grammar Development

In order to support grammar development, as well as speech recognition language model development, we have developed a number of tools. Our most powerful tool is our Nuance Batchrec Analysis Tool, batchdb.[1] Batchdb reads in results of a Nuance batch recognition run, calculates performance metrics (such as word error rate), and stores the results in a MySQL database. With batchdb, we can use SQL queries to compare data from batch runs with different language models or different Nuance parameter settings. We look to optimize for the lowest word error rate, the most accurate semantic slots, and recognition run-time under real time.

Batchdb enters various different views of the data automatically into the database, so that it is easy to work with the exact representation needed for the task at hand. For example, our transcriptions include annotations within square brackets such as *[annoyed]* or *[pause]*. For some data analysis, we are interested in those annotations; for others, we only want to see the transcribed words. Similarly, in some data views we use expletives transcribed as they were spoken, but when we use data with expletives in the language model we sanitized them all into the single word *expletive*, which we give all its various possible pronunciations, so that users of the system will not see any actual expletives on the system display, though our capability to recognize them helps us deal with the full range of military speech!

Batchdb also automatically standardizes the spelling in the transcriptions. We have set forth policies on how transcribers are to enter words, but we have found that it can be hard for transcribers to keep them in mind, so it is good to have automatic processing to eliminate spurious distinctions such as *ok* vs. *okay*, and versions of Navy words with the individual letters treated as distinct words, e.g. *d. c. c. o.* (for *Damage Control Console Operator*), vs. our preferred treatment of them as a single word, e.g. *dcco*.

Batchdb also pre-processes the data before running the sclite speech recognition scoring program. Sclite allows variant words to be scored as equivalent. We have instances of homophones which are sometimes able to be disambiguated by sentence context and other times not. For example, *two four* is right, while *two fore* is wrong, but *fore* vs. *four* in an isolated utterance are equivalent. We keep the versions of the word distinct in our transcript, where the dialogue context can distinguish them, but when we score our speech recognition using a single language model for all contexts, we are able to penalize errors that the language model could detect while overlooking cases of homophone mismatch that speech recognition has no capability to deal with. (Our dialogue manager can automatically correct for these cases.)

---

[1] Available to the public under the GPL open-source license at http://sourceforge.net/projects/nuance-batchrec .

Other tools we use to support our development are a Transcriber Wavefile Preparation Tool, a perl script which takes individual wavefiles from a dialogue and creates a session wavefile and transcript file for Transcriber software (Barras et al. 2000). We use a Transcript Tracker web application to manage transcription file status and coordinate between transcribers, so that they do not duplicate effort, and to provide us a means for automatically standardizing variants or calling the transcriber's attention to items that should be fixed before the transcription is complete.

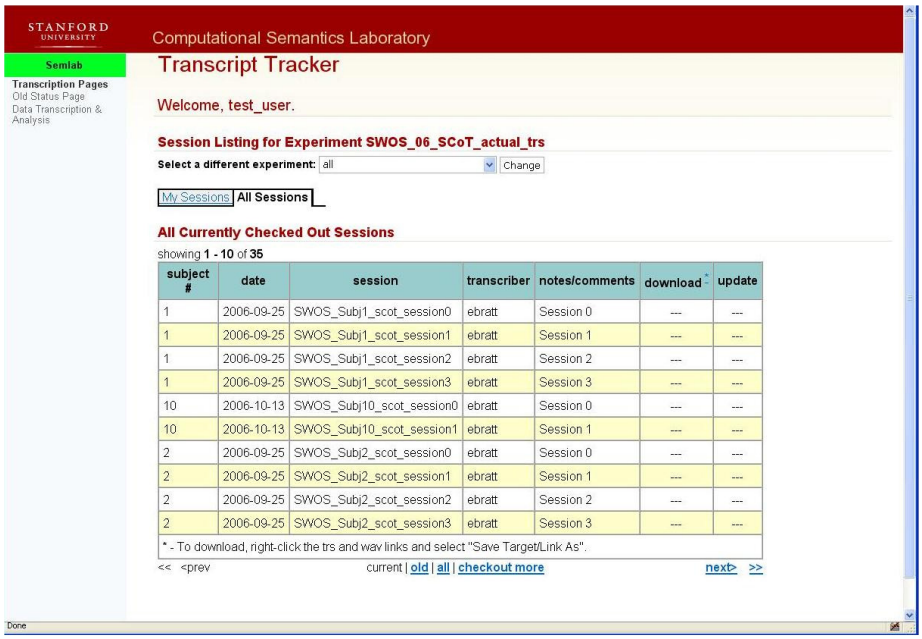
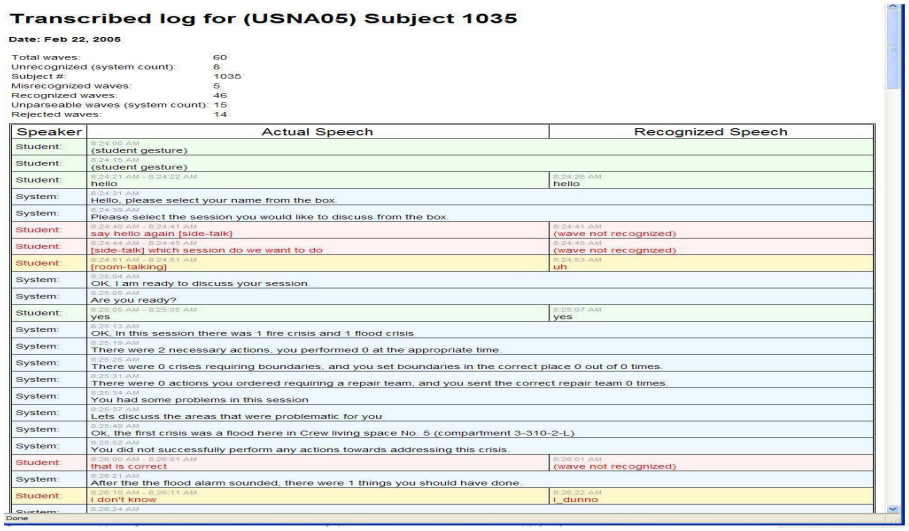


Figure 6: Transcript Tracker

We also use a Transcript Preparation Tool, which combines the automatically logged system transcript with transcriptions from Transcriber .trs files for html viewing, to help us review system performance at a dialogue level. Figure 7 shows an example of the XML-based transcripts produced.



Transcribed log for (USNA05) Subject 1035

Date: Feb 22, 2005

Total waves:	60
Unrecognized (system count):	8
Subject #:	1035
Misrecognized waves:	5
Recognized waves:	46
Unparseable waves (system count):	15
Rejected waves:	14

Speaker	Actual Speech	Recognized Speech
Student:	8:24:00 AM (student gesture)	
Student:	8:24:15 AM (student gesture)	
Student:	8:24:01 AM - 8:24:22 AM hello	8:24:26 AM hello
System:	8:24:31 AM Hello, please select your name from the box.	
System:	8:24:38 AM Please select the session you would like to discuss from the box.	
Student:	8:24:40 AM - 8:24:41 AM say hello again [side-talk]	8:24:41 AM (wave not recognized)
Student:	8:24:44 AM - 8:24:45 AM [side-talk] which session do we want to do	8:24:45 AM (wave not recognized)
Student:	8:24:01 AM - 8:24:51 AM [room-talking]	8:24:53 AM uh
System:	8:25:04 AM OK, I am ready to discuss your session.	
System:	8:25:05 AM Are you ready?	
Student:	8:25:05 AM - 8:25:05 AM yes	8:25:07 AM yes
System:	8:25:13 AM OK, In this session there was 1 fire crisis and 1 flood crisis	
System:	8:25:19 AM There were 2 necessary actions, you performed 0 at the appropriate time.	
System:	8:25:25 AM There were 0 crises requiring boundaries, and you set boundaries in the correct place 0 out of 0 times.	
System:	8:25:31 AM There were 0 actions you ordered requiring a repair team, and you sent the correct repair team 0 times.	
System:	8:25:34 AM You had some problems in this session	
System:	8:25:37 AM Lets discuss the areas that were problematic for you	
System:	8:25:46 AM Ok, the first crisis was a flood here in Crew living space No. 5 (compartment 3-310-2-L)	
System:	8:26:00 AM You did not successfully perform any actions towards addressing this crisis.	
Student:	8:26:00 AM - 8:26:01 AM that is correct	8:26:01 AM (wave not recognized)
System:	8:26:21 AM After the the flood alarm sounded, there were 1 things you should have done.	
Student:	8:26:10 AM - 8:26:11 AM i don't know	8:26:22 AM i_dunno
System:	8:26:24 AM	

Figure 7: XML Transcript of Session

We use a Transcript Search application, based on our Transcript Indexing Tool, which allows a user to search our XML transcripts for instances of words or regular expressions, in modes which include or exclude items such as square-bracketed annotations, word fragments, and punctuation. The search results are displayed as an index web page with links to the dialogues containing the item searched for, as shown in Figure 8. Searching transcripts this way is useful for examining dialogue performance for particular questions, for example.

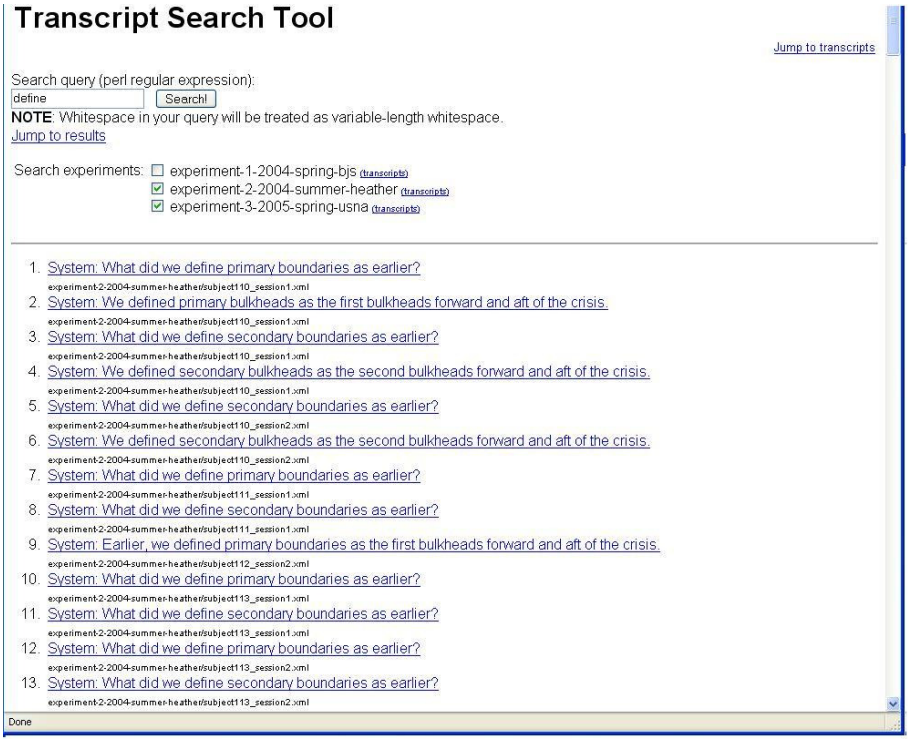


Figure 8: Transcript Search Tool Result Page

4 Interpretation Challenges for Grammar Engineering

4.1 Issues in Training

Because VE-DCT and SCoT are intended to train students, the spoken interfaces need to permit student mistakes. For VE-DCT, the system needs to understand nonexistent compartments and boundary locations, which the student might construct by using the standard format for these items, because simply not recognizing or not understanding the utterance would not help the student realize that the compartment did not exist; rather, it would look like a general system failure. For SCoT, the tutor needs to understand likely wrong answers as well as the correct, expected answer.

Another aspect of training people with a spoken system is how they relate to the tutor or spoken system, and how much they treat it as a person (Nass & Brave 2005, Reeves & Nass 1996). In using our system in a classroom setting at the U.S. Naval Academy, students produced all of the specific examples of uncooperative speech shown in Figure 9, and many other examples besides these. Students who produced uncooperative utterances did not

have significantly different DC-Train performance, overall test scores, or learning gains. At the time of our experiment, our tutor did not understand these utterances, and treated them identically to a student utterance of *I don't know*, and moved on. Providing grammar coverage and suitable interpretations or categorizations of this kind of utterance would be a significant additional task for the grammar engineer; however, in a training system used in the real world, this may be a concern worth addressing.

Polite Complaint	*I can't see what compartment you're talking about*
Swearing	*F--- you*
Insult (to the system)	*I hate you*
Threat (to the system)	*I ought to pulverize your guts out*
Mocking	*There you go it only took you four times*
Inappropriate Response	*Have a beer*
Reaction to Reprimand	*I bet you couldn't do any better man*
Intentionally Using Another Language	*Siete*
Generally Antagonistic or Unresponsive	*This is dumb*

Figure 9. Types of Uncooperativity Encountered by SCoT

4.2 Issues with Military Domain

Another challenge for grammar engineering for a system that involves a simulation of a situation the student may be familiar with in reality is that the student may use more domain knowledge than the simulator supports. For example, the student may not only give a *desmoke* command, which our system would support, but then go on to specify that box fans should be used for desmoking, which is beyond the scope of the simulator. Another area where students familiar with actual ships might use their real world knowledge in ways that make grammar engineering more difficult is to use synonyms in complex compartment names, such as *berthing* for *living space* or *Chief* for *CPO*. Dialogue context might also make it natural to omit certain parts of complex compartment names, such as numerical or directional identifiers, like *number one, forward, aft, port*, or *starboard*. The same compartment might be called *CPO Berthing, Chief's Berthing* or *CPO Living Space Number One.*

Another issue in a training system is how to support standard vs. nonstandard terminology and phrasing. For example, a hyphen written between numbers in compartment identifiers is pronounced as *tac*, but a student less

familiar with military terms might pronounce it as *dash*, or omit it. Similarly, letters used in compartment names are pronounced according to the Navy's phonetic alphabet, e.g. *Charlie*, whereas someone unfamiliar with the convention might just pronounce the letter *c*. Another Navy pronunciation convention is to spell out each digit of a number individually, to minimize misunderstandings such as *fifty* vs. *fifteen*. Thus, the standard pronunciation of *220* would be *two two zero*, not *two hundred twenty*. For a compartment number such as *1-126-0-C*, in addition to the standard pronunciation of *one tac one two six tac zero tac charlie*, there are many possible non-standard pronunciations, such as *one dash one hundred twenty six dash oh dash c*, *one tac one twenty six tac zero tac c*, and so on. We give users an introduction to our system which explains how to pronounce these items, and military users are familiar with these conventions, so most users will produce the standard pronunciations. However, sometimes they will produce the non-standard pronunciations. The challenge is to recognize those times correctly, while not having the non-standard options overwhelm the correct ones, and lead to misrecognitions. Another challenge is the degree to which to represent the non-standard choices in the interpretation, as material for the tutor or coach to comment on.

A similar issue in interpretation in a training dialogue system is how much to assume from the student's wording, how much to clarify, and how to convey to the student that a wording is not ideal. For certain commands, the correct method of delivering them is to include specifications of exactly the relevant areas the commands apply to, or the precise type of action they mean. For example, in the context of the DC-Train simulator, students should be giving the command *electrically and mechanically isolate*, not just the command *isolate*. Similarly, they should give one of the commands *Set fire boundaries*, *Set smoke boundaries*, or *Set flooding boundaries*, not the command *Set boundaries* without specifying which kind. Again, it becomes a question of how much to support the non-standard variants, which do occur, without biasing the system to accept them too much, and how to let the student know not to use them. The simple answer of having the system not understand non-standard variants generally just leads to frustration and the perception of the system as failing, rather than calling attention to the fine points of difference in the student's answer.

Another area of interaction of requiring the student to give a proper full specification and what might in reality be identifiable from dialogue context is the military doctrine that commands should identify the addressee and speaker, or communications system to be used for the addressee. Thus, a proper, full command with the addressee *repair three* and speaker *DCA* would be *Repair three, DCA, investigate compartment two tac two two zero tac oh one tac lima*. A proper full command instructing a phone talker (communications assistant in damage control central) to use the Net-80 communications system, which broadcasts throughout the ship, rather than a lim-

ited radio address only to the specific addressee (in this case the CO, or ship's captain), would be *Net eighty to CO, all stations manned and ready, zebra is set*.

A rich domain can also produce a wide range of possible answers if questions are open-ended. In our experiments at Stanford, USNA, and SWOS, we asked students to provide definitions of terms and to explain why certain actions should be taken. Fortunately for us, the answers students gave ended up being reasonably tractable for interpretation. In the Stanford confidence-sensitivity experiment (2004) and the USNA experiment (2005), the mean utterance length for the 725 answers to open-ended questions was 5.17 words.

4.3 Issues with Spoken Dialogue Context

Because VE-DCT and SCoT involve a system using natural language for its interactions with the student, specifically, speaking to the student in a synthesized voice (Festival for VE-DCT, Taylor et al. 1998, and FestVox limited domain voice for SCoT, Black and Lenzo 2003), grammar development has to account for the possibility that the student is likely to use vocabulary or phrases used by the system. Development of system output must be coordinated with development of the speech recognition language model and the interpretation grammar.

In a system with various numerical parameters, and a series of commands and spoken interactions that create a dialogue context, it becomes natural for the student to use numbers without modifiers or units. The representation of the meaning of these numbers has to allow for their ultimate interpretation by the dialogue manager as specifying the correct kind of item, and not specifying something incorrect. When our system involves the dialogue context of the system issuing an "interrogative" request to the user for missing parameters, we have interpreted the numbers as filling the specific parameters we need, and have ignored the possibility of their filling other parameters. The grammar produces multiple interpretations for bare numbers, such as *two*, which can be interpreted as a pump number, a repair party identifier, a frame number or a deck number. The dialogue manager decides which type of number is likely, and interprets the number accordingly. An alternate method of this would be to have distinct grammars which the system switches between based on dialogue context. In either approach, the dialogue manager must control the interpretation, either by choosing a narrower grammar or by choosing from information provided by the grammar. The complex logical form constructed by Gemini lends itself more naturally to the approach where the dialogue manager chooses the grammar in advance, or an approach where the grammar provides an underspecified number which the dialogue manager adds type information to. The Nuance slot approach can give an underspecified number, but it can also give a series of unrelated slots,

which provide more information than can possibly all be true at the same time. This approach allows the grammar to use any constraints from wording that might be possible to give the dialogue manager an exact set of possible ways to interpret the underspecified utterance. Our Gemini grammar currently gives a set of different interpretations, of which only one is provided to the dialogue manager. If that one does not match the kind of information expected in the dialogue context, the dialogue manager checks the Nuance interpretation for a slot matching the information it expected. It ignores all slots present that are not what it is looking for. The Nuance interpretations involve several different slots, and the dialogue manager adds in several possible slots through reasoning.

4.4 Issues with Speech Input

Our speech recognition word error rate has tended to be around 8% in most of our experiments, though it varies by speaker. Our rejection rate, i.e. the percentage of sentences which our recognizer cannot produce a hypothesized string of words it is sufficiently confident in, is around 4%.

Spoken input can lead to a number of challenges for a grammar, as long as the recognition possibilities are more than what is covered by the grammar. If the utterances involve disfluencies or out of vocabulary items, the recognized string will be an unsuccessful attempt to match what the speaker actually said, so the various places where the grammar backs off to acoustically similar items might produce unexpected results. Acoustic similarity between words can also produce unanticipated strings for the grammar writer to capture, such as *blue* being misrecognized for *below* in our data, as well as other confusion pairs such as *l* and *alpha*, or *set* and *send*. Thus, interpretation rules are more robust when they allow for these.

A general issue for all grammar coverage is dealing with unexpected phrases, though in spoken input the fact that the phrases pass through a speech recognizer first may introduce additional problems if the resulting string does not match what the speaker says.

A disfluency involving repeated words or filled pauses (*um*, *uh*) might present difficulties for a grammar expecting particular phrases with particular constituents without interruptions. Incomplete sentences present another variant of this problem, when a student either is cut off by the speech recognizer endpointer by pausing too long, or the student actually stops speaking part-way through a command. For example, 8% of utterances in the USNA corpus that involve a student beginning to give a compartment number are broken off before the compartment number is complete, and 13% of utterances with a repair party identified break off before giving the repair team their command. It would be desirable to interpret the material actually said, but the phrases will not be complete. Noting where the student breaks off may also help model that this kind of information may be difficult for the student to figure out (Jones, Bratt & Schultz 2007).

Another area in which the spoken input interacts with the grammar interpretation is when setting the word transition weight parameter for the speech recognizer. This parameter can be set so that the speech recognizer is discouraged from hypothesizing short words to match the acoustics of the speech signal in favor of interpreting the same material as part of existing words. This can interact with grammar interpretation of variant formulations of utterances. In our 2006 data collection at SWOS, a student said to the system *Repair repair five send investigators to engine room number two*, which we recognized as *Repair five investigate niner two engine room number two*. Our grammar covered *investigate*, but not *send investigators to*, so we were fortunate that the speech recognition chose a hypothesis that caught the main word and not the more verbose variant form of the command.

Another point about spoken input is that the speech channel allows the user to provide more information than just the straight content of the words (Pon-Barry et al. 2006), as they would if they were choosing a command or an item from a menu. Locations of pauses before items (Jones, Bratt, & Schultz 2007), or the fact that nonstandard vocabulary has been used, can signal additional information that a tutoring system can use to better model what the student knows well and what the student may be struggling with. The interpretation of an utterance given by a grammar can either contain this kind of information, e.g. for nonstandard phrasing, or provide a representation which helps support information from separate processing, as in pause detection.

5 Conclusions

Data from users of VE-DCT and SCoT over the course of multiple experiments provide us with challenges for grammar engineering in various of the complex aspects of our training system: training, representing a complex military domain, recognizing speech, and interpreting speech in dialogue context. Our system architecture provides a preference for canonical input that can be given a complete, structured interpretation, but allows a fallback to robust interpretation of phrases providing partial information. Various system development tools help us detect problems and gauge the success of the solutions we implement.

Acknowledgements

This work is supported by the Department of the Navy under research grant N000140010660, a multidisciplinary university research initiative on natural language interaction with intelligent tutoring systems and research grant N000140510144, on Spoken Language Coaching During Dy-

namic Problem Solving. We are grateful to Perry McDowell of the Naval Postgraduate School, and Lt. Jonathan Hopkins and Lt. Tyson Young of the Surface Warfare Officers School for providing valuable information about the language of the damage control domain. We are responsible for any errors in our use of their information.

References

Barras, Claude, Geoffrois, Edouard, Wu, Zhibiao, and Liberman, Mark. 2000. Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication special issue on Speech Annotation and Corpus Tools*, Vol 33, No 1-2, January 2000.

Black, Alan W. and Lenzo, Kevin A. 2003. Building Synthetic Voices for FestVox 2.0 Edition. Available at http://www.festvox.org/bsv/

Bulitko, Vadim V., & Wilkins., David C. 1999. Automated instructor assistant for ship damage control. In *Proceedings of AAAI-99*.

Dowding, John., Gawron, Jean Mark, Appelt, Doug, Cherny, Lynn, Moore, Robert and Moran, Doug. 1993. *Gemini: A Natural Language System for Spoken Language Understanding*. In the Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics, Columbus, OH.

Jones, Bevan, Bratt, Elizabeth Owen, and Schultz, Karl. 2007. The Prosody of Uncertainty for Spoken Dialogue Intelligent Tutoring Systems. Manuscript in preparation.

Lemon, Oliver, Anne Bracy, Alexander Gruenstein, and Stanley Peters. 2001. The WITAS Multi-Modal Dialogue System I. In *Proceedings of EuroSpeech 2001*.

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

Nass, Clifford. and Brave, Scott. 2005. *Wired for speech: How voice activates and advances the human-computer relationship*. Cambridge, MA: MIT Press.

Pon-Barry, Heather, Schultz, Karl, Bratt, Elizabeth Owen, Clark, Brady, and Peters, Stanley. 2006. Responding to Student Uncertainty in Spoken Tutorial Dialogue Systems. In *International Journal of Artificial Intelligence in Education (IJAIED)* Volume 16, 171-194. Special Issue "Best of ITS 2004" (editors James Lester, Rosa Maria Vicari and Fabio Paraguaçu).

Reeves, Byron. and Nass, Clifford. 1996. *The media equation: How people treat computers, televeision, and new media like real people and places.* New York: Cambridge University Press/CSLI.

Schultz, Karl, Bratt, Elizabeth Owen, Clark, Brady; Peters, Stanley, Pon-Barry, Heather, and Treeratpituk, Pucktada. 2003. A Scalable, Reus-

able Spoken Conversational Tutor: SCoT. In *AIED 2003 Supplementary Proceedings*, University of Sydney. 367-377.

Taylor, Paul A., Black, Alan, and Richard Caley. 1998. The architecture of the Festival speech synthesis system. In *The Third ESCA Workshop in Speech Synthesis*, pages 147-151, Jenolan Caves, Australia.

U.S. Navy Guide to the phonetic alphabet. http://www.history.navy.mil/faqs/faq101-1.htm

The Penn Lambda Calculator: Pedagogical Software for Natural Language Semantics

Lucas Champollion, Joshua Tauberer and Maribel Romero

University of Pennsylvania

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes a novel pedagogical software program that can be seen as an online companion to one of the standard textbooks of formal natural language semantics, Heim and Kratzer (1998). The *Penn Lambda Calculator* is a multifunctional application designed for use in standard graduate and undergraduate introductions to formal semantics: Teachers can use the application to demonstrate complex semantic derivations in the classroom and modify them interactively, and students can use it to work on problem sets provided by the teacher. The program supports demonstrations and exercises in two main areas: (1) performing beta reduction in the simply typed lambda calculus; (2) application of the bottom-up algorithm for computing the compositional semantics of natural language syntax trees. The program is able to represent the full range of phenomena covered in the Heim and Kratzer textbook by function application, predicate modification, and lambda abstraction. This includes phenomena such as intersective adjectives, relative clauses and quantifier raising. In the student use case, emphasis has been placed on providing "live" feedback for incorrect answers. Heuristics are used to detect the most frequent student errors and to return specific, interactive suggestions.

1 Introduction

1.1 Background

For almost ten years now, the textbook by Heim and Kratzer (1998) (henceforth HK) has enjoyed a remarkable success as the textbook of choice for many introductory courses in natural language formal semantics. The semantic framework it presents can be seen as a standardization of Montague-style semantics (Montague, 1974) when applied to Generative Grammar syntax, with lexical items corresponding to simply typed lambda calculus expressions and with a very small number of composition rules. This framework has become a de facto standard in which much formal semantic work has been expressed over the last decade.

Teaching formal semantics can be a challenging classroom experience both to instructors and to students. Anyone who has ever taught a course on formal semantics will be familiar with the problem of drawing ever larger derivations, and changing them on the fly as the class goes on. A sentence of just ten words can easily fill an entire blackboard and take half an hour to draw (see Figure 4 below for an example). As for students, once they have left the classroom, they are often on their

[†]We would like to thank Patrick Blackburn, Chris Potts and the audiences of several demonstration sessions during the 2007 Stanford Linguistic Institute of the Linguistic Society of America (LSA) for helpful comments and suggestions. We are grateful to Yuval Masory for programming advice. We also thank the participants of the Spring and Summer 2007 introductory semantics classes at the University of Pennsylvania and at LSA respectively for their valuable feedback, as well as the participants of our usability tests. This work has been supported by a University of Pennsylvania SAS technology grant.

own with their homework exercises. In our experience, however, early feedback is crucial for student performance on lambda calculus and HK derivations.

For both these problems, the use of educational software suggested itself to us. Experiences with linguistic learning environments such as the *Trees* program (Kroch and Crist, 2002) in syntax courses at the University of Pennsylvania have been positive throughout (Anthony Kroch, p.c.) However, we were surprised to find that there does not seem to exist any educational software suited for our task. While some semantics- or logic-oriented educational programs exist (see Section 4), they are geared towards different (though related) applications, and not specific to the HK framework. Looking beyond natural language semantics, there appears to exist (apart from software geared towards students with a programming background) no training software for the more general field of the lambda calculus.

The present work describes our attempt at filling this much needed gap.

1.2 What It Does

The *Penn Lambda Calculator* is a multifunctional tool, designed for supporting both the instructor and the student in a variety of scenarios. The application is available as a "teacher edition" and a "student edition". The main difference is that the student edition is limited so that it does not provide automatic answers to the exercises the student is working on.

The following functionality is available in both editions of the application:

- **Interactive exercise solving.** Typically, the instructor will prepare exercises ahead of time in the form of a file, though the application also contains a graphical interface ("Scratch Pad") that allows users to input problem statements of their own. In each case, the program reads in the problem statement, internally generates a solution as applicable, displays the exercise and waits for student input. As the student progresses through the exercise, his or her answers are checked for correctness and the program gives appropriate feedback. The application currently supports the following kinds of exercises: type checking, reduction of lambda terms and bottom-up semantic derivations. Section 2 discusses them in further detail.

In addition, the teacher edition provides the following functions:

- **Visual presentation of semantic derivations.** This mode is intended to be used with a digital projector in class. The instructor provides the program with the tree and the lexical entries of the terminal nodes. The application then computes the denotations of nonterminal nodes automatically in a bottom-up fashion. At any point in time, the instructor can interrupt or rewind the derivation and/or modify any of the lexical entries involved. See section 3.1 for details.

- **Automatic scoring and grade management.** Students submit their completed work to their instructors electronically. Section 3.3 describes the tools the program provides to instructors to inspect and grade submitted work.

1.3 What We Aimed For

In the development of this program, we have adopted a few specific goals that go beyond best practices in software development.

- We view the textual feedback as a central component of the functionality of the application. Accordingly, we have made extended efforts to keep this feedback informative without constraining the range of admissible inputs more than absolutely necessary.
- The program has been designed so that it can be used by students with minimal outside instruction beyond the semantics that is needed to complete the exercises. Unlike related software (Barwise and Etchemendy, 1999; Larson et al., 1997), we do not presuppose that students read program documentation. We have performed extensive usability testing to ensure that the student interface is easy and intuitive to use for students at the introductory level of formal semantics with little background in computer usage.

1.4 How to Get It

The *Penn Lambda Calculator* is a stand-alone application available as a platform-independent Java Jar file, which is directly executable on Mac OS X and on most Unix systems. It is also available as a Microsoft Windows executable. All files are downloadable from the project's website. The student edition of the program is open source, licensed with the common GNU GPL license (Stallman, 2007), and the source code is linked from the website. In addition, the "engine" of the program, a fine-grained object-oriented model of simply typed lambda calculus expressions, is also downloadable as a separate library. The special edition for instructors is not provided on our website and is not open source, as this would make cheating very easy — see section 6.2. Instructors should contact the authors for a copy via the project website. The project website is `http://www.ling.upenn.edu/lambda`

2 Kinds of Exercises

As mentioned in the previous section, the application supports three kinds of exercises to be completed by the student. These three exercise kinds — type checking, reduction of lambda terms, and semantic derivations — are first presented by way of a walkthrough to the program. Later on we return to them in greater detail.

2.1 Walkthrough

This section is a detailed walkthrough that allows you to start working with the program and get an idea of its functionality as it presents itself to the student.

Here and in the following, we refer to version 1.0.5, the current version at the time of writing. We assume that you have the student edition available. If not, you can download the appropriate version from `http://www.ling.upenn.edu/lambda`, together with the sample exercise files on this website (right-click and save in most browsers). Even if you do not have access to the application, you can follow this section and refer to the figures to get an idea of how it works.

Double-click the program to start it and select "Interactive Exercise Solver". Click on File in the menu, then Open. Select the file `example1.txt` and click Open.

The first exercise is displayed as in Figure 1.[1] It consists of the term $\lambda x.[P(x) \wedge Q(x)]$, of which you are asked to enter the type, based on the typing conventions displayed in the lower left hand corner of the window. Specifically, here P and Q are one-place predicate constants, and x is a variable of type e. The correct answer is therefore $\langle e, t \rangle$. Type this in and press Return to confirm, and again to move on to the next exercise.

The program now goes to a second type of exercise: reduction of lambda terms. It displays the term $\lambda x.[P(x) \wedge Q(x)]\ (a)$, which you are asked to simplify by lambda conversion. Click Paste to copy the term into your answer box, then modify it, or start writing the reduced term from scratch if you prefer. To enter special characters like λ and $\wedge$, refer to the instructions in the middle left hand box. You can try various incorrect responses such as $[P(x) \wedge Q(x)]\ (a)$ to observe the program's responses.

When you are done (or bored) with the exercises in this file, open the next file `example2.txt` (see Figure 2). This third kind of exercise is very different. What you see is a syntax tree with some of the lexical entries already supplied. As explained in the instructions at the top of the main area of the window, your task consists in adding a lexical entry to the terminal α that is lacking one. As the text points out, the author of the exercise has used α to represent a reflexivizing morpheme.

To do this, click on that terminal α, then click into the text field below the tree and enter a lambda expression, conforming to the typing conventions displayed in the lower left hand corner. For example, to enter a variable of type $\langle e, \langle e, t \rangle \rangle$, use the letter R.

Confirm your choice of a lexical entry (the correct answer, in this case, is $\lambda R.\lambda x.R(x)(x)$) by hitting Return. It should now appear in the tree, under the terminal α. The tree is now ready to be semantically computed. Click on the VP node. You are now presented with a choice of three composition rules taken from

[1] These screenshots have been taken within the Mac OS X operating system. The corresponding windows look slightly different on other operating systems.

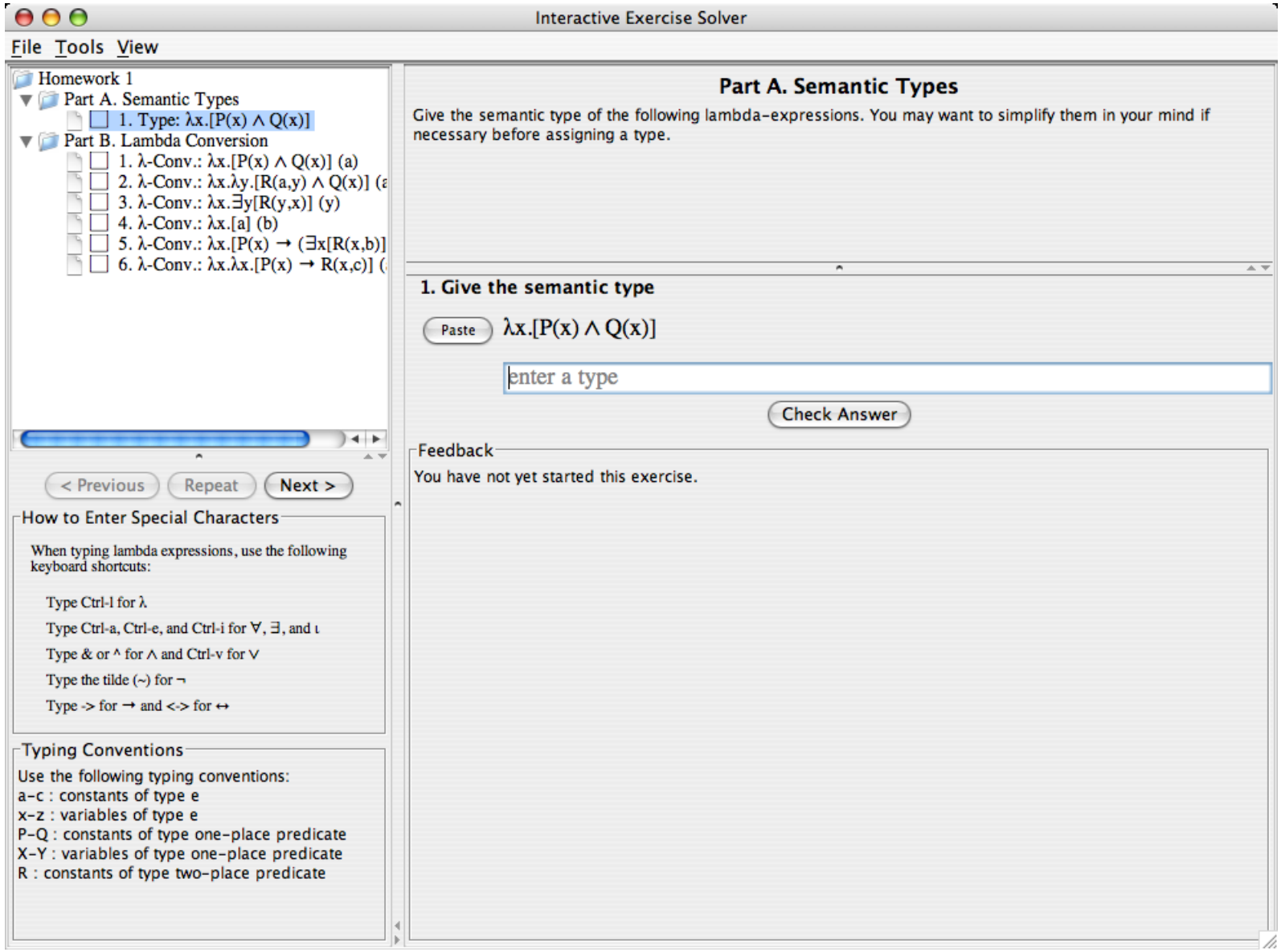


Figure 1: Type identification and lambda-conversion exercises.

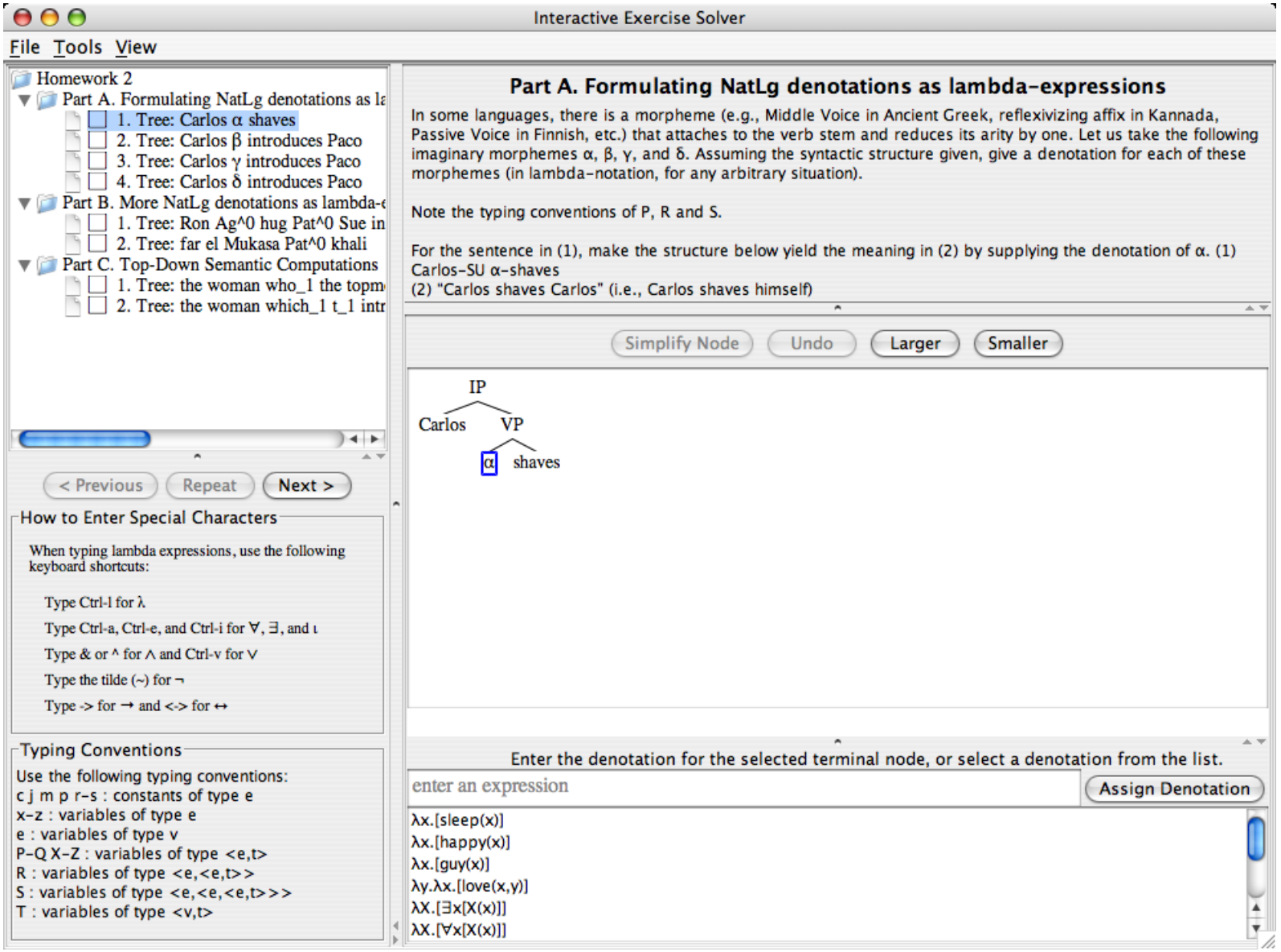


Figure 2: A natural language derivation exercise.

HK: function application, predicate modification, and lambda abstraction. Select the correct rule. The VP denotation will now change to the unreduced term

$$\lambda R.\lambda x.[R(x)(x)]\ (\lambda x.\lambda y.[shaves(y,x)]). \tag{1}$$

At this point, you are asked to reduce this lambda term. This corresponds to the second kind of exercise described above, and the program reacts to your input by feedback and error messages in exactly the same way as before. After three steps, this expression reduces to $\lambda x.shaves(x)(x)$, and you are asked to click on another node to continue. Click on the IP node and repeat the operation. You should end up with the formula $shaves(c,c)$ at the root. You are now free to go back and reassign lexical entries to terminal nodes or to select another exercise.

This completes our first overview of the *Penn Lambda Calculator* as it presents itself to the student. This walkthrough has not touched at all on several important features of the program, in particular the teacher-oriented functions. All of these will be described below. We begin by turning to a more systematic discussion of the kinds of exercises that the program supports.

2.2 Type Checking

The first kind of exercise, expected only to be used for a short time at the start of introductory semantics courses, asks the user to identify the semantic type (e, $\langle e,t\rangle$, $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$, etc.) of expressions. The instructor provides a list of expressions for the student. The instructor does not need to provide the program with the answers, i.e. the type of each expression — this is computed by the program automatically based on typing conventions for constants and variables (either default conventions or ones provided by the instructor).

Some example problems are:

Problem	Answer
$P(x) \wedge \forall y[Q(y)]$	t
$\lambda x.P(x) \wedge \forall y[R(x,y)]$	$\langle e,t\rangle$
$\lambda x.\lambda y.\lambda z.P(x)\ (c)$	$\langle e,\langle e,t\rangle\rangle$
$\lambda x.\lambda w.sleeps(x,w)$	$\langle e,\langle s,t\rangle\rangle$

When the user provides an answer, the program first checks that the answer is a syntactically well-formed description of a type. For instance, $\langle ett\rangle$ is not well-formed. While the program does accept two common shortcuts (both et and $\langle et\rangle$ are acceptable), it is otherwise fairly strict with respect to how to enter semantic types. User answers that could not be understood as types are returned with a hopefully helpful diagnosis as to the problem. In the case of $\langle ett\rangle$, for which the user probably meant $\langle e,\langle tt\rangle\rangle$ or $\langle\langle et\rangle,t\rangle$, the program suggests that the user add brackets.

2.3 Reduction of Lambda Terms

Reduction of lambda terms (or lambda conversion as called in the program, i.e. β-reduction together with α-conversion) is one of the primary kinds of exercises in the program. For these exercises, the user is presented with a lambda expression and is asked to simplify it by performing lambda conversions one at a time. The centerpiece of the program is its informative feedback provided to students when incorrect answers are provided, and this is explained below. Special keyboard shortcuts are available to enter logical symbols. As with the type checking exercises, the instructor provides the program ahead of time with the problem, a lambda expression, but the program will compute the answer and any intermediate steps automatically. Intermediate steps may be necessary both because of the presence of multiple lambdas in the expression and because of the need to create an alphabetical variant:

Problem	Expected Answer
$\lambda x.[P(x) \wedge \forall x[Q(x)]]\ (a)$	$P(a) \wedge \forall x[Q(x)]$
$\lambda x.\lambda y.R(x,y)\ (a)\ (b)$	Step 1: $\lambda y.R(a,y)\ (b)$ Step 2: $R(a,b)$
$\lambda x.\forall y[R(x,y)]\ (y)$	Step 1 (e.g.): $\lambda x.\forall y'[R(x,y')]\ (y)$ Step 2: $\forall y'[R(y,y')]$

Student inputs are first checked for whether they are syntactically well-formed lambda expressions. If they are not, feedback is provided as to the nature of the problem. For instance, the expression $\lambda.P(x)$ is returned with feedback indicating that a lambda must be followed by a variable. The expression $P(a) \wedge Q(a) \vee P(b)$ is returned indicating that the expression is ambiguous and requires parentheses. (Issues that arose in parsing and providing feedback for lambda expressions are described in section 6.1.)

If the student input has passed the test of syntactic well-formedness, it is then checked for well-typedness according to the typing conventions in place. For instance, assume that x is associated with type e and Q is associated with a type other than e. A user response of $\lambda x.P(x)\ (Q)$ will be returned to the user explaining that $\lambda x.P(x)$ denotes a function whose range is over expressions of type e, but it cannot be applied to Q because Q is of another type.

If the student input is well-typed but incorrect, the program checks it to see if the student fell into a number of common pitfalls. These pitfalls are captured by about a dozen abstract triggers applied to the answer roughly in order of decreasing specificity. They represent the most common student errors as observed in a decade of teaching introductory semantics courses.

If a known pitfall is encountered, appropriate feedback is provided. Whenever possible, we generate constructive hints which do not give away the answer but suggest to the student how to proceed with the reduction. If the student input is detected to be wrong but none of the triggers are activated, a generic error message informs the student about this limitation ("I'm afraid I can't help you here.").

The response that is displayed to the student is a collection of diagnoses and hints that may be produced by different triggers. Experience has shown that students confronted with a list of error messages tend to read only the first. For this reason, the diagnoses and hints are rendered as a single paragraph in an attempt to obscure their origin as separate entities. If more than one diagnosis is displayed, they are ordered with the most specific diagnosis first, and joined by discourse connectives such as "also".

Some of the possible feedback responses are given below, preceded by an incorrect attempt at lambda conversion that would activate the corresponding trigger.

PROBLEM: $\lambda x.P(x)\ (a)$
EXPECTED ANSWER: $P(a)$
STUDENT INPUT: $P(a)\ (a)$
PROGRAM RESPONSE: "After each λ-conversion, exactly one argument should be gone on the right hand side. Also, note that your expression also has a problem with types: $P(a)$ cannot be applied as a function to what looks like an argument to its right (a) because $P(a)$ is of type t according to the typing conventions in effect and therefore is not a function."

PROBLEM: $\lambda x.\lambda y.R(x,y)\ (a)\ (b)$
EXPECTED ANSWER: Step 1: $\lambda y.R(a,y)\ (b)$ Step 2: $R(a,b)$
STUDENT INPUT: $\lambda x.R(x,a)\ (b)$
PROGRAM RESPONSE: "When doing λ-conversion, start with the outermost λ. Remember, the leftmost λ-slot corresponds to the leftmost argument to be λ-converted."

PROBLEM: $\lambda x.\forall x[P(x)]\ (a)$
EXPECTED ANSWER: $\forall x[P(x)]$
STUDENT INPUT: $\forall x[P(a)]$
PROGRAM RESPONSE: "Remember to substitute the argument for all instances of the λ variable that are free in the body of the λ expression, and for nothing else. That is, make sure that you substituted into all and only the variable slots that were bound by the λ."

PROBLEM: $\lambda x.\forall y[R(x,y)]\ (y)$
EXPECTED ANSWER: Step 1: $\lambda x.\forall y'[R(x,y')]\ (y)$ Step 2: $\forall y'[R(y,y')]$
STUDENT INPUT: $\forall y[R(y,y)]$
PROGRAM RESPONSE: "Your answer changed the truth conditions of the expression because a free variable in the argument was accidentally bound during substitution. Go back and try to make an alphabetical variant."

PROBLEM: as in the previous example
EXPECTED ANSWER: as in the previous example
STUDENT INPUT: $\lambda x.\forall y[R(x,y)]\ (y')$

PROGRAM RESPONSE: "This is an incorrect alphabetical variant. Only bound variables can be rewritten as other variables while preserving truth conditions. Try making another alphabetical variant."

2.4 Semantic Derivations

Semantic derivations are another important part of our program. In this kind of exercise, a Logical Form syntax tree is presented to the user, who is expected to provide lexical entries for terminal nodes, choose the applicable composition rule at each nonterminal (function application, predicate modification, or lambda abstraction), and evaluate and simplify the nonterminal nodes in a bottom-up fashion. (Top-down evaluation is planned for future work.) The program displays the tree visually, with the user-provided denotations of each node displayed at each node in the tree. The user enters lexical entries and denotations at the bottom of the screen. A blue box shows which node is to be acted on next, and this box can be moved through the tree by clicking a node with the mouse (see Figure 2).

Lambda expressions are parsed and checked for well-typedness as described above for lambda conversion exercises. During the simplification of the denotation of a nonterminal node, the same lambda conversion pitfalls as in those exercises are detected and reported as feedback. Additionally, the choice of an incorrect composition rule, such as the choice of function application on two nodes typed $\langle e, t\rangle$ each, is reported.

Currently, instructors do not provide the correct solutions for lexical entry questions. The mistake of the student providing the wrong lexical entry for a terminal node is expected to be found by the user on his or her own once either 1) the user gets stuck at a nonterminal node that cannot be evaluated because, for instance, the types of the children do not allow for any composition rule, or 2) the tree is fully evaluated, but the student realizes the denotation arrived at for the root node is incorrect. In either case, the user can go back and revise the incorrect lexical entry, and then re-evaluate the affected part of the tree.

One common student error in providing lexical entries is the confusion of the Predicate Logic two-place predicate R, as in $R(x, y)$, and the predicate R denoting a Schönfinkelized (or Curried) function from individuals to functions from individuals to truth values, as in $R(x)(y)$. (Only the latter term is of type $\langle e, \langle e, t\rangle\rangle$.) For instance, the student may be required to provide a function from a predicate of the latter type to a truth value and may incorrectly submit $\lambda R.R(x, x)$. In this case, the application recognizes the type mismatch and gives the feedback "R is a function that takes (first) a single e-type argument alone, but you provided more than one argument. Rewrite your expression so that R is Schönfinkelized (i.e. each argument to R is surrounded by a separate pair of brackets)."

The following section describes the "teacher edition", which can be used for performing semantic derivations in class.

3 Instructor Tools

3.1 Class Presentation Mode for Tree Derivations

The "teacher edition" of the *Penn Lambda Calculator* enhances the bottom-up derivation exercises (see previous section) with on-screen buttons to evaluate nodes in the tree automatically, rather than requiring the user to enter the denotation of each node and simplify it manually. Moreover, the type of each node is displayed in addition to its denotation. This mode is designed for in-class presentations as an alternative to the instructor writing out each step on a blackboard. It can also be used by the instructor to debug exercises he or she is writing for the students. The program can step forward and backward through simplification steps:

$$[\![\text{VP}]\!]^g\ ([\![\text{Carlos}]\!]^g) \ \leftrightarrow\ \lambda x.shaves(x,x)\ (c) \ \leftrightarrow\ shaves(c,c) \tag{2}$$

and can fill in entire subtrees with their denotations in one step to move quickly through the derivation.

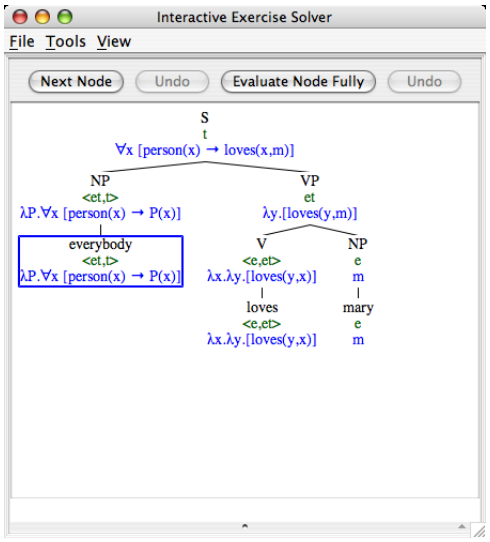
To prepare to use the presentation mode for tree derivations, the instructor creates a file containing the syntactic tree in labeled bracket notation, typing conventions for terms used in denotations, and any lexical entries that the instructor wants available ahead of time. (Additional lexical entries can be added while the program is running as well.) Because the program has the ability to simplify and combine lambda expressions, the instructor need not prepare the denotations of nonterminal nodes ahead of time. The appropriate composition rule at each step (e.g. function application versus predicate modification) is also chosen by the program based on the evaluated types of the daughter expressions, following the HK algorithm.

The program is able to represent the full range of phenomena covered in the HK textbook by function application, predicate modification, and lambda abstraction. This includes phenomena such as intersective adjectives, relative clauses and quantifier raising. As an example, derivations that illustrate different issues arising in connection with quantifiers are displayed in Figure 3. Figure 4 displays a complex noun phrase with two relative clauses of the kind that could easily take half an hour to draw on a blackboard. The simplification history of each node can be displayed in another box in the program (not shown in the figure).

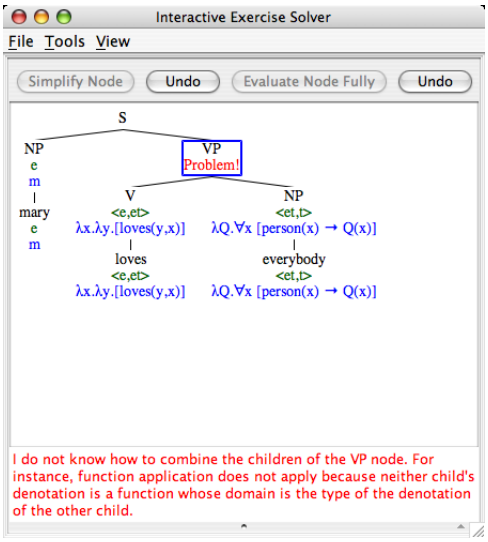
3.2 Creating Exercise Files

Exercises are provided by instructors to students in file form (e.g. via email or a webpage). Currently, exercise files are plain text files in which the instructor writes the title of the assignment, instructions, and each exercise one per line. Point values can be assigned to each problem in order to allow the program to compute grades automatically in the teacher review tool described in the next section. Plain text files can be created using any simple text editor (or word processor). The format of exercise files is documented on the website, which also provides samples.

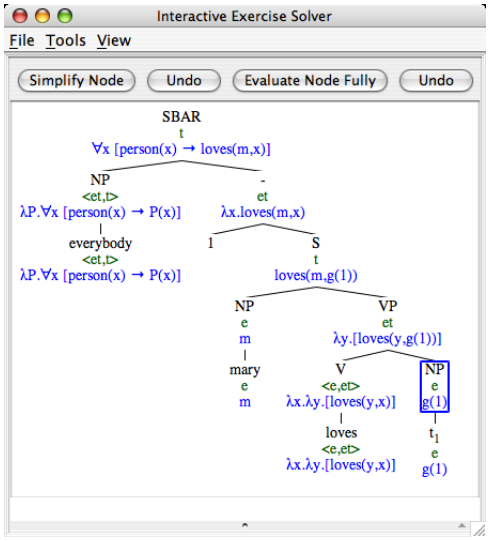
Although a plain text file format was chosen for exercise files for simplicity for the instructor, one drawback is that once sent to the students, the contents of these



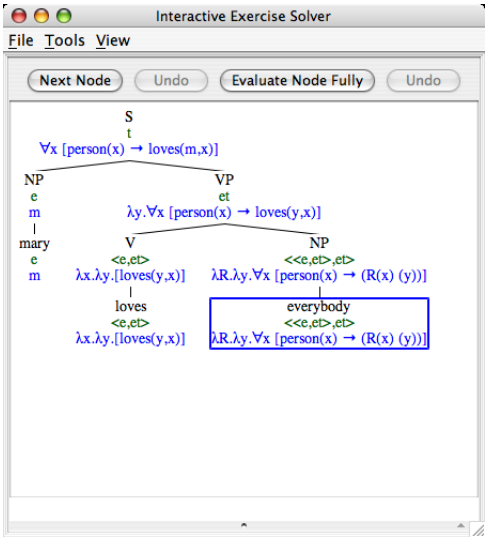
(a) A QNP that works in subject position . . .



(b) . . . does not work as an object unless . . .



(c) . . . you use quantifier raising or . . .



(d) . . . flexible types, i.e. another lexical entry.

Figure 3: Displaying various treatments of quantifiers using the teacher edition

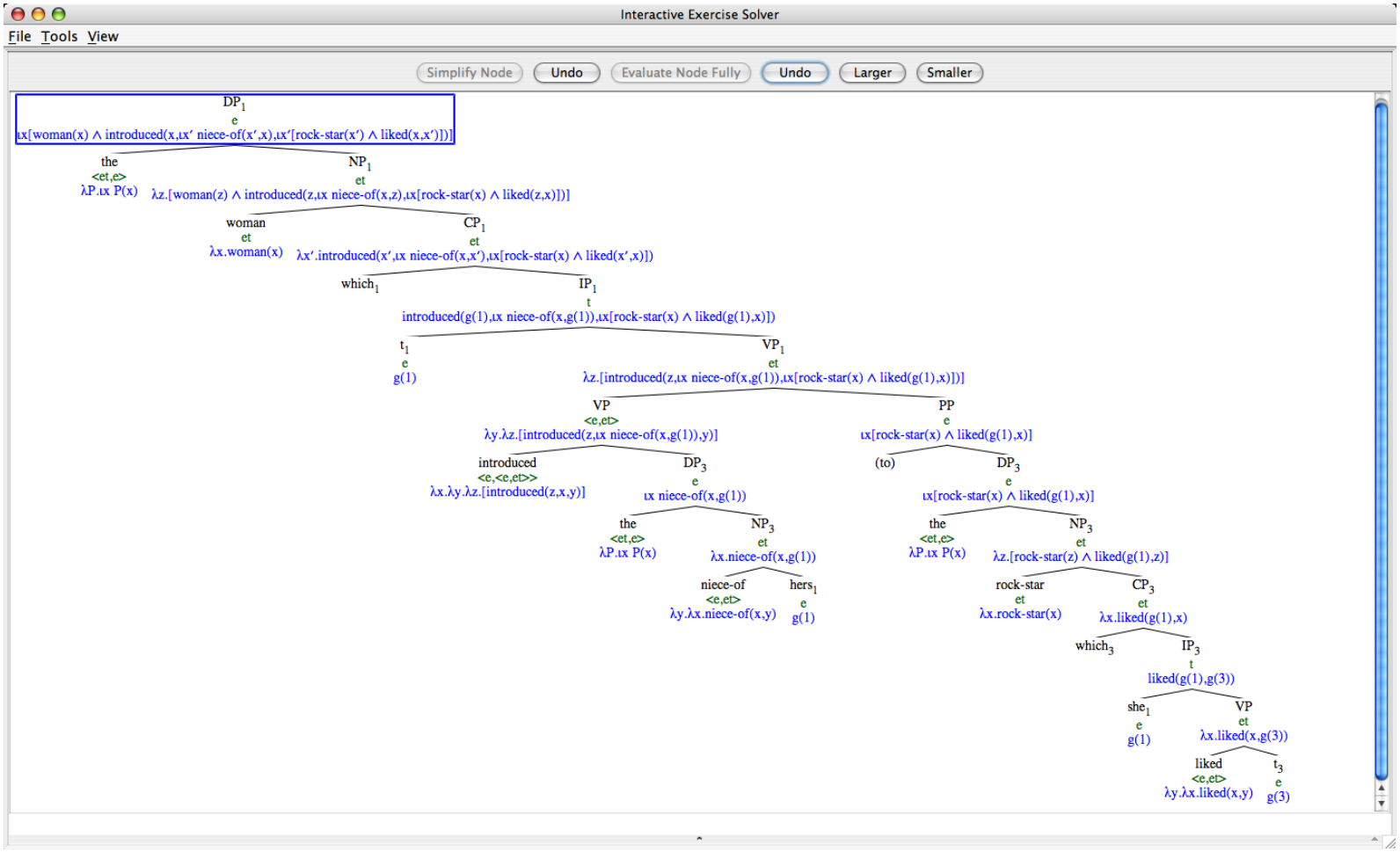


Figure 4: A completed derivation for the noun phrase *the woman who*$_1$ t_1 *introduced her*$_1$ *niece to the rock star which*$_3$ *she*$_1$ *liked* t_3

files can be viewed by the students as well. For this reason, the instructor must be careful *not* to put the answers or any other such information in the file.

3.3 Homework and Teacher Review Tool

Students using the application for homework assignments can submit their work to their instructor by saving their progress to a file, which can then be e-mailed to the instructor. When saving, the program asks the student for his or her name, which is written into the saved-work file, along with the student's answers to the questions. This makes it easy for instructors to keep track of students' performance.

As with any submitted homework, there is, of course, no guarantee that a saved-work file actually represents any particular student's efforts. It was not a goal of the project to anticipate all of the many ways one might cheat using the program. The exercise files sent to students by instructors are plain-text files, as explained in the previous section. However, saved-work files are in binary format to make it at least non-trivial for students to modify a saved-work file once it has been created by the program, such as to put a different student's name in the file.

