Proceedings

of

ESSLLI 2004 Workshop
on
Combining Shallow and Deep Processing for NLP

9th to 13th August 2004

A workshop held as part of the

Sixteenth European Summer School in Logic, Language and Information (ESSLLI-2004)

9th to 20th August 2004, Nancy, France

Workshop Organizers:

Erhard Hinrichs, SfS, Tuebingen University, Tuebingen, Germany
Kiril Simov, LML, Bulgarian Academy of Sciences, Sofia, Bulgaria

Nancy, France
2004

Workshop Organisers

Erhard Hinrichs
Seminar für Sprachwissenschaft
Universität Tübingen
Eh@sfs.uni-tuebingen.de

Kiril Simov
BulTreeBank Project
http://www.BulTreeBank.org
Linguistic Modelling Laboratory, CLPP,
Bulgarian Academy of Sciences
kivs@bultreebank.org

Workshop Programme Committee

Table of Contents

Author Index

Document Style Recognition Using Shallow Statistical Analysis

Pavel Braslavski

Institute of Engineering Science UB RAS
Komsomolskaya 34, 620219 Ekaterinburg, Russia
+ 7 (343) 375 3579
pb@imach.uran.ru

1. INTRODUCTION

Documents differ not only in topic but also in style. Style is a very broad and ambiguous term used in arts, fashion, literary criticism, and linguistics. In case of text documents we can accept an intuitive understanding that style is mainly related to the form (*how*) whereas topic – to the content (*what*) of a document. Although some topics determine strictly the style can be used, most topics allow their expression in various styles. Thus, style can be considered to be orthogonal to topic in a certain sense and represent therefore a useful parameter in many text processing and information retrieval tasks.

The main goal of our research is to develop automated procedures that enable text style recognition in favor of Web information retrieval. The research is related partly to the formal methods in authorship attribution, i.e. *individual style* recognition. There are also several studies aiming theoretical and educational goals that investigate quantitative variations of textual parameters within different text styles (e.g. different readability indices).

The paper by Jussi Karlgren and Douglas Cutting [3] gave the initial impulse to our research. The paper reports on stylistic experiments based on the Brown corpus of English text samples. Three-level genre hierarchy (from the 'imaginative/informative' dichotomy on the top down to 15 genres on the bottom) is used. A number of different features – surface cues along with e.g. part of speech (POS) and present participle counts – are used for classification. Discriminant function analysis is employed for data processing.

Several publications on the topic have appeared recently. Genre classification based on word statistics revealed from the interplay of subject-related and genre-related tagging of the training data is described in [5]. Incorporation of structural information of documents into a digital library navigation tool is introduced in [6]. The latter paper includes a detailed survey of different approaches in automatic genre and style analysis.

This paper describes two series of stylistic experiments conducted 1999-2002 within my work towards PhD and ongoing related research. In the next two sections, we introduce these experiments. Section 4 concludes the papers and outlines the future research suggested by the obtained results.

2. EXPERIMENT I

2.1. Experimental setting

In the first series of experiments we adopted the theory of functional styles, which is well-established and well-founded in Russian linguistics. The main idea of the functionalist approach is the distinction between the language (as a symbolic system) and the speech (as the very process of discourse generation). According to the theory, the style of a text is determined mainly by the com-

munication context. Five functional styles are usually defined, such as *official* style, *academic* style, *journalistic* style, *everyday communication* style, and *literary* style (although some scholars consider literary style, or fiction, to be a special case that is able to incorporate all other styles). More details on the theory of functional styles can be found in [4].

According to this approach, a training sample was composed carefully. The sample contained 305 documents in Russian (50 federal laws, 54 scientific papers in natural sciences, 61 online news articles, 79 short stories by modern Russian authors, and 61 fragments of chat listings).

We opted for linear *classification functions* for text categorization. This approach differs from the one described in [3], where *discriminant functions* are used. Each classification function is a linear combination of the classification variables (features) returning classification scores for each case for each group. The case is classified as belonging to the group for which it has the highest classification score. The assumptions of the technique are weaker than those of discriminant function analysis (see [11] for details).

Since most stylistic studies operate with qualitative descriptions, the selection of quantitative features becomes a challenging task. The initial feature set was composed based on both analysis of previous stylistic studies and examination of the training sample. We opted for easily computable features only; no format-specific parameters (e.g. HTML tags) were employed. The initial set consisted of approximately 30 features and was intentionally redundant. An overview of the initial features is shown in table 1.

The main unit of examination was a single word (i.e. no complete surface syntactic parsing was performed), although sub-word-level features (specific prefixes) and sentence-level features (e.g. sentence length, expressive punctuation marks, and genitive noun chains) were presented. Numerous morphological parameters (POS and distinctive grammatical forms) were determined using the dictionary-based stemmer LINGUIST by Agama Company. Lists of general academic terms and official document names were also composed.

Level	Parameter
Surface	equals sign per sentence rate smiles *:) ;-)* per sentence rate average word length (in characters) average sentence length (in words) expressive punctuation mark (*!*, *?*, ...) per sentence rate
Word formation	scientific prefix (*aqua-*, *aero-*, etc.) per word ratio
Morphology	POS rates (13 in total) neuter noun rate reflexive verb rate acronym rate first person pronoun rate second person pronoun rate particle *бы* rate (conjunctive mood cue) particle *ну, вот, ведь* rate (everyday communication style cue)
Lexis	word from *general academic term* list (37 words, statistically selected) rate word from *official document name* list (43 words, manually composed) rate

Syntax	genitive chain per sentence rate subordinating conjunction per sentence rate

Table 1: Features overview

For each text the first 1000 words (or the whole text if shorter) plus the words until the end of the sentence containing the 1000th word were processed. It is clear that some of the proposed features could not be computed absolutely accurately in full automatic mode. For instance, grammatical ambiguity was not resolved. Inaccuracy was caused also by end-of-sentence and end-of-word (due to invisible characters and hyphens) errors.

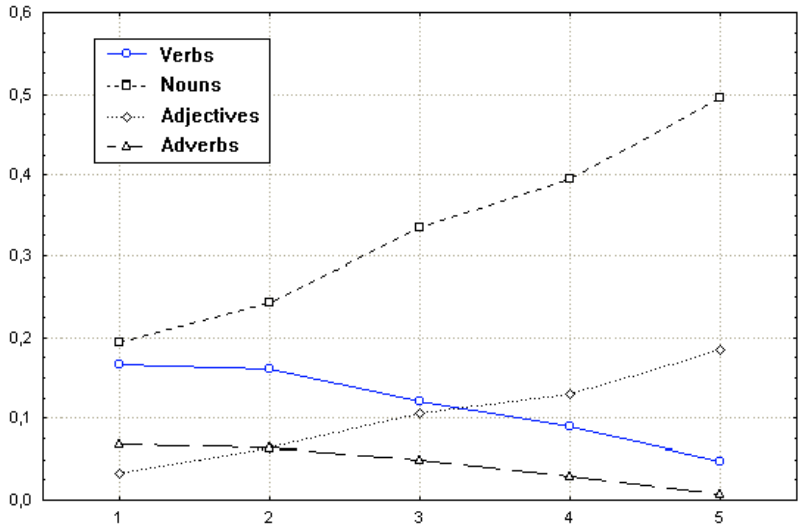


Figure 1: POS average rates across five styles
(1 – *everyday communication style*, 2 – *literary style*, 3 – *journalistic style*, 4 – *academic style*, 5 – *official style*)

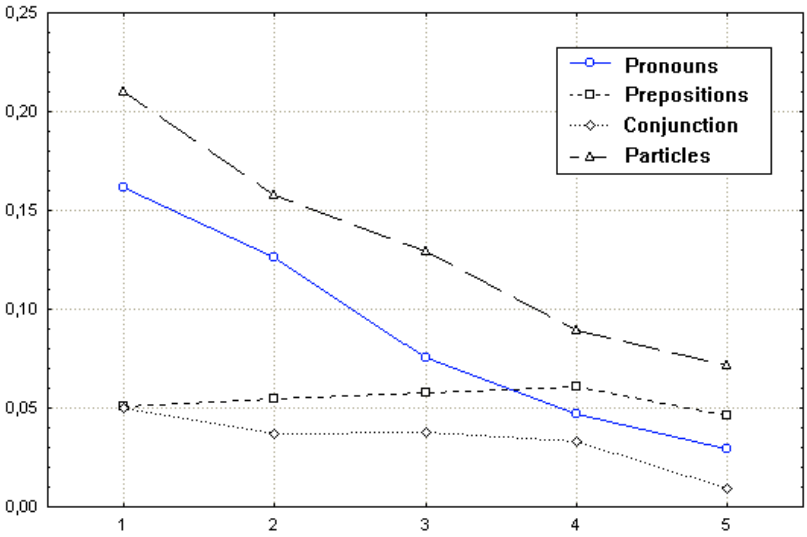


Figure 2: POS average rates across five styles
(1 – *everyday communication style*, 2 – *literary style*, 3 – *journalistic style*, 4 – *academic style*, 5 – *official style*)

2.2. Morphological characteristics

The morphological characteristics of the training sample appeared to be of interest to theoretical stylistics, since there are only few quantitative up-to-date comparative studies on of the functional styles. Moreover, our research, although not absolutely exact because of fully automatic processing, dealt with a reasonably large text collection. The amount of data processed was 239 696 words, the morphological characteristics were determined for 227 257 of them.

Figure 1 and figure 2 show monotonous decline of average verb, adverb, pronoun, and particle rates from the everyday communication style to the official style. Nouns and adjectives demonstrate an inverse behavior. The auxiliary POS rates are approximately the same across the styles.

Table 2 represents a portion of correlation matrix for POS rates across the whole training sample. The results show that knowing for example the noun count in a text we can predict the adverb count with a high degree of confidence.

Part of speech	1	2	3	4	5	6	7
1. Noun	1,00	0,85	-0,87	-0,85	-0,88	0,77	-0,86
2. Adjective	0,85	1,00	-0,81	-0,75	-0,85	0,67	-0,79
3. Pronoun	-0,87	-0,81	1,00	0,70	0,79	-0,78	0,77
4. Adverb	-0,8	-0,75	0,70	1,00	0,80	-0,69	0,76
5. Verb	-0,88	-0,85	0,79	0,80	1,00	-0,75	0,75
6. Participle	0,77	0,67	-0,78	-0,69	-0,75	1,00	-0,77
7. Particle	-0,86	-0,79	0,77	0,76	0,75	-0,77	1,00

Table 2: Correlation coefficient of the most correlated POS.

The results related to the morphological structure of the functional styles are not surprising but give a good overview and are based on experimental work. The morphological characteristics of the training sample are introduced in [1] in detail.

2.3. Results and Evaluation

Discriminant analysis (DA) module of the STATISTICA system [10] was employed for classification function generation.

After numerous optimization runs we obtained five linear classification functions based on only 7 features. The classification functions can be presented in the form $\bar{s} = A\bar{x} + \bar{b}$, where

$$A = \begin{pmatrix} 458{,}75 & 318{,}61 & 37{,}47 & 0{,}09 & 8{,}95 & -252{,}01 & -238{,}23 \\ 471{,}56 & 313{,}31 & 40{,}60 & 0{,}25 & 23{,}62 & -292{,}64 & -244{,}43 \\ 408{,}92 & 275{,}27 & 44{,}98 & 0{,}29 & 67{,}34 & -393{,}37 & -197{,}99 \\ 367{,}12 & 173{,}79 & 50{,}59 & 0{,}11 & 436{,}48 & 157{,}17 & -201{,}34 \\ 287{,}77 & 122{,}34 & 48{,}41 & 0{,}40 & 190{,}42 & -423{,}40 & 160{,}86 \end{pmatrix}, \quad \bar{b} = \begin{pmatrix} -139{,}79 \\ -158{,}25 \\ -173{,}39 \\ -211{,}02 \\ -192{,}68 \end{pmatrix},$$

$\bar{x}$ is a feature vector (x_1 – verb rate, x_2 – adverb rate, x_3 – average word length, x_4 – average sentence length, x_5 – scientific prefix rate, x_6 – general scientific term rate, and x_7 – official document name rate), and $\bar{s}$ contains resulting scores (s_1 – everyday communication style, s_2 – literary style, s_3 – journalistic style, s_4 – academic style, and s_5 – official style). As mentioned above, the case is classified as belonging to the group for which it has the highest classification score. The optimization does not decrease the computational cost significantly since the classification functions are linear and the most consumptive operation is the obtaining of the morphological features. The optimization serves the purpose of the style description clarity and ease.

There was no free and representative Russian corpus that could be used for the evaluation at the moment the experiment was conducted. Besides, we wanted to test the obtained procedure within the proposed Web information retrieval framework. For this purpose we selected a number of fairly generic queries from the Yandex search engine [13] log, which could be of interest for stylistic categorization of search results (i.e. we rejected queries containing specific technical abbreviations, exact institutions' names, etc.). Then, we downloaded the documents from the top of the result lists (up to 130 for each query). After that, we removed documents in Ukrainian, short documents (up to 500 words), and non-coherent text documents (rejection was made using verb rate). On the next stage we had to attribute the documents to the five functional styles. It was difficult in many cases, as there were numerous texts that fall poorly in the five-style system (advertising and religious texts, mixed-content and therefore mixed-style documents, translations of ancient authors, etc.), as well as intermediate style documents (popular science, scientific news, etc.). We marked the texts of the former type as 'undefined style'. Tables 3 and 4 show evaluation of the stylistic categorization of the documents returned in response to queries '*небесные тела*' and '*расход воды в нагревательных печах*', respectively. The columns reflect the automatic categorization, whereas the rows – the manual assessment.

	1	**2**	**3**	**4**	**5**	**Total**	**Recall**
1. Everyday Communication	-	-	-	-	-	-	-
2. Literary	-	2	1	-	-	3	0,67
3. Journalistic	-	2	44	8	-	54	0,81
4. Academic	-	-	10	17	-	27	0,63
5. Official	-	-	-	-	7	7	1,0
Undefined	1	1	7	3	-	12	-
Total	1	5	62	28	7	103	
Precision	0,0	0,2	0,71	0,61	1,0		0,74

Table 3: Search results categorization. Query: '*небесные тела*' ('*celestial bodies*')

	1	**2**	**3**	**Total**	**Recall**
1. Journalistic	9	1	1	11	0,82
2. Academic	3	7	2	12	0,58
3. Official	-	-	5	5	1,0
Undefined		1	-	1	-
Total	12	9	8	29	
Precision	0,75	0,78	0,63		0,72

Table 4: Search results categorization.
Query: '*расход воды в нагревательных печах*' ('*water consumption in heating furnaces*')

As the tables show, both the variety of styles presented in the search engine responses and the categorization quality are query-sensitive to a certain extent. However, the average quality of stylistic categorization for coherent Russian texts lay in the range 0,7-0,8.

3. EXPERIMENT II

3.1. Experimental setting

The goal of the second series of experiments was to develop an automated procedure for maintaining Yandex Web directory [12], which is built using faceted classification (FC). A FC contains a number of independent classifications that allow users to see the Web resources from the different points of view and keep the topical taxonomy reasonably simple at the same time. The FC schema

used in Yandex directory includes a *genre* facet, among others [2]. In this case, *genre* is a property of the whole website; individual documents inherit the genre label from respective site roots.

There were 11 genres presented at the moment the research began. It is to be noticed that the genre set was pretty vague and ill-defined. So we limited the experimental set to four genres: *scientific papers*, *official documents*, *fiction*, and *guidance* (the last genre was still fairly imprecise). Since the selected genres were far from covering all the variety of genres, an additional rejection procedure had to be developed.

The training sample consisted of 285 documents in Russian (*fiction* – 77, *scientific papers* – 48, *official document*s – 94, *guidance* – 66). A set of initial features was constructed similarly to the previous experiment. Three parameters were based on the word lists statistically built upon official documents, scientific papers, and guidance subsets of the training sample. The morphological guesser *mystem* by Yandex was used for obtaining morphological features in this experiment. The advantage of *mystem* is that it works faster, and the morphological features of even unknown words can be resolved (see [9] for details).

3.2. Results and Evaluation

The same methods for building classification functions and similar optimization procedures were employed as in the previous experiment. Additionally, a procedure for type I and II error probability estimation was developed based on the Mahalanobis distance.

The resulting classification function employed again 7 features and had the following parameters:

$$A = \begin{pmatrix} 28{,}11 & -31{,}97 & 85{,}66 & -217{,}40 & 488{,}38 & -46{,}81 & 319{,}67 \\ 32{,}04 & -30{,}84 & 138{,}48 & 93{,}24 & 471{,}69 & -53{,}24 & 168{,}15 \\ 33{,}96 & 67{,}75 & 133{,}35 & -172{,}96 & 426{,}42 & -81{,}72 & 131{,}97 \\ 28{,}98 & -26{,}55 & 305{,}66 & -182{,}24 & 617{,}48 & 11{,}26 & 202{,}92 \end{pmatrix}, \quad \bar{b} = \begin{pmatrix} -105{,}17 \\ -121{,}30 \\ -131{,}84 \\ -107{,}96 \end{pmatrix}, \quad \text{and}$$

vector $\bar{x}$ elements are as follows: x_1 – average word length, x_2, x_3, x_4 – rates of the *OfficialDocument*, *Guidance*, and *Science* word lists items, respectively, x_5 – adverb rate, x_6 – template '*{можно| нужно}* + *Infinitive*'* occurrence rate, and x_7 – verb rate. Vector $\bar{s}$ elements are as follows: s_1 – fiction, s_2 – scientific papers, s_3 – official documents, and s_4 – guidance.

A test sample was gathered from the Yandex document repository. The sample consisted of 'long' (longer than 500 words) Russian documents belonging to the sites both with and without genre labels. The sample was randomly sifted and presented to 5 assessors for manual processing. A document was assigned to a genre if the opinions of three of five assessors agreed. The final test sample consisted of 291 documents, 135 of them were attributed to one of four mentioned genres. In the first test we used the smaller subset of the "known" genres. In the second test the whole sample was categorized. In the third test the rejection procedure was applied. The results can be seen in table 5.

	Test 1		**Test 2**		**Test 3**	
	P	**R**	**P**	**R**	**P**	**R**
Fiction	0.857	0.913	0.506	0.913	0.709	0.848
Science	0.912	0.912	0.553	0.724	0.682	0.517

* English equivalents: '*{should|can}* + *Infinitive*' (guidance genre cue).

OffDoc	0.950	0.864	0.452	0.864	0.842	0.727
Guidance	0.750	0.727	0.267	0.727	0.423	0.333
Other	-	-	-	-	0.633	0.705
Total	0.859		0.436		0.649	

Table 5: Test sample classification (P – precision, R – recall)

The majority of rejected documents could be attributed to news articles. This fact supported the decision to change the genre schema used in the directory. The renewed schema consists of 6 genres that tend to conform to the functional styles: *fiction*, *scientific and technical documents*, *official documents*, *guidance*, and *news articles*.

	P	**R**
Fiction	0.788	0.565
Science	0.447	0.500
OffDoc	0.783	0.818
Guidance	0.618	0.636

Table 6: Site genre vs. document genre (P – precision, R – recall)

Regardless of automated categorization results, the comparison of manual assessments and inheritance of genre labels from site roots to documents allows for making some interesting remarks. As table 6 shows, relatively high precision is observed in case of *fiction* and *official documents*, whereas only one genre – *official documents* – demonstrates high recall. According to those data we can assume that *official documents* are presented on the Russian Web as compact collections for the most part. *Fiction* is accessible both in the form of collections and isolated texts. Both *guidance* and *scientific papers* are pretty scattered across the Russian Web.

3. CONCLUSIONS AND ONGOING WORK

Our experiments have shown that easily computable text features are feasible for effective stylistic categorization. Some obtained results could also be of interest for theoretical stylistics. However, the second experiment series has proven a simple truth that the statistical methods cannot help without a solid conceptual basis.

Unfortunately we failed to implement the approach into a real-word application yet. The commercial Web search services believe that style, as an additional search feature, is unnecessary for the majority of users. We should probably turn to the more exacting realm of digital libraries as [6] suggests. At the moment we are going to make another attempt and research whether the style-related parameters could improve relevance ranking.

In the framework of the fist series of experiments we applied canonical discriminant analysis and the principal components method [11] to the experimental data. In case of correlated features the methods allow for a linear space transformation and subsequent shifting to a lower space dimension with minimal information loss (the fewer coordinates would explain the greater part of the overall variance).

The scatterplot of the training sample in the first and second principal directions can be seen in figure 3. It shows that the first component describes fairly well the variations of features across different styles. The lexical parameters contribute for distinction of academic and official styles (the second component).

This fact suggested the idea to reduce the description of styles to a single continuous parameter (a similar idea – understanding genres in terms of structural similarity rather than as a predefined set of classes – is expressed in [6]). Particularly, the linear combination of the initial features might serve for relevance ranking in information retrieval tasks. An additional advantage is that we do not need the time-consuming phase of training sample building in this kind of experiment.

Now we are planning to conduct an experiment on the ROMIP/RIRES data [7]. This test collection represents a 7+ Gb subset of the narod.ru domain including 600 000+ HTML pages in Russian from more than 20 000 websites. An important part of the collection is 54 evaluated queries for the classical ad-hoc task.

