How Much do Grammars Leak?

ABSTRACT

We present a large-scale evaluation for the coverage of the English Resource Grammar developed in Grammatical Framework (GF) on the Penn Treebank. The English Resource Grammar is a wide-coverage linguistic grammar which was developed independently from the treebank, and for a first time we do a quantitative analysis of its coverage. We measured a coverage of 94.47%, and we identified the main syntactic structures where the grammar leaks. As a side effect of the evaluation, we built a treebank for the grammar by translating the Penn Treebank to abstract syntax trees. The treebank is used in an ongoing project for training of stochastic disambiguation models for the grammar.

KEYWORDS: Grammars, Treebanks, GF.

1 Introduction

One of the famous quotes attributed to Edward Sapir is "All grammars leak". Although this is a generally accepted statement, it is hard to find quantitative evaluations of existing grammars. In our opinion, it must be very hard to capture all possible phenomena appearing in a natural language, but it is a reasonable task to capture the most frequent ones. Since the natural language is governed by the Zipf law, we could expect that in this way we can achieve very good coverage at low cost. The Zipf law is also in favour of all data driven approaches to natural language processing, since given a fixed amount of training material it makes it possible to learn the most common constructions out of the "infinite" set of varieties. The key success factor in both cases is not what kind of processing is used, i.e. a linguistic grammar or a statistical model, but how much the model is robust towards unknown language patterns.

In Grammatical Framework (GF) (Ranta, 2004), the focus has always been on applications which demand high quality and deep understanding of the natural language statements. For instance, an application that makes grammatical errors in natural language generation will not be well received. Similarly, in applications where the framework is used for parsing, the demand is on the precision of the syntactic and semantic processing, while the coverage can be sacrificed. Such requirements are best satisfied with an application specific grammar which is tailored to the particular domain. 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 more than twenty languages. The grammars cover the most common syntactic constructions of the different languages and this makes the creation of new application grammars very lightweight. They simply reuse the syntax from the resource library and specialize it to the target domain. The focus of our work is the evaluation of the English Resource Grammar on the Penn Treebank (Marcus et al., 1993).

The resource grammar provides a collection of syntactic constructions but it does not come with its own lexicon. Typically every application has its own domain specific dictionary. In our case we want to work with an existing corpus, so we needed a lexicon with large coverage. We bootstrapped it from the computer usable version of the Oxford Advanced Learner's Dictionary (Mitton, 1986), and we adapted it to the grammar.

Since the grammar has been developed independently from the treebank, it is not biased in any way to the domain. In fact the design goals of the grammar were quite different. Rather than striving to achieve full coverage for naturally occurring text, its focus was to provide a set of syntactic constructions which makes it possible to express any thought in one or another way. It evolved naturally in different projects, and our experience is that now it is possible to express anything but not in any possible way. If a random sentence from the Penn Treebank has to be rendered with the grammar, it will be possible but it might involve some paraphrasing. Still, our expectation was that the grammar already captures the most common constructions. Indeed we measured a coverage of about 94.46%.

Although the resource grammars are designed to be used as libraries and not as grammars for parsing, (Angelov, 2009) showed that they are good for parsing too. Furthermore, it was known that the theoretical complexity of the formalism is polynomial with a degree dependent on the grammar (Ljunglöf, 2004), but Angelov showed empirically that for the resource grammars the complexity is close to linear. This suggests that the grammars can in principle be used for parsing, although there are still few remaining problems. First of all it is not clear what is the coverage of the grammars (or for the English one in the current work). Another question is how to deal with situations when the grammar is leaking. Finally, these grammars are highly ambiguous which makes their usage

impractical, if they are not complemented with some statistical disambiguation model. The model is also needed for choosing the best chunking, if a partial parsing for out of grammar sentences is desired. The best solution for these problems is outside the scope of the current paper, but we opened the way for further research by converting the Penn Treebank to partial GF trees as part of the evaluation. The tree conversion is the first step in the evaluation. As a second step, we measured how complete are the trees.

The produced treebank is already used in an ongoing project for training of stochastic disambiguation models for the grammar. Since the underlying execution model for GF is based on Parallel Multiple Context-Free Grammars, the results are also relevant for other projects involving non context-free grammars. After further cleanup of the treebank, we plan to make it available for other researchers.