Saved-work files received by the instructor can be reviewed using the application. The review component of the program, called "Teacher Tool", displays detailed information on the student's answers to each individual exercise. A score is computed for those exercises whose answer can be automatically checked for correctness (all but the bottom-up derivations where the student needs to define a new lexical entry) and for which the exercise file has specified a score value. The application can also collect the scoring information of all the students and present it in a table along with mean and standard deviation for the final scores. The tool shows each student's final response to each of the questions in the homework, as well as the percentage of students who answered each problem correctly, and it allows the instructor to enter comments into the saved-work file for his or her own reference later. This table of student scores can be copied and imported into other programs such as spreadsheet applications for further processing.

4 Related Work

We were not able to find any software that would work as a companion to the HK textbook the same way as ours. However, some applications exist that do resemble ours, be it because they are also written for the linguistics classroom or because they support formal natural language semantics as well. In this section, we review and compare some of them to the *Penn Lambda Calculator*.[2]

[2]Space prevents us from doing justice to a number of additional related programs, such as CURT (Clever Use of Reasoning Tools), a collection of tools for first-order inference and translation from natural language that accompanies a textbook (Blackburn and Bos, 2005); and CLEARS (Konrad et al., 1996), an "interactive graphical environment for computational semantics" that supports various semantic formalisms such as Discourse Representation Theory (DRT) and situation semantics.

4.1 Semantica

Semantica (Larson et al., 1997) is perhaps the closest relative of the *Penn Lambda Calculator*. Like our program, it is an interactive, graphics-oriented application designed for assisting the student in learning to use a truth-conditional semantic derivation system. The original release of *Semantica* ran on the now defunct operating system NeXTstep, but its authors have since then re-released it for Windows.

The most important difference between the two programs is a difference in the underlying semantic theories. The HK framework, on which our program is based and which it faithfully reproduces, stands in the tradition of *type-driven translation*. This concept, introduced by Klein and Sag (1985) and Jacobson (1982) (see also (Dowty, 2006, p. 10)), denotes a semantic translation system in which the types of the expressions on the daughters of a syntactic tree node determine which semantic composition rule applies at that node. This allows one to decouple semantic rules from the syntax and to have only very few semantic rules. A pithier term for this idea, which Klein and Sag credit to Emmon Bach, is *shake'n'bake semantics*.

By contrast, many semantic translation systems have taken the grammar to include a set of rule pairs consisting of a phrase structure rule and a semantic composition rule. The best known example of this style of system is likely to be classical Montague semantics (Montague, 1974). Klein and Sag contrast this idea, termed *rule-to-rule hypothesis* by Bach (1976), to type-driven translation.

This dichotomy is also at the core of the main difference between the *Penn Lambda Calculator* and *Semantica*. Only the latter allows (and requires) the user to specify a different semantic composition rule for each syntactic phrase structure rule.[3] In contrast, the *Penn Lambda Calculator* implements a system that is only equipped with a small collection of composition rules. Due to the type-driven nature of the HK computation system, these rules are sufficient to model a wide range of semantic phenomena in English.[4]

Both programs complement each other by offering important functions that the other one lacks:

- On the one hand, *Semantica* not only converts a syntactic tree to a logical formula, it also has the ability to evaluate that formula against a model, which consists of one or several worlds connected by modal and temporal relations. Each world is populated with individual objects of different kinds that stand in spatial relations to one another. The program contains an editor that allows the user to create and edit these models. This editor is quite easy to use. It is similar to and was modeled on the logic teaching program *Tarski's World* (see next section). *Semantica* can thus act as a simple theorem prover. The *Penn Lambda Calculator* is not able to do any of this.

[3] In practice, the *Semantica* user may load a file that contains a number of predefined rule-to-rule mappings of this kind.

[4] It is currently not possible for the user to add rules to this collection.

- On the other hand, *Semantica*'s emphasis on pedagogical issues and classroom management is not as strong. The program does not display the individual steps of the computation of a sentence's truth conditions, nor does it require the student to enter these steps. When the computation fails, only a generic error message is displayed that does not indicate the origin of the failure. ("Recheck rules and input tree.") Support for grading homework files in the style of our teacher tool is absent in *Semantica*. Perhaps for these reasons, using *Semantica* in the classroom has been reported to result in a "heavy initial burden" for the students and to require "considerably heavier time commitment than a traditional lecture-based course, both in terms of preparation and support" (Larson, 1997). Our experience with the *Penn Lambda Calculator* has been more encouraging (see Section 5).

Finally, a central difference is that *Semantica*'s underlying formalism does not make use of types nor of the lambda calculus, while the core functionality of the *Penn Lambda Calculator* consists in assisting students learning how to assign types to lambda terms and to reduce them.

4.2 Tarski's World

Tarski's World (released for Windows and Mac OS) is a pedagogical software program that helps students become fluent in first-order predicate logic. It displays logical formulae alongside graphical depictions of worlds (models) and asks the student to indicate whether any given formula is true in the world. Alternatively, the student could also be directed to build a world from scratch that makes a formula or collection of formulae true. Unlike *Semantica*, this program does not allow for models of modal or temporal logics, i.e. models in which several possible worlds are connected to each other by modal or temporal accessibility relations.

Tarski's World is similar to the *Penn Lambda Calculator* in that it focuses on providing helpful feedback to the student and on classroom management functions. It provides automatic grading via a central server, the *Grade Grinder*, to which students can electronically submit their files. However, this is where the similarities end: *Tarski's World* does not touch on natural language syntax or semantics.

4.3 Nessie

To conclude this section, we mention the *Nessie* project (Blackburn and Hinderer, 2007) as a recent example of an application created in the context of natural language formal semantics. Unlike the other programs presented here, *Nessie*'s approach is not pedagogical, and it is neither graphics-based nor interactive. The novelty of this project consists in its attempt at providing a generic framework for large-scale natural language semantic computation, based on the TY_n family of logics, which has been suggested as a uniform framework for virtually any kind of semantic analysis (Muskens, 1996). TY_n is based on the simply typed lambda

calculus and is therefore very similar to the logic underlying HK and our system. Furthermore, TY_n provides flexible support for any number of basic kinds of entities such as ordinary individuals, belief states, times, and situations. *Nessie*, a platform-independent application, fully implements TY_n and is developed with the aim of providing "a systematic way of combining the insights from many different approaches, ranging from DRT through situation semantics and classical possible world semantics, to event based semantics" (Blackburn and Hinderer, 2007, p. 5).

5 Field Experience

An early version of this program has been field-tested in the Spring 2007 graduate student introductory course to formal semantics at the University of Pennsylvania and has later undergone extensive usability testing in order to improve its user interface. In its current form (the result of about 400 man-hours of work), it has been deployed for the first time in the introductory course to semantics at the Linguistic Society of America Summer Institute 2007, at Stanford. Both courses have been taught by one of us (Romero). We have offered an internet forum in order to collect feedback from the students and to provide technical support. We expected to have to make changes to the program and to redeploy it several times as the course proceeded, but this turned out not to be necessary. Students used the same version of the application throughout the course. The forum was used primarily to clarify questions in the exercises rather than to ask questions about the program itself. Numerous minor improvements to the application were suggested and bugs were collected. As a result, we expect its basic design to remain stable in the near future.

The teacher edition's ability to demonstrate a derivation on the screen turned out extremely helpful in the classroom. Even if one does it slowly enough so that the students have time to assimilate what is on the screen, it looks cleaner and saves time compared to writing the same derivation on the blackboard. A derivation that used to take us 30 minutes on the blackboard takes about 5-10 minutes using the application, depending on how much explanation is needed.

6 Issues in Program Development

6.1 Robust Parsing of a Formal Language

The syntax of the lambda calculus is usually given as a collection of CFG or BNF rules or as a recursive definition to that effect, together with the statement that when the formulae are presented to a human reader, parentheses can be dropped for convenience. To parse typed lambda calculus expressions entered by students and teachers, we needed to implement a "robust" syntax, able to handle these omitted parentheses and similar pitfalls. (We soon discovered that it was not advisable to force users to disambiguate every formula with parentheses, since this soon led to frustration, and it distracted users from the task at hand.)

Informally, parentheses may be dropped just in case the resulting expression appears unambiguous to the human reader. The exact conditions for this, as well as the rules that disambiguate these expressions, turned out surprisingly difficult to determine. Even our own experience with the typed lambda calculus did not allow us to define the rules we seemed to have unconsciously mastered, and so we had to discover them empirically.[5] We discuss a few examples here.

The most striking phenomenon was the significance of spaces in expressions of function application. For instance, when the type of M was not specified, the expression $\lambda x.M(x)\ \ (a)$ was most likely to be interpreted as intending $\lambda x.M(x)$ to be applied to the argument a. However, the expression $\lambda x.M(x)(a)$ was understood as having a as the second argument of M itself (where M is now understood as a Schönfinkelized two-argument function). The difference is one of scope, with (a) in the first case having wide scope relative to the lambda, and in the second case narrow scope. Apparently, though, these structural preferences can be overridden: in our experience, most people would be reluctant to interpret the first x to be bound and the second x to be free in the expression $\lambda x.M(x)\ \ (x)$, regardless of the presence of space.

In some cases, we are able to use the fact that parentheses are regularly omitted because there is only one well-typed bracketing. $\lambda x.T(x)(a) \wedge U(b)$ is an example of this. Without knowing the types of T and U, the program will reject this expression on the grounds that it is ambiguous. However, if T is known to be of type $\langle e, t\rangle$, the program will give the user the benefit of the doubt and understand the expression as $[(\lambda x.T(x))\ (a)] \wedge U(b)$. If T is instead entered as $\langle e, \langle e, t\rangle\rangle$, the expression will be treated as $\lambda x.(T(x)(a) \wedge U(b))$.

6.2 Issues in Distribution

In designing this application, we made the decision early on to make as much functionality available for free under a GPL-like license (Stallman, 2007), including the source code. At the same time, some of the functionality cannot be distributed to students. In this section, we discuss some issues that arise from this conflict. At the time of writing, these are open and serious problems for us, and we are grateful for any suggestions.

As mentioned above, we are currently only offering the student edition of the application on the project website. The reason for this is that the teacher edition has capabilities that would easily allow students to solve any exercise with almost no effort at all. Therefore we feel its distribution must be restricted. We are currently exploring different ways to manage this restriction:

- One option we are considering is making the teacher edition available as a restricted download. Only individuals who can document to us their affiliation as university faculty would be given access to the program. While this will create a certain delay in distribution, we anticipate that the added work for us

[5] The parallel to natural language syntax has not escaped our attention.

will be manageable and the delay short. However, this has the problem that a single instructor who, for whatever reason, makes the teacher edition available to some student might result in both versions being effectively freely available to all students.

- Another option, which avoids this problem, would consist in making the teacher edition available only as a password-protected web-based application. However, this would require that the instructor have Internet access during class sessions.

- A related issue concerns the extent to which we can release the source code. While we would like the full program to be available to be modified and reused by others, providing the full source code would allow others to compile and make the teacher edition available to students. We prefer to err on the side of caution and are currently making the source code available only partially. We will be happy to release the full source code to those that we would provide the teacher edition to.

7 Future Work

The *Penn Lambda Calculator* is usable in its current state; however, improvements are planned in several areas. Lambda expressions understood by the program will continue to be extended and refined to accommodate nonstandard ways of entering lambda expressions and to address pedagogical concerns. We will allow the program to accept expressions containing mathematical, set, and modal operators not yet considered, and situation variables as superscripts on interpretation functions. The set of semantic computation rules, which is hard-coded into the program at the moment, could be made user-extensible. We also plan to add support for top-down HK derivations.

A drawback of the rigid distinction between "teacher" and "student" editions of the application is that it is impossible for the instructor to allow the students to step through derivations at their own pace, unless he or she wants to give students access to the "teacher" edition. Currently, students can only watch the derivations as the instructor steps through them in class. If they try to replicate them in the "student" edition, they have to re-enter by hand all the lambda conversions involved in the derivation. This problem has emerged in the classroom and was not foreseen by us. We plan to address it by providing the instructor with a means to selectively unlock the student edition's features for certain derivations only.

Finally, we intend to improve the integration of our program with related software. In particular, we plan to add the ability to exchange syntactic trees between the *Penn Lambda Calculator* and the *Trees* program, a learning environment for syntactic theory (Kroch and Crist, 2002), as well as the LaTeX tree-drawing package *qtree*. We may also link up the program with *Tarski's World* and/or *Semantica*

(see section 4) in order to provide students with a way to check the truth of their sentences in a self-constructed model.

References

Bach, Emmon. 1976. An extension of classical transformational grammar. In *Problems in Linguistic Metatheory, Proceedings of the 1976 Conference at Michigan State University*.

Barwise, Jon and Etchemendy, John. 1999. *Language, Proof, and Logic*. CSLI Publications, Stanford.

Blackburn, Patrick and Bos, Johan. 2005. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications, Stanford.

Blackburn, Patrick and Hinderer, Sébastien. 2007. From TY_n to DRT: an implementation. In *Proceedings of the 3rd International Language & Technology Conference (L&TC'07)*, pages 384–388.

Dowty, David R. 2006. Compositionality as an empirical problem, unpublished manuscript, online 2007-10-14: http://www.ling.ohio-state.edu/∼dowty/context-free-semantics.pdf.

Heim, Irene and Kratzer, Angelika. 1998. *Semantics in Generative Grammar*. Oxford: Blackwell.

Jacobson, Pauline. 1982. Visser Revisited. In Kevin Tuite, Robinson Schneider and Robert Chametsky (eds.), *Papers from the 18th Regional Meeting, Chicago Linguistic Society, April 15-16, 1982*, volume 18, pages 218–243.

Klein, Ewan and Sag, Ivan A. 1985. Type-driven translation. *Linguistics and Philosophy* 8(2), 163–201.

Konrad, Karsten, Maier, Holger and Pinkal, Manfred. 1996. CLEARS - an education and research tool for computational semantics. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING), Copenhagen*.

Kroch, Anthony and Crist, Sean. 2002. Trees 3 for Windows. Pedagogical software. Online 2007-10-14: http://www.ling.upenn.edu/∼kroch/Trees.html.

Larson, Richard K. 1997. The Grammar as Science project. Day 2 (Semantica). Presentation at the Linguistic Society of America Summer Institute, Workshop on Linguistics and the Language Sciences: New Computer-based Methods and Materials for Undergraduate Education, Cornell University, Ithaca, NY. Online 2007-10-08: http://semlab5.sbs.sunysb.edu/∼rlarson/day2.pdf.

Larson, Richard K., Warren, D.S., de Lima e Silva, J. Freire, Gomez, P. and Sagonas, K. 1997. *Semantica*. MIT Press, Cambridge.

Montague, Richard. 1974. *Formal Philosophy: Selected Papers of Richard Montague. Edited and with an introduction by R. H. Thomason*. Yale University Press, New Haven.

Muskens, Reinhard. 1996. *Meaning and Partiality*. Studies in Logic, Language and Information, CSLI Publications, Stanford.

Stallman, Richard. 2007. GNU General Public License.

Regression Testing For Grammar-Based Systems

Nikos Chatzichrisafis[1], Dick Crouch[3], Tracy Holloway King[2],
Rowan Nairn[2], Manny Rayner[1,3], Marianne Santaholma[1]

(1) University of Geneva, TIM/ISSCO
(2) Palo Alto Research Center
(3) Powerset, Inc.

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

In complex grammars, even small changes may have an unforeseeable impact on overall system performance. As grammar based systems are increasingly deployed for industrial and other large-scale applications, it is imperative to have systematic regression testing in place. Systematic testing in grammar-based systems serves several purposes. It helps developers to track progress, and to recognize and correct shortcomings in linguistics rules sets. It is also an essential tool for assessing overall system status in terms of task and runtime performance.

This paper describes best practices in two closely related regression testing frameworks used in grammar-based systems: MedSLT, a spoken language translation system based on the Regulus platform, and a search and question answering system based on PARC's XLE syntax-semantics parser.

1 Introduction

Regression testing is an important part of all software development, including large-scale, application-oriented grammars. Similar to regression testing in other systems, regression testing in the context of grammar development ascertains the correctness of the code and its output alongside important systems issues such as how quickly the grammar and system run. This helps to ensure systematic development both of the grammar and the system using the grammar.

In the context of grammar engineering, this applies in particular to linguistic rule-sets and the compilers and interpreters used to process them. Systematic testing identifies problems so that they can be fixed before they affect system performance. Furthermore, fixing such problems also requires accurate information about system coverage over time, on a per-component level, so that grammar writers can effectively track down and correct any loss of coverage, as well as identify areas for further development.

Here we focus on two closely related regression systems used in grammar-based systems: one developed at Geneva University for a multi-lingual medical spoken translation system based on the Regulus platform (Rayner et al., 2006), and one for a search and question-answering system that uses the XLE LFG parser and ordered-rewriting system (Crouch et al., 2007). Given the usefulness of these tools for two disparate systems at whose cores are heavily engineered deep grammars, we hope that the techniques described here will be useful to the grammar engineering community regardless of framework, especially since these grammars are increasingly used as central system components. Although many aspects of the regression testing we describe here can be attributed to common sense, the paper pulls together lessons learned in the development and active use of the systems. This includes the fact that ease of use and a well-designed user interface are of great help, even for experienced system developers.

Regression testing for grammar-based systems involves two phases. The first includes systematic testing of the grammar during its 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 grammar 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 test suites so that the developers do not need to change the gold standard annotation every time an intermediate level of representation changes.

In the following we first describe in more detail how the regression tests are used in grammar development (Section 2), and then how they are used for the grammars within a larger system and application environment (Section 3), drawing examples from two systems using complex grammars.

2 Regression Testing During Grammar Development

Most large-scale grammar development efforts use regression testing for the output of their grammars. Many complex systems have separate tests for different levels of output, e.g. syntactic and semantic. The test suites are often defined by the grammar writer, perhaps in conjunction with some corpus, and they aim for coverage of (specific) linguistic and lexical data (see Lehmann et al. (1996) on test suite design for NLP). Here we do not explore the development of these test suites, including tools to assist in the creation of appropriate "gold" standard analyses to compare against (Oepen et al., 1998; Rosén et al., 2005). Instead, we focus on the regression system a grammar developer would need to maximally benefit from the test suites that they have.

The basic idea is extremely simple. Having constructed the regression corpora with annotations as to what the correct outputs are, the system must run the grammar against each test suite and notify the developers how things have changed. The details of how the regression testing system is run and how results are presented are crucial. If the regression testing tools are badly designed, the testing will be time consuming and laborious to the point that it is not performed on a regular basis. If the details are right, it makes a huge difference in ease of grammar development and in overall application maintenance. As we will see, these feature requirements are similar to those of regression testing of the system as a whole.

In addition to the ability to run the tests manually after each change, the regression system should run automatically on a regular schedule, generally overnight so that the results are ready for the developer to review each morning. These results need to be posted in such a way that they can easily be perused by the grammar writer. For example, a result summary might be mailed to the developer, along with a link to a web page with detailed results of the regression run.

A second important feature is a method to compare the results of different test runs. Most regression sets do not yield perfect results, so the grammar developer needs some way to determine whether the incorrect results indicate something that

was recently broken, as opposed, for example, to something that has not been implemented yet. Furthermore, when an example gives a wrong result, it is necessary for the regression system to report if the grammar ever gave the right result, what it was, and when that result was last produced. Hence it is necessary to have tools that enable the grammar engineer to compare the latest results against the immediately previous ones, as well as against the best ever result and the gold standard result for a given example. Examples of displays are shown in the next section.

3 Regression Testing the Grammar in the System

As anyone who has worked as a grammar engineer is painfully aware, linguistic formats change over time. Sometimes these are minor changes, such as the systematic renaming of features, while other changes may involve significant linguistic reanalysis. Annotation schemes that rely on internal representations (parse trees, semantic forms, etc.) run into the problem that the annotations themselves degrade as the representations change. As such, it is not easy to know whether mismatches occurring during regression runs reflect a real problem or just a case of a change in internal representation.[1] For this reason, it is good to have test suites whose annotations do not refer to internal forms; these suites can supplement the more traditional grammar-based test suites discussed briefly in the previous section.

For example, in a dialog system, the annotation can state whether or not Y is a good response to X in a given context. This scheme has been implemented in the Clarissa dialog manager (Rayner and Hockey, 2004). In the Clarissa system, regression testing is performed by performing dialog moves on input states, producing a specific output state and a set of required actions. A context-independent test suite can be built by recording the desired output state and accompanying actions for corresponding input states. In a translation system, annotations can state whether or not a Y is a good translation of X, instead of testing the syntactic representations used internally to perform the translation. In a question answering system, they can state whether passage X should match query Y. In most applications, there can be multiple correct responses, e.g. an MT system can produce multiple good translations. As such, the regression testing must allow for this possibility (see section 3.1.2). These considerations lead us to the conclusion that it is often easiest to perform regression testing of a grammar in the context of a larger system that uses the grammar to perform some concrete task (Spark-Jones and Galliers, 1996).

In this section we describe the Regulus-based medical speech translation system MedSLT and the XLE-based search and question answering systems.

[1] Several annotation systems focus on this problem, providing tools, such as discriminant features, that can be used to quickly bootstrap a new regression suite off of a previous version (Oepen et al., 2002; Rosén et al., 2005).

3.1 Regression Testing Regulus Grammars in MedSLT

3.1.1 The Regulus Toolkit

Regulus is a comprehensive Open Source toolkit for developing grammar-based speech-enabled systems that can be run on the commercially available Nuance speech recognition environment. The platform has been developed by an Open Source consortium, whose main partners have been NASA Ames Research Center (ARC) and Geneva University, and is freely available for download from the project's SourceForge website. The platform supplies a framework which supports development of grammars, compilation of grammars into parsers, generators and recognisers, and use of these compiled resources to build speech translation and spoken dialog applications. It has been used to build several large systems, including NASA's Clarissa procedure browser, and is described at length in Rayner et al. (2006).

The Regulus development model is based on reusable grammars. Developers use general domain-independent unification grammars, which are tailored to the domain at hand by *grammar specialization* using explanation-based learning (EBL). Grammar specialization is conducted using a small training corpus, domain specific lexica, and a set of instructions ("operationality criteria") describing how to perform the specialization (see Figure 1).

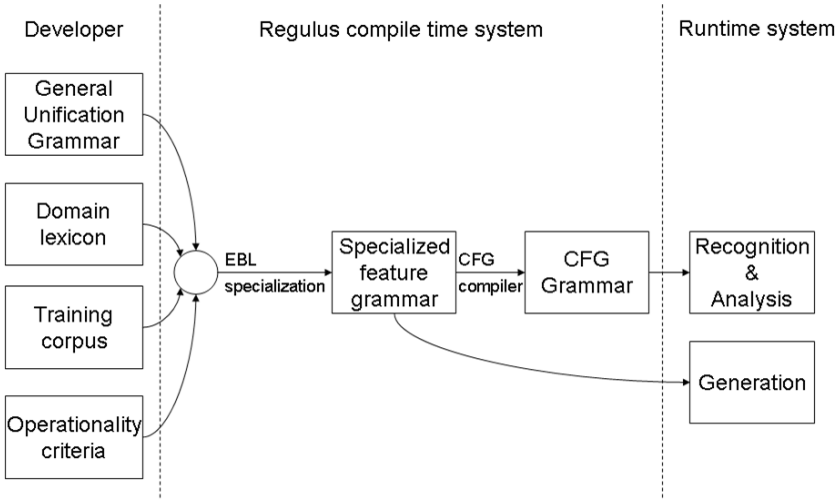


Figure 1: The Regulus processing path

3.1.2 The MedSLT System

Geneva University's MedSLT is a large Regulus-based project, which focuses on automatic interactive translation of spoken doctor/patient examination dialogs (for a screenshot, see Figure 2).

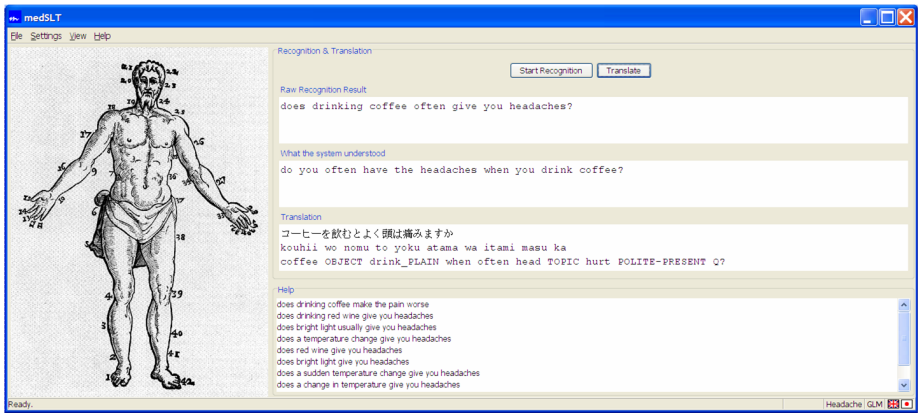


Figure 2: MedSLT application window

As of 2007, there are versions of the system for approximately twenty different language pairs and four subdomains. The subdomains covered are headaches, chest pain, abdominal pain and pharyngitis. Languages handled include English, French, Japanese, Spanish, Catalan and Arabic. Vocabulary size varies depending on the input language, being determined mainly by the number of inflected forms of verbs and adjectives. It ranges from $\sim$400 for Japanese to $\sim$1100 for French. The overall system is stable, and has been tested on medical students in simulated patient-examination situations with good results (Chatzichrisafis et al., 2006).

Typically the Regulus grammar for each MedSLT language is used for several system components, including speech recognition, analysis, and generation of translated sentences. Maintenance and active extension of these different components for the many different language-pair and domain combinations led the MedSLT developers to build an elaborate set of regression testing tools. Each language and domain have regression corpora in text form. Additionally each language has recorded speech corpora for selected domains. The regression testing produces results at four different levels: individual utterances, individual corpus runs, sets of runs for a language, and the complete set.

During regression testing, each utterance is passed through all stages of processing in order to determine how the system as a whole is performing. For MedSLT, a spoken dialog translation system, speech is an important aspect, and is thus thoroughly tested using offline speech recognition with the Nuance `batchrec` utility. Automated tests provide the developers objective runtime performance figures, and enable the team to tune grammars and recognition parameters to match the target platform. Speech recognition regression tests report for each test suite semantic error rates, word error rates, and run-time performance of the test suite in terms of CPU time.

The next processing steps cover source language analysis, ellipsis resolution, translation to interlingua, translation from interlingua, and target language genera-

```
Wavfile: c:/corpora/2004-11-01/USEnglish/13:34:41/utt19.wav
Source: does your pain appear in the morning
                +avez-vous mal des deux cotes
Recognised: does your pain appear in the morning
Target: la douleur survient-elle le matin
*** PREVIOUS OK: "avez-vous mal le matin" ***
 Source rep:   [[possessive,[[[pronoun,you]]]],
                [prep,in_time], [secondary_symptom,pain],
                [state,appear], [tense,present],
                [time,morning], [utterance_type,ynq],
                [voice,active]]
 Resolved rep: [[prep,in_time], [symptom,pain],
                [pronoun,you], [state,have_symptom],
                [tense,present], [time,morning],
                [utterance_type,ynq], [voice,active]]
   Resolution: trivial
  Interlingua: [[prep,in_time], [pronoun,you],
                [state,have_symptom], [symptom,pain],
                [tense,present], [time,morning],
                [utterance_type,ynq], [voice,active]]
   Target rep: [[state,survenir], [symptom,douleur],
                [temporal,matin], [tense,present],
                [utterance_type,sentence], [voice,active]]
     Judgment: ?
```

Figure 3: Result record for individual utterance (slightly simplified). The lines show, in order: the name of the recorded speech file; a transcription of the source utterance with preceding context; the recognised result; the translation; a previously produced correct translation; various internal representations; and the quality judgment (currently unknown '?').

tion. Performance on these steps is summarized by showing the quality of the translated corpus and the number of sentences that did not produce a result. To summarize translation quality of the corpus, translations are judged as being Good, OK (acceptable but not perfect) and Bad.[2] A database of all previous results is stored, so that Bad results can show when the example most recently produced a correct result and what that correct result was. To help keep judgments up to date, the regression testing scripts rebuild the judgment databases, and any sentences without a matching judgment are flagged with '?'. Developers are able to click on the corresponding corpus and start annotating these flagged sentence pairs as Good, OK or Bad.

[2]Judgments are based on both syntactic and semantic criteria. Sentences with good syntax and semantics are judged as Good. In case the semantics is correct, but the syntax could be improved, the sentence would be categorized as OK, otherwise as Bad.

A typical result record for an utterance is shown in Figure 3. In this particular run, a new translation was produced (*la douleur survient-elle le matin*) presumably due to a change in the translation rules, and no stored judgment was available. The system flags this ('?') and shows the user a known OK translation previously produced (*avez-vous mal le matin*).

3.1.3 How the Regression Testing Tools Drive the Debugging Process

This section presents an illustrative example, showing how the regression testing tools are used in the normal MedSLT development cycle. The grammar engineer starts by examining the top-level webpage for the nightly corpus run. This displays a formatted table, with one line for each corpus; tables for individual languages are grouped on different tabs. Figure 4 shows the set of tables for Japanese input. Each line summarizes the judged quality of translations (Good, OK, Bad), how many translation have not been judged yet (Unknown), and for how many sentences no translation is produced (NoResult). Furthermore, the columns NewBad, NewUnknown, and NewNoResult show how the results have changed from the previous run. The number of processed sentences (Processed) and time used (Time) are printed in the last two columns.

Text-to-Text Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Processed	Time
English	Abdominal Pain	65.7%	23.9%	1.3%	3.8%	5.3%	0.2%	2.5%	2.0%	603	21:04.5
English	Chest Pain	76.9%	13.0%	1.3%	2.8%	6.0%	0.2%	1.6%	1.9%	637	21:49.9
English	Headache	79.9%	11.2%	1.0%	3.8%	4.0%	0.3%	2.1%	1.5%	793	32:30.9
French	Headache	40.6%	0.9%	0.0%	38.2%	20.3%	0.0%	5.4%	4.4%	793	14:23.3
Japanese	Abdominal Pain	23.4%	1.2%	1.0%	65.8%	8.6%	0.0%	6.8%	1.0%	603	13:40.5
Japanese	Chest Pain	22.8%	1.1%	0.8%	61.1%	14.3%	0.0%	6.4%	1.3%	637	13:40.6
Japanese	Headache	33.3%	1.0%	1.0%	59.5%	5.2%	0.0%	16.1%	1.4%	793	23:57.9
Spanish	Headache	26.7%	1.5%	1.4%	30.6%	39.7%	0.0%	15.0%	11.6%	793	24:21.2

text-to-text results for Japanese

Text-to-Text Ellipsis Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Processed	Time
English	Headache	7.0%	1.1%	0.5%	27.4%	64.0%	0.0%	1.6%	0.5%	186	6:10.3
French	Headache	5.9%	0.5%	0.0%	30.1%	63.4%	0.0%	1.1%	0.5%	186	3:10.9
Spanish	Headache	5.4%	0.0%	0.0%	28.5%	66.1%	0.0%	0.5%	0.5%	186	4:05.3

text-to-text ellipsis results for Japanese

Speech-to-Text Runs

Target Language	Domain	Good	OK	Bad	Unknown	No Result	New Bad	New Unknown	New No Result	Misrecognised	WER	SER	Total Words	Total Utts.	xRT
English	Headache	68.1%	2.2%	3.4%	17.6%	5.6%	0.6%	12.4%	2.5%	3.1%	2.92%	11.46%	2122	323	0.385

speech-to-text results for Japanese

Figure 4: Part of display summarising the results of a nightly MedSLT test run. Ellipsis processing for Japanese is not yet well developed; speech recognition is in contrast very good.

Looking at the tables it is immediately apparent that performance on Japanese to Spanish has slipped badly (11.6% in "New No Result"). On the other hand, although performance on the ellipsis corpus is not good, little appears to have broken; the poor result is due to the fact that most of the Japanese ellipsis processing rules

have not yet been implemented. The developer's next step is thus to examine the detailed trace for the main Japanese to Spanish corpus.

Having clicked to get the corpus trace, the developer now searches down the file for occurrences of the string "PREVIOUS". As mentioned above, examples which used to work and now give different results are flagged in this way. In the present example, a minute or so of searching is enough to show that most or all examples involving the common Japanese word *tsuzuku* 'continue/last' are failing to produce translations; a target language representation is created, but surface generation fails, as in the example shown in Figure 5. This is a detailed enough analysis to permit direct debugging of the problem in the Regulus development environment.

```
Source: issyuukan ijou tsuzuki masu ka
Target: generation_failed
*** NO TRANSLATION ***
*** PREVIOUS OK: "el dolor dura mas de una semana" ***
    Source rep: [[number,1], [numerical_comparative,ijou],
                 [state,tsuzuku], [tense,present],
                 [unit,syuukan], [utterance_type,sentence]]
  Resolved rep: [[utterance_type,sentence], [number,1],
                 [unit,syuukan], [numerical_comparative,
                 ijou], [tense,present], [state,tsuzuku]]
    Resolution: trivial
   Interlingua: [[prep,duration], [spec,[more_than,1]],
                 [state,last], [symptom,pain],
                 [tense,present], [timeunit,week],
                 [utterance_type,ynq], [voice,active]]
    Target rep: [[comparative,mas_de], [number,1],
                 [state,durar], [symptom,dolor],
                 [tense,present], [timeunit,semana],
                 [utterance_type,sentence], [voice,active]]
      Judgment: error
```

Figure 5: Result record for an individual utterance (slightly simplified) showing an example of a Japanese to Spanish translation problem. The input sentence could be glossed as "One-week more continue POLITE Q" ("Has [the pain] continued for more than one week?"). The meanings of the other fields are described in the caption to Figure 3.

In general, it is extremely useful to be able to descend rapidly in this way from the top-level display, which shows figures for over twenty different language pairs, to the low-level task of debugging a single representative test suite example. The point to note here is that good organization of the trace information has allowed the language engineer to be fairly sure, after only a few minutes, that this is indeed one of the most important outstanding problems. Having a reliable testing framework of this kind makes it possible to focus developer effort effectively, and facilitates overall project management.

3.2 Regression Testing in the XLE QA System

The goal of the XLE search and question answering (QA) system (Bobrow et al., 2007; Crouch and King, 2006) is to find matches between passages and queries, where a "match" is defined, depending on the application, as anything from strict entailment (strict QA) to relevancy (search). The system batch processes texts, mapping them into deep semantic representations, and stores these in a semantically-indexed database. For retrieval or question answering, the query is mapped through a related set of rules, and the query representation is used to retrieve relevant passages from the index. These are then ranked as to relevance (search) or run through a set of entailment and contradiction detection rules (question answering).

In order to run regression testing on this system, passage and query pairs with answers are created and run through the syntax to the semantic representations. A light inference procedure is applied to detect entailments or contradictions between the final representations of passages and queries. The resulting answer is compared against the gold standard answer.[3] The basic system architecture as used by the regression testing system is shown in Figure 6.

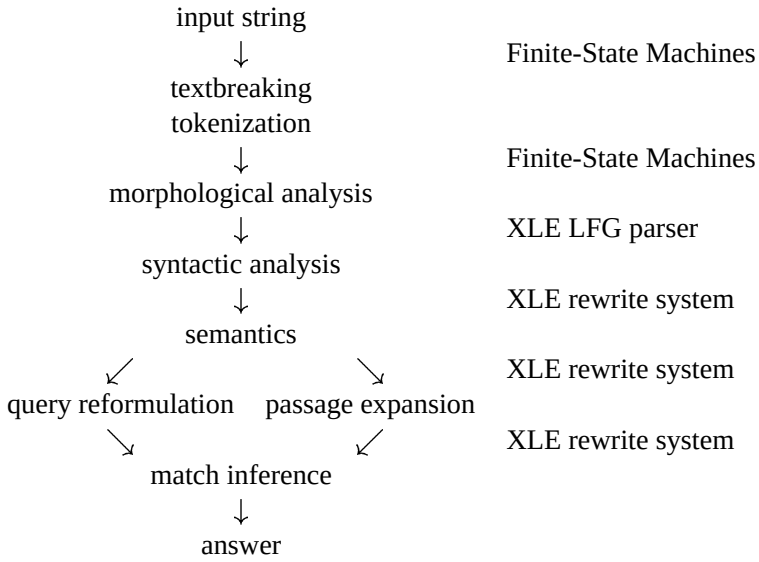


Figure 6: XLE Search and QA System Processing Pipeline

The regression system is set up to provide feedback to the rule writers maintaining both the semantic production system and the inference mechanism. In contrast to the MedSLT application, the XLE Search and QA System only involves one language. Nonetheless, there are several grammar engineers working on the different semantics levels, the inference rules, and the lexical resources used by the system. In addition, the tokenization, morphology, and syntax preprocessing (Kaplan et al.,

[3]This regression platform does not currently test the indexing mechanism. This capability is in the process of being added.

2004) slowly change over time and updates to these components can affect downstream processing in terms of the quality of the answers and of system efficiency.

3.2.1 Regression Testing Question Answering

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).[4] Given an application of use, it is rarely ambiguous what the answer should be. In contrast, one input to a translation system can have many possible outputs of varying correctness that are hard to enumerate. The upshot for QA systems is that regression runs are simpler and easier to interpret once the test suites have been constructed.

Regression test suites in the XLE QA system are separated into three groups: sanity sets, phenomenon sets, and real-world sets.

Sanity sets The entailment and contradiction detection part of the system is tested in isolation by matching queries against themselves (e.g. a passage *John walks.* is tested against a query *John walks.*); note that queries in this system do not have to be syntactically interrogative. The sanity check test suites 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. These tests are run nightly.

Phenomena sets Real-world sentences require analyses of multiple interacting phenomena. Naturally, longer sentences tend to have more diverse sets of phenomena and hence a higher chance of containing a construction that the system does not deal with well. This can lead to frustration for system engineers trying to track progress; fixing a major piece of the system can have little or no effect on a small sample of real-world examples. To alleviate this frustration we have sets of hand-crafted test examples that are focused as much as possible on single phenomena, e.g. anaphora, aliases, deverbals, implicatives. These include externally developed test suites such as the FraCaS (Cooper et al., 1996) and HP test suites (Nerbonne et al., 1988). These focused test suites are also good for quickly diagnosing problems. If all broken examples are in the deverbal test set, for example, it gives system engineers a good idea of where to start looking for bugs. These are the most important tests and are run nightly.

[4] *Wh*-questions receive a yes answer if an alignment is found between the *wh*-word in the query and an appropriate part of the 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. However, currently not all of the test suites include gold standards for this alignment.

Real-world sets The ultimate goal of the system is to work on real-world examples; so tests of those are important for assessing progress on naturally occurring data. These test suites are created by extracting sentences from corpora expected to be used in the run-time system, e.g. newspaper text or the Wikipedia. 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 used to ensure that the real-world sets use an appropriate range of queries. Currently, the XLE systems use a combination of hand-crafted queries and queries from the RTE data which were hand-crafted, but not by the XLE QA system developers. These tests are run once a week.

3.2.2 Comparing with Previous Regression Results

The point of regression testing is to compare system outputs over time. The XLE regression system automates this task as much as possible. Performance on individual test suites is graphed over time. Each test suite can be viewed individually in this way. An example is seen in Figure 7 in which a sharp dip in performance is seen for August 19 but coverage climbs after that, with a plateau in early September.

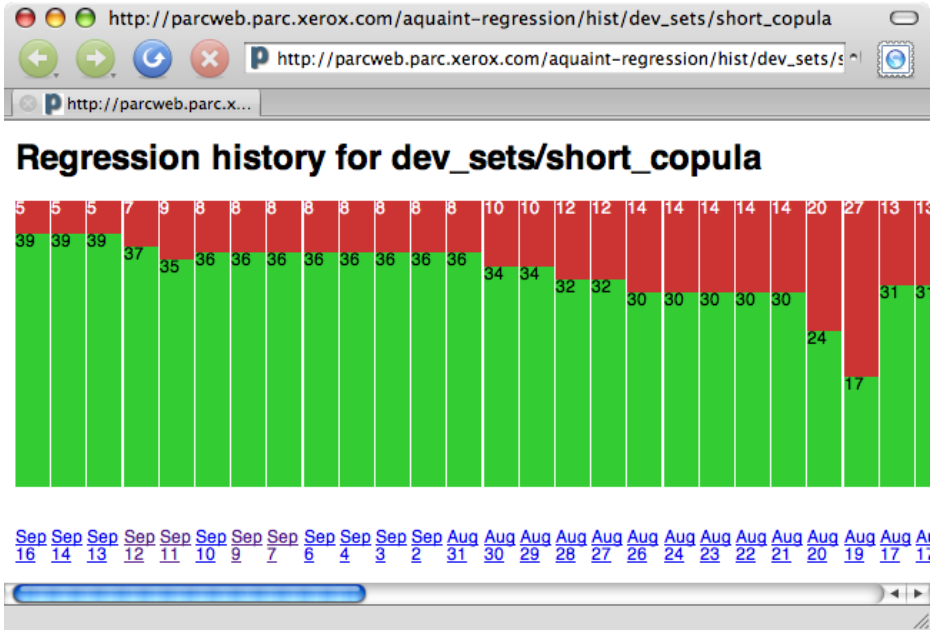


Figure 7: XLE Regression System: Performance over time

In addition, a view of each test suite is available in which the broken and fixed examples are placed at the top of the page, directly under the result summary for that test suite. This is shown in Figure 8. Below those, all the incorrect pairs are shown, then the correct ones, then a full system log.

Diffs of output, including debugging information and representations, between different versions of the system can be easily generated. The query-passage pair is

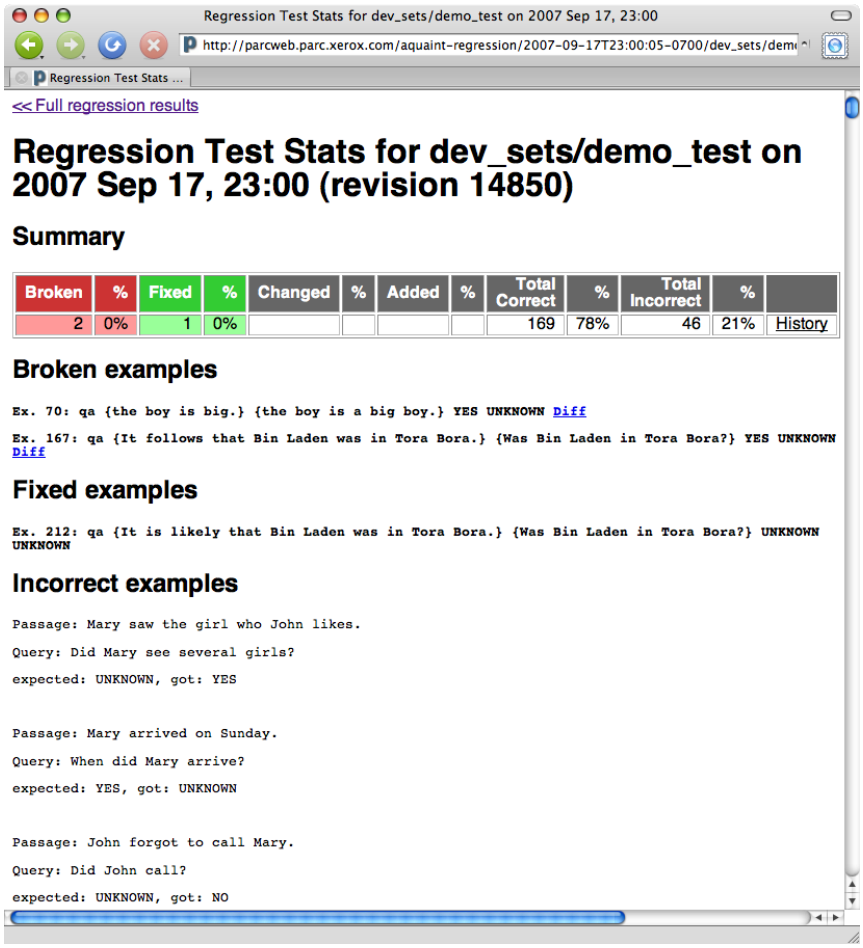


Figure 8: XLE Regression System: Detailed presentation of test suite results

run with light-weight debugging on in two versions of the system and resulting differences in the rule application and the representations are highlighted. This makes it easy for the developers to see the types of representations that were produced by previous versions of the system in comparison with the current version. In addition, the diffs of the rules triggered by each run allow the developer to see more precisely where any divergences occur. The most frequently used diff is between the current system and the previous day's. Part of a sample diff is shown in Figure 9 in which the previous day's run, shown on the left, has more possible analyses, as indicated by the larger choice space. However, images of previous days' systems are stored for rapid comparison, and for comparisons further back in time, system versions are retrieved from the svn repository.

Each time the regression testing completes a run (nightly for most test suites), an email message is sent to the developers with a summary of who committed changes

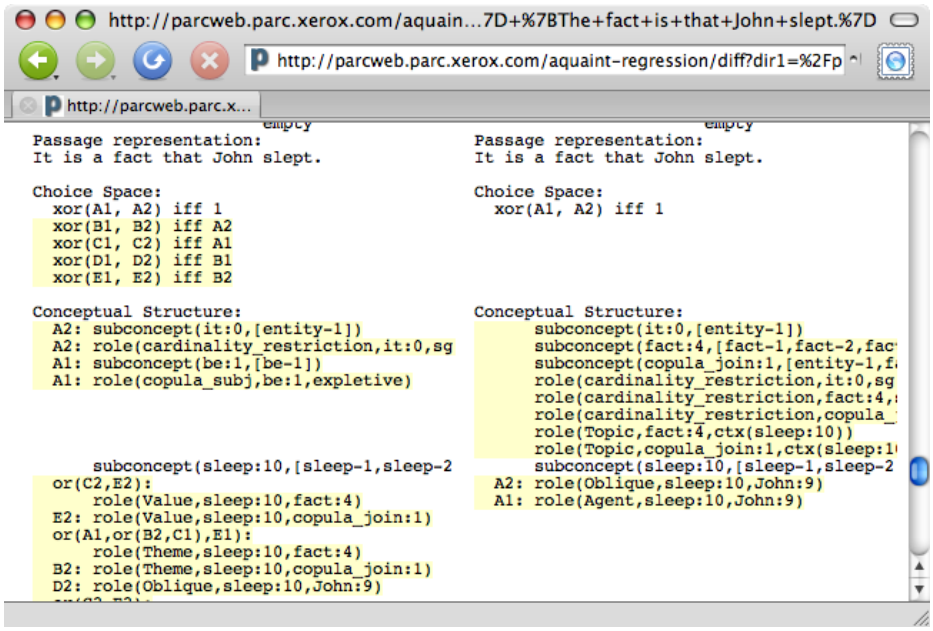


Figure 9: XLE Regression System: Sample Diff for the sentence *It is a fact that John slept.*

and how many examples were fixed and broken. In addition, a link is provided to a web server with the current graphs and diffs, as well as with links to previous results. The combination of automatic nightly regression runs with the graphical presentation of results has proven vital for the maintenance and development of the QA system.

4 Conclusions

As deep grammars are increasingly used as components of larger systems, reliable, accurate, and easy-to-use regression testing is crucial. Here, we have described regression testing techniques used to maintain large-scale grammars in two applications. The regression testing runs automatically each night and reports to the grammar developers how the performance of the system has changed. The reports include summary information as well as which examples changed, how they changed, and when they last worked correctly. It is our hope that other grammar engineering efforts can benefit from our experiences in order to more rapidly and effectively maintain and develop their grammars for applications.

References

Bobrow, Danny, Cheslow, Bob, Condoravdi, Cleo, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan, de Paiva, Valeria and Zaenen, Annie. 2007. Linguistically-based Textual Inference. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF 2007 Workshop*, CSLI Publications.

Chatzichrisafis, Nikos, Bouillon, Pierrette, Rayner, Manny, Santaholma, Marianne, Starlander, Marianne and Hockey, Beth Ann. 2006. Evaluating Task Performance for a Unidirectional Controlled Language Medical Speech Translation System. In *Proceedings of the HLT-NAACL Workshop on Medical Speech Translation.*

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

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John and Newman, Paula. 2007. XLE Documentation, on-line documentation.

Crouch, Dick and King, Tracy Holloway. 2006. Semantics via F-Structure Rewriting. In *Proceedings of LFG06*, pages 145–165, CSLI Publications.

Kaplan, Ron, Riezler, Stefan, King, Tracy Holloway, Maxwell, John T., Vasserman, Alex and Crouch, Richard. 2004. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL'04.*

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

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

Oepen, Stephan, Flickinger, Dan, Toutanova, Kristina and Manning, Chris D. 2002. LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories.*

Oepen, Stephan, Netter, Klaus and Klein, Judith. 1998. TSNLP — Test Suites for Natural Language Processing. In John Nerbonne (ed.), *Linguistic Databases*, CSLI.

Rayner, Manny and Hockey, Beth Ann. 2004. Side Effect Free Dialogue Management in a Voice Enabled Procedure Browser. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, Jeju Island, Korea.

Rayner, Manny, Hockey, Beth Ann and Bouillon, Pierrette. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI.

Rosén, Victoria, de Smedt, Koenraad, Dyvik, Helge and Meurer, Paul. 2005. TREPIL: Developing Methods and Tools for Multilevel Treebank Construction. In *Proceedings of The Fourth Workshop on Treebanks and Linguistic Theories.*

Spark-Jones, Karen and Galliers, Julia Rose. 1996. *Evaluating Natural Language Processing Systems*. Springer Verlag.

An LFG Chinese Grammar for Machine Use

Ji Fang and Tracy Holloway King
PARC

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes the Chinese grammar developed at PARC, including its three basic components: the tokenizer and tagger, lexicon and syntactic rules. Some of the challenges and issues that we have encountered in the process of development are discussed. In addition, we present our methods of handling these issues. We also illustrate how we evaluate our grammar, providing the evaluation results and some error analyses.

1 Background Introduction

This paper describes a Chinese grammar developed at the Palo Alto Research Center (PARC). This grammar is designed for machine use and is implemented in the framework of Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982; Dalrymple, 2001; Bresnan, 2001).

LFG is characterized by its two parallel levels of syntactic representation: Constituent Structure (c-structure) and Functional Structure (f-structure). C-structure encodes information about phrasal structure and linear word order. F-structure encodes information about 'the various functional relations between parts of sentences, information like what is the subject and what is the predicate' (Sells, 1985). Both c-structure information and f-structure information are carried in syntactic rules such as (1).

(1) S –> NP: (ˆ SUBJ) = !;
VP: ˆ = !.

The ˆ refers to the f-structure of the mother node and the ! refers to the f-structure of the node itself. (ˆSUBJ) = ! means that the SUBJ part of the mother's f-structure (the f-structure of the S in (1)) is the f-structure of the node itself (the f-structure of the NP in (1)). ˆ= ! means that the f-structure of the node itself (the VP in (1)) goes into the f-structure of its mother node (the S in (1)); that is, VP is the functional head of S.

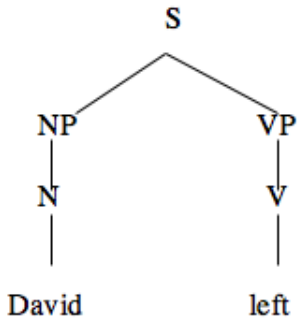
(2) shows two additional phrase structure rules. Together with the rules in (1), these will derive the c-structure and f-structure in (4) and (5) for example (3).

(2) NP –> N: ˆ = !;
VP –> V: ˆ = !.

†Fuji Xerox funded the initial research on the Chinese grammar described in this paper, and we are especially grateful to the support we have received from Tomoko Ohkuma and Hiroshi Masuichi of Fuji Xerox throughout the development. Professor Bing Swen and Professor Shiwen Yu of Beijing University have provided substantial support for the tokenizer and tagger used in this grammar. We also appreciate the feedback they provided during our conversations regarding ways to improve the tokenizer and tagger. We would also like to thank Yuqing Guo from DCU for her work in developing the gold analyses for the 200 gold sentences against which we evaluate our grammar. We also owe our thanks to Emily M. Bender for her helpful feedback and comments on this paper.

(3) David left.

(4) c-structure of (3)



(5) f-structure of (3)

$$\begin{bmatrix} \text{PRED} & \text{'leave<SUBJ>'} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'DAVID'} \end{bmatrix} \end{bmatrix}$$

'leave <SUBJ>' in (5) means that the lexical item 'leave' subcategorizes for a SUBJ. This information comes from the lexicon portion of the grammar.

PARC has been involved in the Parallel Grammar (ParGram) project, which is a world-wide collaborative effort that aims to produce robust and large scale grammars for a wide variety of languages, such as English, German, Japanese, Turkish and Arabic (Butt et al., 1999, 2002). All of these grammars are written within the LFG framework and are implemented on the XLE system (Crouch et al., 2006; Maxwell and Kaplan, 1996) developed by PARC. The Chinese grammar described in this paper is part of the ParGram project.

2 The Chinese Grammar Developed at PARC

Like other grammars in the ParGram project, PARC's Chinese grammar is developed on the XLE system. To parse a sentence, the system minimally requires three types of linguistic specifications: a tokenizer/morphology, a lexicon and syntactic rules. This section describes these three parts of the Chinese grammar.

2.1 Morphology: Segmentation and Tagging

For languages that have morphological inflection such as number, gender, case etc, the morphology processing component of the grammar normally includes a mor-

phological analyzer and a tokenizer.[1] The morphological analyzer also provides part of speech (POS) information for words: therefore, it also functions as a tagger.

In contrast, Chinese is an isolating language, which does not have morphological inflections. Therefore, our grammar does not have a morphological analyzer. Instead, PARC's Chinese grammar uses the tokenizer and tagger developed by Beijing University.[2] The tokenizer and tagger is plugged into the system as a library transducer (Crouch et al., 2006). For an input string such as (6a), the output from the tokenizer and tagger is (6b), which is in turn fed to the XLE system as the input string for syntactic analysis. In order to construct a tree for each lexical item from this type of tagged string, we specify sub-lexical rules such as (7).

(6) a. 窗户开了。
chuāng hu kāi le

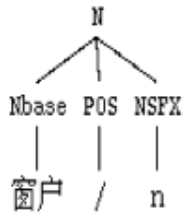
b. 窗户/n 开/v 了/y 。/w
chuānghu kāi le
window open ASP[3] .
'The window is open.'

(7) N –> Nbase
POS
NSFX.

Following (7) and combining information from the lexicon entries for '窗户', '/' and 'n' (as illustrated in (8)), XLE can build a tree and an f-structure for the lexical item "窗户" as shown in (9).

(8) 窗户 Nbase * @ (NON-ANIM-NOUN 窗户).
/ POS * .
n NSFX * (^ CHECK[4]_NSFX)= + .

(9)

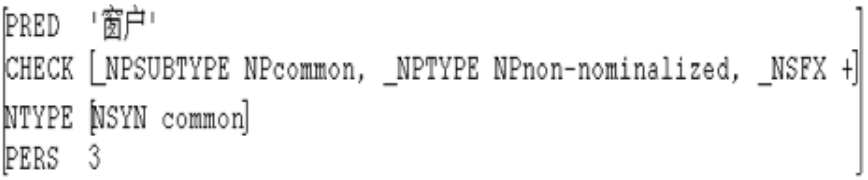


[1]Many languages use finite state morphologies (Beesley and Karttunen, 2003) as part of their XLE grammars (Kaplan et al., 2004a).

[2]http://www.icl.pku.edu.cn/icl_res/

[3]ASP stands for aspect marker.

[4]The CHECK feature is a feature used throughout the ParGram to indicate features that are necessary for internal processing, but not necessary for applications. Applications built on top of the ParGram grammars can delete the CHECK features in their initial processing.



As is broadly acknowledged, Chinese segmentation and tagging are notoriously difficult problems. This is because Chinese does not have morphological inflection, and furthermore, spaces are not inserted between words in written text. For example, the string "有意见" can be segmented as (10a) or (10b), depending on the context.

(10) a. 有 意见
yǒu yìjian
have disagreement

b. 有意 见
yǒuyì jiàn
have the intention meet

The contrast shown in (11) illustrates that even a string that is not ambiguous in terms of segmentation can still be ambiguous in terms of tagging.

(11) a. 白/a 花/n
bái huā
white flower

b. 白/d 花/v
bái huā
in vain spend
'spend (money, time, energy etc.) in vain'

Not surprisingly, the performance of the tokenizer and tagger presents some serious challenges to our grammar as described below.

Challenge #1: Low accuracy of segmentation and tagging Although the tokenizer and tagger developed by Beijing University is state-of-the-art, it achieves about 93% accuracy in segmenting and tagging sentences from the Chinese Treebank5.1 according to our measurements. This level of performance means that each segmented and tagged Chinese sentence of more than 10 words would typically have at least one mistake. Obviously, segmentation and tagging errors directly cause incorrect syntactic analysis and even complete parsing failures.

Challenge #2: Too many verbs In addition to the general accuracy problem, our grammar also suffers from some specific linguistic decisions adopted by the tokenizer and tagger. One such case is illustrated below.

(12) a. 检查/v 病人/n
jiǎnchá bìngren
examine patient

b. 做/v 个/q 检查/v
zuò ge jiǎnchá
do CL examination

In (12), the word 检查 jiǎnchá corresponds to a verb meaning 'examine' in English in (12a), and it corresponds to a noun meaning 'examination' in (12b). Nevertheless, both meanings share the same written form. Zhu (1982, 1985) and Yu (2003) argue that the same word can appear in different syntactic positions and have different grammatical functions; however, that word does not belong to different word categories and should be assigned just one POS tag. Adopting this theory, the tagger developed by Beijing University tags both 检查 jiǎnchá in (12) as a verb. This decision might not be an issue for other tasks or other systems; however, it turns out to be problematic for our grammar.

First, this decision can cause parsing failures. For example, our grammar restricts the category following a classifier in Chinese to be a Noun Phrase (NP), which we believe to be a correct generalization. Following this rule, (12b) will be rejected by the parser because the classifier 个 ge is followed by a verb 检查 jiǎnchá.

This decision also poses an efficiency problem for our grammar. In Chinese, a majority of the verbs have at least two subcategorization possibilities: intransitive and transitive. In the LFG framework, each verb has to satisfy its subcategorization requirements in order to successfully unify. Therefore, putting intransitive and transitive verb entries for everything that is tagged as a verb produces many extra edges in the chart as those verbs try to combine with the words around them as subjects and objects. Consequently, verbs are computationally expensive, and tagging many words as verbs can significantly slow down the parser.

Because our goal is to parse Chinese written text that is not manually segmented or tagged, our grammar implicitly inherits all of the challenges for Chinese segmentation and tagging as well.

Our initial explorations in this area are two-fold. First, we improved the tokenizer and tagger by directly modifying the existing lexical entries and by adding new lexical entries to the dictionary that the tokenizer and tagger use. At the same time, we improve the final segmentation and tagging results by using finite state (FST) rules to post-process the original output from the tokenizer and tagger. For example, a FST rule such as (13) can change an output string from the tokenizer and tagger such as (12b) (repeated below as (14)) to be (15).

(13) v ->n || q "TB" CHARˆ {1,2} "/" _;

(14) 做/v 个/q 检查/v
zuò ge jiǎnchá
do CL examination

(15) 做/v　个/q　检查/n

What (13) specifies is that if a 'v' appears after a '/' following one or two characters, which in turn appear(s) after a 'q' followed by a token boundary (TB), change that 'v' into 'n'. (15) is derived by applying (13) to (14).

In this approach, more information is available in the output string from the tokenizer and tagger compared to the original raw string; by taking into account this additional information, the final segmentation and tagging results are more accurate. For example, compared to the original raw string 做个检查, the output string from the tokenizer and tagger 做/v 个/q 检查/v contains additional information indicating that the string 检查 is preceded by a classifier. With this additional information available, we can safely and accurately change the POS tag of 检查 in this case to be a noun without impacting the tagging of 检查 in other circumstances.

It is also noteworthy that XLE allows non-deterministic segmentation and tagging. In other words, in cases where it is hard to resolve the ambiguity of segmentation or tagging locally, the XLE parser accepts a string with multiple segmentation possibilities and a token with multiple possible tags. For example, the word 选举 xuǎnjǔ 'elect/election' is equally frequently used as a verb and as a noun. In this case, the best solution is to allow both the 'v' and the 'n' tag and hand off the resolution of that ambiguity to the syntactic processor. Similarly, when the ambiguity of segmentation is hard to resolve locally, multiple segmentation results for a string are allowed, and the XLE parser will try all of these different results as input to the grammar.

Based on our initial observations, the system has been improving with the FST rules integrated. However, more work still needs to be done in this area. As part of this process, we plan to evaluate how much the FST rules improve the tokenizer and tagger against grammar performance on real world data.

To summarize, this section describes the tokenizer and tagger that we integrate into our grammar, the challenges that our grammar has to face in this regard and our approach to improve the tokenizer and tagger. The following sections describe the other two important components of the grammar, namely the lexicon and syntactic rules.

2.2 Lexicon

The lexicon component of the grammar contains lexical entries specifying information particular to different lexical items. For example, (16) is the lexical entry for the noun 猫 māo 'cat', and (17) is the lexical entry for the verb 加入 jiārù 'join' in the Chinese grammar.

(16) 猫　　Nbase * @ (ANIM-NOUN 猫).

(17) 加入 Vbase * { @(V-SUBJ 加入)
(^ SUBJ CHECK _SUBJ-TYPE)=c np
|@(V-SUBJ-OBJ 加入)
(^ SUBJ CHECK _SUBJ-TYPE)=c np}.

(16) specifies that the category of the lexical item 猫 māo is Nbase. Combining this entry with information from lexical entries for POS tags produced by the tagger (as shown in (18)) and sub-lexical rules such as (7) (repeated below as (19)), XLE can build a c-structure such as (20) for 猫 māo.

(18) / POS * .
n NSFX * (^ CHECK _NSFX)= +.

(19) N –> Nbase
POS
NSFX.