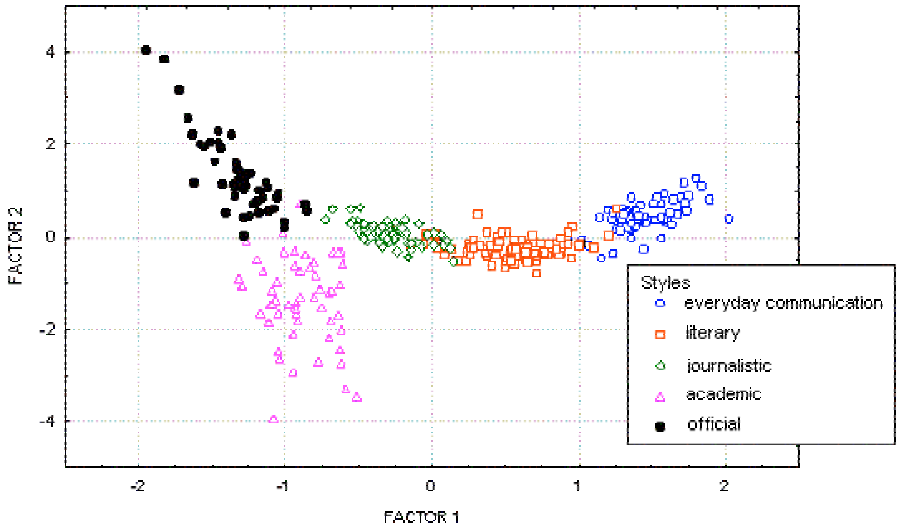


Figure 3: Two-dimensional scatter plot of the learning sample

We are going to conduct both global and local document analysis. In the former case we will obtain the new principal directions based on processing a reasonably large subset of the collection. Then 'stylistic scores' can be computed for every document in the same way. In the latter case every document subset returned in response to a query will be processed separately. In other words, the principal directions will be resolved for each evaluated query independently. We are going also to investigate the interplay between traditional relevance scores and stylistic features.

In addition, we examine the possibilities to make use of the Russian National Corpus [8] for the future stylistic experiments. The project was launched on April 27, 2004. The total amount of the collection is supposed to reach 100 million words by the end of 2005.

4. ACKNOWLEDGEMENTS

I would like to thank Mikhail Shchekotilov and Andrei Tselishchev for helping to develop the software, as well as Mikhail Maslov and Ilya Segalovich from the Yandex team for co-operation and support at the second stage of the experiments.

I am especially grateful to the ESSLLI 2004 Local Organization for a grant that made possible my participation in the summer school.

I would also thank the anonymous reviewers, whose comments helped me to improve the paper.

5. REFERENCES

[1] Braslavski, P. Morphological Structure of the Functional Styles: Case Study on Internet Documents. (in Russian) [Morfologičeskaya struktura funkcional'nyh stilei (na materiale dokumentov Internet)]. Proceedings of the Ural State University, Ekaterinburg, 2001, vol. 21, p. 8-17. Available at: http://proceedings.usu.ru/proceedings/N21_01/win/03.html

[2] Braslavski, P., Maslov, M., and Vovk, E. Facet-Based Internet Directory Design and Automated Genre Classification of Documents (in Russian). [Fasetnaya organizaciya internet-kataloga i avtomatičeskaya žanrovaya klassifikaciya documentov]. Proceedings of the International Workshop "Dialogue-2002. Computational Linguistics and Intelligent Technologies", Moscow, 2002, vol. 2, p. 83-93. Available at: http://company.yandex.ru/articles/article8.html

[3] Karlgren, J. and Cutting, D. Recognizing Text Genres with Simple Metrics Using Discriminant Analysis. Proceedings of the 15th International Conference on Computational Linguistics (COLING), Kyoto, 1994, vol. 2, p. 1071-1075.

[4] Kožina M.N. Foundations of the Functional Stylistics (in Russian). [K osnovaniyam funkcional'noi stilistiki], Perm, 1968.

[5] Lee, Y.-B. and Myaeng, S. H. Text Genre Classification with Genre-Revealing and Subject-Revealing Features. Proceedings of the SIGIR'02, August 2002, Tampere, Finland, p. 145-150.

[6] Rauber, A. and Müller-Kögler, A. Integrating Automatic Genre Analysis into Digital Libraries. In Proceedings of the JCDL'01, June 2001, Roanoke, Virginia, USA, p. 1-10.

[7] Russian Information Retrieval Evaluation Seminar, http://romip.narod.ru

[8] Russian National Corpus, http://www.ruscorpora.ru

[9] Segalovich, I. A Fast Morphological Algorithm with Unknown Word Guessing Induced By a Dictionary for a Web Search Engine. In Proc. of the MLMTA-2003, Las Vegas, June 2003. Available at: http://company.yandex.ru/articles/iseg-las-vegas.html

[10] Statistica, http://www.statsoft.com

[11] StatSoft, Inc. (2004). Electronic Statistics Textbook. Tulsa, OK: StatSoft, http://www.statsoft.com/textbook/stathome.html

[12] Yandex Directory, http://yaca.yandex.ru

[13] Yandex Search Engine, http://www.yandex.ru

Integrating Finite-state Technology with Deep LFG Grammars[1]

Ronald M. Kaplan, John T. Maxwell III, Tracy Holloway King, and Richard Crouch
Palo Alto Research Center
{kaplan|maxwell|thking|crouch}@parc.com

1 Introduction

Researchers at PARC were pioneers in developing finite-state methods for applications in computational linguistics, and one of the original motivations was to provide a coherent architecture for the integration of lower-level lexical processing with higher-level syntactic analysis (Kaplan and Kay, 1981; Karttunen et al., 1992; Kaplan and Kay, 1994). Finite-state methods for tokenizing and morphological analysis were initially incorporated into the Grammar-writer's Workbench for Lexical Functional Grammar (LFG) (Dalrymple, 2001; Kaplan and Bresnan, 1982), a relatively complete LFG parsing system but for relatively small-scale grammars (Kaplan and Maxwell, 1996). Finite-state transducers, also of a relatively small scale, were constructed automatically in this system from specifications provided by the LFG grammar writer, using the compilation techniques described by (Kaplan and Kay, 1994).

The algorithms and interfaces developed in the Grammar-writer's Workbench were re-implemented and extended to create a hardened, industrial-scale parsing and generation environment for broad-coverage LFG grammars, called XLE. By the time the new system was being designed, the finite-state approach to lexical processing had firmly taken hold and substantial morphological transducers were being produced for many computational applications in many languages. Thus, the XLE system was organized to make it easy to combine LFG syntactic specifications with externally developed lexical resources embodied in finite-state transducers. Our experience over the last decade has demonstrated that existing FST morphologies are invaluable in creating broad-coverage grammars and using them for efficient deep LFG parsing and generation.

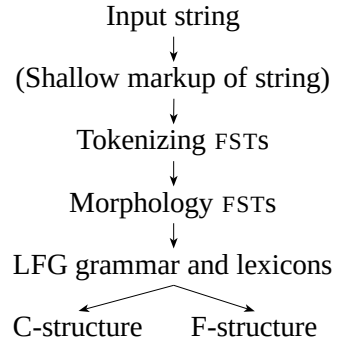
In this paper we describe two uses of FSTs in the XLE ParGram grammars (Butt et al., 1999; Butt et al., 2002).[2] The examples discussed come from the English grammar, but FSTs are used in this way in the majority of the grammars (there are grammars for English, French, German, Japanese, Norwegian, and Urdu; grammars for Danish, Malagasy, and Welsh are also under development). The basic organization of the XLE system is diagramed in (1). So, a string like *Boys appeared.* will pass through the system as in (2). Note that all components of the system are non-deterministic and can pass multiple output to the next component; this is necessary to avoid pruning potentially good parses too early as there is no back-tracking among the components. This non-deterministic cascading can be done efficiently in XLE due to its ambiguity packing technology (Maxwell and Kaplan, 1989).

Section 2 discusses the intergration of morphological FSTs into the grammars. This integration has resulted in greatly decreasing the lexicon development task for the grammars. Section 3 discusses using FSTs for tokenization and the intergration of shallow markup such as part of speech tagging.

[1]This work was supported in part by the Advanced Research and Development Activity (ARDA)'s Novel Intelligence from Massive Data (NIMD) program under contract # MDA904-03-C-0404.

[2]The ParGram project develops parallel deep grammars for a number of languages. The grammar produce similar output structures to aid in multi-lingual applications, such as machine translation, and in porting applications from one language to another.

(1)



(2)

Input string:	Boys appeared.
Tokenizing:	boys TB appeared TB . TB
Morphology:	boy +Noun +Pl
	appear +Verb +PastBoth +123SP .
C-structure:	Figure 1
F-structure:	FIgure 2

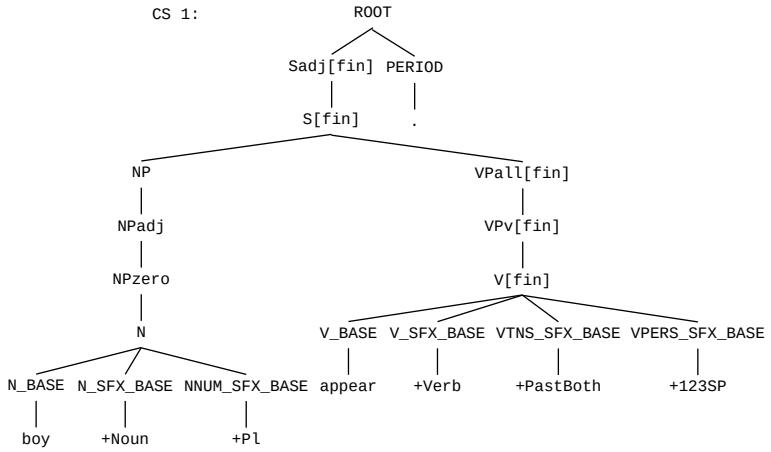


Figure 1: Constituent-structure for *Boys appeared.*

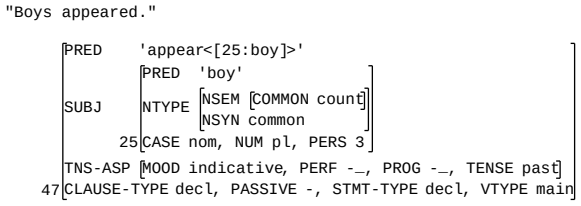


Figure 2: Functional-structure for *Boys appeared.*

2 Morphologies

In this section we describe how large FST morphologies can be incorporated into the XLE LFG grammars. These morphologies can either have been created for other purposes and used "off the shelf" or can be created as part of the grammar development process. Fortunately, large FST morphologies already exist for many languages (e.g., produced by companies like Inxight).

Before demonstrating how the morphologies are integrated into the deep grammars, it is important to explain what they do. FST morphologies associate a surface form with a canonical form (stem or lemma) and a set of tags that encode relevant morphological features. Some English examples are shown in (3).

(3) a. turnips $\Longleftrightarrow$ turnip +Noun +Pl

b. Mary $\Longleftrightarrow$ Mary +Prop +Giv +Fem +Sg

c. wound $\Longleftrightarrow$ wound +Noun +Sg
wound +Verb +Non3sg
wind +Verb +PastBoth +123SP

(3a) states that the surface form *turnips* can be analyzed as a stem *turnip* and two morphological tags. The first tag *+Noun* indicates the part of speech. The second tag *+Pl* indicates that the word is plural in number. Note that the surface form *turnip* would have corresponded to the same stem and part of speech tag, but with the number tag *+Sg* corresponding to its singular status. (3b) states that the surface form *Mary* can be analyzed as a stem *Mary* and four tags which indicate that it is a proper noun, is a given femine name, and is singular. (3c) demonstrates how a single surface form can correspond to more than one analysis. The surface form *wound* has a noun and two verb analyses. That is, it can be a singular noun, in which case it stems to *wound* and is associated with the tags *+Noun* and *+Sg*. Or, it can be either the base form of a verb, in which case it stems to *wound* and is associated with the tags *+Verb* and *+Non3sg*, or it can be a past tense verb, in which case it stems to *wind* and is associated with the tags *+Verb*, *+PastBoth*, and *+123SP*. All three forms are passed to the grammar for parsing in order to avoid early pruning of potentially correct analyses (but see section 3 on using POS tagging as a filter).

The exact set of tags depends on the language and the desired application. In general, the FST morphologies used in the ParGram project were not designed for deep grammars. Instead, their primary intended application was information retrieval. Because of this, the analysis of proper nouns is extremely detailed (witness the four tags for the form *Mary* in (3b)), while from a syntactic standpoint this level of detail is probably not necessary. However, as long as the tag set and tag order is known, it is possible to integrate the FST into the grammar. In addition, it is possible to use FST tools to modify existing FSTs so that their output tags are more suited for deep grammars (see (Kaplan and Newman, 1997) for a full discussion of this issue). The Norwegian FST, for example, was modified because the original representation of the Norwegian gender system did not allow the grammar writers to capture some needed distinctions. The English FST was also restricted to prevent the generator from generating both British and American spellings while still permitting alternative spellings to be recognized in parsing.

The association between surface forms and stems plus tags is extremely useful even in a language like English which has relatively impoverished morphology. The more inflectional morphology that a language has, the more useful such FST morphologies are. This is because the integration of the FST morphology into the grammar allows there to be a single lexical entry for the stem form of the word, avoiding having to list all the surface forms. In addition, as discussed in the next section, words with predictable subcategorization frames do not need to have an explicit lexical entry at all.

2.1 Basic Integration

This section describes how the FST morphologies are integrated into the deep grammars (again, see (Kaplan and Newman, 1997) for more details). The basic idea is as follows. First the surface forms are run through the FST morphology to produce the corresponding stems and tags. Stems and tags each have entries in the LFG lexicon. Sublexical phrase-structure rules produce syntactic nodes covering these stems and tags and standard grammar rules then build larger phrases.

In order to do this, XLE must know which morphological transducers to use with a particular grammar. This is specified in the MORPHCONFIG associated with each grammar. A MORPHCONFIG might look like (4) (the tokenizers are discussed in the section 3).

(4) STANDARD ENGLISH MORPHOLOGY (1.0)
TOKENIZE: ../common/english.tok.parse.fst
ANALYZE: ../common/english.infl.fst
MULTIWORD: english.standard.mw.fst

(4) states that the tokenizer for the grammar is the file english.tok.parse.fst (with the path name where this FST is located). The FST morphological analyzer is the file english.infl.fst. The multiword analyzer, which is not discussed here, is the file english.standard.mw.fst.

When XLE parses a string, it first passes it through the tokenizer to non-deterministically divide the string into tokens. It then passes the tokens through the FST morphology. This results in a set of stems and tags that the LFG grammar builds into phrases and assigns a well-formed functional structure. There may be multiple sets which are all passed to the grammar for parsing. To make this more concrete, consider the sentence in (5a). Once (5a) is run through the FST morphology the result will be as in (5b).

(5) a. Boys appeared.
b. boy +Noun +Pl appear +Verb +PastBoth +123SP .

XLE looks up the lexical entry for each of these items. These are shown in (6). Note that tags and punctuation have lexical entries, just as stems do.

(6)

boy	N	XLE	@(NOUN boy).
+Noun	N_SFX	XLE	@(PERS 3) @(EXISTS NTYPE).
+Pl	NNUM_SFX	XLE	@(NUM pl).
appear	V	XLE	@(V-SUBJ appear).
+Verb	V_SFX	XLE	@(EXISTS VTYPE).
+PastBoth	VTNS_SFX	XLE	@(TENSE past).
+123SP	VPERS_SFX	XLE.	
.	PERIOD	*.	

A lexical entry begins with a head (e.g, *boy*) that matches the stem or tag provided by the morphology. This is followed by a part of speech, e.g, *N*. Next comes a code that XLE uses to determine whether this entry is allowed to match items coming from the transducer morphology, indicated by XLE, or whether it can match against an unanalyzed token, indicated by *. The vast majority of words go through the morphology. A token reading with * is used only when the grammar writer wishes to allow an alternative to the information provided by the morphology or wishes to provide an entry for a token not found in the morphology: both of these cases are rare because the FST morphologies have excellent lexical coverage. Finally, the remainder of the lexical entry comprises calls to templates which provide the functional information used by the grammar. For example, the *+Noun* tag calls templates that provide the feature PERS with the value 3 and that require the f-structure to have a value for NTYPE. As seen by the entries for *+123SG* and the period, some lexical items do not call any templates.

The grammar combines the stems and tags into low-level phrase-structure nodes using sublexical rules. Sublexical rules are similar to regular LFG rules, although they tend to be much simpler in that they do not have complex functional annotations. For example, the sublexical rule for common nouns like *boys* would look like (7).

(7) N $\longrightarrow$ N_BASE N_SFX_BASE NNUM_SFX_BASE.

The only difference between regular rules and sublexical rules is in the display: there is a display toggle to hide the sublexical information so that the leaves of the tree are the inflected tokens instead of the

stem and the tags. The display with sublexical information is shown in Figure 1 for our example sentence *Boys appeared*.[3] The corresponding f-structure is shown in Figure 2; here the person and tense information provided by the tags is seen.

XLE provides an additional feature which makes the use of FST morphologies ideal for simplifying lexicon development. For lexical items with unpredictable subcategorization frames, it is necessary to provide a lexical entry for each lexical item. This is primarily the case for verbs, but also for adjectives that take oblique prepositional phrases and nouns that take clausal arguments, for example. However, the vast majority of lexical items have entirely predictable subcategorization frames. That is, knowing the part of speech and some inflectional information is sufficient for determining the lexical entry. This is the case for most common and proper nouns, adjectives, adverbs, and numbers. XLE defines a special lexical entry *-unknown* which matches any stem that comes from the morphology. This lexical entry contains these predictable parts of speech with the associated template calls. Consider the simplified entry for *-unknown* in (8).

(8) -unknown N XLE @(NOUN %stem);
A XLE @(ADJ %stem);
ADV XLE @(ADVERB %stem).

This entry states that any item can be a noun N, an adjective A, or an adverb ADV. The *%stem* is a variable that denotes the unknown stem and makes it available to the template. For example, if the stem is *boy*, the form *boy* would be passed to the NOUN template. Obviously, most words are not nouns, adjectives, *and* adverbs and so the grammar should not build analyses for all of these possibilities for each stem. The appropriate restrictions are provided by the sublexical rules: the noun entry will only go through if the morphology assigns the stem in question the appropriate noun tags, etc.

A stem is matched against the *-unknown* entry only as a last resort, if it matches no other entry in the lexicon. Thus, given the lexical entries shown in (6), *boy* would not match *-unknown*. But the explicit entry for *boy* can be removed to simplify the lexicon, and *boy* would then match against the N option in the *-unknown* lexical entry in (8).

By integrating FST morphologies into XLE grammars, it is possible to significantly reduce the size of the lexicons and the amount of time spent in their development. Not only do the FST morphologies make it so that only stem forms need lexical entries, but the existence of the *-unknown* entry means that the majority of lexical items need no explicit lexical entry at all. Instead, they are built on the fly based on information in the morphology FST and the sublexical rules.

2.2 Multiple FSTs

The previous section discussed how FST morphologies are integrated into XLE. In this section we briefly review the methods described by (Kaplan and Newman, 1997) for obtaining finer control over the morphology output provided to the deep grammar.

Although the FST morphologies used by the XLE ParGram grammars are extremely large, they can never be complete. New words are constantly being added to the language, by being borrowed, morphed from one part of speech to another, or newly coined. Fortunately, novel words tend to follow regular morphological patterns. This can be exploited by the grammars by creating a guesser FST morphology which will guess the part of speech and other inflectional tags based on the surface form of a word. For example, English words ending in *-ed* can be past tense verbs or deverbal adjectives, words

[3]The detail oriented reader will notice that the part of speech categories in the tree and the sublexical rule in (7) end in _BASE while the lexical entries in (6) do not. XLE automatically adds these endings and uses them in determining how to display these lexical items.

ending in *-s* can be plural nouns or third singular verbs, and words beginning with a capital letter can be proper names.

However, in general it is not desirable to have the grammar guess at a surface word form unless this form cannot be found in the regular morphology. The order in which FSTs (both tokenizers and morphologies) apply is specified in the MORPHCONFIG.

(9) STANDARD ENGLISH MORPHOLOGY (1.0)
TOKENIZE: ../common/english.tok.parse.fst
ANALYZE: ../common/english.infl.fst
../common/english.guesser.fst
MULTIWORD: english.standard.mw.fst

In the sample MORPHCONFIG in (9), the FST morphologies are ordered. XLE will apply the words to the FST morphologies *in order* until an analysis is found. So, for a word like *boys* which is known by the standard English morphology english.infl.fst, XLE will apply that FST to *boys*; this results in the output *boy +Noun +Pl*; XLE then stops applying FSTs to *boys* and moves on to the next word. However, for a word like *Saakashvili*, which is a name not known to the English morphology, XLE will first apply the standard English morphology, but this will result in no output; so, it then applies the next FST english.guesser.fst. Due to the initial capital, the guesser will produce a stem *Saakashvili* and the appropriate tags for a proper noun, as well as a tag *+Guessed*.[4]

Generally, the XLE LFG grammars use two morphological FSTs: a large standard morphology and a guesser. However, grammars adapted for specific corpora (Kaplan et al., 2002), may use additional FSTs. These FSTs can cover novel terminology, multiword expressions, or words that are unknown to the standard morphology but which the grammar writer does not wish to have go through the guesser (e.g., verbs specific to the corpus). For example, the English EUREKA grammar was specialized to parse copier repair tips, and a morphological FST was added to recognize the regular patterns that comprise part and fault numbers, such as those in (10a). In addition, this grammar uses a special FST to recognize multiword copier part names, as in (10b).

