A Treebank for Grammatical Framework

Abstract

The goal of this paper is twofold, from one side we present the first large-scale evaluation of the coverage of the English Resource Grammar in Grammatical Framework (GF), and from another we report the creation of the first GF treebank. The evaluation is done on sections 2–21 from the UPenn Wall Street Journal treebank where we achieved coverage of 91.77% for the different syntactic constructions. As a side effect of the evaluation we got the first treebank for GF which is a translation of the Wall Street Journal treebank to abstract syntax trees for the resource grammar. The treebank will be used later for training of robust statistical parser, which in turn will let us to apply the techniques for high quality translation of controlled language in the framework to unrestricted text.

1 Introduction

Grammatical Framework (GF) is a grammar formalism (Ranta, 2004) which is mostly tailored for applications in natural language generation or for applications where the input text is in some restricted controlled language. In this application domain, the high precision is much highly valued than the wide coverage. For instance, an application that makes syntactic errors in the natural language generation would not be well received. Similarly, in the applications where the framework has been used for parsing, the demand is on the precision of the syntactic and semantic processing, while the coverage can be sacrificed.

The precision is achieved by the development of domain specific grammars for each particular application. However, the development of new grammars from scratch is a costly process, so the GF community has developed the resource grammars library (Ranta, 2009) for now more than twenty languages, which covers the most common syntactic constructions in the different languages. With the help of the resource grammars, the creation of new application grammars is very lightweight since they reuse the syntax from the resource library and just specialize it the particular domain.

Furthermore, the framework makes a distinction between abstract and concrete syntax. The abstract syntax is some language neutral domain representation (ontology) while the concrete syntax maps the abstract into some natural language. One and the same abstract syntax can be equipped with many concrete syntaxes which makes it possible to use it as an interlingua for bidirectional translation between the languages in the grammar. Indeed this is how it is used in application grammars since in this case the domain is limited and it is possible to design a common interlingua. In the resource grammars, however, the situation is different. Although all these grammars share common abstract syntax, it only serves as an ontology of the syntactic constructions that someone can find in the different languages. It is not even obligatory that all abstract constructions are covered in each concrete syntax. Direct translation via the abstract syntax is not recommended since it preserves the syntax of the sentence and not its semantics and pragmatics. Even the syntax preserving translation is possible only if we assume the existence of multilingual lexicon with common abstract syntax. Again the creation of such lexicon is possible on small scale but on large scale its existence is doubtful. Still the usage of

the common abstract syntax in the resource grammar saves a lot of work in the development of application grammars, since in this way the developers have to provide only domain specific lexicon and a few language specific functions when in one language the syntax on abstract level is different from the others.

Now the dichotomy between application grammars and resource grammars must be clear. The former guarantee high precision but are limited to small domains while the later are with wide coverage but with very imprecise semantics. In the first case we get interlingua based translation for free while for the second we need transfer in order to preserve the semantics. Traditionally GF was used for high quality application grammars but now we want to try it for processing of unrestricted text via the resource grammars. In this case we are willing to sacrifice some precision in exchange to wide coverage. For example, if we want to build translation system with wide coverage, then we must learn the transfer rules automatically which might fail in some cases due to data sparseness. Still by using the resource grammar we guarantee more fluent language since the possibility for many grammatical errors is simply eliminated.

Although the resource grammars are designed as libraries and not as grammars for parsing, Angelov (2009) showed that they are good for parsing too. Furthermore their empirical efficiency is linear although in theory it is polynomial with unknown exponent. Still the remaining problem is that these grammars are highly ambiguous which makes their usage impractical if they are not complemented with some statistical disambiguation model. The difficulty is that the training of such model requires manually annotated treebank which we do not have for GF. Fortunately it is possible to build such treebank by transforming an existing treebank such as the UPenn Wall Street Journal (WSJ) treebank (Marcus et al., 1993) to abstract syntax trees for the English Resource Grammar. Since the grammar by itself provides only syntactic constructions but not lexicon[1], we complemented the grammar with large monolingual lexicon derived from Oxford Advanced Learners Dictionary (Hornby, 1974)(Mitton, 1986). As a side effect of the transformation, we are also able to measure for a first time the coverage of the grammar on a large body of text.