(20) c-structure of 猫

(16) also invokes a template ANIM-NOUN (shown in (21)), which defines features and values for all animate nouns in Chinese.

(21) ANIM-NOUN(_P) =
(^ PRED)='_P'
(^ PERS)=3
(^ ANIM)=+
(^ NTYPE NSYN)=common
(^ CHECK _NPTYPE)=NPnon-nominalized
(^ CHECK _NPSUBTYPE)=NPcommon

Combining this information, XLE can build an f-structure for 猫, as illustrated in (22).

(22) f-structure of 猫

```
PRED   '猫'
CHECK [_NPSUBTYPE NPcommon, _NPTYPE NPnon-nominalized, _NSFX +]
NTYPE [NSYN common]
ANIM +, PERS 3
```

Similarly, (16) defines 加入 jiārù ‘join’ as a verb that can be used either intransitively or transitively. It also specifies that the subject of 加入 must be a NP.

Currently, the lexicon component of the Chinese grammar has several hundred manually coded lexical entries, including closed class items such as punctuation. We handle words that do not have a listed lexical entry through a “guesser” lexical entry exemplified in (23).

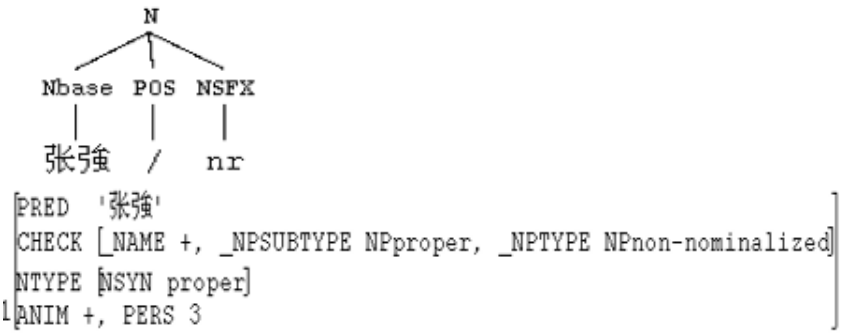
(23) -unknown Nbase * @(PROPER-ANIM-NOUN %stem)
(^ CHECK _NAME)=c +.

(23) specifies that if a lexical item does not have a lexical entry elsewhere, it can be posited as an Nbase; if it has a feature ‘CHECK _NAME’ whose value is ‘+’, then it is an animate proper noun. The value ‘+’ of ‘CHECK _NAME’ is derived from the POS tag produced by the tokenizer and tagger: the tagger tags a person’s name as ‘nr’, and we assign ‘(^ CHECK _NAME) = +’ in the lexical entry of ‘nr’, as shown in (24).

(24) nr NSFX * (^ CHECK _NAME)= +.

Combining information from (23), (24) and the template of PROPER-ANIM-NOUN, the c-structure and f-structure of a person’s name such as 张强 zhāngqiáng ‘Zhang, Qiang’ are illustrated in (25).

(25)



In addition to names, our guesser postulates locative, time and common nouns, as well as adjectives, adverbs, numbers, classifiers, conjuncts, prepositions and verbs in a similar way: we first write a lexical entry for each tag, such as (24) for the ‘nr’ tag; we also assign a value to a feature for each tag, for instance, ‘(^ CHECK _NAME)= +’ is assigned for the tag ‘nr’ (as in (23)), and ‘(^ CHECK _VSFX)= +’ is assigned for the tag ‘v’. The guesser then posits the category of the unknown item based on the ability to form a particular c-structure category via the sublexical rules and on the value of particular features. This process is quite efficient in part because XLE first builds the c-structure, before any unification occurs, and hence many possible entries are eliminated early in the parsing process. For example, if the f-structure of the unknown item contains a feature ‘(^ CHECK _VSFX)’ whose value is ‘+’, that item must be associated with a tag ‘v’, thus the guesser

can postulate the unknown item as a verb. Note that because the tags and the '/' do not have PREDs, their features and values are projected to the mother node's f-structure, which is identical to the f-structure of the unknown lexical item.

Verbs pose the biggest challenge to the guesser. In LFG, subcategorization information is required for verbs. However, this information is not encoded in the 'v' tag of verbs returned by the tagger, and we have not found any suitable resource from which we can extract the subcategorization requirements for verbs in Chinese. We have manually coded the entries for some high frequency verbs.[5] These verbs do not go through the *-unknown* entry. For all other verbs, our compromise solution is to postulate each unknown verb to be either intransitive or transitive. The guesser also allows a verb to subcategorize for a XCOMP, if the PRED form of the XCOMP is the PRED form of one of the verbs, such as 为wéi 'be'.[6]

Nevertheless, the lack of reliable and complete subcategorization information for Chinese verbs poses challenges for our grammar, as discussed in the evaluation section of this paper. Possible enhancements are discussed in section 4.

2.3 Syntactic Rules

The third part of the grammar involves the Chinese syntactic rules. Currently the grammar has 114 rules with 2203 states and 4301 arcs.[7] This means that the grammar has 114 left-hand side categories (such as the X in 'X -> Y Z') in its phrase-structure rules, and these 114 rules compile into a collection of finite-state machines with 2203 states and 4301 arcs (Butt et al., 2002). The grammar covers common phrasal constructions such as NPs, VPs, ADJPs, PPs and ADVPs. The grammar also covers all four clause types in Chinese: declarative, interrogative, exclamatory and imperative.

Due in part to the lack of morphology, Chinese tends to present many ambiguities at both the c-structure and f-structure level. For example, for a NP such as (26), the internal NP structure can be very ambiguous (5 trees), as shown in (27).

(26)	国民	生产	总值
	guómín	shēngchǎn	zǒngzhí
	people	produce	total value
	'GDP'		

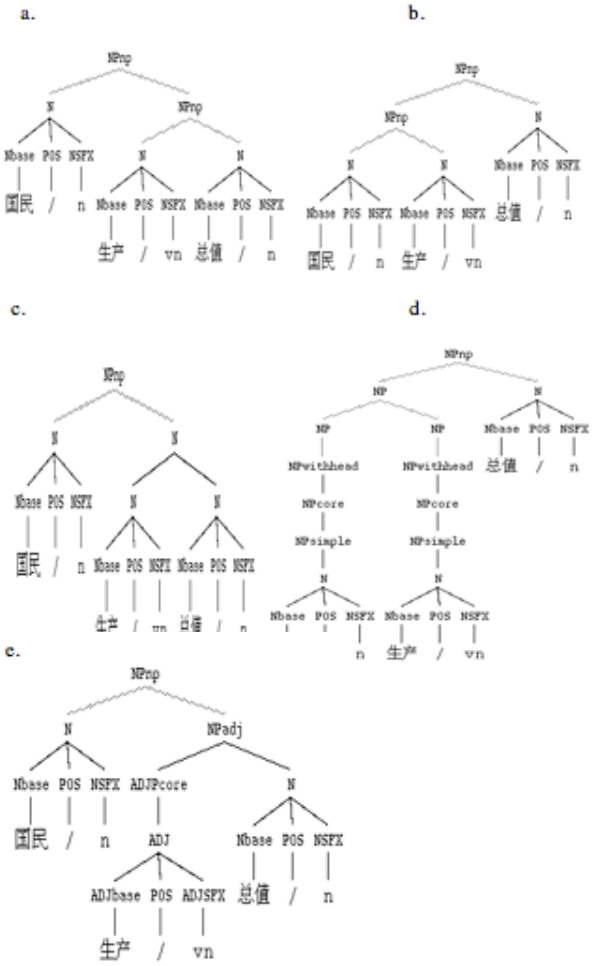
[5]Currently the grammar has manually encoded entries for the 20 most frequent verbs, and our goal is to expand the lexicon to contain entries for the 100 most frequent verbs in Chinese.

[6]Verbs such as 为 wéi 'be' are frequently used as the 补语 bǔyǔ ('complement' in conventional Chinese linguistic terminology) in a 动补结构 dòng bǔ jiégòu 'verb-complement construction' such as 变为 biàn wéi 'change to be'. 补语 bǔyǔ in the 动补结构 dòng bǔ jiégòu is treated as XCOMP in our grammar.

Some common XCOMP verbs are 完wán 'complete', 尽jìn 'complete', 成chéng 'succeed', 上来shànglai 'go up (towards the speaker)' , 上去shàngqu 'go up (away from the speaker)', 下来xiàlai 'go down (towards the speaker)', 下去xiàqu 'go up (away from the speaker)', 起来qǐlai 'get up/begin', 起qǐ 'get up/begin', 去qù 'go', 来lái 'come', 出来chūlai 'come out', 出去chūqu 'go out' and 回来huílai 'come back'.

[7]XLE compiles the grammar into a finite-state machine.

(27)



The corresponding f-structures of these c-structures are also different: while (27c) and (27d) involve coordination, the others not.

The ambiguity issue is one of the contributing factors to the current grammar's efficiency issues. The following section describes this problem in a greater detail.

3 Evaluation and Error Analysis

To evaluate the coverage and accuracy of the grammar (Crouch et al., 2002; Kaplan et al., 2004b), we use a set of 200 sentences from the CTB5 (Xue et al., 2002, 2005) chosen by Dublin City University (DCU) as gold standard sentences for evaluating Chinese deep grammars. These 200 sentences are 10–20 words long. 50.5% of the sentences are chosen from the Xinhua sources, 3.5% are from HKSAR and 46% are from Sinorama. The topics of the gold standard sentences cover economics, politics, culture, sports and entertainment. The writing style of the Xinhua and HKSAR sentences is formal, whereas the writing style of the Sinorama sentences

are mixed: some are formal, and some are colloquial. All 200 sentences were unseen to the grammar prior to the evaluation.

We also use DCU's gold analysis as a basis for evaluating the accuracy of the grammar. However, PARC's analysis is based on the segmentation and tagging results from the integrated tokenizer and tagger, which are different from the segmentation and tagging in CTB5 on which the DCU gold standard analysis is based. Therefore, some of the errors in the results shown below are caused by the different segmentation and tagging standards adopted by CTB5 and the tokenizer and tagger developed by Beijing University. The reason why we did not evaluate our grammar based on gold segmented and tagged sentences is because we want to know how good the results would be over novel, untokenized text.

Parsing the 200 gold standard sentences with PARC's Chinese grammar (as of March, 2007), 188 sentences had full parses, 7 sentences had fragmented parses, 4 sentences ran out of storage (the maximum storage is set as 1500 MB in this grammar), and 1 sentence had 0 parse, as shown in Table 1. Fragmentation Rate is 3.5%.

Table 1: Coverage Results

Total	Full Parses	Fragmented Parses	Ran Out of Storage	No Parse
200	188	7	4	1

To evaluate the accuracy, we adopt the same algorithm described in Crouch et al. (2002). The results are shown in Table 2.

Table 2: Accuracy Results

TOTAL:	precision=73.1	recall=72.4	f-score=72.7
DEPENDENCY	PRECISION	RECALL	FSCORE
adjunct	939/1228 = 76	939/1267 = 74	75
comp	15/22 = 68	15/25 = 60	64
conj	182/271 = 67	182/278 = 65	66
obj	330/452 = 73	330/449 = 73	73
obl_ag	9/9 = 100	9/9 = 100	100
passive	9/9 = 100	9/9 = 100	100
subj	318/478 = 67	318/455 = 70	68
topic	0/0 = 0	0/1 = 0	0
xcomp	42/53 = 79	42/55 = 76	78

The Chinese gold standard has only predicate argument/adjunct structure (that is, everything with a PRED and the path into it). There are no 'easy' features like

CLAUSE-TYPE, V-TYPE, PERS, which tend to be correct if the core structure is correct. Therefore, the f-score would likely be higher if the Chinese gold standard did contain those features.

As mentioned above, some of the mismatches between the analyses produced by PARC's Chinese grammar and DCU's gold analyses are caused by the different segmentation and tagging standards adopted by CTB5 and the tokenizer and tagger that our grammar uses. For example, CTB5 treats 第 dì 'ordinal marker' + number as one word, whereas the tokenizer used in our grammar treats it as two separate words.

The tagging standard between the two systems is also different. For example, for the same string in (28), (29a) is the tagging results from the tagger in our grammar, and (29b) is the tagging result from CTB5.

(28)

证券	市场	健康	发展	的
zhèngquàn	shìchǎng	jiànkāng	fāzhǎn	de
stock	market	healthily/ health	develop/ development	MM[8]
重大	举措			
zhòngdà	jǔcuò			
important	measure			

(29) a. 证券/n 市场/n 健康/a 发展/v 的/u 重大/a 举措/n

b. 证券_NN 市场_NN 健康_JJ 发展_NN 的_DEG 重大_JJ 举措_NN

The major difference between (29a) and (29b) is that in (29a), the word 发展 fāzhǎn is tagged as a verb, meaning 'develop', while in (29b), it is tagged as a noun, meaning 'development'. At first glance, (29b) seems to yield a reading of 'the important measure regarding the healthy development of the stock market', which is very parallel to the English structure. However, (29b) cannot yield such a reading: this is because if 发展 fāzhǎn is a noun, and its adjunct is 健康 jiànkāng, we should be able to insert a 的 de, which introduces a head NP, rather than a 地 de, which introduces a head VP, between them. However, 健康的发展 jiànkāngdefāzhǎn only entails 'the development of health' in Chinese; in contrast, 健康地发展 jiànkāngdefāzhǎn can entail 'healthy development (the nominalization of 'develop healthily')'. Therefore, while (29a) would yield a reading of 'the important measure (which assures that) the stock market develops healthily' or 'the important measure (which assures) the healthy development of the stock market', (29b) would mean 'the important measure regarding the development of the stock market's health'. Such a difference would yield very different analyses. Based on (29a), 证券市场 zhèngquàn shìchǎng 'stock market' is the subject of 发展 fāzhǎn 'develop'; while based on (29b), 证券市场 zhèngquàn shìchǎng 'stock market' is an adjunct.

[8]MM stands for modifier marker.

In addition to this mismatch, another significant source of errors is our incomplete lexicon resources. In the 200 gold sentences, 17 sentences receive incorrect analyses for this reason. Among the 17 sentences, five fail due to the lack of proper subcategorization information for verbs, while the remaining 12 fail due to missing lexical entries for other lexical items.

Chinese tends to be ambiguous in both the c-structure and f-structure levels as described above. One way to control ambiguity is to use a special optimality (OT) mark (Frank et al., 2001) called a STOPPOINT provided by the XLE system. In XLE, if an analysis contains an OT mark that is ranked behind the STOPPOINT, that analysis is not tried unless everything else fails. Therefore STOPPOINT is useful for eliminating rare and incorrect analyses when correct analyses are present and for speeding up the parser in those cases.

Ironically, although Chinese has been recognized as a topic prominent language (Li and Thompson, 1981), we place the topic analysis before the STOPPOINT, because we have observed that allowing the topic analysis significantly slows down the parsing while not greatly increasing accuracy. The following three reasons are likely to be responsible for the inefficiency caused by including the topic analysis in the grammar. First, the position of topics in Chinese is flexible. A topic can occur in the first or second NP position, and a sentence can have more than one topic, as demonstrated in (30).

(30) a. 苹果 我 喜欢。
píngguǒ wǒ xǐhuān
apple I like
'Apples, I like.'

b. 大 城市 北京 我 最 熟悉。
dà chéngshì běijīng wǒ zuì shúxī
big city Beijing I most familiar
'Among big cities, I am most familiar with Beijing.'

(30a) has one topic, which is 苹果 píngguǒ 'apple' that appears in the first NP position; (30b) has two topics. While the external topic 大城市 dà chéngshì 'big cities' occurs in the first NP position, the internal topic 北京 běijīng 'Beijing' occurs in the second NP position. Second, unlike the topic in English, which must be linked to another grammatical function, the topic in Chinese is not necessarily linked to any another grammatical function. For example, while the topic in (30a) is linked to the grammatical function of object, the topics in (30b) are not linked to any other grammatical function. Third, topics generally occur in a 'NP1 NP2' sequence at the sentence-initial position; however, it is very common to analyze NP1 as the adjunct, possessor or conjunct of NP2 in a 'NP1 NP2' sequence. Therefore, allowing topic analyses significantly increases the level of ambiguity for sentence initial 'NP1 NP2' sequences, which are extremely common in Chinese sentences according to our observation.

At the same time, the topic function often overlaps with other grammatical functions such as adjunct in Chinese. For example, in (31), 他 tā 'he' can both be understood as the topic of the entire sentence or the adjunct of 肚子 dùzi 'stomach'. Therefore, it does not seem to be a significant drawback if the topic analysis is blocked unless it is the only possible analysis.

(31) 他 肚子 饿。
tā dùzi è
he stomach hungry
'He is hungry.'

By placing the topic analyses behind the STOPPOINT, the grammar's efficiency is improved. However, occasionally, the intended topic analysis will be suboptimal and hence not available. In the 200 gold sentences, one sentence should have a topic analysis that is incorrectly suppressed by our system as shown in the accuracy results above.

Another method that we adopt to control ambiguity is to use OT marks more generally to rank preferences for different analyses. Through this method, the less common analyses can be suppressed as suboptimal analyses. All of the OT marks in the Chinese grammar are manually coded, and it is noteworthy that a significant number (24 out of the 200 sentences) of correct analyses are incorrectly suppressed as suboptimal analysis by the OT marks specified in the grammar. The suppressed suboptimal analyses cannot be picked to compare against the gold standard, which implies that the OT marks in the Chinese grammar need to be better tuned in order to improve the grammar's performance.

4 Summary and Future Work

This paper describes the Chinese grammar developed at PARC, including its three basic components, namely, the morphology, lexicon and syntactic rules. We also describe the challenges and issues that we have encountered in the process of development, as well as our methods of handling these issues. In addition, we illustrate how we evaluate our grammar, including the evaluation results and some error analysis.

The three major challenges currently confronted by our grammar are (1) the tokenizer and tagger; (2) lexicon resources such as subcategorization requirements of verbs; and (3) ambiguity control.

As far as the tokenizer and tagger is concerned, the initial results of improvements to segmentation and tagging accuracy by using FST patch rules to post-process the original results returned by the tokenizer and tagger are encouraging. We will continue our investigations in this direction and plan to investigate integrating machine learning algorithms in this process.

Because the subcategorization information of verbs is critical to our system, we are looking for suitable resources from which we can automatically extract this

information. Resources such as Chinese Word Net or electronic verb dictionaries can be useful. We are also considering learning the subcategorization information from the Chinese Treebank.

C-structure pruning (Crouch et al., 2006) has proven to be very effective in terms of reducing ambiguity and accelerating the parser for the English grammar developed at PARC and the German grammar developed at the University of Stuttgart. We expect that this technique can help mitigate the ambiguity issue of our Chinese grammar as well.

Despite all of the challenges, the Chinese grammar described in this paper has reached a relatively stable stage, and we are planning to use it as a base to produce Chinese core semantics parallel to that developed for English (Crouch and King, 2006). We also plan to use this grammar to start initial exploration on Chinese-English and English-Chinese machine translation.

References

Beesley, Kenneth R. and Karttunen, Lauri. 2003. *Finite State Morphology*. CSLI Publications.

Bresnan, Joan. 2001. *Lexical-Functional Syntax*. Blackwell Publishers.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7.

Butt, Miriam, King, Tracy Holloway, Niño, María-Eugenia and Segond, Frédérique. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John and Newman, Paula. 2006. XLE documentation, http://www2.parc.com/isl/groups/nltt/xle/doc/.

Crouch, Dick and King, Tracy Holloway. 2006. Semantics via F-Structure Rewriting. In *Proceedings of LFG06, CSLI On-line publications, pp. 145-165.*.

Crouch, Richard S., abd Tracy Holloway King, Ronald Kaplan and Riezler, Stefan. 2002. A comparison of evaluation metrics for a broad coverage parser. In *Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems: LREC 2002 Workshop*.

Dalrymple, Mary. 2001. *Syntax and Semantics. Volume 34: Lexical Functional Grammar*. Academic Press.

Frank, Anette, King, Tracy Holloway, Kuhn, Jonas and Maxwell, John T. 2001. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In Peter Sells (ed.), *Formal and Empirical Issues in Optimality Theoretic Syntax*, pages 367–397, CSLI Publications.

Kaplan, Ron, Maxwell, John T., King, Tracy Holloway and Crouch, Richard. 2004a. Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP (ESSLLI)*.

Kaplan, Ron, Riezler, Stefan, King, Tracy Holloway, Maxwell, John T., Vasserman, Alex and Crouch, Richard. 2004b. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL'04*.

Kaplan, Ronald. M and Bresnan, Joan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan (ed.), *The Mental Representation of Grammatical Relations*, pages 173–281, The MIT Press.

Li, Charles N. and Thompson, Sandra A. 1981. *Mandarin Chinese: A Functional Reference Grammar*. University of California Press.

Maxwell, John and Kaplan, Ron. 1996. An Efficient Parser for LFG. In *Proceedings of the First LFG Conference*, CSLI Publications.

Sells, Peter. 1985. *Lectures on Contemporary Syntactic Theories*. CSLI Publications.

Xue, Nianwen, Chiou, Fu-Dong and Palmer, Martha. 2002. Building a Large-Scale Annotated Chinese Corpus. In *Proceedings of the 19th. International Conference on Computational Linguistics*.

Xue, Nianwen, Xia, Fei, Chiou, Fu-Dong and Palmer, Martha. 2005. The Penn Chinese TreeBank: Phrase Structure Annotation of a Large Corpus. *Natural Language Engineering* pages 207–238.

Yu, Shiwen et al. 2003. *The Grammatical Knowledge-base of Contemporary Chinese — A Complete Specification*. Qinghua University Press.

Zhu, Dexi. 1982. *Yufa Jiangyi (Lectures on Grammar)*. Shangwu Yinshuguan.

Zhu, Dexi. 1985. *Yufa Dawen (Questions and Answers Regarding Chinese Grammar)*. Shangwu Yinshuguan.

On 'Deep Evaluation' for Individual Computational Grammars and for Cross-Framework Comparison

Lars Hellan

Norwegian University of Technology, Trondheim

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

A rather difficult point in grammar engineering evaluation is how to test and compare for analytic adequacy. A test design for 'deep' grammars is here proposed, where a parse is considered valid only if the assignment of syntactic and semantic structures that it displays obey certain conditions. The set of grammatical sentences in the test suite is construed as leaf types in a construction ontology, where the top types introduce the discriminants according to which constructions are categorized. These discriminants conform to notions shared across linguistic frameworks, and the validity conditions are defined within a well-known space of analytic parameters. One may envisage that with such a design, a meeting point can emerge for comparing frameworks with regard to agreed-upon aspects of linguistic content, and individual grammars with regard to their analytic aims and actual achievements relative to the aims.

> In Phillip Pullman's *His dark materials*, humans in one of the worlds have their soul partly realized as a little animal always accompanying them, sharing their thinking and emotions, but still behaving partly as independent agents; they are called ***daemons***.

1 Introduction

One way in which to improve quality and coverage of a grammar is to systematize its test suites by assembling construction types according to a fixed set of theoretically grounded parameters. Once the parameters are decided upon, such a systematicization can provide one dimension of desirable independence from the actual day-to-day development and testing. To be presented here is the composition of a test suite for verb constructions, developed in connection with the Norwegian HPSG-based computational grammar NorSource. The composition of the test suite reflects parameters of verb construction analysis which the grammar aims at implementing.

In this approach, consistency and perspicuity of the test suite is induced by the construction of a formal ontology of construction types, whose classifying properties match those reflected in the composition of the test

suite. Moreover, the ontology is defined in terms of attributes and values, which makes it possible to assign to any one (grammatical) sentence in the test suite a feature structure reflecting those properties held to distinguish the construction type represented by the sentence in question from other construction types. An example of such a feature structure, reflecting the grammatical concerns of NorSource but distinct from the feature structure NorSource itself produces for the sentence, is discussed below. In its capacity as a small system in its own right, we call the ontology, with the test sentences as its leaf types, a *Test Daemon*. [1]

A further perspective on such a system is the following:

For computational grammars aiming at exposing the principles of morpho-syntactic and semantic composition of a language, one would welcome a mechanism which at least semi-automatically could allow one to test for the *adequacy* of the parses produced.[2]

Analytic adequacy is not an absolute measure. Linguistic frameworks differ as to what they recognize as adequate analyses, both for construction types and for over-all analytic design, and such diversities are reflected in computational grammars, reflecting the assumptions of the linguistic framework of which they are part, and within each framework, and even for the same constructions in the same language, differing in their analyses. So, if we want to develop a mechanism of adequacy testing, it cannot be a mechanism of *judgement*, but rather one where each grammar makes (i) a *declaration of what it wants to achieve*, and (ii) a way of displaying, for any parse, how that parse *lives up to the ambitions* of the grammar. Let us call these the *declaration part* and the *fulfillment part* of the mechanism,

---

[1] *NorSource* is an LKB-based grammar of Norwegian, currently developed by L. Hellan, B. Waldron and D. Beermann at NTNU, Trondheim (http://www.ling.hf.ntnu.no/forskning/norsource/). It was initiated in 2002, in the EU-project *DeepThought*, and is by now rather large; one of its features is a fairly detailed semantics (cf. Hellan and Beermann 2005). NorSource is part of the DELPH-IN consortium (http://www.delph-in.net/).

This work started in the TROLL project in the late '80s, as a construction inventory enterprise, much prior to the construction of a parsing grammar. (Both then and now, the notion *construction* is used in its standard meaning, and not necessarily adopting criteria of the Construction Grammar framework (Goldberg 1995).)

The Daemon version described here is still in its infancy. For comments on many aspects of the work I thank Dorothee Beermann, and for the design of the Ga system, also Mary Esther Kropp Dakubu and Felix Ameka. The system itself can most easily be obtained from the author at *lars.hellan@hf.ntnu.no*.

[2] We take for granted the availability of standard test suites and the apparatus developed in each framework based on whether sentences parse at all or not.

respectively. The test suite design described above - the Test Daemon - would be one form of a *declaration* part of such a system. We now comment on these parts.

2 The *declaration* part

The declaration part will require that the analytic specifications employed are perspicuous, and that their space of notions and distinctions is predictable. In the present connection, the latter means that the factors dealt with are those that the linguistic community at large may expect to be considered in connection with verbal constructions, such as aspect, tense, grammatical functions, thematic roles and their linking, and control patterns. Perspicuity will mean that the terms used are sufficiently rooted in at least one tradition of research to allow the identification of equivalents or correspondents in other frameworks. Below is an example of a type display aimed at meeting these requirements:

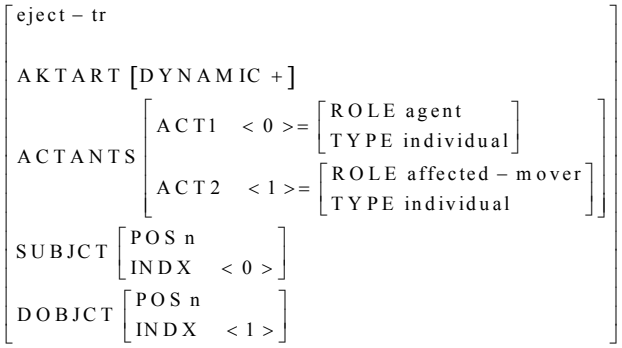


Fig. 1. The type *eject-tr,* as exemplified by *She throws the ball*

This feature structure displays grammatical functions by (slightly abbreviated) labels generally understood, and their linking to semantic 'participants' through perspicuous reentrancy. Thematic roles and ontological type are indicated by recognized labels, and the partial Aktionart specification is also standard. Such a feature structure, thus, illustrates what one might want as a perspicuous display of the factors mentioned. (In effect, it is also what one of the systems mentioned below provides.)

Intellectually, as said, the declarations part, or the *Test Daemon*, may have to live slightly outside of the grammar itself, since it must attain a level of intelligibility and perhaps quality which the parses produced by the grammar cannot always be expected to provide. If the test Daemon sits outside the grammar, then its most trivial materialization would be as a hand-provided recipe accompanying each sentence token in the test suite. A

more interesting design is to organize the set of grammatical sentences as leaf types in a construction ontology, where salient specifications are induced through inheritance, and the top types introduce the discriminants according to which constructions are categorized. These discriminants conform to notions shared across linguistic frameworks. The choice of ontology is in principle open, and the Test Daemons should be developed for all kinds of constructions.

The present design has been implemented at NTNU for two computational grammars, the Norwegian medium-size grammar *NorSource*, as mentioned above, and the much smaller grammar *Ga-gramm* for Ga (a language spoken in the Volta Basin area of West Africa), for both only in the verbal domain. Currently, for NorSource, there are two such Daemons, one representing syntactic valence and control/coreference properties, and the other in addition covering thematic roles and aspectual/Aktionsart properties (such as illustrated in fig. 1). The richer one comprises about 230 construction types, the leaner one about 170. Both inventories subsume only 'basic' patterns, that is, not passive constructions, and productive syntactic patterns of modification, wh-movement, subject-verb inversion and more are also abstracted away from. (In this respect, the construction inventory therefore has a straightforward connection to the verb lexicon, as this is standardly conceived in an HPSG or LFG grammar.) For Ga, only a set of 40 construction types has so far been encoded, based on the richer structure of the larger NorSource Daemon.

Each construction type is, apart from its type notion in the ontology, represented by one example sentence, serving as a leaf type in the ontology. The present ontology is stated in an LKB hierarchy (cf. Copestake 2002), as this system readily lends itself to the kinds of attribute paths often preferred by linguists.

This LKB hierarchy has been modelled as a small LKB-grammar, consisting of only one type of constructs, namely sentences formally modelled as multi-word lexical items. These sentences are identical to those entered as leaf types in the ontology. In this way, one is able to 'parse' the construction token, as a mock-sentence, or really, as a single constructional item. For example, for the sentence *she throws the ball* (now exemplifying with English), which is entered as a token of the type *ejct-tr* (cf. fig. 1), the 'construction entry' will have an identifier such as 'she_throws_the_ball', and in the orthographic specification, the string '<"she", "throws", "the", "ball">'; this string, thus, is defined as if it were a multi-word lexical item, from an LKB point of view. The Test Daemon can then, as a one-lexical-item parse, produce that string, with a feature structure being exactly the syntactic-semantic structure defined by the ontology for that type. Using the interface possibilities of an LKB grammar, one can thereby obtain a view of the properties of a given construction type as identified by its *construction name*

(through 'View Expanded Type') and through the feature structure exposed by the parse for the example sentence.

3 The *fulfillment* part

Technically the fulfillment part can be done in three ways. Common to all is that one defines a test suite for the parse grammar identical to (or overlapping with) the test suite designed for the Daemon.

Mode 1) Independent mode:
For each sentence, one verifies that the parse grammar and the Daemon separately produce feature structures (FSs) that correspond to each other in the respects focussed on. (Essentially, the Daemon FS will be a subpart of the FS produced by the parse grammar.)

Mode 2) Gently dependent mode:
The Daemon is integrated in the parse grammar, so that for each sentence parsed in the standard way, a parse is also displayed of the sentence-qua-construction, in the manner produced by the Daemon. (Thus, they can be viewed in parallel, in the same 'parse-forest'.)

For each sentence, one still needs to verify by hand that the systems separately produce feature structures that match in the respects focussed on.

Mode 3) Strongly dependent mode:
Again, the Daemon is integrated in the parse grammar, so that for each sentence parsed in the standard way, a parse is also displayed of the sentence-qua-construction, in the manner produced by the Daemon. However, in the FS produced by the parse grammar, the Daemon FS is replicated, with explicit declarations of how the information provided in the Daemon FS is reflected in the FS of the parse grammar.

We illustrate this mode with an edited excerpt from an FS of the parse grammar for Ga:

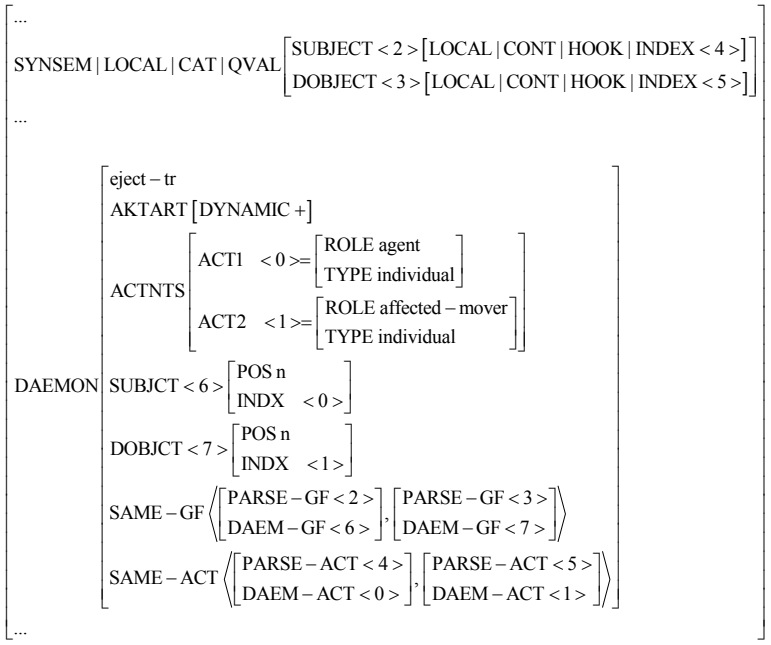


Fig.2 A view of a Mode 3 integration of a Daemon specification inside of a parse-grammar feature structure.

Its general feature architecture (derived from the HPSG Grammar Matrix, cf. Bender et al. 2002) includes the feature path `SYNSEM|LOCAL|CONT|HOOK|INDEX`, and in addition exposes some grammatical functions with dedicated attributes such as `SUBJECT` and `DOBJECT`, introduced by a feature `QVAL`. The counterparts of these features in the Daemon are `INDX`, `SUBJCT`, and `DOBJCT`; as attribute names in LKB can only be introduced with one unique type, and the types employed in the Daemon involve much less feature structure than those in a Matrix LKB grammar, both types and attributes must be distinct between the two systems. Hence the *correspondences* that one aims to expose can only be stated as relations, not by reentrancy, and these correspondences are given in the lists `SAME-GFs` and `SAME-ACTs` ('Same Grammatical Functions' and 'Same Actants', respectively).

The Norwegian LKB grammar so far uses only mode 1; the grammar for Ga is smaller, and thus allows more easily for the excercise, but its architecture of a mode 3 integration can in principle be generalized to all Matrix-based grammars. A technical desription of the integration between the parse grammar and the Daemon, combining general features and some specifics about the grammars employed, is given in the Appendix.

How the modes 2 and 3 might be implemented in other frameworks, or with other platforms, has so far not been explored.

In the next section we describe how the mode 3 version can be put to use as a plug-in-like mechanism in an LKB grammar, and after that we return to the structure of the Daemon and issues that presuppose no more than mode 1 of integration.

4 A plug-in possibility derived from mode 3

From the apparatus used in mode 3, a certain plug-in effect for, e.g., semantic specification can be derived. One can make the specification in the Daemon richer than the specification in the parse grammar, in such a way that when including the Daemon in the parse grammar, through unification, it makes the extra specification from the Daemon also part of the parse grammar FS. An example is the following: In the Ga grammar, the attribute `INDEX|SORT` is unspecified for value. In the integration file, the value of `INDX` from the Daemon specification, which is declared for `ROLE` and `TYPE`, can be reentrant with the value of `SORT` in the parse grammar. Thereby also the `INDEX|SORT` specification of the parse grammar will provide semantic `ROLE` and `TYPE` information. Fig. 3 illustrates the effect, superimposed on the constellation already shown in fig. 2, with the plug-in effect shown in the `SORT` values on the uppermost lines:

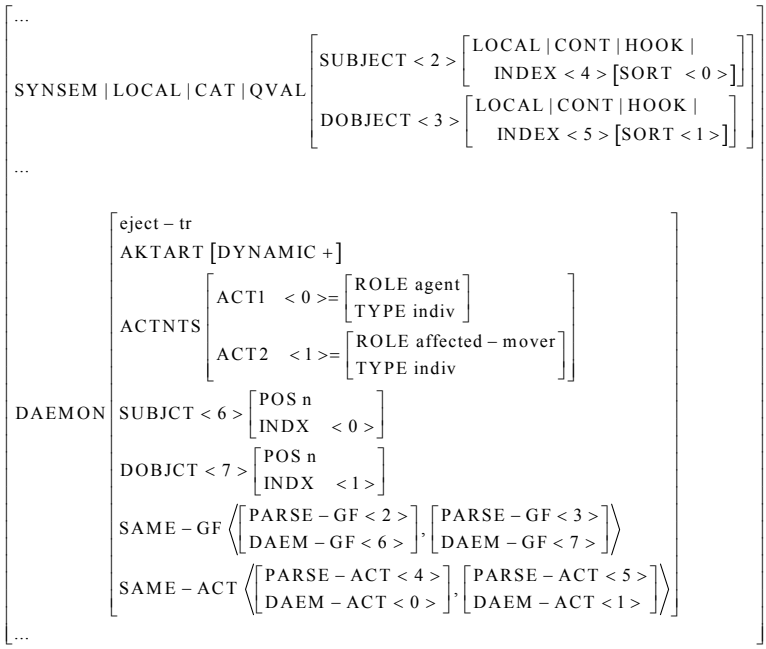


Fig.3 A view of a Mode 3 + plug-in integration of a Daemon specification inside of a parse-grammar feature structure, with the plug-in effect shown in the SORT values on the uppermost lines.

This way of inducing a richer semantic specification may be compared with, e.g., rewriting MRS representations, or connecting MRS representations with semantic ontologies. It is to be noted that this is a configuration the grammar writer can create solely inside the LKB, with no need for external components. (The exact steps needed are described in the Appendix.)

5 Properties of the Daemon

5.1 Attributes for grammatical functions

For the Norwegian Daemon, the following syntactic attributes representing grammatical functions are used:

`SUBJCT` - headed by a nominal constituent or consisting of a clause, and with either argument or non-argument status relative to the verb (see below)

DOBJCT ['direct object'] - headed by a nominal constituent or consisting of a clause, and with either argument or non-argument status relative to the verb

IOBJCT ['indirect object'] - headed by a nominal constituent, and with argument status relative to the verb

OBLIQUE - either headed by a pre- or postposition, or marked by a relevant NP-case, and with argument status relative to the verb; moreover, the governee of the adposition/case can in some cases be seen as having an indirect argument relation to the verb.

SECPRED ['predicative', in Jespersen's sense] - a 'secondary predicative' constituent, headed by any open class part of speech

EPON ['extraposition'] - a so-called 'extraposed' clause

IDENT ['identity term'] - the second argument in an identity predication

PRESENTED - the NP 'presented' in a presentational construction

PARTIKEL - an adverb- or particle-like element

The functions SUBJCT and DOBJCT are *unrestricted* in their values in the following ways: a) Their value can have either *argument* status relative to the verb they are functionally related to, or to another item; the latter applies to 'raised' subjects and objects, and is marked by the specification *'*SUBJCT *nonarg'/'*DOBJCT *nonarg'*; this corresponds to the situation where in an LFG f-structure, an item is entered outside the angled brackets of a PRED value (as in *PRED = 'seem < XCOMP > SUBJ'*) (cf., e.g., Butt et al.). Otherwise the specification is *'*SUBJCT *arg'/'*DOBJCT *arg'*. b) Their value can be either a 'full' item or an expletive noun, represented, resp., as *'*SUBJCT *full'/'*DOBJCT *full'* or *'*SUBJCT *expletive'/'*DOBJCT *expletive'*.

These dimensions cross-classify, in that a nonarg ('raised') item may be either full or expletive; specifying a position as 'expletive' in turn entails that this position does not carry a semantic role, but it does not entail 'nonarg', since it is not given that it has an argument role relative to another item. (The notion 'expletive' actually covers three cases, all encoded: *expl-pres* is the item introducing a presentational construction, *expl-epon* is the item correlated with an 'extraposed' clause, and *expl-absolute* is the item introducing an impersonal.)

For cases where the function OBLIQUE introduces a PP whose NP constituent has a specific relation to the verb or other constituents, the NP is exposed by the feature D-ARG (for 'dependent argument') and D-POS ('part-of-speech of dependent argument'); moreover, the NP is exposed in the semantic specification by the attribute ARG-OF-OBL, linked to the D-ARG of the functional feature (since the INDX of OBLIQUE is then not linked, this means that the PP as such is not a semantic argument, only its contained

NP). For those cases where the oblique PP has its whole specification contributing as an argument (as in the type *presentational-loc*, exemplified by *det sitter en katt i trappen* 'there sits a cat in the stairs'), the semantic argument role contributed by the full PP *i trappen* is represented as LOC, and linked to INDX of OBLIQUE; the construction at the same time illustrates the feature PRESENTED (see fig4, a and b):[3]

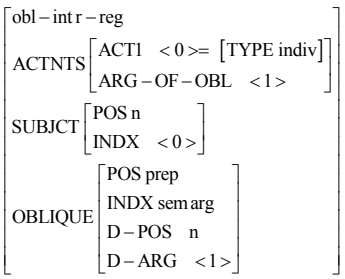


Fig 4a

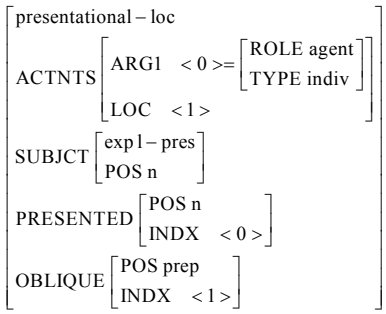


Fig 4b

The grammatical function SECPRED (for constituents carrying the predicational content in a secondary predication construction) is illustrated here by a construction with a causative semantic structure, *bøtta regner full* 'the bucket rains full' (= 'it rains (a situational ACT1) such that the bucket gets full (a situational ACT2)'), where the constituent *full* is the secondary predicate, and the reentrancy of '1' reflects the 'raising' structure of this construction:

[3] In the literature on Norwegian presentationals, the NP 'presented' is not uncommonly analyzed as a direct object, since its occupies a position much like that of direct objects (e.g., following the indirect object); due to its logical status, it is often also counted as a subject. Using the attribute PRESENTED is for descriptive convenience, and a freedom allowed by the Daemon purpose. A similar expedient is the use of the attribute EPON - that of 'extraposed' clauses and infinitives.

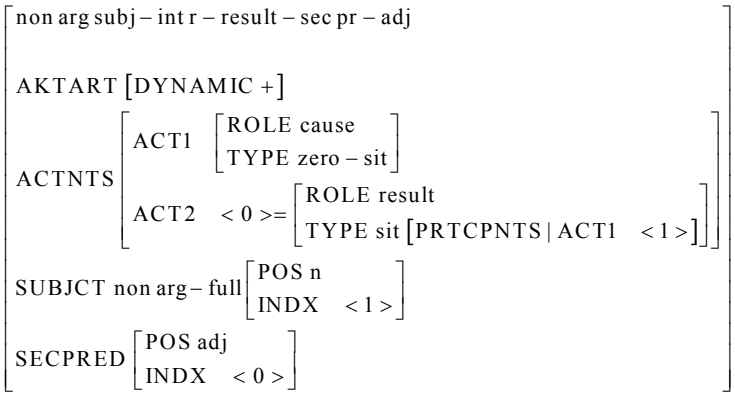


Fig. 5

This brief survey of some of the attributes used illustrates the following:

i) The analytic reasoning behind the inventory and value assignment in the FSs of a Daemon is no different from what applies in the building of a normal parse grammar.

ii) Attributes are not necessarily universal, but on the contrary can sometimes highlight properties particular to a language or family of languages; for instance, the use of `SECPRED` and `PRESENTED` reflect properties typical of Germanic languages.

iii) As a Daemon of NorSource, the attributes used do not necessarily match, attribute by attribute, those used in NorSource. For instance, although PREDIC in the NorSource grammar is equivalent to `SECPRED` as used in the Daemon, the feature PRESENTED corresponds to the specification

`...QVAL|DOBJECT|LOCAL|CAT|HEAD|PRESENTED +`

in NorSource. With the 'mode 3' of linking described in the previous section, such relationships are quite explicit, whereas with the modes 1 and 2, they have to be known. Still, in the latter case, the correspondences are not very many.

5.2 Principles of labelling construction types

A constructional type label should reflect the composition of the construction in as much detail as can conveniently be perceived when reading the label, and recalled when writing it. The more consistent and systematic the composition of the labels, the easier it is to reflect more content in them. In the three Daemons created so far - the syntactic and the syntactic-semantic Daemons for Norwegian, and the syntactic-semantic Daemon for Ga, the strategy of labelling differs among the three.

First, the *syntactic* labels for the first Norwegian Daemon are exemplified by the following samples: first in the label, *intr* or *tr* indicates degree of

transitivity, and it is then indicated if subject or object lacks argument status; then the presence of further constituents such as obliques or secondary predicates is signalled, and in the latter case its head category; and if the subject or object or the governee of a preposition is anything other than a regular NP, the category is indicated (like *obl-decl* meaning that the governee of a preposition is a declarative clause, or *tr-absinf-n* meaning that the subject is an absolute (arbitrary control) infinitive (and the object a normal NP), or *tr-raistoobj-bareinf* meaning that the object is 'raised' out of a succeding bare infinitive); all these pieces of information are connected by hyphens, and the resulting system is presumably within the limits of user-friendliness:

intr-n	gutten sover 'the boy sleeps'
intr-obl-decl	de snakker om at det er for sent 'they talk about that it is too late'
intr-nonargsubj-secpr-adj	kjelen koker varm 'the kettle boils hot'
tr-n-n	Kari sparker ballen 'Kari kicks the ball'
tr-equi-inf	Kari prøver å komme 'Kari tries to come'
tr-absinf-n	Å bygge høyhus interesserer Kari 'to build highrises interests Kari'
tr-raistoobj-bareinf	jeg hørte ham synge 'I heard him sing'
tr-nonargsubj-secpr-adj	han synes meg syk 'he seems me sick'
tr-nonargobj-secpr-adv	han sang sorgene bort 'he sang the sorrows away'
tr-epon-decl	det bekymrer meg at han kommer 'it worries me that he comes'

Table 1. *Some labels for syntactically identified construction types in Norwegian*

If one wants to mark semantic information in addition to syntactic specifications, the maneuvering space gets more strained. One has to choose whether to use 'global' labels to indicate something like *situation type*, paired with the same syntactic specification as before, or *role* indicators tied to each constituent (and possible aspectual specification in addition). In the Norwegian system, global situation labels are used, but with much inconsistency as to how much syntactic information is also supplied, and in which order syntactic and semantic information is given; the assembly below is representative (in parenthesis behind the construction label is given a

situation type label, reflecting the semantic type intersecting with the syntactic type - cf. next sub-section):

nonargsubj-intr-result-secpr-adj (zerocause-event-causation)
  koppen renner full - 'the cup runs full' - "the cup fills up"
raistosubj-intr-ascrpt-secpr-adj (unary-sit-ascription)
  gutten virker syk - 'the boy seems sick'
path-endpnt-tr (path-endpnt)
  stien når toppen - 'the path reaches the top'
weight-tr (weight-sit)
  stenen veier 5 kg - 'the stone weighs 5 kg'
tr-exp-raistosubj-inf (sit-exper-sit)
  han synes meg å komme - 'he seems me to come'

Table 2. *Some labels for syntactically and semantically identified construction types in Norwegian*

In the Ga system, more consistency is achieved, as is seen below; here global syntactic information comes first, then role specification with the roles in the order of the constituents, and then with capital letters a global semantic characterization. While the latter might seem redundant given the role specifications, for some readers they may be informative, since the compositionality in many of the Ga constructions is not what most directly comes to mind from a European language perspective. The syntactic specification includes information as to whether the 'logical' actant is embedded in a postpositional NP, a counterpart to 'oblique', and possible identity with other constituents; *unifobj* stands for 'inherent complement' of the type discussed in Essegbey 1999. The last example is a serial verb construction, indicated by *sv-*, and by a succession of two verbal constructions, initiated as *vtr-* and *vditr-*, respectively, each having its internal roles indicated; the * attached to a role means that the role-bearing actant is repeated in the second VP, either just understood, or as a pronominal prefix, specified as **PRONPREF*. (In the feature structures for these constructions, some specifications will be particular to West African languages, as some of those discussed in 5.1 are to Germanic languages.) Here are some examples:

v-intr-postpsubj-locus-PROPTY	Nsɛɔ lɛ mli jɔɔ sea the inside cool-HAB 'The sea is cool'
v-tr-mover_endpt-MOTION	Kofi ba biɛ Kofi came here

v-tr-partwhlsubj_idobj-locth_exper-SENS	
	o-he jɔɔ bo
	your-self rest you
	'you are at ease, relaxed'
v-ditr-unifobj2-agsens_locus_materialzr-PERCPT	
	wɔ-bo lɛ toi
	we-listen him ear
	'we listened to him'
sv-vtr_ag*PRONPREF_th*-vditr_endpnt-PLACEMENT	
	wɔ-tsi amɛ wɔ-gbee shi
	we-push them we-fell [them] down

Table 3. *Some labels for syntactically and semantically identified construction type in Ga*

On layout consistency grounds, the first (for syntactic annotation) and third system seem preferable. Which one to choose among these of course depends on what kind of information the parse grammar in question exposes (or is made to expose).

Although the matter of labelling may seem theoretically insignificant, for the building of a test suite, an instructive label system is quite helpful, and far easier to relate to than feature structures of the types illustrated earlier.

Such a labelling design may also be helpful in the building of a construction inventory not (yet) linked to a computational grammar, both by its potential for enhancing clarity of organization, and for linking up to feature structures of the type provided by the Daemon. Before a consolidated system of labelling can be considered, the usefulness of candidate systems outside of the domain of grammar engineering test suites would thus also be important.

5.3 Structure of the ontology

The ontology is organized into one syntactic and one semantic part, with construction types inheriting from both sides. Below is an example of how a syntactic and a semantic specification are joined to define a constructional type, for *nonargsubj-intr-result-secpr-adj*, illustrated by *bøtta regner full* 'the bucket rains full' (= 'it rains such that the bucket gets full', cf. fig. 5 above):

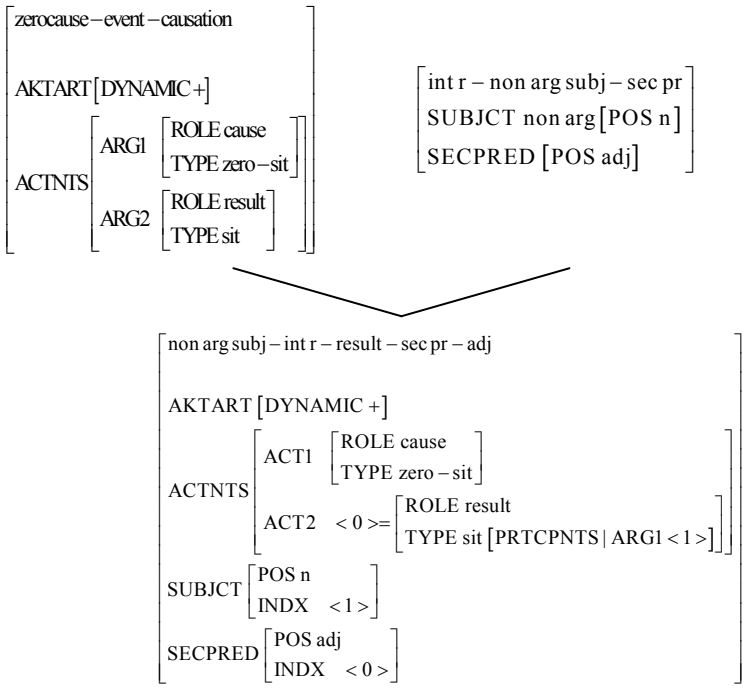


Fig. 6 Example of constructional type inheritance

In the same vein, the following is an approximate view of some of the top level syntactic types in the Norwegian system, and some of the semantic ones, with one example of the stitching together of the syntactic and the semantic side:

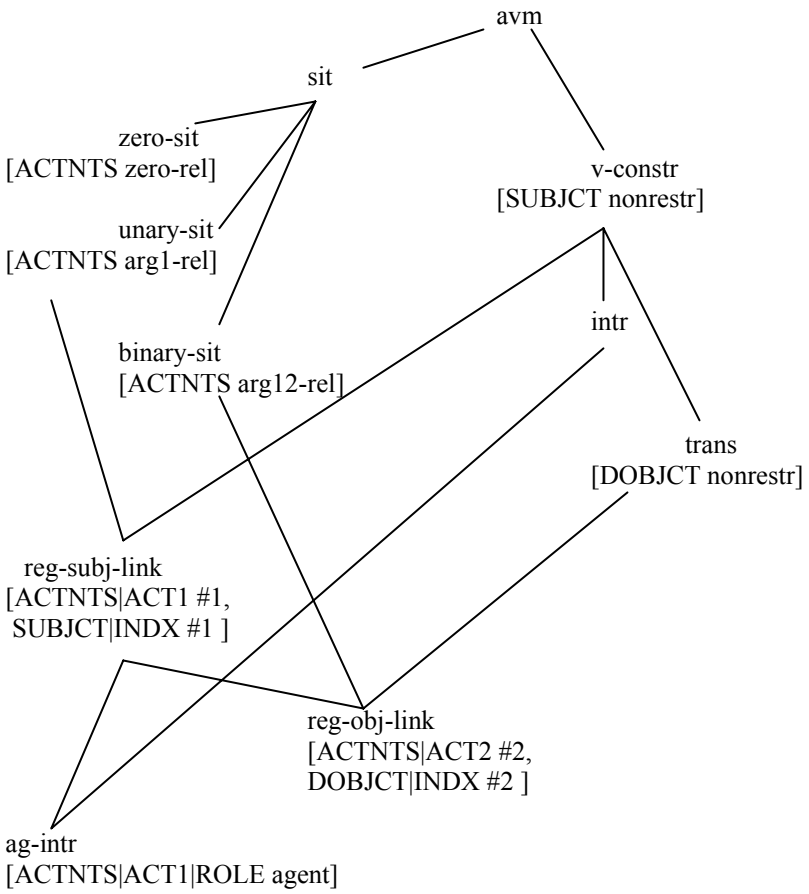


Fig. 7 Partial view of constructional type inheritance

With the ontology populated by more languages, the syntactic part of the system would most likely have to grow, while the semantic part (apart from improvements of the system as such) in principle might remain constant. The design might then make it possible to view, for any identified situation type, which construction types in various languages embody that situation type, and likewise for going in from a given syntactic specification to see construction types across languages supplying different situation types.

In order for a construction ontology of this kind to be implemented for grammar engineering frameworks less likely to employ an LKB system, it will be an interesting question to see to what extent the present ontology can be ported, e.g., to OWL.

6 Summary

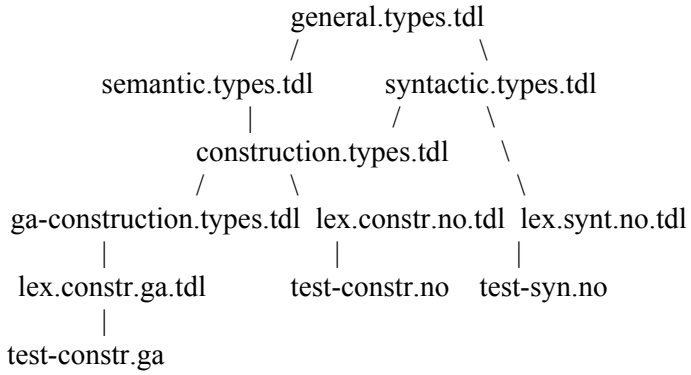
In order to expose the content of grammars for evaluation across frameworks, very much of their formalism and terminology has to be abstracted away from, in order to attain a common ground of comparison. A requirement on such a common ground is that it is still identifiably sound according to both theoretical and empirical criteria of linguistics. If such a ground can be found, in turn, it is helpful not only for the construction of grammars and the evaluation and development of the various frameworks, but it may also help mediate the contribution of computational grammars to a wider linguistic community.

In the system presented above, the concern of designing a common ground has been combined with a strategy of creating systematic test suites for use in individual grammars. This strategy, in turn, tries to improve classification of construction types in a way that may be useful for linguists outside the computational community, not the least for purposes of grammar documentation.

An attempt is made to embed the system in an actual mechanism, which on the one hand exposes an ontology of verbal constructions, and on the other can be integrated in a computational grammar so as to 'witness' how the specifications produced by a parse relate to declared aims phrased in terms of a 'common ground' analytic language. The mechanism is light-weight, and is only a means by which one can *start* becoming more systematic about evaluation in analytic respects. As the mechanism is built on the LKB platform, it - especially in the latter respect - is confined mainly to HPSG-based grammars. However, these two functions can clearly be split, and hopefully the present system may serve as a partial starting point for the construction of systems based on different platforms.

Appendix

Below is a sketch of the structure of the Daemon as realized by an LKB system. It consists of the directory 'constructions', which, in addition to the lkb folder and roots.tdl, has the following files, here displayed so as to reflect which files depend on which:



Purely syntactic aspects of the construction specifications are defined in syntactic.types.tdl, purely semantic aspects in semantic.types.tdl, and combined constructional specifications (combining types from the former two) are defined in construction.types.tdl. In addition, general.types.tdl states over-arching general types. For Ga, there is also a type file ga-construction.types.tdl deriving new types from construction.types.tdl.

In the lexical files, each 'lexical item' is a full sentence, corresponding to the items in the test files. For Norwegian there are two test files, corresponding to the lexicon files lex.synt.no.tdl and lex.constr.no.tdl: the former has types reflecting only the syntactic specification, the latter has types reflecting a full constructional specification. For Ga, there is only one test file, representing full constructional specification.

To view the full construction hierarchy, enter 'sign' in the LKB Top View window. To view either the syntactic part of the construction hierarchy exclusively, or the situation hierarchy exclusively, comment out construction.types.tdl (and ga-construction.types.tdl) in the script file (in this case, also comment out lex.constr.no.tdl and lex.constr.ga.tdl), and then, to view the syntactic construction hierarchy, enter 'syncons' in the View window, and to view the situation hierarchy, enter 'sit' in the View window. Whichever type hierarchy is displayed, to see the feature specification of the type, do normal leftclick on the type label and view 'Expanded type'.

Exemplifying sentences can also be viewed. From any of the test-files, any sentence can be selected and entered in the LKB Top parse window (or '(do-parse-tty "...")' in the commonlisp buffer), and a minimal parse tree emerges, rooted by 'constr' (or '?').[4] When clicking for 'Enlarged tree' and then 'Feature structure', one sees the type and feature structure specification (the latter being the same as what one sees for that type on the previous

[4] When, for Norwegian, a given sentence appears in both lexicons, two parses are displayed, the upper one rendering the full construction specification, the lower one the syntactic part of that specification.

view).

To create the 'fulfillment' effects described in section 3:
Mode 1 requires no further steps than the creation of a common test suite.
Mode 2 is realized in the following way:

a. In the parse grammar, define a subdirectory 'constructions', populated by the type files and the relevant lexicon(s) of the language chosen, except the file general.types.tdl, since its definitions are already covered in the parse grammar.
b. To secure that each attribute is introduced with a unique type, the attribute names in the Daemon will be largely distinct from those used in the parse grammar, since in the Daemon, information is much less complex than in the parse grammar.
c. Two root values are defined in the parse grammar, one being the root category of the parse grammar, and one the root category of the Daemon.

Mode 3 requires the same steps as mode 2, and in addition:

d. In the type declaration for 'sign', add a feature 'DAEMON' with value 'sign-min', which will take as value the Daemon FSs plus correspondence declarations.
e. In the parse grammar, for those verb lexeme types for which one wants to display the Daemon correspondence, introduce a type file defining subtypes of these types which (i) provide a slot for the Daemon FS, and (ii) state exactly which feature paths in the parse FS provide counterparts of the relevant values in the Daemon FS.
f. Corresponding to these 'Daemon'-related lexeme types, introduce a lexicon file with the verbs used in the shared test file, now defined as items of the 'Daemon'-related types.

Mode 3 has been realized for a small parse grammar for Ga, based on the HPSG Grammar Matrix. Here, step e. is implemented through the file 'ga-daem.tdl', and step f. through the file 'lexicon-ga.tdl'. Thus, in the 'Ga grammar/lkb/script' file, the following lines are active on mode 3,

```
(lkb-pathname (parent-directory) "ga-daem.tdl")
(lkb-pathname (parent-directory) "constructions/syntactic.types.tdl")
(lkb-pathname (parent-directory) "constructions/semantic.types.tdl")
(lkb-pathname (parent-directory) "constructions/construction.types.tdl")
(lkb-pathname (parent-directory) "constructions/ga-construction.types.tdl")
    ....
(lkb-pathname (parent-directory) "lexicon-daem.tdl")
(lkb-pathname (parent-directory) "constructions/lex.constr.ga.tdl")
```

and are commented out to return the grammar to its mode 1 operation (then also deleting the extra 'Daemon' line in roots.tdl). To go from mode 3 to mode 2, only ga-daem.tdl and lexicon-ga.tdl are commented out (leaving roots.tdl with both root definitions).

References

Bender, Emily M., Dan Flickinger, and Stephan Oepen. 2002. The Grammar Matrix: An open-source starterkit 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 Esther Kropp, 2004: Ga clauses without syntactic subjects. *Journal of African Languages and Linguistics* 25.1: 1-40.

Essegbey, James. 1999. Inherent Complement Verbs Revisited. MPI Series in Psycholinguistics, Nijmegen.

Goldberg, Anne. *Constructions. A Construction Grammar Approach to Argument Structure*. Chicago University Press, Chicago.

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

Hellan, Lars and Dorothee Beermann. 2005. Classification of Prepositional Senses for Deep Grammar Applications. In Kordoni, Valia and Aline Villavicencio (eds) *Proceedings of the Second ACL-SIGSEM Workshop on The Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications.* University of Essex.