(10) a. 600T40506
D-034
PL3-E11

b. cam follower
CRT controller PWB
airknife solenoid assembly

Thus, being able to use multiple FST morphologies for a grammar allows for finer control of the types of words recognized by the grammar and the types of analyzes that they receive.

3 Tokenizers and Shallow Markup

The use of FST morphologies described above presupposes that the string of characters of a sentence has been split up into individual tokens. Tokenizing is relatively simple, although not trivial, when processing written text with standard punctuation marks. It becomes more complicated when the input string contains shallow markup, such as part of speech tagging. In this section, we first discuss standard tokenization using FSTs and how this interfaces with the deep grammars. We then examine the modifications that are needed to allow for shallow markup of the grammar input.

3.1 Tokenizers

The process of tokenization breaks up a string (e.g. a sentence) into tokens (e.g. words). As with the output of the FST morphologies, the output of the tokenizer need not be deterministic and all possible

[4]All guessed words are provided with a distinctive tag. This makes it possible to provide them with a feature in the f-structure marking their source, which can be used in debugging and by applications to determine the reliability of the analysis. In addition, it allows the grammar writer to defi ne OT marks (Frank et al., 2001) to (dis)prefer analyses with guessed words.

tokenizations are passed to the next component, namely the morphologies. Most deep grammars work on tokenized input, in the simplest case treating all punctuation as spaces. Here we describe a more complex system that allows the grammar writer to use the information provided by the punctuation to constrain the grammar. In English,[5] tokenization involves: breaking off punctuation, breaking off clitics, and lower casing certain letters. We consider these in turn.

Breaking off punction is often a simple task. Consider the examples in (11) where the string is on the left and the tokenized form on the right. TB indicates a token boundary.[6]

(11) a. I see them. $\Longrightarrow$ I TB see TB them TB . TB

b. the dog, a poodle, $\Longrightarrow$ the TB dog TB , TB a TB poodle TB , TB

In each example, the punctuation has been split off from the preceding word. This is crucial because the morphology and lexicon will recognize forms like *them* and *dog* but not forms like *them.* and *dog.*, which are what occur in the string.

It turns out that in many languages, it is not enough simply to insert token boundaries around punctuation. This is because it may be much easier for a punctuation-sensitive syntactic grammar to require punctuation marks for certain constructions even though the rules of normal English typography do not allow those marks to appear in the surface string. For example, the rules for English appositives and parentheticals are simpler if they specify that those constructions are set off by commas, but according to rules of English haplology, those marks are not always present in the string. In English, sentence periods consume commas, as in (12a), and abbreviatory periods, as in (12b).

(12) a. Find the dog, a poodle. $\Longrightarrow$ Find TB the TB dog TB , TB a TB poodle TB , TB . TB

b. Go to Palm Dr. $\Longrightarrow$ Go TB to TB Palm TB Dr. TB . TB

In (12a), a comma must be inserted by the tokenizer between the *poodle* token and the . token. In fact, an arbitrary number of commas can potentially be inserted to account for nested parentheticals; this is straightforward to do with FSTs.[7] In (12b), the ending period token must allow for an abbreviatory period that is part of the *Dr.* token. All of these possibilities are non-deterministically passed to the FST morphology to avoid potentially incorrect pruning of analyses.

Another role of the tokenizer in English is to split off clitics. An example is seen in (13).

(13) I'll go. $\Longrightarrow$ I TB 'll TB go TB . TB

By splitting off the clitic auxiliary *'ll*, the grammar can treat this auxiliary similarly to its full counterpart *will*.

The above examples are relatively straightforward and are intended to provide a feel for the tokenization process. This process can become more complex because it is not always clear when to break off punctuation. For example, hyphens may be word internal or join separate words; in the former case they should not be broken off by the tokenizer while in the latter they should. The syntactic processing algorithms in XLE operate on arbitrary regular languages and thus allow for the tokenizing transducer to be nondeterministic as well as not linear-bounded. For example, any sentence final period will allow for zero or more haplology commas.

[5]The Japanese ParGram grammar uses the well-known ChaSen tokenizer (Asahara and Matsumoto, 2000). This tokenizer combines some of the features of a tokenizer with those of a morphology. It breaks the string of characters into words and then provides some basic part of speech tagging for those words. For details on how the ChaSen tokenizer was integrated into the XLE Japanese grammar, see (Masuichi et al., 2003).

[6]By convention XLE requires a final token boundary but not an initial one. Nothing crucial rests on this.

[7]This behavior is provided by transducers that formally have the property of not being linear bounded. A technical adjustment to LFG's prohibition against nonbranching dominance chains is required to avoid computational difficulties that could otherwise arise from the composition of an LFG grammar and a non-linear-bounded transducer.

Another source of non-determinism for English tokenizers is to lowercase capitalized words in certain positions. This may occur in sentence initial position, as in (14a), although it must be optional, as in (14b). In addition, it may occur in other positions, such as after colons, as in (14). (In the examples below, only the lowercasing is shown, not the insertion of TB token boundaries.)

(14) a. The boy left. $\Longrightarrow$ the boy left.
b. Mary left. $\Longrightarrow$ Mary left.

(15) The boy left: He was unhappy. $\Longrightarrow$ the boy left: he was unhappy.

Thus, a tokenizer needs to know which environments (e.g., sentence initially and after colons) trigger optional lower casing. Otherwise, the input tokens will not match the morphology and lexicon. For example, *Bush* and *bush* are different words with different morphological and syntactic properties.

To provide an integrated example, consider (16). The initial word can be lower cased or not and haplology commas are provided for, shown here by curly braces for the disjunction and the Kleene star notation for the haplology.

(16) Bush saw them. $\Longrightarrow$ { Bush | bush } TB saw TB them TB [, TB]* . TB

As the rules of tokenization vary from language to language, it is necessary to write tokenizers for each language. This can be done relatively easily with off-the-shelf finite-state technology (e.g., that which comes with (Beesley and Karttunen, 2003)). XLE provides a default parsing and generation tokenizer which simply splits off punctuation. The French and German grammars have used the English tokenizer with moderate success, but these are being specialized for those languages to improve accuracy.

3.2 Shallow Markup

As discussed in (Kaplan and King, 2003), it is possible to use FSTs in conjunction with minor changes to the XLE LFG grammars to allow for input strings which contain shallow markup. The idea behind using shallow markup is that such markup may decrease ambiguity and increase accuracy while decreasing the amount of processing time. Shallow markup can include part of speech (POS) tagging (17) and named entities (18).

(17) a. I/PRP_ saw/VBD_ her/PRP_ duck/VB_.
b. I/PRP_ saw/VBD_ her/PRP$_ duck/NN_.

(18) a. <person>General Mills</person> bought it.
b. <company>General Mills</company> bought it.

An evaluation of the usefulness of this markup in conjunction with deep grammars is discussed in (Kaplan and King, 2003). Basically, the POS tagging was found to improve speed but to decrease accuracy because of compatibility failures between the tagging and the correct analysis; further investigations indicate that using partial POS tagging with improved back-off techniques when matching fails can result in an increase in speed and accuracy. The named entity markup improved both speed and accuracy, in part because the internal structure of proper names was not built by the parser. Here we summarize how FSTs can be used to process this marked up input in such a way that the deep grammar can operate on it correctly.

The first issue is that the normal rules of tokenization discussed above, such as lowercasing and comma haplology, need to skip the markup. The second is that this markup should not be broken up or tokenized as if it were standard punctuation. Finally, the markup must be interpreted correctly by the grammar: with POS tags this is done at the level of the FST morphology; with named entities this is done by a combination of FST tokenization and the grammar.

Part of Speech Tagging POS tags are not relevant to tokenization, but the tokenizing transducer must be modified so that the POS tags do not interfere with the other patterns that the tokenizer is concerned with. The proper effect of a POS tag is to reduce the number of outputs from the morphological analyzer. Consider the words *walks* in (19).[8]

(19) She walks/VBZ_.

The morphological analyzer normally would produce two outputs for *walks* of which only one is appropriate in this context, the verb. However, with unmarked strings, XLE passes both analyses to the grammar. The POS tags are used to trim the analyses early by only passing the morphological analyses compatible witht the POS tagger to the grammar.

Obtaining the desired behavior requires a specification of the morphological output sequences that are consistent with a given POS tag. This is done by a hand-written mapping table that states legal correspondences between POS tags and the morphological analyzer tags.

Given this mapping table, we produce a FST that filters the normal output of the morphology FST. The filtering FST, for example, allows pairs of lemmas and morphological indicators to be followed by VBZ if and only if the indicators are compatible with the allowable sequences drawn from the table. Thus the indicator +Noun is not allowed in front of VBZ. The filtering FST also maps the POS tags to epsilon, so that they do not appear in its output. This FST is put in a cascade with the tokenizer and morphology FSTs. The overall effect is that the tags in the input are preserved by the tokenizer, pass through the morphological analyzer, and then cause incompatible morphological-indicator sequences to be discarded. Thus only the contextually appropriate interpretations are passed on to the grammar. Another version of the filtering FST allows through both the compatible sequences and the incompatible ones, but marks the incompatible ones with an additional tag so that the grammar can disprefer parses with these tags, only using them on a second pass if parsing with the compatible tags fails; this two-stage parsing is controlled by OT marks (Frank et al., 2001).

Named Entities The named entities appear in the text as XML mark-up, as in (20).

(20) <company>General Mills</company> bought it.

The tokenizer was modified to include an additional tokenization of the strings whereby the material between the XML mark-up was treated as a single token with a special morphological tag on it, as in (21a). The underscore represents the literal space occurring in the middle of the named-entity. As a fall back mechanism, the tokenization that the string would have received without that mark-up is also produced, as in (21b).

(21) a. General_Mills TB +NamedEntity TB bought TB it TB . TB

b. General TB Mills TB bought TB it TB . TB

The morphological analyzer was modified to allow the additional morphological indicator +NamedEntity to pass through. A lexical entry was added for that indicator and the grammar was changed to allow it to follow nouns. The grammar first tries to parse only with the named enity version. If this parse fails, then a second pass is tried without any markup; as with the POS tagging fall-back mechanism, this second pass grammar is controlled by OT marks (Frank et al., 2001). Since the named entity finder recognition is quite good for proper names, this second pass is rarely needed, thereby increasing both speed and accuracy of the grammar.

[8] In their initial experiments, (Kaplan and King, 2003) used deterministic, gold standard POS tagging derived from the Penn Treebank. However, the POS tag filter can also allows for alternative POS tags for a given lexical item, to prevent early pruning when the tagger is not certain.

4 Conclusions

We have described a language-processing architecture that carefully integrates finite-state techniques for tokenization and morphological analysis with XLE's algorithms for parsing and generation with LFG grammars. This architecture has important theoretical consequences, as it allows many generalizations about the properties of words and strings to be factored out of the specification of more complicated syntactic patterns. It also offers significant practical advantages, since it allows the information included in independently developed, large-scale lexical resources to be used directly as a component of grammar. This avoids the cost of redundant specification and helps to boot-strap broad-coverage capabilities in a cost-effective way. The ParGram grammars have particularly benefited from XLE's facilities for integrating finite-state technology with deep LFG grammars, allowing for the development of truly broad-coverage deep grammars (for coverage and accuracy evaluation statistics, see (Riezler et al., 2002) on the English grammar, (Masuichi et al., 2003) on the Japanese grammar, and (Kaplan et al., 2004) for a comparison of the English grammar to shallow parsers).

References

M. Asahara and Y. Matsumoto. 2000. Extended models and tools for high-performance part-of-speech tagger. In *Proceedings of COLING*, pages 21–27.

K. Beesley and L. Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

M. Butt, T.H. King, M.-E. Niño, and F. Segond. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

M. Butt, H. Dyvik, T.H. King, H. Masuichi, and C. Rohrer. 2002. The Parallel Grammar project. In *COLING 2002: Workshop on Grammar Engineering and Evaluation*, pages 1–7.

M. Dalrymple. 2001. *Lexical Functional Grammar*. Academic Press.

A. Frank, T.H. King, J. Kuhn, and J.T. Maxwell. 2001. Optimality theory style constraint ranking in large-scale LFG grammars. In P. Sells, editor, *Formal and Empirical Issues in Optimality Theoretic Syntax*, pages 367–397. CSLI Publications.

R.M. Kaplan and J. Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. The MIT Press.

R.M. Kaplan and M. Kay. 1981. Phonological rules and fi nite state transducers. Presented at the annual meeting of the LSA.

R.M. Kaplan and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

R.M. Kaplan and T.H. King. 2003. Low-level mark-up and large-scale LFG grammar processing. In *Proceedings of the LFG03 Conference*. CSLI On-line Publications.

R.M. Kaplan and J.T. Maxwell. 1996. LFG grammar writers workbench. ftp://ftp.parc.xerox.com/pub/lfg.

R.M. Kaplan and P. Newman. 1997. Lexical resource reconciliation in the Xerox Linguistic Environment. In *Computational environments for grammar development and linguistic engineering*, pages 54–61.

R.M. Kaplan, T.H. King, and J.T. Maxwell. 2002. Adapting existing grammars: The XLE approach. In *COLING 2002: Workshop on Grammar Engineering and Evaluation*, pages 29–35.

R.M. Kaplan, S. Riezler, T.H. King, J.T. Maxwell, A. Vasserman, and R. Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT-NAACL*.

L. Karttunen, R.M. Kaplan, and A. Zaenen. 1992. Two level morphology with composition. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 141–148.

H. Masuichi, T. Ohkuma, H. Yoshimura, and Y. Harada. 2003. Japanese parser on the basis of the Lexical-Functional Grammar formalism and its evaluation. In *Proceedings of The 17th Pacific Asia Conference on Language, Information and Computation (PACLIC17)*, pages 298–309.

J.T. Maxwell and R.M. Kaplan. 1989. An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27.

S. Riezler, T.H. King, R.M. Kaplan, R. Crouch, J.T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL*.

Document filtering and ranking using syntax and statistics for open domain question answering

Milen Kouylekov[1,2], Hristo Tanev[2]
Department of Information and Communication Technology, University of Trento[1]
Via Sommarive 14, 38050 (TN), Italy
ITC-irst, Centro per la Ricerca Scientifica e Tecnologica [2]
Via Sommarive 18, 38050 (TN), Italy
{kouylekov, tanev}@itc.it

Abstract

This paper presents a strategy for a syntax based ranking of documents specifically oriented to Question Answering (QA). This strategy should limit the number of documents, processed by an answer extraction module of an syntax oriented QA system. Several measures for statistical scoring of expressions are presented and evaluated on 400 factoid questions from the TREC-12 competition. We prove that syntax based document filtering can outperform classical inverse document frequency approaches *(idf)*.

1 Introduction

Open domain Question Answering (QA) systems search for answers to a natural language question either on the Web or in a local document collection. Different techniques, varying from surface patterns [Subbotin2001] to deep semantic analysis [Zajac2001], are used to extract the text fragments containing candidate answers. One of the main challenges in QA and Information Retrieval is the potential mismatch between the expressions in questions and the expressions in texts. Answers may occur in text passages with low similarity with respect to the question. Passages telling facts may use different syntactic constructions, sometimes are spread over more than one sentence, may reflect opinions and personal attitudes, and often use ellipsis and anaphora. Therefore, it is very important to evaluate how much a document preserves the meaning of the question, or in other words, how relevant is the document according to the query that the question presents. Our approach makes use of invariant syntactic fragments from the question.

The goal of QA is to find a text passage which answers to a question posed by the user. A semantic relation between the question, on the one hand, and the answer and its context, on the other hand, should be found by a QA system. Although syntactic and lexical variations and discourse phenomena like anaphora and ellipsis can make this semantic relation difficult to detect, often syntactic constructions and lexical elements from the question remain invariant in the context of the answer. A QA systems can use these invariant constructions as a clue to the position of the right answer.

For example, consider the following question from TREC-11 QA track:

$$Question: George\ Bush\ purchased\ a\ small\ interest,\ in\ which\ baseball\ team? \quad (1)$$

The matched words in two documents extracted from the TREC-11 (TREC-12) collection *AQUAINT* are:

1. “Bush”, “purchased” and “team”
2. “George”, “Bush” and “baseball”

The matched words in these documents don’t give an indication on which of them contains the right answer to the question. In the first document the word *Bush* is a common English name, and the fact that the person it represents *purchased* something doesn’t provide us with enough information to conclude that the answer is contained in the document. The word *team* doesn’t help us to reach conclusion, because it is likely to be replaced in the document by the answer. In the second document we have some certainty about the person about which the document is speaking. Still we need additional information in order to be certain, that in this document *George* is the first name of *Bush* and not a person, that occurs closely to *Bush*. The word *baseball* determines in some way the domain of the document. But still this document will not be a relevant document if *Bush* is the name of a baseball player.

A general architecture of an question answering system is the following:

1. Question Type identification.
2. Document Retrieval.
3. Answer Extraction.
4. Answer Validation.

The basic scenario of an Trec QA system is to: analyze the question; according to this analysis to extract relevant documents from the TREC collection; extract the answer from the retrieved documents using deep or shallow syntactic analysis; test the answer relevance with some kind of statistical measure.

Document retrieval is still an open problem for state-of-the-art systems in open domain QA [Monz2001]. As the answer extraction modules are handling more complex questions, they tend to integrate complex linguistic analysis techniques. The top perfroming system from the last years in the TREC QA track - the LCC System [Moldovan2002] uses deep syntactic parsing and representation in Logical form for answer extraction. Deep syntactic analysis is used also by the Shapaqa system [Buchholz2001] and DIMAP-QA system [Litkowski2001]. Unfortunately, this tendency brings problems with processing speed. Advanced anaphora resolution combined with some semantic resolution and rhetorical parsing can increase the processing speed dramatically. Using a good document filter, that limits the number of processed documents is crucial for the performance of a QA system.

There are several approaches to the problem of filtering documents using syntax. Most of them are related to filtering documents from search engine retrieval systems. [Chandrasekar and Srinivas1997] proposes a scheme for filtering documents that are the output of a search engine query by matching words, their part of speech and elementary trees (NP, PP phrases). But most of the approaches in document filtering are based on matching terms and giving them an appropriate weight. This paper presents a strategy for document filtering, oriented to syntax based answer extraction.

This strategy relies on the following steps:

1. Matching expressions (dependency subtrees) of the question in a candidate document.
2. Scoring the matched expressions with statistical measures based on the words and the relations between words in the expressions.
3. Sorting the documents according to the sum of the scores of the expressions matched in them.

The paper is organized as follows. Section 2 presents the algorithm for matching expressions from the question in the retrieved documents. Section 3 describes several schemes for scoring the matched expressions. Section 4 discusses the efficiency of the different scoring schemes on 400 questions from the TREC-12 competition.

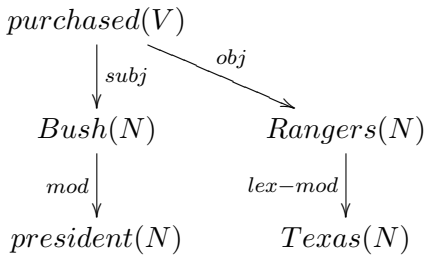
2 Matching expressions from the question in a document

This section presents a scheme for matching expressions of the question in the retrieved documents. We define an expression from the question (Q(expression)) as a subtree of the dependency tree of the question. This subtree can include several words connected with dependency relations. The matching of such an expression in a document expression is a subtree of one of the dependency trees of the parsed document.

As an example, the expression - fragment from the dependency tree of 1:

$$purchased(V) \xrightarrow{subj} Bush(N)$$

matches in the follwing syntactic fragment:



The first dependency tree representing the phrase *Bush purchased*, matches as a sub-tree of the second dependency tree which represents the phrase *president Bush bought Texas Rangers*.

Matching of expressions in a document is defined with the following recursive definition:

- A Q(expression) of *one* word can be matched in a document, if the same word exists in the document, and the two words share the same part of speech assigned (to them) by the parser.
- An Q(expression) of *two* words a_1 and a_2 can be matched in a document, if :
  1. a_1' matches a_1 and a_2' matches a_2.
  2. Exists a relation *R* between a_1 and a_2 in the dependency tree of the question.
  3. Exists a relation *R'* between words a_1' and a_2' in the dependency tree of a sentence in the document.

4. Relation *R* is equal to relation *R′*

- An expression of *n* words a_1, a_2 ... a_n from the question is matched by the words a'_1, a'_2 ... a'_n if:

    1. The Q(expression) a_1, a_2 ... a_{n-1} is matched in the document by the words a'_1, a'_2 ... a'_{n-1}.
    2. Word a_n is matched in the document by word a'_n.
    3. Exists a word a_k such as: exists a relation *R* between words a_k and a_n in the dependency tree of the question; exists a relation *R′* between words a'_k and a'_n in the dependency tree of a sentence in the document; relation *R* is equal to relation *R′*.

We define *matched expression* as a set of words that match an expression of words from the question.

