Lexicalized Tree Adjoining Grammar Extraction For KBGEN

Wei Qiu Jiri Marsik

January 16, 2013

1 Introduction

Natural language generation is the process to generate coherent natural language from knowledge base triples. There are generally two approaches in this area: some researchers use decent grammar formalism such as TAG(Tree adjoining grammar) to model the interaction of linguistic objects. This approach normally requires expert knowledge of grammar formalism, a lot of investigation into corpus and the domain where the system will be used. The main drawback of this approach is that it's not easy to migrate from domain to domain. The other approach is statistics based. It often directly work on the surface form of natural language. At the very beginning of this area, people use canned text to build simple NLG systems. The problem of this approach is that it's hard to model complicated linguistic phenomena and control the quality of the text generated.

Our project aims to reduce the effort of construct grammars for the first approach. We want to minimize the expert knowledge involved for building grammar for certain domain. A deep grammar formalism is still used, but the user doesn't need to handcraft the grammar from scratch anymore.

This report is organized as below: firstly we will generally introduce the background of this task including the related KBGEN task and the corpus; Then the semantic alignment annotation will be described; Then two sections will describe our approach to extract Lexicalized Tree Adjoining Grammar which linguistically make sense.

2 Background and previous work

Our project is based on KBGen shared task.Banik et al. (2012) The goal of KBgen task to produce a coherent description of biological entities, processes and connections between them. The task involves aggregation adn the generation of intra-sentential, syntactically bound pronouns but it does not require the generation of any discourse anaphora of referring expressions. The corpus we used from KBgen task and contains 107 instances.

The idea using domain specific grammar extracted from parsed corpus for NLG system is proposed by DeVault, Traum, and Artstein (2008a) and DeVault, Traum, and Artstein (2008b). And extracting grammars from treebanks has already attracted a lot of attention from researchers. Several proposals for TAG extraction are made by F Xia (2001); Fei Xia (1999); Fei Xia and Palmer (2006); Chen and Shanker (2005) not mentioning other deep grammar formalisms. Our approach is quite similar to DeVault, Traum, and Artstein (2008a)'s method, but we payed more attention to semantic alignment annotation to make sure the grammar extracted has good linguistic meaning.

3 Semantic Annotation

Extracting the elementary trees and aligning them to the triples without any prior knowledge about the link between parts of the sentences and the triples would have been very difficult. Therefore, the first step of developing our system was to manually align sets of triples to words from the sentences.

In order to shed light on the kind of alignment we were looking for, I will list the properties we would like the alignment to have:

- Every word is aligned to some set of triples, i.e. every word is "semantically motivated". We would like our system to be able to generate the kind of sentences we had at the start. However, our surface realizer, GenI, never produces superfluous output (every word of the output is part of an elementary tree which was induced by some set of triples in the input).

- Every triple is aligned to some set of words in the sentence. This is a reasonable requirement since the training sentences we have should be complete expressions of their semantic models. Furthermore, we would like to be able to generate at least the training data we have and GenI only considers generation successful if every semantic formula is realized via some elementary trees, which means there should not be any orphan triples.

- The relation between the sets of words and sets of triples should be an alignment. By this, we mean to say that for every two different pairs of aligned sets of words and triples, the sets of words and the sets of triples should both be disjoint (in short, a word should not be aligned to two different sets of triples and a triple should not be aligned to two different sets of words).

- The alignment we want to create should be the finest such alignment, meaning that any alignment which is a refinement of our alignment must lack one of the above properties. This is a natural requirement if we were doing things manually, since any rougher alignment can be easily arrived at by merging aligned groups. Furthermore, finer alignments will let us arrive at smaller elementary trees which generalize better, though they might overgenerate (which is a problem we were not afraid of, since we were afraid whether we will even be able to generate something).

For further discussion, we will label the sets of words that end up aligned to some sets of triples *groups*. Before we proceed with a more linguistic description of the annotation guidelines, there is one more technicality we would like to bring to light.