Sag, Ivan A., Thomas Wasow and Emily Bender. 2003. *Syntactic Theory. A Formal Introduction*. CSLI Publications, Stanford.

Overlay Mechanisms for Multi-level Deep Processing Applications

Tracy Holloway King and John T. Maxwell III
PARC

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

Deep grammars that include tokenization, morphology, syntax, and semantic layers have obtained broad coverage in conjunction with high efficiency. This allows them to play a crucial role in applications. However, these grammars are often developed as a general purpose grammar, expecting "standard" input, and have to be specialized for the application domain. This paper discusses some engineering tools that are used in the XLE grammar development platform to allow for domain specialization. It provides examples of techniques used to allow specialization via overlay grammars at the level of tokenization, morphology, syntax, the lexicon, and semantics. As an example, the paper focuses on the use of the broad coverage, general purpose ParGram English grammar and semantics in the context of an Intelligent Document Security Solutions (IDSS) system. Within this system, the grammar is used to automatically identify sensitive entities and relations among entities, which can then be redacted to protect the content.

1 Introduction

Deep grammars that include tokenization, morphology, syntax, and semantic layers have obtained broad coverage in conjunction with high efficiency (e.g., Kaplan et al., 2004b). This allows them to play a crucial role in applications. Sometimes grammars are developed exclusively for a given application in a given domain. However, a grammar is often developed as a general purpose grammar, expecting "standard" input, and has to be specialized for the application domain. This is done, for example, in MedSLT which is a speech translation system built on top of the Regulus platform (Rayner et al., 2006; Chatzichrisafis et al., 2006).

This paper discusses engineering tools that are used in the XLE grammar development platform (Maxwell and Kaplan, 1996; Crouch et al., 2007) to allow for the domain specialization necessary for applications. Some of the techniques used are similar to those developed for building parallel cross-linguistic grammars (Bender et al., 2002; Butt et al., 2002) but many of them are more fine-grained and involve components that are unlikely to be shared across languages. The focus of this paper is not on how to determine which components to specialize, but instead on what tools have proven useful in allowing the specializations required by the grammar engineers. As an example, the paper focuses on the use of the broad coverage, general purpose ParGram English grammar and semantics in the context of an Intelligent Document Security Solutions (IDSS) system. Within this system, the grammar is used to automatically identify sensitive entities and relations among entities, which can then be redacted via mechanisms such as encryption in order to protect the content.

[†]The IDSS portions of this work were supported in part by Xerox Corporation. We thank the audience of GEAF2007 for comments on the presented version of this paper, and Eric Bier and Jessica Staddon for input on the IDSS application description.

1.1 The IDSS Application

The IDSS application takes document collections, helps a knowledge worker find sensitive entities and relations among entities, and then provides the user to choose mechanisms to protect these entities, including encrypting them so that these sensitive items are only available to those with appropriate keys. The application can be used, for example, to redact documents with sensitive material in them. The documents can simply be printed or produced as a pdf file with the redacted material "black boxed". However, the availability of fine-grained encryption in conjunction with detailed entity and relation analysis allows for documents to be created where each entity type is tied to a particular encryption key. Different end users will have different keys and hence be able to view different parts of the same redacted document. For example, with mortgage documents, some users could see phone number, name, and address information, while others might have access to social security numbers and financial information.

A deep grammar is used to provide an initial list of entities and of relations among entities that the knowledge worker might be interested in. This component is discussed in detail in this paper. There are two other major system components. One is a user interface called Entity Workspace (Bier et al., 2006) which is used to manipulate the document collection and the entities and relations, including adding sensitive entities and relations that were missed by the initial automatic extraction. This component also allows the specification of how much to redact: entities can be redacted at the entity level, the sentence level (any sentence with a sensitive entity is redacted), or the paragraph level (and paragraph with a sensitive entity is redacted).

The second major component is the encryption system that is used to redact entities and relations. This system not only provides the encryption of the sensitive entities, but also allows for fine-grained specification of who can decrypt which sections of the document. This ability to do selective encryption/decription is important in an increasingly electronic workplace where documents are passed from user to user without being printed and where different users of the same document may have much different information needs and rights.

To return to the automatic extraction of entities and relations, as an example, a sentence like (1a) or (1b) would yield the list of facts in (2a) or (2b). These facts are identical except for some byte position information and the word facts for *work* and *employ*. Each fact is designated as being an entity, a relation between entities, or a content word. Entities and relations are typed (e.g. *person*, *location*, *works-for*). Entities can occur with lists of alternative realizations or aliases (e.g. *[Robin, Abramov, Robin Abramov]* in (2a)). Content words can occur with a list of synonyms (e.g. *[hire, use]* in (2b)). Facts are associated with sentence numbers within the document and with byte position of the entity within the sentence.

(1) a. Robin Abramov works for International Business Machines.

b. Robin Abramov is employed by International Business Machines.

(2) a. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(25), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(25), [company])
WORD(work, sent_num(1), byte_position(15), [work, influence, make, cultivate, shape, bring, function, knead, exploit, solve, ferment, sour, exercise])
sentence_num(1)

b. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(27), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(27), [company])
WORD(employ, sent_num(1), byte_position(15), [hire, use])
sentence_num(1)

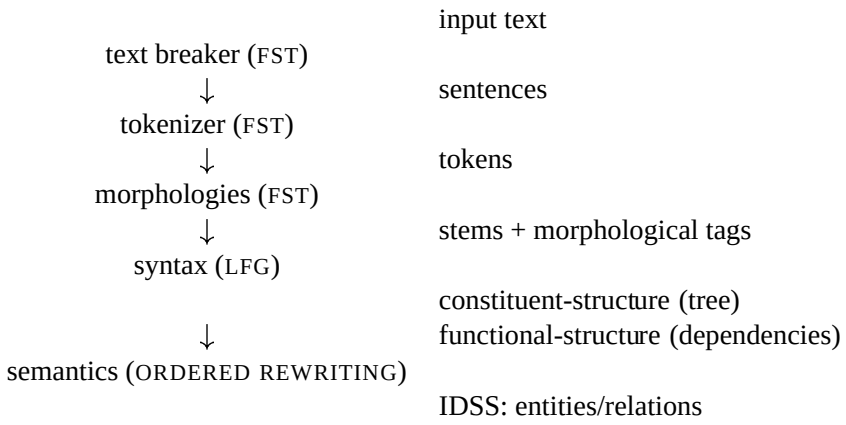
The details of these representations and how they are produced in the IDSS application are discussed in more detail later in the paper.

1.2 The IDSS Natural Language Component

The general XLE parsing pipeline used in the IDSS system is shown in (3).[1]

[1]There is a Makefile which produces a run-time version of the entire pipeline. This depends on XLE's release-grammar mechanisms that allow a single-directory version to be created and frozen for export into the run-time application. In addition to putting the grammar files in a single directory, the release version can include version number information and encrypts the lexicon files (the grammar files themselves are not encrypted).

(3) **XLE Grammar Processing**



This paper focusses on how application-specific extensions were made to the core pipeline, which is used in several different applications and research projects.

Extensions to the finite-state morphologies were needed to allow for additional entities, to the lexicon for Arabic and Russian names, and to the grammar for unusual punctuation and lists. The semantics was extended to pick up the additional entities, to find entities and entity relations, and to delete all other semantic facts. In all cases, we want to ensure that upgrades to the base system can be included in the IDSS application without losing any of the application-specific specialization. We achieve this by allowing for overlay systems at each level. The tools for these overlays, along with examples of how they are applied, are described in this paper.

2 Tokenizers and Morphologies

In the XLE grammars, there is a configuration file for the text breaker, tokenizers, and morphologies. The file specifies which text breaker, tokenizer, and morphology are used by the grammar. When there is more than one tokenizer or morphology, the configuration file specifies how they are combined, e.g., the morphology for recognizing phone numbers may take precendence over the general English morphology which in turn takes precedence over the guesser. The morphology configuration file, called the morphconfig, is called by the syntactic grammar. The XLE ParGram English grammar uses finite-state (FST) text breakers, tokenizers, and morphologies (Kaplan et al., 2004a; Beesley and Karttunen, 2003); these are described in this section.

The input string is first run through the text breaker. The text breaker deterministically breaks the text into sentences. It is a high-precision text breaker: if it is unsure whether something represents a sentence boundary, it will put in a mark (+SB) instead of forcing a sentence break. This way the grammar can be used to provide further information in complex cases. Such cases can occur, for example, when the string *Dr.* appears followed by a form that could be either a common or a proper noun (e.g. *Dr. Bush*); in such cases the text breaker cannot determine whether the

Dr. is a sentence final abbreviation for *Drive* or a sentence internal abbreviation for the title *Doctor*. If this uncertainty is marked and passed through to the syntax, syntactic knowledge can be used to determine whether there should be one sentence or two.

After textbreaking, the tokenizer non-deterministically breaks the string into tokens.[2] The tokenized string is run through an industrial morphology produced by Inxight which in the parsing direction converts inflected forms into lemmas and a set of morphological tags (4a). This morphology covers many proper names, (4b, c), as well as the inflected forms of common nouns, verbs, adjectives, etc.

(4) a. hunts $\longrightarrow$ hunt +Noun +Pl | hunt +Verb +3sg

b. Robin $\longrightarrow$ Robin +Prop +Giv +Fem +Sg | Robin +Prop +Giv +Masc +Sg

c. Detroit $\longrightarrow$ Detroit +Prop +Place +City

In addition, the tokens are run through a set of specialized FSTs to recognise times and dates (5a) and to convert spelled out numbers into digits (5b).

(5) a. April 23rd $\longrightarrow$ month(4) day(23)

b. twenty-four $\longrightarrow$ 24

Items unrecognized by the morphology or one of the specialized FSTs are run through a guesser that uses clues such as capitalization or string ending (e.g. *ing*, *s*) to posit part of speech and other morphological tags. The guesser is currently quite simplistic. An example is shown in (6).

(6) fooing $\longrightarrow$ foo +Noun +VProg +Sg +Guessed
| +Verb +Prog +Guessed
| +Adj +VProg +Guessed

Applications often require special entity recognizers to either supplement or override the morphology. The morphconfig file allows additional FST machines to be called either in an override (USEFIRST) or a supplemental (USEALL) capacity. The override is used when only the analyses in that FST are to be used. For example, the FST that recognizes phone numbers could override any analyses that would recognize the same string as a range of numbers. The supplemental version is used to add in additional analyses. For example, the FST that recognizes years as dates (e.g., *They left in 2000.*) is used in a supplemental capacity in order to allow the analysis of these digits as regular numbers (e.g., *They bought 2000 boxes.*).

[2]The IDSS application did not involve extensions to the tokenizer since the texts parsed followed standard written English punctuation conventions. Other overlays, such as the header/title grammar for parsing technical manuals and web pages, where much of the input has initial upper case or all upper case letters, do use different tokenizer versions.

For IDSS, an FST was added to recognize phone numbers, addresses, and social security numbers. These are provided with unique tags (i.e., *+PhoneNumber*, *+Address*, *+SSNumber*); the syntax and semantics were then extended to recognize these tags and form nouns based on the forms with the tags. An example of the output of the stages of the system for a phone number are shown in (7). The *+PhoneNumber* tag provides the *NE-TYPE phone* feature in the syntax which in turn provides the *ENTITY(. . ., phone, . . .)* feature in the semantics. These all key off of the specialized output of the IDSS FST.

(7) a. Input: They called 123-4567.

b. Tokenizer/morphology: 123-4567 +PhoneNumber +Sg +PreferMorph

c. Syntax:

$$\begin{bmatrix} \text{PRED} & \text{'123-4567'} \\ \text{NTYPE} & \begin{bmatrix} \text{NSYN} & \text{common} \end{bmatrix} \\ \text{NE-TYPE} & \text{phone} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 3 \end{bmatrix}$$

d. Semantics:
ENTITY(123-4567, phone, sent_num(1), byte_position(13))
WORD(123-4567, sent_num(1), byte_position(13), [entity])

The IDSS entity FST was given priority over other FSTs so that only the special named entity analyses would surface. An additional guesser was created to hypothesize certain common person names for nationalities that the standard morphology did not have lists for, namely Arabic and Russian last names; as will be seen in the next section, the grammar was also supplemented with lexicons for the more common of these names. The lexical entries for names provide additional information such as gender and are given higher confidence ratings relative to purely guessed names.

Since the morphology configuration calls both the FSTs for the base grammar and those for the IDSS grammar, any improvements to the base grammar FSTs (e.g., a new time-date FST) can be incorporated into the system by a version update to those files. The morphology configuration allows relative path names so that the FSTs do not need to be copied into the IDSS grammar directory but instead can automatically reference the current version of the base grammar FSTs.

Although not used in the IDSS overlay, XLE also has a command that allows tokenizers to be pushed onto the front of the transducer stack (or popped off of it). That is, the grammar is loaded with the tokenizers specified in the grammar morphconfig, but then an additional tokenizer is run before the ones in the grammar.

This can be used, for example, to have a FST that does spelling correction or named entity markup apply before the regular grammar.[3]

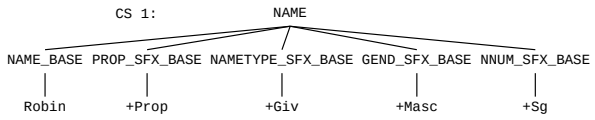
3 Syntax

The output of the tokenizers and morphologies serves as input to the syntax, forming the leaves of a syntactic tree structure. The output of the syntax is a pairing of trees (referred to as c(onstituent)-structures) and dependencies in an attribute value matrix (referred to as f(unctional)-structures). The structures for the sentence in (8a) are shown in (8b,c). The c-structure and f-structure categories are relatively detailed in comparison to most theoretical LFG descriptions (Dalrymple, 2001). XLE's computational approach to syntax and semantics manages ambiguity by combining alternative interpretations into a single packed structure that can be further processed without the typically exponential cost of unpacking (e.g., Robin as a man's name and as a woman's). The XLE syntax and semantics use the same packing mechanism (Maxwell and Kaplan, 1991; Crouch, 2005b).[4]

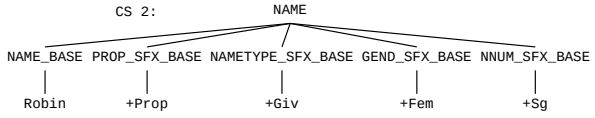
[3]If there is only one grammar being used by the system, then modifying the tokenizer FST via an overlay morphology configuration can be done. However, if multiple versions of the grammar are being run (e.g., one for headers and one for regular text), then using the pop/push facility can save space compared to having two grammars loaded.

[4]There are, in fact, two c-structures for (8a) which differ at the sublexical level due to the two analyses for *Robin* related to the two morphological analyses shown in (4b). Since the display in (8c) does not show the sublexical structure, the difference between the trees is not visible. The two different sublexical trees are shown in (i).

(i) a.

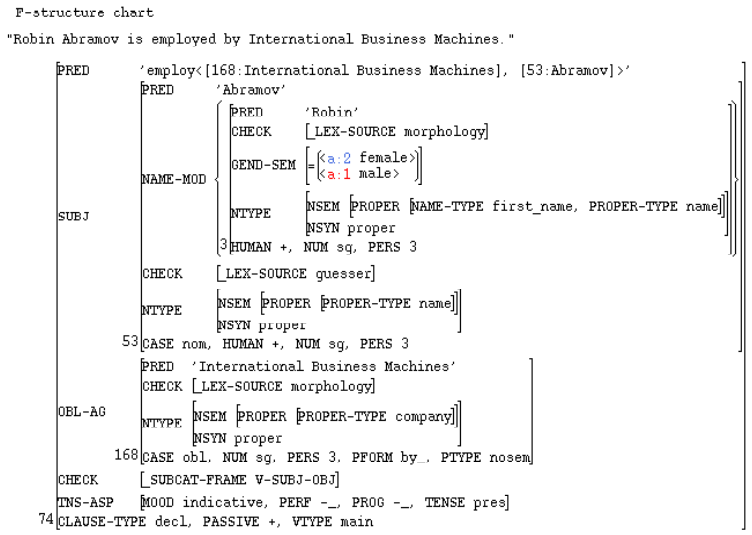


b.

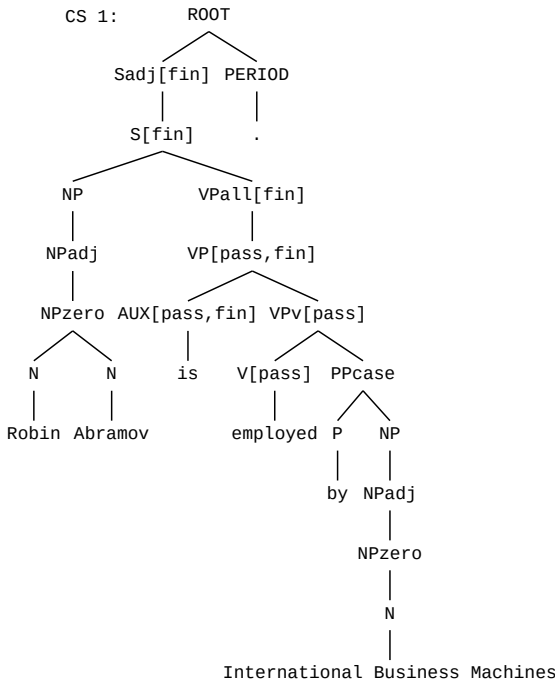


(8) a. Robin Abramov is employed by International Business Machines.

b.



c.



The syntax comprises a configuration file, lexicons, and LFG annotated phrase

structure rules. The lexicons and phrase structure rules can call grammar-defined templates. The configuration file, in conjunction with complex lexicon edit entries (Kaplan and Newman, 1997), is what allows for overlay grammars (Kaplan et al., 2002). The configuration file states which lexicon, rule, template, and system parameter files are used by the grammar. It also states the priority order of these so that application-specific changes take precedence over the more general rules. In addition, a configuration file can state that it is identical to another configuration file except for any stated changes. Such inheritances can be deeply nested, although in practice for this application they only go three levels deep with the standard English grammar as the ultimate base, as is described below for IDSS.

The IDSS syntax overlay is relatively simple; more complex overlays are required for applications used with "non-standard" English, including unedited English or the English used in emails. The IDSS English calls the Aquaint grammar (Bobrow et al., 2005, 2007) which the semantics generally assumes as its input; the Aquaint grammar in turn calls the standard English grammar configuration as its base.[5] In addition, the IDSS grammar calls:

- two lexicon files (one for ∼1900 Arabic names and one for ∼2300 Russian names)
- the morphology configuration as described in the previous section
- a rule file with two sublexical rules for the phone and address entities and a modified version of the sentence final punctuation rule

The configuration file also calls a system parameter file which uses OT mark rankings to effectively remove some unused rules in the standard grammar for efficiency and coverage reasons (e.g., topicalization, initial vocatives) and to set time, memory, and processing limitations for the IDSS system.[6] The IDSS configuration file is shown in (9).

(9) AQUAINT ENGLISH CONFIG (1.0)

```
BASECONFIGFILE ../english-aquaint.lfg
PERFORMANCEVARSFILE
        +idss-performance-vars.txt.
MORPHOLOGY (IDSS ENGLISH).
RULES (STANDARD ENGLISH)
        (AQUAINT ENGLISH)
        (IDSS ENGLISH).
```

[5]Over time, more general solutions are integrated into the standard grammar. Currently, the main overlay in the Aquaint grammar is for certain types of coreference markup used in anaphora resolution.

[6]These could be defined via XLE commands when the system is loaded. However, by including them in the grammar, it is easier to ensure that they are always loaded and always set to the same values. These values can be overridden on the XLE command line to allow for experimentation.

```
FILES +eng-lex-arabic-names.lfg
      +eng-lex-russian-names.lfg
      +english-idss-morphconfig.lfg
      +english-idss-rules.lfg
```

3.1 Overlay Rules

The relative simplicity of the IDSS overlay grammar is due both to the design of the configuration file which allows inheritance and fine-grained modification and to the design of the syntax rules which are divided into subrules to allow for substitution in overlay grammars. The sublexical rules, e.g., the rules used to compose verbs and nouns from combinations of stems and morphological tags (Kaplan et al., 2004a), and the root level rules are particularly finely divided because most applications have required some overlay to these rules. For example, corpora for different applications differ widely as to the type of punctuation allowed sentence finally. As such, there is a rule ROOT-DECL-PUNCT which states the punction options for matrix (root) declarative clauses. In the IDSS grammar, this is redefined to allow colons and, dispreferedly, nothing (as represented in (10) by *e*), in addition to the usual period and exclamation point.

(10) ROOT-DECL-PUNCT –>

```
{   PERIOD
|   EXCL-POINT
|   COLON
|   e: @(OT-MARK NoFinalPunct)
}.
```

This situation highlights the fact that having the proper system tools for overlay grammars is not enough: the grammar developer must design the grammar itself in anticipation of its modification for applications. Fortunately, any changes in modularity to the base grammar benefit all overlay grammars and future applications, and often such changes, such as increased subdivision of rules, are simple to implement. At this point, such subdivisions rarely have to be made; when the standard grammar was first used with overlays, approximately twenty rules were refactored.

3.2 Lexicons

The IDSS grammar calls two lexicon files (one for ∼1900 Arabic names and one for ∼2300 Russian names). These provide information that the forms are person names and indicate whether they are family or given names. When the given name is known as a woman's or a man's name, this information is also included (cf. the discussion of the morphology associated with the English name *Robin* in (4)).

Overlay lexicons can be more complicated. New entries can be added for any part of speech. In addition, entries that exist in the standard grammar can be: (1)

removed, (2) replaced, or (3) altered. This is controlled not just at the level of the stem but also the part of speech and even the possible entries associated with each of these. For example, if the standard grammar had the entry for *push* as in (11a), the overlay grammar could have an enty as in (11b) which would produce the effective entry as in (11c) where the two entries have been merged.

(11) a. push V XLE @(V-SUBJ-OBJ push); ETC.

b. push +V XLE @(V-SUBJ push); ETC.

c. push V XLE { @(V-SUBJ-OBJ push) | @(V-SUBJ push) }; ETC.

The mechanism for lexical edit entries is introduced in Kaplan and Newman (1997) and the current state is described in the XLE documentation (Crouch et al., 2007).

3.3 Performance Variables

The IDSS grammar configuration also calls a system parameter file which effectively removes some unused rules in the standard grammar for efficiency and coverage reasons and sets time, memory, and processing limitations for the IDSS system to allow for effective parsing of large document collections.

The ability to remove and rerank rules takes advantage of the Optimality Theory (OT) mechanism in the XLE system (Frank et al., 2001). The XLE OT system is inspired by theoretical OT (Prince and Smolensky, 1993) but differs from it in crucial respects: in XLE, rules do not need to be ranked, preference as well as dispreference marks are available, and special status marks exist for allowing multiple pass grammars and for declaring rules NOGOOD. Parts of the grammar and lexicon associated with NOGOOD marks are removed from the compiled system.[7] The ability to declare a given OT mark NOGOOD is extremely useful in overlay grammars because both whole rules and specific disjuncts within them can be removed from the grammar in this way. Consider the made-up simple rule in (12).

```
(12)  S –>   (NP:   (^  TOPIC)=!
                    (^  TOPIC)=(^  XCOMP* OBJ)
                    @(OT-MARK TopicMark))
             NP:    (^  SUBJ)=!
             VP:    ^  =!
```

The rule states that an S can consist of an optional NP which will be the topic which also serves as the object somewhere in the structure (e.g. *Bagels, I like.*, *Grammars, I want to write.*), an obligatory NP subject, and a VP that heads the S. The NP topic annotations an OT mark called TopicMark. In the standard grammar, this mark is dispreferred, and so topics will surface only when no more preferrable analysis is

[7] This contrasts with theoretical OT in which constraints can be very lowly ranked but are always violable. NOGOODs could be thought of as inviolable constraints.

possible. However, in many overlay grammars used in applications including IDSS, this mark is declared NOGOOD via the statement in (13) in the overlay performance variables file.

(13) set-OT-rank TopicMark NOGOOD

This effectively creates the rule in (14) without having had to alter the one in (12) in the standard grammar.

(14) S –> NP: (^ SUBJ)=!
VP: ^ =!

The OT marks can similarly be used in the template space to alter the effective behavior of the template. This is often used to control how dispreferred mismatched subject-verb agreement is. In the standard grammar, the OT mark NoVAgr is heavily dispreferred because the grammar expects edited standard written English. However, when used in less formal domains, such as emails, this mark is only slightly dispreferred. This reranking is done in the performance variables file and hence the templates and rules themselves do not need to be altered or have explicit overlay versions.

4 Semantics

The semantics for the ParGram English grammar is written using XLE's ordered rewrite system, referred to as XFR. It takes the f-structure output of the syntax and converts it to a flattened, normalized, skolemized form (Crouch and King, 2006). The output of the semantics is ideal for applications like IDSS because it abstracts away from idiosyncracies of the syntax such as whether the verb was used in the active or the passive.[8] In addition, the semantics provides a mapping to WordNet synsets while also retaining the stemmed word forms from the output of the morphology and syntax. The full semantic structure produced for (15a) is shown in (15b), where the numbers represent WordNet synonym sets (synsets). The output produced from the overlay rules is shown in (15c). In (15c), only relevant entities and relations are kept from the semantics, and the information in these have been rearranged for the application (e.g., the overt marking of sentence and byte position information, the deletion of context information).

(15) a. Robin Abramov is employed by International Business Machines.

b. alias(Abramov:n(7, 1), [Robin, Abramov, Robin Abramov])
alias(International Business Machines:n(30, 1), [International Business

[8]The semantics is a level of linguistic semantics. For greater abstraction, the system can further map into Abstract Knowledge Representation (Crouch, 2005a; Bobrow et al., 2005, 2007). However, this component is not yet as stable and well-developed.

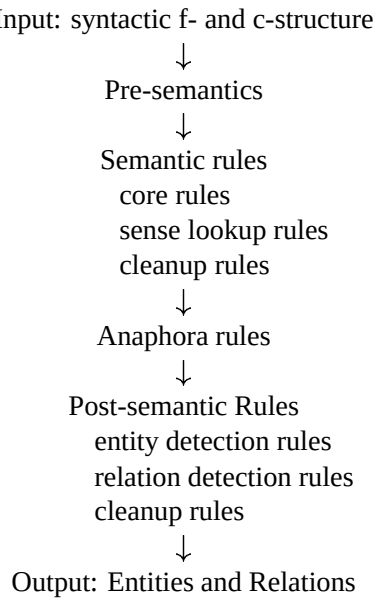
Machines, IBM])
context_head(t, employ:n(18, 1))
in_context(t, pres(employ:n(18, 1)))
in_context(t, cardinality(Abramov:n(7, 1), sg))
in_context(t, cardinality(International Business Machines:n(30, 1), sg))
in_context(t, proper_name(Abramov:n(7, 1), name, Abramov))
in_context(t, proper_name(International Business Machines:n(30, 1), company, International Business Machines))
in_context(t, role(Agent, employ:n(18, 1), International Business Machines:n(30, 1)))
in_context(t, role(Patient, employ:n(18, 1), Abramov:n(7, 1)))
lex_class(employ:n(18, 1), vnclass(unknown))
lex_class(employ:n(18, 1), wnclass(1147708, verb(consumption)))
sortal_restriction(Abramov:n(7, 1), Thing, employ)
sortal_restriction(International Business Machines:n(30, 1), Thing, employ)
word(Abramov:n(7, 1), Abramov, noun, 1, 7, t, [[9487097, 7626, 4576, 4359, 3122, 7127, 1930, 1740]])
word(International Business Machines:n(30, 1), International Business Machines, noun, 1, 30, t, [[7948427, 7943952, 7899136, 7842951, 29714, 2236, 2119, 1740]])
word(employ:n(18, 1), employ, verb, 1, 18, t, [[1147708], [2385846]])

c. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(27), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(27), [company])
WORD(employ, sent_num(1), byte_position(15), [hire, use])
sentence_num(1)

Since the semantics is run on an ordered rewrite system, the overlays take the form of additional rule sets which occur in the stack of ordered semantics rules. To do this effectively, the rules have to be factored so that new rule sets can be interwoven in the stack without having to alter the base files. If the base files have to be altered, then whenever a new version of the base semantics is released, the changes for the overlay will be lost and have to be hand added. In order to overlay the semantics, XLE provides a way to call the new, overlayed stack and to implement application specific commands.

The semantics rules used in IDSS and as the base semantics for the ParGram English grammar are divided into two main sets: semantic rewrites and anaphora resolution. The semantic rewrites are further divided into six sets, including core semantic rules, sense lookup rules, and cleanup rules. For IDSS, two additional rule sets are added before the semantic rules and after the anaphora rules. For other applications, such as consumer search, different sense lookup rules may be overlayed. The basic semantic XFR rule stack used in IDSS is shown in (16). Details of the pre- and post-semantic rules are discussed in this section.

(16) **Semantic XFR Rule Stack**



4.1 Pre-semantic Rewrite Rules

The IDSS pre-semantic rules are very simple (three calls to the same template) and are used to pick up the special entities provided by the IDSS morphology, e.g., the addresses, phone numbers. These convert the entities into a format that resembles that of proper names and other aliased items and hence is recognized by the semantics.

4.2 Post-semantic Rewrite Rules

The post-semantics/anaphora rule set is more complex. These rules operate on the output of the semantics to extract the entities and entity relations needed for IDSS: they identify entities such as proper nouns, time expressions, phone numbers, nouns in certain classes (e.g., currencies and explosives); they identify relations such as who works where, who lives where, and who knows whom; they provide information such as synonyms of each content word.

The entity detection rules are relatively straightforward. They take a subset of the *word* facts already present in the semantics and rewrite them to contain the information needed in the IDSS application. For example, all proper nouns are marked as entities and are included with information as to their type and location in the sentence, as shown in (17).

(17) a. ENTITY(Detroit, location, sent_num(1), byte_position(24), [Detroit])

b. ENTITY(Smith, person, sent_num(1), byte_position(10), [John, Smith, Mister Smith, Mister John Smith])

At the level of the semantics, no lexicon is needed to determine which entities to mark. Instead, it is features from the syntactic f-structure such as the PROPER-TYPE which provide the trigger for the rule.

The rules also allow for words with certain meanings to be extracted. This is done by determining what WordNet (Fellbaum, 1998) synset describes the class of interest and then creating entity facts for any words with this synset somewhere in the hypernyms of the word's semantics. For example, in certain application domains, explosives and weapons may be of interest and hence should be recognized as entities, which can then be highlighted or redacted as appropriate. If this is the case, the extracted entities for a sentence like (18a) will include an entity fact as in (18b) since WordNet knows that dynamite is a type of explosive.

(18) a. The dynamite arrived on Friday.

b. ENTITY(dynamite, explosive, sent_num(1), byte_position(1))

After the entities are identified, relations among them are posited. By identifying the entities first, more general relation rules can be written that look for relations between entities of a particular type, e.g. certain relations hold between person entities and company entities but not between persons and other persons.

The rules to extract relations among entities are more complicated than the entity detection rules. In general, the relations of interest are specific for a given IDSS application. For example, some application domains have rules to extract information as to which people work for which company. Detecting these relations at the semantic level is simpler than at the text string or the syntactic f-structure level. For example, all of the forms in (19) will have the same basic role relations in the semantics.

(19) a. IBM employs Robin Abramov.

b. Robin Abramov is employed by IBM.

c. IBM's employee, Robin Abramov,

d. IBM's employment of Robin Abramov

e. Robin Abramov is an employee of IBM.

f. Robin Abramov's employer is IBM.

These role relations are then used to extract an ENTITY-REL fact as in (15c). However, even at this highly normalized level, several rules can be required to extract a given relation. In the *works-for* relation example in (15), the same relation expressed by the corresponding *work for* phrases have slightly different roles assigned to them by the semantics. As such, for high value relations, there may be several rules to extract the relevant relation facts.

There is a default rule for relation extraction that marks all entities in a sentence as occuring together. This information could be reconstructed from the entity facts because the sentence number is recorded as part of the fact. However, by combining them into a single fact, applications can immediately see co-occurrences. An example is shown in (20).

(20) a. Mary left and John arrived.

b. ENTITY(John, person, sent_num(1), byte_position(15), [John])
ENTITY(Mary, person, sent_num(1), byte_position(1), [Mary])
ENTITY-REL(cooccuring(1), [Mary, John])

Once the entities and relations are identified, the rest of the semantic facts are deleted, leaving just the IDSS specific information, as shown in (15c).

Since the rules operate after all the base semantic rules, improvements to the semantics can be automatically incorporated by updating to the newest version of the base semantics. If there is a change in analysis to the semantics, it may be necessary to change the IDSS rules to be sensitive to these changes. The rules which define the feature space of the base semantics, as well as the svn version control system and the regular use of regression testing whenever changes are incorporated (Chatzichrisafis et al., 2007), make such changes in the base semantics relatively easy to track.

4.3 Flags

The rewrite system also allows flags to be set that can be used to trigger or block rules. The rules check for the setting of the flags and then trigger (or not) based on the setting for the run-time system. These flags are set when loading the rules to produce the desired behavior.

Even in non-overlay grammars, a flag of this type is used to trigger feature checking rules when used in debugging mode. Consider the feature checking rule in (21). The flag *debug(%%)* is set to 1 when the system is being run in debug mode. If it is, then the rule in (21) fires whenever there is a two argument predicate that is not listed as a licensed_feature.

(21) {getp(debug(1))},
qp(%Feat, [%Arg1, %Arg2]), −licensed_feature(%Feat,2)
==>
NOT_LICENSED_FEAT(qp(%Feat, [%Arg1, %Arg2])).

In the run-time system, this flag is turned off by setting *debug(%%)* to 0 in order to avoid the insertion of warning messages in the output structures. The use of flags in the current IDSS overlay system is kept to a minimum, being largely restricted to debugging, but it does offer a feature similar to the syntactic OT marks for removing or inserting (but not ranking) rules without altering the XFR semantic rule files themselves.

5 System Issues and Conclusions

System Issues All the above components are kept under an svn version control system and undergo regular regression testing (Oepen et al., 1998, 2002; Chatzichrisafis et al., 2007). The versioning allows easy access to previous versions of the system. This is useful not only for backing out of changes that turned out not to be improvements, but also for allowing the use of previous versions of the grammar and the semantics until the overlay grammars can catch up to the changes made. In addition, svn makes it possible for multiple developers to work on the system at the same time, helping to merge changes made by different people. The regular regression testing highlights changes, whether improvements or not, to each component and to the system as a whole. Sometimes changes to a given component will have no effect on a specific application while at other times even small changes to components can significantly alter the behavior of the system.

Conclusions Adapting a complex deep processing system to an application requires changes to all levels of the processing pipeline. As such, it is important that easy-to-use overlay mechanisms are provided at each level and that the levels are modular. The form of these mechanisms may vary depending on the type of system component (e.g., overlaying a unification-based grammar requires different techniques than overlaying an ordered rewrite system). Having such mechanisms allows the application to seamlessly incorporate improvements to the base system over time, while maintaining the specialization features. This is particularly important when base components of the system are still undergoing rapid development (e.g., with the semantics in the IDSS application described here), but even relatively stable components will improve over time and applications need to take advantage of these improvements without a major system overhaul.

This paper has outlined a series of tools that are used in XLE to overlay all levels of analysis from tokenization to semantics, using the IDSS application as an example. The XLE overlay mechanisms have been refined over time based on experiences with a number of specialized domains and applications. Even with the overlay mechanisms in place, the base rules of each component have to be designed to

allow overlays through appropriate rule factoring and modularization of rule sets and system components.

References

Beesley, Kenneth and Karttunen, Lauri. 2003. *Finite State Morphology.* CSLI Publications.

Bender, Emily, Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-linguistically Consistent Broad-Coverage Precision Grammars. In *COLING Workshop on Grammar Engineering and Evaluation.*

Bier, Eric, Ishak, Eddie and Chi, Ed. 2006. Entity Workspace: an evidence file that aids memory, inference, and reading. In *IEEE International Conference on Intelligence and Security Informatics (ISI 2006)*, pages 466–472, Springer Verlag.

Bobrow, Daniel G., Cheslow, Bob, Condoravdi, Cleo, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan, de Paiva, Valeria, Price, Charlotte and Zaenen, Annie. 2007. PARC's Bridge and Question Answering System. In *Proceedings of the Grammar Engineering Across Frameworks 2007 Workshop*, CSLI On-line Publications.

Bobrow, Daniel G. Condoravdi, Cleo, Crouch, Richard, Kaplan, Ron, Karttunen, Lauri, King, Tracy Holloway, de Paiva, Valeria and Zaenen, Annie. 2005. A Basic Logic for Textual Inference. In *AAAI Workshop on Inference for Textual Question Answering.*

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

Chatzichrisafis, Nikos, Bouillon, Pierrette, Rayner, Manny, Santaholma, Marianne, Starlander, Marianne and Hockey, Beth Ann. 2006. Evaluating Task Performance for a Unidirectional Controlled Language Medical Speech Translation System. In *Proceedings of the HLT-NAACL Workshop on Medical Speech Translation.*

Chatzichrisafis, Nikos, Crouch, Dick, King, Tracy Holloway, Nairn, Rowan, Rayner, Manny and Santaholma, Marianne. 2007. Regression Testing For Grammar-Based Systems. In *Proceedings of the Grammar Engineering Across Frameworks 2007 Workshop*, CSLI On-line Publications.

Crouch, Dick. 2005a. Packed Rewriting for Mapping Semantics to KR. In *International Workshop on Computational Semantics.*

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John T. and Newman, Paula. 2007. XLE Documentation, on-line documentation.

Crouch, Dick and King, Tracy Holloway. 2006. Semantics via F-structure Rewriting. In *Proceedings of LFG06*.

Crouch, Richard. 2005b. Packed Rewriting for Mapping Semantics to KR. In *Proceedings Sixth International Workshop on Computational Semantics, Tilburg, The Netherlands.*

Dalrymple, Mary. 2001. *Lexical Functional Grammar*. Academic Press.

Fellbaum, Christiane (ed.). 1998. *WordNet: An Electronic Lexical Database.* The MIT Press.

Frank, Anette, King, Tracy Holloway, Kuhn, Jonas and Maxwell, John T. 2001. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In Peter Sells (ed.), *Formal and Empirical Issues in Optimality Theoretic Syntax,* pages 367–397, CSLI Publications.

Kaplan, Ron, King, Tracy Holloway and Maxwell, John T. 2002. Adapting Existing Grammars: The XLE Experience. In *COLING Workshop on Grammar Engineering and Evaluation.*

Kaplan, Ron, Maxwell, John T., King, Tracy Holloway and Crouch, Richard. 2004a. Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP (ESSLLI).*

Kaplan, Ron and Newman, Paula. 1997. Lexical Resource Concilation in the Xerox Linguistic Environment. In *ACL Workshop on Computational Environments for Grammar Development and Engineering.*

Kaplan, Ron, Riezler, Stefan, King, Tracy Holloway, Maxwell, John T., Vasserman, Alex and Crouch, Richard. 2004b. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL'04.*

Maxwell, John and Kaplan, Ron. 1991. A Method for Disjunctive Constraint Satisfaction. *Current Issues in Parsing Technologies* .

Maxwell, John and Kaplan, Ron. 1996. An Efficient Parser for LFG. In *Proceedings of the First LFG Conference,* CSLI Publications.

Oepen, Stephan, Flickinger, Dan, Toutanova, Kristina and Manning, Chris. 2002. LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. In *First Workshop on Treebanks and Linguistic Theories.*

Oepen, Stephan, Netter, Klaus and Klein, J.1998. TSNLP — Test Suites for Natural Language Processing. In John Nerbonne (ed.), *Linguistic Databases,* CSLI.

Prince, Alan and Smolensky, Paul. 1993. Optimality Theory: Constraint Interaction in Generative Grammar, ruCSS Technical Report #2, Center for Cognitive Science, Rutgers University.

Rayner, Manny, Hockey, Beth Ann and Bouillon, Pierrette. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI.

Towards a Generic Multilingual Dependency Grammar for Text Generation

François Lareau
Pompeu Fabra University

and

Leo Wanner
Pompeu Fabra University
and ICREA

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

For practical multilingual text generation, efficient development and representation of large scale grammatical and lexical resources are crucial. One way to ensure efficiency is to share resources as much as possible between languages. We present some preliminary work on shared grammatical resource development within the framework of the Meaning-Text Theory, emphasizing the lexicalist point of view. We show that rich dictionaries allow for more generic grammar rules which can be used for several languages, so that the number of language-specific rules is kept low. We also discuss the benefits of shifting the workload to the dictionaries from the viewpoint of extension and consistency control as well as the impact it has on the organization of work. Furthermore, we address evaluation methodologies for the shared grammatical resources we develop.

1 Introduction

Practical multilingual natural language generation (MNLG) cannot be achieved without large scale grammatical and lexical resources. For an efficient development of multilingual broad coverage grammatical resources, two different strategies have been applied: *grammar porting* (Alshawi, 1992; Kim et al., 2003) and *grammar sharing* (Avgustinova and Uszkoreit, 2000; Bender et al., 2002; Bateman et al., 2005; Santaholma, 2007). In this article, we present some preliminary work on the development of shared grammatical MNLG resources in the framework of the dependency-based *Meaning-Text Theory*, MTT (Mel'čuk, 1988). MTT has traditionally been popular in text generation due to its multi-stratal linguistic model, which allows us, on the one hand, to select for the input structure a degree of abstraction that suits best the application in question, and, on the other hand, to keep the generation resources as modular and as simple as possible.

According to MTT, sentence generation is viewed as a sequence of transductions between structures of adjacent strata. Depending on the required degree of abstraction, generators may start from the conceptual, semantic, or syntactic structure (see also below). Each transduction is realized by a separate language-dependent grammar such that grammar developers are faced with the task of developing $n \times (m-1)$ grammars for each application (with n being the number of languages covered and m the number of strata involved in the generation process). The need for efficient sharing of grammatical resources across languages is thus obvious.

[†]The work described in this paper has been funded by the European Commission in the framework of the *eContent* Programme under the contract number EDC-11258. We would like to thank all colleagues who have contributed to the development of the resources presented here: Margarita Alonso Ramos, Bernd Bohnet, Kim Gerdes, Simon Mille, Christophe Onambele Manga, Patrycja Przewoźnik, and Vanesa Vidal. Special thanks go to Bernd Bohnet, who acted as firefighter whenever MATE was not behaving as the grammarians expected. Many thanks also to Emily Bender and Tracy Holloway King for suggestions that significantly improved the final version of the paper.

In our current application, we cover six languages (Catalan, English, French, Polish, Portuguese, and Spanish) for the domain of air quality, using as development framework and generator the graph grammar-based workbench MATE (Bohnet et al., 2000; Bohnet, 2006). It turned out that the effect of resource sharing even across languages that belong to different families (Romance, Germanic, and Slavic) is considerable. In what follows, we present our experience.

The remainder of the article is structured as follows. In Section 2, we give a short introduction to MTT. Section 3 describes the formalism used for the dictionaries and grammars in MATE. Section 4 contains the general principles that underlie our grammatical resource architecture. In Section 5, we assess the benefits of this architecture for efficient grammar development, before presenting in Section 6 an evaluation of the resources thus obtained. Section 7, finally, summarizes the central aspects of our approach and offers some conclusions.

2 Overview of MTT

As already mentioned above, MTT is based on a multi-stratal linguistic model. In total, seven different strata are distinguished, of which five are immediately relevant to written language generation: (i) the semantic stratum, (ii) the deep-syntactic stratum, (iii) the surface-syntactic stratum, (iv) the deep-morphological stratum,[1] and (v) the surface-morphological stratum. For generation applications that start from a non-linguistic content representation or even from numerical data series (as we do), an additional *conceptual* stratum is added.

Each stratum has its own alphabet over which structures for that stratum are defined, and its own interpretation for those structures. Thus, conceptual structures (ConS) are *conceptual graphs* in the sense of Sowa (2000). Semantic structures (SemS) are predicate-argument graphs with nodes labeled by semantemes and arcs labeled by the ordinal numbers of the argument relations (ordered in ascending degree of obliqueness). Deep-syntactic structures (DSyntS) are dependency trees with nodes labeled by "deep" lexical units (LUs)[2] and arcs labeled by universal syntactic relations: argument (I, II, III, ...), attributive (ATTR), and coordinative (COORD). Surface-syntactic structures (SSyntS) are dependency trees with nodes labeled by any kind of lexeme (including closed class lexemes) and arcs labeled by grammatical functions (subject, direct object, ...); SSyntS is thus equivalent to the f-structure in LFG. Deep-morphological structures (DMorphS) are chains of lemmas annotated with all relevant morpho-syntactic features. Surface-morphological structures (SMorphS) are similar to DMorphS except that contractions, elisions,

[1] In the MTT literature, the deep-morphological stratum has recently also been referred to as "Topological Stratum" (Gerdes and Kahane, 2007).

[2] The set of deep LUs of a language $\mathcal{L}$ contains all LUs of $\mathcal{L}$—with some specific additions and exclusions. Added are two types of "artificial" LUs: (i) symbols of *lexical functions* (LFs), which are used to encode lexico-semantic derivation and lexical co-occurrence (Mel'čuk, 1996); (ii) fictitious lexemes, which represent idiosyncratic syntactic constructions of $\mathcal{L}$. Excluded are: (i) structural words, (ii) substitute pronouns and values of LFs.

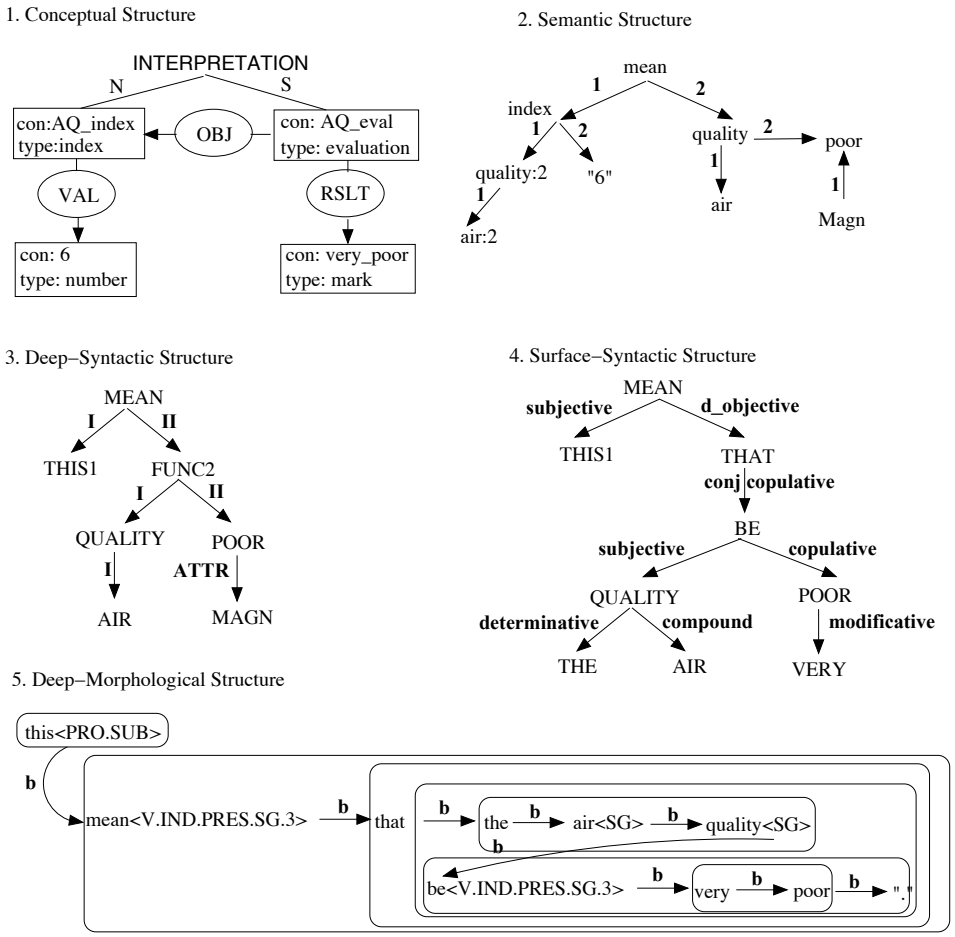


Figure 1: Sample structures at different strata of an MTT model

epenthesis and morph amalgamation have been performed. Figure 1 illustrates the first five types of structures for the sentence *This means that the air quality is very poor*; the SMorphS is obvious and does not need explicit illustration.

For each pair of adjacent strata $\mathfrak{S}_i$ and $\mathfrak{S}_{i+1}$, a separate grammar module $\mathfrak{G}^i_{i+1}$ is defined such that any well-formed structure S_{i_j} of $\mathfrak{S}_i$ can be mapped by $\mathfrak{G}^i_{i+1}$ onto a well-formed structure S_{i+1_k} of $\mathfrak{S}_{i+1}$, with S_{i_j} and S_{i+1_k} being equivalent with respect to their meaning. For convenience, we introduce a further grammar module to map a SMorphS onto a text string. As a rule, the mapping requires access to dictionaries containing information concerning the units of S_{i_j} and S_{i+1_k}.

3 Formal Framework: MATE

The MATE workbench consists of a number of support modules for the development of dictionaries and grammars and a transduction-based generator that maps

any automatically derived or manually specified input structure S_{i_j} of the stratum $\mathfrak{S}_i$ onto its equivalent structure S_{i+1_k} of the stratum $\mathfrak{S}_{i+1}$ by applying the corresponding grammar module $\mathfrak{G}^i_{i+1}$ to S_{i_j} under the use of dictionaries.

3.1 Dictionary Encoding in MATE

Dictionaries contain two types of information: (i) information concerning the elements of the different node alphabets (the vocabulary) and (ii) information concerning the correspondence between elements of node alphabets of adjacent strata. Therefore, three main dictionaries are available: a conceptual dictionary, a semantic dictionary and a lexical dictionary. All are organized in terms of recursive feature structures.

The *conceptual dictionary* is used, first of all, to encode concept-semanteme mapping information; cf. a simplified entry for the concept CONCENTRATION:[3]

concentration: property_attribute {
sem = 'concentration'
MATR = {relation = 1 target=referent}
VAL = {relation = 2 target=referent}
ATTR = {relation = 1 source=referent}}

The concept CONCENTRATION has two argument slots: something which has a concentration (referred to as MATR in accordance with Sowa (2000)), and a value (referred to as VAL), i.e., an absolute concentration figure. The concept may also be modified by a qualitative characterization of the concentration ("high", "low", etc.), referred to as ATTR. The corresponding semanteme 'concentration' takes MATR as its first semantic argument (indicated by the "relation=1" parameter embedded in MATR's value) and VAL as its second. The attributes "target=referent" and "source=referent" indicate the direction of the semantic relation (for MATR and VAL, the semantic predicate is 'concentration', which takes MATR's and VAL's corresponding semantemes as its arguments, while ATTR's semantic correspondent is a predicate taking 'concentration' as its argument).

The *semantic dictionary* gives, for every semanteme described, all its possible lexicalisations. For instance, the meaning 'cause' would be mapped to the LUs CAUSE$_{[V]}$, CAUSE$_{[N]}$, RESULT$_{[V]}$, RESULT$_{[N]}$, DUE, BECAUSE, CONSEQUENCE, etc. Note that we do not consider at this stage the valency of the LUs. Thus, it does not matter that *X causes Y* means that *Y results from X*; what interests us here is only that these two lexemes can both be used to denote the same situation, regardless of the communicative orientation. Cf., for illustration, the entry for the semanteme 'concentration' as specified in the semantic dictionary:

concentration {
label = parameter
lex = concentration }

[3]More information can be added to this basic entry, but we leave it aside in this paper.

The semantic type of a semanteme can be specified in the semantic dictionary (cf. “label=parameter”). It is also possible to specify the semantic type of the arguments of a predicate. For instance, adding the attribute “1=substance” here would force the first semantic argument of ‘concentration’ to be of type “substance”.

The *lexical dictionary* contains, for each LU, at least information on its part of speech and minimal sub-categorization information. For more elaborate generation, the whole variety of sub-categorization patterns and lexical co-occurrence (i.e., *collocation*) information should also be captured. As already mentioned above (see footnote 2), the latter is specified in terms of *lexical functions* (LFs). A lexical co-occurrence LF is a directed lexico-semantic relation that holds between two LUs that form a lexically restricted expression (e.g. *heavy smoker*, *make a statement*, etc.) (Mel’čuk, 1996). When applied as a function to the semantic head of the expression, such an LF provides the second element.[4] There is a specific (simple or complex) LF for each recurrent co-occurrence pattern in language. LFs are shown to be language- and domain-independent. But note that LF instances are by nature language- and even domain-specific. Consider, for illustration, the entry for CONCENTRATION:

```
concentration {
    // Grammatical characteristics:
    dpos = N // deep part of speech is N(oun)
    spos = common_noun // surface part of speech is common noun
    // Government pattern (subcategorization):
    gp = {
        // Sem-DSynt valency projection (1⇒I, 2⇒II):
        1 = I // first semantic actant is first deep-syntactic actant
        2 = II // second semantic actant is second deep-syntactic actant
        // First syntactic actant can be realized as "ozone concentration":
        I = {
              dpos=N // actant is a noun
              rel=compound // linked with compound relation
              det=no // takes no determiner
        }
        // First syntactic actant can be realized as "concentration of ozone":
        I = {
          dpos=N // actant is a noun
          rel=noun_completive // linked with noun_completive relation
          prep=of // takes preposition "of"
          det=no // takes no determiner
        }
        // Second syntactic actant can be realized as "concentration of 180 μg/m3":
        II = {
            dpos=Num // actant is a number
            rel=noun_completive // linked with noun_completive relation
```

[4] An LF may provide as second element several alternative LUs; cf. *g*ive|*d*eliver|*m*ake| *a speech*. LFs are thus *maps* rather than functions. However, for convenience, LFs are usually referred to as functions in the literature.

```
            prep=of // takes preposition "of"
        }
    }
    // Lexical functions:
    Magn = high
    AntiMagn = low
    Adv1 = in // "(we found) ozone in a concentration (of 180 μg/m3)"
    Func2 = be // "the concentration (of ozone) is 180 μg/m3"
    Oper1 = have // "ozone has a concentration (of 180 μg/m3)"
    IncepFunc2 = reach // "the concentration (of ozone) reached 180 μg/m3"
    IncepOper1 = reach // "ozone will reach a concentration (of 180 μg/m3)"
}
```

We use two levels of granularity for the part of speech, referred to as *deep* and *surface* part of speech (resp. *dpos* and *spos*). This allows for quick reference to a whole family of parts of speech in grammar rules (for example, "N" refers to any proper noun, common noun, or pronoun). All specific grammatical characteristics of an LU would be described here as a feature-value pair (for example, its gender or its ability to take or not plural, definiteness, a certain tense, etc.).

The sub-categorization must contain the projection of the semantic to the syntactic valency and all possible ways of syntactically connecting the LU with its dependents. Governed prepositions must be indicated here, as well as case assignment if it exists in the language being described. One can optionally restrict the part of speech of the dependents (for instance, the first actant of CONCENTRATION must be a noun, while its second must be a number).

As mentioned above, lexical functions are an efficient way of referring to recurrent semantic and syntactic patterns of restricted lexical co-occurrence. In the example above, $Magn$ points to an LU which is a syntactic modifier and has a meaning of intensification ($AntiMagn$ is its antonym). The function $Func_2$ refers to a semantically emptied verb that takes the keyword (CONCENTRATION) as its subject and the keyword's second semantic actant as its object. $IncepOper_1$ points to a verb meaning roughly 'start' which takes the keyword as its object and the first actant of the keyword as its subject. The more information on restricted lexical co-occurrence the lexical dictionary contains, the more natural and idiomatic the generated text will feel.

MATE lets the user define as many dictionaries as necessary. We have presented the three main ones we have in our resources, but there can be more. For instance, it is possible to use a full-form dictionary instead of a proper morphological model, or a hybrid model as we implemented in our system. We also had a pseudo-dictionary for each language where we stored information such as the name of the language, the branch/family it belongs to, its being a "pro-drop language" or not, etc. This information forces or blocks the application of specific rules. For example, marking a language as "pro-drop" blocks the rules of SSynt $\Rightarrow$ DMorph that realize pronominal subjects.[5]

[5]We still need the pronoun in the surface syntactic structure in order to perform agreement, which

3.2 Grammar Encoding in MATE

A grammar $\mathfrak{G}^i_{i+1}$ consists of a set of minimal grammar rules of the following general format (see (Bohnet, 2006, 39ff) for details):

leftside (ls): $< g_i >$
rightside (rs): $< g_{i+1} >$
rightcontext (rc): $< g'_{i+1} >$
conditions (cd): <Boolean expr. over $\mathfrak{D}_{conc} \cup \mathfrak{D}_{sem} \cup \mathfrak{D}_{lex} \cup \mathfrak{S}_i \cup \mathfrak{S}_{i+1} >$
correspondences (cr): $\{n_{i_j} \Leftrightarrow n_{i+1_k}\}$

with g_i being a graph defined over the node and arc alphabets of $\mathfrak{S}_i$, g_{i+1} and g'_{i+1} being defined over the node and arc alphabets of $\mathfrak{S}_{i+1}$; $\mathfrak{D}_{conc}$, $\mathfrak{D}_{sem}$, $\mathfrak{D}_{lex}$ being the conceptual, semantic and lexical dictionaries; and $n_{i_j} \in g_i$, $n_{i+1_k} \in g_{i+1}$. The application of a rule consists in the identification of an isomorphic image of g_i in a given source structure S_{i_j} and subsequent introduction of an isomorphic image of g_{i+1} in the target structure S_{i+1_k} which is under construction. The statement '$n_{i_j} \Leftrightarrow n_{i+1_k}$' establishes a link between corresponding nodes in S_{i_j} and S_{i+1_k} in order to ensure that (i)information can be propagated from node to node across strata, (ii) the isolated fragments of the target structure as introduced by the individual rules can be unified to a connected well-formed structure. A rule is applicable if the specified conditions are fulfilled. As indicated, conditions may be defined over all dictionaries and both strata.[6] The rules in $\mathfrak{G}^i_{i+1}$ are minimal in the sense that the left-hand side of each rule is maximally elementary from the linguistic perspective: its g_i consists either of an elementary meaningful graph defined over the alphabets of $\mathfrak{S}_i$ or a graph that is transduced to an elementary meaningful graph defined over the alphabets of $\mathfrak{S}_{i+1}$. As a rule, an elementary meaningful graph consists either of a single node (a name) or a single arc (a linguistic relation)—although sometimes bigger structures are required.

In the remainder of this section, we illustrate the system with sample rules for the first four types of transduction involved in MTT-based generation.[7]

Rule 1 (Sample Con ⇒ Sem rule)

```
ls:  ?Xcon{PTIM->?T{con="tomorrow"}}
rc:  ?Xsem{tense=FUT}
cr:  ?Xcon ⇔ ?Xsem
```

Rule 1 maps the conceptual time relation between the concept denoted by the variable '?Xcon' and the "universal"[8] concept TOMORROW onto the tense feature

takes place in the SSynt ⇒ DMorph transduction.

[6] As a matter of fact, the conditions may also draw on the context, the discourse structure, the user model, etc. However, for simplicity's sake, we neglect this issue here.

[7] The rules of the DMorph ⇒ SMorph and SMorph ⇒ Text transductions are less interesting since they simply spell out morphological features of the words and pass the strings to an external morphological model.

[8] It is not absolutely true that all concepts are universal; some could be said "culture-specific". For instance, periods of the day vary considerably from one culture to another. In Spain, for example,

"FUT" of the semanteme denoted by the variable '?Xsem'. Note that '?Xsem' is specified in the right context slot—which means that the corresponding semanteme is assumed to have been already introduced into the target structure by another rule.

Rule 2 (Sample Sem ⇒ DSynt rule)

```
ls:  ?Xsem{?r->?Ysem}
rs:  ?Xds{I->?Yds}
rc:  ?Xds
cr:  ?Xsem ⇔ ?Xds
     ?Ysem ⇔ ?Yds
cd:  lexicon::(?Xds.lex).(gp).(?r)=I
```

Rule 2 maps any semantic relation (denoted by the variable '?r') of the semanteme denoted by '?Xsem' onto the first deep syntactic actant of the corresponding LU (denoted by '?Xds'). The node '?Xds' being also in the right context slot, must be already present in the target structure. This rule has a condition that accesses a dictionary called "lexicon" (which is the lexical dictionary introduced in Section 3.1). It searches for the entry that corresponds to the lexicalisation on the node '?Xds' and browses its attributes to verify that the projection of the semantic to the syntactic valency of the LU is such that the semantic relation '?r' is mapped to the deep-syntactic relation 'I'. For instance, this rule would apply to the first semantic argument of 'concentration' (cf. the sub-categorization for CONCENTRATION in Section 3.1). This rule can be further refined to handle any deep-syntactic actantial relation and to retrieve more information from the dictionary, such as grammatical features imposed on the actant by its governor (part of speech, mood, definiteness, etc.). For the sake of clarity, we shall consider only this simplified version.

Rule 3 (Sample DSynt ⇒ SSynt rule)

```
ls:  ?Xds{dpos=V; finiteness=FIN; mood=IND; tense=FUT}
rs:  ?Yss{slex=will
          dpos=lexicon::(will).dpos
          spos=lexicon::(will).spos
          tense=PRES
          finiteness=?Xds.finiteness; mood=?Xds.mood
          aux_completive->?Xss{finiteness=INF}
rc:  ?Xss
cr:  ?Xds ⇔ ?Yss
     ?Xds ⇔ ?Xss
cd:  language::(id).(iso)=ENG
```

Rule 3 introduces for an English verbal LU (referred to by '?Xds') that carries in the DSyntS the grammemes FIN, IND, and FUT the auxiliary WILL. WILL

the afternoon does not start before 3PM, while in Germany it starts as early as 12:00. We leave this problem aside as it is beyond the scope of this paper.

inherits from ?Xds the grammemes of finiteness and mood, but not of tense—which is for WILL PRES(ent) since the auxiliary itself has the present form (though it does express a future tense). Note that in this case, one deep-syntactic node corresponds to two surface-syntactic nodes.