An Q(expression) that can be matched in the document is a subtree of the dependency tree of the question. Its corresponding M(expression) in the document is a subtree of one of the dependency trees of the parsed document. One expression of the question can be matched by several expressions in the same document.

We define an expression of *n* words a_1, a_2 ... a_n as the *largest possible* expression that can be matched in a document, if it is matched in the document, and the union of all words of the expression with any other word from the question can not be matched in the document.

3 Scoring schemes for matched expressions

In this section several schemes for scoring matched expressions in a document are presented. We define the score of the document as the sum of the scores of the expressions matched in this document.We define the score of a matched expression as a function of the words in the expression and the dependency relations in which they are.

The standard document retrieval task uses the *term frequency* measure - the number of occurrences of a term in the document. However this measure has a limited use in the information retrieval for open domain QA as the answer to a question can occur with the question terms in only one sentence of the document. Therefore, even if the set of matched expressions for a document contains sub-expressions of the expressions that are in the set or duplicates they will not be considered in evaluating the score of a the document.

In this section we present 13 ad-hoc measures based on different formulas for calculating a score of a dependency tree (the matched expression). To select the measures we took into account the presumption that we should give higher score to expressions rarely found in the document collection, as they are likely to express more important information about the semantics of the question.

The importance of a word is defined through a measure commonly used, in Information Retrieval, namely *the inverse document frequency (idf)*. If *N* is the number of documents in the collection and N_{word} is the number of documents of the collection that contain the word then the *idf* is given by:

$$idf(word) = \log \frac{N}{N_{word}} \tag{2}$$

We define the *idf* of a matching expression with the same formula. If N_e is the number of documents in which the expression *e* can be matched, the *idf* of *e* is given by:

$$idf(e) = \log \frac{N}{N_e} \tag{3}$$

It is impractical to parse all the documents that contain any of the words of a question in a document collection, as it is a computationally expensive process. So we define the number of documents that contain an expression as the documents that contain the words from the expression in a small distance from each other.

In all formulas *idf($w_i w_j$)* stands for the sub-expression formed by words w_i and w_j having a dependency relation between them. According to this the two formulas:

$$sumarcs(e) = \sum_{j,i=1}^{n} idf(w_i w_j) \tag{4}$$

$$prodarcs(e) = \prod_{j,i=1}^{n} idf(w_i w_j) \tag{5}$$

stand for the sum and the product of the *idfs* of all sub-expressions of two words, connected with any relation.

In our evaluation as a baseline measure is used the sum of the *idfs* of all words of the expression.

Measure 0 (baseline):

$$score(e) = \sum_{i=1}^{n} idf(w_i) \tag{6}$$

We will present the rest of the measures organized in groups. The first group is a group of measures that score the expression based on *idf* of all words of the expression.

Measure 1.1:

$$score(e) = idf(e) \tag{7}$$

Measure 1.2:

$$score(e) = idf(e) * \prod_{i=1}^{n} idf(w_i) \tag{8}$$

Measure 1.3:

$$score(e) = idf(e) + \sum_{i=1}^{n} idf(w_i) \tag{9}$$

The score of expression for the measures of the second group is created from the sum or the product of the subexpressions of couples of words, that are in relation and the rest of the words in the expression.

Measure 2.1:

$$score(e) = \prod_{i=1}^{n} idf(w_i) * prodarcs(e) \tag{10}$$

Measure 2.2:

$$score(e) = \sum_{i=1}^{n} idf(w_i) + sumarcs(e) \tag{11}$$

The third group is related to measures that give higher score to expressions with bigger number of words, as they don't take into account the *idf* of the words for expressions bigger than one word. In the following formulas *nw(e)* stands for the number of words in the expression *e*.

Measure 3.1:

$$score(e) = \begin{cases} 0 & if\ nw(e) = 1 \\ prodarcs(e) & otherwise. \end{cases} \tag{12}$$

Measure 3.2:

$$score(e) = \begin{cases} 0 & if\ nw(e) = 1 \\ sumarcs(e) & otherwise. \end{cases} \tag{13}$$

Measure 3.3:

$$score(e) = \begin{cases} 1 & if\ nw(e) = 1 \\ prodarcs(e) & otherwise. \end{cases} \tag{14}$$

Measure 3.4:

$$score(e) = \begin{cases} 1 & if\ nw(e) = 1 \\ sumarcs(e) & otherwise. \end{cases} \tag{15}$$

Measure 3.5:

$$score(e) = \begin{cases} idf(e) & if nw(e) = 1 \\ prodarcs(e) & otherwise. \end{cases} \tag{16}$$

Measure 3.6:

$$score(e) = \begin{cases} idf(e) & if nw(e) = 1 \\ sumarcs(e) & otherwise. \end{cases} \tag{17}$$

The last group consists of a single measure. It is a modified version of the mutual information measure commonly used to estimate the association strength between two words [Church and Hanks1989][1].

[1]The mutual information between two events x and y is given by the formula:

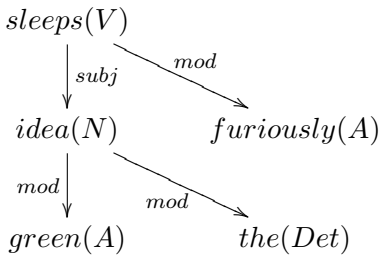
$$mi(x, y) = \log \frac{P(x, y)}{P(x)P(y)} \tag{18}$$

We define the mutual information for an expression in the following way: for an expression *e* of words w_1,...,w_n

Measure 4:

$$mi(e) = \begin{cases} \log \frac{P(w_0,...,w_n)}{P(w_0)...P(w_n)} & n > 1 \\ 0 & n = 1 \end{cases} \quad (19)$$

For example, let the matched expression be *the green idea sleeps furiously*. The dependency tree of this expression is:



The score of this expression for measure 3.6 is:

score(e) = idf(the, idea) + idf(green, idea) + idf(idea, sleeps) + idf(sleeps,furiously)

We selected these measures taking into account the different characteristics of the matched expressions. When choosing the measures we didn't have presumption about their efficiency.

4 Effectiveness of the scoring measures

To evaluate the effectiveness of the measures discussed in the previous section we used the documents retrieved by the Information Retrieval module of the *Diogene system* [Magnini et al.2001] for 400 factoid questions from the TREC-12 competition. For each question, we looked for the answer reported by the TREC assessors in the top-ranked documents according to each measure. We processed each question in the following way:

1. We parsed the question into a set of dependency trees.
2. We extracted documents relative to the words of the question from the official collection *AQUAINT* used in TREC and parsed them into sets of dependency trees.
3. For each document we created the set of matched expressions according to the match algorithm presented in section 2 and scored them according to the evaluation measures presented in 3.
4. We calculated the position of each document according to the ranks defined by each measure.
5. We calculated the *reciprocal rank (*RR^j_{1i}*)* [2]of the first and *(*RR^j_{2i}*)* of the last document, that contains the answer according to the ranks defined by each measure.

[2]in this formula *i* is the number of the question and *j* is the number of the measure

The *reciprocal rank (RR)* is the measure used in QA track at TREC-11 for scoring the position of the right answer in an ordered set of answers. The reciprocal rank RR^j_{1i} for the first document that contain the answer is defined by:

$$RR(pos) = \begin{cases} 0 & if\ no\ answer \\ \frac{1}{pos} & if\ have\ answer \end{cases} \tag{20}$$

where *pos* is the position of this document in the sorted according to a measure set of retrieved documents. We evaluate the RR^j_{2i} of the last document containing the answer using $n - pos$ as the position of this document.

The final score of a measure is the *mean reciprocal rank* MRR^j_1 of the reciprocal ranks RR^j_{1i} and the *mean reciprocal rank* MRR^j_2 of the reciprocal ranks RR^j_{2i}. The *mean reciprocal rank* is defined by:

$$MRR(e) = \frac{\sum_{i=1}^{n}(RR_i)}{N} \tag{21}$$

The mean reciprocal rank MRR^j_1[3] will give us an evaluation of how well a measure pushes the documents, containing the answer ahead in the ranks. The mean reciprocal rank MRR^j_2 will evaluate the density of the answers in the documents ranked by this measure.

The best measures should receive higher MRR_1 and lower MRR_2.

For parsing we used the *Minipar* parser. *Minipar* is a principle-based English parser [Lin1998]. We decided to use this parser for its processing speed and high precision and recall on newspaper corpora.

For the estimation of *idf* of an expression we used *AltaVista* as a search engine of the on-line corpus that is the *Internet*, because this is one of the search engines with largest indexes. Also, *AltaVista* allows the user to make queries using the operator NEAR which plays an important role in our approach for calculating *idf* of an expression. We decided to use the web as a corpus, because of the high number of indexed documents in comparison to the *AQUAINT* collection. This gave us the possibility to identify more exactly the rare words and phrases from the question.

Results in Table 1 show an equality of the density (MRR_2) for different measures, which means that often only one word of the question is found in the document that contains the answer. The results show that most of the measures perform better then the baseline in bringing answers higher in the ranks.

Significant influence on the results is given by the fact that most of the matched expressions contained no more then 2-3 words. This is explained by the fact that the keywords of the questions are rarely expressed in the same way in the documents as in the question. Taking this into account, we can make several conclusions about the results:

- The measures that multiply the *idf* of the words of a matched expression have *close* scores with respect to the corresponding measures that sum the *idfs*.
- The measures of group 1, which take into account all the words from the expression are receiving a vary bad MMR_1 score.

[3] in this formula *j* is the number of the measure

Measure	MMR_1	MMR_2
0 (baseline)	0.2409	0.0487
1.1	0.2272	0.0471
1.2	0.2494	0.0473
1.3	0.2339	0.0485
2.1	0.2680	0.0470
2.2	0.2678	0.0471
3.1	0.2332	0.0907
3.2	0.2327	0.0457
3.3	0.2782	0.0457
3.4 (best performing)	0.2803	0.0457
3.5	0.2416	0.0466
3.6	0.2327	0.0407
4	0.2501	0.0772

Table 1: Results of measure evaluation

- The measure 4, which evaluates *PMI* of an expression, is better than the baseline in moving documents with answers in the top positions. However it has problems with the density caused by the fact that it does not take into account single words.

Interesting are the measures of group 3. The measures in this group, although using the same measure for scoring expressions with more than one word, have a significant difference in the results. The measures 3.1 and 3.2 receive low scores, because of the small average number of words in the matched expressions. These two measures do not take into account single words. The fact that measures 3.3 and 3.4 have better scores than measures 3.5 and 3.6 is due to the fact that they do not score very high single words.

5 Conclusions and future work

This paper presented a syntax oriented document filtering strategy, that can present a stable base for a syntax oriented QA system. This strategy is useful not only to filter the candidate documents, but also to identify entities from the question in the candidate paragraphs. Although the method proposes deep syntactic analysis for determining relations between it can process relatively fast if parsing is used only when necessary. Also some shallow techniques like chunkers and shallow parsers can be used as the size of the matched structures remains relatively small.

Our experiments show that the measures that score higher the expressions with more than one word and give some but not high score to the single words expression perform slightly better than the baseline model. This shows that the presence of rare single word expressions which are scored higher by the baseline model is a good indication for the presence of the answer in the document, but can be outperformed by measures that take in to account the relations between the matched words.

The 4% difference between the baseline and the top-performing measure 3.4 can be extended by additionally taking into account several techniques for improving the matching. A significant improvement of the matching will be achieved by introducing *paraphrases* [Spark Jones and Tait1984], *variants* [Fabre and Jacquemin2000] or *inference rules* [Lin2001]. Using *paraphrases* will improve significantly the size of the matched expressions which will boost significantly the measures that score

higher expressions than single words. Adding a module for anaphora resolution, will also improve significantly the size of matched expression.

An improvement of the result can also be obtained by some statistical evaluation of the types of relations between the words of the matched expression. For instance, the subject relation which is weaker and is often matched can be scored lower than the more stable object relation. It will be also interesting to investigate the difference in the performance of the different measures on different question types, since some of the scoring schemes can perform much better on certain types of questions.

References

[Buchholz2001] Buchholz, S. 2001. Using Grammatical Relations, Answer Frequencies and the World Wide Web for TREC Question Answering. *In Proceeding of the TREC-10 Conference 2001*, pages 496-503.

[Chandrasekar and Srinivas1997] Chandrasekar, R. and Srinivas, B. 1997. Using syntactic information in document filtering: A comparative study of part-of-speech tagging and supertagging. *In Proceedings of RIAO-97*, pages 531-545. Montreal, Canada.

[Church and Hanks1989] Church, K. and Hanks, P. 1989. Word Association Norms, Mutual Information, and Lexicography. *In Proceedings of ACL-89*, pages 76-83. Vancouver, Canada.

[Fabre and Jacquemin2000] Fabre, C. and Jacquemin, C. 2000. Boosting Variant Recognition with Light Semantics. *In Proceedings of COLING-2000*. Sarrebrcken, Germany.

[Lin1998] Lin, D. 1998. Dependency-based evaluation of MINIPAR. *In Proceedings of the Workshop on Evaluation of Parsing Systems at LREC-98*. Granada, Spain.

[Lin2001] Lin, D. and Pantel, P. 2001. Discovery of inference rules for Question Answering. *Natural Language Engineering, 7(4)*, pages 343-360.

[Litkowski2001] Litkowski, K. 2001. Discovery of inference rules for Question Answering. *In Proceeding of the TREC-10 Conference 2001*, pages 251-260.

[Magnini et al.2001] Magnini B., Negri M., Prevete R. and Tanev, H. 2001. Multilingual Question/Answering: the Diogene System. *In Proceeding of the TREC-10 Conference 2001*.

[Moldovan2002] Moldovan, D., Harabagio, S., Girju, R., Morarescu, P., Lacatsu, F., Novischi, A., et al. (2003). 2002-2003. LCC Tools for Question Answering. *NIST Special Publication: SP 500-251 The Eleventh Text Retrieval Conference (TREC 2002)*.

[Monz2001] Monz, C. 2003. Document Retrieval in the Context of Question Answering. *In: F. Sebastiani (Ed.) Proceedings of the 25th European Conference on Information Retrieval Research (ECIR-03)*

[Spark Jones and Tait1984] Spark Jones, k. and Tait, J. 1984. Automatic Search Term Variant Generation. *Journal of Documentation, 40(1),* pages 50-66.

[Subbotin2001] Subbotin, M. 2001. Patterns of Potential Answer Expressions as Clues to the Right Answers. *In Proceeding of the TREC-10 Conference 2001*, pages 175-182.

[Zajac2001] Zajac, R. 2001. Towards Ontological Question Answering. *In Proceedings of the ACL-2001 Workshop on Open-Domain Question Answering* Toulouse, France.

Using XSLT for the Integration of Deep and Shallow Natural Language Processing Components

Ulrich Schäfer
Language Technology Lab,
German Research Center for Artificial Intelligence (DFKI),
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
email: ulrich.schaefer@dfki.de

Abstract

WHITEBOARD is a hybrid XML-based architecture that integrates deep and shallow natural language processing components. The online system consists of a fast HPSG parser that utilizes tokenization, PoS, morphology, lexical, named entity, phrase chunk and (for German) topological sentence field analyses from shallow components. This integration increases robustness, directs the search space and hence reduces processing time of the deep parser. In this paper, we focus on one of the central integration facilities, the XSLT-based WHITEBOARD Annotation Transformer (WHAT), report on the benefits of XSLT-based NLP component integration, and present examples of XSL transformation of shallow and deep annotations used in the integrated architecture. Furthermore, we report on a recent application of XSL transformation for the conversion of XML-encoded typed feature structures representation in the context of the DEEPTHOUGHT project where deep-shallow integration is performed on the basis of Robust Minimal Recursion Semantics (RMRS).

1 Introduction

Deep processing (DNLP) systems try to apply as much linguistic knowledge as possible during the analysis of sentences and result in a uniformly represented collection of the knowledge that contributed to the analysis. The result often consists of many possible analyses per sentence reflecting the uncertainty which of the possible readings was intended - or no answer at all if the linguistic knowledge was contradictory or insufficient with respect to the input sentence.

Shallow processing (SNLP) systems do not attempt to achieve such an exhaustive linguistic analysis. They are desigend for specific tasks ignoring many details in input and linguistic framework. Utilizing rule-based (*e.g.* finite-state) or statistics-based approaches, they are in general much faster than DNLP. Due to the lack of efficiency and robustness of DNLP systems, the trend in application-oriented language processing system development in the last years was to improve SNLP systems. They are now capable of analyzing Megabytes of texts within seconds, but precision and quality barriers are so obvious (especially on domains the systems were not designed for or trained on) that a need for 'deeper' systems re-emerged. Moreover, semantics construction from an input sentence is quite poor and erroneous in typical shallow systems.

However, development of DNLP made advances during the last few years, especially in the field of efficiency [Callmeier, 2000; Uszkoreit, 2002]. A promising solution to improve quality of natural language processing is the combination of deep and shallow technologies. Deep processing benefits

from specialized and fast shallow analysis results, shallow processing becomes 'deeper' using at least partial results from DNLP.

Many natural language processing applications could benefit from the synergy of the combination of deep and shallow, *e.g.* advanced information extraction, question answering, or grammar checking systems.

This paper is structured as follows. In section 2, we motivate and introduce the XSLT-based WHITEBOARD annotation transformer. In section 3, we describe the deep-shallow integration. In section 4, we focus on transformation of feature structure XML transformation. Section 5 describes related work, we conclude and give an outlook to future work in section 6.

2 Annotation Access and Transformation through XSLT

2.1 Motivation

The WHITEBOARD architecture integrates shallow and deep natural language processing components. Both online and offline coupling of existing software modules is supported, *i.e.*, the architecture provides direct access to standoff XML annotation as well as programming interfaces. Applications communicate with the components through programming interfaces. A multi-layer chart holds the linguistic processing results in the online system memory while XML annotations can be accessed online as well as offline. The different paradigms of DNLP and SNLP are preserved throughout the architecture, *e.g.* there is a shallow and a deep programming interface.

WHAM (WHITEBOARD Annotation Machine) offers programming interfaces which are not simply DOM interfaces isomorphic to the XML markup they are based on, but hierarchically defined classes. E.g., a fast index-sequential storage and retrieval mechanism based on XML is encapsulated through the shallow programming interface. However, while the typed feature structure-based programming interface to deep components became stable early, it turned out that the initial, XML interface was too inflexible when new, mainly shallow, components with new DTDs had to be integrated. Therefore, a more flexible approach had to be devised.

The solution is an XSLT-based infrastructure for NLP components that provides flexible access to standoff XML annotations produced by the components. XSLT [Clark, 1999] is a W3C standard language for the transformation of XML documents. Input of an XSL transformation must be XML, while output can be any syntax (*e.g.* XML, text, HTML, RTF, or even programming language source code, *etc*). The power of XSLT mainly comes from its sublanguage XPath [Clark and DeRose, 1999], which supports access to XML structure, elements, attributes and text through concise path expressions. An XSL stylesheet consists of templates with XPath expressions that must match the input document in order to be executed. The order in which templates are called is by default top-down, left to right, but can be modified, augmented, or suppressed through loops, conditionals, and recursive call of (named) templates.

2.2 WHAT, the WHITEBOARD Annotation Transformer

WHAT is built on top of a standard XSL transformation engine. It provides uniform access to standoff annotation through queries that can either be used from non-XML aware components to get access to information stored in the annotation (V and N-queries), or to transform (modify, enrich, merge) XML annotation documents (D-queries). While WHAT is written in a programming language such as Java

or C, the XSL query templates that are specific for a standoff DTD of a component's XML output are independent of that programming language, *i.e.*, they must only be written once for a new component and are collected in a so-called template library.

Based on an input XML document (or DOM object), a WHAT query that consists of component name, query name, and query-specific parameters such as an index or identifier is looked up in the XSLT template library for the specified component, an XSLT stylesheet is returned and applied to the XML document by the XSLT processor. The result of stylesheet application is then returned as the answer to the WHAT query. There are basically three kinds of results: (1) strings (including non-XML output), (2) references to nodes in the XML input document (IDs), (3) XML documents.

In other words, if we formulate queries as functions, we get the following query signatures, with C being the component, D denoting an XML document, P^* a (possibly empty) sequence of parameters, S^* a sequence of strings, and N^* a sequence of nodes.

- **V-queries**. getValue: $C \times D \times P^* \longmapsto S^*$
  V-queries return string values from XML attribute values or text. The simplest case is a single XPath lookup, *e.g.* of the gender of a word encoded in a shallow XML annotation.
- **N-queries**. getNodes: $C \times D \times P^* \longmapsto N^*$
  N-queries compute and return lists of node identifiers that can again be used as parameters for subsequent queries, *e.g.* all named entity nodes within a token or character range specified as query parameters.
- **D-queries**. getDocument: $C \times D \times P^* \longmapsto D$
  D-queries return transformed XML documents – this is the classical, general use of XSLT. Complex transformations that modify, enrich or produce (standoff) annotation can be used for many purposes. Examples are (1) conversion from a different XML format, (2) merging of several XML documents into one, (3) auxiliary document modifications, *e.g.* to add unique identifiers to elements, sort elements *etc*, (4) providing interface to NLP applications, (5) visualization and formatting (Thistle, HTML, PDF, ...) (6) complex computations on XML input (turning a query into a kind of NLP component itself).

More details on WHAT and examples of the query types are presented in [Schäfer, 2003].