This is not the first attempt to map the WSJ treebank to some linguistically established framework. For instance Clark and Curran (2007) learned a stochastic CCG from WSJ, and Miyao and Tsujii (2005) semi-automatically extracted HPSG grammar. As far as we know, however, Riezler et al. (2002) is the only one so far who used hand written grammar (LFG) for processing the WSJ treebank. In this sense, our work is more similar to Riezler et al. (2002) but the difference is that while his grammar is specifically tailored for English, we use grammar which is designed to be more cross lingual.

In the next section we compare some aspects of the annotation schema in the WSJ treebank with the linguistic model of the grammar, and we give some examples of transformation rules. Although our primary goal is to evaluate the grammar as it is and in the future to develop statistical disambiguation model for it, still we added some minor extensions to cover some very common constructions which we would miss otherwise. Those extensions are listed in Section 3. It is possible to add more extensions until we get full coverage of the treebank but since this will still not guarantee full coverage on unseen text, we do not consider this as a high priority. In Section 4, we evaluate the coverage of the extended grammar, and in Section 5, we make a conclusion.

2 Transforming WSJ to GF

The first major difference between the annotations in the treebank and an abstract tree from the grammar is that the former encode the parse tree of a sentence while the later is composed of function applications and represents the abstract syntax of the sentence.

The parse tree has the advantage to be very flexible since it contains the original words of the sentence plus annotations marking the phrases. In this representation, even completely random sequence of characters can be annotated as an English sentence. The abstract syntax tree, on the contrary, is very rigid since we cannot create new functions on demand, and if something in the sentence is not representable with a fixed set of functions, then the abstract syntax tree for the sentence can be only partial. At the same time, the abstract syntax tree is richer since the differ-

[1]There is only a small lexicon with about 300 words that is used for testing.

ent functions encode word lemmas, grammatical and morphological features like tense and number and even syntactic combinators for composition of new phrases from other sub-phrases. The text of the sentence itself is not present in the abstract tree since it is compositionally computed by applying the functions in the nodes to the result from the evaluation of the child nodes. For instance in the tree on Figure 1, the function *PredVP* is applied to two arguments. The first represents the subject "BELL INDUSTRIES Inc.", and the second represents the rest of the sentence, i.e. the verb phrase. When the children are computed, they are evaluated to records of strings and feature values, which are after that combined by *PredVP* to get the representation of the whole sentence. The parsing process is the opposite, i.e. we search for an abstract syntax tree which is computable to the given sentence. If the sentence is not in the coverage of the grammar, then we have no other choice but to construct a partial abstract tree which covers only the phrases which can be interpreted by the grammar. In exchange, we retain the generative capabilities of the grammar which is a key feature when the grammar is used for machine translation.

The GF engine has service which is able to convert every abstract tree to a corresponding parse tree but this tree is usually far too different from the tree in the treebank. Since most of the functions in the abstract tree are unary or binary, the corresponding parse tree is quite dense. For comparison, the trees in the treebank are too shallow, i.e. many intermediate phrases are omitted, and as a result there are many phrases with more than two children. This shallow representation is a known problem which leads to data sparseness. For instance Collins (2003) proposed a method which automatically breaks each phrase into a sequence of binary nodes. In our case, we want to preserve the binarization that is already present in the abstract syntax. This means that we must reparse the treebank with the resource grammar, and the only usage of the annotations in the treebank is to resolve the ambiguities.

However, we did not use the original resource grammar directly for parsing because in general the theories behind the WSJ annotations and behind the grammar are not compatible. Changing the grammar to respect the incompatible annotations would be too ad hoc solution. Instead we developed another grammar by using parser combinators (Hutton, 1992)[2] in Haskell (Peyton Jones, 2003). This new transformation grammar follows the WSJ theory when parsing but as output produces abstract trees which are compatible with the resource grammar.