This is not the first attempt to map the Penn Treebank to some linguistically established framework. For instance, (Hockenmaier and Steedman, 2007) converted the treebank to CCG and later Clark and Curran (2007) used it for learning statistical models. Similarly (Miyao et al., 2004) semi-automatically extracted an HPSG corpus which was used for training statistical parsers (Miyao and Tsujii, 2005). The Penn Treebank has been annotated with F-structures (Cahill et al., 2002) which led to building large coverage LFG lexicons (O'Donovan et al., 2005). There is also a lot of research about using the treebank withing the LTAG formalism (Xia (1999),Xia et al. (2000), Xia (2001),Shen and Joshi (2005) and Chen et al. (2006)). Neither of this approaches, however, tried to apply an existing grammar. Although we could apply similar techniques for extracting a GF compatible grammar from the treebank, this is not what we want. The grammars in the Resource Library are designed with a shared abstract representation of the syntax, which is such that it makes it easier to translate from one language to another with a minimal transfer. Any automatically extracted grammar is likely to break this abstraction which will make the library less useful in multilingual applications. As far as we know, Riezler et al. (2002) is the only one so far who used hand written grammar (LFG) for processing the Penn Treebank. In this sense, our work is more similar to Riezler et al. (2002). The difference is that we are interested in measuring the coverage of the grammar and in the creation of a GF compatible treebank, while Riezler focused on learning discriminative model for the LFG grammar directly from the original treebank. It is interesting that Xia et al. (2000) reports about the evaluation of the XTAG grammar (The XTAG-Group, 1998) on the automatically converted LTAG treebank. Their evaluation procedure is different but they also reported coverage of 97.2% which is close to our result. Unfortunately, the direct comparison of the numbers is difficult due to the differences in formalisms.

In the next section we give a very brief introduction into the GF formalism. After that in Section 3 we compare some aspects of the annotation schema in Penn Treebank with the linguistic model of the English Resource Grammar, and we give some examples of transformation rules. Section 4 focuses on the implementation. Although our primary goal is to evaluate the grammar as it is, we still added some extensions to cover the most common constructions which we would miss otherwise. Those extensions are listed in Section 5. It is possible to add more extensions until we get a full coverage of the treebank but this will still not guarantee full coverage on unseen text, so we did not consider this as a high priority. In Section 6, we evaluate the coverage of the extended grammar, and in Section 7, we make a conclusion.

2 Grammatical Framework

The most distinguishing feature of GF is its separation between abstract and concrete syntax. The abstract syntax is a set of function signatures which describes an abstract representation of the language, while the concrete syntax is the actual implementation of these functions. They take

as arguments records of strings and grammatical parameters (gender, number, etc.) and produce another record for the target category. Since the functions are restricted to be non-recursive, the expressive power of the formalism is equivalent to Parallel Multiple Context-Free Grammar (Seki et al., 1991). The output from the GF parser and the input to the language generator in GF is a tree of function applications. When the tree for a sentence is computed the result of the computation is the string for the sentence. Parsing proceeds in the opposite direction, we search for an abstract syntax tree which is computable to the given sentence.

For illustration, Figure 1 shows a sentence from Penn Treebank which is converted to an abstract tree. Here 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, which are further combined by `PredVP` to get the representation for the whole sentence. The signatures in the abstract syntax of the resource grammar assign types for the functions. For instance the definition:

```
fun PredVP : NP -> VP -> Cl
```

Specify that PredVP takes two arguments – one noun phrase and one verb phrase and returns a clause. Further on the concrete syntax will specify that `NP` is mapped to some kind of record structure. The concrete definition of `PredVP` in a highly simplified version of English would be:

```
lin PredVP np vp = {s = np.s ++ vp.s}
```

i.e. we just concatenate the strings in the fields `s` for the subject and the verb phrase, to compute a new string which is put in the field `s` for the clause.

It is possible to have more than one concrete syntax for the same abstract syntax, and this is how the application grammars in GF are used in translation – the abstract syntax serves as an interlingua. In fact even the grammars in the resource library share the same abstract syntax, but in this case it is used more as a common API and not as a real translation interlingua. Since we want to preserve the abstract syntax for the library, we cannot automatically extract the grammar from a treebank.

