International Journal of Computational Linguistics and Applications vol. 7, no. 2, 2016, pp. 11–27
Received 08/02/2016, accepted 04/03/2016, final 21/09/2016
ISSN 0976-0962, http://ijcla.bahripublications.com

Dependency-Based Sentence Simplification for Large-Scale LFG Parsing: Selecting Simplified Candidates for Efficiency and Coverage

ILNAR SALIMZIANOV AND ÖZLEM ÇETINOĞLU

University of Stuttgart, Germany

ABSTRACT

Large scale LFG grammars achieve high coverages on corpus data, yet can fail to give a full analysis for each sentence. One approach proposed to gain at least the argument structure of those failed sentences is to simplify them by deleting subtrees from their dependency structure (provided by a more robust statistical dependency parser). The simplified versions are then re-parsed to receive a full analysis. However, the number of simplified sentences this approach generates is infeasible for parsing. As a solution, only a subset of candidates is selected based on a metric. In this work we apply the so-called parsability metric [1], introduced as an error-mining technique for grammar writing, for selecting among simplified candidates to be parsed and show that we improve over the previous results that use sentence length as the selection metric.

1 INTRODUCTION

Hand-crafted Lexical Functional Grammars (LFGs) can provide deeper analyses for sentences when they are able to parse them, but usually are less robust than statistical parsers and cannot give full-fledged solutions for 100% of sentences. Of course it is possible to

increase the coverage of an LFG grammar by adding more rules or by relaxing existing ones. But such modifications are, firstly, labor-intensive and time-consuming, and secondly, they require a high level of linguistic expertise. Thirdly, the sentences to be parsed could be in non-standard orthography or grammar. Hence we would prefer an automatic way of dealing with sentences an LFG grammar failed to parse. This paper is on one of the ways to deal with such sentences, namely on dependency-based sentence simplification.

Figure 1 illustrates a sentence in German from the TIGER corpus [2] German ParGram grammar failed to fully parse[1]. The problem with the sentence is that the article *des* 'of the' and the adjective *japanischen* 'Japanese' are both in genitive case, indicating that that they are part of a genitive construction, but the noun *Außenministerium* 'foreign ministry' is in nominative case and therefore does not agree with the surrounding words. Because of that the German ParGram grammar outputs only a fragmented analysis for the sentence as shown in Figure 2.

Fragmented analysis is what XLE[2] delivers when it cannot produce a complete parse. That is, no rule would allow connecting the fragments found. This is not enough for our research purposes, in part because fragments identified are often incorrect. E.g., as Figure 2 shows, the problematic phrase *des japanischen Außenministerium* ended up in two separate phrases. The functional structure[3] is similarly affected and *des japanischen* does not modify the noun *Außenministerium* as it would normally do. Thus, sentences with fragments are lost in terms of getting information from the analyses.

[1] The example was taken from [3].

[2] XLE [4] is a framework for parsing and generating with an LFG grammar.

[3] LFG theory [5] distinguishes between two core syntactic structures – c-structure (constituent structure) and f-structure (functional structure). C-structures are constituent or phrase structure trees. F-structures are sets of attribute-value pairs; attributes may be features, such as tense or gender, or functions, such as subject or object.

(1) *Ein Sprecher **des japanischen Außenministerium** verkündete*
A speaker of the japanese foreign ministry proclaimed
daraufhin , man werde Jelzins Aussage " vorsichtig analysieren " ,
then , one would Yelzin's statement " carefully analyze " ,
bevor man sie kommentiere , aber :
before one it comment , but :
'A speaker **of the Japanese foreign ministry** then proclaimed that Yeltsin's statement would be " carefully analyzed " , before commenting on it , but :'

Fig. 1. A sentence which the German ParGram grammar failed to parse. The problem is in the genitive phrase *des japanischen Außenministerium*, where the noun is missing the genitive ending *-s* and hence does not agree with the article and adjective in case.