The usage of parser combinators or definite clause grammars is nowadays not so common but in our case it worked well since we parse annotated sentences which eliminates the ambiguities. We also managed to avoid left recursion by restructuring the grammar. The usage of parser combinators, however, gave us the robustness that we need for the transformation of phrases that are otherwise not covered by the resource grammar.

We achieve the robustness by parsing the annotations level by level instead of parsing the plain sentence. If we take as an example the sentence on Figure 1, then the structure at the top level is:

```
(S (NP ...) (VP ...))
```

which we can immediately map into the abstract syntax tree:

$$\mathit{UseCl}\ t\ p\ (\mathit{PredVP}\ np\ vp)$$

Here *np* and *vp* are variables which correspond to the outcome from the conversion of the child phrases `NP` and `VP`, and *PredVP* is a function which builds a clause from the children. Still the produced clause does not have fixed tense and polarity but they are fixed by the first and the second arguments of function *UseCl*. The tense and the polarity are features that can be extracted from the verb phrase so here they are represented as two more variables t and p. If we are not able to find matching abstract syntax tree for some level, then we simply convert all children and combine them into one phrase with a placeholder, i.e. a symbol indicating that the sub-phrases are related in an unknown way. For instance the grammar does not have a rule for combining two NP phrases into another NP phrase but still the conversion is able to produce:

$$\begin{array}{l}(?34\ (\mathit{DetCN}\ (\mathit{DetQuant}\ \mathit{IndefArt}\ \ldots)\\ \qquad\qquad (\mathit{UseN}\ \mathit{cent\_N}))\\ \qquad (\mathit{DetCN}\ (\mathit{DetQuant}\ \mathit{IndefArt}\ \mathit{NumSg})\\ \qquad\qquad (\mathit{UseN}\ \mathit{share\_N})))\end{array}$$

[2] In the functional programming, the parser combinators play the same role as the use of definite clause grammars in Prolog.

```
(S
  (NP-SBJ (NNP BELL) (NNP INDUSTRIES) (NNP Inc.) )
  (VP (VBD increased)
    (NP (PRPS its) (NN quarterly) )
    (PP-DIR (TO to)
      (NP (CD 10) (NNS cents) ))
    (PP-DIR (IN from)
      (NP
        (NP (CD seven) (NNS cents) )
        (NP-ADV (DT a) (NN share) )))))
```