3.1 Contiguous groups

In our first rounds of annotation, we were working with an overly strict constraint. Due to some confusion about how the realizer will work and which realizer we will actually use, we restricted ourselves to contiguous group only (groups whose all words form a contiguous substring of the sentence). Under this constraint, we had trouble accounting for several phenomena, the first one being conjunction. Coordination was particularly tricky and required special treatment even without the contiguity constraint (see Section 4.2 for more details). However, other phrase structures such as "... X **, which also requires** Y **,** ..." (mind the second comma) with the semantics (X, requires, Y) or "**the function of** X **is** Y" with semantics (X, has-function, Y).

After a while, we realized that the groups that we were manually forming would correspond exactly to the final elementary trees (it was the only alignment between semantics and syntax we had, after all). Therefore, the sets of words in a group would not have to be contiguous at all (the yield of an elementary tree may have substitution or foot nodes surrounded by (co-)anchors).

3.2 Annotation guidelines

The realization that the groups of the alignment will correspond to elementary trees restricts the space of possible alignments significantly. These restrictions came both from the surface realization of the sentence, where we imagined an idealized parse tree and made sure that the groups would correspond to sensible and composable elementary trees in that parse tree. On the semantic side, we also needed the semantic links from the set of triples assigned by the alignment to line up with the syntactic links in the elementary tree (the substitution and adjunction sites).

Here are the guidelines for several common situations:

- NPs are aligned to their entity declarations.
- For simple intransitive or transitive verbs, the event entity declaration and the triples which specify the agent (and the patient or whatever role the object takes) are aligned to the verb.
- In an n-nary conjunction, all of the $n-1$ interposed conjunctions (comma, *and*) are aligned to specialized **coordinates** relations which group all the coordinated entities into a single artificial entity.
- For prepositions, we align the preposition (e.g. "X **of** Y") to a single triple which links some two entities (X and Y). In the resulting grammar, the preposition will correspond to an auxiliary tree where one the two semantic links will correspond to a substitution site and the other to the adjunction site.

And here is our guideline for a more involved but quite common scenario:

- "**The function of** X[NP] **is to provide energy for** Y[NP] **to** Z[VP]."

  Everything in bold in the sentence above will form a single group. This will be aligned to the following triples: (X, has-function, Z), (Z, raw-material, X) and (Z, agent, Y).

In the last example, you can see that I use the same meta-variables for parts of the input syntax and the semantic entities. I assume there is a natural assignment of the entities of the semantic representation to certain constituents in the phrase structure. This assumption is based on the fact that some groups (elementary trees) are aligned to entity declarations and the idea that these entities could be propagated up the tree according to headedness. It was this idea that gave us hope that it should be possible to provide semantic indices for the generated elementary trees automatically.

4 Preprocessing

There was a long way before we could start work on the problem itself and we had to write a few auxiliary programs along the way. This section describes the major steps and the tools we used there.

4.1 Parsing and headedness

Our plan is to extract the elementary trees for a TAG from a treebank. Since our corpus lacks phrase structure information, we ran an off-the-shelf parser on it. The parser that we used for this task is the Stanford parser.

At the outset, we had plans to use both the phrase structure and dependency parses that the Stanford parser could produce. Previous research has shown that having a dependency parse facilitates the task of determining headedness, which is usually a subtask of elementary tree extraction.

Finally, we have found a simpler way of getting the head information directly in the phrase structure parse trees through a command line switch in the Stanford parser.

Furthermore, the Stanford parser offers two pre-trained models for parsing English, an unlexicalized one and a lexicalized one. We have done the parsing using both of them and try to review their performance. Their outputs were significantly different and both performed their own idiosyncratic errors. We judged which of these errors would likely be more harmful to the following grammar extraction process and decided to stick with the output of the unlexicalized parser (though the result of the unlexicalized parsing almost as just as bad as that of the lexicalized one).