3 Architecture of the Hybrid Deep-Shallow System

The fully online-integrated hybrid WHITEBOARD architecture consists of the efficient deep HPSG parser PET [Callmeier, 2000] utilizing tokenization, PoS, morphology, lexical, compound, named entity, phrase chunk and topological sentence field analyses from shallow components in a pipelined architecture. This integration significantly increases robustness, directs the search space and reduces processing time of the deep parser, cf. [Crysmann *et al.*, 2002; Frank *et al.*, 2003; Schäfer, 2003] for details and evaluation results.

The simplified diagram in Fig. 1 depicts the components and places where WHAT comes into play in the hybrid integration of deep and shallow processing components (V, N, D denote the WHAT query types, *i.e.* XSLT transformations). Solid boxes depict components that produce annotation, dashed boxes depict the produced annotation (XML markup). Solid-line arrows depict the transformations used in the online integration. Dashed-line arrows indicate possible access to the intermediate annotation that could be accessed from an application (bottom box), *e.g.* the Thistle [Calder, 2000]

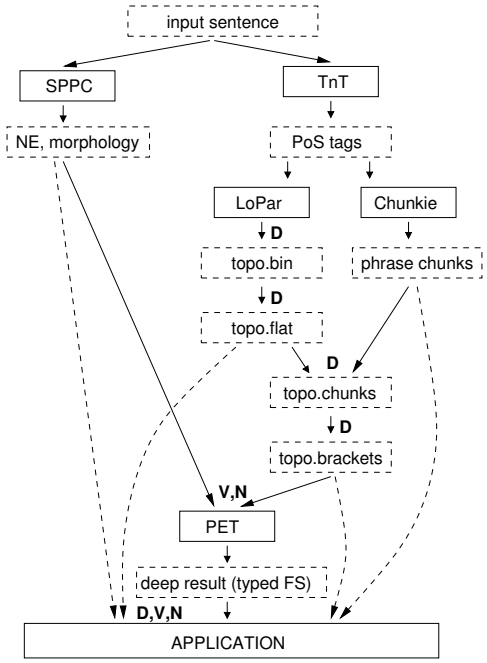


Figure 1: XSLT-based architecture of the hybrid parser.

tree visualizations that show the XML annotation tree structures (Fig. 2 through 4) have been created through WHAT D-queries out of the intermediate topo.* XML trees.

The system takes an input sentence, and runs three shallow systems on it:

- the rule-based shallow SPPC [Piskorski and Neumann, 2000] for named entity recognition, compound analysis for German, and morphology and stemming of words unknown to the HPSG lexicon,
- TnT/Chunkie, a statistics-based shallow PoS tagger and chunker [Skut and Brants, 1998],
- LoPar, a probabilistic context-free parser [Schmid, 2000], which takes PoS-tagged tokens as input, and produces binary tree representations of sentence fields, *e.g.* topo.bin in Fig. 2. For a motivation for binary vs. flat trees cf. [Becker and Frank, 2002].

The results of these components are three standoff annotations of the input sentence. Named entity, compound, morphological and stem information from SPPC is used by the deep parser to initialize the chart with prototypical feature structures that are filled with shallow information through V-queries for words unknown to the HPSG lexicon and for named entities. In addition, preference information on part-of-speech is used for prioritization of the deep parser. For details, cf. [Crysmann *et al.*, 2002]. PoS tagging from TnT is used as input for Chunkie to produce chunking and as input for the shallow topological PCFG parser [Frank *et al.*, 2003].

The examples in Fig. 2 through 5 show the analyses of the German sentence

> *Untergebracht war die Garnison in den beiden Wachlokalen Hauptwache und Konstablerwache. (Located was the garrison at the two guard houses, the main guard house and the Constabler guard house.)*

with a fronted verb in topic position, which the topological parser identifies correctly. This macro-sentential information can be used to direct the deep parser's search space towards the correct (and

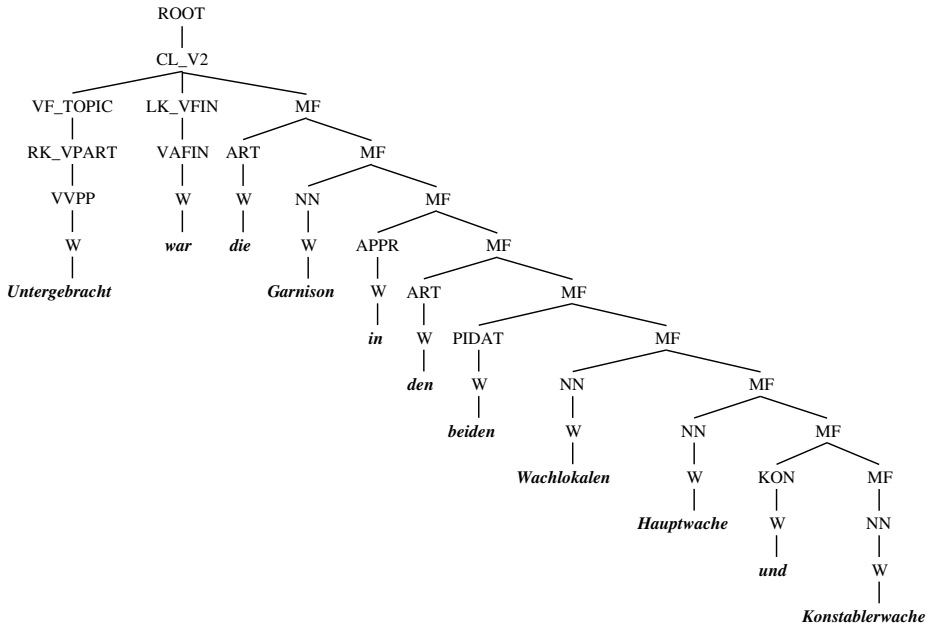


Figure 2: A binary tree as the result of the topological parser (topo.bin).

rather infrequent) construction, avoiding alternative exploration of the search space.

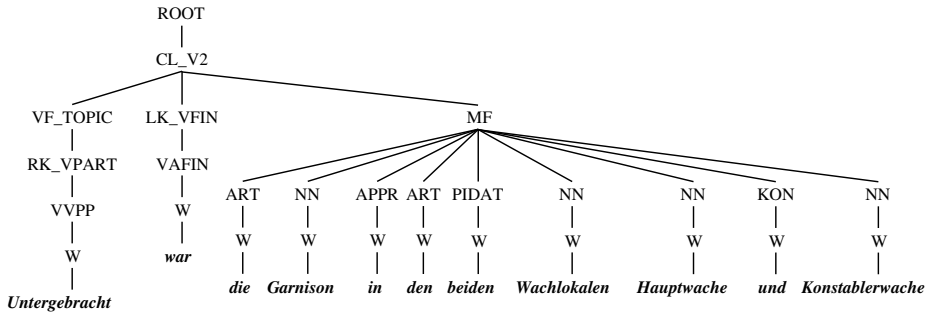


Figure 3: The topological tree after flattening (topo.flat).

In order to extract this type of global constituent-based information, a sequence of D-queries is applied to flatten the binary topological trees that are output by LoPar (result is topo.flat, Fig. 3) and merge the tree with shallow chunk information from Chunkie (topo.chunks, Fig. 4). In a next step, we apply the main D-query which computes bracket information for the deep parser from the merged topological tree and chunks (topo.brackets, Fig. 5).

The bracket information is used to prioritize chart elements of the deep parser that match the constituent boundaries computed by the shallow parser and chunker. The stylesheet directly generates the names of the appropriate HPSG types (value of the rule attributes in Fig. 5)[1].

Finally, the deep parser PET [Callmeier, 2000], modified as described in [Frank *et al.*, 2003], is started with a chart initialized using V-queries to access lexical (morphology, stemming, compounds, PoS preferences) and named entity information gathered from SPPC. The computed bracket information (topo.brackets; Fig. 5) is accessed through WHAT V and N-queries during parsing in order to prioritize

[1]The tree visualizations in Fig. 3 through 5 themselves have been generated through a generic WHAT D-query transforming the XML document into a Thistle tree (arbora DTD; [Calder, 2000]).

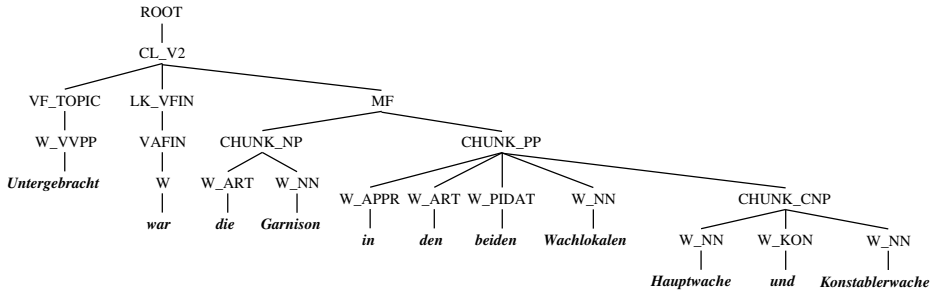


Figure 4: The topological tree merged with chunks (topo.chunks).

```
<TOPO2HPSG>
  <MAPC type="chunk_np+det" rule="chunk" left="W2" right="W3"/>
  <MAPC type="chunk_pp" rule="chunk" left="W4" right="W10"/>
  <MAPC type="v2_cp" rule="vfronted" left="W0" right="W10"/>
  <MAPC type="vfronted_vfin-rk" rule="vfronted" left="W1" right="W1"/>
  <MAPC type="vfronted_vfin+vp-rk" rule="vfronted" left="W1" right="W10"/>
  <MAPC type="v2_vf" rule="vfronted" left="W0" right="W0"/>
  <MAPC type="v2_vfin_pvp-rk" rule="vfronted" left="W1" right="W1"/>
</TOPO2HPSG>
```

Figure 5: The extracted brackets (topo.brackets).

constituent analyses motivated by the topological parser and additional syntactic information. Again, WHAT abstraction facilitates exchange of the shallow input components of PET, *e.g.* it would be possible to exchange some of the used components without rewriting of the deep parser's code.

The result of deep parsing including the constructed semantics representation of the analysed sentence can be accessed through the chart interface of PET, or through XSLT transformation – which leads to the next topic, WHAT access to and transformation of typed feature structure markup.

4 Accessing and Transforming Typed Feature Structure Markup

In the sections so far, we have shown examples for shallow XML annotation. But annotation access could also include deep analysis results. In this section, we turn to deep XML annotation. Typed feature structures provide a powerful, universal representation for deep linguistic knowledge.

While it is in general inefficient to use XML markup to represent typed feature structures during processing (*e.g.* for unification, subsumption operations), there are several applications that may benefit from a standardized system-independent XML markup of typed feature structures, *e.g.* as exchange format for (1) deep NLP component results (*e.g.* parser chart), (2) grammar definitions like in the SProUT system [Drozdzynski *et al.*, 2004], (3) feature structure renderers or editors like in SProUT or Thistle [Calder, 2000], (4) feature structure 'tree banks' of analyzed corpora.

We adopt a DAG-like XML markup for embedded typed feature structures originally developed by the Text Encoding Initiative (TEI) which is compact and widely accepted (cf. [Lee *et al.*, 2004]). An in-depth justification for the naming and structure of the TEI feature structure DTD is presented in [Langendoen and Simons, 1995]. We focus here on the feature structure DTD subset that is able to encode the basic data structures of deep systems such as LKB [Copestake, 2002], PET [Callmeier, 2000], PAGE, or the shallow system SProUT [Drozdzynski *et al.*, 2004] which have a subset of TDL [Krieger and Schäfer, 1994] as their common basic formalism.

```
<MATCHINFO rule="en_city" cstart="3" cend="7">
  <FS type="sprout_rule">
    <F name="OUT">
      <FS type="ne-location">
        <F name="LOCNAME">
          <FS type="&quot;Paris&quot;"/>
        </F>
        <F name="LOCTYPE">
          <FS type="city"/>
        </F>
      </FS>
    </F>
  </FS>
</MATCHINFO>
```

```
-->
```

```
<rmrs cfrom="3" cto="7">
  <label vid="1"/>
  <ep cfrom="3" cto="7">
    <gpred>ne-location</gpred>
    <label vid="2"/>
    <var sort="x" vid="2"/>
  </ep>
  <rarg>
    <label vid="2"/>
    <rargname>CARG</rargname>
    <constant>"Paris"</constant>
  </rarg>
</rmrs>
```

Figure 6: Transforming feature structure XML markup (SProUT) to RMRS (DEEPTHOUGHT).

```
<?xml version="1.0" ?>   <!-- minimal typed feature structure DTD -->
<!ELEMENT FS ( F* ) >
<!ATTLIST FS type  NMTOKEN #IMPLIED
             coref NMTOKEN #IMPLIED >
<!ELEMENT F  ( FS ) >
<!ATTLIST F  name  NMTOKEN #REQUIRED >
```

The FS tag encodes typed feature structure nodes, F encodes features. Atoms are encoded as typed feature structure nodes with empty feature list. An important point is the encoding of coreferences (reentrancies) between feature structure nodes which denote structure sharing.

An application of WHAT access or transformation of deep annotation would be to specifiy a feature path under which a value (type, atom, or complex FS) is to be returned. Because of limited space, we give only a short example here, namely access to output of the shallow SProUT system [Drozdzynski *et al.*, 2004] that produces disjunctions of typed feature structures as output. The `select` expression assigns the value (type) of a feature under the specified attribute path YEAR in the feature structure typed 'point' to an XSLT variable.

```
<xsl:variable name="year" select='FS[@type="point"]/F[@name="YEAR"]/FS/@type'/>
```

The complex stylesheet from which this example is taken, translates SProUT analysis results of named entity recognition encoded as typed feature structures into XML-encoded RMRS structures (Fig. 6) which form the common representation for deep and shallow NLP processing results in the DEEPTHOUGHT architecture [Callmeier *et al.*, 2004; Copestake, 2003]. Deep-shallow integration is then performed on the basis of these RMRS representations. For a list of further applications of XML-based feature structure transformation cf. section 5 in [Lee *et al.*, 2004].

To complete the picture of abstraction through WHAT queries, we can imagine that the same types of query are possible to access, *e.g.* the same morphology information in both shallow and in deep annotation, although their representation within the annotation might be totally different.

5 Related Work

Architectures that combine deep and shallow processing are emerging, but none of the systems described so far cover as much different NLP processing layers as WHITEBOARD. E.g., integration in

[Grover *et al.*, 2002] was limited to tokenization and PoS tagging, in [Daum *et al.*, 2003] to PoS tagging and chunks. DEEPTHOUGHT [Callmeier *et al.*, 2004] is a recent and promising approach that concentrates integration on the level of uniform underspecified semantics representation.

Several XML-based or XML-supporting architectures and tools have been proposed and developed for natural language processing, *e.g.* LT-XML [Brew *et al.*, 2000], RAGS [Cahill *et al.*, 1999] and XCES [Ide and Romary, 2001]. Overviews are also given by [McKelvie *et al.*, 1998; Ide, 2000; Carletta *et al.*, 2002].

As argued in [Thompson and McKelvie, 1997], standoff annotation is a viable solution in order to cope with the combination of multiple overlapping hierarchies and the efficiency problem of XML tree modification for large annotations. We adopt the pragmatic view of [Carletta *et al.*, 2002], who see that computational linguistics greatly benefits from general XMLification, namely by getting for free standards and advanced technologies for storing and manipulating XML annotation, mainly through W3C and various open source projects. The trade-off for this benefit is a representation language somewhat limited with respect to linguistic expressivity.

NiteQL [Evert and Voormann, 2002] can be seen as an extension to XPath within XSLT, it comes with a more concise syntax especially for document structure-related expressions and a focus on timeline support with specialized queries (for speech annotation). The query language in general does not add expressive power to XSLT and the implementation currently only supports Java XSLT engines.

6 Conclusion and Future Work

We have presented WHAT, an open, flexible and powerful infrastructure based on standard XSLT technology for the online and offline combination of natural language processing components, with a focus on, but not limited to, hybrid deep and shallow architectures.

The infrastructure is portable. As the programming language-specific wrapper code is relatively small, the framework can be quickly ported to any programming language that has XSLT support (which holds for most modern programming and scripting languages). XSLT makes the transformation code portable which it could not be when being based on a DOM implementation (which is always programming language-dependent).

The WHAT framework can easily be extended to new NLP components and document DTDs. This has to be done only once for a component or DTD through XSLT query library definitions, and access will be available immediately in all programming languages for which a WHAT implementation exists.

WHAT can be used to perform computations and complex transformations on XML annotation, provide uniform XML annotation access in order to abstract from component-specific namings and DTD structure. WHAT makes it easier to exchange results between components (*e.g.* to give non-XML-aware components access to information encoded in XML annotation), and to define application-specific architectures for online and offline processing of NLP XML annotation.

Due to its flexibility, the infrastructure is well suited for rapid prototyping of hybrid NLP architectures as well as for developing NLP applications, and can be used to both access NLP XML markup from programming languages and to compute or transform it.

Besides the integration within NLP architectures described in this article, the XSLT-based infrastructure (WHAT) could also be used for interfacing applications, *e.g.* to translate to Thistle [Calder, 2000] for visualization of linguistic analyses and back from Thistle in editor mode, *e.g.* for manual, graphical correction of automatically annotated texts for training *etc*.

Because of unstable standardization and implementation status, we did not yet make use of XQuery [Boag *et al.*, 2002]. XQuery is a powerful, SQL-like query language on XML documents with XPath being a subset of XQuery rather than a sublanguage like of XSLT. Because the WHAT framework is open, it is worth considering XQuery as a future extension. Which engine to ask, an XSLT or an XQuery processor, could be coded in each `<query>` element of the template library. Similarly, extension of the current framework to XSLT 2.0 which among other things supports user-definable functions that can be part of XPath expressions, should be straightforward.

It should be possible to combine WHAT with SDL [Krieger, 2003] to declaratively specify XSLT-based NLP architectures (pipelines, loops, parallel transformation) that can be compiled to Java code. Here, WHAT queries can be used within mediators that convert between modules, or as proper modules that perform computations.

The proximity to W3C standards suggests using XSLT also for transformation of NLP results into application-oriented (W3C) markup like VoiceXML or into RDF or OWL for semantic web integration.

7 Acknowledgements

I would like to thank my collegues, especially Anette Frank, Bernd Kiefer, Hans-Ulrich Krieger, Günter Neumann and Melanie Siegel, for cooperation and many discussions. I would also like to thank three anonymous reviewers and the audience at the SEALTS workshop at HLT-NAACL 2003 for helpful comments. This work has been supported by grants from the German Federal Ministry of Education and Research (FKZ 01IW002, 01IWC02). This document was generated partly in the context of the DEEPTHOUGHT project, funded under the Thematic Programme User-friendly Information Society of the 5th Framework Programme of the European Community (Contract No. IST-2001-37836).

References

[Becker and Frank, 2002] Markus Becker and Anette Frank. A Stochastic Topological Parser of German. In *Proceedings of COLING 2002*, pages 71–77, Taipei, Taiwan, 2002.

[Boag *et al.*, 2002] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. *XQuery 1.0: An XML Query Language*. W3C, http://w3c.org/TR/xquery, 2002.

[Brew *et al.*, 2000] Chris Brew, David McKelvie, Richard Tobin, Henry Thompson, and Andrei Mikheev. *The XML Library LT XML. User documentation and reference guide*. LTG, Univ. of Edinburgh, 2000.

[Cahill *et al.*, 1999] Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. Towards a reference architecture for natural language generation systems (the RAGS project). Technical report, HCRC, University of Edinburgh, 1999.

[Calder, 2000] Joe Calder. *Thistle: Diagram Display Engines and Editors*. HCRC, U. of Edinburgh, 2000.

[Callmeier *et al.*, 2004] Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. The DeepThought core architecture framework. In *Proceedings of LREC-2004*, pages 1205–1208, Lisbon, 2004.

[Callmeier, 2000] Ulrich Callmeier. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6 (1):99 – 108, 2000.

[Carletta *et al.*, 2002] Jean Carletta, David McKelvie, Amy Isard, Andreas Mengel, Marion Klein, and Morten Baun Møller. A generic approach to software support for linguistic annotation using XML. In G. Sampson and D. McCarthy, editors, *Readings in Corpus Linguistics*, London and NY, 2002. Continuum International.

[Clark and DeRose, 1999] James Clark and Steve DeRose. *XML Path Language (XPath)*. W3C, http://w3c.org/TR/xpath, 1999.

[Clark, 1999] James Clark. *XSL Transformations (XSLT)*. W3C, http://w3c.org/TR/xslt, 1999.

[Copestake, 2002] Ann Copestake. *Implementing Typed Feature Structure Grammars*. CSLI publications, Stanford, CA, 2002.

[Copestake, 2003] Ann Copestake. Report on the design of RMRS. Technical Report D1.1b, University of Cambridge, Cambridge, UK, 2003.

[Crysmann *et al.*, 2002] Berthold Crysmann, Anette Frank, Bernd Kiefer, Stefan Müller, Jakub Piskorski, Ulrich Schäfer, Melanie Siegel, Hans Uszkoreit, Feiyu Xu, Markus Becker, and Hans-Ulrich Krieger. An Integrated Architecture for Deep and Shallow Processing. In *Proceedings of ACL 2002*, Philadelphia, PA, 2002.