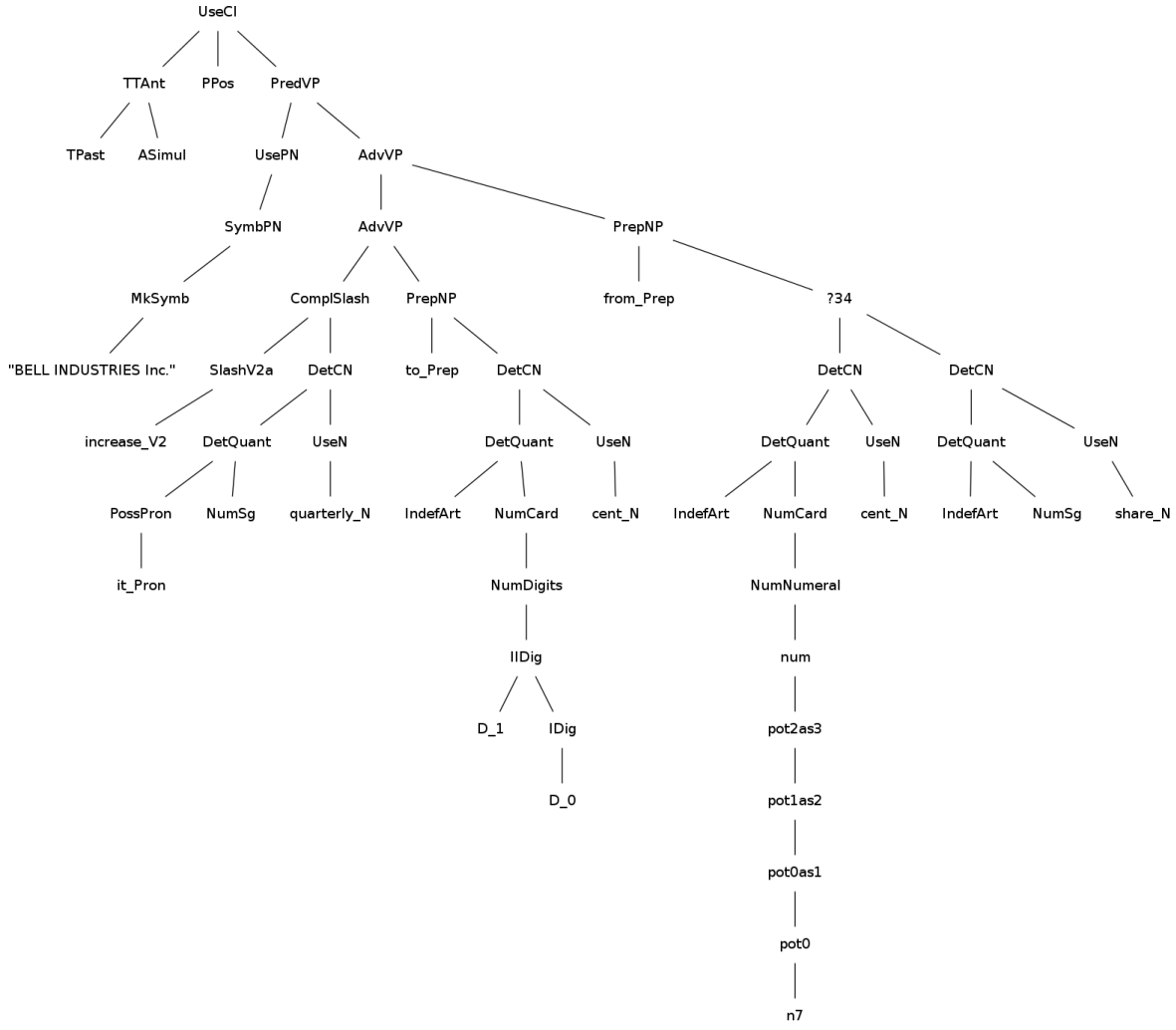

Figure 1: An example WSJ treebank tree (above) and the corresponding abstract syntax tree in the English Resource Grammar (below)

for the phrase “seven cents a share”, i.e. it converts each of the embedded phrases and combines them into one phrase by leaving the placeholder ?34 for the missing abstract function. The new phrase is further on embedded in the abstract syntax for the prepositional phrase “from seven cents a share”.

The traditional parser combinators are used for parsing sequences of tokens and in our case we have tree structures, but still if we are only interested in the children of one particular node, then since the children are ordered we can use traditional parsing techniques. For instance the transformation for a sentence composed of one NP and one VP phrase can be encoded as the following Haskell code:

```
"S" :-> do np <- cat "NP"
           vp <- cat "VP"
           return (PredVP np vp)
```

Here the operator :-> assigns some rule to the category S and every time when we see the same category in the treebank, we will fire the same rule. The rule processes the children of the node and combines the results by using the return combinator. In this case, the processing consists of calling the combinator cat twice, once for "NP" and once for "VP". Each time the cat combinator checks that the current child has the corresponding category and applies recursively the corresponding transformation rule for it. If the application of the rule is successful, then cat returns the output from the rule. If it is not, then the result is a placeholder applied to the transformation of the grandchildren.

This first example is oversimplified and it does not work even in this simple case because it does not let us to extract the tense and the polarity of the sentence. The solution is to refactor the rule into:

```
"S" :-> do
      np <- cat "NP"
      (t,p,vp) <- inside "VP" pVP
      return (UseCl t p (PredVP np vp))
```

Now we have a new combinator called inside which also checks that the current child is "VP" but instead of calling the generic rule for transformation of verb phrases, it calls the custom rule pVP which returns both the transformed phrase and its tense and polarity. While if we were using only cat, our transformation rules will be equivalent to context-free grammar, with the introduction of inside we get both some context-sensitivity and the ability to go deeper in the nested levels of the parse tree.