4.2 Aggregating coordinations

When we started working on the manual alignment with our self-enforced constraint to produce only contiguous groups (see section 3), one of the difficulties we faced was coordination. Consider the following example:

```
Electrogenic pumps, which consist of a hydrophobic amino acid,
a polar amino acid and a monomer, create  membrane potential.

(KBGEN-INPUT
    :TRIPLES (
            (|Electrogenic-Pump21300| |has-part| |Monomer21323|)
            (|Electrogenic-Pump21300| |has-part| |Polar-Amino-Acid21291|)
            (|Electrogenic-Pump21300| |has-part| |Hydrophobic-Amino-Acid21290|)
```

```
        (|Create21297| |agent| |Electrogenic-Pump21300|)
        (|Create21297| |result| |Membrane-Potential21299|))
:INSTANCE-TYPES (
        (|Monomer21323| |instance-of| |Monomer|)
        (|Polar-Amino-Acid21291| |instance-of| |Polar-Amino-Acid|)
        (|Hydrophobic-Amino-Acid21290| |instance-of| |Hydrophobic-Amino-Acid|)
        (|Electrogenic-Pump21300| |instance-of| |Electrogenic-Pump|)
        (|Create21297| |instance-of| |Create|)
        (|Membrane-Potential21299| |instance-of| |Membrane-Potential|)))
```

To what group should the comma and the conjunction *and* belong, i.e. what semantics correspond to them and only to them? The **has-part** relations belong to the verb *consists*, the instance declarations belong to the NPs. Yet we need to assign the coordination constructions to some semantics, otherwise we will not be able to generate them. To resolve this issue, we have developed a tool to reify the coordination structures within the triples themselves.

In our first approach, our program would look for every instance where one entity was in the same relation with multiple entities, such as above, and converted the triples to the following:

```
:TRIPLES (
        (|Coordination2085| |coordinates| |Monomer21323|)
        (|Coordination2085| |coordinates| |Polar-Amino-Acid21291|)
        (|Coordination2086| |coordinates| |Coordination2085|)
        (|Coordination2086| |coordinates| |Hydrophobic-Amino-Acid21290|)
        (|Electrogenic-Pump21300| |has-part| |Coordination2086|)
        (|Create21297| |agent| |Electrogenic-Pump21300|)
        (|Create21297| |result| |Membrane-Potential21299|))
```

What we did is that we built a very specific kind of binary tree which covers the coordinated entities (the generated tree is specific in that it always leans to the right, so it looks more like a linked list). Here, one **Coordination** entity, **Coordination2085**, subsumes the **Monomer21323** and the **Polar-Amino-Acid21291**. This coordination corresponds to the conjunction *and*. The second coordination connects another entity, the **Hydrophobic-Amino-Acid21290**, to the previously formed **Coordination2085**. The resulting **Coordination2086** can then serve as the singular object to the verb *consists*. Now, all of the NP coordinating constructs (commas and *and*) are easily accounted for. They form singleton groups that are aligned to a pair of **coordinates** relations.

Once we made the switch from contiguous groups, we revised our approach to coordination. Instead of making this hierarchy of coordination constructs, we just create a single **Coordination** entity which encapsulates all of the coordinated entities directly. Obviously, this would not have been possible for coordinations of more than 2 constituents using only contiguous groups, since we would not to place two non-adjacent conjunction markers into one group.

The upsides of this new approach are that it reduces the complexity of the newly added triples and that it does not overgenerate like the last approach, which would end up extracting two elementary trees for conjunction (one for comma and one for *and*), but would not be able to distinguish between them and use them properly. The only downside of the new approach is that it needs to learn different elementary trees for each different arity of conjunction present in the corpus. However, since the number of coordinated constituents is always between 2 and 4 in our corpus, this is a non-issue.

Here is the above example, now rendered using the new coordination aggregation rules:

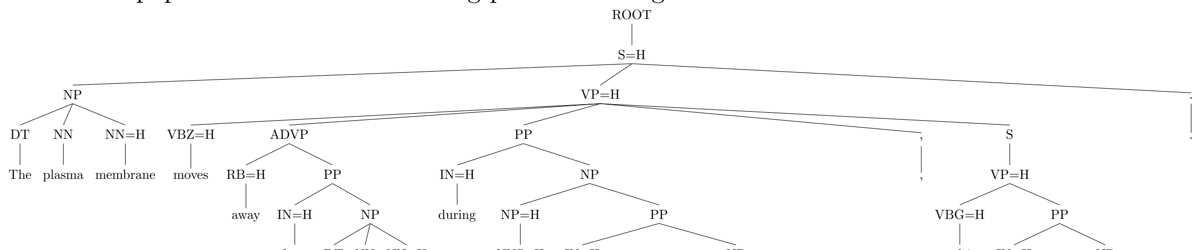
```
:TRIPLES (
        (|Electrogenic-Pump21300| |has-part| |Coordination2015|)
        (|Coordination2015| |coordinates| |Monomer21323|)
        (|Coordination2015| |coordinates| |Polar-Amino-Acid21291|)
        (|Coordination2015| |coordinates| |Hydrophobic-Amino-Acid21290|)
        (|Create21297| |agent| |Electrogenic-Pump21300|)
        (|Create21297| |result| |Membrane-Potential21299|))
```

The code for performing the former aggregation method is still present in the aggregation program, but is no longer used (see function `coordinate-objects-list`). Switching to the new method was just a matter of writing a new function, `coordinate-objects-flat`, and calling that one instead of the former.

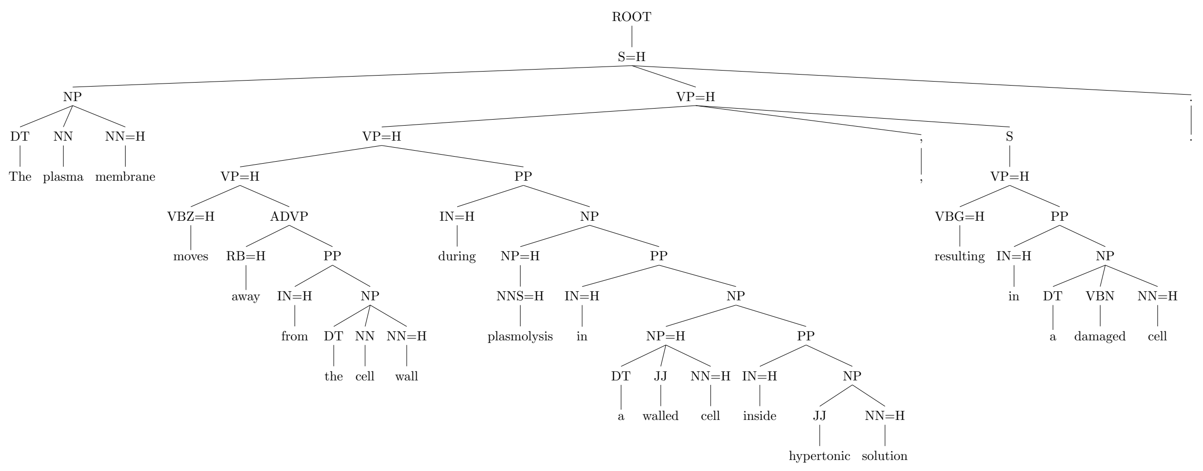
4.3 Normalizing

Once we had the manually aligned data with the aggregated coordinations and parsed sentences, we thought ourselves ready to extract the elementary trees. However, the trees produced by the Stanford producer do not look very much like derived trees produced by a TAG.