[Daum *et al.*, 2003] M. Daum, K.A. Foth, and W. Menzel. Constraint Based Integration of Deep and Shallow Parsing Techniques. In *Proceedings of EACL 2003*, Budapest, 2003.

[Drozdzynski *et al.*, 2004] Witold Drozdzynski, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. Shallow processing with unification and typed feature structures — foundations and applications. *Künstliche Intelligenz*, 1:17–23, 2004. http://www.kuenstliche-intelligenz.de /archiv/2004_1/sprout-web.pdf.

[Evert and Voormann, 2002] Stefan Evert and Holger Voormann. *NITE Query Language, NITE Project Document*. University of Stuttgart, Stuttgart, 2002.

[Frank *et al.*, 2003] Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schäfer. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of ACL-2003*, pages 104–111, Sapporo, Japan, 2003.

[Grover *et al.*, 2002] Claire Grover, Ewan Klein, Alex Lascarides, and Maria Lapata. XML-based NLP tools for analysing and annotating medical language. In *Proceedings of the Second International Workshop on NLP and XML (NLPXML-2002)*, Taipei, Taiwan, 2002.

[Ide and Romary, 2001] Nancy Ide and Laurent Romary. A common framework for syntactic annotation. In *Proceedings of ACL-2001*, pages 298–305, Toulouse, 2001.

[Ide, 2000] Nancy Ide. The XML framework and its implications for the development of natural language processing tools. In *Proceedings of the COLING Workshop on Using Toolsets and Architectures to Build NLP Systems*, Luxembourg, 2000.

[Krieger and Schäfer, 1994] Hans-Ulrich Krieger and Ulrich Schäfer. $\mathcal{TDL}$—a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, pages 893–899, 1994.

[Krieger, 2003] Hans-Ulrich Krieger. SDL—a description language for building nlp systems. In *Proceedings of the HLT-NAACL Workshop on the Software Engineering and Architecture of Language Technology Systems, SEALTS*, pages 84–91, 2003.

[Langendoen and Simons, 1995] D. Terence Langendoen and Gary F. Simons. A rationale for the TEI recommendations for feature structure markup. In Nancy Ide and Jean Veronis, editors, *Computers and the Humanities 29(3)*. Kluwer Acad. Publ., The Text Encoding Initiative: Background and Context, Dordrecht, 1995. Reprint.

[Lee *et al.*, 2004] Kiyong Lee, Lou Burnard, Laurent Romary, Eric de la Clergerie, Ulrich Schaefer, Thierry Declerck, Syd Bauman, Harry Bunt, Lionel Clément, Tomaz Erjavec, Azim Roussanaly, and Claude Roux. Towards an international standard on feature structure representation (2). In *Proceedings of the LREC-2004 workshop on A Registry of Linguistic Data Categories within an Integrated Language Resources Repository Area*, pages 63–70, Lisbon, Portugal, 2004.

[McKelvie *et al.*, 1998] David McKelvie, Chris Brew, and Henry Thompson. Using SGML as a basis for data-intensive natural language processing. *Computers and the Humanities*, 31(5), 1998.

[Piskorski and Neumann, 2000] Jakub Piskorski and Günter Neumann. An intelligent text extraction and navigation system. In *Proceedings of 6th RIAO-2000, Paris*, 2000.

[Schäfer, 2003] Ulrich Schäfer. WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components. In *Proc. of the Workshop on the Software Engineering and Architecture of LT Systems (SEALTS), HLT-NAACL03*, pages 9–16, Edmonton, Canada, 2003.

[Schmid, 2000] Helmut Schmid. *LoPar: Design and Implementation*. IMS, University of Stuttgart, Stuttgart, 2000. Arbeitspapiere des SFB 340, Nr. 149.

[Skut and Brants, 1998] W. Skut and T. Brants. Chunk tagger: statistical recognition of noun phrases. In *ESSLLI-1998 Workshop on Automated Acquisition of Syntax and Parsing*, Saarbrücken, 1998.

[Thompson and McKelvie, 1997] Henry S. Thompson and David McKelvie. Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML-EU-1997*, 1997.

[Uszkoreit, 2002] Hans Uszkoreit. New Chances for Deep Linguistic Processing. In *Proceedings of COLING 2002*, pages xiv–xxvii, Taipei, Taiwan, 2002.

Combining Shallow and Deep Processing for a Robust, Fast, Deep-Linguistic Dependency Parser

Gerold Schneider

Department of Linguistics, University of Geneva *
Institute of Computational Linguistics, University of Zurich
gerold.schneider@lettres.unige.ch

Abstract

This paper describes Pro3Gres, a fast, robust, broad-coverage parser that delivers deep-linguistic grammatical relation structures as output, which are closer to predicate-argument structures and more informative than pure constituency structures. The parser stays as shallow as is possible for each task, combining shallow and deep-linguistic methods by integrating chunking and by expressing the majority of long-distance dependencies in a context-free way. It combines statistical and rule-based approaches, different linguistic grammar theories and different linguistic resources. Preliminary evaluations indicate that the parser's performance is state-of-the-art.

1 Introduction

A variety of rule-based deep-linguistic parsers, usually following a formal linguistic theory, have existed for a number of years. Formal Grammar parsers have carefully crafted grammars written by professional linguists. In addition to expressing local relations, i.e. relations between a mother and a direct daughter node, a number of non-local relations, i.e. relations involving more than two generations, are also modeled. Unrestricted real-world texts pose a problem to many NLP systems that are based on Formal Grammars. Robust statistical parsers offer solutions to this problem [5], [8], [14], but they typically produce CFG constituency data as output, trees that do not express long-distance dependencies, grammatical functions or empty nodes, although Treebanks such as the Penn Treebank [21], on which they are typically trained, provide them[1]. This entails that the training cannot profit from valuable annotation data, and that the extraction of long-distance dependencies (LDD) and the mapping to shallow semantic representations is not always possible from the output of these parsers. The problem is even aggravated by parsing errors across an LDD [17].

Thanks to integrating statistics, some deep-linguistic Formal Grammar systems have now also achieved the coverage and robustness needed to parse large corpora: [27] show how a hand-crafted LFG grammar scales to the Penn Treebank with Maximum Entropy probability models. [3] acquire a wide-coverage LFG grammar from the Penn Treebank automatically, [15] a CCG grammar. We use a tag-sequence based, hand-crafted functional dependency grammar (FDG, [13], [29]), combined with

*This research was partly made possible by the Swiss National Science Foundation, grant 21-59416.99. Also many thanks to the anonymous reviewers for their valuable comments

[1] [8] Model 2 uses some of the functional labels, and Model 3 some long-distance dependencies

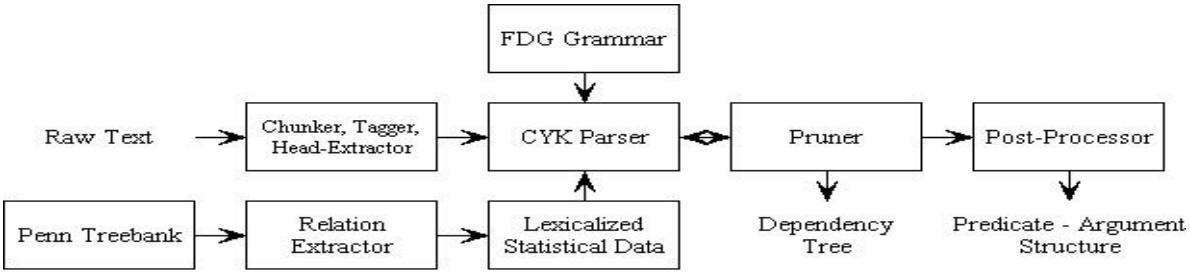


Figure 1: Pro3Gres flowchart

Maximum Likelihood Estimation (MLE) lexicalized probabilities extracted from the Penn Treebank, similar to [9], but for a large subset of dependency relations instead of for PP-attachment only, including the majority of LDDs.

Speed and complexity remain a serious challenge for Formal Grammar based parsers. Typical Formal Grammar parser complexity is much higher than the O(n^3) for CFG [11]. Parsing algorithms able to treat completely unrestricted long-distance dependencies are NP-complete [24]. Here, we suggest a particularly low-complexity, robust solution, offering on the one hand context-free parsing complexity, but on the other hand a deep-lingusitic analysis as with a type of Formal Grammar. Pro3Gres parses about 300,000 words per hour. Its flow chart is in fig. 1, a sample output in fig. 2.

The low complexity is due to the following reasons:

- Exploiting Functional Dependency Grammar, a Formal Grammar theory that directly delivers simple predicate-argument structures, that is in Chomsky Normal Form and does not know empty nodes, which allows us to employ a low-complexity parsing algorithm (CYK). In terms of [17], we apply the subtree-patterns not after, but before parsing, on the gold-standard and associate groups of patterns to dedicated dependency labels. Like [16] we include empty-node and functional label information to get near-full precision.[2]
- Tagging and chunking: As suggested by [2], we rely on shallow finite-state based approaches at the base phrase level and only parse between heads of chunks.
- Hand-crafted near-full coverage grammar: fully comprehensive grammars are difficult to maintain and considerably increase parsing complexity. We intentionally employ a near-full coverage grammar that does not cover very rare and marked constructions, and that leaves distinctions that are less relevant for the predicate-argument recognition task underspecified.
- Aggressive pruning: in order to keep search spaces small we discard improbable alternatives during the parsing process. Assuming a fixed beam size in a parse-time pruning system, which means that the total number of alternatives kept is constant from a certain search complexity onwards, real-world parsing time can be reduced to near-linear. If one were to assume a constantly full beam, or uses an oracle [25] it is linear in practice [3].

The rest of the paper is structured as follows. First we discuss that the parser stays as shallow as is possible for each task, combining shallow and deep-linguistic methods by integrating chunking and by expressing long-distance dependencies in a context-free way. Second, the probability model is illustrated. Finally, the performance of the parser and some of its elements are evaluated.

[2]Unlike Jijkoun, we apply the patterns directly to the Treebank, not a dependency-mapped version. Our patterns constitute a selective mapping from the Penn Treebank to Functional Dependency

[3]In practical terms, beam or oracle approach have very similar effects. If reasonable parameters are used, parser performance decreases only marginally while time behaviour improves by one to several orders of magnitude

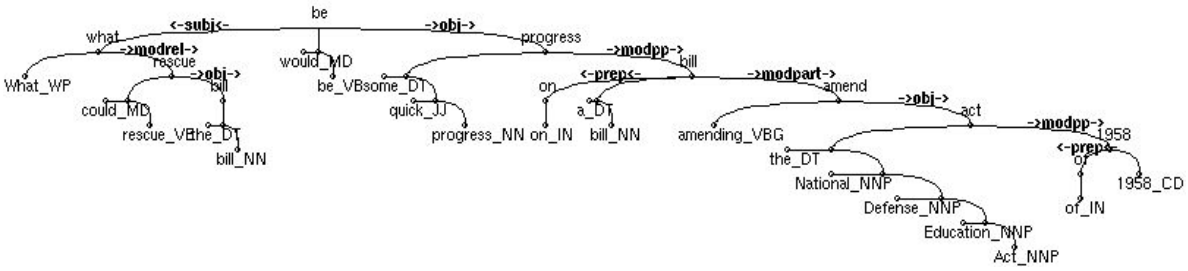


Figure 2: SWI Prolog Implementation Dependency Tree sample output

2 Stay as Shallow as You Can

In order to keep parsing complexity as low as possible, aggressive use of shallow techniques is made. For low-level syntactic tasks, tagging and chunking is used, and context-sensitive tasks are reduced to context-free tasks as far as is possible by the use of patterns that are deep-linguistic as they are non-local, but shallow as they are fixed. But it also entails that a hand-written, intentionally only almost fully comprehensive grammar, combined with aggressive pruning and a robust method for collecting partial parses is employed.

2.1 Tagging and Chunking

Low-level linguistic tasks that can be reliably solved by finite-state based techniques are handed over to them; namely the recognition of part-of-speech by means of tagging, and the recognition of base NPs and verbal groups and their heads by means of chunking [1], [2], [22]. The chunker and the head extraction method are completely rule-based. A small evaluation shows about 98 % correct head extraction. The extracted heads are lemmatized [23]. Parsing takes place only between the heads of chunks, and only using the best tag suggested by the tagger. The average number of words per chunk is 1.52 for Treebank section 0. In a speed test on section 0 with a toy NP and verb-group grammar parsing was about 4 times slower when using unchunked data as input[4]. The average number of possible tags per token is 2.11 for the entire Treebank. With untagged input, every possible tag would have to be taken into consideration by the parser. Although untested, at least a similar slowdown for untagged input as for unchunked input can be expected.

2.2 The Hand-Written Grammar

Acknowledged facts, for example that a verb has typically one but never two subjects, that adjuncts follow after all complements, that verbs can maximally have two noun objects etc. are expressed in hand-written declarative rules. The rules are based on the Treebank tags of heads of chunks. Since the tagset is limited and dependency rules are binary, even a broad-coverage set of rules can be written in relatively little time.

Linguistic constructions that are possible but very marked and rare are ruled out in this practically oriented system. For example, while it is generally possible for nouns to be modified by more than one PP, only nouns seen in the Treebank with several PPs are allowed to have several PPs. Or, while it is generally possible for a subject to occur to the immediate right of a verb (*said she*), this is only

[4]Due to the insufficiency of the toy grammar the lingusitic quality and the number of complete parses decreased, the same quality of analysis would even have been slower

Relation	Label	Example	Relation	Label	Example
verb–subject	subj	*he sleeps*	verb–prep. phrase	pobj	*slept in bed*
verb–first object	obj	*sees it*	noun–prep. phrase	modpp	*draft of paper*
verb–second object	obj2	*gave (her) kisses*	noun–participle	modpart	*report written*
verb–adjunct	adj	*ate yesterday*	verb–complementizer	compl	*to eat apples*
verb–subord. clause	sentobj	*saw (they) came*	noun–preposition	prep	*to the house*

Table 1: The most important dependency types used by the parser

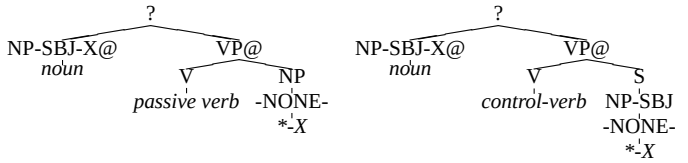


Figure 3: The extaction patterns for passive subjects (left) and subject control (right)

allowed for verbs seen with a subject to the right in the training corpus, typically verbs of utterance, and only in a comma-delimited or sentence-final context.

In a hand-written grammar, some typical parsing errors can be corrected by the grammar engineer, or rules can explicitly ignore particularly error-prone distinctions. Examples of rules that can correct tagging errors without introducing many new errors are allowing *VBD* to act as a participle (*VBN*) or the possible translation of *VBG* to an adjective. As an example of ignoring error-prone distinctions, the tagger's disambiguation between prepositions and verbal particles is unreliable. The grammar therefore makes no distinction between verbal particles and prepositions.

2.3 Long-Distance Dependencies

Treating long-distance dependencies is very costly [24], as they are context-sensitive. Most statistical Treebank trained parsers thus fully or largely ignore them.

[17] presents a pattern-matching algorithm for post-processing the Treebank output of such parsers to add empty nodes expressing long-distance dependencies to their parse trees. Encouraging results are reported for perfect parses, but performance drops considerably when using parser output trees. We have applied structural patterns to the Treebank, where like in perfect parses precision and recall are high, and where in addition functional labels and empty nodes are available, so that patterns similar to Johnson's but relying on functional labels and empty nodes reach precision close to 100%. Unlike in Johnson, also patterns for local dependencies are used; non-local patterns simply stretch across more subtree-levels. We use the extracted lexical counts as lexical frequency training material. Every dependency relation has a group of structural extraction patterns associated with it. This amounts to a selective mapping of the Penn Treebank to Functional DG. Table 1 gives an overview of the most important dependencies.

The $subj$ relation, for example, has the head of an arbitrarily nested NP with the functional tag *SBJ* as dependent, and the head of an arbitrarily nested VP as head for all active verbs. In passive verbs, however, a movement involving an empty constituent is assumed, which corresponds to the extraction pattern in figure 3, where VP@ is an arbitrarily nested VP, and NP-SBJ-X@ the arbitrarily nested surface subject and X the co-indexed, moved element. Movements are generally supposed to be of arbitrary length, but a closer investigation reveals that this type of movement is mostly fixed and can as well be replaced by a single, local and thus context-free dependency. Since the verb form allows a clear identification of passive structures, we have decided to keep the relation label $subj$, but to use

	Antecedent	POS	Label	Count	Description	Example
1	NP	NP	*	22,734	NP trace	*Sam* was seen *
2		NP	*	12,172	NP PRO	* to sleep is nice
3	WHNP	NP	*T*	10,659	WH trace	the woman *who* you saw *T*
(4)			*U*	9,202	Empty units	$ 25 *U*
(5)			0	7,057	Empty complementizers	Sam said 0 Sasha snores
(6)	S	S	*T*	5,035	Moved clauses	*Sam had to go*, Sasha said *T*
7	WHADVP	ADVP	*T*	3,181	WH-trace	Sam explained *how* to leave *T*
(8)		SBAR		2,513	Empty clauses	*Sam had to go*, said Sasha (SBAR)
(9)		WHNP	0	2,139	Empty relative pronouns	the woman 0 we saw
(10)		WHADVP	0	726	Empty relative pronouns	the reason 0 to leave

Table 2: The distribution of the 10 most frequent types of empty nodes and their antecedents in the Penn Treebank (adapted from [17])

Type	Count	prob-modeled	Treatment
passive subject	6,803	YES	local relation
indexed gerund	4,430	NO	Tesnière translation
control, raise, semi-aux	6,020	YES	post-parsing processing
others / not covered	5,481		
TOTAL	22,734		

Table 3: Coverage of the patterns for the most frequent NP traces [row 1]

separate probability estimations for the active and the passive case.

The same argument can be made for other relations, for example control structures, which have the extraction pattern shown in figure 3. Some relations include local alongside non-local dependencies. For example, the obj relation includes copular verb complements and small clause complements.

Grammatical role labels, empty node labels and tree configurations spanning several local subtrees are used as integral part of some of the patterns. This leads to much flatter trees, as typical for DG, which has the advantages that (1) it helps to alleviate sparse data by mapping nested structures that express the same dependency relation, (2) less decisions are needed at parse-time, which greatly reduces complexity and the risk of errors [17], (3) the costly overhead for dealing with unbounded dependencies can be largely avoided. [5]

Let us consider the quantitative coverage of these patterns in detail. The ten most frequent types of empty nodes cover more than 60,000 of the approximately 64,000 empty nodes of sections 2-21 of the Penn Treebank. Table 2, reproduced from [17] [line numbers and counts from the whole Treebank added], gives an overview.

Empty units, empty complementizers and empty relative pronouns [lines 4,5,9,10] pose no problem for DG as they are optional, non-head material. For example, a complementizer is an optional dependent of the subordinated verb. Moved clauses [line 6] are mostly PPs or clausal complements of verbs of utterance. Only verbs of utterance allow subject-verb inversion in affirmative clauses [line 8]. The linguistic grammar provides rules with appropriate restrictions for all of these. In a dependency framework, none of them involve non-local dependencies or empty nodes, [line 6] and [line 8] need rules that allow an inversion of the dependency direction under well-defined conditions.

[5] In addition to taking less decisions, it is ensured that the lexical information that matters is available in one central place, allowing the parser to take one well-informed decision instead of several brittle decisions plagued by sparseness. Collapsing deeply nested structures into a single labeled dependency relation is less complex but has the same effect as carefully selecting what goes in to the parse history in history-based approaches [5]

NP Traces The analysis of NP traces ([line 1] of table 2) is shown in table 3. It reveals that the majority of them are recognized by the grammar, and except for the indexed gerunds, they participate in the probability model. In control, raising and semi-auxiliary constructions, the non-surface semantic arguments, i.e. the subject-verb relation in the subordinate clause, are created based on lexical probabilities at the post-parsing stage, where minimal predicate-argument structures are output.

Unlike in control, raising and semi-auxiliary constructions, the antecedent of an indexed gerund cannot be established easily. The parser does not try to decide whether the target gerund is an indexed or non-indexed gerund nor does it try to find the identity of the lacking participant in the latter case. This is an important reason why recall values for the subject and object relations are lower than the precision values.

NP PRO As for the 12,172 NP PRO [line 2] in the Treebank, 5,656 are recognized by the $modpart$ pattern (which covers reduced relative clauses), which means they are covered in the probability model. The dedicated $modpart$ relation typically expresses object function for past participles and subject function for present participles.[6] A further 3,095 are recognized as non-indexed gerunds. Infinitives and gerunds may act as subjects, which are covered by [30] translations, although these rules do not participate in the probability model. Many of the structures that are not covered by the extraction patterns and the probability model are still parsed correctly, for example adverbial clauses as unspecified subordinate clauses. Non-indexed adverbial phrases of the verb account for 1,598 NP PRO, non-indexed adverbial phrases of the noun for 268. As the NP is non-indexed, the identity of the lacking argument in the adverbial is unknown anyway, thus no semantic information is lost.