3 Transforming Penn Treebank to GF

The major difference between the annotations in the treebank and the abstract trees from the grammar is that the former encode the parse tree of a sentence, while the later are trees of function applications. 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 (erroneously) as an English sentence. The abstract 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 tree for the sentence can be only partial. At the same time, the abstract tree is richer since the different functions encode word lemmas, grammatical and morphological features such as tense and number and even syntactic combinators for composition of new phrases from other sub-phrases.

The GF engine has a 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. On the contrary, the trees in the treebank are too shallow, i.e. many intermediate phrases are omitted, and as a result there

```
(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) )))))
```

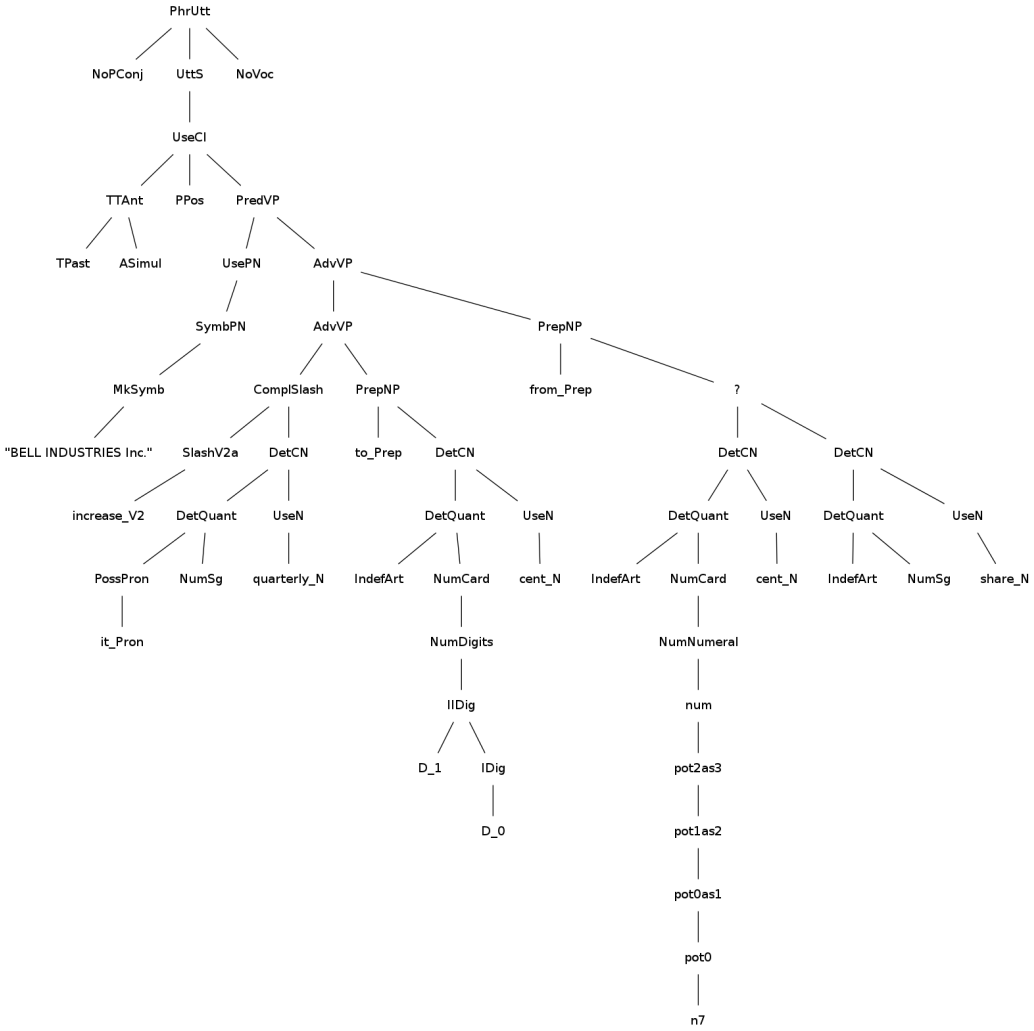

Figure 1: An example Penn Treebank tree (above) and the corresponding abstract tree in the English Resource Grammar (below)

are many phrases with more than two children. This shallow representation is a known problem which leads to data sparseness. This led, for instance, (Collins, 1997) to propose 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 somehow re-parse the treebank with the resource grammar, and the only use for the annotations in the treebank is for ambiguity resolution.