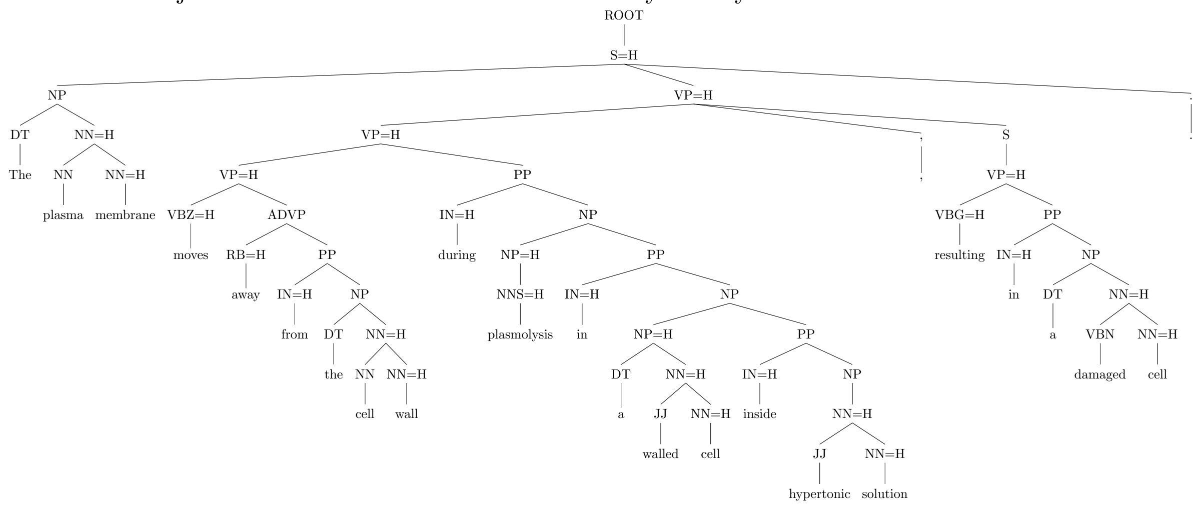
Xia and Palmer were facing a similar problemFei Xia and Palmer (2006). We considered applying their approach. However, their approach is not as simple as it seems on paper. Consider the following parse tree we got from



The PP *"during..."* and the S *"resulting..."* with its preceding comma both modify their parent, the VP of our sentence. We can imagine that in our implementation of the algorithm, these would be identified as modifiers and they would be put on different levels from the argument ADVP, like so.


ROOT
S=H
NP VP=H .
DT NN NN=H VP=H , S .
The plasma membrane VP=H PP , VP=H
VBZ=H ADVP IN=H NP VBG=H PP
moves RB=H PP during NP=H PP resulting IN=H NP
away IN=H NP NNS=H IN=H NP in DT VBN NN=H
from DT NN NN=H plasmolysis in NP=H PP a damaged cell
the cell wall DT JJ NN=H IN=H NP
a walled cell inside JJ NN=H
hypertonic solution

So far so good, nice and consistent. However, then we look at the NPs and we see that the adjectives are sisters to the nouns that they modify.


ROOT
S=H
NP VP=H .
DT NN=H VP=H , S .
The NN NN=H VP=H PP , VP=H
plasma membrane VBZ=H ADVP IN=H NP VBG=H PP
moves RB=H PP during NP=H PP resulting IN=H NP
away IN=H NP NNS=H IN=H NP in DT NN=H
from DT NN=H plasmolysis in NP=H PP a VBN NN=H
the NN NN=H DT NN=H IN=H NP damaged cell
cell wall a JJ NN=H inside NN=H
walled cell JJ NN=H
hypertonic solution

This asymmetry in the parser output made the approach of implementing a procedure like theirs that would handle all of the flattening phenomena in one unified manner seem not so viable. Instead, we chose to implement a simple rule-based logic program to solve the flattening we have discovered in our data.

We implemented two deflattening rules. One rule handled the general case demonstrated on the PP and SBAR modifying the VP. This rule uses a simple heuristic to determine whether a child of a node with more than two children is a modifier or not. We refined this heuristic by querying the output of the program and modifying the rules to account for unforeseen circumstances. The final rule for detecting modifiers/adjuncts follows.

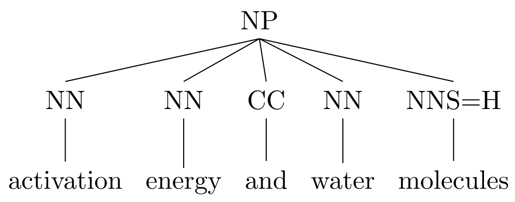
A is an adjunct if one of the following is true:

- A is a PP
- A is an S which does not have the form [S [VP=H [TO=H to] ...]]
- A is a comma (we treat commas as adjuncts, so that, along with the “real” adjunct, they will form a two-level auxiliary tree)