WH Traces Only 113 of the 10,659 WHNP antecedents in the Penn Treebank [line 3] are actually question pronouns. The vast majority, over 9,000, are relative pronouns. For them, an inversion of the direction of the relation they have to the verb is allowed if the relative pronoun precedes the subject. This method succeeds in most cases, but linguistic non-standard assumptions need to be made for stranded prepositions.

Only non-subject WH-question pronouns and support verbs need to be treated as "real" non-local dependencies. In question sentences, before the main parsing is started, the support verb is attached to any lonely participle chunk in the sentence, and the WH-pronoun pre-parses with any verb.

3 Probability Model

We now explain Pro3Gres' main probability model by way of comparing it to [8]. We first consider the non-generative model [6] and then show how Pro3Gres' rules implement the extensions of [7] Model 2.

3.1 Relation of Pro3Gres to Collins 1996

Both [6] and Pro3Gres are mainly dependency-based statistical parsers parsing over heads of chunks, a close relation can therefore be expected. The [6] MLE and the main Pro3Gres MLE can be juxtaposed as follows (a and b are the lexical heads, R the relation, $dist$ the distance):

[6] The possible functional ambiguity is not annotated in the Treebank, hence the reduced relative clause is an unindexed empty NP

[6] MLE estimation: $P(R|\langle a, atag\rangle, \langle b, btag\rangle, dist) \cong \frac{\#(R,\langle a,atag\rangle,\langle b,btag\rangle,dist)}{\#(\langle a,atag\rangle,\langle b,btag\rangle,dist)}$

Main Pro3Gres MLE estimation: $P(R, dist|a, b) \cong p(R|a, b) \cdot p(dist|R) = \frac{\#(R,a,b)}{\#(\sum_{i=1}^{n} R_i a,b)} \cdot \frac{\#(R,dist)}{\#R}$

Four differences are observed: First, Pro3Gres does not use tag information. Reasons for this are because the licensing, hand-written grammar is based on Penn tags, and because Pro3Gres backs off to semantic WordNet classes [12] for nouns and to Levin classes [18] for verbs instead of to tags, which has the advantage that it is more fine-grained. Second, Pro3Gres uses real distances, measured in chunks, instead of a vector of features. While the type of relation R is lexicalized, i.e. conditioned on the lexical items, the distance is assumed to be dependent only on R. This is based on the observation that some relations typically have very short distances (e.g. verb-object), others can be quite long (e.g. Verb-PP attachment). This observation greatly reduces the sparse data problem. Third, the co-occurrence count in the MLE denominator is not the sentence-context, but the sum of competing relations. For example, the $object$ and the $adjunct$ relation are in competition, as they are licensed by the same tag sequence ($VB* \; NN*$). Pro3Gres models attachment probabilities, decision probabilities, which is in accordance with the view that parsing is a decision process. Fourth, relations (R) have a Functional Dependency Grammar definition, including long-distance dependencies.

3.2 Relation of Pro3Gres to Collins 1997 Model 2

[7] Model 2 extends the parser to include a complement/adjunct distinction for NPs and subordinated clauses, and it includes a subcategorisation frame model.

Let us first review [7] Model 2 with the example of the rewrite rule ***S(bought)*** → ***NP(week), NP-C(IBM), VP(bought)***. For the subcategorisation-dependent generation of dependencies in Model 2, first the probabilities of the possible subcat frames to the right p_{rc} and to the left p_{lc} are calculated, conditioned on the RHS mother category P, the LHS head category H and the lexical head h. The selected subcat frame is added as a condition to the left context l, respectively the right context r.

$$P_{head}(\text{VP}|\text{S,bought}) \cdot P_{lsubcat}(\{\text{NP-C}\}|\text{S,VP,bought}) \cdot P_{rsubcat}(\{\}|\text{S,VP,bought}) \cdot P_l(\text{NP-C(IBM)}|\text{S,VP,bought,}\{\text{NP-C}\}) \quad (1)$$

Once a subcategorized constituent has been found, it is removed from the subcat frame, so that if *IBM* is NP-C, h=*week* has an empty subcat frame.

$$P_l(\text{NP(week)}|\text{S,VP,bought,}\{\}) \quad (2)$$

This ensures that non-subcategorized constituents cannot be attached as complements, which is one of the two major function of a subcat frame. The other major function of a subcat frame is to ensure that, if possible, all the subcategorized constituents are found. In order to ensure this, the probability when a rewrite rule can stop expanding is calculated. Importantly, the probability of a rewrite rule with a non-empty subcat frame to stop expanding is very low, the probability of a rewrite rule with a non-empty subcat frame to stop expanding is higher.

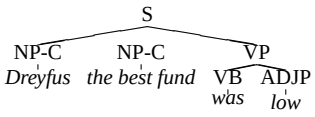
$$P_l(\text{STOP}|\text{S,VP,bought,}\{\}) \cdot P_r(\text{STOP}|\text{S,VP,bought,}\{\}) \quad (3)$$

The entire probability of the phrase S(bought) → NP(week), NP-C(IBM), VP(bought) is therefore

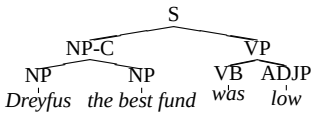
$$\begin{aligned} &P_{head}(\text{VP}|\text{S,bought}) \cdot P_{lsubcat}(\{\text{NP-C}\}|\text{S,VP,bought}) \cdot P_{rsubcat}(\{\}|\text{S,VP,bought}) \\ &\cdot P_l(\text{NP-C(IBM)}|\text{S,VP,bought,}\{\text{NP-C}\}) \cdot P_l(\text{NP(week)}|\text{S,VP,bought,}\{\}) \\ &\cdot P_l(\text{STOP}|\text{S,VP,bought,}\{\}) \cdot P_r(\text{STOP}|\text{S,VP,bought,}\{\}) \end{aligned}$$

Pro3Gres includes a complement/adjunct distinction for NPs. All the examples given in support of the subcategorisation frame model in [7] are dealt with by the hand-written grammar.

(a) incorrect

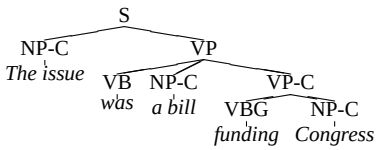


(b) correct

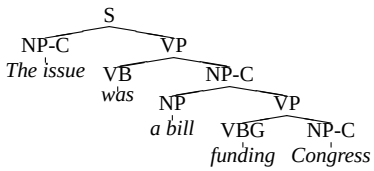


The grammar constraint that a verb is not allowed to have more than one subject forbids the incorrect analysis in Pro3Gres.

(a) incorrect



(b) correct



The obj and $obj2$ relations, the latter exclusively for ditransitive verbs, are separate relations in Pro3Gres. The $obj2$ relation probability is largely a verb ditransitivity probability.

In other words, every complement relation type, namely $subj$, obj, $obj2$, $sentobj$, can only occur once per verb, which ensures one of the two major functions of a subcat frame, that non-subcategorized constituents cannot be attached as complements. This amounts to keeping separate subcat frames for each relation type, where the selection of the appropriate frame and removing the found constituent coincide, which has the advantage of a reduced search space: no hypothesized, but unfound subcat frame elements need to be managed. As for the second major function of subcat frames – to ensure that if possible all subcategorized constituents are found – the same principle applies: selection of subcat frame and removing of found constituents coincide; lexical information on the verb argument candidate is available at frame selection time already. This implies that Collins' Model 2 takes an unnecessary detour.

As for the probability of stopping the expansion of a rule – since DG rules are always binary – it is always 0 before and 1 after the attachment. But what is needed in place of interrelations of constituents of the same rewrite rule is proper cooperation of the different subcat types. For example, the grammar rules only allow a noun to be $obj2$ once obj has been found, or a verb is required to have a subject unless it is non-finite or a participle, or all objects need to be closer to the verb than a subordinate clause.

4 Evaluation

In traditional constituency approaches, parser evaluation is done in terms of the correspondence of the bracketting between the gold standard and the parser output. [19] suggested evaluating on the linguistically more meaningful level of syntactic relations. For the current evaluation, a hand-compiled gold standard following this suggestion is used [4]. It contains the grammatical relation data of 500 random sentences from the Susanne corpus. The mapping between Carroll's grammatical relation format and our dependency output is explained in [28]. Mapping one annotation scheme to another is non-trivial and can only lead to indicative results [10]. The results in table 4 are similar to those reported in [20] and [26]. They suggest that the performance of the parser is roughly state-of-the-art[7], but more research will be needed. The new local relations corresponding to LDDs in the Penn Treebank have been selectively evaluated as far as the annotations permit, also shown in table 4:

[7] [20] gives results on the whole Susane corpus, [26] compares a number of successful probabilistic parsers using the same gold standard [4] as we do

	Percentages for some relations, on Carroll testset					only LDD-involving					
	Subject	Object	noun-PP	verb-PP	subord. clause	WHS	WHO	PSubj	CSubj	modpart	RclSubjA
Precision	91	89	73	74	68	92	60	n/a	80	74	89
Recall	81	83	67	83	n/a	90	86	83	n/a	n/a	63

Table 4: Evaluation on Carroll's test suite on subject, object, PP-attachment and clause subordination relations and a selection of 6 LDD relations

Error Classification of PP-Attachment Errors of the first 100 evaluation corpus sentences						
Description	Attachment Error	Head Extraction Error	Chunking or Tagging Error	compl/prep Error	Grammar Mistake or incomplete Parse	Grammar Assumption
Noun-PP Attach Prec.	22	1	8	0	3	3
Verb-PP Attach Prec.	12	1	5	1	1	2
Noun-PP Attach Recall	25	1	14	0	12	5
Verb-PP Attach Recall	2	0	1	0	0	0
Percentages	51 %	3 %	24 %	1 %	13 %	12 %

Table 5: Analysis of PP-Attachment Errors

WH-Subject (WHS), WH-Object (WHO), passive Subject (PSubj), control Subject (CSubj), reduced relative clause (modpart), and the anaphor of the relative clause pronoun (RclSubjA). Table 5 shows that about half of the PP-attachment errors are real attachment errors. The second most frequent error is deficient tagging or chunking – the price to pay for shallowness.

5 Conclusions

We have presented a fast, lexicalized broad-coverage parser delivering grammatical relation structures as output, which are closer to predicate-argument structures than pure constituency structures, and more informative if non-local dependencies are involved. A selective evaluation at the grammatical relation level indicates that its performance is state-of-the-art, but a full evaluation will be needed as future research.

We have shown that the parser stays as shallow as is possible for each task, combining shallow and deep-linguistic methods by integrating chunking and by expressing long-distance dependencies in a context-free way, thus offering on the one hand a parsing complexity as low as for a typical probabilistic parser, but on the other hand a deep-lingusitic analysis as with a type of Formal Grammar. In LFG terms, we parse directly for an f-structure without a c-structure detour. We model the majority of empty nodes and co-references by using an ID/LP grammar, dedicated functional labels, conservative DG assumptions [30] and statistical post-processing.

References

[1] Steven Abney. Parsing by chunks. In *Principle-Based Parsing*. Kluwer Academic Publishers, Dordrecht, 1991.

[2] Steven Abney. Chunks and dependencies: Bringing processing evidence to bear on syntax. In Jennifer Cole, Georgia Green, and Jerry Morgan, editors, *Computational Linguistics and the Foundations of Linguistic Theory*, pages 145–164. CSLI, 1995.

[3] M. Burke, A. Cahill, R. O'Donovan, J. van Genabith, and A. Way. Treebank-based acquisistion of wide-coverage, probabilistic LFG resources: Project overview, results and evaluation. In *The First International Joint Conference on Natural Language Processing (IJCNLP-04), Workshop "Beyond shallow analyses - Formalisms and statistical modeling for deep analyses"*, Sanya City, Hainan Island, China, 2004.

[4] John Carroll, Guido Minnen, and Ted Briscoe. Corpus annotation for parser evaluation. In *Proceedings of the EACL-99 Post-Conference Workshop on Linguistically Interpreted Corpora*, Bergen, Norway, 1999.

[5] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the ACL*, pages 132–139, 2000.

[6] Michael Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Philadelphia, 1996.

[7] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proc. of the 35th Annual Meeting of the ACL*, pages 16–23, Madrid, Spain, 1997.

[8] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, 1999.

[9] Michael Collins and James Brooks. Prepositional attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge, MA, 1995.

[10] Richard Crouch, Ronald M. Kaplan, Tracy H. King, and Stefan Riezler. A comparison of evaluation metrics for broad-coverage stochastic parsers. In *Beyond PARSEVAL workshop at 3rd Int. Conference on Language Resources an Evaluation (LREC'02)*, Las Palmas, 2002.

[11] Jason Eisner. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 5th International Workshop on Parsing Technologies*, pages 54–65, MIT, Cambridge, MA, September 1997.

[12] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[13] Jan Hajič. Building a syntactically annotated corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 106–132. Karolinum, Charles University Press, Prague, 1998.

[14] James Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of HLT-NAACL 2003*, Edmonton, Canada, 2003.

[15] Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, 2002.

[16] Valentin Jijkoun. Finding non-local dependencies: beyond pattern matching. In *Proceedings of the ACL 03 Student Workshop*, Budapest, 2003.

[17] Mark Johnson. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Meeting of the ACL*, University of Pennsylvania, Philadelphia, 2002.

[18] Beth C. Levin. *English Verb Classes and Alternations: a Preliminary Investigation*. University of Chicago Press, Chicago, IL, 1993.

[19] Dekang Lin. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of IJCAI-95*, Montreal, 1995.

[20] Dekang Lin. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*, Granada, Spain, 1998.

[21] Mitch Marcus, Beatrice Santorini, and M.A. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[22] Andrei Mikheev. Automatic rule induction for unknown word guessing. *Computational Linguistics*, 23(3):405–423, 1997.

[23] Guido Minnen, John Carroll, and Darren Pearce. Applied morphological generation. In *Proceedings of the 1st International Natural Language Generation Conference (INLG)*, Mitzpe Ramon, Israel, 2000.

[24] Peter Neuhaus and Norbert Bröker. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th ACL and 8th EACL*, pages 337–343, Madrid, Spain, 1997.

[25] Joakim Nivre. Inductive dependency parsing. In *Proceedings of Promote IT*, Karlstad University, 2004.

[26] Judita Preiss. Using grammatical relations to compare parsers. In *Proceedings of EACL 03*, Budapest, Hungary, 2003.

[27] Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadephia, PA, 2002.

[28] Gerold Schneider. Extracting and using trace-free Functional Dependencies from the Penn Treebank to reduce parsing complexity. In *Proceedings of Treebanks and Linguistic Theories (TLT) 2003*, Växjö, Sweden, 2003.

[29] Pasi Tapanainen and Timo Järvinen. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71. Association for Computational Linguistics, 1997.

[30] Lucien Tesnière. *Eléments de Syntaxe Structurale*. Librairie Klincksieck, Paris, 1959.

An XML Architecture for Shallow and Deep Processing

Kiril Simov, Alexander Simov, Petya Osenova
BulTreeBank Project
http://www.BulTreeBank.org
Linguistic Modelling Laboratory, Bulgarian Academy of Sciences
Acad. G. Bonchev St. 25A, 1113 Sofia, Bulgaria
kivs@bultreebank.org, alex@bultreebank.org, petya@bultreebank.org

Abstract

The paper presents a set of XML tools for natural language processing such as regular grammars, constraints, transformations, remove and insert operations. The architecture allows any combinations of the tools depending on the task and the concrete analysis. The main control mechanism is the backtracking which depends on achieving a particular subgoal in the analysis.

The main advantage of the architecture is better control over interleaving of "sure" steps (the shallow processing) and the "uncertain" steps (the deep processing). In this way the grammar developers can apply shallow processing not just as a first step, but at any level of processing.

We define shallow processing as a sequence of deterministic analyses and deep processing as a sequence of non-deterministic analyses. Thus the shallow and the deep components can be applied to each language level (morphology, syntax, semantics and pragmatics). The complexity of the processing depends on the complexity of the concrete task, not on the language level.

1 Introduction

The idea of robustness [Stede 2003] has trailed the NLP research into combining the shallow and deep techniques in the so called 'mixed-depth' processing [Hirst and Ryan 1992]. There is no strict boundary between the two approaches. After having discovered the advantages and weaknesses of either technique of language analysis, the researchers started to pursue hybrid strategies to solve the variety of tasks. Another priority in NLP is the preference of model-tasks to general and broad applications [Stede 2003]. Thus, different potential scenarios should be envisaged and made executable. Additionally, best practices in the creation of language resources rely on bootstrapping techniques and knowledge-assisting databases (lexicons). This means that platform flexibility, information complexity and non-determinism should be well supported. For the variety of tasks a core set of techniques is also required, such as: handling frequent cases as well as cases with great data impact, analogy mechanisms and incrementality.

In our opinion such a software architecture for supporting shallow and deep processing has to ensure at least the following functionalities: (1) application of the tools in a cascaded way (with possible repetition of the application); (2) context dependent application of the tools; (3) usage of temporary annotation (non-monotonicity); (4) change of the parsing strategy: mixing of top-down and bottom-up parsing; (5) dynamic reordering of the tool applications; (6) possibilities to redraw some of the previously generated analyses in cases of failure. A hybrid strategy for shallow processing covering points from 1 to 5 we have already outlined in [Simov and Osenova 2004]. In this paper we present an extension of the shallow processing architecture towards deep linguistic processing. The new architecture

can be considered as an incremental partial processing interleaved with deep processing steps. The main goal is to maximize the deterministic processing and to apply more complex non-deterministic processing only when it is unavoidable. The main application of the architecture for the moment is the construction of a treebank for Bulgarian. This application requires a semi-automatic construction of full syntactic analysis of sentences. This is achieved by using limited (at least with respect to the coverage) language resources and processors. Thus, the main task is to maximize the utility of the available resources in order to meet the requirements of the deep processing. This aim is handled by experimenting with various customizing strategies.

Most of the existing architectures for combining shallow and deep processing offer a central representation of the potential linguistic analyses which are produced by different processors like taggers, partial parsers and deep parsers. Thus these architectures are a media for exchange and combining of the different analyses. One of the prominent hybrid architectures, which provides possibilities for access to XML standoff annotation is WHAT: [Schäfer 2003]. Our approach is compatible with this view because it provides an interface to other programs based on XML as an exchange protocol, but it also provides a complete set of tools for the creation of full analysis inside the system. This is important when there are not enough external processors for a language. In this case it is simple for the developers to use the same software environment for all the processing instead of learning several platforms.

The structure of the paper is as follows: in Section 2 our architecture for shallow and deep processing is outlined. Section 3 describes the XML implementation of this general architecture in the CLaRK system. In Section 4 some applications to Bulgarian are considered with respect to the proposed ideas for combining shallow and deep processing in a dynamic way.

2 General Architecture for Combined Processing

The key ideas behind our XML architecture are in accordance with the annotation schema of the HPSG-based treebank for Bulgarian (BulTreeBank). This annotation schema can be viewed as a combination of shallow and deep processing where the shallow component is used to minimize the possible analyses resulting from the deep processing step. The actual selection of the true analyses is performed on the basis of information supplied by the annotator. Hence, the annotation schema comprises the following steps:

Partial parsing step. This step includes all the processing steps before the application of the HPSG grammar: (1) Sentence extraction from the corpus. (2) Automatic pre-processing. Each sentence needs first to be pre-processed at all the levels, that precede deeper syntactic annotation. These include: Morphosyntactic tagging; Named entity recognition; Part-of-speech disambiguation; Partial parsing. We aim at a result of a 100 % accurate partial parsed sentence. Thus at this level we do not apply the rules that are likely to produce errors in some contexts.

HPSG step. The result from the previous step is encoded into an HPSG compatible representation with respect to the HPSG sort hierarchy. The result is sent to the TRALE System [Meurers et al. 2002]. It takes the partial sentence analyses as an input and evaluates all the attachment possibilities for them. The output is encoded as feature graphs. In these feature graphs the initial status of the phrases is restored, i.e. their representation before the application of the HPSG grammar is still available.

Resolution step. The output feature graphs from the previous step are further processed in the following way: (1) their intersection is calculated; (2) then, on the basis of the differences, a set of constraints over the intersection is introduced; (3) during the actual annotation step, the annotator's

task is to extend the intersection to full analysis by adding the missing information. The constraints determine the appropriate extensions and also propagate the information, added by the annotator, in order to minimize the number of the incoming possibilities.

This annotation schema can be considered as an example of the *generate and test* approach to the exploration of the search space in order to find the right analyses. The role of shallow processing is mainly to reduce the initial size of the search space. The resolution step can be considered as a consultation with linguistic knowledge sources. The three steps design was guided by practical solutions but in general we think it can be a sequence of *generate and test* steps where the generate steps are of two kinds. These are called *shallow* and *deep* steps. Thus our architecture comprises two main types of processing mechanisms:

1. **Deterministic.** A processing step which produces a unique analysis within a given context.
2. **Non-deterministic.** A processing step which produces several different analyses within a given context, but only some of them are correct.

We consider the first kind of processing steps as shallow processing because their application is efficient and straightforward. The second kind of processing steps requires some additional source of linguistic knowledge to select the correct analysis from the potential analyses that have been generated. Thus we consider these steps as deep processing.