Unfortunately we cannot simply parse the treebank with the ordinary GF parser since the grammar has only partial coverage. Instead, we developed a set of patterns which convert the annotations level by level. This gives us both robustness and an automatic disambiguation for free. For instance, if we take the sentence on Figure 1, then the structure at the top level:

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

can be map into the abstract syntax tree:

```
UseCl t p (PredVP np vp)
```

Here `np` and `vp` are variables which correspond to the outcome from the conversion of the child phrases `NP` and `VP`. Function `PredVP` builds a clause from the children, where a clause is a sentence with a variable tense and polarity. The tense and the polarity are not fixed until the clause is not converted to the sentence category `S` by function `UseCl`. In the example, the tense and the polarity are shown as the variables `t` and `p` since they are also extracted from the verb phrase. The implementation of the pattern looks inside the content of the verb phrase and extracts the triple `(t,p,vp)` from it.

The developed transformation patterns cover the most common syntactic constructions but it is always possible to find a case where none of the patterns match. In this case we make the conversion robust by simply converting all children and gluing them into one phrase with a placeholder which indicates 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 for the tree on Figure 1 is able to produce:

```
(? (DetCN (DetQuant IndefArt ...)
          (UseN cent_N))
   (DetCN (DetQuant IndefArt NumSg)
          (UseN share_N)))
```

for the phrase “seven cents a share”. It converts each of the embedded phrases and combines them by leaving the placeholder ? 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”. Note, we know that the second argument of function `PrepNP` is of type `NP`, and we know that function `DetCN` returns `NP`. From this we can derive that the phrase with the placeholder is a noun phrase containing two other noun phrases. In this sense, we do not loose any information during the conversion.

In general the transformation patterns are not compositional since it is not always possible to directly combine the children of a phrase into a larger abstract tree. This holds even on the lexical level. Table 1 shows the mapping between the parts of speech in the treebank and the corresponding lexical

categories. As it can be seen there is a many-to-many relation, which means that the patterns must look into the context in order to reconstruct the right category.

A notable example are the ordinals which in the treebank are tagged as adjectives, but in the grammar they are modelled as a separate category. This means that the transformation must involve not only retagging but also some transformations of the abstract tree. If the ordinals were not handled specifically, the abstract tree for the phrase "the first presentation" would be:

```
(DetCN (DetQuant DefArt NumSg) (AdjCN (PositA ?) (UseN presentation_N)))
```

This is wrong and the tree must be changed to:

```
(DetCN (DetQuantOrd DefArt NumSg ?) (UseN presentation_N))
```

here in both cases the question mark is the slot for the ordinal. A similar situation arises with the determiner "many" which in the treebank is sometimes annotated as adjective (`JJ`), sometimes as adverb (`RB`) and only occasionally as determiner (`DT`). Similarly, some predeterminers like "such" are sometimes tagged as adjectives and sometimes as `PDT`, which we must correct too. Yet another instance of this is that the tag `RB`, which in principle denotes adverbs, is also abused for tagging the word "not" or its contraction "n't". This words does not have their own category in the grammar, and they are generated syntactically when a clause is linearized with a negative polarity. The transformations capture such cases and always turn them into the representation that is determined by the grammar. The change, however, involves transformations of the tree too.

Figure 1 shows one more key difference between the Penn Treebank and the grammar trees. While the leaves in the treebank trees contain words, the leaves in the abstract trees contain function names which are a combination of lemma and category. When this terminal functions are evaluated, they compute a full-form inflection table. For instance function `increase_V2` computes the table:

```
s VInf      : increase
s VPres     : increases
s VPPart    : increased
s VPresPart : increasing
s VPast     : increased
```

The inflection forms are used by the syntactic functions on the upper levels for the formation of phrases. Since the part of speech tags in the treebank characterize a word and the leaves in the abstract tree correspond to inflection tables, Table 1 maps both a category name and a constituent name to a single part of speech tag.