So far we have used inside as a way to get more information for the phrase, but we can also use it when the annotation structure of the treebank is incompatible with the abstract syntax of the resource grammar. For instance the grammar does not have SBAR category like in the treebank, and we must do the modelling differently. For example the verb said takes as object another sentence but in the treebank the object is annotated as a sentence wrapped in SBAR phrase:

```
(S (CC But)
  (NP-SBJ (NNP Mr.) (NNP Lane) )
  (VP (VBD said)
    (SBAR (IN that)
      (S
        .....
                ))))
```

Still we can do the transformation by using a rule like:

```
"VP" :-> do v <- pV "VS"
            s <- inside "SBAR"
                   (do inside "IN"
                         (word "that")
                       cat "S")
            return (ComplVS v s)
```

where we go inside the SBAR annotation and we look for an S annotation which is processed by using cat. The rule also shows the usage of the combinator word, which checks for the presence of specific words.

Here we called the custom rule pV instead of calling the generic rule for verbs, because we need to select the verb with the right valency which is controlled by the argument VS. Currently the transformation rules recognize the following verb types from the resource grammar:

V	intransitive verb
V2	transitive verb
VV	verb phrase as object
VA	adjectival phrase as object
VS	sentence as object
V2V	a noun phrase and a verb phrase as object

and the only way to distinguish the different types is to look in the context. If for instance the object is a sentence, then in the abstract syntax we should use the lexical constant:

fun *say_VS* : *VS*;

instead of *say_V2* or some other valency type. Note that currently we treat as verb complements only the direct objects in the sentence, while the indirect objects are attached as modifiers. In order to have more fine grained distinction between complements and modifiers, we will need a lexicon with more detailed categorization for the verbs.

The selection of the lexical constants itself is done by using the combinator *lemma*. The complication here is that while in the treebank we have the word form, in the abstract tree we must place a function which corresponds to the lemma of the word. The linearization of the function is a record which enumerates all inflection forms. For instance the linearization of the function *say_VS* is the record:

```
s VInf      : say
s VPres     : says
s VPPart    : said
s VPresPart : saying
s VPast     : said
```

The combinator *lemma* takes as arguments an abstract category and the name of a constituent in the record type for the given category and searches for function which has the current word in the specified constituent. For instance the word `said` is processed by the call:

lemma "VS" "s VPast"

It will search in all constants of type *VS* and will find that *say_VS* has the given word in the right place in its linearization record.

A rough mapping of the part of speech tags to pairs of category and constituent name is given in Table 1. As it can be seen the verbs are not the only exception and in general the relation is many to many. The only way to find the right matching is to take into account the context. For instance the tag `IN` corresponds to both prepositions (*Prep*) and numeral-modifying adverbs (*AdN*) like `about`. Still the disambiguation is possible since the adverbs are always used before numerals and the whole numeral phrase is embedded in the annotation `QP` for quantifier phrases. Another interesting exception is when a verb is used as a noun or as an adjective. While in the treebank, in this case, the verb is retagged as `NN` or `JJ` correspondingly, in the grammar we have syntactic functions which build a noun from the present participle and an adjectival phrase from the present or past participle. In this way, for instance `proposed` can be used as an adjective and `reporting` can be used as both noun and adjective.

There is a similar situation with the adverbial tag `RB`. Every adjective in the resource grammar includes in its inflection table its adverbial form and in this way for instance the adverb `historically` can be generated from the adjective *historical_A* by applying the function