Rule 4 (Sample SSynt ⇒ Top rule)

```
ls:  ?Xss{dpos=V
          subj-> ?Yss
          ?r-> ?Zss}
rc:  ?Ytp{b-> rc:?Ztp}
cr:  ?Ytp ⇔ ?Yss
     ?Ztp ⇔ ?Zss
cd:  not ?r=circumstantial
```

Rule 4 defines the relative ordering between the subject ('?Yss') of a verbal lexeme ('?Xss') and any other dependent of the verb ('?Zss'): the subject goes before. The circumstantials (roughly speaking, the adjuncts) are excluded since they may come before the subject.

4 Principles of Grammatical Resource Development

The sample rules cited above for illustration already give a hint that writing and maintaining comprehensive MTT-based generation grammar modules is a complex and very costly task—in particular, if the generation is to be multilingual.

To achieve the maximal efficiency possible, we adopt the following guidelines when organizing the grammatical resources:

(a) extracting recurrent core rule patterns across languages and factorizing them out into a "meta-grammar",

(b) modularizing language-specific rules,

(c) shifting the bulk of the grammarian's work to the lexicon,

(d) generalizing recurrent lexical patterns and introducing an inheritance mechanism.

Let us discuss the application of each of these guidelines in practice.

4.1 Sharing Core Grammar Components Across Languages

When developing grammatical resources for several languages in parallel, one quickly finds that many of the rules are the same in more than one language—to the point that some are identical for all languages under consideration. For instance, Rule 2 mentioned in Section 3.2 would apply no matter whether the language is

Catalan, English, French, Polish, Portuguese or Spanish. It makes no reference to any specific LU, nor does it refer to any language-specific relation (semantic and deep-syntactic relations being universal by definition). In that sense, it is a "universal" rule in our system. However, we do not claim that our resources contain even one single rule that could be applied in any possible language. Although Rule 2 seems a good candidate to universality (since it merely activates lexical information), consider for instance Rule 4 in Section 3.2. This rule also applies to all the languages we considered in our application (even supposedly "free-order" Polish looked better when the subject came first for the texts we had to generate). However, it is clear that it cannot apply to all known languages.[9]

In contrast with these two generic rules, Rule 3 given in Section 3.2 is only valid for English. It refers to a specific lexeme (WILL) and it even explicitly requires that the ISO identification code of the language being currently processed be "ENG" (see the note on the pseudo-dictionary for languages in Section 3.1).

Between these two extremes, it is possible to have rules that apply to a family (or any arbitrary set) of languages. For example, consider noun-determiner agreement. It does not exist in English nor in Polish. However, it is functionally the same in all Romance languages under consideration (the determiner agrees in gender and number with its governing noun). It is therefore possible to have only one rule for all those languages (cf. Rule 5).

Rule 5 (An SSynt ⇒ DMorph agreement rule for Romance languages)

```
ls:  ?Xss{dpos=N
          det->?Yss}
rc:  ?Ytp{gender=?Xss.gender
          number=?Xss.number}
cr:  ?Xtp ⇔ ?Xss
     ?Ytp ⇔ ?Yss
cd:  language::(id).(family)=romance
```

The general principle is to minimize the number of language-specific rules (such as Rule 3) and maximize the number of generic rules. The degree of generalization one can achieve for a module depends on the language and the strata involved. Languages that have agreement or a lot of lexical markers for grammatical meanings (articles, auxiliaries, etc.) require more language-specific rules. Table 1 shows, for each module in our system, the number of generic and language-specific rules we have and the percentage (in parentheses) of language-specific rules for each language within each module. The last row shows the percentage of language-specific rules for the individual languages over all modules.

As one can observe from Figure 2, the deeper the strata, the more generic a module tends to be. The Con ⇒ Sem module is entirely language-independent, as it relies on a more or less ad-hoc dictionary where a lot of information is hard-coded. It is however highly domain-specific. We do not consider it as part of the

[9] Cf., e.g., the word order in relative clauses in German.

Table 1: Number of generic and specific rules per module and language

Module	**Core**	CT	EN	ES	FR	PL	PT
Con ⇒ Sem	50	0	0	0	0	0	0
Sem ⇒ DSynt	59	8 (12%)	8 (12%)	7 (11%)	7 (11%)	11 (16%)	6 (9%)
DSynt ⇒ SSynt	64	13 (17%)	16 (20%)	11 (15%)	16 (20%)	7 (10%)	12 (16%)
SSynt ⇒ DMorph	70	13 (16%)	3 (4%)	12 (15%)	19 (21%)	8 (10%)	14 (17%)
DMorph ⇒ SMorph	7	8 (53%)	5 (42%)	6 (46%)	10 (59%)	10 (59%)	6 (46%)
SMorph ⇒ Text	12	1 (8%)	1 (8%)	1 (8%)	1 (8%)	1 (8%)	1 (8%)
Language-specific rules (%)		14%	11%	12%	17%	12%	13%

linguistic model as such, since it has to be rewritten for each application,[10] while the other modules are intended to be as domain-independent as possible.

The Sem ⇒ DSynt module has, on average, 12% of language-specific rules (with figures ranging from 9% for Portuguese to 16% for Polish). Language specific rules at this level are essentially for handling deep anaphora and some idiomatic expressions that cannot be captured by standard lexical functions, such as *in the afternoon* in a sentence like *In the afternoon, the ozone concentration was <will be> high*, which can be used only if the afternoon in question is already in the past or has not come yet (but not if it is present). The rest of the rules are generic and handle deep lexicalisation, syntactic tree building, support verbs described via standard lexical functions, quantification, etc. For example, Rule 2 in Section 3.2 activates the projection of the semantic to the syntactic valency found in the dictionary for any LU of any language.

The DSynt ⇒ SSynt module has an average of a little more than 16% language-specific rules. This ratio varies considerably from one language to another (from 10% for Polish to 20% for English and French). This is because auxiliaries, articles and all other grammatical words are handled at this level. Thus, languages with more lexical markers for grammatical meanings will require more specific rules in this module than languages that tend to express these meanings morphologically. For the treatment of verbal tense and aspect, we have a separate rule for each possible auxiliary combination (*will do*, *will be doing*, *will have done*, *will have been doing*, etc.). It would certainly have been possible to write only one rule for each auxiliary, with conditions handling the correct composition when more than one auxiliary is used. As a matter of fact, this would have better followed our general guidelines for grammar development (we tend to generalize the rules as

[10] We are investigating ways of automating this task.

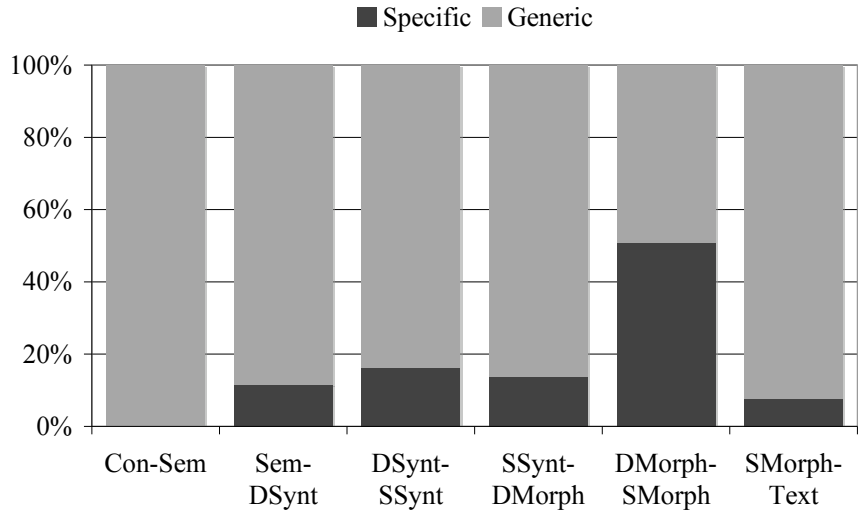


Figure 2: Average generic / language-specific rule ratio by module

much as we can, in order to keep a high maintainability of the resources). Doing so would have reduced the number of rules necessary for the auxiliaries from 12 to 4 for English. However, it would also have made those four rules significantly more complex. We preferred keeping a higher number of simpler rules, so that grammarians with less experience in formal linguistics would easily understand, maintain and port them to other languages.

The SSynt ⇒ DMorph rules model two main phenomena: word-order and agreement. Both phenomena vary greatly from one language to another. It is therefore a little surprising that we have less than 14% language-specific rules on average here. We believe that this is something of a statistical anomaly in that most of the time, the best word-order for the texts we had to generate was more or less the same for all languages considered in this application, including Polish. Obviously, if we had to generate other kinds of texts or in other languages, we would certainly see the number of language-specific rules go up for this module. However, the good news is that word-order rules are among the simplest, so writing and maintaining them is easy.

The DMorph ⇒ SMorph module shows the highest ratio of language-specific rules: almost 51% on average (ranging from about 42% for English to nearly 59% for French and Polish). However, this module contains very few rules (only 12 to 17 rules depending on the language, of which 7 are generic). It prepares the strings that will be passed to the morphological module (or the full-form dictionary), with all grammatical features in the correct order (so that an English finite verb, for instance, would look like “reach<V><IND><PRES><SG><3>”). Basically, most language-specific rules of this module only recopy the attributes found on the nodes at previous levels as explicit codes in the chain that labels the node. It is a purely

technical process that has little linguistic relevance, but the number of language-specific rules still has a strong correlation with the nature of the language: the more complex the morphology of the language is, the higher the number of language specific rules will be. It is also in this module that operations such as elision (Fr. *le homme → l'homme*), contraction (Eng. *does not → doesn't*) and epenthesis (Pl. *w wjezdzie → we wjezdzie*) are computed.

The number of language-specific rules in the SMorph ⇒ Text module[11] is quite low (8% for all languages). However, this figure does not reflect any interesting linguistic fact. This module only manages the final realization of wordforms; it handles spelling out, capitalization, and so on. The number of rules in this module is very low (13 only for any language, of which 12 are generic). The only rule that is language-specific simply calls the appropriate two-level morphology model or full-form dictionary for the language in question, passing on the string that was built by the previous module.

Given that in each module we need some language-specific rules, the rules are sensitive to the language that is being processed. While most can always apply, some are marked in order to apply (or not to apply) for a given language (or set of languages). In fact, what we have is a set of rules from which the grammar of a specific language is a subset. From the point of view of the developer, it can be seen as a pool of shared rules to which one can "subscribe" for the language he wants to describe.

4.2 Rule packaging

The rules inside each module are further organized into packages. A package is a set of rules that work together, or on the contrary, are in competition; it handles one specific linguistic phenomenon. For example, in the DSynt ⇒ SSynt module, there are separate packages for idioms, coordination, auxiliaries, etc. Formally, a package is defined by an abstract rule from which other rules depend. An abstract rule is always empty, but it may have conditions associated to it, which are inherited by all the rules that depend on it. A package can be composed of sub-packages. It is notably the case of the language packages. Language-specific rules are grouped into a separate package for each language, which consists of a number of sub-packages for various language-specific phenomena (see Section 4.1 for a list of such phenomena by module).

In each module, there is a so-called "core" package that contains all essential rules that are needed for processing any input structure. For instance, the SSynt ⇒ DMorph module's core rules handle lexicalisation, actantial relations and the ATTR-relation (for modifiers), without which nothing can be done. Also in this module are packages for lexical co-occurrence (in particular support verbs), quantification, circumstantials, voice, and language-specific packages (mainly for deep anaphors). Phenomena that span over more than one module, such as coordination,

[11] Recall that we use an additional transduction from the SMorphS to text not foreseen in MTT.

have a corresponding package in each module involved.

With this design it is possible to assign different packages to different developers who are specialists of a specific domain and who need not worry about the problems outside of their sandbox. It is also easier to modify the grammar to meet specific needs by choosing the desired modules or by adding in new ones.

So far, we have limited the package-based design to grammars only. Eventually, we will also adopt the same architecture for the dictionaries. The lexical core of each language should constitute the main package, while additional packages could be developed by qualified lexicographers for specific domains (air quality, traffic information, healthcare, etc.).

4.3 Rich Hierarchical Dictionaries

As is customary in many modern grammar theories (among others, HPSG, SFG, Word Grammar, etc.) and their implementations, we use inheritance in the lexical resources, factorizing all possible lexical information into abstract entries from which the LUs depend. Consider, for illustration, a fragment of the verbal hierarchy $predicate \rightarrow verb \rightarrow direct\ transitive\ verb$.

The $predicate$ node provides the default projection of the semantic valency to the syntactic valency of a predicative unit. We assume that a predicate possesses at most six actants, with the ith semantic actant (denoted by an Arabic numeral) usually corresponding to the ith syntactic actant (denoted by a Roman numeral):

```
"predicate" {
   gp={ 1=I; 2=II; 3=III; 4=IV; 5=V; 6=VI } }
```

A verbal lexeme is a predicate (i.e., inherits, if not overridden, all features defined for the $predicate$ unit). Furthermore, its surface and deep part of speech are respectively 'V' and 'verb' and, by default, its first syntactic actant is realized as a grammatical subject, usually a noun (this can of course be overwritten for any given verb):

```
"verb" :  "predicate" {
   dpos=V; spos=verb
   gp={ I= {dpos=N; rel=subj} }
}
```

Note that such abstract entries are not necessarily universal. For each language, we keep a separate hierarchy since the parts of speech and the morpho-syntactic behavior of their members can vary cross-linguistically. For example, Polish verbs usually assign the nominative case to their subject, unless otherwise specified, so this information would be added to the abstract $verb$ entry for Polish. One could prefer having a $polish\_verb$ entry that would inherit from the $verb$ entry above (or even from a more generic one that would not specify the nature of the subject, for instance) and refine it. However, it is not possible in the current version of the

system to have an entry shared by several languages. All that can be done is to copy the $verb$ entry into the respective dictionary of each language. Since it was not possible to have the information written once for all languages, and since the differences between the parts of speech of the languages we had to deal with were not great, we did not seek a more refined hierarchy.

English direct transitive verbs inherit from the $verb$ class. Furthermore, they realize their second syntactic actant as a direct object, and it is by default a noun:

```
"verb_dt" :  "verb" {
   gp={ II= {dpos=N; rel=dobj} }
}
```

Now, adding a direct transitive verb to the lexicon is just a matter of expressing its membership in the $verb\_dt$ class. Consider, for illustration, the entry for the verb EXCEED below, where we have added information on its lexical co-occurrence:

```
exceed :  "verb_dt" {
   Magn = "by far"
   AntiMagn = "a little"
}
```

All information about the projection of the semantic to the syntactic valency, part of speech and surface realization of the actants has been inherited. Of course this information can be overridden, simply by overwriting it. For example, the verb EXPECT has two possible sub-categorization patterns, none of which corresponds to the default pattern for verbs:

```
expect :  "verb" {
   gp={ II={dpos=V; finiteness=FIN; mood=IND; prep=that} }
   gp={ II={dpos=V; finiteness=INF; prep=to; rel=iobj}
        raise={ II={rel=dobj} } }
}
```

The first pattern corresponds to *We* [=I] *expect that the ozone concentration will increase* [=II]. The second pattern points to a subject-raising construction where the subject of actant II is raised to become the direct object of EXPECT, downgrading actant II to an indirect object position, as in *We* [=I] *expect the ozone concentration* [=raised subject] *to increase* [=II].

5 Benefits of the Proposed Grammar Design

The principles outlined above and followed in our work ensure that

(i) no parts of resources are repeated,

(ii) the resources are linguistically sound, and

(iii) the acquisition and maintenance (i.e., evaluation, correction and extension) of the resources can be carried out easily by grammarians without an extensive training in the linguistic theory underlying the generator.

To be underlined in particular are the extensibility to new languages and the extension towards the coverage of new linguistic constructions. Thus, this design allows for quick addition of new languages to the generator. Few changes need to be made to the grammar rules, since most of the language-dependent information is in the dictionaries. Rules that take care of articles, auxiliaries and other lexical markers of grammatical meanings, as well as agreement and word-order rules do need to be modified, but they are usually very simple. Hence, the task of adding a new language basically boils down to describing the LUs of the language in question.

Similarly, adapting the generator to a new domain is even more simple. Many of the existing dictionary entries can be reused, and one only needs to describe the new lexemes found in the vocabulary of the new domain. As a matter of fact, while the grammar described here was first used for a project in the domain of air quality, we have successfully reused the surface modules (from SSynt $\Rightarrow$ DMorph) in another project for a totally different domain (patents on optical recording technology) with very little modifications to the rules.

A benefit that is not to be underestimated for the design described here is the ease of development that follows from it in terms of work organization. Broad coverage grammars, especially in a multilingual context, require a relatively large team. It is then unrealistic to hope for a homogeneously qualified team who can work on any aspect of the problem, in particular within a lesser-taught framework such as MTT. Fine-grained modularity allows for efficient task separation. The number of language-specific grammar rules being kept as low as possible, most of the work for adding a new language lies in writing the dictionaries for it.

One could argue that all we have done was just to shift the workload from the grammar to the dictionary. It is true to some extent, but the formalism used for grammar rules is much more complex than the one used for dictionaries. We have shown here simplified rules, but a serious grammar cannot consist only of such simple rules. They can get rather complex, enough to scare away a potential contributor who is not necessarily very comfortable with formal languages. Dictionaries, however, are written in a very simple formalism that can be mastered quickly. Indeed, our experience showed that it was much easier for people to learn how to write dictionaries than how to write grammar rules. Hence, by shifting the workload to the dictionary, we simplify the process of describing new languages as resources become more easily maintainable and extensible. By adopting this architecture, we are able to have a (more or less) permanent core team that has enough experience with the grammar formalism to work on the generic rules, and short-term collaborators who can join for a specific project to develop resources for a new language without having to learn in detail the formalism used for the codification of grammar rules.

6 Evaluation

In accordance with the evaluation principles in software engineering, we use a twofold evaluation procedure: what we may call *micro-testing* and *macro-testing*. We also built a tool to verify the consistency of our dictionaries.

6.1 Micro-testing

Micro-testing refers to the evaluation of single rules. For each rule, a set of test structures has been set up. These structures must be as simple as possible, in order to avoid noise, but still designed to cover all the phenomena the rule has to be able to cope with. In most cases, a single rule cannot be tested in isolation; its application depends on the application of some core rules. For instance, it is not possible to test only the construction of a given syntactic relation without also applying the rules that create the nodes linked by the relation. Therefore, it is necessary for evaluation to keep track of rule dependencies. Hence, not only do we associate a set of test structures with each rule, but we also associate each test structure with the set of rules it activates. When a rule is modified, all executed test routines that involved it are reset and run again.[12] Micro-testing thus verifies elementary components separately.

By the very nature of micro-testing, it is difficult to have language-independent test structures. For example, even if one wants to test a generic DSynt $\Rightarrow$ SSynt rule, the input test structure will have to be a DSynt structure, which by definition contains LUs of a specific language. Therefore, only the rules of the Con $\Rightarrow$ Sem module can be micro-tested with language-independent test structures (since our conceptual structures are the same for all languages). However, it is often safe to assume that generic rules tested in one language will work just as well in other languages, though of course prudence must be used. Figure 3 shows a DSynt structure that we used for testing the rules that handle subject-raising verbs. Though this structure uses information from the English dictionary, it tests rules that are generic.

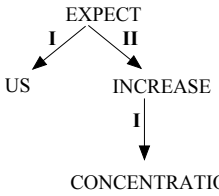


Figure 3: A sample DSynt structure for testing subject-raising

[12] Of course, this process can be automated if we have, for all test structures, the expected result structures (be they manually created from scratch, or result from previous test runs that have been validated by a human).

Phenomena that are described by language-specific rules obviously require language-specific structures for micro-testing. For instance, rules that handle English auxiliaries were tested with a set of nearly identical structures containing only a verb with a subject and an object, where the sole difference was the tense and aspectual information. One structure was created for each possible combination of tense and aspect.

6.2 Macro-testing

Macro-testing refers to a more global evaluation procedure. Its aim is to assess the coverage of the linguistic resources for a more or less specific purpose. The structures used for this task are designed to cover the largest possible range of situations the generation system must be able to handle. The goal here is not to test specific rules, but to make sure that the system can handle the expected input we are going to provide it with. Macro-testing is best applied after micro-testing, as it verifies the interaction between the various components of the grammar. For example, Figure 4 shows a structure that was used for macro-testing.[13] The system is required to produce all expected ways to express this meaning:

> *Between 8 AM and 11 AM, the concentration of sulfur dioxide remained stable at 3.*
> *Between 8 AM and 11 AM, the sulfur dioxide concentration was stable at 3.*
> ...

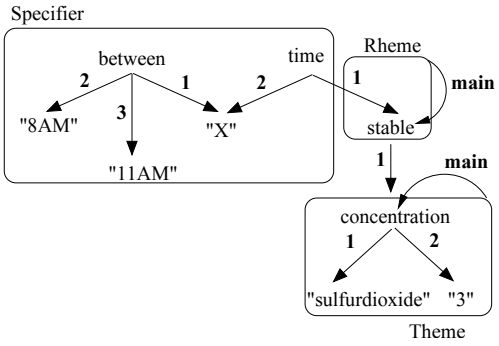


Figure 4: A sample semantic structure for macro-testing

6.3 Dictionary consistency testing

In addition to micro- and macro-testing, it was necessary to make sure that all concepts that might appear in the input structures could be expressed in any lan-

[13] We show here an English semantic structure, though the real input structure is a conceptual structure. The corresponding conceptual structure takes too much space and would be difficult to read, so for the sake of clarity we show only this semantic representation.

guage. Concepts are mapped by the conceptual dictionaries to language-specific semantemes, which in turn are mapped to LUs. These LUs point to prepositions in their sub-categorization patterns, and to other LUs through lexical functions. All these links form a complex network where errors are hard to spot for a human, so we created a small MATE grammar that consisted essentially of simplified lexicalisation rules. This grammar takes as input a list of structures containing one concept each (one structure for every concept expected in the input of the system) and produces structures representing the lexical links encoded by the dictionaries. Then, a set of consistency-check rules is applied to make sure that there is no pointer to non-existent entries, and that each entry contains all the necessary information (for example, that French nouns have a gender, that every LU has a part of speech, that syntactic relation names are specified in the sub-categorization patterns, etc.). If an error is found, an appropriate message is added to the output in the form of extra nodes and relations, as illustrated in Figure 5, where IN is marked as missing in the lexical dictionary. MATE's graphs being encoded as text files, it is easy to scan the output structures for error messages (or have a script do it).

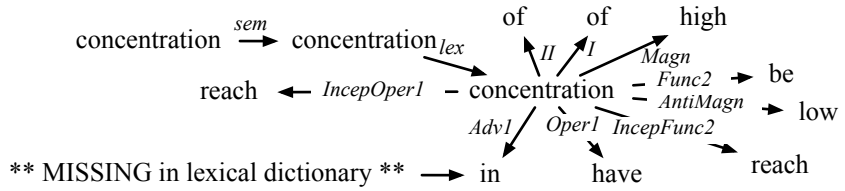


Figure 5: A sample output structure from the lexical consistency check grammar

7 Conclusion

We presented an efficient organization of grammatical resources in an MTT multilingual generation system. This organization follows the principles of sharing, modularization, and inheritance and adopts a strongly lexicalist perspective on the grammatical resources. The implementation of resources for six languages that belong to three different families and their practical use have proven that these principles are valid and allow for the development of large scale grammars.

As part of future work, we plan to extend the resources with respect to both the coverage of linguistic constructions and further languages.

References

Alshawi, Hiyan. 1992. *The Core Language Engine*. Cambridge, MA: The MIT Press.

Avgustinova, Tania and Uszkoreit, Hans. 2000. An ontology of systemic relations for a shared grammar of Slavic. In *Proceedings of the 18th International Conference on Computational Linguistics, COLING*, pages 28–34.

Bateman, John, Kruijff-Korbayova, Ivana and Kruijff, Geert-Jan. 2005. Multilingual resource sharing across both related an unrelated languages: An implemented, open-source framework for practical natural language generation. *Journal for Research on Language and Computation* 3(2), 191–219.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics, COLING*, pages 8–14, Taipei, Taiwan.

Bohnet, Bernd. 2006. *Textgenerierung durch Transduktion linguistischer Strukturen*. Berlin: Akademische Verlagsgesellschaft, DISKI Series.

Bohnet, Bernd, Langjahr, Andreas and Wanner, Leo. 2000. A Development Environment for MTT-Based Sentence Generators. In *Proceedings of the XVI SEPLN Conference*, Vigo, Spain.

Gerdes, Kim and Kahane, Sylvain. 2007. Phrasing It Differently. In L. Wanner (ed.), *Selected Lexical and Grammatical Issues in the Meaning-Text Theory. In honour of Igor Mel'čuk*, pages 297–335, Amsterdam: Benjamins Academic Publishers.

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

Mel'čuk, Igor. 1988. *Dependency Syntax: Theory and Practice*. Albany, NY: SUNY Press.

Mel'čuk, Igor. 1996. Lexical Functions: A Tool for the Description of Lexical Relations in a Lexicon. In L. Wanner (ed.), *Lexical Functions in Lexicography and Natural Language Processing*, pages 37–102, Amsterdam: Benjamins Academic Publishers.

Santaholma, Marianne. 2007. Grammar Sharing Techniques for Rule-Based Multilingual NLP Systems. In *Proceedings of NODALIDA 2007*, pages 253–260.

Sowa, John. 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove: Brooks Cole Publishing Company.

Test-suite Construction for a Spanish Grammar

Montserrat Marimon, Núria Bel and Natalia Seghezzi
Institut Universitari de Lingüística Aplicada
Universitat Pompeu Fabra

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes the testing component we use for the development and maintenance of the Spanish Resource Grammar, an open-source multi-purpose broad-coverage HPSG grammar for Spanish implemented within the LKB system. Following a brief description of the main features of the grammar, we describe the set of test suites we have manually constructed and the way we have extended them with publicly available data with the aim of producing better resources for testing our grammar.

1 Introduction

Natural language is a system of rather complex interactions. Grammar writing, thus, requires broad and systematic testing to be successful in both research and industrial contexts.

This paper describes work on the development and maintenance of the testing component for a multi-purpose broad-coverage precise grammar for Spanish implemented within the LKB system, the Spanish Resource Grammar. On the one hand, for the development of a multi-purpose grammar, the linguistic phenomena included in the testing component should be abstracted away from any particular application.[1] On the other hand, broad-coverage grammar writing requires testing material which not only includes all variations of a particular phenomenon, but also reflects the real world language complexity; thus, in addition to traditional controlled test items, test data should include examples that present combinations of different phenomena.

We see the development and maintenance of the testing module as part of the process of grammar writing. The idea behind evaluation is to determine its usability for different applications. The grammar can also be evaluated in terms of recall (i.e. coverage) and precision (i.e. overgeneration) with large natural language corpora.

The rest of the paper is organized as follows. Section 2 presents the main features of the grammar. Then, section 3 describes the set of test suites we have manually constructed. In section 4 we explain the way we have extended them with publicly available data with the aim of producing better resources for testing our grammar.[2]

[†]This research was suported by the Spanish *Ministerio de Educación y Ciencia* under the programmes *Ramón y Cajal* and *Juan de la Cierva*.

[1]In previous experience in grammar writing – in the *Advanced Linguistic Engineering Platform* platform – during the European projects LS-GRAM (LRE-61029), MELISSA (ESPRIT-22252) and IMAGINE (IST-2000-29490), the test data we used to define (and refine) the coverage of the grammar was designed according to the user need analysis; consequently, breath and depth of grammar coverage was defined by the applications.

[2]The test data we describe may be downloaded from: http://www.upf.edu/pdi/iula/montserrat.marimon/

2 The Spanish Resource Grammar

The Spanish Resource Grammar (SRG) is an open-source[3] multi-purpose large-coverage grammar for Spanish.

The grammar is grounded in the theoretical framework of HPSG (*Head-driven Phrase Structure Grammar* (Pollard and Sag, 1987, 1994)). HPSG is a constraint-based, lexicalist approach to grammatical theory where all linguistic objects are represented as typed-feature structures. The grammar uses *Minimal Recursion Semantics* (MRS) for the semantic representation. MRS is a flat approach to semantic representation for large-coverage linguistically-motivated computational grammars of natural language that can be used for both parsing and generation (Copestake et al., 2006). The SRG is implemented within the *Linguistic Knowledge Builder* (LKB) system (Copestake, 2002), based on the basic components of the grammar Matrix,[4] an open-source starter-kit for the development of HPSG grammars developed as part of the LinGO consortium's multilingual grammar engineering (Bender et al., 2002; Bender and Flickinger, 2005).

The SRG has a full coverage lexicon of closed word classes (pronouns, determiners, prepositions and conjunctions) and it contains about 50,000 lexical entries for open classes (6,600 verbs, 28,000 nouns, 11,200 adjectives and 4,000 adverbs). These lexical entries are organized into a type hierarchy of about 400 leaf types (defined by a type hierarchy of around 5,500 types). The grammar also has 50 lexical rules to perform valence changing operations on lexical items (e.g. movement and removal of complements), and 150 phrase structure rules to combine words and phrases into larger constituents and to compositionally build up the semantic representation.

The range of linguistic phenomena that the grammar handles includes: all types of subcategorization structures, surface word order variation and valence alternations, subordinate clauses, raising and control, determination, null-subjects and impersonal constructions, compound tenses, modification, passive constructions, comparatives and superlatives, cliticization, relative and interrogative clauses, sentential adjuncts, negation, and coordination among others. Appendix A includes a more detailed (though not complete) list of the phenomena covered by the grammar.

Following previous experiments within the *Advanced Linguistic Engineering Platform* (ALEP) platform (Marimon, 2002), we have integrated a shallow processing tool, the FreeLing tool, as a preprocessing module of the grammar with the aim of improving both coverage and robustness. The FreeLing tool is an open-source[5] language analysis tool suite performing shallow processing functionalities which include: text tokenization (includ-

[3]The Spanish Resource Grammar may be downloaded from: http://www.upf.edu/pdi/iula/montserrat.marimon/

[4]http://www.delph-in.net/matrix/

[5]The FreeLing tool may be downloaded from http://www.lsi.upc.edu/nlp/freeling

ing MWU and contraction splitting), sentence splitting, morpho-syntactic analysis and disambiguation, proper name detection and classification, date / number / currency / ratios / physical magnitude (speed, weight, temperature, density, etc.) recognition, chart-based shallow parsing, WordNet-based sense annotation and dependency parsing[6] (Atserias et al., 2006). The integration of FreeLing allows us to release the parser from certain tasks (i.e. morphological analysis and recognition and classification of special text expressions, e.g. numbers, dates, percentages, currencies, proper names, etc.) that may be reliably dealt with by shallow external components. Our hybrid architecture also permits the implementation of default lexical entry templates for unknown words for virtually unlimited lexical coverage (Marimon et al., 2007).

We are also investigating Machine Learning (ML) methods applied to the acquisition of the information contained in the lexicon of the SRG (Bel et al., 2007; Marimon et al., 2007). The automatic acquisition of lexical information is a very active area of research. It is specially important for deep linguistic analysis due to the central role that lexical information has in lexicalized grammars and the cost of hand-crafting them (Korhonen, 2002; Carroll and Fang, 2004; Baldwin, 2005; Blunsom and Baldwin, 2006; Zhang and Kordoni, 2006). The most successful systems of lexical acquisition are based on the linguistic idea that the contexts where words occur are indicative of the particular types of words. Although the methods used are different, most of the systems work upon the syntactic information of words as collected from a corpus, and they develop different techniques to decide whether this information is relevant for type assignment or whether it is noise. In the LKB system, lexical types are defined as a combination of grammatical features. For our research, we have worked with morpho-syntactically motivated features. Thus, words are assigned a number of features, the combination of which will help in defining the particular lexical type the word belongs to.

The SRG is part of the DELPH-IN open-source repository of linguistic resources and tools for writing (the LKB system), testing and benchmarking (the [incr tsbd()] competence and performance profiler (Oepen and Carroll, 2000)) and efficiently processing HPSG grammars (the PET system (Callmeier, 2000)), as well as an architecture for integrating deep and shallow natural language processing components to increase robustness of HPSG grammars (the Heart of Gold (Schäfer, 2007)). Further linguistic resources that are available in the DELPH-IN repository include broad-coverage grammars for English, German and Japanese as well as smaller

[6] FreeLing also includes a guesser to deal with words which are not found in the lexicon by computing the probability of each possible PoS tag given the longest observed termination string for that word. Smoothing using probabilities of shorter termination strings is also performed. Details can be found in (Brants, 2000; Samuelsson, 1993).

grammars for French, Korean, modern Greek, Norwegian and Portuguese.[7]

3 Hand-built Test Suites

Together with the linguistic resources (grammar and lexicon) the SRG includes a set of test suites. A test suite is a hand-constructed collection of test cases, e.g. sentences, that exemplify the grammatical – and related ungrammatical – constructions that the grammar should parse, or not, in the case of the ungrammatical ones.

The construction and maintenance of the test suites plays a major role in the development of the SRG.[8] Test suites provide a fine-grained diagnosis of the grammar behaviour in terms of coverage, overgeneration and efficiency when we change and/or extend the grammar components. To determine that the output produced is correct we have to inspect it manually. Test suites also allow us to compare the SRG with other DELPH-IN grammars. Comparison with other DELPH-IN grammars, e.g. English Resource Grammar and *La Grenouille* (i.e. the French Resource Grammar), is done by producing parallel test data (i.e. data that covers the same phenomena) and comparing the outputs at the MRS level.

In building the test suites, we followed the guidelines for test suite construction and maintenance of the TSNLP project (LRE-62-089) to meet the demands for systematicity and exhaustivity (i.e. systematic increase in depth of coverage), and control over data (i.e. control of interaction of phenomena and ambiguity). Thus,

- We test linguistic phenomena in isolation or in controlled interactions. Most of our test cases include a single grammatical phenomenon in each test sentence.

- Starting from simple test items and increasing their complexity progressively (e.g. in (1), where we show the positive test items we have created to test the non-universal quantifier/adjective *poco* (few)), we provide test cases which show systematic and exhaustive variations over each phenomenon, including infrequent phenomena and variations recognized as linguistically interesting but which do not occur commonly in corpora.

[7] See http://www.delph-in.net/

[8] Note that there are no standard general purpose test suites publicly available for Spanish – like the Hewlett Packard (HP) test suite for English (Flickinger et al., 1987), the DITO test suite for German (Nerbonne et al., 1991), or the TSNLP for English, French and German (Lehmann et al., 1996; Oepen et al., 1997) – we could use. Our test suites have been primarily aimed at the SRG, in that some of the test data has been designed to test its linguistic modules. Nevertheless, most test data reflect central language phenomena, and this makes them adequate and reusable in other parsing systems. It is hoped that the availability of this testing material will be of value to the NLP community.

(1)

a. *Pocos muchachos lloran.* (A few boys cry.)

b. *Otros pocos muchachos lloran.* (Other few boys cry.)

c. *Pocos otros muchachos lloran.* (Few other boys cry.)

d. *Los pocos muchachos lloran.* (The few boys cry.)

e. *Los pocos otros muchachos lloran.* (The few other boys cry.)

f. *Todos los pocos otros muchachos lloran.* (All the few other boys cry.)

g. *Casi todos los pocos otros muchachos lloran.* (Almost all the few other boys cry.)

h. *Muy pocos otros muchachos lloran.* (Very few other boys cry.)

- We avoid irrelevant variation (i.e. different instances of the same lexical type or same syntactic structure) and both structural and lexical ambiguity. Note that even the simplest sentences may turn to be ambiguous as the grammar coverage increases. For example, when testing the Spanish definite articles, a sentence like (2.a) becomes ambiguous when dealing with elliptical constructions, as we show in (2.b) and (2.c).

(2)

a. *Los chicos lloran.*

b. DET NOUN VERB (The boys cry.)

c. DET ADJ VERB (The small (ones) cry.)

- We include negative or ungrammatical data to check both overgeneration and coverage. Following the TSNLP, negative cases are derived from well-formed ones by one of the following operations:
  - replacement, e.g. change of agreement features (**ambos* **muchacho** *lloran* (both boys cry)), change of mood (**quiero que* **vienes** ((I) want that (you) come)), change of marking preposition (**el muchacho desertó* **desde** *su regimiento* (the boy deserted from his regiment)), change of copular verb (**los muchachos* **son** *contentos* (the boys are happy)).
  - addition, e.g. of subject in impersonal constructions (***el cielo** *llueve* (the sky rains)).
  - deletion, e.g. of an obligatory complement (**los muchachos fabrican 0* (the boys produce)), of an obligatory complementizer (**las muchachas intentaron 0 los muchachos lloraran* (the girls tried the boys cried)).

– permutation, e.g. inversion of word order (*__tres unos__ *muchachos lloran* (three about boys cry), *__los todos muchachos__ *lloran* (the all boys cry)).

Test cases have been divided by linguistic phenomena. We currently use 16 files testing linguistic structures plus one file for special text constructions. Test cases include a short annotation describing the phenomenon that we are actually testing and the number of expected results when ambiguity cannot be avoided (e.g. when we test optionality). Note that even ungrammaticality may be due to different reasons, for example the ungrammatical sentence in (3) may be derived by removing the definite article under the reading where *todo* (all) is a definite quantifier, or by changing the agrement features under the reading where *todo* is an indefinite quantifier, in which case it co-occurs with singular nouns.

(3) **todos muchachos lloran.* (all boys cry.)

Table 1 shows the set of test suites of the grammar with the number of test items that each contains.

Test suites	**Phenomenon**	**Number of items**
t01_basic_subcat	basic subcategorization structures for verbs, nouns (and pronouns) and adjectives	99
t02_null_subj	pro-drop and impersonal verbs	6
t03_det	determiners/quantifiers	32
t04_val_alt	surface word order variation and valence alternations	10
t05_cl_comp	finite/non-finite completive clauses and indirect questions	77
t06_rais_cntrl	raising and control	26
t07_aux	compound tenses	7
t08_pass	passive constructions	7
t09_mod	basic modifiers	58
t10_compar	comparatives and superlatives	13
t11_sent_mod	sentential adjuncts	3
t12_rel_cl	relative clauses	77
t13_ques	interrogative clauses	30
t15_se_constr	impersonal and passive constructions with *se*	20
t16_clitics	clitics	71
t17_coord	coordination	60
txx_messy_details	special text constructions	53

Table 1: Hand-built test suites for the SRG.

4 Extending the Test Suites

Test suites have traditionally been used to test linguistically-motivated computational grammars.[9] Though simple, test cases included in hand-built test suites are crucial to determine progress in grammar development.

Controlled hand-built test suites are certainly necessary for incremental grammar maintenance and development to detect unintended interactions of extensions and/or changes in the linguistic resources that cause the treatment of some phenomena already covered to deteriorate. However, from the point of view of building a large-coverage grammar, test data that shows the real world language complexity is also necessary. Therefore, test cases that present combination of different phenomena should also be included in the testing module.

Combining all different phenomena could lead to a combinatorial explotion; besides, not every combination of phenomena produces grammatical sentences or shows interesting cases. Instead, we have re-used available data reflecting natural combination of phenomena.

As the coverage of our grammar increased, hand-constructed sentences were complemented by real corpus cases we took from:

a. the Spanish questions from the Question Answering track at CLEF (CLEF-2003, CLEF-2004, CLEF-2005 and CLEF-2006). We built up a test suite with 619 test items we took from the 800 available sentences. We only removed those sentences which only differed in a proper name.

b. the general sub-corpus of the Corpus Tècnic de l'IULA (IULA's Technical Corpus; (Cabré and Bach, 2004)); this sub-corpus includes newspaper articles and it has been set up for contrastive studies. We built up a test suite with some of the articles that we chose randomly.

CLEF cases include short queries, sentences and a few NPs showing none or very little combination of phenomena, and an average of 9.2 words. Most of these test cases include core linguistic phenomena, e.g. verbs with only one complement (DO, attribute), passives with *ser* and *estar*, impersonal constructions, comparatives and superlatives, and basic nominal and verbal modifiers (i.e. APs, PPs, temporal NPs), and we find just a few examples of more complex structures such as relatives clauses, coordination or ellipsis. Very rarely more than two or three different phenomena appear in the same sentence.

Newspaper articles include more interesting and challenging linguistic structures showing a high level of syntactic complexity due to the combination of several phenomena in a sentence. Sentences are longer, ranging up to 35 words.

[9] Other testing mechanisms are briefly described in (Butt and King, 2003).

The parsing of this new data displayed unanticipated analyses showing errors/deficiencies not only in our linguistic modules (grammar rules and lexical entries were not restrictive enough to exclude some ungrammatical examples which had not been considered), but also in the FreeLing tool (and, for example, we realized that the FreeLing tool allowed enclitics to appear on all verbal forms; in Spanish, clitics can only be attached to imperatives, gerunds and infinitives).

We are currently shifting to much more varied corpus data, and we are extending the test suites with more specialized tests (these have also been chosen randomly) from the *Corpus Tècnic de l'IULA*. This includes specialized corpora of written text in the areas of computer science, environment, law, medicine and economics, collected from several sources, such as legal texts, textbooks, research reports, user manuals, etc. In these texts sentence length may range up to 100 words. In addition, this corpus contains highly specialized words which must be added to the lexicon.

5 Conclusions

We have presented the set of test suites we have manually constructed for the development and maintenance of an open-source multi-purpose broad-coverage HPSG grammar for Spanish, and the way we have extended them with publicly available data with the aim of producing better resources for testing our grammar. Even tough our test suites have been primarily aimed at the SRG, most test data reflect central language phenomena, and this makes them adequate and reusable in other parsing systems. It is hoped that the availability of this testing material will be of value to the NLP community.

A Grammar Coverage

List of linguistic phenomena that the SRG handles. Note that we have not included all variants of the phenomena.

- main clauses with canonical word order, i.e. SVO.
- subcategorization structures: unaccusative verbs (*nacieron* ((they) were born), *viven en la ciudad* ((they) live in the town)), intransitive verbs (*claudicaron* ((they) gave in), *desertó del regimiento* ((s/he) deserted from the regiment), *me gusta el muchacho* (I like the boy)), transitive verbs (*fabrican juguetes* ((they) make toys), *abastecieron la ciudad de víveres* ((they) provided the town with provisions), *colgaron los cuadros en el salón* ((they) hang the paintings in the living-room)), *acercó la sal a la muchacha* ((s/he) brought over the salt to the girl)), quantifying nouns (*el grupo de los muchachos* (the group of boys)),

argument taking nouns (*el padre de la muchacha* (the father of the girl), *el apoyo de los muchachos a la muchacha* (the boys' support of the girl)), transitive adjectves (*es atento con los muchachos* ((s/he) is kind to the boys)).

- null-subjects: pro-drop (*lloraremos* ((we) will cry)) and impersonal verbs (*llueve* ((it) rains), *hay muchos muchachos* (there are a lot of boys)).

- determination: def/indef. articles (*el/un muchacho llora* (the/a boy cries)), demonstratives (*estos muchachos lloran* (these boys cry)), possessives (*mis muchachos lloran* (my boys cry)), quantifiers (*todos los/ algunos/muchos muchachos lloran* (all of the/some/many boys cry)).

- surface word order variation: subject-predicate inversion (*en ese parque anidan pájaros* (in this park nest birds)), complement permutation (*el muchacho acercó a la muchacha la sal* (the boy brought the girl the salt)).

- subordinate clauses (finite and non-finite), e.g. *quieren que lloren* ((they) want that (they) cry), *la ventaja de claudicar/que claudicaran* (the advantage of giving in/that (they) gave in), *está segura de que lloraron* ((she) is sure that (they) cried)).

- indirect questions (finite and non-finite), e.g. *preguntó cómo ir/que cuándo claudicaron* ((s/he) asked how to go/when (they) gave in), *la incógnita de si claudicarán* (the question of whether (they) will give in), *está seguro de dónde fue/ir* ((he) is sure about where (s/he) went/to go).

- raising and control verbs (subj-control (*intentaremos claudicar* ((we) will try giving in)), obj-control (*me gusta llorar* (I like crying)), subj-to-subj raising (*deberían claudicar* ((they) should give in)), perception (*vieron a la muchacha llorar* ((they) saw the girl crying))), and adjectives (subj-control (*es capaz de claudicar* ((s/he) is able to give in)), subj-rasing (*es libre de claudicar* ((s/he) is free to give in)), and obj-raising (*el es fácil de querer* ((s/he) is easy to love))).

- compound tenses, e.g. *hemos llorado* ((we) have cried), *estamos llorando* ((we) are crying), *hemos estado llorando* ((we) have been crying), *fue invadido* ((it) was invaded), *ha sido invadido* ((it) has been invaded), *está siendo invadido* ((it) is been invaded), *ha estado siendo invadido* ((it) has been being invaded).

- passive constructions: with *ser*, with and without optional PPpor (i.e. by-agent) (see example above); and with *estar* (*el país está invadido* (the country is invaded)).

- constructions with *se*: impersonal constructions (*se invadió el país* (the country has been invaded)), passive constructions (*los países se invaden* (the countries are invaded)) and unaccusative constructions (*el tren se abre* (the train opens)).

- modification

  - verbal modifiers: PPs (*corrió desde el parque/ahí* ((s/he) ran from the park/there)), adverbs (*quizás claudicarán* (maybe (they) will give in), *claudicarán pronto* ((they) will give in soon)), temponal NPs (*correrá el lunes*, ((s/he) will run on Monday)).
  - nominal modifiers: APs (*el muchacho alto murió* (the tall boy died)), PPs (*el muchacho de ahí llora* (the boy from there cries)), participles (*el país invadido* (the invaded country)), proper names (*mi amigo Juan llora* (my friend Juan cries)), adverbs (*fabrican sólo juguetes* ((they) make only toys)).
  - adjectival modifiers: adverbs (*es muy guapo* ((he) is very pretty), PPs (*fiel hasta la muerte* (loyal to death)).
  - adverbial modifiers: adverbs (*corre sólo muy raramente* ((s/he) runs only very occasionaly)).
  - prepositional modifiers: adverbs (*corrió sólo por el parque* ((s/he) ran only along the park)), PPs (*lloró desde el aeropuerto hasta la ciudad* ((s/he) cried from the airport to the town)).

- comparatives, e.g. *es mejor* ((s/he) is better), *es más listo* ((he) is cleverer), *tiene tantos libros como revistas* ((s/he) has as many books as journals); and superlatives, e.g. *es el mejor muchacho* ((he) is the best boy).

- clitics: complement cliticization (*los abrió* ((s/he) opened them)), clitic doubling (*a él le gusta el muchacho* (he likes the boy)), clitic climbing (*los ha comido* ((s/he) has eaten them)).

- relative clauses: restrictive RC (*el hombre que claudicó llora* (the man that gave away is crying)), non-restrictive RC (*el chico, con cuyos padres cuento, claudicará* (the boy, whose parents I count on, will give in)), free RC (*yo vivo donde tú vives* (I live where you live)), semi-free RC (*el que claudica claudica* (who gives in gives in)).

- interrogative clauses: polar-questions (*Quieres libros?* (do you want books?)), wh-questions (*con quién cuentas?* (whom do you count with?), *quién tiene qué?* (who has what?)).

- sentential adjuncts, e.g. *las muchachas lloraron porque los muchachos claudicaron* (the girls cried because the boys gave in), *el muchacho corrió hasta morir* (the boy run to die).

- negation, e.g. *los muchachos no claudicaron* (the boys didn't give in).
- coordination of all major categories: binary (*fabrica coches y juguetes* ((s/he) makes cars and toys), *es guapo y listo* ((he) is pretty and clever)), multiple (*es guapo, guapo y guapo* ((he) is pretty, pretty and pretty)), doubled conjunctions (*no sólo guapo sino listo* (not only pretty, but also clever)), unlike categories (*habló alta y claramente* ((s/he) talked loud and clearly), *habló claramente y sin parar* ((s/he) talked clearly and without stopping)).

References

Atserias, Jordi, Casas, Bernardino, Comelles, Elisabet, González, Meritxell, Padró, Lluís and Padró, Muntsa. 2006. FreeLing 1.3: Syntactic and semantic services in an open-source NLP library. In *Proceedings of the 5th International Conference on Language Resources and Evaluation LREC'06*, Genoa, Italy.

Baldwin, Timothy. 2005. Bootstrapping Deep Lexical Resources: Resources for Courses. In *Proceedings of Workshop on Deep Lexical Acquisition, ACL-SIGLEX 2005*, Ann Arbor, Michigan, USA.

Bel, Núria, Espeja, Sergio and Marimon, Montserrat. 2007. Automatic Acquisition of Grammatical Types for Nouns. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Rochester, NY, USA.

Bender, Emily M. and Flickinger, Dan. 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 M., Flickinger, Dan and Oepen, Stephan. 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 at the 19th International Conference on Computational Linguistics COLING-02*, Tapei, Taiwan.

Blunsom, Phil and Baldwin, Timothy. 2006. Multilingual deep lexical acquisition for HPSGs via supertagging. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia.

Brants, Thorsten. 2000. TnT: A statistical part-of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing*, Seattle, USA.

Butt, Miriam and King, Tracy H. 2003. Grammar Writing, Testing, and Evaluation. In A. Farghaly (ed.), *Handbook for Language Engineers*, pages 129–179, CSLI Publications.

Cabré, Teresa and Bach, Carme. 2004. El corpus tècnic del IULA: corpus textual especializado plurilingüe. *Panacea, V. 16* pages 173–176.

Callmeier, Ulrich. 2000. Pet a platform for experimentation with efficient HPSG processing. In D. Flickinger, S. Oepen, J. Tsujii and H. Uszkoreit (eds.), *Natural Language Engineering (6)1 —Special Issue: Efficiency Processing with HPSG: Methods, Systems, Evaluation*, pages 99–108, Cambridge University Press.

Carroll, John and Fang, Alex C. 2004. The automatic acquisition of verb subcategorisations and their impact on the performance of an HPSG parser. In *Proceedings of the 1st International Joint Conference on Natural Language Processing IJCNLP-04*, Sanya City, China.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. Chicago: CSLI Publications, CSLI lecture notes, number 110.

Copestake, Ann, Flickinger, Dan, Pollard, Carl J. and Sag, Ivan A. 2006. Minimal Recursion Semantics: an Introduction. *Research on Language and Computation* 3(4), 281–332.

Flickinger, Dan, Nerbonne, John, Sag, Ivan A. and Wassow, Thomas. 1987. Toward evaluation of NLP systems. In *Technical Report, Hewlett-Packard Laboratories. Distributed at the 24th Annual Meeting of the Association for Computational Linguistics ACL-87*, Stanford, USA.

Korhonen, Anna. 2002. *Subcategorization acquisition. A Technical Report UCAM-CL-TR-530*. UK: University of Cambridge.

Lehmann, Sabine, Oepen, Stephan, Regnier-Prost, Sylvie, Netter, Klaus, Lux, Veronika, Klein, Judith, Falkedal, Kirsten, Fouvry, Frederik, Estival, Dominique, Dauphin, Eva, Compagnion, Herve, Baur, Judith, Balkan, Lorna and Arnold, Doug. 1996. TSNLP: test suites for natural language processing. In *Proceedings of the 16th International Conference on Computational Linguistics COLING-96*, Copenhagen, Denmark.

Marimon, Montserrat. 2002. Integrating shallow linguistic processing into a unification-based Spanish grammar. In *Proceedings of the 19th International Conference on Computational Linguistics COLING-02*, Taipei, Taiwan.

Marimon, Montserrat, Bel, Núria, Espeja, Sergio and Seghezzi, Natalia. 2007. The Spanish Resource Grammar: Pre-processing Strategy and Lexical Acquisition. In *Proceedings of the Workshop on Deep Linguistic Processing, ACL-2007*, Prague, Czech Republic.

Nerbonne, John, Netter, Klaus, Diagne, Abdel K., Dickmann, Ludwig and Klein, Judith. 1991. A Diagnostic Tool for German Syntax. In J. G. Neal and S. M. Walter (eds.), *Natural Language Processing Systems Workshop: Final Technical Report RL-TR-91-362*, New York: Rome Laboratory.

Oepen, Stephan and Carroll, John. 2000. Performance Profiling for Parser Engineering. In D. Flickinger, S. Oepen, J. Tsujii and H. Uszkoreit (eds.), *Natural Language Engineering (6)1 —Special Issue: Efficiency Processing with HPSG: Methods, Systems, Evaluation*, pages 81–97, Cambridge University Press.

Oepen, Stephan, Netter, Klaus and Klein, Judith. 1997. TSNLP: test suites for natural language processing. In J. Nerbonne (ed.), *Linguistic Databases*, CSLI Publications, CSLI lecture notes.

Pollard, Carl J. and Sag, Ivan A. 1987. *Information-Based Syntax and Semantics. Volume 1: Fundamentals*. Stanford: CSLI Publications.

Pollard, Carl J. and Sag, Ivan A. 1994. *Head-driven Phrase Structure Grammar*. Chicago: The University of Chicago Press and CSLI Publications.

Samuelsson, Christer. 1993. Morphological tagging based entirely on Bayesian inference. In *Proceedings of 9th Nordic Conference on Computational Linguistics*, Stockholm, Sweden.

Schäfer, Ulrich. 2007. *Integrating Deep and Shallow Natural Language Processing Components – Representations and Hybrid Architectures*. Ph. D.thesis, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany.

Zhang, Yi and Kordoni, Valia. 2006. Automated deep lexical acquisition for robust open text processing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation LREC'06*, Genoa, Italy.

Towards Framework-Independent Evaluation of Deep Linguistic Parsers

Yusuke Miyao[1] Kenji Sagae[1] Jun'ichi Tsujii[1,2,3]
[1]Department of Computer Science
University of Tokyo
[2]School of Computer Science
University of Manchester
[3]National Center for Text Mining

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper describes practical issues in the framework-independent evaluation of deep and shallow parsers. We focus on the use of two dependency-based syntactic representation formats in parser evaluation, namely, Carroll et al. (1998)'s *Grammatical Relations* and de Marneffe et al. (2006)'s *Stanford Dependency* scheme. Our approach is to convert the output of parsers into these two formats, and measure the accuracy of the resulting converted output. Through the evaluation of an HPSG parser and Penn Treebank phrase structure parsers, we found that mapping between different representation schemes is a non-trivial task that results in lossy conversions that may obscure important differences between different parsing approaches. We discuss sources of disagreements in the representation of syntactic structures in the two dependency-based formats, indicating possible directions for improved framework-independent parser evaluation.

1 Introduction

Despite the rapid progress made in recent years on deep linguistic parsing (Cahill et al., 2002; Hockenmaier, 2003; Kaplan et al., 2004; Burke et al., 2004b; Clark and Curran, 2004; Malouf and van Noord, 2004; Oepen et al., 2004; Toutanova et al., 2004; Miyao and Tsujii, 2005), shallow phrase-structure parsers (Collins, 1997; Charniak and Johnson, 2005) are still often chosen over linguistically richer approaches in natural language processing (NLP) research and applications where syntactic analysis is needed. This is due in part to the perception that deep parsing is not robust and efficient enough for handling practical tasks, and that its accuracy is below that of shallow parsing approaches. In addition, the advantages of deep syntactic analysis over shallow phrase-structures, although clear to those in the deep parsing community, has not been demonstrated convincingly to the general NLP community. While shallow parsers may in fact be better suited for some NLP tasks, an informed decision on that regard requires a fair comparison between different kinds of parsers, especially when they deal with different ways of representing syntactic information. However, comparison of different parsing approaches is challenging even among deep parsers, since accuracy measurements used in different systems are largely incompatible, making it difficult to determine the advantages of specific deep parsing approaches. Meanwhile, the most widely used evaluation metric in current parsing research, precision and recall of labeled brackets, follows a view of syntax that is simplistic and at the same time quite specific to one particular type of syntactic representation. While the use of bracketing precision and recall in simplified trees from the Penn Treebank (Marcus et al., 1994) fueled much of the development of current wide-coverage data-driven parsing by providing a way to evaluate parsers on a common test set, it is now too

[†]We thank John Carroll for providing the gold-standard GR data, and for numerous insightful discussions and comments. This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and Grant-in-Aid for Young Scientists (MEXT, Japan).

limited to deal with recent developments that go beyond what is represented in that test set[1]. Hence, framework-independent parser evaluation is necessary not only for informed development of NLP applications, where different types of parsers may be more or less suited for certain NLP tasks (Clegg and Shepherd, 2007), but also for progress in parsing research itself, where it would allow for a more direct comparison between different parsing approaches (Clark and Curran, 2007).

This paper discusses several challenges and practical issues in framework-independent evaluation of syntactic parsers. Specifically, we focus on two existing proposals for representing syntactic relationships between words, and examine practical issues through the evaluation of parsing accuracy of a deep parser based on Head-Driven Phrase Structure Grammar (HPSG), *Enju* (Miyao and Tsujii, 2005). The first representation scheme we consider is *Grammatical Relations* (GR) (Carroll et al., 1998; Carroll and Briscoe, 2004; Briscoe, 2006), which aims to provide a better parser evaluation framework than PARSEVAL measures (Black et al., 1991) of constituent bracketing precision and recall. This scheme has been adopted for the evaluation of RASP (Briscoe and Carroll, 2006; Briscoe et al., 2006), shallow parsers derived from Penn Treebank (PTB) (Preiss, 2003), and recently, a deep parser based on Combinatory Categorial Grammar (CCG) (Clark and Curran, 2007). The other is the *Stanford Dependency* scheme (SD) (de Marneffe et al., 2006), which was proposed for providing NLP applications with more useful syntactic representations than phrase structures. Although gold standard data is not available, a program attached to the Stanford parser (Klein and Manning, 2003) automatically converts PTB-style phrase structures into this format. This conversion is only approximate, making SD-based evaluation problematic. In addition, the lack of detailed documentation on the specific syntactic representation choices highlights that this format was not originally intended for parser evaluation. However, because of its recent use in the evaluation of shallow PTB-style parsers in the biomedical domain (Clegg and Shepherd, 2007; Pyysalo et al., 2007a), and the availability of a conversion tool that uses shallow PTB-style trees[2] as input, we investigate the use of SD as a scheme for framework-independent parser evaluation.

Our basic strategy for the evaluation of Enju is to establish a program for converting Enju's output into these two formats, and measure accuracy of converted output. We also develop a conversion program from SD to GR, which allows for GR-based evaluation of PTB-style parsers (Collins, 1997; Charniak, 2000), since a conversion tool from shallow PTB-style output to SD is available. We can therefore compare the performances of Enju and shallow PTB parsers directly, in addition to previously reported results for RASP (Briscoe and Carroll, 2006; Briscoe et al., 2006) and the C&C CCG parser (Clark and Curran, 2007).

One might expect that format conversion is straightforward among GR, SD,

[1]By "test set" we refer to the set of shallow brackets extracted from the Penn Treebank data. While the original treebank data includes richer syntactic analyses, information such as long-distance dependencies, ellipsis, and functional tags are removed in the extraction of shallow brackets.

[2]We use "shallow PTB-style trees" to refer to the Penn Treebank trees with empty-nodes and function-tags removed.

and Enju's output, because they all represent labeled dependencies between words and are similar in concept. In fact, however, our experiments revealed that format conversion is not trivial. We had to implement complex mapping rules for Enju-to-GR/SD and SD-to-GR conversion, and there remain a lot of disagreements for which resolution is unlikely and which may obscure not just differences in performance among individual parsers, but also differences in the strengths of general parsing approaches.

The idea of parser evaluation across frameworks is not new, and its difficulty has been reported repeatedly in the literature (Carroll et al., 1998; Kaplan et al., 2004; Burke et al., 2004a; Clark and Curran, 2007). The results in this paper add to this discussion by focusing on actual challenges in format conversion, providing in-depth analyses of sources of format disagreements. It is our hope that such work will provide the direction for the development of a better scheme for framework-independent evaluation of deep and shallow parsers.

Section 2 presents an overview of the two schemes for parser evaluation. Section 3 describes methods for conversion from Enju's output to GR/SD, and from SD to GR. Section 4 shows experimental results on the accuracy evaluation of Enju, PTB parsers, RASP, and a CCG parser. Section 5 discusses sources of difficulties in format conversion.

2 Parser Evaluation Schemes

In the context of wide-coverage deep parsing, the de facto standard metric for parsing accuracy is precision/recall of labeled dependency relations such as predicate argument dependencies (Kaplan et al., 2004; Clark and Curran, 2004; Miyao and Tsujii, 2005). However, dependency relations used to evaluate different parsers are based on each parser's formalism and resources. For example, the PARC 700 Dependency Bank (King et al., 2003) was used for the evaluation of LFG parsers (Kaplan et al., 2004; Burke et al., 2004a), a CCG treebank (CCGBank) (Hockenmaier and Steedman, 2002) was used for the evaluation of CCG parsing models (Hockenmaier, 2003; Clark and Curran, 2004), and HPSG treebanks, which were created manually (Oepen et al., 2004) or derived from PTB data (Miyao et al., 2005), were used for the evaluation of HPSG parsers (Toutanova et al., 2004; Miyao and Tsujii, 2005; Ninomiya et al., 2007; Sagae et al., 2007). Direct relationships among different dependency schemes are unclear, and we have no way for fair comparison of these parsers.