An important information that is omitted in the part of speech tag is the valency for the verbs. While in a treebank, the valency is recoverable from the context, the grammar needs an explicit record of the valency for every verb. The lexicon that we had bootstrapped did not have any valency information, so we had to extract it from the treebank. For every verb we collected the set of valency types and we enriched the lexicon with them. Currently we distinguish between the following types:

V	intransitive verb
V2	transitive verb
V2A	transitive verb with an additional adjectival phrase as object
V2V	transitive verb with an additional verb phrase as object
V2Q	transitive verb with an additional embedded question as object
V2S	transitive verb with an additional embedded sentence as object
V3	ditransitive verb
VV	a verb which is complemented with another verb phrase
VA	a verb which is complemented with an adjectival phrase
VS	a verb which which takes an embedded sentence as object
VQ	a verb which which takes an embedded question as object

This is actually the set of all verb categories in the grammar, and as Table 1 shows, they all map to the same parts of speech. One shortcoming in the current conversion is that we cannot distinguish between adverbial modifiers and indirect objects since this is not marked in the treebank. In a future version of the conversion, we can try to recover this information by consulting publicly available resources such as VerbNet (Kipper et al., 2000). Another related feature that must be considered is the detection of phrasal verbs. Since our lexicon contains only few phrasal verbs, currently verb phrases with phrasal verbs are only partly converted.

The valency frame reconstruction is yet another example of non-compositional transformation. For verbs with more than two arguments, the objects are grouped in a covering phrase. For instance the objects of a V2V type verb like “permit”:

```
(S
   (NP-SBJ (PRP they) )
   (VP (VBP permit)
      (S
         (NP-SBJ (NN portfolio) (NNS managers) )
         (VP (TO to)
            (VP (VB retain)
               (NP (RB relatively) (JJR higher) (NNS rates) )
               (PP-TMP (IN for)
                  (NP (DT a) (JJR longer) (NN period) )))))))
```

are embedded in a `S` phrase, but this is not a sentence in our grammar. In this case we must split the phrase into a noun phrase and a verb phrase which we attach to the verb with the appropriate grammar functions:

```
(ComplSlash (SlashV2V permit_V2V
                      (.. to retain ..))
            (.. portfolio managers ..))
```

The conversion is further complicated by the need to detect empty elements which is necessary for the proper handling of passive voice, relative clauses and other cases.

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.

Marcus et al. (1993) mentioned few cases when lexically recoverable information was stripped from the part of speech tag. For instance the `HV`, `BE` and `DO` tags from the original Brown tag set were merged with the `VB` tag. In the resource grammar verbs like “have”, “be” and “do”, are handled differently when they are used as markers for tense and voice. For the cases when “have” and “do” are used as main verbs, the lexicon

provides functions like `have_V2` and `do_V2` but when they are used as markers, the corresponding words are generated syntactically as part of the clause. Similarly, the verb "be" is always generated syntactically either on the clause level or from the function for passive voice `PassVPSlash`. In that sense, the handling of this auxiliary verbs in the grammar is closer to the model taken in the Brown corpus.

In a similar fashion, we had to split into different categories the reflexive pronouns from the personal pronouns which in the treebank are merged into the `PRP` tag. It is exactly the same situation with the subordinating conjunctions, the prepositions and some of the numeral-modifying adverbs (like `about_AdN`) which are both tagged as `IN`.

There are also some words that are tagged with `NN` in the treebank, but are complete noun phrases in the grammar. Typical examples are words like "someone" and "everyone", i.e. these are words that are quantifiers by themselves 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 phrase, i.e. `someone_NP`, `everyone_NP`, etc.

The transformation also needs a named entity recognizer since the names in the abstract trees must be extracted as literals. 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 (`PN`) can be lifted to a noun phrase by applying the function `UsePN`. We look for sequences of `NNP` and `NNPS` tags in the treebank, and we convert those to names in the abstract syntax.