POS	category	constituent
IN	Prep	s
IN	AdN	s
NN	N	s Sg Nom
NN	V	s VPresPart
NN	NP	s (NCase Nom)
NNS	N	s Pl Nom
PRP	Pron	s (NCase Nom)
PRP	Pron	s NPAcc
PRP$	Pron	s (NCase Gen)
RB	A	s AAdv
RB	Adv	s
RB	AdA	s
RB	AdV	s
JJ	A	s (AAdj Posit Nom)
JJ	V	s VPresPart
JJ	V2	s VPPart
JJR	A	s (AAdj Compar Nom)
JJS	A	s (AAdj Superl Nom)
VB	V,V2,...	s VInf
VBD	V,V2,...	s VPast
VBG	V,V2,...	s VPresPart
VBN	V,V2,...	s VPPart
VBP	V,V2,...	s VInf
VBZ	V,V2,...	s VPres
MD	VV	s (VVF VPres)
MD	VV	s (VVF VPast)
PDT	Predet	s
WP	RP	s (RC Masc (NCase Nom))
WP	RP	s (RC Masc NPAcc)
WDT	RP	s (RC Neutr (NCase Nom))
WP$	RP	s (RC Masc (NCase Gen))
DT	Quant	s False Sg
DT	Quant	s False Pl
DT	Det	s

Table 1: Rough equivalence table between part of speech tags in the treebank and abstract categories from the grammar

PositAdvAdj. Still there are adverbs which are not derived from an adjective and in this case they are simply listed in the lexicon as adverbs (*Adv*). There is also a category of adverbs which can modify only adjectives (*AdA*, i.e. like `very`) or only verb phrases (*AdV*, like `also`). Those are also tagged with `RB` but since they form disjoint set, it is easy to distinguish them. The tag *RB* is also abused for tagging the word `not` or its contraction `n't`. This word does not have its own category in the grammar, and we generate it syntactically when a clause is linearized with negative polarity. The transformation rules for the verb phrase check for the presence of `not` and return either positive or negative polarity. The polarity is further on used when a clause is converted to a sentence.

Finally, some words that are tagged with `NN` in the treebank are complete noun phrases in the grammar. Typical examples are words like `someone` and `everyone`, i.e. this are words that are quantifiers by themself and cannot be used in combination with other quantifiers. In this case, we skip the `NP` annotation around the noun, and we return a single constant which is already a noun phrase, i.e. *someone_NP*, *everyone_NP*, etc.

Another special handling is needed for proper names. In the treebank, they are encoded as sequence of words tagged with `NNP` or `NNPS` while the grammar representation is a plain string which is lifted to the category *PN*. For instance the name `Mr. Lane` is represented as:

SymbPN (*MkSymb* "Mr. Lane")

Here the function *MkSymb* converts the string to the category *Symb* which is further converted to *PN* by the function *SymbPN*. Furthermore, every proper name can be lifted to a noun phrase by applying the function *UsePN*. The grammar itself does not have any rules for named entity recognition, so when it is used for parsing it must be complemented with a specialized recognizer.

There are few more combinators which allow flexible control over the tree transformation. For instance, the combination `r1 `mplus` r2` allows us to try the rule `r1` first, and if it fails to continue with `r2`. The combinations `many r` and `many1 r` act similarly to the Kleene star and plus operators, i.e. they repeat the application of rule `r`, and they return as result a list with the outputs produced from `r`. Similarly `opt r x` makes the application of rule `r` optional. If the rule is applicable, then the combinator returns the output from `r`, and if it is not applicable then the result is the value `x`. With the help of the control combinators and the combinators that we described earlier, we managed to encode all transformation rules in about 800 lines of Haskell code.

3 Grammar Extensions

Although our primary goal was not to extend the grammar, we found that there are few very common patterns that were not covered, and if we implement them this will improve the overall coverage of the grammar.

For instance in the resource grammar the copula (the verb to be) is treated separately from the other verbs, and we found that some of its valencies are missing. The grammar only supports the cases where the object is either adjective phrase, noun phrase, adverbial phrase or a common noun, but we also found many cases where the object is a sentence or a verb phrase in infinitive. These two cases were covered by adding two more functions: *CompS* and *CompVP*.

Another common case is that when the object in the sentence is a nested sentence, then it is usually topicalized and moved to the front. The subject and the verb are left at the end of the main sentence and separated from the nested sentence with a comma. Fortunately since the object in the verb phrase is implemented as a discontinuous constituent, it was easy to implement the topicalization by providing another function *ComplPredVP*. When the new function is used instead of *PredVP*, then the object is moved to the front.