Çetinoğlu et al. [3] propose a dependency-based sentence simplification approach to achive a full parse of a simplified version, where at least the arguments of the original sentence are kept. Sentences with no full parses are simplified by deleting one or more subtrees, and each shorter sentence is a candidate to reparse and get a full analysis. However, reparsing all candidates is not a feasible approach, as the number of candidates gets too high for long sentences with the combination of multiple subtree options to be deleted. To overcome this problem, the authors take 10 shortest simplified sentences as their candidates for full parses. Although this method proves that for more than half of the cases, the full analysis for the argument structure of failed parses could be obtained, it has a drawback: the system does not give an opportunity to longer candidates where the arguments *and* other constituents might be kept and parsed into a full analyses. In this work we keep the argument preservation criterion and we propose a more informed metric to select the candidates that lead to higher coverages with longer simplified sentences.

A few words have to be said on what we mean by coverage. Original sentence is considered "covered" if at least one of its simplified variants receives a full parse. However, our aim is not to

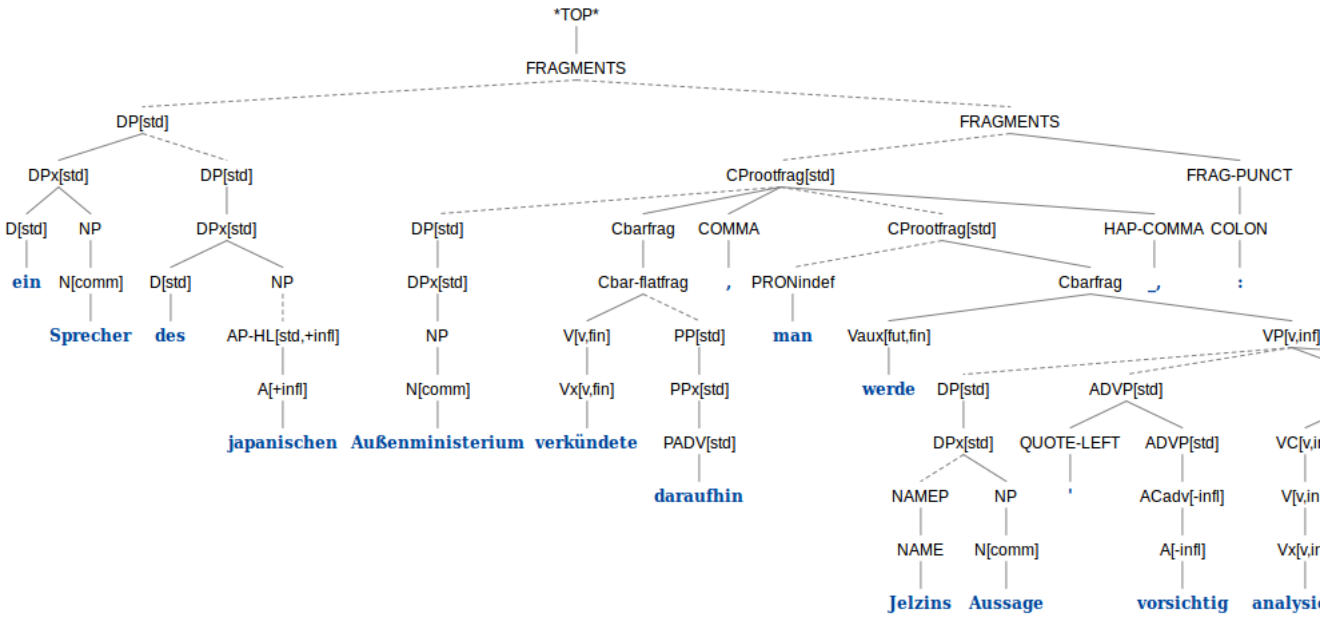

Fig. 2. The c-structure German ParGram Grammar provides for the sentence in Figure 1 (to focus on the problematic part we only show part of the tree)

get a full parse for just any of the simplified candidates, but for candidates that are as long as possible.