As a summary, with the help of the transformation patterns, a large fraction of the annotations were converted to function applications. Apart from the cases where the grammar is just missing an appropriate syntactic construction, there are two more cases when the transformation might fail. First of all we noticed that the treebank annotations are not always absolutely correct which leads to a failure in the conversion. Another cause of failures is that if the conversion fails for whatever reason, the failure might propagate due to the noncompositional character of the patterns. If a pattern does not have to look inside a child phrase, then a failure in the embedded phrase does not propagate to the parent. However, if the pattern needs to look inside, it might fail completely. We manually edited some of the sentences, and in Section 6, we report both the plain automatic coverage, and the coverage after the changes.

4 Implementation

While in the previous section we highlighted some of the difficulties that we had to handle in the transformation, in this section we give an overview of our Haskell (Jones, 2003) implementation. The transformation patterns are implemented as rules build with the help of parsing combinators (Hutton, 1992)[1]. The usage of parser combinators and definite clause grammars is nowadays not so common for natural language processing 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 transformation patterns. The usage of parser combinators gave us the robustness that we need for the transformation of phrases that are otherwise not covered by the resource grammar. The source code for the implementation is available for other researchers at:

```
http://www.grammaticalframework.org/examples/PennTreebank/
```

Traditionally 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 they are ordered we can use

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

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
JJ	Ord	s Nom
JJ	Predet	s
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: A rough equivalence table between part of speech tags in the treebank and abstract categories from the grammar

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 since 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 the context-sensitivity that we need to describe noncompositional patterns.

The `inside` combinator is used also when the annotation schema for the treebank is incompatible with the abstract syntax of the resource grammar. For instance the grammar does not have a category analogous to the `SBAR` phrase, so we must change the modelling of the sentence:

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

Here the verb “said” takes as object a sentence which is embedded in an `SBAR` phrase. We have to take out the sentence and attach it directly to the verb. We can do this 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. We call the custom rule pV instead of the generic cat combinator, because we need to select the verb with the right valency. The valency is controlled with the argument VS. The rule also shows the usage of the combinator word, which can check for specific words like "that".

For the conversion of a single word, we use the combinator lemma which takes as arguments an abstract category and a name of constituent (see Table 1), and it searches for a function which has the current word in the specified constituent. For instance the word "said" is processed by the call:

```
lemma "VS" "s VPast"
```

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 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 patterns in about 800 lines of Haskell code.

5 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 an adjective phrase, a noun phrase, an adverbial phrase or a common noun, but we also found 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 two new functions PredVPovs and PredVPosv. The first implements an object-verb-subject word order and the second implements object-subject-verb.

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. 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 all this inconsistencies is that the lexicon is imported from an external resource which is not entirely compatible with the resource grammar.

Another issue is that our lexicon covered only British English while the treebank is based on American English. We encountered some words with different spellings and we extended the lexicon it with the alternatives.

6 Grammar Evaluation

The simplest way to compute the coverage of the grammar is to calculate the percentage of nodes in the abstract trees that are filled in with function symbols instead of placeholders. A plain counting shows that in average 94.47% of the nodes are filled in with functions, and the remaining 5.53% are placeholders. However, this percentages include both placeholders which are due to missing syntactic constructions and those that are due

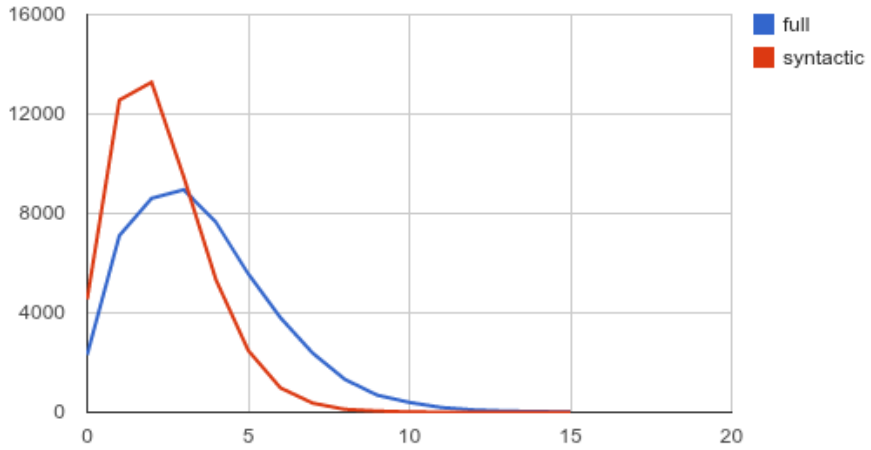

Figure 2: The distribution of the number of sentence with a given number of placeholders