A third example is the usage of reflexive pronouns like `myself`, `itself`, etc. The grammar supports the usage of reflexives only in the limited case where they are directly objects of a verb, but there are examples where this is not the case, for instance:

I did not feel very good about myself

The solution was to add the reflexive pronouns directly as noun phrases which let us to use them more freely.

Perhaps most extensions, however, were in the lexicon. One of the main shortcomings of the

original lexicon was that it did have only limited information about the verb valency. Fortunately we were able to extend it with more verb valencies by looking at the context in which every verb appears, and we used the information to extend the lexicon. In the process, we also discovered that some words had imprecise categories, for instance *Adv* instead of *AdV*, *AdN* or *AdA*. We also removed entries which were incompatible with the grammar. For instance the definitive article `the` was listed as adverb, and all numerals were listed as both nouns and adjectives although we already had the numerals as functions in the grammar. The explanation for this inconsistency is that the lexicon is imported from external resource which is not entirely compatible with the resource grammar.

4 Evaluation of the Coverage

We measure the coverage of the grammar as the percentage of nodes in the abstract trees that are filled in with function symbols instead of placeholders. The statistics are that in average 91.77% of the nodes are filled in with functions, and the remaining 8.23% are placeholders. As a whole, out of 39832 trees, 1633 trees were fully converted, 4668 contained only one placeholder, 4803 had two placeholders, and the rest had more than two. In average there are 4.91 placeholders per tree.

We looked at some of the incomplete trees, and we identified three main reasons for incomplete matching. In many cases, simply the grammar did not have the corresponding syntactic rule. In other cases, there were inconsistencies in the treebank, and although the conversion would be possible in principle, our automatic translation simply failed. We fixed some of the inconsistencies by adding extra rules in the transformation, but this is not always possible. One trivial example is the word `many` which is sometimes annotated as adjective `JJ`, sometimes as adverb `RB`, and only occasionally as a determiner `DT`. In the resource grammar, it is always considered to be a determiner, and we fixed this case by adding special rules for `many`. We also noticed other cases of inconsistencies but since they are more rare and less regular we did not try to fix all of them. Finally, we are aware that we might have missed some transformation patterns just because they are rare and we have not noticed it. It is difficult to evaluate the impact of the different factors because this would mean that we have to check all trees manually which will take a long time.

5 Conclusion

The grammar evaluation is very positive. Although the grammar was never designed for wide coverage parsing, we found that it actually has very good coverage. For full parsing, however, we must make the existing GF parser more robust. The evaluation tells us that in average the parser will have to do five guess steps per sentence (the number of placeholders), and this is a good estimation of the amount of robustness that we need. As a side effect of the evaluation we also built a partial translation of the WSJ treebank to GF. The translated treebank is going to be used for training of statistical disambiguation models for the otherwise too ambiguous grammar.

Since we already have the transformation rules, in fact, we already have a way for robust parsing with a wide coverage. We can use any existing statistical parser like the Stanford parser or the Collins parser, and then we can convert the output to an abstract syntax tree. This is not completely satisfactory, however, since in this way we bypass the grammar completely, and we cannot take advantage of it. By using the more detailed information in the grammar, we can potentially achieve better precision. Still parsing via transformation can be used as a way to evaluate the precision and the recall of the future GF parsers.

References

Krasimir Angelov. 2009. Incremental parsing with parallel multiple context-free grammars. In *European Chapter of the Association for Computational Linguistics*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Comput. Linguist.*, 33:493–552, December.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637, December.

Albert Sydney Hornby. 1974. *Oxford Advanced Learner's Dictionary of Current English, Third Edition*. Oxford University Press.

Graham Hutton. 1992. Higher-order functions for parsing. *Journal of Functional Programming*, 2(3):323–343, July.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330, June.

Roger Mitton. 1986. A partial dictionary of English in computer-usable form. *Literary & Linguistic Computing*, 1:214–215, December.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 83–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

Simon Peyton Jones. 2003. *Haskell 98 Language and Libraries: the Revised Report*. Cambridge University Press.

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

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Iii Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th meeting of the ACL*, pages 271–278.