The remainder of the paper proceeds as follows. In section 2 we describe in more detail the dependency-based sentence simplification, as proposed by [3] and the problem with a number of candidates infeasible for parsing this approach generates. In section 3 we define the parsability metric which will be used for selecting candidates likely to receive a parse. In section 4 we describe the experiments we conducted. We discuss the experimental results in section 5 and give a short overview of other methods for increasing the coverage of LFG grammars in section 6. We conclude with section 7

2 Dependency-based Sentence Simplification

The idea of dependency-based sentence simplification is as follows: sentences which LFG parser failed to fully parse are parsed with a more robust statistical dependency parser. Having a dependency tree for the sentence allows to identify subtrees which are considered to be deletable. A subtree is defined to be deletable if removing it would not lead to a change in the arguments structure. For instance, subjects are not deleteble, whereas adjuncts are. The list of deletable subtrees is prepared manually based on heuristics, the actual list is language and treebank dependent. Once the subtrees are deleted from the dependency tree, the shorter candidates are reparsed with the LFG parser.

Figure 3 shows a part of the dependency parse tree for the sentence in Figure 1. Despite the mistake on the noun *Außenministerium*, *des* and *japanischen* correctly modifiy the noun as noun kernels (NK) and the whole phrase is identified as the genitive adjunct (AG) of the noun *Sprecher* ‘speaker’.

The way the deletion of subtrees works is the following: if the edge label of the node is in the list of deletable labels, then the node itself and all its descendants are deleted from the tree. Edge labels of the heads of deletable subtrees for German are shown in Table

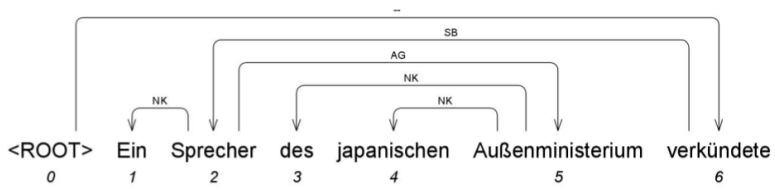

Fig. 3. Part of the dependency tree for the sentence in Figure 1. The tagset from the TIGER Corpus [2] is used as the edge labels.

1. So, for the parse tree shown in Figure 3, the following subtrees would be deletable: *japanischen* and *des japanischen Außenministerium*.

A careful reader might have noticed that the subtree covering the word *des* 'of the' was not deleted from the tree although its edge label – NK – is in the list. This is because the edge label NK is subject to additional conditions for being deleted, namely it is deletable only when it functions as an adjunct (see Figure 4 for the implementation).

If all possible combinations of deletable subtrees are deleted from the original sentence, the number of generated candidates is infeasible for parsing. Çetinoğlu et al. [3] report it to reach up to 608255 candidates per sentence while the average number is 924. The authors address this problem in two ways. In the first approach, they delete only one subtree at a time from the sentence ('one subtree shorter'). In the second approach, all possible combinations of subtrees are deleted, and only 10 shortest candidates plus the shortest candidate without punctuation symbols are taken ('ten shortest').

3 PARSABILITY METRIC

Parsing one subtree shorter or ten shortest candidates improves the coverage of fully parsed candidates. However, it does not give a chance to longer hence more informative candidates that would also receive full parses.

Table 1. Edge labels of the heads of deletable subtrees. Tags are the tags used in the TIGER corpus [2].

AG	genitive adjuncts
APP	appositions
JU	discourse-marker like
MNR	PP adjuncts (in noun-phrases)
MO	modifiers
NG	negation
PAR	head of parenthesis
PG	possessive PP adjuncts
PH	placeholders (e.g. German Vorfeld *es*)
PNC	proper noun components
RC	relative clauses
RE	infinite clauses attached to nominals
SBP	PP subjects in passive
UC	inside foreign language phrases
VO	vocatives
NK	noun kernels
DA	datives

It is easier to see the reason why this is the case when 10 shortest candidates are taken compared to the 'one subtree shorter' setting. In the latter case we are deleting at most one subtree to produce the candidate, and we might think that candidates produced would be longest possible. However, subtrees can yield n-grams of different length (and we measure length of sentences in terms of tokens) or two or more subtrees may yield n-grams of the same length. Maybe there are longer sentences which would also receive a full parse or two or more candidates of equal length with one being more likely to receive a full parse than the other(s)?