to unknown words. It is a good idea to separate them since in any kind of processing, unknown words are easier to handle than unknown syntax. If we pretend that the placeholders on the leafs of the trees are actually known words we get the much higher coverage of 96.51%.

Another way to evaluate the coverage is to look at how many sentences were fully converted and how many have only one, two, three or more placeholders. As a whole, out of 49208 sentences, 2313 sentences (4.7%) were fully converted to abstract trees. Again, if we exclude the unknown words, we get much higher numbers - 4543 sentences or 9.23%. The overall distribution of the number of sentences with a given number of placeholders is shown on Figure 2. It can be seen that the majority of sentences is clustered in the region from one to four placeholders per sentence. More concretely the expected average number is 3.55 with a standard deviation of 3.77. If we ignore the unknown words we get a sharper curve which gives an expected value of 2.26 placeholders with a deviation of 2.27.

Although, the plain percentage of recovered function names is quite high, the percentage of sentences that are actually parsable is quite low. This shows that grammar can explain most of the syntactic constructions in the sentence, but still given the usual distribution of sentence lengths in the Penn Treebank, there is still a high chance to encounter an unknown construction in most of the sentences. In general the chance to find new constructions is higher in a longer sentence. This points to an alternative measure for the coverage. We can compute the ratio between the number of tokens in a sentence and the number of placeholders. We found that in average it can be expected to encounter a new placeholder once in every 8.08 tokens, or if we ignore the unknown words in every 11.84 tokens. This correlates well with the observation that the average length of the fully converted sentences is 9.86 tokens although there are sentences of length up to 47 tokens. In general the grammar works better on shorter sentences.

The statistics above are obtained after we had manually checked about 5000 sentences, and we had enriched some of them. We found cases where the automatic transformation patterns were not smart enough to recover all details of the abstract tree, and we filled them in with manual or semiautomatic postediting. If the same evaluation is done on the rough unedited corpus the coverage drops from 94.47% to 92.48%. This shows that

there is still potential for improving the coverage but we doubt that it will increase with more than one or two percents.

The main outcome from the manual checking was that we identified the major reasons for failures in the grammar coverage. One very common way to introduce people and organizations in the Wall Street Journal articles is to use apposition:

> Lorillard Inc., the unit of New York-based Loews Corp. that makes Kent cigarettes

Although the grammar has some functions for modelling apposition, they are not sufficient to cover most of the examples. Another common feature is that the articles often cite dates, percentages, currencies and other numbers. The grammar does not have specific support for these, and even when only plain numbers are used, it is not always possible to cover all cases. The reason is that the numeral system in the resource library is designed to be general accross languages[2], but it covers only numbers up to 999 999. Furthermore it does not cover mixed numerals like "500 million". There is also a need to implement some common English idioms like "years ago" and "years old". All these extensions are relatively easy and would lead to a significant improvement in the coverage.

Two other issues would require more involving changes. The first case is when an adverbial modifier is inserted between the arguments of a verb. For instance the following sentence is not covered in the grammar:

> Such disclosures of big holdings often are used **by raiders** to try to scare a company's managers.

because the phrase "by raiders" is inserted before the object "to try to scare a company's managers". It is parsable only if the modifier is moved to the end of the sentence. This would not be a problem, however, if the object was a noun phrase. The transitive verbs are first turned into an object less verb phrase `VPSlash` and after that they are applied to the object. The grammar allows insertion of modifiers between the verb and an argument of type noun phrase.

An even harder example is when part of a sentence is moved into an embedded sentence, for instance:

> Working students, **she explains**, want some satisfaction.

Here "she explains" have to be moved either to the beginning or to the end of the sentence. Currently there is no way in the grammar to model this kind of complex movements.