The main tools of the CLaRK System that we use to implement the two kinds of processing steps are: cascaded regular grammars and constraints. As an approximation of the constituent grammar we use the cascaded regular grammars. For the propagation of the information along the trees we use the constraints in insertion mode. For the selection of the appropriate analysis we use the constraints in validation mode. Note that we use these tools for both processing steps, but in different modes. The deterministic mode applies the tool and stores the result in the current analysis. The non-deterministic mode stores temporarily one of the possible analyses and checks for its validity. If the check is satisfied the system proceeds with the next processing. If the check failed then the next analysis would be selected.

The architecture allows a dynamic interleaving of the processing steps. Thus, after some non-deterministic steps deterministic ones can be applied. The general idea is for the non-deterministic steps to be applied as rarely as possible. This is done in accordance with the idea of dynamic networks of grammars presented in [Simov and Osenova 2004]: a dynamic network of grammars is a set of grammars (or other grammar networks) which apply to different contexts. The set is ordered (including cycles). The application of each grammar in the network depends on the place in the order and the context to which it is applicable. The crucial novelty in the current architecture is that in the previous work we applied the grammars only deterministically and now they are extended to handle some non-deterministic tasks. Thus the same idea about the dynamic nature of application is relevant in the new architecture.

3 Implementation in the CLaRK System

In this section we first describe the basic technologies of the CLaRK System[1] — [Simov et. al. 2001]. Then we describe the definition of macro language comprising tool queries and control operators which is the basis of the implementation of the architecture described above. The backtracking facility is an extension of the definition of the queries.

[1] For the latest version of the system see `http://www.bultreebank.org/clark/index.html`.

CLaRK is an XML-based software system for corpora development. It incorporates several technologies: *XML technology*; *Unicode*; *Regular Grammars*; and *Constraints over XML Documents*.

XML Technology

The XML technology is at the heart of the CLaRK System. It is implemented as a set of utilities for data structuring, manipulation and management. We have chosen the XML technology because of its popularity, its ease of understanding and its already wide use in description of linguistic information. In addition to the XML language [XML 2000] processor itself, we have implemented an XPath language [XPath 1999] engine for navigation in documents and an XSLT engine [XSLT 1999] for transformation of XML documents. We started with basic facilities for creation, editing, storing and querying XML documents and developed further this inventory towards a powerful system for processing not only single XML documents but an integrated set of documents. The main goal of this development is to allow the user to add the desirable semantics to the XML documents. The XPath language is used extensively to direct the processing of the document pointing where to apply a certain tool. It is also used to check whether some conditions are present in a set of documents.

Tokenization

The CLaRK System supports a user-defined hierarchy of tokenizers. At the very basic level the user can define a tokenizer in terms of a set of token types. In this basic tokenizer each token type is defined by a set of UNICODE symbols. Above this basic level tokenizers the user can define other tokenizers for which the token types are defined as regular expressions over the tokens of some other tokenizer, the so called parent tokenizer. For each tokenizer an alphabetical order over the token types is defined. This order is used for operations like the comparison between two tokens, sorting and similar.

Regular Grammars

The regular grammars in CLaRK System [Simov, Kouylekov and Simov 2002] work over token and element values generated from the content of an XML document and they incorporate their results back in the document as XML mark-up. The tokens are determined by the corresponding tokenizer. The element values are defined with the help of XPath expressions, which determine the important information for each element. In the grammars, the token and element values are described by token and element descriptions. These descriptions could contain wildcard symbols and variables. The variables are shared among the token descriptions within a regular expression and can be used for the treatment of phenomena like agreement. The grammars are applied in cascaded manner. The evaluation of the regular expressions, which define the rules, can be guided by the user. We allow the following strategies for evaluation: 'longest match', 'shortest match' and several backtracking strategies.

Constraints over XML Documents

The constraints that we have implemented in the CLaRK System are generally based on the XPath language (see [Simov, Simov and Kouylekov 2003]). We use XPath expressions to determine some data within one or several XML documents and thus we evaluate some predicates over the data. Generally, there are two modes of using a constraint. In the first mode **validation**, the constraint is used for a validity check, similar to the validity check, which is based on a DTD or an XML schema. In the second mode **insertion**, the constraint is used to support the change of the document to satisfy the constraint. The constraints in the CLaRK System are defined in the following way: `(Selector, Condition, Event, Action)`, where the selector defines to which node(s) in the document the constraint is applicable; the condition defines the state of the document when the constraint is applied. The condition is stated as an XPath expression, which is evaluated with respect to each node, selected by the selector. If the XPath expression is evaluated as true, then the constraint is applied; the event

defines when this constraint is checked for application. Such events can be: selection of a menu item, pressing of a key shortcut, an editing command; the action defines the way of the actual constraint application.

Cascaded Processing

The central idea behind the CLaRK System is that every XML document can be seen as a "blackboard" on which different tools write some information, reorder it or delete it. The user can arrange the applications of the different tools (not just regular grammars) to achieve the required processing. This possibility is called **cascaded processing**.

In the CLaRK System most of the tools support a mechanism for describing their settings. On the basis of these descriptions (called *queries*) a tool can be applied only by pointing to a certain description record. Each query contains the states of all settings and options which the corresponding tool has. In other words, each query has all the necessary information for applying the tool without any additional information or user interaction.

For user convenience and debugging purposes the queries themselves are represented in XML format. Within the system they can be treated like ordinary XML documents having their names and DTD assignments. For each kind of queries there is a special DTD included in the distribution package of the system. There the user can see the required structure for an XML document to serve as a query.

Once having this kind of queries there is a special tool for combining and applying them in groups (macros called **multiqueries**). During application the queries are executed successively and the result from an application is an input for the next one. The final result is given by the last query application.

Control Operators

For a better control on the process of applying several queries in one we introduce several conditional operators. These operators can determine the next query for application depending on certain conditions. When a condition for such an operator is satisfied, the execution continues from a location defined in the operator. The mechanism for addressing queries is based on user defined labels. When a condition is not satisfied the operator is ignored and the process continues from the position following the operator. In this way constructions like `IF-THEN-ELSE` and `WHILE-DO` easily can be expressed.

The system supports five types of control operators: `IF (XPath)`: the condition is an XPath expression which is evaluated on the current working document. If the result is a non-empty node-set, non-empty string, positive number or true boolean value the condition is satisfied; `IF NOT (XPath)`: the same kind of condition as the previous one but the result from the evaluation of the XPath expression is negated; `IF CHANGED`: the condition is satisfied if the preceding operation has changed the current working document or has produced a non-empty result document (depending on the operation); `IF NOT CHANGED`: the condition is satisfied if either the previous operation did not change the working document or did not produce a non-empty result; `GOTO`: unconditional changing the execution position.

Each macro defined in the system can have its own query and can be incorporated in another macro. In this way some limited form of subroutine can be implemented.

Backtracking

Two basic tools can be a subject of backtracking: the regular grammars and the constraints in insertion mode. A regular grammar can backtrack when over a given input it can succeed more than once, i.e. producing more than one different analysis. A constraint can backtrack when in a context of application it selects more than one value for insertion. In the case of a grammar the backtracking mechanism works in left to right scanning mechanism enumerating all possible analyses. In the case

of a constraint an order over the selected values is imposed and they are inserted in this order.

The backtracking over these basic queries is generalized over composite queries in the following way: (1) for a group of grammars the backtracking mechanism works for each grammar depending on the order of the grammars in the group. Thus, for a given input the first withdrawn analysis is the last one of the last grammar from the group which succeeded. So, if, for example, the first grammar does not produce any acceptable analysis, the second one is tried and so on. Of course, in some cases the result could be a result of analyses of several grammars in the group; (2) for a query containing several constraints the values of the first one are checked before the values of the second one and so on; (3) for a multiquery the backtracking mechanism works over all tools: the grammars, constraints, grammar group and constraints group queries in the order in which they appear in the multiquery.

A backtracking occurs when some of the `IF-THEN` operators or the `GOTO` operators are used with a special label `backtrack` instead of ordinary labels. Then the system withdraws all changes of the document to the point where the last check point for backtracking was saved and an attempt for a new analysis is done. For a better control over the backtracking mechanism backtracking cancellation operator (`CUT`) is implemented in the macro language. It causes deletion of all choice points.

4 Applications to Bulgarian

Here we present a few examples which demonstrate a possible application of the proposed architecture. Assuming the modularity of the linguistic knowledge available to the system we show how different levels of deeper linguistic analyses can be achieved. The language data normally provides conditions for both: deterministic and non-deterministic analyses. Some of them depend on universal criteria, but other are language-dependent. Deterministic ones are true: *always* or *in certain circumstances*. Non-deterministic ones resist unary interpretation due to insufficient knowledge. So, the idea is: knowing the selections of the deterministic steps, to reduce the triggers of non-determinism as much as possible and to reach the best parse/parses. Thus, besides the general principles which restrict all grammatical sentences, we also rely on storing more specific instances of these principles in order to manipulate deterministically concrete cases.

In our application to Bulgarian we consider as deterministic the following pre-processing steps:

1. Always applicable:
   (a) Unary or one-option analyses: morphosyntactic, chunks, one to one mappings from the other lexicons;
   (b) Lexicalized units or multiword expressions;
2. Applicable in certain contexts
   (a) Morphosyntactic disambiguation;
   (b) Chunking of phrases with certain right or left context;
   (c) Assigning dependency relations within chunks. See in [Osenova 2002].

We consider as non-deterministic the non-resolved ambiguities or in other words, overgeneration, on each level: grammatical, lexical, structural, constituent, dependency, semantic. The linguistically rich lexicon is of a crucial importance for resolving the ambiguities. See [Bouma 2001] and [Grover and Lascarides 2001] among others. The main problem remains the parsing failure due to the

unavailability or the insufficient coverage of such lexicons. This problem can be handled well with a provisional hierarchy of lexicons, which to supply the information, necessary for the parsing. In BulTreeBank we have several lexicons: morpho-syntactic, valency, named-entity, semantic, lexicon of multi-word and parenthetical expressions.

The idea is to order the lexicons according to the coverage and granularity, i.e. first would come the lexicon with the greatest coverage irrespectively of granularity. Then we should try to derive as much information as possible from the ones with best coverage while the others would be 'helpers' in that task. In our case the morpho-syntactic lexicon has the biggest coverage. Then come the lexicons of named-entities (16 000 items), which provide not only semantic, but also grammatical information. However, the more detailed lexicons suffer from insufficient coverage: the valency lexicon covers only 1000 most frequent verbs and the semantic one - 3000 nouns. The lists of multi-word expressions, parenthetical expressions, introductory expressions have also been compiled from scratch on the corpus language data. Note that (1) the semantic lexicon is incorporated as semantic constraints over the verb's arguments in the valency lexicon and (2) the introductory expressions when being verbs are considered extensions of the valency lexicon. It is clear that parsing would collapse given the availability of the presented above lexicons only. For that reason we are trying to customize the encoded knowledge as much as possible.

As it was mentioned above, the morpho-syntactic lexicon is the most elaborate and thus the most reliable one. Elsewhere we have already discussed the transfer of representation structures from this lexicon to attribute:value encodings [Osenova and Simov 2003]. But here we are concerned with the task how to derive as much linguistic knowledge as possible at this very level. We view the morpho-syntactic lexicon as a proto-valency lexicon. From the grammatical features we have derived some simple subcategorization rules, such as: if the verb is intransitive, then there can occur only one NP, which is the subject; if there is only one masculine noun with a full article inflection, it is the subject of the sentence etc. Also, agreement patterns are explored. The named-entity lexicons being provided with grammatical information according to their headwords, do not disturb the flow of the linguistic information at this level. Thus when there is no support from the valency or semantic dictionary, at least the rough proto-valency level survives. Note that in longer sentences more rules interfere and their management becomes more complex. The processing follows the basic algorithm below:

1. First, all the strings are tokenized; tokens are assigned classes according to the token classification — common words, names, or abbreviations; then they are morphologically tagged and disambiguated (including named-entities). It is done by the morphological tagger and two disambiguators. All multi-words and parenthetical expressions are analyzed as well;
2. If there exists an appropriate frame in the valency dictionary, it is mapped to the corpus;
3. If such a frame does not exist, then the proto-valency level is activated from the morphological lexicon via rules and propagation principles. As an additional source of information the accusative and dative clitics are used because they correspond to the arguments of the verb;
4. If it cannot ensure enough support, then some guessing mechanisms are activated on the base of the context information and the general token classification information is consulted.

Let us consider some example sentences with respect to the possibilities of the proposed architecture.

(1) Dyzhdyt prodylzhavashe da shumi navyn.
rain-the continued to babble outside
The rain continued babbling outside.

The morpho-syntactic annotation and disambiguation is trivial in the example and can be achieved with rules. The only composite chunk is the string ‘da shumi’. The analysis after the first shallow processing steps is:

```
<N>Dyzhdyt</N><V>prodylzhavashe</V><V-da>da shumi</V-da><Adv>navyn</Adv>
```

At this level the valency lexicon is consulted and the information is copied to the chunk level. In this example, this is done for the chunk ‘da shumi’ which receives practically the same valency frame as the verb form ‘shumi’[2]. The chunk ‘da shumi’ receives an intransitive frame specifying only the subject position. The frame for the matrix verb ‘prodalzhavashe’ determines an NP subject and a ‘da-clause’ complement additionally with the information for a control relation between the subject of the matrix verb and the subject of the ’da-clause’.

After the propagation of the information to the chunk top level a set of competing grammars is applied. The order of the grammars corresponds to the order of the realization of the head dependants in our grammar which is as follows: *complements, subjects, adjuncts*. Thus, first the grammar for verbal head-complement phrases is run, but it fails because: (1) the ‘da-clause’ is still not analyzed and there is no way to satisfy the complement requirements of the verb ‘prodalzhavashe’ and (2) the verb form ‘da shumi’ does not take a complement. Then the grammar for verbal head-subject phrases is applied, but it also fails because the complement requirements of the matrix verb are not satisfied. The N ’dyzhdyt’ is not taken as a subject of the ‘da shumi’ because our grammar presumes that in subject to subject control relation the subject of the complement clause cannot be extracted unless the matrix verb is impersonal. So, the subject is indicated by a coreferential *pro* element. However, the next grammar for verbal head-adjunct phrases succeeds and attaches the adverb to the verb form ‘da shumi’. Also a choice point is created, because the adjunct could be attached higher in the tree. The last grammar is applied once more because in general there can be more than one adjunct. For our example it fails and then the grammar for clauses is applied completing the ‘da-clause’ analysis. Then again the same grammar group is applied and in this case the analysis for the matrix verb is completed. It the end, a group of constraints for the control relation is applied. The resulting analysis is as follows:

```
<S>
  <VPS>
   <N id="1">Dyzhdyt</N>
   <VPC>
    <V>prodylzhavashe</V>
     <CLDA>
       <VPA><V-da pro-ss="1">da shumi</V-da><Adv>navyn</Adv></VPA>
     </CLDA>
   </VPC>
  </VPS>
</S>
```

Thus the analysis is complete. In case we want to enumerate all analyses we have to cause backtracking. In this case all the data added after the choice point is removed and an attempt for a different analysis is done. In this case we receive an analysis in which the adverb is attached at the sentential level. We will not present this analysis here.

In case that there is no information from the valency dictionary, the proto-valency information from the grammatical features is triggered. This, of course, raises the number of possible analyses, because

[2]Note that if it was a verb with accusative and/or dative clitics presented, then the valency frame would be changed accordingly.

the provided linguistic information is less constrained. Thus we have the information that both verbs are intransitive and by the guessing rules we come up with the following possibilities: the verb form ‘da shumi’ receives an intransitive valency frame, because it is the second verb and therefore - the governed one. However, the first verb, i.e. the matrix one ‘prodylzhavashe’ has two possibilities: either it receives also an intransitive valency frame, or it receives a valency frame of an intransitive subject-subject control verb as above. In the latter case there are two possible analyses which are identical to the ones mentioned above. In the case when the verb ‘prodylzhavashe’ receives an intransitive frame, the ‘da-clause’ can be only an adjunct to the matrix verb. Because there is no rule which requires the matrix clause and the adjunct clause to share their subjects there are again two possibilities. Thus in this case there are four possible analyses. Some of these analysis can be ruled out if additional preference rules are executed.

(2) Losha shega izigra na DPS zhurnalisticheskata nablyudatelnost.
Bad joke played to DPS journalistic-the watchfulness
Journalisic watchfulness played a practical joke on the Movement of Rights and Freedom

According to the token classification there is one abbreviation and common words. The abbreviation is matched against the appropriate lexicon and receives its extension and grammatical characteristics according to its head word. The result after the morphological and chunk processors is as follows:

```
<NPA>Losha shega</NPA> <V>izigra</V> <PP> na <NPA>DPS</NPA></PP>
<NPA>zhurnalisticheskata nablyudatelnost</NPA>
```

The valency lexicon provides the frame of the verb: izigraya(Subj-NP, DirObj-NP, IndirObj-PP). But here we have to deal also with the word order and to map the correct grammatical roles. Thus, the two NP(Adjunct)s, which are output of the chunking stage, undergo head-dependency transformation. Then, via a special principle of definiteness, the information from the non-head daughters is propagated to the mother nodes. Thus, the first NPA is indefinite while the second (we do not consider here the NPA inside the PP) is definite. Both competing NPAs are singular, so the agreement with the verb is not of a help to us. Thus without additional information we have two possible ananlyses. In order to rule the unlikely analysis we can apply a preference rule which says that in all other circumstances being equal the subject is more likely to be definite and the dependants are more likely to be indefinite.

5 Conclusion

In this paper we presented an architecture for a shallow and deep processing. The processing tools are the same in both kinds of analysis, but with different modes of application. The shallow processing is mainly deterministic and the deep processing includes also possibilities for non-deterministic steps. The architecture allows for a dynamic interleaving of the two kinds of the steps as well as different degree of depth of the used linguistic knowledge. The main processing tools are: (1) cascaded regular grammars arranged in networks, and (2) constraints used for information propagation and validation of the analyses. The architecture is implemented within an XML-based system for corpora development: CLaRK System.

From implementation point of view our future work is connected with the refinement of backtracking strategies and with more efficient ways for storing the linguistic information.

From linguistic point of view we plan to make more reliable integration between the language resources themselves, and between language resources and implementation possibilities.

References

[Bouma 2001] Gosse Bouma. 2001. *Extracting Dependency Frames from Existing Lexical Resources.* In: Proc. of the NAACL Workshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations, Somerset, NJ, USA.

[Grover and Lascarides 2001] 2001. Claire Grover and Alex Lascarides. *XML-Based Data Preparation for Robust Deep Parsing.* In: Proc. of the Joint EACL-ACL Meeting (ACL-EACL 2001), Toulouse, Frace.

[Hirst and Ryan 1992] Graeme Hirst and Mark Ryan. *Mixed-depth representations for natural language text.* In: Jacobs, Paul S. (editor). Text-basedintelligent systems, Hillsdale, NJ: Lawrence Erlbaum Associates.

[Meurers et al. 2002] Detmar Meurers, Gerald Penn, and Frank Richter. 2002. *A Web-based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing*. In *Proc. of the Effective Tools and Methodologies for Teaching NLP and CL*. ACL. Philadelphia, PA, USA.

[Osenova 2002] Petya Osenova. 2002. *Bulgarian Nominal Chunks and Mapping Strategies for Deeper Syntactic Analyses.* In: *Proc. of The Workshop on Treebanks and Linguistic Theories*. Sozopol, Bulgaria.

[Osenova and Simov 2003] Petya Osenova and Kiril Simov. 2003. *Between Chunk Ideology and Full Parsing Needs*. In: *Proc. of the Shallow Processing of Large Corpora (SProLaC 2003) Workshop*. Lancaster, UK.

[Schäfer 2003] Ulrich Schäfer. 2003. *WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components.* In: Proc. of HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems. Edmonton, Alberta, Canada.

[Simov and Osenova 2004] Kiril Simov and Petya Osenova. 2004. *A Hybrid Strategy for Regular Grammar Parsing.* In: Proc. of LREC 2004. Lisbon, Portugal.

[Simov et. al. 2001] Kiril Simov, Zdravko Peev, Milen Kouylekov, Alexander Simov, Marin Dimitrov, Atanas Kiryakov. 2001. *CLaRK - an XML-based System for Corpora Development.* In: Proc. of the Corpus Linguistics 2001 Conference. Lancaster, UK.

[Simov, Kouylekov and Simov 2002] Kiril Simov, Milen Kouylekov, Alexander Simov. *Cascaded Regular Grammars over XML Documents.* In: Proc. of the 2nd Workshop on NLP and XML (NLPXML-2002), Taipei, Taiwan.

[Simov, Simov and Kouylekov 2003] Kiril Simov, Alexander Simov, Milen Kouylekov. *Constraints for Corpora Development and Validation.* In: Proc. of the Corpus Linguistics 2003 Conference. Lancaster, UK.

[Stede 2003] Manfred Stede. 2003. *Shallow - Deep - Robust.* In: G. Willee, B. Schroder, H.-C. Schmitz (eds.): Computerlinguistik -Was geht, was kommt? Computational Lingusitics - Achievements and Perspectives. Sankt Augustin.

[XML 2000] XML. 2000. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/REC-xml

[XPath 1999] XPath. 1999. *XML Path Language (XPath) version 1.0.* W3C Recommendation. http://www.w3.org/TR/xpath

[XSLT 1999] XSLT. 1999. *XSL Transformations (XSLT). version 1.0.* W3C Recommendation. http://www.w3.org/TR/xslt