This work is about exploring metrics other than the length to select simplified candidates to be parsed. For this purpose, we use van Noord's parsability metric[1]. Van Noord used the metric for the purposes of error detection in grammar writing. In one of their experiments, [3] use the metric to identify and then delete least parsable n-grams from problematic sentences. Nonetheless, that is

Function 1: Produce True if the token (read: the node) is the head of a deletable subtree.

```
## Token Sentence -> Boolean
def head_of_deletable_subtree(t, s):
    """
    Produce True if token is the head of
    a deletable subtree of the sentence.
    """
    if t.deprel in DELETABLE_SUBTREES:
        if t.deprel == 'NK':
            if t.pos in ['ADJA', 'ADJD', 'ADV', 'KOUS']:
                return True
            elif t.pos == 'NN' and \
                    s[int(t.head) - 1].pos == 'NN':
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Fig. 4. A function (in Python 3) for deciding whether the node is the head of a deletable subtree

not the best way to simplify sentences since deleting n-grams only according to their parsability does not guarantee the grammaticality of the remaining sentence or preservation of the core arguments.

Parsability score of an n-gram is defined as follows:

$$parsability(\text{n-gram}) = \frac{count(\text{n-gram occurs in parsed sentences})}{count(\text{n-gram occurs in *all* sentences})}$$

We prioritize the removal of subtrees with zero or very low parsability scores to narrow down the candidate space.

The parsability score table we work with contains scores for 1, 2 or 3-grams, but no longer than that. If the subtree yields an n-gram longer than 3 tokens, or there are no scores for that particular n-gram in the table, we take the minimum of the parsability scores of lower order n-grams it contains. If there is no score for 1-gram

in the parsability table, we set its parsability score to 1.0. Here is how parsability score of an n-gram is defined formally:

```
if ngram in ParsabilityTable:
    score = ParsabilityTable[ngram]
elif len(ngram) == 1:
    score = 1.0
else:
    min(scores of n-1-grams ngram contains)
```

In other words, the algorithm deletes subtrees for which no parsability score is known latest, if at all.

We hypothesize that by deleting least parsable subtrees first we can recover a number of sentences comparable to that reported by [3] while finding longer solutions.

4 Experimental Setup

In this section we first give the data and tools we use in our experiments and then continue with the description of different experiments we conducted.

4.1 *General Setup*

We use TIGER treebank [2] as the source of German sentences which German LFG parser fails to parse but for which a gold standard dependency tree is available. We use the dependency version of the TIGER treebank which was converted by [6].

The number of failed sentences is 9160. We use the same version used in [3] of the German ParGram grammar[7]. Although the dataset and grammar versions are the same, the number of sentences XLE fails to parse fully is 9373 in [3]. Our reasoning is that either the newer version of XLE we used in these experiments brought improvements which lead to a higher coverage or the hardware (e.g. memory size) that experiments ran on played a role.

For a fair comparison between previous and new approaches, we start the next section by reproducing the experiments from [3]. The reconducted experiments serve as baseline numbers for our setting.

4.2 *Reproducing the Results of (Çetinoğlu et al., 2013)*

ONE SUBTREE SHORTER In the first experiment, only one deletable subtree at a time is deleted from the original sentence. The number of possible candidates equals the number of deletable subtrees in the original sentence.

10 SHORTEST + SHORTESTNOPUNCT Sometimes the problem which prevents the sentence from being parsed involves several subtrees. To account for that, in this experiment we delete all possible combinations of subtrees from the sentence and then take the 10 shortest candidates. From the shortest candidate, we remove punctuation symbols and include the result as the eleventh candidate.

4.3 *New Experiments*

10 LONGEST + LONGESTNOPUNCT Following our goal of finding parses for sentences as long as possible, we take 10 longest candidates out of the set of all possible candidates. The longest candidate without punctuation symbols is included as the eleventh candidate.

The motivation for this experiment is twofold. First, we would like to see how naively taking longest candidates without applying the parsability metric for selecting candidates affects coverage. Second, intuitively, taking longest candidates should lead to longest solutions. This experiment is an intuitive upper bound for the average length of successfully parsed candidates.

10 WITHOUT LEAST PARSABLE SUBTREES + FIRSTNOPUNCT In this experiment, up to ten (depending on how many deletable

subtrees there are in the original sentence) subtrees are deleted, starting with least parsable ones. Punctuation symbols are removed from the first candidate, and this version is included as the eleventh candidate.

We delete punctuation symbols from one of the candidates because we think that sometimes they can cause the parser to fail to parse a sentence. Punctuation symbols are deleted from the first candidate generated. In the '10 Shortest + ShortestNoPunct' and '10 Longest + LongestNoPunct' experiments these are the shortest and the longest candidates, respectively. In the '10 Without Least Parsable Subtrees + FirstNoPunct' experiment, this is the sentence after deleting the least parsable subtree from it.

The reason for deleting the punctuation symbols from the first candidate was initially mere technical, but the result lines up with the motivation behind experiments. '10 Shortest + ShortestNoPunct' can be seen as the pessimistic approach – we start with the shortest candidate possible, which we get by deleting punctuation from the shortest candidate. In the '10 Longest + LongestNoPunct' experiment, our goal is to preserve as much information as possible, and punctuation, putting examples like "A woman, without her man, is nothing" vs "A woman: without her, man is nothing" aside, arguably contains least amount of information. In the '10 Without Least Parsable Subtrees + FirstNoPunct' the motivation is again to preserve as much information as possible and therefore it is a good idea to start by deleting punctuation.

However, deleting punctuation symbols always from the shortest candidate might have been a better approach.

5 Experimental Results

Table 2 presents results for all experiments and some of their combinations. In the second column of it, coverage results are shown. By coverage we mean receiving a full parse after simplification. In addition, the average length of fully parsed simplified sentences (FPSS), calculated by *count*(tokens in FPSS) / *count*(FPSS) and the

average number of candidates explored before a solution is found are given.