One issue that must be considered in parser evaluation is that an evaluation scheme must represent information needed by applications and cover real-world texts, because the goal of parser development is usability in NLP applications. Another important issue is that the evaluation framework should account for syntactic structures that are not tied specifically to any single formalism. For example, an evaluation scheme should be sensitive to grammatical phenomena such as control/raising and long-distance dependencies, even though such structures are not

```
(ncsubj market They _)
(iobj market on)
(dobj market cable-TV)
(dobj on opportunities)
(det opportunities the)
(ncmod _ opportunities grazing)
(cmod _ opportunities seeks)
(ncsubj seeks CNN _)
(ncsubj discourage CNN _)
(dobj discourage opportunities)
(xcomp to seeks discourage)
(ncmod _ opportunities very)
```

Figure 1: GR annotation for *They market cable-TV on the very grazing opportunities CNN seeks to discourage.*

accounted for in shallow PTB parsers. At the same time, the inclusion of such syntactic phenomena must not make it unnecessarily difficult to evaluate parsers that output shallow brackets.

Considering these issues, we focus on two dependency-based schemes, *Grammatical Relations* (GR) (Carroll et al., 1998; Carroll and Briscoe, 2004; Briscoe, 2006) and the *Stanford Dependency* (SD) scheme (de Marneffe et al., 2006), which were proposed outside the deep parsing community, while aiming to represent not only surface syntactic structures but also deep structures such as long distance dependencies. In what follows, we describe these schemes and compare them briefly.

2.1 Grammatical Relations (GR)

The *Grammatical Relation* scheme (GR) was proposed aiming at a framework-independent metric for parsing accuracy (Carroll et al., 1998). A set of 700 sentences extracted from Section 23 of the Penn Treebank (the same set as the PARC 700 Dependency Bank) was manually annotated and made publicly available as gold standard data (Briscoe and Carroll, 2006), in addition to an older set of 500 sentences from the SUSANNE corpus[3]. While this evaluation scheme is not as widely used as PARSEVAL, it has recently gained some traction as a more framework-independent alternative, and has been used in the evaluation of parsers including RASP (Carroll and Briscoe, 2004; Briscoe and Carroll, 2006; Briscoe et al., 2006) and the C&C CCG parser (Clark and Curran, 2007). Preiss (2003) reported GR-based evaluation of PTB parsers including the Collins parser (Collins, 1997) and the Charniak parser (Charniak, 2000), although the SUSANNE-based gold data was used, and the results are not directly comparable to the results in this paper, where we use the data based on the PARC 700 selection of Penn Treebank sentences.

[3]http://www.informatics.sussex.ac.uk/research/groups/nlp/carroll/greval.html.

Figure 1 shows an example of GR annotation. GR represents labeled syntactic dependencies between words. For example, `ncsubj` means a non-clausal subject (e.g. `(ncsubj market They _)`), `dobj` indicates a direct object (e.g. `(dobj market cable-TV)`), and `ncmod` expresses a non-clausal modifier (e.g. `(ncmod _ opportunities grazing)`). Most relations are binary, while a few relation types have additional slots that represent subtypes of the relations. For example, `(xcomp to seeks discourage)` means that *discourage* is a to-infinitival complement of *seeks*. Refer to Briscoe (2006) for the definition of these relation types.

GR annotations are almost purely *syntactic* and therefore lack the means to evaluate the true potential of deep linguistic parsers that compute relationships based on semantics. However, it should be noted that GR represents non-local dependencies such as control/raising and movement. In this example, `(ncsubj discourage CNN _)` indicates a control relation, `(dobj discourage opportunities)` expresses a moved object of *discourage*, and `(cmod _ opportunities seeks)` means a relation between a relative clause and its antecedent. Since these relations are not explicitly represented by PTB parsers, this scheme may serve as a starting point in the identification of the added benefits of deep parsing and the discussion of problems in framework-independent evaluation. On the other hand, identifying most of the relationships in the GR scheme in the output of shallow phrase structure parsers requires matching of tree patterns, which makes it challenging to evaluate those parsers.

Relation types in the GR scheme are arranged in a hierarchy. Upper types represent more generalized and coarse-grained relations. This hierarchy is used for partial matching of relation types, which is intended for reducing disagreements involving relation types. For example, when a parser outputs `(ncmod _ market on)`, where the gold standard relation is `(iobj market on)`, this output is regarded as incorrect at the leaf level, but judged as correct at upper levels, `arg_mod` and `dependent`. This matching in the hierarchy is considered in scoring as described below.

Standard metrics for the GR scheme are *microaveraged* and *macroaveraged* scores. Microaveraged scores are similar to standard precision/recall/f-score, but take accuracy of non-leaf relation types into consideration. For example, `ncmod` is a subtype of `mod`, `arg_mod`, and `dependent`. A single `ncmod` dependency is regarded as expressing these four relations, and correctness of each of these relations is counted. Hence, as indicated above, disagreements of relation types are discounted, because higher level types are easier to identify. In general, microaveraged scores are higher than the accuracy of leaf relation types. A macroaveraged score is an average of the accuracy for each relation type, and frequencies of relation types are ignored. Hence, infrequent relation types affect macroaveraged scores. A program for computing microaveraged/macroaveraged scores is publicly available[4]. We also report the overall accuracy of leaf relation types only, which is the same metric used in our evaluation using SD.

[4] http://www.informatics.sussex.ac.uk/research/groups/nlp/carroll/greval.html

```
nsubj(market-2, They-1)
dobj(market-2, cable-3)
det(opportunities-8, the-5)
amod(opportunities-8, very-6)
nn(opportunities-8, grazing-7)
prep_on(market-2, opportunities-8)
nsubj(seeks-10, CNN-9)
rcmod(opportunities-8, seeks-10)
aux(discourage-12, to-11)
xcomp(seeks-10, discourage-12)
```

Figure 2: SD annotation for *They market cable-TV on the very grazing opportunities CNN seeks to discourage.*

2.2 Stanford Dependency (SD) scheme

The *Stanford Dependency* (SD) scheme was originally proposed for providing dependency relations that are more useful for applications than phrase structures (de Marneffe et al., 2006). This scheme was designed based on Carroll et al. (1998)'s grammatical relations and King et al. (2003)'s dependency bank, and modified to represent more fine-grained and semantically valuable relations such as apposition and temporal modification, while at the same time leaving out certain relations that are particularly problematic for the parsers it was intended to work with. Although no hand-annotated data is available, a program to convert PTB-style phrase structures into SD relations is available as part of the Stanford Parser[5]. That is, in principle, any PTB-style treebank can be converted into SD gold standard data. In practice, however, the conversion from phrase structure trees to SD is only approximate, and converting gold standard phrase structure trees results in only partially correct SD annotations. Unfortunately, the accuracy of these annotations is unknown. This scheme was recently used for the evaluation of shallow PTB-style parsers in the biomedical domain (Clegg and Shepherd, 2007; Pyysalo et al., 2007a), using GENIA (Kim et al., 2003) and BioInfer (Pyysalo et al., 2007b) as sources of gold standard PTB-style data.

Figure 2 shows an example of SD annotation for the same sentence as Figure 1. SD looks very similar to GR, and represents many equivalent relations such as `nsubj(market-2, They-1)`, `dobj(market-2, cable-3)`, and `nn(opportunities-8, grazing-7)`. The example also includes long-distance dependencies, but they are not as explicit as in GR, and often not as reliable. Here, `xcomp(seeks-10, discourage-12)` indicates a control relation between *seeks* and *discourage*, and `rcmod(opportunities-8, seeks-10)` expresses a relation between a relative clause and its antecedent. However, relations indicating that *CNN* is the subject of *discourage* and *opportunities* is the direct object of *discourage* are not represented. Refer to de Marneffe et al. (2006) for details of these

[5] http://nlp.stanford.edu/software/lex-parser.shtml

```
(ncsubj ordered Regulators _)
(ncsubj stop CenTrust _)
(ncsubj buying CenTrust _)
(ncmod _ ordered also)
(xcomp to ordered stop)
(xcomp _ stop buying)
(dobj buying stock)
(det stock the)
(passive preferred)
(ncsubj preferred stock obj)
(ncmod _ stock preferred)
(ncmod prt buying back)
(dobj ordered CenTrust)
```

```
nsubj(ordered-3, Regulators-1)
advmod(ordered-3, also-2)
dobj(ordered-3, CenTrust-4)
aux(stop-6, to-5)
xcomp(ordered-3, stop-6)
partmod(stop-6, buying-7)
prt(buying-7, back-8)
det(stock-11, the-9)
amod(stock-11, preferred-10)
dobj(buying-7, stock-11)
```

Figure 3: GR (left) and SD (right) for *Regulators also ordered CenTrust to stop buying back the preferred stock.*

relation types, and note that while some of these relations are described as part of the SD scheme, they are not implemented in the provided conversion software.

Although SD relation types are also organized in a hierarchy, it is intended for convenience in use by applications, and they are not aimed at parser evaluation purposes. Hence, we use standard precision, recall, and f-score as metrics for SD-based evaluation.

2.3 Comparison of the two schemes

As noted above, and illustrated in Figure 3, the GR and SD schemes are very similar in concept, and they represent equivalent dependencies in many cases. In this example, they share subject/object relations such as *(ordered, Regulators)*, clausal complements such as *(ordered, stop)*, and modifiers such as *(ordered, also)*. However, disagreements can also be found; for example, *preferred* is recognized as a past-participle modifier in GR (which is indicated as `(ncsubj preferred stock obj)`), while it is an adjectival modifier in SD (which is represented as `amod(stock-11, preferred-10)`). This comes from a difference in the part-of-speed (POS) of *preferred*. Representations of long-distance dependencies are also different, as previously mentioned. In this example, the subject of *stop* and *buying* is expressed in GR, while not in SD. Finally, we once again note that SD data converted from gold standard PTB data includes errors: in figure 3 *buying* is incorrectly recognized as a participial modifier to *stop*.

An advantage of GR is the availability of hand-annotated data, although the data size is relatively small. Another advantage is that partial matching of relation types may reduce the labor of format conversion. An advantage of SD is that we can use any data annotated with the more common PTB annotation policy. In addition, evaluation of PTB parsers (Collins, 1997; Charniak and Johnson, 2005) is convenient because software for format conversion is already available. However,

conversion errors make the evaluation only approximate, and the lack of a detailed definition of relation types is an obstacle to further development of conversion rules. This leads to a greater problem in framework-independent evaluation, since many of the same conversion errors are present in the SD data converted from gold standard PTB data and the converted output of shallow PTB-style parsers. For example, the SD data used in experiments includes many relations assigned "dep", which is the most underspecified relation type. This relation is chosen as output when the conversion program could not determine a relation type properly. In fact, more than 5% of relations are assigned "dep", meaning that the actual upper bound for PTB-to-SD conversion is below 95%. Because these errors are undocumented, in practice they result in inflated accuracy figures for shallow PTB parsers when compared to the accuracy of parsers that use other formats that must be converted with different software (and may contain a number of different conversion errors). In other words, the accuracy of parsers that use PTB-like output is overestimated, and the accuracy of parsers that use other output formats is likely to be underestimated.

3 Format Conversion

Our strategy for format conversion is based on post-processing. That is, we convert the output of parsers without changing the original parsers. During the development of conversion programs, we validate our progress using a development set of gold standard data in the format used by each parser, i.e., we run the conversion on parts of the HPSG treebank (Miyao et al., 2005) and the Penn Treebank. Automatic parsing results are used in the final evaluation. This is because accuracy obtained by converting gold standard data indicates the quality of conversion, and we can separate issues of format conversion from actual parsing errors. Accuracy figures from converted gold standard are also meaningful as upper-bounds of scores obtained with these evaluation schemes.

3.1 From Enju's XML format to GR/SD

We implemented conversion rules for the Enju XML format (Miyao, 2007). This format represents constituent structures and predicate argument structures in an XML format. To start with, we mapped predicate argument relations into GR/SD. Figure 4 shows an example of the XML format of Enju, and its mapping to GR. Arguments of *ordered* and *stop* can be mapped into GR in a fairly straightforward manner. Relation types are determined depending on argument labels (e.g. `ARG1` and `ARG2`), categories and POS tags of predicate words (e.g. `VBD`), and syntactic categories of argument constituents (e.g. `NP` and `VP`).

However, this simple method produced poor accuracy, mainly because of non-trivial disagreements between the formats. Hence, we had to implement heuristic conversion rules to fix these disagreements.

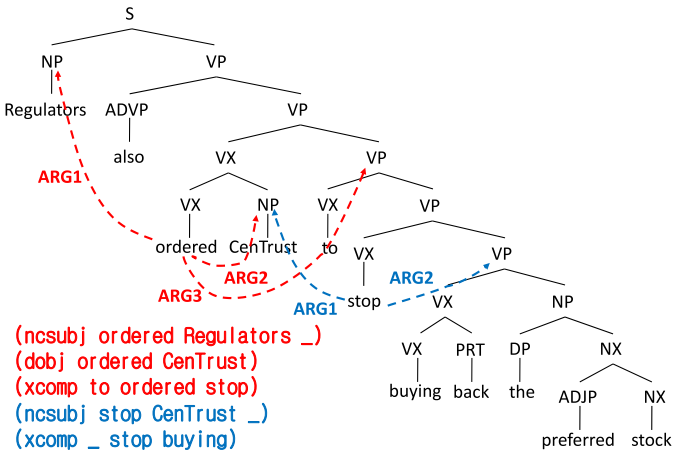


Figure 4: Enju XML format and mapping to GR

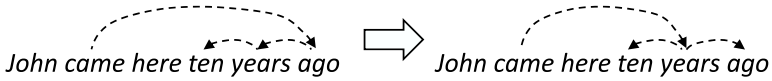


Figure 5: Conversion of lexical heads

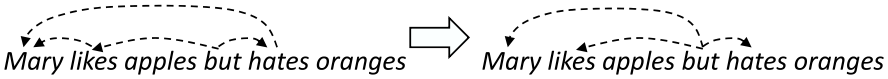


Figure 6: Conversion of coordination

It is often the case that certain types of relations are expressed in one format, but not in the other. For example, GR has "text adjunct" as a distinct relation, while Enju (and SD) does not distinguish such a relation type from others. Text adjunct is a text region delimited by some punctuation (Briscoe, 2006). Our conversion program outputs parentheses and appositive relative clauses as text adjuncts, but does not identify other text adjuncts. Another example is that GR does not represent internal structures of named entities, while Enju does. Hence, we detect text regions of named entities using simple patterns on POS tags, and remove dependencies inside named entity regions.

A major source of format disagreements was differences of lexical heads. Figure 5 shows an example of a temporal modifier. In Enju's output (left), the lexical head of *ten years ago* is *ago*, while in the GR scheme (right), it is *years*. Hence, our conversion rule changes lexical heads of such temporal modifiers. Similar conversion is applied to such constructions as number expressions.

Coordination was another major source of disagreements. Figure 6 shows an example of VP coordination. Enju outputs subject relations of conjunct VPs sep-

arately (left), while in GR the head of coordinated phrase is a coordinator and it has a subject relation (right). We therefore reduce two subject relations in Enju's output into one.

We also found systematic disagreements in specific constructions including relative clauses, quotations, copulas, and small clauses. For example, GR and SS represent a syntactic relation between the head verb of a relative clause and its antecedent. However, Enju does not output such relations explicitly, and instead, expresses a relation between a relativizer and its antecedent. We therefore developed conversion rules specialized to these constructions.

3.2 From SD to GR

Although the typed dependencies in the Stanford Dependency scheme are superficially similar to those in the Grammatical Relations scheme, conversion from SD to GR is problematic for many of the same reasons cited above in our discussion on conversion from Enju's XML format. However, a more serious problem with the use of SD (and SD to GR conversion) for parser evaluation is the lack of a gold-standard SD corpus. In our experiments, SD annotations are obtained from shallow PTB-style phrase-structure trees (which correspond to PTB trees with empty-node and function-tag information removed), using a conversion program included in the Stanford Parser. As can be expected, given our present discussion about parser format conversion, the PTB to SD conversion program is far from perfect. In addition to noticeable errors in the output of the conversion program, more than 5% of the dependencies are left completely underspecified, labeled only with the general `dep` type (which does not correspond to a specific grammatical relation, indicating only a head-dependent relation between two words). However, the accuracy of this conversion is not known, and cannot be easily computed without a gold-standard corpus for SD.

Conversion from SD to GR followed a similar pattern than the one described above for conversion from Enju XML to GR. First, a simple mapping between corresponding relations was attempted. As was the case with Enju, the resulting conversion was poor for all but the simplest relations (`det` and `aux`). A telling example is the conversion of SD's `nsubj` to GR's `ncsubj`. Although the two relations appear very similar, a number of undocumented differences make the conversion less straightforward than a simple mapping. For example, sentences involving the copula are treated differently by the two schemes, with GR attaching the subject as a dependent of the verb, and SD attaching the verb as a dependent of the predicate nominal. While some additional information required by GR's `ncsubj` (such as whether the subject position is inverted, or the initial relation of the subject) can be determined reliably by looking at aspects of the SD structure that go beyond the `nsubj` dependency, some information (such as subjects in control structures) simply cannot be determined from SD structures. Although de Marneffe et al.'s description of the SD scheme seems to indicate that enough information for such a conversion should be available, the actual implementation of the SD scheme in the

Stanford Parser lacks information relating to, for example, control structures and long-distance dependencies. This is understandable, given the difficulty in producing such information from shallow PTB-style trees. However, undocumented differences between the description and implementation of SD make the conversion even more difficult. A reasonably accurate conversion to GR's `ncsubj` was finally obtained with the use of development data, by inspecting specific examples annotated in each format. However, the efficacy of this approach varied in other types of relations, especially since the amount of development data available was limited.

One of the most problematic aspects of SD to GR conversion is the distinction of complements from adjunct, especially in prepositional phrases. SD does not assign a grammatical function to PPs, making it difficult to determine the correct relation in the GR scheme. As a result, the identification of indirect objects (`iobj`) has low accuracy compared to other relations. Of course, this also affects the accuracy of non-clausal modifiers. Another source of conversion errors is coordination, which is annotated in such a way in SD that its scope cannot be determined. As with Enju XML, recognizing the text adjunct (`ta`) relation is challenging, since it is not represented in the SD scheme. Although issues relating to headedness were less problematic in SD to GR conversion than in Enju XML to SD/GR, there were still differences, probably related to the use of an automatic conversion from the original PTB data to SD, compared to the manual annotation of the GR gold-standard data.

Finally, we reiterate that the conversion errors in our conversion from SD to GR are added on top of the PTB to SD conversion errors made by the Stanford parser implementation when a complete PTB to GR conversion is performed. For this reason, the GR results we obtain from parsers that produce PTB-style output do not do these parsers justice. While it is possible that a one-step conversion, from shallow PTB-style trees directly to GR, could produce more accurate results, an attempt by Preiss (2003) shows that this is not guaranteed to be much more successful, or at least is far from trivial. While our PTB to GR conversion does not provide completely fair grounds for comparison between shallow PTB-style parsers and Enju, a deep parser, it does serve to highlight some of the challenges in attempting such a comparison.

4 Experiments

Table 1 shows the sizes of the data sets used in experiments. For the development of conversion rules, we used 140 sentences extracted from the GR-annotated version of the PARC 700 Dependency Bank and the same set of sentences annotated automatically with SD (by running the Stanford Parser's automatic conversion on the corresponding Penn Treebank gold standard trees). For the final test, we used 560 sentences of the GR data and the same set of SD-annotated sentences. The GR data for the final test is the same set as previous works on GR-based evaluation

Table 1: Statistics of test data

scheme	# sent.	# rels.	# avg. rels/sent.	# rel. types
GR	560	10386	18.55	18
SD	560	9343	16.68	40

(Briscoe and Carroll, 2006; Briscoe et al., 2006; Clark and Curran, 2007).

The parsers we evaluate are Enju 2.2[6], Charniak and Johnson (2005)'s reranking parser (C&J parser), Charniak (2000)'s parser, and the Stanford parser (Klein and Manning, 2003). We also show previously reported microaveraged and macroaveraged scores for the GR evaluation of RASP (Briscoe and Carroll, 2006; Briscoe et al., 2006) and the C&C CCG parser (C&C parser) (Clark and Curran, 2007). Enju 2.2 includes a feature forest model (Miyao and Tsujii, 2005) and an extremely lexicalized model (Ninomiya et al., 2007), while excluding more advanced technologies such as deterministic parsing (Matsuzaki et al., 2007) and combination with shallow dependency parsing (Sagae et al., 2007).

Tables 2 and 3 show the accuracy of Enju and the PTB-style parsers obtained after the format conversion. In these tables, "auto" denotes figures obtained from automatic parsing results, while "gold" indicates accuracy figures obtained by converting gold standard data (establishing upper-bounds for the corresponding "auto" figures). In the case of Enju, "gold" figures are obtained by converting the HPSG treebank, and indicate the upper bound in accuracy in these evaluation schemes. Because the HPSG treebank lacks several sentences due to failures in the PTB-to-HPSG conversion (Miyao et al., 2005) that created the HPSG treebank, we excluded missing sentences from the evaluation of "gold". The evaluation of "auto" includes all sentences in the test data. For the evaluation of PTB parsers on GR, we applied our SD-to-GR conversion program to the output of the existing PTB-to-SD conversion software. In this case, "gold" indicates the accuracy obtained in the two-step conversion from PTB to GR. The "gold" accuracy for SD is 100%, because in SD evaluations we take the output of the Stanford Parser's PTB-to-SD conversion to be correct. Although we know the conversion is in fact not 100% correct, in our SD-based evaluation we do take the conversion of gold standard PTB trees to be our gold standard SD corpus, since a manually curated gold standard corpus is not available.

First, we note that these results show that the accuracy levels obtained by converting gold standard data are fairly low when format conversion is needed. This means that format conversion is far from perfect. For both GR and SD evaluation of Enju, "gold" accuracy figures are slightly higher than 80%, indicating that nearly 20% of dependencies cannot be converted properly. This is discouraging because reported accuracy levels of shallow and deep parsing are around 90%. However,

[6] Available at `http://www-tsujii.is.s.u-tokyo.ac.jp/enju/`

Table 2: Accuracy for GR

	gold			auto		
	precision	recall	f-score	precision	recall	f-score
Enju	84.27	83.67	83.97	80.60	78.74	79.66
C&J parser	78.60	68.51	73.21	75.86	62.92	68.79
Charniak parser	78.60	68.51	73.21	75.18	62.97	68.53
Stanford parser	78.60	68.51	73.21	70.88	60.24	65.13

this is an indication that previously reported accuracy figures might be inflated. The accuracy of "gold" PTB conversion to GR is even worse, since in this case we do suffer from the errors in PTB-to-SD conversion, and the errors in the subsequent SD-to-GR conversion. As we have described, these schemes are superficially similar, but this result reveals the difficulty of format conversion even between SD and GR. A possible reason is that a significant portion of GR dependencies could not be produced accurately from shallow phrase structures, which resulted in lower recall.

Results for "auto" reveal that Enju outperforms PTB parsers significantly in our GR evaluation, which is, as previously noted, unfair to the PTB parsers that suffer a double penalty in conversion. In our SD evaluation, which in turn heavily favors the PTB parsers and penalizes Enju, as previously discussed, the PTB parsers do have higher accuracy than Enju. It is then obvious that these contradictory results are heavily affected by the quality of format conversion, and this highlights how challenging (and even misleading) cross-framework evaluations can be. If we focus on an argument on the neutral nature of the GR scheme, we might be able to say that Enju is better in recognizing deeper dependency relations. However, this result relies on SD-to-GR conversion after PTB-to-SD conversion for PTB parsers, and it is likely that the figures for PTB parsers may be improved by directly converting their PTB-style to GR. However, it should be noted that our results for PTB parsers are better than the results reported by Preiss (2003) that implemented direct conversion from PTB to GR, although actual figures are not comparable because the test data is different, and the test set used by Preiss was in a different domain.

Tables 4 and 5 show microaveraged and macroaveraged scores for GR, respectively. We also show previously reported results for RASP (Briscoe and Carroll, 2006; Briscoe et al., 2006) and the C&C parser (Clark and Curran, 2007), which used the same evaluation scheme. Table 6 shows the accuracy of Enju for each relation type. As described in Section 2, microaveraged scores are higher than the accuracy in Table 2, which means that disagreements of relation types are reduced to some extent. However, nearly 13% of relations still cannot be produced. Similar results were also reported for the CCG parser, and this suggests that format disagreements may not be a simple matter of relation type mismatch.

Although the problem of format conversion remains, and we do not claim to

Table 3: Accuracy for SS

	gold			auto		
	precision	recall	f-score	precision	recall	f-score
Enju	83.43	81.44	82.42	77.38	74.54	75.93
C&J parser	100.00	100.00	100.00	88.36	88.45	88.40
Charniak parser	100.00	100.00	100.00	87.05	87.10	87.07
Stanford parser	100.00	100.00	100.00	85.36	83.16	84.25

Table 4: Microaveraged scores for GR

	gold			auto		
	precision	recall	f-score	precision	recall	f-score
Enju	87.49	86.79	87.14	83.57	81.73	82.64
C&J parser	80.84	69.16	74.54	79.08	67.46	72.81
Charniak parser	80.84	69.16	74.54	78.41	67.68	72.65
Stanford parser	80.84	69.16	74.54	74.76	64.83	69.44
RASP	—	—	—	77.66	74.98	76.29
C&C parser	86.86	82.75	84.76	82.44	81.28	81.86

Table 5: Macroaveraged scores for GR

	gold			auto		
	precision	recall	f-score	precision	recall	f-score
Enju	81.19	75.70	78.35	77.87	71.10	74.33
C&J parser	62.64	49.30	55.17	60.20	47.97	53.39
Charniak parser	62.64	49.30	55.17	59.39	48.08	53.14
Stanford parser	62.64	49.30	55.17	57.93	46.81	51.78
RASP	—	—	—	62.12	63.77	62.94
C&C parser	71.73	65.85	68.67	65.61	63.28	64.43

have achieved the best possible results with the PTB parsers, Tables 4 and 5 show that the accuracy of Enju is significantly higher than those of other parsers evaluated using the same test set, including PTB parsers, RASP, and the C&C parser. In particular, Enju achieved impressively higher macroaveraged scores, indicating that Enju is able to recognize infrequent relation types accurately.

Table 6: Relation type-wise accuracy

	gold			auto		
	precision	recall	f-score	precision	recall	f-score
ncmod	77.01	85.17	80.88	72.82	79.42	75.98
xmod	60.48	61.21	60.84	51.83	55.62	53.66
cmod	75.44	55.48	63.94	66.67	52.38	58.67
pmod	0.00	0.00	0.00	0.00	0.00	0.00
det	96.35	97.85	97.09	94.24	94.49	94.37
ncsubj	88.39	86.46	87.41	83.23	81.02	82.11
xsubj	100.00	57.14	72.73	75.00	42.86	54.55
csubj	75.00	100.00	85.71	100.00	100.00	100.00
dobj	91.80	93.53	92.66	88.35	89.36	88.85
obj2	56.52	68.42	61.90	61.90	65.00	63.41
iobj	82.24	60.08	69.43	82.25	58.33	68.26
xcomp	82.48	78.22	80.29	80.90	71.13	75.70
ccomp	87.39	79.39	83.20	81.92	73.45	77.45
pcomp	100.00	63.64	77.78	94.12	66.67	78.05
aux	95.88	95.36	95.62	94.66	92.77	93.70
conj	89.93	84.74	87.26	81.17	73.31	77.04
ta	61.54	25.91	36.47	59.46	22.68	32.84

5 Analysis of Format Disagreements

In what follows, we discuss sources of disagreements we found through our experiments. Figure 7 shows classification of dependency mismatches between the converted HPSG treebank and GR gold standard. That is, these come from format disagreements, and do not include parsing errors.

Text adjunct As described in Section 3, GR has a relation type called *text adjunct*, which is not explicitly identified by Enju. Although our conversion program tries to produce such relations, Table 7 shows that a significant number of text adjuncts were not recognized correctly.

Argument/modifier distinction It is widely recognized that a clear distinction between arguments and modifiers is difficult even for humans. In fact, there are no formal criteria for argument/modifier distinction in GR/SD annotation, and they are different even between GR and SD. Our conversion program approximately reproduces their intended distinctions, but a significant portion of them remain mismatched. One reason is that Enju outputs most prepositional phrases as modifiers. However, we found many other cases such as a distinction between adverbial clauses and clausal complements.

Table 7: Classification of dependency disagreements

Remaining disagreements	107
text adjunct	35
argument/modifier distinction	34
lexical head	25
POS	7
attachment	6
Conversion errors	36
named entity	15
number expression	6
coordination	6
others	9
Limitation of the HPSG treebank	14
noun phrase structure	10
others	4
Errors in the gold standard	13

Lexical head Although headedness is considered an agreed upon notion of syntactic structures, it is not obvious for a certain portion of syntactic constructions. For example, the head of "30.5 million" is "30.5" in GR and SD, while it is "million" in Enju. This example is rather simple and could be converted easily, but the important implication here is that there is a potential disagreement in headedness in many different types of constructions such as this one. This is critical because dependency-based evaluation heavily relies on the identification of lexical heads, and disagreements on heads may unfairly decrease apparent accuracy.

A similar problem is the necessary portion of arbitrary annotation policies that is inherent in this type of exercise. A typical example is the dependency structure of "*more than 2%*". In GR, "*more*" is the head of this phrase, and "*than 2%*" is a modifier of "*more*". However, in SD and Enju, "*more than 2*" constitutes a quantifier phrase, and "*more*" modifies "*%*". Either of these is acceptable, and we should regard this as a difference of annotation policies.

Part of Speech (POS) While the POS tags of some words are legitimately ambiguous, their differences significantly hurt accuracy because different relation types are assigned to words with different POS tags. For example, in Figure 3, GR recognizes "*preferred*" as a past participle, while SD (and Enju) treats it as an adjective, which results in assigning different relation types.

Conversion errors Our conversion rules for named entities, number expressions, coordination, and others did not work properly in some cases, and conversion errors

remained. We expect that these errors can be reduced with further improvement of our conversion rules, although this complicates the process of format conversion.

Limitations of the HPSG treebank The HPSG treebank does not represent some syntactic structures correctly. An example is internal structures of noun phrases. In the HPSG treebank, most noun phrases are annotated as right-branching trees, which are not necessarily correct. This is because the HPSG treebank was translated from the Penn Treebank, in which the internal structure of noun phrases is not annotated.

Errors in the gold standard We found a few cases that we simply disagreed or did not understand the intention of the GR gold standard annotation. Clearly, this type of problem is much more serious in our SD evaluation, because the SD evaluation sets are automatically converted from PTB, and they contain unjustifiable relations caused by imperfect conversion.

6 Summary and Future Directions

In this paper, we described an attempt to perform framework-independent parser evaluation. We focused on two dependency-based schemes for parser evaluation, namely, GR and SD, and evaluated the accuracy of an HPSG parser and shallow PTB-style parsers by converting their output into the dependency formats in these two schemes. In a series of experiments, we found that non-trivial conversion of parser output format was required. Experimental results showed that nearly 20% of dependency relations are problematic even when we converted a gold-standard HPSG treebank, demonstrating the difficulty of format conversion. In practice, it is difficult to have reliable conversion between different dependency representations, even between GR and SD, which are superficially similar. While we identified several of the major problems in our format conversion programs, their solution is unclear and would likely require a more complex conversion process. These remaining problems may obscure the results of parser evaluation. In fact, the results we obtain using the two evaluation schemes do not agree, confirming previous findings that framework-independent evaluation remains a challenge. Our experience suggests that GR evaluation is a step in the right direction, but a more accurate conversion procedure from PTB-style output to GR format is necessary.

From these observations, we conclude that a possible direction for improved parser evaluation includes machinery for dealing with multiple valid heads and dependency types in gold standard data. Following the discussion in Section 5, it is important to reduce the disagreements in relation types and lexical heads. While GR provides a partial solution to the former through its hierarchy of relations, a significant portion of remaining problems are relation type mismatches caused by the argument/modifier distinction, text adjuncts, and ambiguity of POS tags. For

the latter, a possible solution would be to annotate multiple candidate dependencies, any of which may be matched to parser output. It may also be desirable to determine the relative importance of relations in the evaluation. For example, in current schemes, the attachment of prepositional phrases is weighted identically to the attachment of determiners. While this is addressed in GR evaluation by having separate figures for precision and recall of each relation, complex results that include several dimensions can be difficult to interpret. Another example involves dependencies concerning idiomatic expressions, which may need to be excluded from the evaluation, since their structures vary significantly in different frameworks.

While this paper focused on parser evaluation at an intermediate representation, another direction is evaluation in end-to-end applications, such as information extraction and machine translation. In application-oriented, or task-based evaluation, some differences between parsers might be obscured because many other components contribute to overall system performance. However, this type of evaluation is indispensable for further understanding of how the characteristics of specific parsers make them more suitable in certain situations, and even to validate the results of more straightforward synthetic evaluations using gold-standard parsed data.

References

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B. and Strzalkowski, T. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proc. DARPA Speech and Natural Language Workshop*, pages 306–311.

Briscoe, T. 2006. An introduction to tag sequence grammars and the RASP system parser. Computer Laboratory Technical Report 662, University of Cambridge.

Briscoe, T. and Carroll, J. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proc. COLING/ACL 2006 Poster Session*.

Briscoe, T., Carroll, J. and Watson, R. 2006. The second release of the RASP system. In *Proc. COLING/ACL-06 Demo Session*.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J. and Way, A. 2004a. Evaluation of an automatic annotation algorithm against the PARC 700 Dependency Bank. In *Proc. 9th International Conference on LFG*, pages 101–121.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J. and Way, A. 2004b. Treebank-based acquisition of wide-coverage, probabilistic LFG resources: project overview, results and evaluation. In *Proc. "Beyond Shallow Analyses" workshop at IJCNLP-04*.

Cahill, A., McCarthy, M., van Genabith, J. and Way, A. 2002. Parsing text with a PCFG derived from Penn-II with an automatic F-structure annotation procedure. In *Proc. 7th International Conference on LFG*, pages 76–95.

Carroll, J. and Briscoe, T. 2004. High precision extraction of grammatical relations. In H. Bunt, J. Carroll and G. Satta (eds.), *New Developments in Parsing Technology*, Kluwer Academic.

Carroll, J., Briscoe, T. and Sanfilippo, A. 1998. Parser evaluation: a survey and a new proposal. In *Proc. LREC 1998*, pages 447–454.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL 2000*, pages 132–139.

Charniak, E. and Johnson, M. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. ACL 2005*.

Clark, S. and Curran, J. 2007. Formalism-independent parser evaluation with CCG and DepBank. In *Proc. ACL 2007*.

Clark, S. and Curran, J. R. 2004. Parsing the WSJ using CCG and log-linear models. In *Proc. 42th ACL*.

Clegg, A. B. and Shepherd, A. 2007. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics* 8(1), 24.

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. 35th ACL*.

de Marneffe, M.-C., MacCartney, B. and Manning, C. D. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. LREC 2006*.

Hockenmaier, J. 2003. Parsing with generative models of predicate-argument structure. In *Proc. 41st ACL*, pages 359–366.

Hockenmaier, J. and Steedman, M. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proc LREC-2002*, Las Palmas, Spain.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell, J. T. and Vasserman, A. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc. HLT/NAACL'04*.

Kim, J. D., Ohta, T., Tateisi, Y. and Tsujii, J. 2003. GENIA corpus — a semantically annotated corpus for bio-textmining. *Bioinformatics* 19, i180–182.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M. and Kaplan, R. M. 2003. The PARC 700 Dependency Bank. In *Proc. LINC'03*.

Klein, D. and Manning, C. D. 2003. Accurate unlexicalized parsing. In *Proc. ACL 2003*.

Malouf, R. and van Noord, G. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proc. IJCNLP-04 Workshop "Beyond Shallow Analyses"*.

Marcus, M., Santorini, B. and Marcinkiewicz, M. A. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.

Matsuzaki, T., Miyao, Y. and Tsujii, J. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *In Proc. IJCAI 2007*.

Miyao, Y. 2007. Enju 2.2 Output Specifications. Technical Report TR-NLP-UT-2007-1, Tsujii Laboratory, University of Tokyo.

Miyao, Y., Ninomiya, T. and Tsujii, J. 2005. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (eds.), *Natural Language Processing - IJCNLP 2004*, volume 3248 of *LNAI*, pages 684–693, Springer-Verlag.

Miyao, Y. and Tsujii, J. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. ACL 2005*, pages 83–90.

Ninomiya, T., Matsuzaki, T., Miyao, Y. and Tsujii, J. 2007. A log-linear model with an n-gram reference distribution for accurate HPSG parsing. In *Proc. IWPT 2007*.

Oepen, S., Flickinger, D. and Bond, F. 2004. Towards holistic grammar engineering and testing — grafting treebank maintenance into the grammar revision cycle. In *Proc. IJCNLP-04 Workshop "Beyond Shallow Analyses"*.

Preiss, J. 2003. Using grammatical relations to compare parsers. In *Proc. EACL 2003*, pages 291–298.

Pyysalo, S., Ginter, F., Haverinen, K., Laippala, V., Heimonen, J. and Salakoski, T. 2007a. On the unification of syntactic annotations under the Stanford dependency scheme: A case study on BioInfer and GENIA. In *Proc. BioNLP 2007*, pages 25–32.

Pyysalo, S., Ginter, F., Heimonen, J., Bjorne, J., Boberg, J., Jarvinen, J. and Salakoski, T. 2007b. BioInfer: a corpus for information extraction in the biomedical domain. *BMC Bioinformatics* 8(50).

Sagae, K., Miyao, Y. and Tsujii, J. 2007. HPSG parsing with shallow dependency constraints. In *Proc. ACL 2007*.

Toutanova, K., Markova, P. and Manning, C. 2004. The leaf projection path view of parse trees: exploring string kernels for HPSG parse selection. In *EMNLP 2004*.

The Grammix CD-ROM
A Software Collection for Developing Typed Feature Structure Grammars

Stefan Müller
Freie Universität Berlin

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

This paper discribes a bootable CD-ROM that contains grammar development software for teaching and research.

1 Purpose

The CD-ROM contains the TRALE system and various other software components that are useful for grammar development (see Section 3). It can be used for research projects as well as for teaching syntax or grammar engineering courses. It contains example grammars for German that correspond to chapters in a textbook (Müller, 2007). The texbook motivates the complex feature geometry that is used in current HPSG publications. The book starts with basic head argument structures and head features and extends this feature geometry to more complex structures including a semantic representation, and features for nonlocal dependencies and relative clauses. Students who work with this textbook can use the CD to look at structures corresponding to their respective knowledge in detail. They can modify and extend the toy grammars and study the consequences of their changes. While the set of reference grammars in the HPSG framework is growing larger and larger, large scale grammars are not usable for grammar teaching for various reasons. For instance such grammars often use auxiliary features that are needed for technical reasons and that are not documented in the literature. Feature descriptions of large scale grammars have an enourmous complexity which makes them unusable for teaching. The Grammix CD-ROM tries to fill the gap and provide a set of grammars with increasing complexity which allows to focus on certain features when they are introduced.

Apart from the textbook grammars the CD contains grammar fragments for German, Chinese, and Maltese. These grammars use a common core and can be seen as reference grammars for multi-lingual grammar engeneering with the TRALE system.

In addition, the CD contains the textbook by Frank Richter on *Grammar Formalisms and Parsing* and the software that comes with it.

The CD-ROM is intended to be a reference installation of TRALE and the attached software components. The system is Unicode-enabled and displays grammar files and grammar output that use Simplified Chinese, ISO-Latin-1, and Maltese characters correctly. As a bootable, stand-alone CD including an operating system, this reference CD provides not only the software mentioned but also all appropriate fonts, configurations, and other collateral files.

The CD-ROM also contains an installation of the LKB system. Readers who have other textbooks, grammars, and/or software that they want to distribute together with TRALE or LKB, should contact the author of this paper.

2 Description

The Grammix CD-ROM is a bootable Knoppix-based CD-ROM that contains two complete grammar development systems (the TRALE system (Meurers, Penn and Richter, 2002; Penn, 2004) and the LKB system (Copestake, 2002)) together with the grammar profiling tool [incr tsdb()] and example grammars written for the TRALE system, which correspond to the respective chapters in the text book *Einführung in die Head-Driven Phrase Structure Grammar* (Müller, 2007). In addition, the CD-ROM contains the Babel System (Müller, 1996, 1999, 2004) and TRALE grammars for Chinese, Maltese, and German which have a common core and use Minimal Recursion Semantics (Copestake, Flickinger, Pollard and Sag, 2005) for semantic representations.

Users may use the software contained on the CD to:

- load grammars, display analyzes as trees or as feature structure with arbitrary level of detail
- display type hierarchies and signatures (types together with the features that are introduced by the respective types)
- change and extend existing grammars (and of course write new ones . . .)
- to watch the Bottom-Up-Chart-Parser doing its work, visualize grammar rules, and explore newly developed grammars systematically.
- do systematic grammar testing and profiling with the profiling tool [incr tsdb()].

3 Grammar development software

The Grammix CD-ROM contains the following software (in addition to some of the software that usually comes with Knoppix):

- Main components:
  - TRALE
    Developers and Project Coordinators: Gerald Penn (University of Toronto), Detmar Meurers (The Ohio State University), Frank Richter (Universität Tübingen), with help by Nick Pendar (University of Toronto), Thilo Götz (Universität Tübingen), Stephan Kepser (Universität Tübingen), Dale Gerdemann (Universität Tübingen), Frederik Fouvry (Universität Tübingen), Vanessa Metcalf (The Ohio State University), Markus Dickinson (The Ohio State University), Holger Wunsch (Universität Tübingen), Martin Lazarov (Universität Tübingen), Oliver Suhre (Universität Tübingen), Mike Daniels (The Ohio State University), and Stefan Müller (Freie Universität Berlin)

  - Chart-Display
    Author: Patric Stiffel (DFKI Saarbrücken)
    TRALE- and Babel-Interface: Stefan Müller (Freie Universität Berlin)
  - [incr tsdb()] distributed with the LKB
    Author: Stephan Oepen (University of Oslo)
    TRALE-Interface: Stefan Müller (Freie Universität Berlin) and Frederik Fouvry (Universität des Saarlandes, Saarbrücken)
  - utool: The Swiss Army Knife of Underspecification
    Authors: Alexander Koller, Stefan Thater, and Michaela Regneri, with help by Marco Kuhlmann
    TRALE-Interface: Stefan Müller (Freie Universität Berlin)
  - Grammars for TRALE
    Author: Stefan Müller (Freie Universität Berlin)
  - Babel with grammar
    Author: Stefan Müller (Freie Universität Berlin)

- Other material:
  - MoMo
    Authors: Katja Ovchinnikova (Universität Osnabrück), Frank Richter (Universität Tübingen), Beata Trawiński (Universität Tübingen), Ashley Brown and Levente Barczy
  - Textbook Grammar Formalisms and Parsing
    Author: Frank Richter (Universität Tübingen)
  - Grammars for Grammar Formalisms and Parsing
    Author: Frank Richter (Universität Tübingen), Manfred Sailer (Universität Göttingen), and Beata Trawiński (Universität Tübingen)

4 System Background and system Requirements

The CD-ROM is based on Knoppix (currently version 5.0.1). Knoppix is a Linux distribution that is based on Debian. Knoppix uses a compressed file system which makes it possible to store more than 2 Gbyte of software on a CD-ROM. Recent versions of Knoppix use the Union file system which makes it possible to "change" configuration and application files although they are stored on the non-writable CD-ROM. The changed files are stored in the RAM. The operating system looks at the modified files in the RAM rather than at the earlier versions of these files on the CD-ROM. The modifications in the RAM will be lost when the machine is switched off or rebooted. However, users can store such information permanently on the hard disc or on an USB stick. This makes it possible to modify example grammars that are delivered with Grammix and store these grammars permanently.

The graphical desktop that is delivered with Grammix is KDE. The system requirements correspond to the system requirements imposed by Knoppix:

- Intel-compatible CPU (i486 or later, including Intel Macs with firmware 1.0.1 or later),
- at least 128 MB,
- bootable CD-ROM drive,
- standard SVGA-compatible graphics card,
- serial or PS/2 standard mouse or IMPS/2-compatible USB-mouse.

If you want to work with more complex grammars (starting from Chapter 10 of the textbook), you need 256 MB memory, 512 MB is recommended. If you work with [incr tsdb()] to debug TRALE grammars, another TRALE process is started and you need the corresponding amount more memory. Since the LKB and [incr TSDB()] are more tightly integrated, this does not apply to the profiling of LKB grammars.

Since Knoppix uses open source drivers for hardware components, Grammix may not run on very new hardware. For instance the graphics chip may not be recognized which may result in failure to display anything or in a suboptimal display with low resolution. In such cases using virtualization software may help (see Section 8).

5 Performance Issues

Grammix contains a standalone version of TRALE, which consists of saved states. Saved states do no contain the SICStus Prolog compiler. Instead the TRALE code is interpreted, which slows down the system considerably. Users who want to use TRALE professionally should consider buying a SICStus license and installing a SICStus version on top of Grammix.

Note also that Grammix is based on a 32bit operating system. SICStus Prolog runs much faster (factor two in certain situations) on modern CPUs with 64bit architecture.

6 Localization

Since the CD-ROM is a supplement of a textbook in German, the default language for menus and desktop information is German. If you prefer English, you may boot the system and enter the language code as an option at the boot prompt and press return:

```
knoppix lang=en
```

This will select the respective keyboard driver and provide menus and icon names in KDE according to the language you selected. Since the localization files are very

big not all languages are included in the CD-ROM. If your language is missing, please send an email to the author. Please refer to the Grammix web page to find out which languages are supported.

7 Download

You need a good internet connection for the download, since the size of the CD image is approximately 500 MB.

The CD image is available at:
`http://dg.fu-berlin.de/Software/Grammix/`.

8 Grammix and Other Operating Systems

The Grammix CD contains an operating system, so no other operating system is required to run the software. However, users may find it more convenient to run their normal operating system while using Grammix. Thanks to some auxiliary files provided by Doug Arnold it is possible to use VM Player[1] on Windows to run Grammix in a virtual machine. It is then possible to switch between your normal Windows environment and Grammix. Grammix can use the network facilities used by the host operating system without any further configuration. Depending on personal preferences, users may run Grammix from the CD-ROM or from a copy of the ISO image on their hard disk. The necessary files for setting up VM PLayer can be downloaded on the Grammix page, which was given in the previous section. VM Player is provided free of charge. Mac users with Intel Macs can use VM Ware Fusion[2].

References

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Lecture Notes, No. 110, Stanford: CSLI Publications.

Copestake, Ann, Flickinger, Daniel P., Pollard, Carl J. and Sag, Ivan A. 2005. Minimal Recursion Semantics: an Introduction. *Research on Language and Computation* 4(3), 281–332. `http://lingo.stanford.edu/sag/papers/copestake.pdf`, 11.10.2006.

Flickinger, Dan, Koller, Alexander and Thater, Stafan. 2005. A New Well-Formedness Criterion for Semantics Debugging. In Stefan Müller (ed.), *The Proceedings of the 12th International Conference on Head-Driven Phrase Structure Grammar, Department of Informatics, University of Lisbon*, pages 129–142,

[1] `http://www.vmware.com/products/player/`
[2] `http://www.vmware.com/products/fusion/`

Stanford: CSLI Publications. http://cslipublications.stanford.edu/HPSG/6/, 05.13.07.

Koller, Alexander and Thater, Stefan. 2005. Efficient Solving and Exploration of Scope Ambiguities. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 9–12, Ann Arbor: Association for Computational Linguistics. http://acl.ldc.upenn.edu/P/P05/P05-3003.pdf, 04.09.2007.

Meurers, Walt Detmar, Penn, Gerald and Richter, Frank. 2002. A Web-Based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing. In Dragomir Radev and Chris Brew (eds.), *Effective Tools and Methodologies for Teaching NLP and CL*, pages 18–25, proceedings of the Workshop held at 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia, PA. http://www.ling.ohio-state.edu/~dm/papers/acl02.html, 08.01.2004.

Müller, Stefan. 1996. The Babel-System—An HPSG Prolog Implementation. In *Proceedings of the Fourth International Conference on the Practical Application of Prolog*, pages 263–277, London. http://dg.fu-berlin.de/~stefan/Pub/babel.html, 24.11.2007.

Müller, Stefan. 1999. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Linguistische Arbeiten, No. 394, Tübingen: Max Niemeyer Verlag. http://dg.fu-berlin.de/~stefan/Pub/hpsg.html, 24.11.2007.

Müller, Stefan. 2004. Continuous or Discontinuous Constituents? A Comparison between Syntactic Analyses for Constituent Order and Their Processing Systems. *Research on Language and Computation, Special Issue on Linguistic Theory and Grammar Implementation* 2(2). http://dg.fu-berlin.de/~stefan/Pub/discont.html, 24.11.2007.

Müller, Stefan. 2007. *Head-Driven Phrase Structure Grammar: Eine Einführung*. Stauffenburg Einführungen, No. 17, Tübingen: Stauffenburg Verlag. http://dg.fu-berlin.de/~stefan/Pub/hpsg-lehrbuch.html, 24.11.2007.

Oepen, Stephan and Callmeier, Ulrich. 2000. Measure for Measure: Parser Cross-Fertilization. Towards Increased Component Comparability and Exchange. In *Proceedings of the 6th International Workshop on Parsing Technologies*, pages 183–194, Trento, Italy. http://www.delph-in.net/itsdb/publications/fertilization.ps.gz, 04.09.2007.

Oepen, Stephan and Carroll, John A. 2000a. Ambiguity Packing in Constraint-Based Parsing—Practical Results. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*

(NAACL'00), Seattle, WA, pages 162–169. http://www.delph-in.net/itsdb/publications/packing.ps.gz, 04.09.2007.

Oepen, Stephan and Carroll, John A. 2000b. Parser Engineering and Performance Profiling. *Natural Language Engineering* 6(1), 81–97. http://www.delph-in.net/itsdb/publications/parsing.ps.gz, 04.09.2007.

Oepen, Stephan and Flickinger, Daniel P. 1998. Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language* 12(4), 411–436, (Special Issue on Evaluation). http://www.delph-in.net/itsdb/publications/profiling.ps.gz, 04.09.2007.

Ovchinnikova, Ekaterina and Richter, Frank. 2007. Morph Moulder: Teaching Software for HPSG and Description Logics. *Logic Journal of the IGPL* . http://jigpal.oxfordjournals.org/cgi/content/abstract/jzm024v1, 04.09.2007.

Penn, Gerald. 2004. Balancing Clarity and Efficiency in Typed Feature Logic Through Delaying. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 239–246, Barcelona, Spain.

Penn, Gerald and Carpenter, Bob. 1999. ALE for Speech: a Translation Prototype. In *Proceedings of the 6th Conference on Speech Communication and Technology (EUROSPEECH)*, Budapest, Hungary.

Richter, Frank. 2006. A Web-based Course in Grammar Formalisms and Parsing, electronic Textbook. http://milca.sfs.uni-tuebingen.de/A4/Course/PDF/gramandpars.pdf, 20.09.2007.

Grammars and Programming Languages: To Further Narrow the Gap

Paula S. Newman
newmanp@acm.org

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

Symbolic parser/grammar combinations can be viewed as programming systems for natural language processing applications. From this perspective, they can be compared with conventional programming systems, and seen to require more effort in the important development activities of testing and debugging. This paper describes tools associated with the RH (Retro-Hybrid) parser that facilitate these activities and are, to varying extents, more broadly applicable. The paper also suggests a new approach to improving parser efficiency using constrained inputs, based in part on one of the RH debugging tools.

1 Introduction[1]

Developing general-purpose symbolic natural language parsers and grammars is difficult. One way of appreciating the difficulties is to view parser/grammar combinations as programming systems for natural language applications, and to compare them to traditional programming systems. From this perspective, it can be seen that far more effort must be devoted to testing grammars, reviewing test results, and debugging, than is required for traditional programs. Also, although considerable effort has been devoted to efficiency, parser execution tends to be slow, especially for deep-unification based grammars. Many innovative approaches have been applied to these problems. Copestake (2002) describes a comprehensive development system for HPSG grammars, and the XLE Online Documentation (2006) does the same for LFG grammars.

This paper explores some additional possibilities, based on tools developed for the relatively shallow RH parser (Newman, 2007a, 2007b), or suggested by those tools. The necessary background for the RH parser is provided in section 2. Then three problem areas are addressed. Section 3 discusses test/review problems, and broadly applicable methods of alleviating them using tools exploiting the TextTree display format (Newman, 2005). Section 4 discusses the difficulty of identifying failure points in debugging declarative grammars, shows how the difficulty is avoided in RH, and suggests some related approaches that are more widely relevant. Finally, section 5 suggests an approach to improving parser efficiency that adapts and combines: (a) a technique for leveling differences among parser outputs, for purposes of measurement (Ringger et al., 2004) with (b) methods of constrained execution, one of which is used in RH debugging

[1] Acknowledgments: Thanks are due to John Sowa, who made many helpful comments on an early draft of this paper, and also to an anonymous reviewer of another paper, who partially motivated this one by implying that developing deep-unification-based grammars is easy.

2 Background: The RH Parser

The RH (Retro-Hybrid) parser combines two major components, a shallow parser, and an overlay parser. These components are outlined below.

2.1 Shallow Parser

The shallow parser used in RH is the Xerox Incremental Parser (XIP), developed by Xerox Research Center Europe. XIP is actually a full parser, whose per-sentence output consists of a single tree of basic chunks, together with identification of (sometimes alternative) typed dependences among the chunk heads (Ait-Mokhtar et al. 2002, Gala 2004). But because the XIP dependency analysis for English was not mature at the time that work on the RH parser began, and because a classic parse tree annotated by syntactic functions can be more convenient for some applications, the overlay parser uses only the output chunks.

XIP is astonishingly fast, contributing very little to overall RH parse times (about 20%). It consists of the XIP engine, plus grammars for many languages. The grammar for a particular language consists of:

(a) a finite-state lexicon that produces alternative part-of-speech and morphological analyses for each token, together with bit-expressed subcategorization and control features, and (some) semantic class features,
(b) a substitutable tagger that identifies the most probable part of speech for each token, and
(c) sequentially applied rule sets that extend and modify lexical analyses, disambiguate tags, identify named entities and other multiwords, produce basic output chunks, and identify inter-chunk head dependences. (Note: the dependency rule results are not used in the RH hybrid.)

2.2 Overlay Parser

The overlay parser uses a guiding grammar expressed as a collection of networks that are similar to Augmented Transition Networks (Woods, 1970), thus the term "retro". A recursive control mechanism traverses the grammar networks top-down and depth-first to build constituents.

The grammar network arcs are labeled by references to tests for specialized categories. These tests, if successful, either return a shallow parser chunk or a parse forest, with the latter obtained by recursively invoking the control to traverse another grammar network. The specialized category tests are context-free, and, if performed, their results are cached. But the tests are often gated by pre-tests referring to contextual considerations such as parent category and left-sibling features. An extensive preference scoring system (see Newman, 2007b)

is used to prune partial parses early, and to select a single "best" parse, with scoring ties resolved by low attachment considerations.

3 Grammar Test and Review

3.1 The Problem

Conventional applications based on traditional programming languages are built using a more-or-less standard development process that assumes applications consist of relatively independent, modular units. As an application is specified, written, and unit-tested, a collection of global tests reflecting meaningfully different input combinations is constructed. Before the application is released, the tests are executed iteratively, and errors are removed, until the results are correct. In general, the number of input combinations that must be tested on a global level is relatively small, and determining whether results are correct is straightforward.

In contrast, for symbolic NL grammars, the test/review process is an integral part of grammar construction, and involves applying the grammars as a whole to as many input examples as feasible. This is because grammar elements are not modular in the same way as conventional programs, and the number of meaningfully different input combinations for a target genre is usually enormous.

Furthermore, determining the correctness of test results is far more difficult than for conventional applications. A standard result representation is a parse tree rendered in node + edge form. This representation, although useful for short sentences, is not easily scanned for longer sentences, which are very frequent. For example, the many non-fiction documents (in several genres) that were used to refine the RH English grammar have an average sentence length of roughly 20 words, with a standard deviation of about 11. Thus many parse trees obtained by parsing those documents are twenty or more words wide at the leaves, plus intervening blanks, so that the trees do not fit into a single window. Parse trees can also be very deep, making it difficult to grasp the structure.

3.2 TextTrees

TextTrees were developed for RH grammar test and review, and can also substantially reduce test/review effort for other types of syntactic grammars. TextTrees are linearly rendered, flattened, parse trees that convey right-side dependency by indentation. They can be read as prose, but at the same time expose most parsing errors.

An excerpt from a TextTree display obtained by parsing section J42 of the Brown Corpus is given in Figure 1. The indentations of the first TextTree show

4 (2) Thus the Congress marks a formal recognition of the political system that was central to world politics for a century. best more morett chunks

```
Thus
the Congress
marks
    a formal recognition
        of the political system
            {that
             was
                 central
                     to world politics
                 for a century}.
```

5 (8) International law had to fit the conditions of Europe , and nothing that could not fit this system , or the interests of the great European nations collectively , could possibly emerge as law in any meaningful sense. best more morett chunks

```
International law
had to fit
    the conditions
        of
            [Europe,
             and
             nothing
                 {that
                  |could not| fit}]
                 {[this system,
                   or
                   the interests
                       of the great European nations]
                  collectively,
                  could possibly emerge
                      as law
                          in any meaningful sense}.
```

Figure 1. Example TextTrees

a correct structure, but those of the second show a coordination of "Europe" and "nothing" as the object of the preposition "of". The probable correctness of the first sentence and the errors of the second sentence can be seen quickly, even though the sentences contain 20 and 35 words respectively.

The TextTree construction algorithm is given by Newman (2005). The algorithm is almost parser-independent, requiring only a <category, style-name> pair

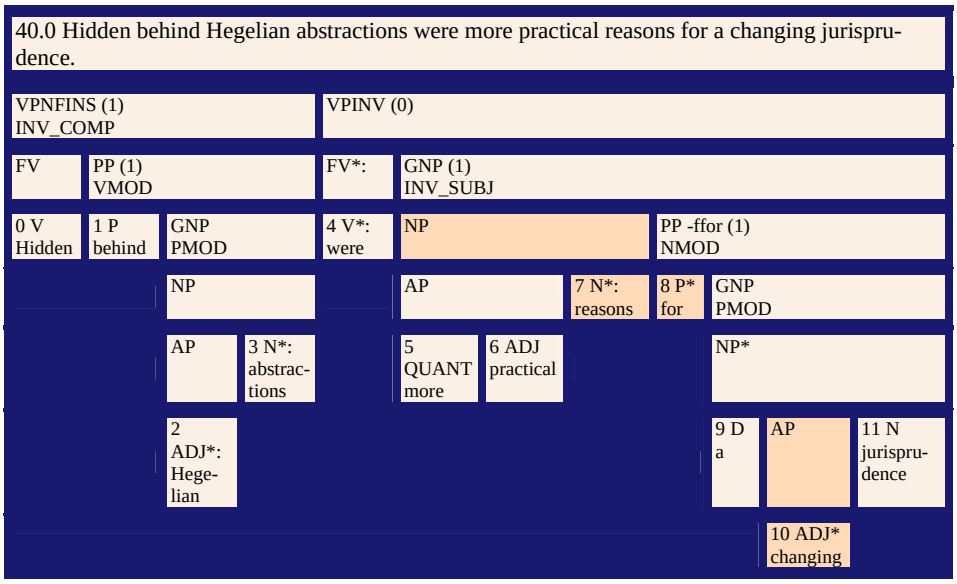


Figure 2. A full parse tree

for each constituent category type used by the parser. Limited empirical tests (see reference) show review speed improvements of 25% to 50% relative to using full parse trees, depending on the effort made to obtain a high review speed. Also, TextTree display files provide a good grasp of overall document problems.

3.3 Other Uses of TextTrees

TextTree displays, like those of Figure 1, have been the primary means of reviewing RH parser results, allowing rapid determination of whether parses identified as "best" are probably correct, and providing access to other results. Each sentence in the display has the following links:

a) The **best** link leads to a display of the full parse tree[2] for the best parse. Figure 2 gives an example. We note that the Figure 2 tree is for a sentence of only 12 words, yet spans the page in this format, even with some tokens split between lines. Although full-screen presentations accommodate wider trees, trees for long sentences must be split into segments, arranged vertically.

[2] The full parse tree displays are in a TreeTable format (Newman, 2002). Parent nodes physically span children, giving a good appreciation of structure. For applications with narrower trees, the cells can contain indicative content. In the parse tree adaptation, node features are accessed by mouse-over of category names.

b) The **chunk** link leads to a display of shallow parser output in the same format as for full parse trees, with basic chunks shown as children of the (omitted) top node.
c) The **more** and **morett** links lead to displays of other parses retained, in full parse tree form and TextTree form, respectively. The number of parses retained depends on parser invocation options, outlined below.

0. International law had to fit the conditions of Europe, and nothing that could not fit this system, or the interests of the great European nations could possibly emerge. 0.40 (s = 4) 0.35 (s = 4)

```
 International law
 had to fit
     the conditions
     of
         [Europe,
          and
          nothing
              {that
               |could not| fit}]
              {[this system,
                or
                the interests
                    of the great European nations]
               could possibly emerge}.
```

0. International law had to fit the conditions of Europe, and nothing that could not fit this system , or the interests of the great European nations could possibly emerge . 0.4 (s = 3)

```
 [{International law
  had to fit
      the conditions
          of Europe},
 and
 {nothing
      {that
       |could not| fit
           [this system,
            or
            the interests]
               of the great European nations}
  could possibly emerge.}]
```

Figure 3. Excerpt from **morett** display

The number of trees retained by the parser depends on the parser invocation mode. In "production" mode, only the best parse is retained, so the **more** and **morett** links are not of interest. In "standard" mode, all parses with the highest preference score are retained, while in "noprune" mode preference-based pruning is disabled, and many more parses may be completed and retained.