7 Conclusion

In general, we found the outcome of the grammar evaluation quite satisfactory. Although only less than 10% of the treebank sentences are really parsable, the completeness of the partly reconstructed trees shows that actually the grammar has quite high coverage. Still given that most of the Penn Treebank sentences are quite long, there is a high chance to encounter unknown constructions in most sentences. The success rate would be much higher if the sentences were first segmented into smaller chunks which are treated individually. This is in fact one of the many ways to do robust processing with the grammar.

The translated Penn Treebank is useful in many ways. First of all we already identified important directions in which the grammar should be extended. Further investigations will show even more interesting linguistic constructions which are still missing. We are not aiming at full coverage, however, since this will still not guarantee coverage on unseen sentences. Rather than that we aim at using the translated treebank as a corpus for training statistical models which will be used both for disambiguation and as a compensation for the incomplete coverage.

[2] In fact it provides a common abstract syntax for the numeral systems of around 100 languages (Hammarström and Ranta, 2004)(Hammarström, 2004)

References

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

Cahill, A., Mccarthy, M., Genabith, J. V., and Way, A. (2002). Automatic annotation of the Penn Treebank with LFG F-structure information. In *Proceedings of the LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data, Las Palmas, Canary Islands*, pages 8–15.

Chen, J., Bangalore, S., and Vijay-Shanker, K. (2006). Automated extraction of tree-adjoining grammars from treebanks. *Natural Language Engineering*, 12(3):251–299.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98, pages 16–23, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hammarström, H. (2004). Deduction of numeral grammars. In i Alemany, L. A. and Egré, P., editors, *Student Session: 16th European Summer School in Logic, Language and Information, Nancy, France, 9-20 August, 2004*, pages 94–104.

Hammarström, H. and Ranta, A. (2004). Cardinal numerals revisited in GF. WORKSHOP ON NUMERALS IN THE WORLD'S LANGUAGES, 29-30 March 2004, Dept. of Linguistics, Max Planck Institute for Evolutionary Anthropology, Leipzig, Germany.

Hockenmaier, J. and Steedman, M. (2007). CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Hutton, G. (1992). Higher-order Functions for Parsing. *Journal of Functional Programming*, 2(3):323–343.

Jones, S. P. (2003). *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press.

Kipper, K., Dang, H. T., and Palmer, M. (2000). Class-based construction of a verb lexicon. In *AAAI-2000 Seventeenth National Conference on Artificial Intelligence*.

Ljunglöf, P. (2004). *Expressivity and Complexity of the Grammatical Framework*. PhD thesis, Department of Computer Science, Gothenburg University and Chalmers University of Technology.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

Mitton, R. (1986). A partial dictionary of English in computer-usable form. *Literary & Linguistic Computing*, 1(4):214–215.

Miyao, Y., Ninomiya, T., and Tsujii, J. (2004). Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *Proceedings of the First international joint conference on Natural Language Processing*, IJCNLP'04, pages 684–693, Berlin, Heidelberg. Springer-Verlag.

Miyao, Y. and Tsujii, J. (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.

O'Donovan, R., Burke, M., Cahill, A., Van Genabith, J., and Way, A. (2005). Large-scale induction and evaluation of lexical resources from the Penn-II and Penn-III treebanks. *Computational Linguistics*, 31(3):329–366.

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

Ranta, A. (2009). The GF resource grammar library. *Linguistic Issues in Language Technology*.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell, J. T., and Johnson, I. M. (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.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Shen, L. and Joshi, A. (2005). Building an LTAG treebank. Technical Report MS-CIS-05-15, University of Pennsylvania.

The XTAG-Group (1998). A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 98-18, University of Pennsylvania.

Xia, F. (1999). Extracting Tree Adjoining Grammars from bracketed corpora. In *Natural Language Processing Pacific Rim Symposium*.

Xia, F. (2001). *Automatic Grammar Generation From Two Different Perespectives*. PhD thesis, University of Pennsylvania.

Xia, F., Palmer, M., and Joshi, A. (2000). A uniform method of grammar extraction and its applications. In *Proceedings of the 2000 Conference on Empirical Methods in Natural Language Processing*, pages 53–62, Hong Kong.