Note that since *count*(FPSS) is different for each of the experiments (the number in the second column of Table 2), experiments are not strictly comparable against each other in terms of of the average length of solutions they generate. In particular, the 'OneSubtreeShorter' experiment led to solutions on average longer than the '10Longest + LongestNoPunct' experiment. This is because the 'OneSubtreeShorter' experiment found solutions for more input sentences than '10Longest + LongestNoPunct' setting did, and thus we are calculating the average length of two different sets of sentences. If we consider only the input sentences which received a solution in both of the approaches, then the average length of fully parsed sentences in 'OneSubtreeShorter' is 17.46, and in '10Longest + LongestNoPunct' it is 19.16 tokens.

When evaluating effectiveness of a combination of two approaches, if two solutions were found, only the longest solution was considered.

Table 2. Number of original sentences which are fully parsed after simplification, average length of fully parsed simplified sentences and average number of candidates explored before the first full parse is found

	Experiment	**Sent. parsed after simplification**	**Avg. length of fully parsed simplified sent. (tokens)**	**Avg. # of candidates explored**
1	OneSubtreeShorter	4188 (45.72%)	18.09	4.87
2	10Shortest + ShortestNoPunct	4714 (51.46%)	7.50	4.82
3	Combination of 1 and 2	5265 (57.48%)	15.52	–
4	10Longest + LongestNoPunct	3548 (38.73%)	17.89	6.39
5	Combination of 1 and 4	4657 (50.84%)	18.48	–
6	10WithoutLeast-Parsable + FirstNoPunct	4490 (49.02%)	16.35	5.83
7	Combination of 1 and 6	4648 (50.74%)	17.20	–

The parsability metric proposed by [1] as the error mining technique for hand-coded grammars proves to be useful for selecting among simplified versions of a sentence to be parsed.

- So far, taking 10 shortest candidates from the set of candidates which can be generated by deleting all possible combinations of subtrees (experiment 2) allowed to recover the highest number of original sentences (51.46%),
- while taking the combination of 10 longest and one subtree shorter (experiment 5) on average expectedly led to longest solutions.
- Deleting 10 subtrees yielding least parsable n-grams in a greedy manner (experiment 6) delivers solutions comparable in length to that obtained by considering 10 longest candidates per each sentence (experiment 4), but allows to re-parse almost as many sentences as in the '10 shortest' setting (experiment 2).

6 Related Work

Several other researchers have worked on achieving high coverage scores with LFG grammars. For English, Riezler et al. [8] report a full coverage on the Wall Street Journal, by including fragmented and skimmed analyses in the results. For German, Rohrer et al. [7] employ the same techniques and parse the TIGER Treebank. Additionally they modify the German ParGram Grammar to better handle troublesome phenomena such as coordination, subject gaps, reported speech clauses, and parentheticals. Dost and King [9] go beyond the standard treebank sentences to test the robustness and coverage of the ParGram English Grammar. They choose Wikipedia articles as their test medium to discover lexical and syntactic shortcomings, and use their findings for improving the grammar coverage.

Cahill et al. [10] guarantee a high coverage by using a statistical constituency parser to obtain the c-structure of a sentence, and then annotating the nodes of the c-structure with f-structure constrains and applying a constraint solver. Despite that the c-structure

coverage is 100% thanks to the robust statistical parser, not all sentences can have an f-structure due to the constraint solver. Moreover f-structure outputs carry much less information than usual ParGram grammar outputs. Hautli et al. [11] close this gap by enriching f-structures, yet cannot achieve a full coverage.

Although sentence simplification is not commonly used in LFG grammars for coverage purposes, there has been research on sentence simplification itself. Riezler et al. [12] employ transfer rules to simplify English computer news articles, paying attention to meaning preservation. They apply transfer rules to parsed f-structures and obtain reduced versions. The reduced f-structures are then disambiguated and passed into the generator to get shorter versions. The transfer rules used in this system are described in more detail in [13].

The most similar work to ours is a recent paper from [14], where authors rely on dependency relations obtained from an independent data-driven dependency parser to compose fragment f-structures for parsable parts of Polish sentences into a full f-structure. This approach is not applicable to us as the fragment f-structures might be incorrect themselves in the German ParGram Grammar, as discussed in section 1 and shown in Figure 1 (although on the figure you see the c-structure for the sentence, not the f-structure, f-structure is affected in a similar way). However, authors claim that "in contrast to English, where partial parsing is used to parse incorrect sentences, Polish sentences with FRAGMENT parses are not necessarily incorrect" and that "many of them are well-formed but contain linguistic phenomena for which POLFIE[4] rules have not been defined yet". One might argue that such cases where fragments themselves are wrong are indeed infrequent.

7 Conclusion and Future Work