The **morett** link supplies additional TextTree-based information. Figure 3 excerpts a **morett** display of "noprune" mode results for a reduced version of the incorrectly parsed sentence of Figure 1. Duplicate structures are grouped. Figure 3 first shows a strange, incorrect structure, reflected in two parses, both given the preference score ***s = 4***, and then a better structure, with score ***s = 3***, indicating that one source of the problem is in the preference assignments. (The correct structure appears slightly further on in the display.) If the display were produced in standard mode, only the highest scoring parses would be shown.

The **morett** display thus serves several purposes. First, it highlights probably duplicate parses. Also, when pruning is enabled, it shows parses that were retained when they should not have been. Finally, by grouping duplicate structures (thus limiting the number of structures to be scanned), it facilitates determining whether the correct parse was found, even if not identified as best [3]

A further RH use of TextTrees (not illustrated) allows a more direct determination of whether a correct structure was obtained. A "find" parser option accepts single sentences in TextTree format and produces as output a list of matches found in the retained parses, with links to the full parse trees.

All these TextTree-based RH tools should be useful in other parser development environments. Another potential use beyond those discussed is in regression testing, to illustrate changes in results. But a caveat is needed. TextTrees are clearly useful for SVO languages with mostly projective grammars (i.e., with mostly contiguous constituents). Their adaptability to other types of languages has not been explored.

4 Grammar Debugging

4.1 The problem

Debugging erroneous grammar outputs addresses several questions: why a correct parse was not found, why some incorrect parses were found, and why duplicate parses were found.

[3] The number of trees shown via the **more** and **morett** links is somewhat constrained, to limit the size of files containing full parse trees. Currently, if only one sentence is being parsed, the highest scoring 100 trees are shown; otherwise a random sample of 30 is used.

In comparison with conventional applications, grammar debugging involves more effort, partly because it is far more frequent, being an integral part of the continuous test/review process. But the major reason for the increased difficulty is that traditional debugging approaches, which rely to a large extent on tracing program execution, are usually not applicable. The tracing can involve obtaining application specific intermediate results at critical points in the program, and/or using language-processor-supplied debugging tools to step through code related to errors. Analogous tracing of grammar "execution" would follow the application of the grammar rules. However, useful traces require the existence of a simple, accurate mental model of the execution sequence and, unfortunately, parsers rarely conform to such a model. As an important example, chart parsing makes the sequence of parser operations difficult to predict and follow, because adding an edge to the chart can activate the continuation of many rules to which that edge might be relevant. Also, general traces of rule application are likely to be extremely voluminous, because of the number of alternatives examined.

Therefore, debugging tools for grammars tend to focus on accumulated results. For bottom up parsing, displays can be constructed to illustrate the successive node attachments that were made, linked to the associated grammar rules, to aid in detecting both inappropriate and missing attachments. Such displays, however, can be quite complex, even for relatively short sentences. Copestake (2002) discusses how the complexity can be reduced, by highlighting nodes which are ancestors or descendants of an interactively selected node.

For top-down parsing, creating equally informative post-parse displays is more challenging. A bottom-up display of attachments can only contain constituents that were eventually expanded down to tokens, and so are of less help in discovering missing rules or rule components.

In the next subsection we discuss why and how traces are used successfully in RH grammar development. A subsequent subsection discusses an extension which is more generally applicable.

4.2 RH debugging with traces

While, for many parsers, tracing is not practical for grammar debugging, it is useful for debugging an RH overlay parser grammar, for three reasons:

1. Most important, overlay parser execution is a simple, top-down/depth-first process, and therefore can be traced by a hierarchically indented sequence recording significant parser actions.

2. Trace volume is limited, because the overlay parser operates on disambiguated tags and chunks.[4]
3. The ATN-like grammar networks provide symbolic information that can be replicated in the traces. An excerpt of a grammar network is shown in Figure 4, with each row representing one or more network arcs in terms of a single **From** state, possibly multiple **To** states, and a **Label** of a specialized category test. (For a fuller description of the networks, see (Newman, 2007a)). The relevant aspect here is that searching and following the traces is aided by the inclusion of the network names, row information, as well as general category names within the trace output. Figure 5 shows an excerpt from an overlay parser trace that is attempting to develop a coordinated NP (category NPC) using the network of Figure 4, starting at token position 0. The trace shows a test for pre-coordinators, and then goes on to a test for a simple (non-coordinated) noun phrase. (The excerpt is edited to remove detail and expand some abbreviations).

From	To	Label	Other material & comments
NC_1	NC_2	T_PRE_COORD	// e.g., both, either, between
NC_1	-1	T_CUT	// meta-test, like prolog cut // stops testing if prior test satisfied
NC_1	NC_3	T_NPS	// simple noun phrase
… Omitted**…**			
// After first simple noun phrase			
NC_3	NC_5	T_COMMA	
…Omitted**…**			

Figure 4. Excerpt from NPC_NET for parsing coordinated noun phrase

- **Parsing NPC:0 in NPC_NET entry NC_1**
- Continue NPC(0:0) NPC_NET at NC_1, test T_PRE_COORD
  - do_T_PRE_COORD:0
  - do_T_PRE_COORD:0 : failed
- Continue NPC(0:0) NPC_NET at NC_1, test T_CUT
- Continue NPC(0:0) NPC_NET at NC_1, T_NPS
  - do_T_NPS:0
  - **Parsing GNP:0 in NPS_NET entry N_1**
  - Continue GNP(0:0) NPS_NET at N_1, test T_TITLE

Figure 5. Excerpt from trace for a coordinated noun phrase at token position 0

[4] Erroneous RH parses that are the result of errors in tagging and chunking are found via the **chunks** display, and can be investigated using the very helpful XIP trace facilities.

4.3 Debugging with Constraining Inputs

A more generally applicable approach to grammar debugging uses constraining inputs, which are sentences with some substrings bracketed and category-labeled, for example "{NP: *Time*} *flies* {PP: *like an arrow*}.}".

Constraining inputs can be applied in different ways, depending on parser direction and type. For bottom-up parsers, the constraining inputs can be used to identify points of failure by limiting a bottom-up chart to constituents that are bottom-up consistent with the brackets and labels, For top-down chart parsers, the constraining inputs might dictate a parse of each bracketed element, starting with the lowest. A failure would occur if there is no parse for a bracketed element, or if no parse for a bracketed element includes all its contained bracketed elements in the input.[5]

For the RH overlay parser, constraining inputs are used to restrict parser execution. This reduces trace volume and limits the effects of an inconvenience introduced by caching, namely that all but the first invocation of a test at a token position returns a cached result, and the trace indicates only either failure or a success. In that case, the trace must be searched from the beginning to find the trace of that first invocation.

However, only simple bracketings have been used in RH debugging, because effective top-down bracketing requires that the brackets serve as barriers. Any bracket beginning at a token position ***p*** must be preceded by any higher level brackets also beginning at ***p***. Figure 6 shows effective and ineffective bracketings to constrain processing of a sentence to a declarative, rather than imperative, interpretation with an initial NP.

The constraining brackets need not be precise with respect to punctuation enclosure. Methods for allowing this are described further on, in the context of using constraining inputs to improve parser performance.

Summarizing the debugging tools discussed above, literal traces of parser activity are used directly for RH debugging, and their convenience is improved by the use of constraining inputs. While tracing parser activities may not be suitable to other parser environments, constraining inputs may well be helpful in other ways to aid in debugging.

---

[5]A somewhat related facility is provided by XLE (XLE Online Documentation, 2006). It allows the complete trees retained after the parse to be searched for ones consistent with a bracketing. The facility might thus be used to find a point of failure by applying the facility multiple times, each time adding higher brackets

ANY: {NP: You} love Mary}}	will not rule out imperative reading
{ROOT: {NP: You} love Mary}}	will rule it out
{NP: You} love Mary	parse fails because *NP* not outermost at 0

Figure 6. Constraining inputs

5 Parser Efficiency

5.1 The Problem

Deep-unification-based parsers do not currently approach the efficiency of the fastest parsers. Figure 7 gives approximate relative parse times of several parsers for the Penn TreeBank Wall Street Journal Section 23. The relative times are derived from reports comparing the efficiency of Collins Model 3 (Collins, 1999) with one of the other parsers, when executed on the same machine.[6]

The relative times for the XLE LFG parser, considered a very fast unification-based parser, are based on a report by Kaplan et al. (2004). Results are given for both core (reduced) and full English grammars. The relative time for the fast stochastic parser by Sagae and Lavie (2005) is derived from that reference, and that for the RH parser is based on results reported by Newman (2007a).

Deep-unification-based parsers tend to be slower because unification is a destructive operation that requires copying of the structures to be unified, using large amounts of time and space. Sophisticated methods have been developed to limit this cost (Maxwell and Kaplan, 1996), but do not remove the gap.

5.2 Current Approaches

An important current approach to the efficiency problem uses the results of fast partial parsers to constrain, or establish preferences for, follow-on parser search paths (Frank et al, 2003 and Daum et al., 2003).

	Time
Sagae & Lavie 2005	.25
RH 2007	.31
Collins model 3	1
XLE/LFG core English grammar 2004	1.5
XLE/LFG complete English grammar 2004	5

Figure 7. Relative time comparison

[6] We do not include relative times for the probably faster stochastic CCG parser by Clark and Curran (2007), because their comparisons are explicitly stated to be indicative only, in that the CCG results were obtained on a faster machine than the other parser results.

{N'' {SPEC the} {N'{N boy} {P'' … }}}

{NP {NP {DET the} {N boy}} {PP … }}

Figure 8. Two parses for "the boy in the park"

It should be possible to pursue this direction further, that is, to constrain deep, relatively slow, parsers by the results of full, fast, parsers. The difficulty with doing so is that the results of an arbitrary front-end parser are unlikely to be consistent with those of a back-end parser in terms of constituent structure, labeling, and punctuation enclosure. For example, different parsers might deliver the different structures shown in Figure 8 for the same noun phrase.

Cahill et al (2007) describe one way of addressing this difficulty. They train a fast statistical parser on the uncorrected outputs of the target back-end parser for a large corpus. The resulting trained parser then serves as the constraining front end. This removes the inconsistency problem, at the price of not training on a gold standard. It can thus improve performance and coverage (the latter because the constraints can prevent the parser from exceeding imposed resource limits), but the potential improvements in accuracy that might be obtained by training on fully-correct parses are not realized. The method is reported to obtain speedups of approximately 1/3.

5.3 Alternative Possibility: Leveling Differences

Another way of using a faster parser to constrain the execution of a slower, deeper one is to adapt an approach developed by Ringger et al (2004). That approach is intended to remove differences between parse trees for purposes of measurement, for example, to compare the correctness of parser results against a treebank. It focuses on removing brackets from non-maximal projections of heads, and essentially consists of the following steps:

1. General transformation rules developed for the pair of gold standard trees and the output trees of the parser to be measured are applied. Most transformation rules just modify the identification of phrasal heads.
2. Then, brackets representing non-maximal projections of heads are removed from both sets of trees. After this operation the result for the first parse of Figure 8 would be simply: {N" the boy {P" …}}
3. Finally, after other (not specified) pair-specialized transformations are performed, the resulting trees are compared using unlabeled bracketing.

To adapt the approach for purposes of constraining parser execution, transformation rules would necessarily be restricted to the outputs of the fast front-end parser. Then brackets and labels would be retained only for maximal projections of heads.

The resulting transformed results of the front-end parser could be used in different ways to constrain a back-end parser, depending on the back-end parser approach. Two examples are given.

Top-down usage. For top-down parsing, the approach sketched in the preceding section on debugging the RH overlay parser using constraining inputs is applicable. Bracketing by maximal projections of heads (with heads identified by the front-end parser adjusted if necessary) should satisfy the requirement that the brackets serve as barriers, that is, that any bracket beginning at a token position ***p*** must be preceded by any higher level brackets also beginning at ***p***. To deal with differences in category labeling and punctuation enclosure, the following method is used:

1. Opening brackets in the input are considered to extend over an interval <***b'***, ***b***>, where ***b*** is the actual position of the bracket, and ***b'*** is the first of possibly several punctuation tokens immediately preceding ***b***.
2. Each back-end parser category is considered equivalent to ANY of the constraining categories which it might match.
3. In parsing a constituent, when the parser reaches token position ***p*** within the next open bracket interval <***b'***, ***b***>, and that bracket has category ***c***, the parser will not process a test for a category ***sc*** unless ***sc*** = ***c***, or ***sc*** is considered equivalent to ***c***, or ***sc*** is a punctuation test.
4. When processing a constituent corresponding to a bracket pair ending at a position ***eb***, no tests are processed for the constituent beyond ***eb*** except for immediately following sequences of punctuation.

Bottom-up usage. For bottom-up head-driven parsing, the constraining structure might be used in roughly the following way:

1. Restricting token tags to those given by the constraining parse (suitably mapped)
2. Initiating a constituent ***c*** using a lexical head ***h*** only if there is a constraining subtree containing ***h*** that is headed by ***h***.
3. When a constituent ***c*** with lexical head ***h*** is to be extended by some dependent ***d***, the extension is accepted only if there is a lowest constraining subtree ***LCTc*** larger than ***c*** that also has lexical head ***h***, and ***d*** is either a token or equal to another constraining subtree, and ***LCTc*** includes ***d***.

In either usage case, if no parse is obtained by the combination, the deeper parser might be used directly.

These approaches would also address the ambiguity problem, limiting the burden on current methods of supplementing unification-based grammars with stochastic, corpus-based post-processors for disambiguation (Kaplan et al. 2004, and Toutanova et al. 2005). (The suggested approaches do not replace post-processing, because a particular syntactic structure might have multiple associated deep structures requiring disambiguation.)

5.4 Experiment

A small experiment was performed to test the potential performance improvement for the RH overlay parser using the method combination for top-down parsers described in the previous subsection. Specifically, we combined a simulated result of the described adaptation of the Ringer et al. (2006) approach, with the RH approach to debugging using constraining inputs.

For a sentence that required an inordinate amount of parse time, the overlay parser was constrained by a manually constructed bracketed input, shown in Figure 9, with the brackets used only for maximal projections of heads.[7] The results were not encouraging, because even for a sentence that the parser found difficult, the performance improvement was only about 35%.

However, the approach should realize more significant gains if used with a slower parser, because: (a) the RH overlay parser builds on the results of the fast front-end shallow parser, so no time is spent in considering alternative token tags or basic attachments, and (b) the work performed by the overlay parser for each potential constituent is far less than that performed, for example, by unification-based parsers.

6 Summary

We have described some development tools associated with the RH parser, both mainstays, and some more recent extensions. These tools bring the grammar development process closer to that of applications built using traditional programming languages, in the sense that they reduce the amount of effort required for result review and debugging. Many of the tools are relevant across grammar frameworks. We have also described an approach to improving efficiency partially suggested by one of the RH development tools. Figure 10 summarizes the existing and potential tools discussed, their usage in RH development, and their wider relevance

[7] We note that the illustration is formatted just for readability. It is not a TextTree because it contains many brackets and labels, and the first PP is not indented.

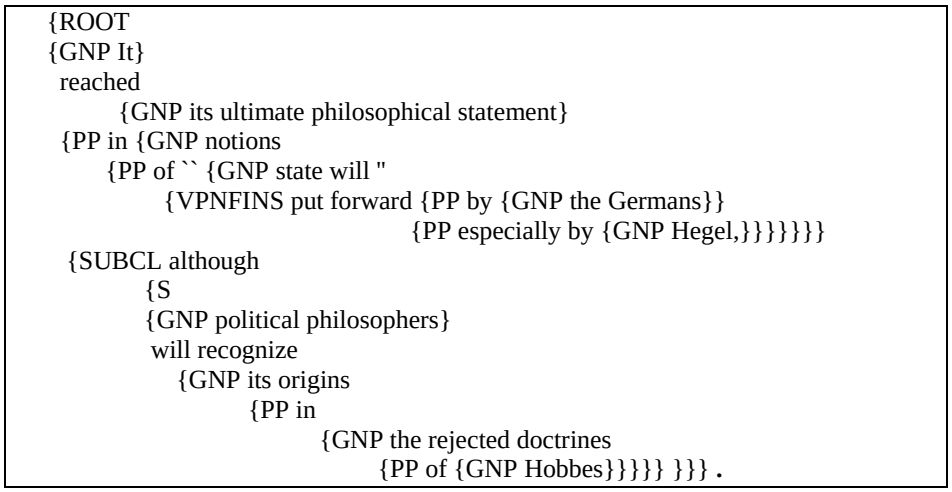


Figure 9. A constraining input using maximal projections of heads

The most important of the tools described, TextTrees, allow most erroneous parser results to be rapidly identified. Also, displaying all retained parses in TextTree form assists in finding parses that were obtained, but were not identified as "best", and in identifying duplicates for further study and removal. TextTree displays are relevant to reviewing parser results for SVO languages with mostly projective grammars. Their relevance to other types of grammars is yet to be studied.

Parser execution traces have served as the primary tool for debugging the RH overlay parser grammar. A recent extension constrains parser execution by partially labeled and bracketed inputs, to make following traces more convenient. While execution traces are of questionable relevance to chart parsers, using constraining inputs in debugging are of wider relevance.

The paper also suggests a technique for using the results of a fast parser to constrain the activities of a slower one. The technique combines: (a) an adaptation of a method for leveling differences between parser results with (b) different methods of constraining the activities of a back-end slower parser, depending on parser approach. While the technique is not very promising for the RH hybrid, because the overlay parser builds on the results of a very fast shallow parser, it may provide significant performance gains for deep-unification-based parsers

Tool Area	Usage in RH development	Relevance beyond RH?
Test/Review		
Best texttrees	heavy	yes
All texttrees	some (relatively new)	yes
Find texttree in outputs	new	yes
Debug		
Via tracing	heavy	limited
Using constraining input	limited (relatively new)	yes
Efficiency		
Using constraining input	no	probably

Figure 10. Summary of tools, usage in RH, and applicability

References

Cahill, Aoife, King, Tracy Holloway, Maxwell, John T. 2007. Pruning the Search Space of a Hand-Crafted Parsing System with a Probabilistic Parser. *Proceedings of the Workshop on Deep Linguistic Processing*, pages 65-72

Clark, Stephen and Curran, James R 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. To appear in *Computational Linguistics* 33(4). Draft at www.cs.usyd.edu.au/~james/pubs/pdf/cl07parser.pdf

Collins, Michael. 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, University of Pennsylvania.

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

Daum, Michael, Foth, Kilian A., and Menzel, Wolfgang. 2003. Constraint Based Integration of Deep and Shallow Parsing Techniques. In *Proc. 11th Conf. of the European ACL*, pages 99-106

Frank, Anette, Becker, Markus, Crysmann, Berthold, Kiefer, Bernd, and Schaefer, Ulrich. 2003. Integrated Shallow and Deep Parsing: TopP Meets HPSG. In *Proc 41st Annual Meeting of the Association for Computational Linguistics*

Kaplan, Ronald M., Riezler, Stephan, King, Tracy H., Maxwell, John T., Vasserman, Alex, and Crouch, Richard. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 97-104

Maxwell, John T., Kaplan, Ronald M. 1996. Unification-based parsers that automatically take advantage of context freeness. *Proc First Annual LFG Conference*

Newman, Paula S. 2002. Exploring discussion lists: steps and directions. In *Proc Second Joint ACM/IEEE-CS Conference on Digital Libraries*, pages 126-134

Newman, Paula S. 2005. TextTree Construction for Parser and Grammar Development. In *Proc Workshop on Software* at *43rd Annual Meeting of the Association for Computational Linguistics*.
Available at http://www.cs.columbia.edu/nlp/acl05soft/

Newman, Paula S. 2007a. RH: A Retro-Hybrid Parser. In Proc Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, pages 121-124

Newman, Paula S. 2007b. Symbolic Preference Using Simple Scoring. In *Proc 10th International Workshop on Parsing Technology,* pages 83-92

Ringger, Eric K., Moore, Robert C., Vanderwende, Lucy, Suzuki, Hisami, and Charniak, Eugene. Using the Penn Treebank to Evaluate Non-Treebank Parsers, In *Proc 2004 Language Resources and Evaluation Conference,* pages 867-870

Sagae, Kenji, and Lavie, Alon. 2005. A classifier-based parser with linear runtime complexity. In *Proc. 9th Int'l Workshop on Parsing Technologies*.

Toutanova, Kristina, Manning, Christopher D., Flickinger, Dan, and Oepen, Stephan, 2005. Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. *Research on Language and Computation* 3(1), pages 83-106

XLE Online Documentation. 2006.
Available at http://www2.parc.com/isl/groups/nltt/xle/doc/xle.html#SEC

Soft Constraints at Interfaces

Nick Pendar

Iowa State University

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

Assuming that grammar is best modeled as a set of modules each representing a constraint system, like other human cognitive faculties, these constraint systems are bound to disagree and soft constraints are needed for conflict resolution among modules. This paper outlines an approach to incorporate soft constraints among grammar modules. The paper first goes over the issue of modularity and conflicting requirements. It then summarizes a generalized theory of soft constraint satisfaction (SCSP). It then outlines a linguistic constraint system based on SCSP theory. Finally, it shows how the type antecedent constraints of HPSG can be extended in an SCSP-based framework to allow for constraint violations and gradient constraints.

1 Introduction

Inquiry in theoretical linguistics, cognitive science, and AI has led many researchers to believe that constraint-based approaches in modeling human behavior capture our understanding of the phenomena in question better than procedural approaches. The advantage of these approaches is that expressing what we know about the data in the form of constraints can capture generalizations more accurately and intuitively. A procedural formalization has the disadvantage of mixing *knowledge* with the *processing* of that knowledge. By keeping the two separate, we as researchers give ourselves the opportunity to improve each separately. Constraint-based linguistic theories have been making headway in better understanding of language. Two noteworthy examples are Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag, 1987, 1994), and Lexical Functional Grammar (LFG, Bresnan, 1982, 2001). Another remarkable example is Optimality Theory (OT, Prince and Smolensky, 1993); the underlying assumption in this theory is that constraints are not absolute (*crisp*), but instead they are violable (*soft*). Constraints are thus ranked according to their importance, and the form that violates the fewest high-ranking constraints is considered the optimal form. Constraint-based systems are also widely used to solve real-life problems in computer science. Network management, scheduling, and transportation problems are most easily solved in a constraint-based approach.

In cognitive science and AI, problems are envisioned as constituting discrete objects with constraints imposed on their interactions. A keyword in the preceding statement is *objects* as objects are more or less independent entities with certain properties and they perform a set of predefined functions. Several theories in cognitive science have found modular approaches beneficial. For example, Newell's (1990) unified theory of cognition paints a modular picture of the mind in which each cognitive faculty takes the form of a discrete entity that communicates with

[†]I thank Elizabeth Cowper, Elan Dresher, Frank Keller, Jean-Pierre Koenig and Gerald Penn for their insightful comments on my work. This work was partially funded by the Social Sciences and Humanities Research Council of Canada and Ontario Graduate Scholarships.

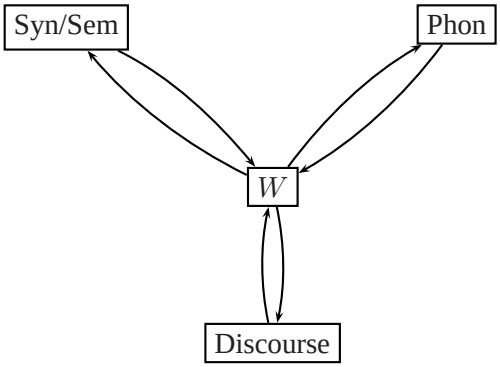


Figure 1: Modular grammar architecture proposed by Haji-Abdolhosseini (2003, 2005)

other modules. Oatley and Johnson-Laird (1987) and Oatley (1992) develop a theory of emotions within a larger context of cognitive science. This theory also relies on the fundamental assumption that human cognitive processes are modular and need to communicate with one another. In linguistics, Jackendoff (1992) also argues that human cognitive faculties form modules that need to communicate with one another. He also believes that each module (e.g., language, vision, or musical perception) is itself made up of its own sub-modules which in turn communicate with one another. Jackendoff (1997, 2002) argues for a tripartite architecture of grammar where phonological, morpho-syntactic and semantic components work in parallel and communicate at interface levels. Haji-Abdolhosseini (2003, 2005) takes a modular approach to HPSG where different linguistic modules interact through a shared list of domain objects.

However, disagreement among modules in any given intelligent system is a fact of life. Having crisp constraints, therefore, is not considered a desirable feature because as we gradually grow out of toy models and move towards approximating real-life problems, a system with crisp constraints quickly turns into what is known as an *over-constrained* system; i.e., one that yields no solution; or it becomes so complicated that it takes the system an inordinate amount of time to find an answer. In the AI community, several approaches have been proposed to remedy such problems. *Partial constraint satisfaction* (Freuder and Wallace, 1992), *constraint hierarchies* (Borning et al., 1992), *probabilistic soft constraint satisfaction* (Fargier and Lang, 1993), *valued constraint satisfaction* (Schiex et al., 1995), and *fuzzy soft constraint satisfaction* (Rosenfeld et al., 1976; Dubois et al., 1993; Ruttkay, 1994) are most notable.[1]

In computational linguistics, probabilistic approaches are dominant, and have led to some theoretical contributions (Abney, 1996, 1997; Bod, 1998; Bod, Hay and Jannedy, 2003a; Bod, Scha and Sima'an, 2003b; Foth, Menzel and Schröder,

[1]Some useful literature reviews can be found in Bistarelli (2001) and at `http://kti.ms.mff.cuni.cz/~bartak/constraints/` (accessed 9/11/2007).

2005; Schröder, 2002, among others). One notable approach within linguistics proper that relies on non-crisp constraints is OT.

Jackendoff (1997; 2002), who is a proponent of a modular approach, introduces *correspondence rules* that apply at the interfaces among major linguistic modules. Haji-Abdolhosseini (2005) observes that soft constraints tend to be interface constraints and hard constraints tend to be intramodular. This work advocates the use of soft constraints at interfaces.

From a practical point of view, there are additional reasons why it is important to do research in grammatical interfaces in constraint-based and multi-partite frameworks: A modular theory is easier for the researcher to work with. A grammar written in this approach is certainly more readable and more convenient to maintain. Furthermore, with the emergence of large-scale grammars a modular approach becomes even more significant to promote code readability and reuse.

2 A Generalized Theory of Constraint Satisfaction

Constraint programming has been a very exciting area of research in artificial intelligence in the past decade. The holy grail of constraint programming is to find ways of describing a problem in terms of constraints without having to worry about how those constraints are processed in finding a solution. This will allow one to concentrate on the problem as opposed to the details of algorithms and processing (for an excellent introduction, see Marriott and Stuckey, 1998). This constraint-based view of characterizing problems has also found its way into linguistics. Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag, 1987, 1994), Lexical Functional Grammar (LFG, Bresnan, 1982, 2001), and Optimality Theory (OT, Prince and Smolensky, 1993) are all constraint-based theories of language, and their claim is that by expressing linguistic generalizations in terms of constraints, we are better able to see the phenomena involved without getting entangled in procedural details.

The problem of over-constrained systems has led researchers to seek ways of relaxing or weighting constraints so that the less important ones can be violated in favor of the more important ones. This is also a route that OT has taken.

In the following subsection, we review a generalized theory of soft constraint satisfaction introduced by Bistarelli (2001). This theory is based on a certain algebraic structure called the *semiring*. Based on a solid mathematical foundation, Bistarelli's theory of Semiring-based Constraint Satisfaction Problems (SCSP) illustrates that several of the previous models of soft constraint satisfaction are instances of SCSP. The next section provides a formal introduction to constraint satisfaction problems in general, and section 2.2 introduces Bistarelli's semiring-based account. We will also outline an SCSP-based extension to HPSG type antecedent constraints in section 3.1.

2.1 Constraint Satisfaction Problems

This subsection is based on section 1.1 of Bistarelli (2001).

DEFINITION 2.1 ***Constraint Satisfaction Problem*** *A Constraint Satisfaction Problem is a sextuple* $\langle V, D, C, con, def, a\rangle$ *where*

- V *is a finite set of variables, i.e.,* $V = \{v_1, \ldots, v_n\}$*;*
- D *is a set of values, called* the domain*;*
- C *is a finite set of constraints, i.e.,* $C = \{c_1, \ldots, c_m\}$*.* C *is ranked, i.e.,* $C = \bigcup_k C_k$ *such that* $c \in C_k$ *if* c *involves* k *variables;*
- con *is called the* connection function *and it is such that*

$$con : \bigcup_k (C_k \rightarrow V^k),$$

  where $con(c) = \langle v_1, \ldots, v_k\rangle$ *is the tuple of variables involved in* $c \in C_k$*;*

- def *is called the* definition function *and it is such that*

$$def : \bigcup_k (C_k \rightarrow \wp(D^k)),$$

  where $\wp(D^k)$ *is the power set of* D^k*, that is, all the possible subsets of* k*-tuples in* D^k*;*

- $a \subseteq V$*, and represent the* distinguished *variables of the problem.*

con describes which variables are involved in which constraint; def specifies which are the domain tuples permitted by the constraint. The set a is used to point out the variables of interest in the given Constraint Satisfaction Problem (CSP), i.e., the variables for which we want to know the possible assignments, compatible with all the constraints. This set is equal to V if all the variables are of interest. This does not have to be the case however. In fact, it is reasonable to think that the CSP representation of a problem contains many details (in terms of constraints and/or variables) which are needed for a correct specification of the problem but are not important as far as the solution of the problem is concerned.

The solution $Sol(P)$ of a CSP $P = \langle V, D, C, con, def, a\rangle$ is defined as the set of all instantiations of the variables in a which can be extended to instantiations of all the variables which are consistent with all the constraints in C.

DEFINITION 2.2 ***Tuple Projection and CSP Solution*** *Given a tuple of domain values* $\langle v_1, \ldots, v_n\rangle$*, consider a tuple of variables* $\langle x_{i1}, \ldots, x_{im}\rangle$ *such that for all* $j = 1, \ldots, m$*, there exists a* $k_j \in \{1, \ldots, n\}$ *such that* $x_{ij} = x_{kj}$*. Then the projection of* $\langle v_1, \ldots, v_n\rangle$ *over* $\langle x_{i1}, \ldots, x_{im}\rangle$*, written* $\langle v_1, \ldots, v_n\rangle_{|\langle x_{i1}, \ldots, x_{im}\rangle}$*, is the*

tuple of values $\langle v_{i1}, \ldots, v_{im} \rangle$*. The solution* $Sol(P)$ *of a CSP* $P = \langle V, D, C, con, def, a \rangle$ *is defined as*

$$\left\{ \langle v_1, \ldots, v_n \rangle_{|a} \text{ such that } \left\{ \begin{array}{l} v_i \in D \text{ for all } i; \\ \text{for all } c \in C, \langle v_1, \ldots, v_n \rangle_{|con(c)} \in def(c). \end{array} \right. \right\}$$

The solution to a CSP is therefore an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. We may want to find just one solution, with no preference as to which one, or all solutions.

2.2 A Semiring-Based Theory of Constraint Satisfaction Problems

The constraint satisfaction approach defined above clearly employs crisp constraints, which can lead to over-constrained problems. To solve the problem of over-constrained CSPs, researchers have proposed several alternative approaches which enable one to relax some constraints in order to find a solution to the problem. As mentioned earlier, Bistarelli (2001) shows that some of these approaches (e.g., probabilistic, fuzzy, and weighted CSPs) can be thought of as special instances of a more general soft-constraint satisfaction framework, which he calls the Semiring-based Constraint Satisfaction Problems (SCSP). The present and the following sections, which are based on Chapter 2 of Bistarelli (2001), briefly introduce this theory.

Bistarelli's main idea is that

> ... a semiring (that is, a domain plus two operations satisfying certain properties) is all that is needed to describe many constraint satisfaction schemes. In fact, the domain of the semiring provides the levels of consistency (which can be interpreted as cost, or degree of preference, or probabilities, or others), and the two operations define a way to combine constraints together. More precisely, we define the notion of constraint solving over any semiring. Specific choices of the semiring will then give rise to different instances of the framework, which may correspond to known or new constraint solving schemes.

DEFINITION 2.3 **Semiring** *A semiring is a quintuple* $\langle A, sum, \times, 0, 1 \rangle$ *such that*

- A *is a set and* $0, 1 \in A$*;*
- sum*, called the* additive operator*, is a commutative, i.e.,* $sum(a, b) = sum(b, a)$*, and associative, i.e.,* $sum(a, sum(b, c)) = sum(sum(a, b), c)$*, operation with* 0 *as its identity element, i.e.,* $sum(a, 0) = a = sum(0, a)$*;*
- $\times$*, called the* multiplicative operator*, is an associative operation such that* 1 *is its* identity element *and* 0 *is its* absorbing element*, i.e.,* $a \times 0 = 0 = 0 \times a$*;*
- $\times$*, distributes over* sum*, i.e., for any* $a, b, c \in A, a \times sum(b, c) = sum((a \times b), (a \times c))$.

The reader may have noted that the set of real numbers between 0 and 1 (inclusive) together with arithmetic $+$ and $\times$ form a semiring, for example.

Bistarelli introduces semirings with additional properties for the two operations. He calls this algebra a *c-semiring* (c for "constraint"), and defines it as follows:

DEFINITION 2.4 **C-Semiring** *A c-semiring is a quintuple* $\langle A, \oplus, \otimes, 0, 1\rangle$ *such that*

- A *is a set and* $0, 1 \in A$*;*
- $\oplus$ *is defined over (possibly infinite) sets of elements of* A *as follows:*[2]
  - *for all* $a \in A, \sum(\{a\}) = a$*;*
  - $\sum(\emptyset) = 0$ *and* $\sum(A) = 1$*;*
  - $\sum(\bigcup A_i, i \in I) = \sum(\{\sum(A_i), i \in I\})$ *for all sets of indices* I *(flattening property);*
- $\otimes$ *is a binary associative and commutative operation such that* 1 *is its identity element and* 0 *is its absorbing element;*
- $\otimes$ *distributes over* $\oplus$*, i.e., for any* $a \in A$ *and* $b \subseteq A, a \otimes \sum(B) = \sum(\{a \otimes b, b \in B\})$.

The fact that $\oplus$ is defined over *sets* of elements, and not *pairs* or *tuples*, automatically makes such an operation commutative, associative, and idempotent. It is also possible to show that 0 is the identity element of $\oplus$. By using the flattening property, we get $\sum(\{a, 0\}) = \sum(\{a\}\emptyset) = \sum(\{a\}) = a$. This means that a c-semiring is a semiring (where the sum operation is $\oplus$) with some additional properties. It is also possible to prove that 1 is the absorbing element of $\oplus$. By flattening and by the fact that we set $\sum(A) = 1$, we get $\sum(\{a, 1\}) = \sum(\{a\} \cup A) = \sum(A) = 1$.

According to Bistarelli, the advantage of using c-semirings instead of semirings is as follows: The idempotency of the $\oplus$ operation is needed in order to define a partial ordering $\leq_s$ over the set A, which will enable us to compare different elements of the semiring. Such a partial order is defined as: $a \leq_s b$ iff $a \oplus b = b$. Intuitively, $a \leq_s b$ means that b is "better" than a, or, from another point of view, that between a and b, the $\oplus$ operation chooses b. This ordering is used to choose the "best" solution in constraint problems.

Given any c-semiring $S = \langle A, \oplus, \otimes, 0, 1\rangle$, consider the relation $\leq_s$ over A such that $a \leq_s b$ iff $a \oplus b = b$. Then Bistarelli proves that $\leq_s$ is a partial order. He also proves that $\oplus$ and $\otimes$ are monotones over $\leq_s$. That is, given any c-semiring $S = \langle A, \oplus, \otimes, 0, 1\rangle$, consider the relation $\leq_s$ over A. Then that $\oplus$ and $\otimes$ are monotones over $\leq_s$ means that $a \leq_s a'$ implies $a \oplus b \leq_s a' \oplus b$ and $a \otimes b \leq_s a' \otimes b$.

[2] We use $\oplus$ in infix notation for a two-element set, and the symbol $\sum$ in prefix notation for more elements.

Since 1 is also the absorbing element of the additive operation, then $a \leq_s 1$ for all a. Thus 1 is the maximum element of the partial ordering. This implies that the $\otimes$ operation is *intensive*, that is, $a \otimes b \leq_s a$. This is important since it means that combining more constraints leads to a "worse" result in terms of the $\leq_s$ ordering.

Sometimes we need the $\otimes$ operation to be closed on a certain finite subset of the c-semiring.

DEFINITION 2.5 **AD-closed** *Given any c-semiring $S = \langle A, \oplus, \otimes, 0, 1\rangle$, consider a finite set $AD \subseteq A$. Then $\otimes$ is AD-closed if for any $a, b \in AD, (a \otimes b) \in AD$.*

It is shown that c-semirings can be assimilated to complete lattices. We also sometimes need to consider c-semirings where $\otimes$ is idempotent, which makes the c-semiring equivalent to distributive lattices.[3]

DEFINITION 2.6 **LUB, GLB, (Complete Lattice)** *Consider a partially ordered set S and any subset I of S. Then we define the following:*

- *an* upper bound *(resp.* lower bound*) of I is any element x such that for all $y \in I, y \leq x$ (resp., $x \leq y$);*
- *the* least upper bound (LUB) *(resp.* greatest lower bound (GLB) *of I is an upper bound (resp. lower bound) x of I such that for any other upper bound (resp. lower bound) x' of I, we have that $x \leq x'$ (resp., $x' \leq x$).*

A lattice *is a partially ordered set where every subset of two elements has a LUB and a GLB. A* complete lattice *is a partially ordered set where every subset has a LUB and GLB.*

Bistarelli proves that $\langle A, \leq_s\rangle$ is a complete lattice, which entails $\sum(I) = LUB(I)$ for any set $I \subseteq A$. Thus every subset I of A has a least upper bound (which coincides with $\sum(I)$). This means that $\langle A, \leq_s\rangle$ is a LUB-complete partial order. Note that the $\oplus$ operator coincides with the LUB of the lattice $\langle A, \leq_s\rangle$.

Bistarelli also proves that given a c-semiring $S = \langle A, \oplus, \otimes, 0, 1\rangle$ and a corresponding complete lattice $\langle A, \leq_s\rangle$, $\otimes$ is also idempotent. Furthermore, in the particular case in which $\otimes$ is idempotent and $\leq_s$ is total, we have that $a \oplus b = \max(a, b)$ and $a \otimes b = \min(a, b)$.

2.3 Constraint Systems and Problems

The notions of constraint system, constraint, and constraint problem in this theory are parametric with respect to the notion of c-semiring discussed in the previous section. Intuitively, a constraint system specifies the c-semiring $\langle A, \oplus, \otimes, 0, 1\rangle$ to be used along with the set of all variables and their domain D.

[3] For an introduction to lattices and ordered sets, see Davey and Priestley (1990).

DEFINITION 2.7 **Constraint System** *A constraint system is defined as a triple $CS = \langle S, D, V \rangle$, where S is a c-semiring, D is a finite set, and V is an ordered set of variables.*

A constraint over a given constraint system specifies the involved variables and the "allowed" values for them. More precisely, for each tuple of values (of D) for the involved variables, a corresponding element of A is given. This element can be interpreted as the tuple's weight, or cost, or level of confidence, etc.

DEFINITION 2.8 **Constraint** *Given a constraint system $CS = \langle S, D, V \rangle$, where $S = \langle A, \oplus, \otimes, 0, 1 \rangle$, a constraint over CS is a pair $\langle def, con \rangle$, where*

- *$con \subseteq V$, it is called the* type *of the constraint;*
- *$def : D^k \rightarrow A$ (where k is the cardinality of con) is called the* value *of the constraint.*

A constraint problem is then just a set of constraints over a given constraint system, plus a selected set of variables (thus a *type*). These are the variables of interest in the problem, i.e., the variables for which we want to know the possible assignments compatibly with all the constraints.

2.4 Instances of the SCSP Framework

Having laid out the c-semiring-based theory of constraint satisfaction, Bistarelli shows that some of the previous constraint satisfaction approaches can be seen as instances of this theory differing only in the choice of the semiring. Below I list the different CSPs and the semirings used in them as discussed by Bistarelli.

- **Classical CSPs:** A classical CSP is just a set of variables and constraints, where each constraint specifies the tuples that are allowed for the involved variables. Since the constraints in a CSP are crisp, they can be modeled with a semiring containing only 0 and 1 in A. Also we can model constraint combination with logical *and*, and the projection over some of the variables (to obtain the value of the tuples of the variables in the type of the problem) with logical *or*. Thus, a CSP can be seen as just an SCSP with the following c-semiring:

$$\mathrm{S_{CSP}} = \langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$$

- **Fuzzy CSPs:** Fuzzy CSPs allow for non-crisp constraints, which associate a preference level with each tuple of values. This level of preference is always between 0 and 1. The solution to a fuzzy CSP is defined as the set of tuples of values for all the variables which have the maximal value. Fuzzy CSPs can be modeled in the SCSP framework by choosing the following c-semiring:

$$\mathrm{S_{FCSP}} = \langle \{x | x \in [0, 1]\}, \max, \min, 0, 1 \rangle$$

- **Probabilistic CSPs:** In probabilistic CSPs, each constraint c has an associated probability $p(c)$. Saying that c has probability p, means that the situation corresponding to c has probability p of occurring in the real-life problem. The c-semiring corresponding to the probabilistic CSPs is as follows:

$$\mathrm{S}_{\mathrm{prob}} = \langle \{x | x \in [0,1]\}, \max, \times, 0, 1 \rangle$$

- **Weighted CSPs:** Contrary to fuzzy CSPs whose constraints come with preferences, in weighted CSPs, constraints have associated costs. The solution to a problem in such models is the one with minimum cost (e.g., time, space, number of resources, etc.). Therefore, the associated c-semiring for a weighted CSP is the following:

$$\mathrm{S}_{\mathrm{WCSP}} = \langle \mathbb{R}^*, \min, +, +\infty, 0 \rangle$$

- **Set-Based CSPs:** The SCSP framework gives rise to an interesting class of its instances that are based on set operations such as union and intersection. The corresponding c-semiring for this class of CSPs is this:

$$\mathrm{S}_{\mathrm{set}} = \langle \wp(A), \bigcup, \bigcap, \emptyset, A \rangle$$

This section presented a brief overview of Bistarelli's c-semiring-based generalized theory of soft constraint satisfaction systems. As Bistarelli shows, many previous CLP approaches to soft constraints are in fact instances of this generalized framework, which is parametric with respect to the semiring used. In the next section, we will show that an instance of this theory, the *weighted soft constraint satisfaction* approach is suitable for modeling linguistic constraints.

3 Soft Linguistic Constraints

This section outlines a theory of linguistic soft constraint satisfaction based on the SCSP framework (the c-semiring-based theory of Soft Constraint Satisfaction Problems).

Section 3.1 briefly talks about how a c-semiring-based approach might be incorporated into a unification-based theory of grammar. We can think of a grammar in SCSP terms as a constraint system, $CS = \langle S, D, V \rangle$, where $S = \langle A, \oplus, \otimes, 0, 1 \rangle$ is a semiring, and V is a set of variables characterizing the candidate. D is a finite set of values that the variables in V can take. Therefore, a constraint over this CS is a tuple $\langle def, con \rangle$ such that $con \subseteq V$ and $def : D^k \rightarrow A$. Values in the carrier set A correspond to the overall compliance of a candidate linguistic structure, can, with the whole constraint system.

The function def in SCSP, takes the vector representation D^k of the candidate and maps it to a global valuation. Figure 2 shows how Can is mapped to A. Embedding applies to $can \in Can$ returning a vector of features, which is passed to

$\bar{c}_i$, for $1 \leq i \leq m$ perhaps, which returns a vector of valuations. These valuations are then combined (in this case, weighted and summed up) by the global valuation function g returning a value in A.

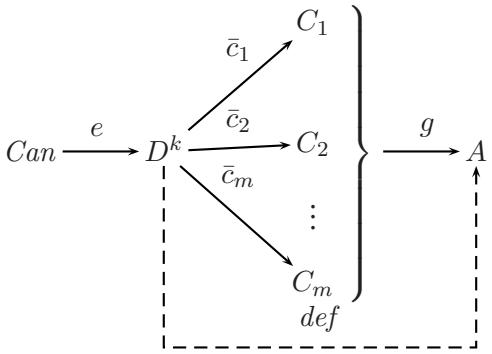


Figure 2: Global valuation calculation of candidate structures

The optimal candidate, to use optimality theoretic terminology, will be the one that has the smallest value for its global valuation; that is, the candidate with a global valuation closest to zero is optimal and the ones with larger values are increasingly suboptimal (in other words, $\mathcal{V}(can)$ represents a cost). In this context, the semiring used will be the one shown below in (1), where $\mathbb{R}^*$ is the set of non-negative real numbers, min chooses the solution, and $+$ combines the values in the carrier set A, (which in this semiring is $\mathbb{R}^*$). Absolute consistency is denoted by 0 and absolute inconsistency by $+\infty$.

(1) $$S_{\mathrm{SCSP}} = \langle \mathbb{R}^*, \min, +, +\infty, 0 \rangle$$

A semiring with $\min$ as the additive operation and $+$ as the multiplication operation with $+\infty$ and 0 corresponding to 0 and 1, respectively, is also known as the *tropical semiring*. S_{V} is a c-semiring since the additive operation, $\min$, is idempotent (i.e., $\min(a, a) = a$ for all a), and the multiplicative operation, $+$, is commutative. Also, $+\infty$ is the identity element for $\min$ (i.e., $\min(a, +\infty) = a$ for all a), and 0 is the identity element for $+$, (i.e., $a + 0 = a$ for all a). $\mathbb{Z}^*$ is the set of non-negative integers (including 0). The associated ordering $\leq_s$ corresponds to $\geq$ over non-negative integers, which means that smaller numbers correspond to better candidates.

A linguistic constraint system is then defined as follows:

DEFINITION 3.1 *A linguistic constraint system based on SCSP, $CS = \langle S_{\mathrm{SCSP}}, D, V \rangle$, will then have the following components:*

- **C-Semiring:** $S_{\mathrm{SCSP}} = \langle \mathbb{R}^*, min, +, +\infty, 0 \rangle$.
- **Variables:** *An ordered set V representing the candidate structure.*

- **Domain:** D *a finite set of values that members of* V *can take.*

DEFINITION 3.2 **Connection:** $con \subseteq V$*, is called the type of the constraint.*

The set con tells us which variables are involved in each constraint.

DEFINITION 3.3 **Domain Function:** $dom : V \rightarrow \mathsf{D}$*, where* $\mathsf{D} \subseteq D$*.*

The domain function dom tells us which members of D can be assigned to each member of V.

DEFINITION 3.4 **Embedding:** $e : Can \hookrightarrow D^k$ *is a bijective function that maps the set of linguistic structures onto vector representations; in particular, for all* $\vec{d} \in D^k$*,* $d_i \in dom(v_i)$*, where* $1 \leq i \leq k$*.*

Embedding ensures a representation of linguistic structures that is suitable for the constraint solver. The embedding function must be bijective because once the solver returns a vector as the solution, we want to be able to identify a candidate with that vector.

Bistarelli (2001) also defines a function def as follows:

DEFINITION 3.5 **Definition:** $def_k : D^k \rightarrow \mathbb{R}^*$

This function is actually the composition of $\bar{c}_i$ with g (see Figure 2 above and definition 3.9 below), that is, $def = g \circ \bar{c}_i$. Based on this definition, a constraint in SCSP is defined as follows:

DEFINITION 3.6 **Constraint:** $\langle \bar{c}_i, con^n \rangle$*, for some* $1 \leq n \leq k$ *where* n *is the cardinality of* con*.*

DEFINITION 3.7 **Constraint Weight:** w_i *is a numerical weight associated with each constraint* $\langle \bar{c}_i, con_i^n \rangle$*. In OT literature, this is known as the* rank *of the constraint.*

DEFINITION 3.8 **Valuation:** $\bar{c}_i : D^k \rightarrow C_i$*, a function that evaluates the value* d *of each variable* v*.*

DEFINITION 3.9 **Global Valuation:** $g : C_1 \times C_2 \times \ldots \times C_m \rightarrow \mathbb{R}^*$ *is the combination function that calculates the global valuation of* $\vec{d}$ *based on a vector of valuations returned by all of the* $\bar{c}_i$*. In this model,* $g(c_1, c_2, \ldots, c_m) = \Sigma_{i=1}^m w_i c_i$ *for all* $c_i \in C_i$*.*

3.1 Toward Graded Unification-Based Grammars

In section 1, we talked about the benefits of a parallel modular grammar architecture saying that such an architecture leads to simpler modules and captures generalizations better. One important advantage of implementing such an architecture in a unification-based framework is that unification naturally allows for the modules to constrain one another. Through structure-sharing, even though the modules may not care about nor see the details inside other modules, they cannot generate structures that are unacceptable to other modules. It would then be natural to try to implement the proposed soft-constraint satisfaction system in a unification-based framework such as HPSG. In order to do this, we need to change how type antecedent constraints are enforced without modifying the unification mechanism. Standard HPSG type antecedent constraints are crisp; their violation causes the generated structure to be rejected. Multiple constraints on a type are explicitly connected by logical AND, which also means the violation of any one constraint results in the rejection of the generated structure. This constraint system roughly corresponds to Bistarelli's $\mathrm{S}_{\mathrm{CSP}}$ mentioned in section 2.4.

Malouf (2003) argues that the fact that an analysis naturally falls out of OT's notion of ranked violable constraints does not necessarily mean that it *has* to be analyzed that way. He states that OT suffers from a "procedural metaphor;" that is, the theory relies on some cognitively unreal and intractable operations to account for acceptable structures. The most notable part of this metaphor is the generate-and-test procedure of the theory where a partial representation (such as a logical form) is fed to a component called *Gen* that generates a potentially infinite number of candidate output structures to be evaluated against a set of constraints by *HEval*. This is a common concern. As a solution, Malouf discards the procedural metaphor along with the violability of the constraints, and accounts for his data using an HPSG type hierarchy. His analysis, albeit elegant, does not leave any room for graded grammaticality judgments, accounting for multiple violations of the same constraint, and ganging up effects, not to mention the graded constraints discussed by Haji-Abdolhosseini (2005). This section shows that we can incorporate soft constraints within constraint-based grammars such as HPSG without resorting to any procedural metaphors.

In order to account for violable constraints as well as degrees of constraint violation, multiple constraint violations, and ganging up effects discussed in Keller (2000), we can use the weighted CSP paradigm defined in the previous section ($\mathrm{S}_{\mathrm{WCSP}} = \langle \mathbb{R}^*, min, +, +\infty, 0 \rangle$).

We now go over some illustrative examples. It should be mentioned that the goal of the following examples is not to derive the "correct" analysis but to show how a system of type antecedent constraints based on the tropical semiring would calculate costs for different analyses.

In the case of sentences (2) and (3), we can formulate a constraint on head-complement phrases (*hd-comp-ph*) as in Figure 3. For simplicity of exposition, this constraint only employs two non-head daughters. The extension of the constraint

to accommodate more daughters is straightforward.

(2) a. He wanted to demonstrate it to us.
b. He wanted to demonstrate that life to us.
c. He wanted to demonstrate the consequences to us.
d. ? He wanted to demonstrate the consequences of such an unholy life to us.
e. ?? He wanted to demonstrate the consequences of such a horribly filthy and unholy life to us.

(3) a. * He wanted to demonstrate to us it.
b. ?? He wanted to demonstrate to us that life.
c. He wanted to demonstrate to us the consequences of such an unholy life.

$$\textit{hd-comp-ph} \Rightarrow \begin{bmatrix} \text{PHON} & \langle \boxed{1}, \boxed{3} \rangle \\ \text{NON-HD-DTRS} & \left\langle \begin{bmatrix} \text{PHON} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{PHON} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix} \wedge length\Big(\boxed{1}\Big) \leq length\Big(\boxed{2}\Big)$$

Figure 3: An HPSG formulation of the LH constraint on verb complements

The valuation function for the LH constraint as formulated above can be calculated according to the following function:

(4) **Valuation Function for LH:**

Given the description $\begin{bmatrix} \textit{hd-comp-ph} & \\ \text{PHON} & \boxed{3} \oplus \langle \boxed{1}, \boxed{2} \rangle \\ \text{NON-HD-DTRS} & \left\langle \begin{bmatrix} \text{PHON} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{PHON} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$,

$$val(\text{LH}) = \frac{length(\boxed{1})}{length(\boxed{1}) + length(\boxed{2})}$$

Let us assume, for now, that $length(x)$ is the number of words in x, and that the weight of the constraint LH is 1. The formula in (4) returns a number between 0 and 1. If the two complements are of equal size, the valuation returned will be 0.5; the valuation approaches 1 as the first complement gets longer than the second, and it approaches 0 as the second complement gets longer than the first. Of course, one can think of many ways to formulate LH. The definition presented here is just one of them. This formulation is flexible because it reflects the magnitude of the difference between the two complements. We can now see how the sentences in (2) are evaluated in terms of LH. The valuations of LH calculated for (2a–e) are shown in (5) below.

(5) a. For (2a): $val(\text{LH}) = \frac{1}{1+2} \approx .33$
b. For (2b): $val(\text{LH}) = \frac{2}{2+2} = .5$
c. For (2c): $val(\text{LH}) = \frac{2}{2+2} = .5$
d. For (2d): $val(\text{LH}) = \frac{7}{7+2} \approx .78$
e. For (2e): $val(\text{LH}) = \frac{10}{10+2} \approx .83$

As can be seen, this accounts for the declining acceptability of the examples in (2).

The examples in (3) demonstrate the interaction of two constraints: (i) LH, and (ii) the constraint that requires verbal complements to appear in descending order of obliqueness (call it COMPORD). If obliqueness is a total order defined over verbal complements represented with $>_o$, then we can formulate COMPORD as in Figure 4. The valuation function for COMPORD is defined in (6).

$$\textit{hd-comp-ph} \Rightarrow \left[\text{NON-HD-DTRS} \left\langle \boxed{1}, \boxed{2} \right\rangle\right] \wedge \boxed{1} >_o \boxed{2}$$

Figure 4: An HPSG formulation of COMPORD

(6) **Valuation Function for** COMPORD**:**

Given the description $\begin{bmatrix} \textit{hd-comp-ph} \\ \text{NON-HD-DTRS} \quad \left\langle \boxed{1}, \boxed{2} \right\rangle \end{bmatrix}$,

$$val(\text{COMPORD}) = \begin{cases} 0 & \text{iff } \boxed{1} >_o \boxed{2} \\ 1 & \text{otherwise} \end{cases}$$

$val(\text{COMPORD})$ is defined as a *characteristic* or *selector* function returning either 0 or 1, but notice that since we are using the tropical semiring these values do not have their traditional *true* or *false* meaning. In this constraint system, 0 corresponds to no violation and 1 corresponds a violation of degree 1. Also notice, since we are adding costs, multiple instances of such constraint violations will incrementally increase global evaluation as it does in Linear Optimality Theory (Keller, 2000). Let us assume that the two constraints LH and COMPORD have equal weights. Then the valuation of the sentences in (3a–c) with respect to LH and COMPORD is calculated as in (7).

(7) a. For (3a): $val(\text{LH}) + val(\text{COMORD}) \approx .66 + 1 = 1.66$
b. For (3b): $val(\text{LH}) + val(\text{COMORD}) = .50 + 1 = 1.50$
c. For (3c): $val(\text{LH}) + val(\text{COMORD}) \approx .22 + 1 = 1.22$

This analysis quantitatively captures the increasing acceptability of the sentences in (3) as the sentence-final direct object gets longer than the indirect object.

An interesting outcome of this analysis is that it naturally captures speakers' intuitions about the relative acceptability of forms like the ones in (8) without the need to posit an arbitrary constraint prohibiting ending a dative construction with

$$\textit{clause} \Rightarrow \begin{bmatrix} \text{DOM} & \boxed{1} \oplus \boxed{2} \\ \text{INFO} & \left\langle \begin{bmatrix} \textit{theme} \\ \text{I-DOM} \quad \boxed{1} \end{bmatrix}, \begin{bmatrix} \textit{rheme} \\ \text{I-DOM} \quad \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

Figure 5: An HPSG formulation of THRH

a pronoun (which would completely rule out (8b), incorrectly). According to this analysis, we not only capture the graded grammaticality of each example, we can also show how much each example is worse than the other.[4]

(8) a. Give it to me.
$val(\text{LH}) + val(\text{COMPORD}) \approx .33 + 0 = .33$

b. ?? Give me it.
$val(\text{LH}) + val(\text{COMPORD}) = .5 + 1 = 1.5$

c. * Give to me it.
$val(\text{LH}) + val(\text{COMPORD}) \approx .66 + 1 = 1.66$

To incorporate more modules, let us now consider how information structure can be integrated into this model. Let THRH stand for the violable constraint that requires the theme to appear before the rheme. A version of this constraint for just one theme and one rheme is shown in Figure 5;[5] an extension to the constraint for multiple themes and rhemes is also straightforward. The valuation function for THRH is formulated in (9).

(9) **Valuation Function for** THRH**:**
Given the description $\begin{bmatrix} \textit{clause} \\ \text{INFO} \quad \left\langle \boxed{1}, \boxed{2} \right\rangle \end{bmatrix}$,
$val(\text{THRH}) = \begin{cases} 0 & \text{iff } type(\boxed{1}) = \textit{theme} \wedge type(\boxed{2}) = \textit{rheme} \\ 1 & \text{otherwise} \end{cases}$

Let us assume that the preferred response to the question "What did John give to the man?" is (10a) as opposed to (10b).[6]

(10) a. [He gave]$_{\text{theme}_1}$ [money]$_{\text{rheme}}$ [to the man]$_{\text{theme}_2}$.
b. [He gave the man]$_{\text{theme}}$ [money]$_{\text{rheme}}$.

[4]Note that we are assuming equal weights for these constraints. Estimating the exact weights of the constraints requires having access to training data obtained through corpus analysis or experimental work. I shall leave this for future research.

[5]I am following the analysis in Haji-Abdolhosseini (2003, 2005) where *sign* has the features DOM and INFO taking a list of lexical items and *info* objects, respectively. The types *theme* and *rheme* are subtypes of *info*.

[6]Again note that we are not making any strong claims as to what sentence is actually the preferred response. This should be determined through separate studies. The point here is to illustrate how valuation calculations work.

Again assuming equal weights for LH, COMPORD, and THRH, we can calculate the global valuations of these sentences with respect to these three constraints as in (11). It can be seen that (10a) gets a lower global valuation (i.e., is preferred by the model).

(11) a. For (10a): $val(\text{LH}) + val(\text{COMPORD}) + val(\text{THRH}) = .25 + 0 + 1 = 1.25$

b. For (10b): $val(\text{LH}) + val(\text{COMPORD}) + val(\text{THRH}) \approx .66 + 1 + 0 = 1.66$

In this example, THRH has been violated in favor of COMPORD and LH. Note that since the difference in the lengths of the two verb complements is small the two sentences show a small difference in their global valuation (0.41).

Let us look at another example in which the difference in the lengths of the verb complements is larger.

(12) a. [He gave]$_{\text{theme}_1}$ [a lot of his hard earned money]$_{\text{rheme}}$ [to the man]$_{\text{theme}_2}$.

b. [He gave the man]$_{\text{theme}}$ [a lot of his hard earned money]$_{\text{rheme}}$.

The global valuations of these sentences are given below:

(13) a. For (12a): $val(\text{LH}) + val(\text{COMPORD}) + val(\text{THRH}) = .70 + 0 + 1 = 1.70$

b. For (12b): $val(\text{LH}) + val(\text{COMPORD}) + val(\text{THRH}) \approx .22 + 1 + 0 = 1.22$

Here we see that (12b) is preferred. We also see that the difference between the global valuations of (12a) and (12b) is larger than before (0.48), which means that in this example the alternative is costlier than in the previous example. In other words, the gradient characterization of LH captures the fact that larger differences in the lengths of the complements result in higher degrees of constraint violation if the heavier constituent appears before the lighter one, an observation made by Arnold et al. (2000) as well as Haji-Abdolhosseini (2005). In addition, the c-semiring-based implementation of type antecedent constraints in HPSG allows for capturing the ganging up effects of constraint violation as well as multiple violations of the same constraint (since valuations are summed up).

The cost of a feature structure, f, of type τ is the weighted sum of the valuations of all the constraints imposed on τ with respect to f plus the sum of the costs of all the feature values of f. This is formalized in (14).

(14) $cost(f_\tau) = \sum_i w_i \cdot val(c_i^\tau) + \sum_j cost(g_j)$
where f_τ is the feature structure of type τ to which we want to assign a cost; c_i^τ is a constraint on the type τ; $val(c_i^\tau)$ is the valuation of c_i^τ; and g_j is a feature value of f.

The formula in (14) implies that the cost of a feature structure is never less than the sum of the costs of its substructures (provided that there are no negative

weights, which is what we have been assuming). If there are any cases where the same description gets different valuations in different contexts (i.e., in different feature structures), then we can replace our original constraint with other more specific constraints.

The reason for this desideratum is twofold: (i) We want to make sure that every part of the feature structure is contributing information about constraint violations in substructures; and (ii) we want the constraints to be local; that is, every constraint has to be sensitive only to the description on its consequent and should not be affected by the context in which it is applied. For instance, consider the following feature structures:

(15) a. $\begin{bmatrix} t & \\ \text{F} & a \end{bmatrix}$

b. $\begin{bmatrix} u & \\ \text{G} & a \end{bmatrix}$

If the cost of a feature structure f of type a depends on whether f is the value of F or G, then we cannot formulate a single constraint on type a. This is because such a constraint on a would be non-local as it would have to have information about whether f occurs in a feature structure of type t or u. In order to keep our constraint local in this case, we must formulate two constraints on t and u making reference to the value of the F and G features, respectively.

4 Conclusion

This paper incorporated a generalized c-semiring-based theory of soft constraint satisfaction within a constraint-based grammar architecture. The stipulation that soft constraints apply at interfaces while hard constraints apply inside modules is advantageous from a grammar engineering point of view. Linguistic modules can be developed separately and then put together allowing soft constraints to resolve conflicts among modules.

This work can be pursued in many different directions. One obvious way to follow up this work would be to work out the formal details of incorporating SCSP in a theory like HPSG. Another way to pursue would be to implement an SCSP-based constraint solver in a logic-programming language like ALE (Carpenter and Penn, 1999). One can also investigate different learning algorithms for an SCSP-based grammar.

References

Abney, Steven. 1996. Statistical Methods and Linguistics. In Judith Klavans and Philip Resnik (eds.), *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, Cambridge, MA: The MIT Press.

Abney, Steven. 1997. Stochastic Attribute-Value Grammars. *Computational Linguistics* 23(4), 597–618.

Arnold, Jeniffer E., Wasow, Thomas, Losongco, Anthony and Ginstrom, Ryan. 2000. Heaviness vs. Newness: The effects of structural complexity and discourse status on constituent ordering. *Language* 76(1), 28–55.

Bistarelli, Stefano. 2001. *Soft Constraint Solving and Programming: A General Framework*. Ph.D. thesis, Università di Pisa.

Bod, Rens. 1998. *Beyond Grammar: An Experience-Based Theory of Language*. CSLI Publications, Cambridge University Press.

Bod, Rens, Hay, Jennifer and Jannedy, Stefanie (eds.). 2003a. *Probabilistic Linguistics*, Cambridge, MA, MIT Press.

Bod, Rens, Scha, Remko and Sima'an, Khalil (eds.). 2003b. *Data-Oriented Parsing*. CSLI Publications.

Borning, Alan, Freeman-Benson, Bjorn and Wilson, Molly. 1992. Constraint Hierarchies. *Lisp and Symbolic Computation* 5(3), 223–270.

Bresnan, Joan (ed.). 1982. *The Mental Representation of Grammatical Relations*. Cambridge, MA: The MIT Press.

Bresnan, Joan. 2001. *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics, Malden, MA: Blackwell.

Carpenter, Bob and Penn, Gerald. 1999. ALE The Attribute Logic Engine: User's Guide, available online at `www.cs.toronto.edu/~gpenn/ale/files/aleguide.ps.gz`.

Davey, Brian A. and Priestley, Hilary A. 1990. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge: Cambridge University Press.

Dubois, Didier, Fargier, Hélèn and Prade, Henri. 1993. The Calculus of Fuzzy Restrictions as a Basis for Flexible Constraint Satisfaction. In *Proceedings of IEEE International Conference on Fuzzy Systems*, pages 1131–1136, IEEE.

Fargier, Hélèn and Lang, Jérôme. 1993. Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach. In *Proceedings of the European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU), number 747 in LNCS*, pages 97–104, Springer-Verlag.

Foth, Kilian, Menzel, Wolfgang and Schröder, Ingo. 2005. Robust Parsing with Weighted Constraints. *Natural Language Engineering* 11(1), 1–25.

Freuder, Eugene C. and Wallace, Richard J. 1992. Partial Constraint Satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89*, volume 58, Detroit, MI.

Haji-Abdolhosseini, Mohammad. 2003. A Constraint-Based Approach to Information Structure and Prosody Correspondence. In Stefan Müller (ed.), *Proceedings of The HPSG-2003 Conference*, pages 143–162, `http://cslipublications.stanford.edu/HPSG/4/`.

Haji-Abdolhosseini, Mohammad. 2005. *Modularity and Soft Constraints: A Study of Conflict Resolution in Grammar*. Ph.D. Thesis, University of Toronto.

Jackendoff, Ray. 1992. *Languages of the Mind: Essays on Mental Representation*. The MIT Press.

Jackendoff, Ray. 1997. *The Architecture of the Language Faculty*. Linguistic Inquiry: Monograph Twenty-Eight, Cambridge, Mass.: The MIT Press.

Jackendoff, Ray. 2002. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. New York, NY: Oxford.

Keller, Frank. 2000. *Gradience in Grammar: Experimental and Computational Aspects of Degrees of Grammaticality*. Ph.D. thesis, University of Edinburgh.

Malouf, Robert P. 2003. Cooperating Constructions. In E. Francis and L. Michaelis (eds.), *Mismatch: Form-function Incongruity and the Architecture of Grammar*, pages 403–424, Stanford: CSLI Publications.

Marriott, Kim and Stuckey, Peter J. 1998. *Programming with Constraints: An Introduction*. Cambridge, MA: The MIT Press.

Newell, Allan. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Oatley, Keith. 1992. *Best Laid Schemes: The Psychology of Emotions*. Cambridge: Cambridge.

Oatley, Keith and Johnson-Laird, Philip N. 1987. Towards a Cognitive Theory of Emotions. *Cognition and Emotions* 1, 29–50.

Pollard, Carl and Sag, Ivan A. 1987. *Information-Based Syntax and Semantics, Volume I: Fundamentals*. CSLI Lecture Notes, No. 13, Stanford: CSLI Publications.

Pollard, Carl and Sag, Ivan A. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics, Chicago: CSLI Publications.

Prince, Alan and Smolensky, Paul. 1993. Optimality Theory: Constraint Interaction in Generative Grammar. Technical Report 2, Rutgers University Center for Cognitive Science, Piscataway, NJ.

Rosenfeld, Azriel, Hummel, Robert A. and Zucker, Steven W. 1976. Scene Labelling by Relaxation Operations. *IEEE Transactions on Systems, Man, and Cybernetics* 6(6), 420–433.

Ruttkay, Zsofi. 1994. Fuzzy Constraint Satisfaction. In *Proceedings of the 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268.

Schiex, Thomas, Fargier, Hélèn and Verfaille, Gérard. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of IJCAI95*, pages 631–637, Morgan Kaufman.

Schröder, Ingo. 2002. *Natural Language Parsing with Graded Constraints*. Ph. D.thesis, Fachbereich Informatik der Universität Hamburg.

A Morpho-Syntactic Analyzer of Controlled Japanese

Yukiko Sasaki Alam

Department of Digital Media
Hosei University, Tokyo

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

The proposed morpho-syntactic analyzer parses controlled Japanese texts such as articles in newspapers, technical magazines and professional journals and public documents that are transcribed wherever applicable by using *Joyo Kanji* (frequently used Chinese characters). The analyzer parses sentences in controlled Japanese texts into morpho-syntactic units, further dividing them into the content and the functional parts, and assigning a functional role or roles to each unit in the sentences. As the system is not equipped with a dictionary, the parsing algorithm is based on the orthographic characteristics of words and morphemes, and the role assignment to each unit is based on the functional elements located at the end of the unit, which is a feature of a Head-final language like Japanese. The system is a light-weight rule-based morpho-syntactic analyzer that could be a useful tool for natural language processing. As the system identifies syntactic units rather than individual morphemes, together with the functional and/or syntactic roles of the units, it would help a computational system understand the syntactic and functional structures of sentences, and eventually interpret the semantics of the sentences.

1 Introduction

There being no spaces between words in Japanese, a main concern of Japanese morphological and syntactic analyzers has been word segmentation. Word-breaking is a fundamental task in natural language processing for Japanese, and various approaches have been taken. While many morphological analyzers, notably *Juman* (Kurohashi and Nagao, 2003) and *Chasen* (Matsumoto et al., 2000), have concentrated on the segmentation of morphemes (such as prefixes, suffixes, inflections, Case markers, particles and the components of compound words), the current analyzer focuses on the segmentation of phrases and the identification of the functional roles of the phrases in sentences.

The proposed system is intended to parse controlled Japanese texts which are written wherever applicable by using *Joyo Kanji* (frequently used Chinese Characters), the members of which are determined by the Ministry of Education and Science of Japan. Such texts typically include articles in newspapers, technical magazines and journals, and official documents. The analyzer draws on the information of the orthographic types of words and morphemes used in sentences as well as linguistic knowledge of functional morphemes and words.

In the past, other researchers have also developed morphological analyzers exploiting the information of orthographic types of words and morphemes: to name a few, Asahara (2003), Kazama (2001), Kashioka et al (1998), and

Kameda (1996). Unlike such previous studies, however, the main focus of the present analyzer is not on phrase segmentation per se, but on identifying the functional roles of phrases played in the sentences.

Futhermore, unlike Kudo (2002), Sekine (2001), Uchimoto (2000), Kanayama et al (2000), and Haruno et al (1999), all of which are statistically modeled systems, the current analyzer runs on a purely rule-based algorithm. The purpose of the present paper is to demonstrate that a light-weight rule-based analyzer can successfully identify phrases in sentences, and determine the functional roles of the phrases.

The paper first gives an overview of the current system, and then describes the algorithm used in the system. Before concluding, it discusses what are the difficulties faced by the current system, and what areas need further research.

2 Overview of the Morpho-Syntactic Analyzer

The current analyzer runs by referring to the different orthographic types of Japanese words and morphemes. Japanese sentences are transcribed in several orthographic types: *Kanji* (Chinese characters), *Katakana* (phonetic characters for words of foreign origin), *Hiragana* (phonetic characters for words of Japanese origin, inflections, particles, etc.), Arabic numerals, the Roman alphabet, special symbols and punctuation.

The most important feature used by the current analyzer is that most functional morphemes in Japanese are transcribed in *Hiragana*, including all the particles indicating Case markers, verbal inflections, auxiliaries, and suffixes indicating different types of clauses. In addition to their special orthographic feature, unexceptionally these functional elements are located at the end of phrases, [1] thus marking phrase boundaries. The current analyzer is based on these two characteristics, i.e. *Hiragana*-transcribed functional elements and their phrase-final positions.

A sequence of *Kanji* characters followed by *Hiragana* characters would be a good candidate for a phrase, which consists of a content word followed by a functional element, as illustrated below:

[$_{PP}$ [$_{NP}$ content word in *Kanji*] [$_{P}$ functional element in *Hiragana*]]

It is relatively straightforward to identify such phrases, as demonstrated by the example output of the analyzer in Table 1.

[1] This is because Japanese is a Head-final language where the non-Head content part is followed by the Head functional part at all the morphological and syntactic levels of Japanese including words, phrases and clauses.

TABLE 1 A successful output of the analyzer

研究グループの鈴木宏志教授によると、
kenkyuu-guruupu-no-suzuki-hiroshi-kyouju-niyoruto,
research-group-of-suzuki-hiroshi-professor-according-to,
全国の盲導犬協会から
zenkoku-no-moudouken-kyoukai-kara
entire-country-of-guide-dog-association-from
盲導犬の口腔粘膜や血液の提供を
moudouken-no-koukou-nenmaku-ya-ketsueki-no-teikyou-o
guide-dog-of-oral-membrane-and-blood-of-donation-OBJECT
受け、遺伝子を解析する。
uke, idenshi-o-kaiseki-suru
receiving, gene-OBJECT-analysis-do

'According to Professor Hiroshi Suzuki in the research group, they will analyze genes by receiving the oral membranes and the blood of guide dogs donated by the Associations of Guide Dogs in the entire country.'

CONTENT WORD	FUNCTION ELEMENT	GRAMMATICAL ROLE
研究グループ	*の*	名詞修飾句 (NOMINAL MODIFIER)
鈴木宏志教授	*によると*	出典 (SOURCE)
全国	*の*	名詞修飾句 (NOMINAL MODIFIER)
盲導犬協会	*から*	始点 (POINT OF DEPARTURE)
盲導犬	*の*	名詞修飾句 (NOMINAL MODIFIER)
口腔粘膜	*や*	列挙接続語 (CONJ – ETC)
血液	*の*	名詞修飾句 (NOMINAL MODIFIER)
提供	*を*	目的語 (OBJECT)
受	*け*	述語−接続形　(PREDICATE - CONJUNCTIVE)
中段　(Break)	、	読点　(COMMA)
遺伝子	*を*	目的語 (OBJECT)
解析	*する*	述語- 現在・未来　(PREDICATE - PRESENT or FUTURE)

Whenever a content word in *Kanji* (and/or *Katakana*) is followed only by a functional element in *Hiragana* (colored in red in Table 1), which is further

followed by another content word in *Kanji* (and/*or Katakana*), word and phrase boundaries are clearly distinguished as in:

[$_{PP}$ *KENKYU-GURUUPU* ('research group')-*no* (nominal modifier marker)]
[$_{PP}$ *SUZUKI-HIROSHI-KYOUJU* ('Prof. Hiroshi Suzuki')-*niyoruto* ('according to')] …(omitted) …
[$_{PP}$ *KETSUEKI* ('blood')-*no* (nominal modifier marker)]
[$_{PP}$ *TEIKYO* ('donation')-*o* (Object marker)]
[$_{VP}$ *UK* ('receive')-*e* (verbal conjunctive form)]
[$_{PP}$ *IDENSHI* ('genes')-*o* (Object marker)]
[$_{VP}$ *KAISEKI* ('analysis')-*suru* ('do')].

The words in uppercase are written in *Kanji* or *Katakana*, while those in lowercase are in *Hiragana*.

As long as a content word is transcribed all in *Kanji* and/or *Katakana*, it is relatively straightforward to identify phrases, but unfortunately a content word can be transcribed by a mixture of *Kanji* and *Hiragana* characters, followed by *Hiragana*-written functional elements as in:

[$_{PP}$ [$_{NP}$ content word both in *Kanji and Hiragana*] [$_{P}$ functional element in *Hiragana*]],

or a content word can be transcribed all in Hiragana as in:

[$_{VP}$ [$_{V}$ content verb stem *in Hiragana*] [$_{INFL}$ verbal inflection in *Hiragana*]] [$_{CONJ}$ clause-final suffix in *Hiragana*].

Both undesirable cases are exemplified by the last phrase in Table 2 below.

TABLE 2 An unsuccessful output of the phrase analyzer

電力業界では、九州、四国が
denryoku-gyoukai-dewa, Kyuushuu, Shikoku-ga
electricity-industry-in-TOPIC, Kyushu-Shikoku-SUBJECT
０６年度採用を横ばいにとどめるが、...。
06-nendo-saiyou-o-yokobai-ni-todom-eru-ga, ...
06-fiscal-year-employment-OBJECT-the same level-in-keep-but,

'In the electricity industry, Kyushu and Shikoku keep the employment in the 06 fiscal year in the same level, …'

CONTENT WORD	FUNCTION ELEMENT	GRAMMATICAL ROLE
電力業界	*では*	話題 (TOPIC)
中段 (Break)	、	読点　(COMMA)
九州	(省略)	次の内容要素と同じ (SAME AS NEXT CONTENT ELEMENT)
中段 (Break)	、	読点　(COMMA)
四国	*が*	主語 (SUBJECT)
０６年度採用	*を*	目的語 (OBJECT)
横	*ばいにとどめるが*	逆接接続語節 (CLAUSE-BUT)

The last row of Table 2 contains a content word in a mixture of *Kanji* and *Hiragana*, and the analyzer fails to recognize the end of the content word, leaving out part of the content word and placing it in the box for the functional element as: [[$_{NP}$ *YOKO*] [*bainitodomeruga*]]. The proper analysis would be:

[$_{PP}$ [$_{NP}$ *YOKObai* ('same level')] [$_{P}$ *ni* (postposition indicating state)]]
[$_{VP}$ [*todom* ('keep')]+[*eru* (non-past verbal inflection)]]
[$_{CONJ}$ [(preceding clause] [*ga* (suffix meaning 'but')]].

The failure is due to the content noun words that often consist of a mixture of *Kanji* and *Hiragana* as well as due to the fact that the content verb stem *todom* 'remain' was transcribed not in the regularly expected *Kanji* but exceptionally in *Hiragana*.

3 Algorithm of the Morpho-Syntactic Analyzer

As the above two examples illustrate, the success of the present analyzer in detecting phrases depends upon whether phrases are (a) typical ones consisting of a content word in *Kanji* (and/or *Katakana*) followed by a functional element in *Hiragana*, or whether they are complex ones, for instance, (b) consisting of a complex functional element in *Hiragana* or whether they are atypical ones (c) containing a content word transcribed in *Hiragana*. The current analyzer attempts to handle (a) and (c).

The algorithm of the analyzer, illustrated in Figure 1, begins to look for a new phrase by checking special characters and suffixes including a period, a comma, a parenthesis, and a complementizer. It then checks for an atypical case of a phrase, i.e., whether the phrase begins with a *Hiragana* or a *Hiragana* sequence (the loop marked (1) in Figure 1). When it finds only one *Hiragana* followed by a non-*Hiragana* sequence, it asks whether the *Hiragana* is equal to an Honorific prefix or not. If it is, it flags the phrase as prefixed with an honorific, and goes on to process the non-*Hiragana* sequence that follows. On the other hand, when it finds more than one *Hiragana* that precedes a non-*Hiragana*, the *Hiragana* chunk is treated as a phrase and sent to the procedure to identify the grammatical role, primarily by analyzing the final portion that is expected to comprise a functional morpheme or morphemes.

When a phrase begins with a non-*Hiragana* character, the analyzer keeps reading it (the loop marked (2) in Figure 1) until it hits a comma, a bracket, a period or a *Hiragana*, and assigns the non-*Hiragana* sequence as the content part of the phrase. The algorithm then checks whether the non-Hiragana content part ends with a period. If it does, the phrase is determined to be the final nominal phrase of the sentence with the functional element omitted.

On the other hand, when the non-*Hiragana* content part is followed by a *Hiragana*, it is likely to embody a typical phrase structure, and the following *Hiragana* sequence is sent to the procedures so as to find out first (i) how much of the *Hiragana* sequence represents a functional element or elements, and then (ii) what is the functional role or the final functional role if there is more than one element.

When the non-*Hiragana* content part is not followed by *Hiragana*, the algorithm checks for two possible instances. First, when it finds the content part to be an expression of a date, time or a clause ending with a suffix denoting time, it marks the phrase as the one whose functional element is omitted. Second, when it finds the character in question to be a comma, it indicates that the phrase is without the functional part, and that the functional role is the same as that of the following phrase, because the comma is treated the same as a conjunction.

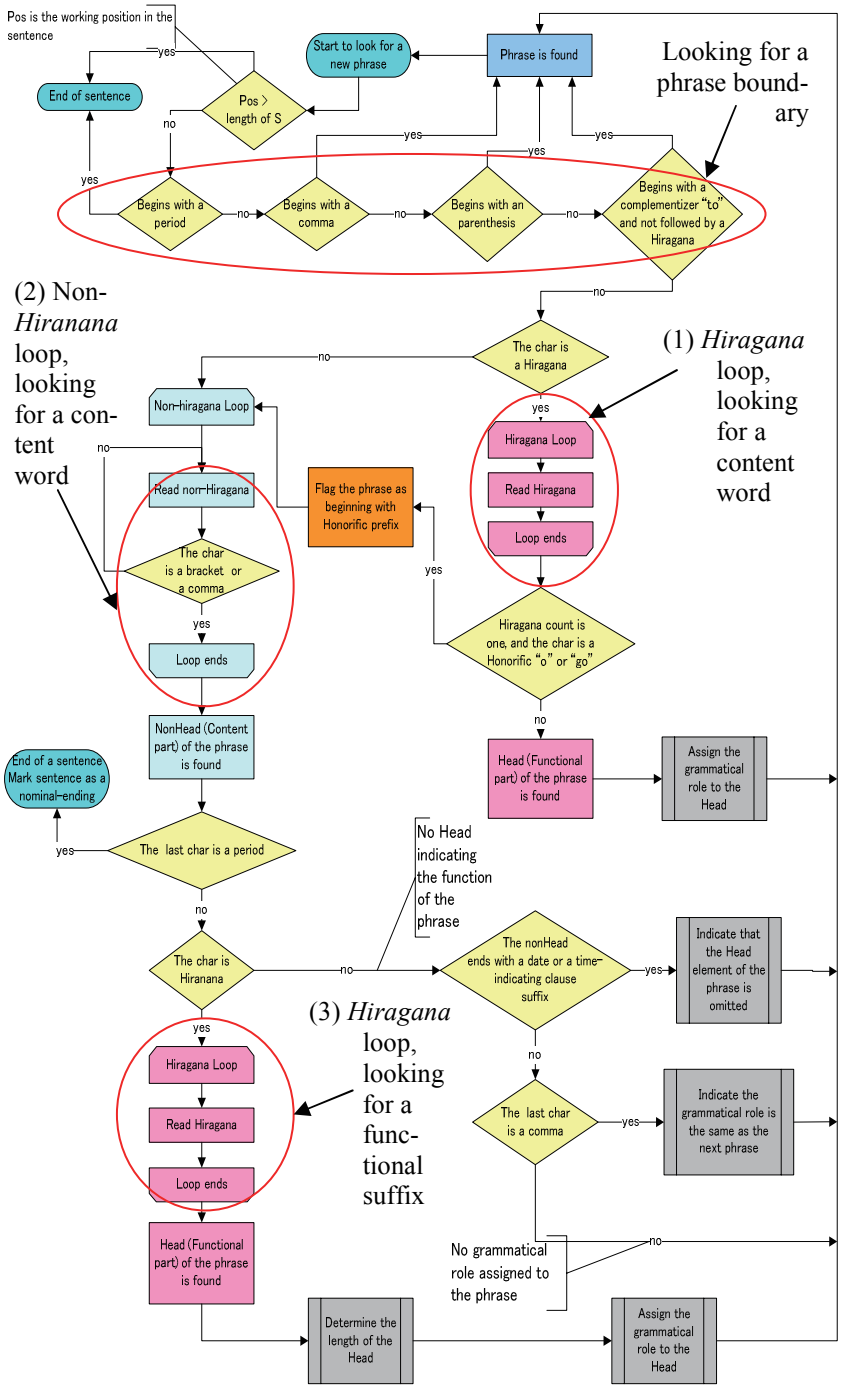


FIGURE 1 Algorithm of the morpho-syntactic analyzer

4 Architecture of the Present System

The current system is constructed on an object-oriented design, comprising four Java programming language classes (programs): *MorphAlgorithm*, *Phrase*, *CharIdentifier* and *Grammar*. The *MorphAlgorithm* is the main program that runs on the algorithm introduced in the previous section and charted in Figure 1. The *Phrase* simulates a phrase (a syntactic unit), thus housing access methods to the Head and Complement.[2] The *CharIdentifier* provides the *MorphAlgorithm* with several methods that identify characters. The *Grammar* is instanced by the *MorphAlgorithm* to find out the functional role of the Head of a phrase. The grammatical roles identified are listed in the following tables.

TABLE 3 Case markers/Particles denoting thematic relations

Case/ Particles	Pronunciation	Functional role(s)
が	ga	Subject marker
を	o	Object marker
は	wa	Topic marker
で	de	Place/Instrument/conjunctive
へ	e	Goal
から	kara	Point of departure
まで	made	'up to/till'
より	yori	Point of departure (formal or archaic)
として	toshite	'as' (Representative)
による	niyoru	Means
について	nitsuite	'concerning'
によると	niyoruto	'according to'

TABLE 4 Particles denoting conjunction

Particles	Pronunciation	Functional role(s)
も	mo	'too'/conjunction for nouns
や	ya	conjunction for nouns (inclusive)
と	to	conjunction for nouns (exclusive)
か	ka	'or'/Question particle
および	oyobi	conjunction for nouns (formal)

[2] It is based on the linguistic assumption that a phrase consists of a *Head* component and a *Complement* component.

TABLE 5 Particles that form modifiers of clauses

Particles	Pronunciation	Functional role(s)
ので	node	'because'
ため	tame	'because'
ために	tameni	'because'
けど	kedo	'although' (informal)
けれど	keredo	'although'
のに	noni	'even though'
ても	temo	'even though'
とき	toki	'when'
れば	reba	'if'
あいだ	aida	'while'

TABLE 6 Particles that form modifiers of verb phrases

Particles	Pronunciation	Functional role(s)
ものの	monono	'even though' (formal)
ながら	nagara	'while'
したまま	shitamama	'while doing'

TABLE 7 Particles denoting approximation or comparison

Particles	Pronunciation	Functional role(s)
ほど	hodo	'or so' (a little formal)
くらい	kurai	'or so'

In addition, the *Grammar* is able to identify the past and non-past affirmative and negative inflections of verbs and adjectives, and the conjunctive forms.

5 Discussion

Accuracy rates could be very high (a) when a text is written primarily in controlled Japanese (i.e., when the text is transcribed wherever applicable by using *Joyo Kanji*), (b) when the content words are followed by single functional elements, (c) when the content words are transcribed exclusively in *Kanji* and/or *Katakana*, and (d) when the text does not contain a long word in Hiragana such as a long adverb or conjunction. Table 1 shows such a sentence, and the accuracy rate is 100%. Accuracy rates become lower when the above conditions are not satisfied.

When the content words of a text are followed by a long sequence in *Hiragana* (counter to (b) above), the sequence is likely to comprise:

(i) more than one compound verbal suffix, or
(ii) a sequence of compound particles such as a Case marker followed by other particles.

It would not be very difficult to parse compound verbal suffixes consisting of long *Hiragana* sequence because of the following two facts: verbal and adjectival inflections in Japanese exhibit systematic paradigms, and such suffixes as causative, passive, aspectual and modal auxiliaries are aligned in rigid and thus predictable orders. To deal with a long predicate comprising more than one verbal suffix, a morphological analyzer is being prepared. Because this kind of a long predicate verb or adjective phrase occurs at least once in a sentence (that requires a predicate), and twice or more when the sentence contains a subordinate clause or clauses, significant improvement is expected, once the morphological analyzer for treating the complex verb phrase is incorporated into the current system.

Compound particles (for instance, consisting of a Case marker followed by a focus particle) also have a fairly rigid order, and it would be possible to analyze them in the system, once the orders are identified and implemented. However, it would be necessary to conduct a comprehensive linguistic study in this area for the successful identification of each functional element in sequence. At present a sequence of particles is treated as one chunk, the functional role of which is identified by the final particle.

When the content words in a sentence are transcribed in a mixture of *Kanji* (or *Katakana*) and *Hiragana* (counter to (c) above), the current system is unable to deal with such content words, because it does not have a dictionary. It would be interesting to investigate how frequently such words are transcribed in a mixture of *Kanji* (and/or *Katakana*) and *Hiragana*. Most adverbs and conjunctions are transcribed in *Hiragana*, even though there are some such as *OMOigakezu* ('by chance') and *sorenimoKAKAwarazu* ('in spite of that') that are transcribed in a mixture of *Kanji* and *Hiragana*. As a result, it is not so problematic to parse words in the two categories. Problems are caused mainly by nouns and compound verbs. However, nouns derived from verbs and adjectives are written in a mixture of the two characters: for instance, the noun *KAri* ('loan') derived from the verb *KAriru* ('borrow') and the noun *TANOshisa* ('pleasure') derived from the adjective *TANOshii* ('pleasant'). Such derivations are predictable, so it would be possible to prepare a morphological analyzer to handle them. Further research on derivations would be needed to improve the current system.

Quasi-compound verbs such as KAkeKOmu ('run into'), TAmeKOmu ('save up'), HIkiNObasu ('stretch out') and HIkiHANAsu ('separate') are problematic. They take the form of compound verbs, but they do not seem to be semantically compound verbs, because the original meanings of the following suffix verb or the preceding prefix verb are no longer independent but incorporated into the meanings of the main stem verbs. Therefore it is appropriate to handle such compound verbs as single verbs. As the current system

aims at analyzing sentences into phrases, it is undesirable to treat them as separate verbs. This problem cannot be solved without a dictionary that lists quasi-compound verbs or a morphological engine that deals with such verbs.

The current system is not equipped with a dictionary, and does not contain an exhaustive list of adverbs and conjunctions. At present it identifies twenty-two adverbs and thirteen conjunctions. Since the numbers of adverbs and conjunctions are relatively definitive and not large, a future task would be to see how much improvement can be achieved, once an exhaustive list of words in these categories is incorporated into the system.

Finally, the system is unable to handle elements in parentheses, which are often semantically related to the preceding elements in various manners. Parenthetical elements could be explanations of the preceding abbreviations or vice versa. There are no formal clues to the understanding of the relations between the two elements. This area remains to be explored.

6 Conclusion

The current morpho-syntactic analyzer, without a dictionary, aims at parsing into phrases texts written in *Joyo Kanji* (frequently used Chinese characters). The phrases are divided into content and functional sections and functional roles are assigned. The results suggest that this light-weight phrase analyzer could be a useful tool for natural language processing, while awaiting further study and additional modules of implementation for better results. In machine translation, once the functional roles of phrases are identified, it will not be necessary to further break up phrases into morphemes, thus saving time and avoiding unnecessary parsing. Text understanding would be improved when the phrases of sentences are understood.

References

Asahara, Masayuki. 2003. *Corpus-based Japanese morphological analysis*. Ph.D. Thesis: Nara Institute of Science and Technology.

Fuchi, Takeshi and Shinichiro Takagi. 1998. Japanese Morphological Analyzer using Word C0-occurrence. In *Proceedings of the COLING*, pp. 409-413.

Haruno, Masahiko, Satoshi Shirai, and Yoshifumi Ooyama. 1999. Using Decision Trees to Construct a Practical Parser. *Machine Learning*, 34:131-149.

Kameda, Masayuki. 1996. A Portable & Quick Japanese Parser: QJP. In *Proceedings of the COLING*, pp. 616-621.

Kanayama, Hiroshi, Kentaro Torisawa, Yutaka Mitsuishi and Jun'ichi Tsujii. 2000. A Hybrid Japanese Parser with Hand-crafted Grammar and Statistics. In *Proceedings of the COLING*, pp. 411-417.

Kashioka, Hideki, Yasuhiro Kawata and Yumiko Kinjo. 1998. Use of Mutual Information Based Character Clusters in Dictionary-less Morphological Analysis of Japanese. In *Proceedings of the COLING*, pp. 658-662.

Kazama, Jun'ichi. 2001. *Adaptive Morphological Analysis with a Small Tagged Corpus*. Master Thesis: University of Tokyo.

Kurohashi, Sadao and Makoto Nagao. 2003. Building a Japanese Parsed corpus — while improving the parsing system. In Anne Abeille (ed.), *Treebank Building Using Parsed Corpora*, pp. 249-260. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Matsumoto, Yuji, Akira Kitauchi, TatsuoYamashita, Yoshitaka Hirano, Hiroshi Matsuda, Kazuma Takaoka and Masayuki Asahara. 2001. *Morphological Analysis System ChaSen version 2.2.4 Manual*. Nara, Japan: Nara Institute of Science and Technology.

Sekine, Satoshi. 2001. A Fast Japanese Sentence Analyzer. In *Proceedings of the First International Workshop on MultiMedia Annotation*.

Suzuki, Hisami, Chris Brockett, and Gary Kacmarcik. 2000. Using a Broad-Coverage Parser for Word-Breaking in Japanese. In *Proceedings of the COLING*, pp. 822-828.

Uchimoto, Kiyotaka, Masaki Murata, Satoshi Sekine and Hitoshi Isahara. 2000. Dependency model using posterior context. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pp. 321-322.

Framework Independent Summarized Parser Output in XML and its Example-based Documentation

Tam Wai Lok
University of Tokyo

Miyao Yusuke
University of Tokyo

Tsujii Jun'ichi
University of Tokyo
University of Manchester
National Centre for Text Mining

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

We see a communication problem between the grammar engineering community and the NLP community. The information to be communicated is the results produced by a grammar. This paper is about our solution to the problem. Our solution has two components: an alternative output format and its documentation. Our alternative output format carries constituency information that parsers are built for computing, but in lesser quantity and a simpler form than the standard attribute-value matrix (AVM) output format. The documentation for it provides a shallow and static explanation different from the deep and dynamic explanation found in literature about grammar formalisms meant for grammar writers. The shallow and static explanation is meant to enable members of the NLP community to achieve a shallow level of understanding of the results produced by a grammar for the sake of developing NLP systems that interoperate with parsers.

1 Introduction

Grammar engineering presents an especially difficult tension between grammar writers who are predominantly interested in carrying research in the field forward and developers who build NLP systems that interoperate with deep parsers but are not very interested in the grammars behind them. The source of this tension is that the results produced by grammars are not designed and documented as a language resource for the wider NLP community. Members of the grammar engineering community can proceed with their research without documentation that explains the meaning of the results produced by a grammar. The common knowledge acquired from the literature about the formalism on which a grammar is based and shared among them in the computation of the results render such documentation unnecessary for the grammar writers. However, given that grammars are built for practical use in the development of larger NLP systems, the paucity of documentation for users who should not need to acquire the common knowledge shared among members of the grammar engineering community and the design of the output format of deep parsers which require such knowledge for deciphering the results are practical problems, if not theoretical ones. In this paper, we present a solution to these practical problems. It is our hope that our work can draw the attention of the grammar engineering community to the need of the wider NLP community for a simpler design of and documentation for the results produced by a grammar.

We are not being critical of the non-existence of documentation for grammar writers. There may not be a practical problem in that area as long as members of the grammar engineering community can carry on with their work by relying on common knowledge shared among them and on the literature about the grammar formalisms on which their work is based. Such documentation, while good to have for the sake of new members of the grammar engineering community, is not a solution to the practical problem in communicating the results produced by the grammar engineering community to the wider NLP community. Developers of NLP systems outside the grammar engineering community are not equipped with the background knowledge needed for understanding such documentation. The solution we present here is meant for these developers who share knowledge about linguistic concepts like POS, semantic representations and subcategorization with grammar writers but lack the knowledge in a specific framework required for finding information about these concepts from framework specific representations.

Our solution is built on top of ENJU (Miyao et al. [2004]). ENJU is built with a view to being a practical parser that accepts real text and forms a part of larger NLP systems. It includes a mostly induced, partly handcrafted grammar which keeps grammar engineering work at a minimum. This is in part why we are less concerned with meeting the needs of grammar writers but more concerned with providing support for

use in NLP system development. This support is provided by means of an alternative output format which carries information summarized from that carried in the standard AVM output format and documentation for the alternative output format.

To illustrate what is kept and what is left out in our summary of a complete feature structure representation, we give the representation of the relative pronoun ”who” in the AVM format and the representation of it in our alternative output format one after the other.

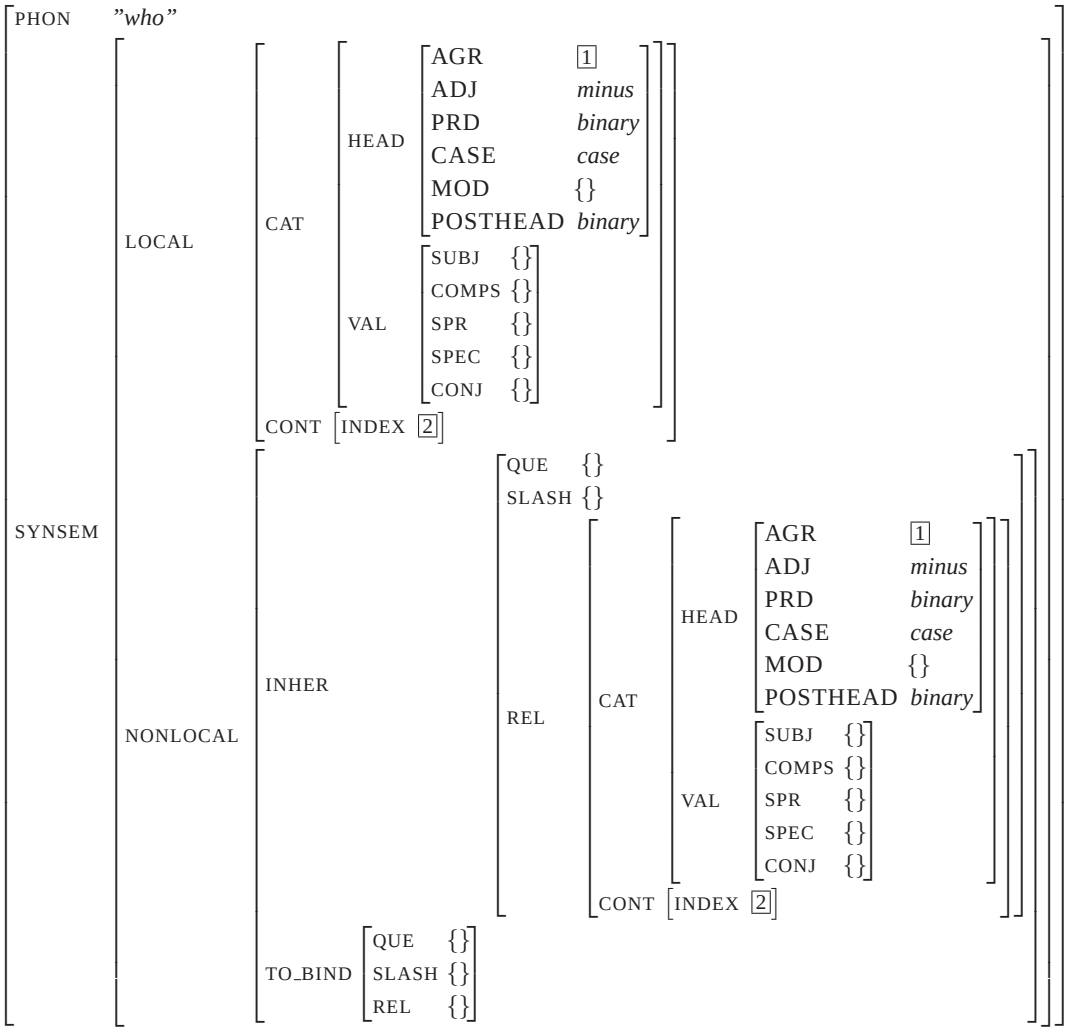


Figure 1: the complete AVM representation of ”who”

```
1 <tok id="t4" cat="N" pos="WP" base="who" lexentry="N.3sg/[&lt;NP.3sg&gt;]" pred
    ="relative_arg1" arg1="c8">
  who
3 </tok>
```

As an alternative to the standard AVM output format, our format is different from MRS (Copestake et al. [2005]), another alternative output format but similar to the bracketing style used in the Penn Treebank. While

both MRS and the Penn Treebank bracketing style are grounded in linguistic theories, the Penn Treebank bracketing style is meant to be understood (at a shallow level) without in-depth knowledge of the theories whereas MRS is meant to be understood with deep knowledge of them. This difference is obvious in the literature about the two formats. The Penn Treebank annotation manual (Bies [1995]) provides a large number of examples without explaining how they are computed with the transformationalist theories. MRS comes with a paper that describes in details the steps for computing a MRS representation and the theories on which the computation is grounded (Copestake et al. [2005]). For the Penn Treebank, the point of enabling users to achieve a shallow level of understanding of the analysis by annotators is to help developers to debug NLP systems that use the Penn Treebank as a language resource by giving them an idea how the analysis of a linguistic phenomenon looks, not to enable them to do the computation done by annotators. Likewise, we want to enable our users to achieve a shallow level of understanding of the results produced by our grammar for debugging NLP systems that use these results as a language resource: we do this by giving them an idea how the analysis of a linguistic phenomenon looks. Our goal is not to enable them to do the computation done by parsers and grammar writers.

This paper is organized as follows: We start with giving more details on the communication problem we mention above. Then we describe our solution by highlighting some of its characteristics and providing some examples. Finally, we conclude this paper with a summary and some thoughts on the direction our work is heading.

2 Problem definition

In the beginning of this paper, we describe the problem we are addressing as one of communication between the grammar engineering community and the wider NLP community. This kind of communication problems between research communities is not uncommon in the academic world. But the problem involving the grammar engineering community and the NLP community is particularly serious for two reasons.

The first reason is that substitute for more canonical documentation that serves members of the grammar engineering community well does not function well for members of the wider NLP community. By substitute of documentation, we are referring to textbooks that introduce students to a formalism like (Sag et al. [2003]) or handbooks that cover everything essential about a formalism like (Pollard and Sag [1994]). Literature does not function well for members of the wider NLP communities because the parser results are different from those in papers. The former comes with much more information than the latter. This is because linguists who propose these formalisms omit feature-value pairs they consider irrelevant to the linguistic phenomena they are interested in when they present the computation in papers.

Let us illustrate the difference between the result produced by a parser and the analysis given on paper with an example. A sign of any POS carries the *MOD* feature in HPSG. A noun or a verb that does not function as an adjunct is assigned an empty value for this feature. When talking about control and raising, linguists know that there is not much point in specifying the *MOD* value of the control (raising) verb and its NP arguments on paper. However, parsers do not know this. They can only display all feature-value pairs or rely on users to choose which features to display. The knowledge required for filling in the gaps between the output of parsers and the output given on paper, like the common knowledge that members of the grammar engineering community rely on for carrying research in their field forward, is missing for members of the wider NLP community. This renders the literature about the grammar formalism on which a grammar is based less useful for members of the wider NLP community.

The second reason is that there is an explosion in the information being communicated between (the systems built by) the two communities. The grammar engineering community places little restriction on the introduction of new features for covering new phenomena in a grammar. Often the new features are included in the feature structure representations of all signs. So the introduction of new features for covering a new phenomenon does not only put more information in the feature structure representations of sentences related to the phenomenon for which the features are introduced. It also puts more information in the feature structure representations of sentences not related to the phenomenon. The result of this is an explosion of information. With wide-coverage being the pursuit of the grammar engineering community, we are witnessing such explosions in every well-known deep parser. For example, the features structure representation of example sentence 1 has more than 500 feature-value pairs in the output produced by ENJU, the deep parser we use.

(1) John is the man who Mary loves

Common current attempts at providing a solution to the communication problem we identify here are not satisfactory in two aspects:

- Reducing the information to be communicated to (the systems built by) the NLP community is recognized as a means of providing a solution to the problem. However, the information left to be communicated to the NLP community is very often packed in a new format which demand them to acquire new knowledge for the purpose of making sense of and using the packed information. One such new format is MRS. It may be true that the design of a new format is inevitable for the purpose of packing the information to be communicated. However, the reduced information is often in an unfamiliar format. If such a format is significantly different from what developers are familiar with it would create the same hurdle created by the original grammar frameworks.

- MRS and dependencies are two formats sometimes cited as a solution to the problem. Both formats carry no constituency information, which is the information parsers are supposed to compute according to the widely accepted definition of a parser as a program that identifies the phrase structure of an input sentence. As a result, many other research communities and systems built by them expect this information from parsers and the research community working on parsers. An example of NLP systems that needs constituency information from parsers is a speech synthesiser. It needs the phrase structure of an input sentence to determine the prosodic structure of it. Providing constituency information with other information would help to solve the communication problem between the grammar engineering community and the NLP community.

3 Solution

3.1 Our alternative output format: summarized parser output format

Our alternative output format has the following characteristics:

Fixed number of attributes In feature structure based grammar formalisms, the number of attributes of every sign increases proportionally with the coverage of a grammar. In our simplified output format, we define a fixed set of attributes for terminal nodes and a fixed set of attributes for non-terminal nodes.

Framework independent attribute names In feature structure based grammar formalisms, features may be embedded as the value of some other features. Path information, that is, the names of all the embedding features of a feature, is needed for identifying the embedded feature. Different feature structure based formalisms have different paths and names for features that carry similar information. In our simplified output format, attributes take atomic values and are given framework independent names based on the type of information they carry.

Hidden value-sharing In feature structure based grammar formalisms, unification of values occurs between features found in multiple locations, essentially repeating the same information. In our simplified output format, inheritance of attribute values from a daughter node to its mother is not shown. Only sharing of values between sisters and constituents in a long distance dependency relation is visible. The visibility of value-sharing of the later kind is enough for capturing a wide range of linguistic phenomena.

It is not difficult to see that these three characteristics deal with the following sources of complaints about the complete AVM output:

1. There are too many feature-value pairs in a feature structure representation of a constituent.
2. It is difficult to tell what kind of information is contained in an embedded feature with a long path name.
3. The sharing of values between features in a large feature structure is difficult to trace and make sense of.
4. The same piece of information appears in multiple locations.

These complaints are not only about the quantity of information represented in the complete output. Some of these complaints are about the way information is carried. HPSG allows phrase structure trees whose non-root nodes carry information produced by the parsing of large constituents. For example, the SYNSEM|LOCAL|CONT|LOVER feature of the root node ”loves” in a HPSG-style phrase structure tree of example sentence 1 is assigned the
SYNSEM|LOCAL|CONT|INDEX value of the root node ”Mary”. This information is produced by the parsing of the nonterminal embedded sentence node ”Mary loves”.

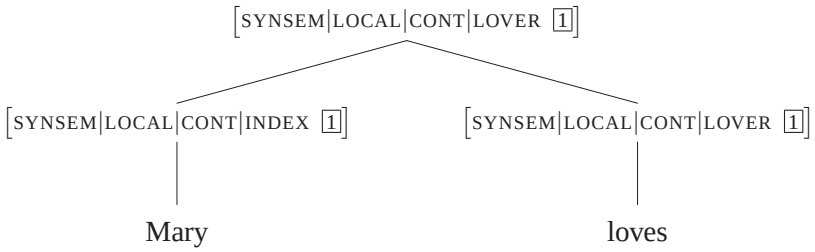


Figure 2: Mary loves

In lambda calculus based semantics found in other frameworks like LFG (Dalrymple [2001]) and CCG (Steedman [2000]), the agent role of the semantic representation of the root node ”loves” would not be filled by the reference marker of the root node ”Mary”. Instead, it would remain uninstantiated and the variable corresponding to the argument slot would be marked by a λ:

$$\lambda X.\lambda Y.love(X, Y)$$

Grammar writers who are familiar with the HPSG formalism have little problem in understanding why the agent role of the semantic representation of the root node "loves" is filled by the reference marker of the root node "Mary" in a HPSG-style phrase structure tree. However, developers in the NLP community may find it confusing. The problem is that the complete AVM output includes steps in the computation but these steps look different from the way they look during the computation. To address this problem, we remove these steps from our alternative output format by hiding the sharing of values.

Our alternative output format represents an attempt to simplify feature structure based grammar formalisms without sacrificing the power of deep processing in capturing linguistic phenomena like long distance dependencies and raising which proves difficult for shallow processing. Our approach is different from the approach of output formats like MRS and dependencies. We try to summarize the complete AVM output. They extract some specific information (e.g. semantics in the case of MRS) from the complete AVM output. For this reason, we name our output format **Summarized Parser Output** (SPO). In SPO, it is possible to distinguish between the output produced by parsing example sentence 1 and the output produced by parsing:

(2) John is a man and Mary loves John

Capturing this difference between different constructions is what we mean by capturing linguistic phenomena. This is important for the output of a parser. In other alternative output formats, it would be impossible to distinguish output produced by sentences with the same meaning or the same dependencies between constituents.

SPO is meant to be a format for making it easy to use the parser results in the development of NLP systems. During the development of an NLP system that interoperates with a parser, developers are not involved in the computation done by the parser but they are often required to check the results produced by the parser for debugging purpose. Textbooks and handbooks which explain how the computation is done do not meet their need. They need a large collection of examples in the style of the Penn Treebank manual against which they can check the results produced by the parser without doing the computation. Therefore, our documentation for SPO is modelled on the Penn Treebank manual.

3.1.1 Specifications of SPO

Nodes of a phrase structure tree in feature structure based formalisms are structured complexes of features and values. These nodes are represented by XML elements in SPO. The structure of a parse tree is determined by mother-daughter relations and sister relations between its nodes. The two relations are captured in the following way:

mother-daughter relations Two nodes in a mother-daughter relation are represented by an enclosure relation between two *cons* elements. The node represented by the enclosed element is the daughter. The node represented by the enclosing element is the mother.

sister relations Two nodes in a sister relation are represented by two non-mutually-enclosing *cons* elements which are both enclosed by the same *cons* element.

A *cons* element can represent the root node, a terminal node and a nonterminal node. The outermost *cons* element represents the root node. A leaf node of a parse tree is represented by a *tok* element.

To enable our readers to visualize what we have just described, we give the following empty template with all attributes except *id* removed .

```
<cons id="c1">
 <!--this is the root node -->
 <!--this is the mother of the constituents represented by c2 and c3 -->
        <cons id="c2">
          <!--this is the daughter of the constituent represented by c1 -->
          <!--this is the sister of the constituent represented by c3 -->
          <tok id="w1">
          <!--this is a leaf node -->
          </tok>
        </cons>
        <cons id="c3">
          <!--this is the daughter of the constituent represented by c1 -->
          <!--this is the sister of the constituent represented by c2 -->
         <tok id="w2">
         <!-- this is a leaf node-->
         </tok>
        </cons>
</cons>
```

As for the attributes carried by the *cons* and *tok* elements:

Attributes carried by both *cons* and *tok* elements POS information (*cat*), reference marker (*id*)

Attributes carried only by *cons* elements syntactic head (*head*), semantic head (*sem_head*), the rule responsible for rewriting an element as its daughter (s) (*schema*)

Attribute carried only by *tok* elements base form (*base*), references to lexical rules or lexical entries (*lex-entry*), tense (*tense*), aspect (*aspect*), verb type (*aux*), the argument variables to which the semantic representations of the corresponding nodes apply (*argn*), semantic representation (*pred*)

Attributes like *tense*, *aspect* and *voice*, which correspond to features whose values are passed up from the lexical entry of a verb to the terminal verb node and from a terminal verb node to a non-terminal verb phrase node in a phrase structure tree of feature structures, are not included as attributes of the *cons* elements which we use for representing terminal nodes and non-terminal nodes. This is what we mean by hiding values shared between a mother and a daughter.

3.1.2 Summarizing features

The attributes of *cons* and *tok* elements are summarized from features of the corresponding nodes. Some features of a node are captured by straightforward one-to-one conversion. Others are captured by generalizing over a few features of the node and producing one attribute in the corresponding XML representation for several features of the node. The rest are simply not represented in the XML form.

The case of one-to-one conversion and the case of neglecting a feature are trivial but the idea of generalizing over several features of a node requires some explanation. To illustrate, let us take the *cat* attribute of a *cons* element or a *tok* element as an example. Its value is determined by both the value of the SYNSEM|LOCAL|CAT|HEAD feature and the value of the SYNSEM|LOCAL|CAT|SUBCAT feature of the corresponding node. We say the *cat* attribute is a generalization over the HEAD feature and the SUBCAT feature. By neglecting some features and generalizing over others, we greatly reduce the number of attributes in SPO while keeping the number of elements the same as the number of nodes in the parse tree being represented by it.

3.1.3 An example

The specifications and the methods of summarization produce less expressive power in exchange for reducing complexity. But they can be used creatively for capturing a wide range of linguistic phenomena in a deep but simple way. Let us illustrate how this can be done with our analysis of the relative clause contained in example sentence (1).

```
<cons id="c36" cat="NX" xcat="" head="c37" sem_head="c37" schema="
    head_relative">
 <cons id="c37" cat="NX" xcat="" head="t16" sem_head="t16">
   <tok id="t16" cat="N" pos="NN" base="man" lexentry="[D&lt;N.3sg&gt;]_lxm"
       pred="noun_arg0">
     man</tok></cons>
 <cons id="c38" cat="S" xcat="REL" head="c40" sem_head="c40" schema="
     filler_head">
   <cons id="c39" cat="NP" xcat="REL" head="t17" sem_head="t17">
     <tok id="t17" cat="N" pos="WP" base="who" lexentry="N.3sg/[&lt;NP.3sg&gt
         ;]" pred="relative_arg1" arg1="c37">
       who</tok></cons>
   <cons id="c40" cat="S" xcat="TRACE" head="c43" sem_head="c43" schema="
       subj_head">
     <cons id="c41" cat="NP" xcat="" head="c42" sem_head="c42" schema="
         empty_spec_head">
       <cons id="c42" cat="NX" xcat="" head="t18" sem_head="t18">
         <tok id="t18" cat="N" pos="NNP" base="mary" lexentry="[D&lt;N.3sg&gt
             ;]_lxm" pred="noun_arg0">
           Mary</tok></cons></cons>
     <cons id="c43" cat="VP" xcat="TRACE" head="t19" sem_head="t19">
       <tok id="t19" cat="V" pos="VBZ" base="love" tense="present" aspect="
           none" voice="active" aux="minus" lexentry="[NP.nom&lt;V.bse&gt;NP.
           acc]_lxm-movement_rule-singular3rd_verb_rule" pred="verb_arg12" arg2
           ="c37" arg1="c41">
         loves</tok></cons></cons></cons></cons>
```

We make use of the idea of gaps in our analysis of relative clauses. In HPSG, this is done by introducing the SYNSEM|NONLOCAL|INHER|SLASH feature, the SYNSEM|NONLOCAL|INHER|REL feature and the SYNSEM|NONLOCAL|TO-BIND|SLASH feature. We try to do this without introducing any new attribute. A gap is formed when the relativized argument (object) of the embedded verb (”loves”) is removed from the subcategorization frame temporarily in the phrasal projection of the verb is formed without the argument being

sister to the verb. The phrasal projection of the verb formed as a result is gapped. A gapped verb phrase is simply marked by being assigned a *cat* value which says something different from the XML representation of the phrase structure of the sentence in question about the subcategorization frame of the verb. In the XML representation of example sentence (1), the lexical entry of the transitive verb ”love” (t19) is dominated by a verb phrase node (c43) whose subcategorization frame contains only a subject. This is indicated by its *cat* value VP. But we cannot find any other element that is enclosed by the element representing the verb phrase node. This is what we mean by having the *cat* value of a verb phrase saying something different from the phrase structure.

The semantic representation of the embedded verb ”loves” is given as the value of the *pred* attribute of the lexical entry of ”loves” (t19). Its theme role is represented by the *arg2* attribute of t19, which is assigned the *id* value c37 of the nonterminal head noun node. *id* can be understood as the entity a constituent refers to. Two different *id*s refer to the same entity if they come from two elements one of whose *id* value is assigned as the *sem_head* value of the other. So the *id* value c37 of the terminal noun node and *id* value t16 of the root node ”man” refers to the same entity.

3.2 Shallow documentation for parser output

In this section, we first provide a more detailed description of our example-based documentation and give an excerpt of it to illustrate the difference between theory-centred literature and example-based documentation. Then we offer more explanation as to why the latter is better suited for developers in the NLP community.

Our documentation is indexed by linguistic phenomena. It is organized into sections, each of which includes:

1. a section title that describes a linguistic phenomenon
2. an example sentence that illustrates the linguistic phenomenon
3. the translation of the result produced by parsing the example sentence with our parser to a format based on the Penn Treebank bracketing style
4. explanation for our analysis of the linguistic phenomenon

Here is an excerpted section broken into the mentioned elements:

Section title non-subject wh-relatives

Example sentence (3) John is the man who Mary loves

Simplified output

```
     (S  (NP  (NX John))
        (VP  (VX is)
            (NP  (DP the)
                (NX  (NX man[id=c37]))
                (S-REL  (NP-REL who[pred=relative_arg1,arg1=c37])
                       (S-TRACE  (NP  (NX Mary[id=c41]))
                                 (VP-TRACE
    loves[pred=verb_arg12,arg1=c41,arg2=c37))))))
```

Explanation

Syntax

- The relative pronoun ”who” is assigned the POS label (cat) NP .
- The embedded transitive verb ”loves” forms a gapped verb phrase, which is assigned the POS label VP, with no daughters.
- The gapped verb phrase is sister to the subject noun phrase ”Mary”. Together they form the gapped sentence ”Mary loves’, which is assigned the POS label S.
- The gapped sentence is sister to the relative pronoun. Together they form the relative clause ”who Mary loves”, which is assigned the POS label S.
- The relative clause is sister to the head noun ”man”.

Semantics

- The object position (arg2) of the embedded transitive verb ”loves” is relativized. It is assigned the reference marker (id) of the head noun ”man” (c37).

Note the similarity in style to the Penn Treebank annotation manual. Our explanation and the explanation offered in the Penn Treebank annotation manual are shallow and static. A shallow explanation does not give the readers the reason for a certain output. For example, we do not tell our readers the reason that a certain attribute is assigned a certain value is because a particular feature structure unifies with another feature structure and some values of the features carried by them are shared. The Penn Treebank annotation manual does not account for the existence of a trace in a specific position in terms of transformations. A static explanation does not include the steps taken to compute the result. Such steps are transformations in a transformationalist framework and unifications in a feature structure based framework. Our explanation does not mention unification . Likewise, transformations are hardly mentioned in the Penn Treebank annotation manual.

Also note the difference in style between our explanation and the explanation offered for the analysis of linguistic phenomenon in textbooks like (Sag et al. [2003]), handbooks like Pollard and Sag [1994] and literature like Copestake et al. [2005]. The explanation offered in these textbooks, handbooks and literature meant for members of the grammar engineering community and hence is deep and dynamic. A deep explanation gives the readers the reason for a certain output. A dynamic explanation goes through the steps taken to compute the result meant to be explained.

The importance of deep and dynamic explanation for grammars is obvious. (A deep and dynamic explanation for the results necessarily becomes a holistic explanation for the grammar.) In order to understand how a grammar works, grammar writers have to know which feature structure unifies with which and what values are shared between them. It is the unification and the sharing that enable a grammar to rule out ungrammatical sentences and construct the meaning of a sentence from its parts. The existence of such explanations, which are so useful to grammar engineering, obviates the task of creating documentation that provides the same kind of explanation.

However, it is easy to underestimate the importance of shallow and static explanation for results produced by a grammar. Such documentation is a major means of communication between the producers and the consumers of the parser, less often the means of communication among the producers. The need of the consumers is determined by the purpose for which they use the parser results: in our case, this purpose is the development of NLP systems that interoperate with parsers. What is needed is a shallow understanding of the results produced by a grammar.

What is a shallow level of understanding of the results produced by a grammar? It is some ideas about what the correct analysis of a linguistic phenomenon looks like. We provide examples in our shallow explanation to allow developers to check their results. Likewise, the shallow explanation offered in the Penn Treebank annotation manual comes with examples that allows developers to check their results they get from systems trained with the treebank against the examples directly. There is no question about the usefulness of shallow explanation to developers because it is simply designed to meet their needs during development.

4 Conclusion and future work

We have outlined and illustrated with examples our solution to the communication problems between the grammar engineering community and the wider NLP community. We attempt to solve these problems by simplifying the output of a deep parser in an alternative output format and providing documentation for that format.

Our alternative output format SPO is different from alternative output formats proposed for other deep parsers in our concern with preserving the syntactic information in the AVM format. We preserve this information so that the output of our parser in its simple form can be used for a wide range of NLP applications. (Chun et al. [2006], Miyao et al. [2006], Yakushiji et al. [2006])

The documentation for SPO provides shallow and static explanations to developers in the NLP community. This differs from the deep and dynamic explanation found in literature that serves grammar writers well as documentation. Though not very useful to grammar engineering, shallow and static explanation is needed by developers in the NLP community for the purpose of building NLP systems that interoperate with parsers. In showing that there is no substitute for documentation meant for developers, we argue that documentation targeted at the NLP community is an urgent task for those developing parser for NLP applications.

Our idea of a simplified but information-rich output format and documentation of it for members of the NLP community presented here are tested on a partly-handcrafted grammar should help development of applications fed on the output of more handcrafted grammars like LKB/ERG Copestake and Flickinger [2000]) as well.

We create the summarized output format described in this paper by summarizing the output of a deep parser which do HPSG-based parsing. Our simplified output format can be used for summarizing the output of other deep parsers which use other grammar formalisms. In fact, our simplified output format has some similarities to LFG. For example, subcategorization information is implicitly represented by the POS label and argument slots of the semantic representation in our simplified output and in LFG. The idea of leaving information that can be read off the phrase structure tree (in XML format) unrepresented in attribute-value pairs is also similar to the idea of separating constituency information from the functional-structure. Currently, we are in talks with groups working on parsing in LFG to explore using the same output format for summarizing output of deep parsers based on different formalisms. Our next step would be to extend the use of our summarized output to parsers built on feature based CCG. A common output format between deep parsers based on different formalisms would be very useful for parser evaluation, if accompanied by documentation created in the manner described in this paper for the output produced by each of the deep parsers.

Acknowledgments

This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and Grant-in-Aid for Young Scientists (MEXT, Japan).

References

Ann Bies. Bracketing guidelines for treebank II style Penn treebank project, 1995. URL citeseer.ist.psu.edu/bies95bracketing.html.

Hong-Woo Chun, Yoshimasa Tsuruoka, Jin-Dong Kim, Rie Shiba, Naoki Nagata, Teruyoshi Hishiki, and Jun'ichi Tsujii. Extraction of gene-disease relations from Medline using domain dictionaries and machine learning. In *Proceedings of the Pacific Symposium on Biocomputing 2006*, pages 4–15, Maui, 2006.

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

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. Minimal recursion semantics: An introduction. In *Journal of Research on Language and Computation*, volume 3, pages 281–332. Springer, 2005.

Mary Dalrymple. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press, 2001.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In *Proceedings of the First International Joint Conference on Natural Language Processing*, pages 684–693, Hong Kong, 2004.

Yusuke Miyao, Tomoko Ohta, Katsuya Masuda, Yoshimasa Tsuruoka, Kazuhiro Yoshida, Takashi Ninomiya Takashi, and Jun'ichi Tsujii. Semantic retrieval for the accurate identification of relational concepts in massive textbases. In *Proceedings of COLING-ACL 2006*, pages 1017–1024, Sydney, 2006.

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

Ivan A. Sag, Tom Wasow, and Emily M. Bender. *Syntactic Theory: A Formal Introduction*. CSLI Publications, second edition, 2003.

Mark Steedman. *The Syntactic Process*. MIT PRess, 2000.

Akane Yakushiji, Ysuke Miyao, Tomoko Ohta Tomoko, Yuka Tateisi, and Jun'ichi Tsujii. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 284–292, Sydney, 2006.