In this work we make use of a metric in selecting better candidates to be parsed in a sentence simplification system that aims to in-

[4] Polish LFG grammar.

crease the number of sentences that have a full LFG analysis. We use the German ParGram Grammar to parse sentences from the TIGER corpus and fail to get a complete analyses in 19.44% of the cases. In order to get the analyses of the core arguments of the sentences, we simplify them and parse the shorter sentences. To decide what constituents to retain and what to delete during simplification we utilize dependency parses. We define deletable subtrees, that is subtrees that would not be part of the core arguments of a sentence, and delete one or more of them to generate shorter candidates.

The combination of multiple deletions lead to high number of candidates, making the task infeasible for parsing. To overcome this problem we select a subset of candidates to be parsed. We first repeat the experiments from [3] as our baseline ('OneSubtreeShorter' and '10Shortest + ShortestNoPunct' in Table 2) and then suggest the use of the parsability metric ('10WithoutLeastParsable'), which gives us an on par coverage, with more than double the average length of fully-parsed candidates. It shows that with a more informed metric, it is possible to get more complete analyses out of sentences that failed to have a full parse in their original form.

The work presented can be extended in several ways. Firstly, the problem of searching for a simplified candidate can be phrased as a learning problem. Secondly, we would like to make the program learn to paraphrase subtrees (including core parts of a sentence) before or instead of deleting them. Thirdly, a factorized representation for sentences so that subtrees causing problems can be identified and candidates containing them pruned might be a good idea. Finally, it is yet to be seen whether taking shortest possible candidates is an upper bound for what can be recovered after simplification/paraphrasing.

The source code and data are publicly available for research purposes. The project repository can be found at the following address: `https://gitlab.com/selimcan/DBSS`.

ACKNOWLEDGMENTS This research was funded by the German Research Foundation (DFG) grant SFB 732 “Incremental Specification in Context”, project D2.

REFERENCES

1. van Noord, G.: Error mining for wide-coverage grammar engineering. In Scott, D., Daelemans, W., Walker, M.A., eds.: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain., ACL (2004) 446–453
2. Brants, S., Dipper, S., Eisenberg, P., Hansen-Schirra, S., König, E., Lezius, W., Rohrer, C., Smith, G., Uszkoreit, H.: TIGER: Linguistic interpretation of a German corpus. Research on Language and Computation **2**(4) (2004) 597–620
3. Çetinoğlu, Ö., Zarrieß, S., Kuhn, J.: Dependency-based sentence simplification for increasing deep LFG parsing coverage. In: Proceedings of the LFG13 Conference. (2013) 191–211
4. Maxwell, J., Kaplan, R.: An efficient parser for LFG. In: Proceedings of LFG. Volume 96. (1996) 131
5. Bresnan, J., Asudeh, A., Toivonen, I., Wechsler, S.: Lexical-functional syntax. Volume 16. John Wiley & Sons (2015)
6. Seeker, W., Kuhn, J.: Making ellipses explicit in dependency conversion for a German treebank. In Calzolari, N., Choukri, K., Declerck, T., Dogan, M.U., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., eds.: Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012, European Language Resources Association (ELRA) (2012) 3132–3139
7. Rohrer, C., Forst, M.: Improving coverage and parsing quality of a large-scale LFG for German. In: Proceedings of the 2006 Language Resources and Evaluation Conference. (2006)
8. Riezler, S., King, T.H., Kaplan, R.M., Crouch, R.S., III, J.T.M., Johnson, M.: 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, July 6-12, 2002, Philadelphia, PA, USA., ACL (2002) 271–278
9. Dost, A., King, T.H.: Using large-scale parser output to guide grammar development. In: Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks, Association for Computational Linguistics (2009) 63–70
10. Cahill, A., Burke, M., O’Donovan, R., van Genabith, J., Way, A.: Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In Scott, D., Daelemans, W., Walker,

M.A., eds.: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain., ACL (2004) 319–326
11. Hautli, A., Çetinoglu, Ö., van Genabith, J.: Closing the gap between stochastic and hand-crafted LFG grammars. Proceedings of LFG10 (2010) 270–289
12. Riezler, S., King, T.H., Crouch, R.S., Zaenen, A.: Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-functional grammar. In: HLT-NAACL. (2003)
13. Crouch, R., King, T.H., Maxwell III, J.T., Riezler, S., Zaenen, A., Butt, M.: Exploiting f-structure input for sentence condensation. In: Proceedings of the LFG04 Conference. (2004) 167–187
14. Przepiórkowski, A., Wróblewska, A.: Supporting LFG parsing with dependency parsing. In: International Workshop on Treebanks and Linguistic Theories (TLT14). 168

ILNAR SALIMZIANOV
INSTITUTE FOR NATURAL LANGUAGE PROCESSING,
UNIVERSITY OF STUTTGART,
GERMANY
E-MAIL: <ILNAR.SALIMZIANOV@IMS.UNI-STUTTGART.DE>

ÖZLEM ÇETINOĞLU
INSTITUTE FOR NATURAL LANGUAGE PROCESSING,
UNIVERSITY OF STUTTGART,
GERMANY
E-MAIL: <OZLEM@IMS.UNI-STUTTGART.DE>