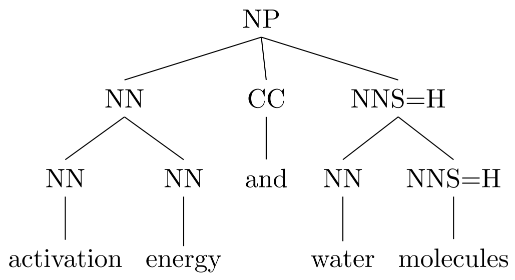
- A is an SBAR of the form [SBAR [WHNP=H [WDT=H which]] ...] (appositive relative clauses)

The second rule we have implemented is specialized for NPs. We look at every noun in a final position (the last child of an NP or one that is followed by a comma or a conjunction, since the Stanford parser sometimes even flattens entire conjunctions of nouns, see below). For every such noun, we look at its left sister and if it is a permissible modifier (NN, NNP, JJ, VBN or ADJP). If so, we can mark the modifier as an adjunct and put it together with the noun under a new noun tag.

This tree then...



...becomes this...



After applying these two rules, we have examined all the possible problem sites. That is, the nodes which had more than two children after the deflattening (possible false negatives of our deflatter) and all the adjunctions performed by our deflatter (possible false positives). Upon their inspection, we have not found much room for improvement and declared our algorithm finished.

The downside of this approach is that the further refining of our heuristic predicates would get much harder as the size and variety of our corpus grew larger. The upside is that it handles the language found in our corpus very well and it has a short and (hopefully) transparent implementation. For a discussion of some of the uncommon implementation techniques we used, see the post at the following link.

`http://jirka.marsik.me/2012/12/19/the-working-hipster-s-clojure/`

5 TAG extraction

After we get the normalized tree with head information, we can extract the TAG from it with the help of the alignment of surface sentence to underlying KB-GEN triples. The key idea is extract adjunction trees greedily, then extract the substitution trees. Of course conjunction trees as a specific form of substitution trees require some special treatment before adjunction trees can be extracted.

First we will present the pseudo code Algorithm 1, 2, 3, 4 for the tree extraction algorithms, then a working example will be given to illustrate the idea.

Algorithm 1 LTAG extraction algorithm

```
Require: $Tree$, $alignment$
 1: for all $node$ in $Tree$ do
 2:    add semantic group information according to $alignment$
 3: end for
 4: $agenda \Leftarrow [Tree]$
 5: $well\_formed\_tree \Leftarrow []$
 6: while $agenda$ is not empty do
 7:    $current\_tree = agenda.pop()$
 8:    if $current\_tree$ only contains one group then
 9:       $well\_formed\_tree.append(.current\_tree)$
10:    end if
11:    extract conjunction subpart
12:    extract adjunction subpart
13:    if adjunction found then
14:       continue
15:    else
16:       extract substitution subpart
17:    end if
18: end while
19: return $well\_formed\_tree$
```

Algorithm 2 extract conjunction subpart

```
 1: for all $subtree$ in $Tree$ do
 2:    if $subtree$ has a conjunction structure then
 3:       cut the conjunction tree off the subtree
 4:       remove all arguments in the conjunction structure
 5:       copy group information as candidate information
 6:       change the empty node'group to empty set
 7:       update the group information of all separated parts
 8:       put subtree, conjunction tree, argument trees into $agenda$
 9:    end if
10: end for
```

Algorithm 3 extract adjunction subpart

```
1: for all subtree in Tree do
2:   for all leftmost child or rightmost child of subtree do
3:     TODO
4:     if child has the same pos as subtree then
5:       if child is well separated from other parts then
6:         cut off subtree from its parent
7:         copy the group information as candidate information
8:         update the hole's group as empty set
9:         cut off child from subtree
10:        copy the group information of child hole as candidate information

11:        update child hole's group as empty set
12:        update group information of all subparts
13:        put adjunction mark
14:        put all subparts into agenda
15:      end if
16:    end if
17:  end for
18: end for
```

Algorithm 4 extract substituition subpart

```
1: $headgroup \Leftarrow current\_tree'sheadgroup$
2: for all subtree in current_tree do
3:   if subtree's group is disjoint from headgroup then
4:     cut subtree off $current_tree$
5:     copy the relative group information to candidate
6:     update all group information
7:     put substitution mark
8:   end if
9: end for
```

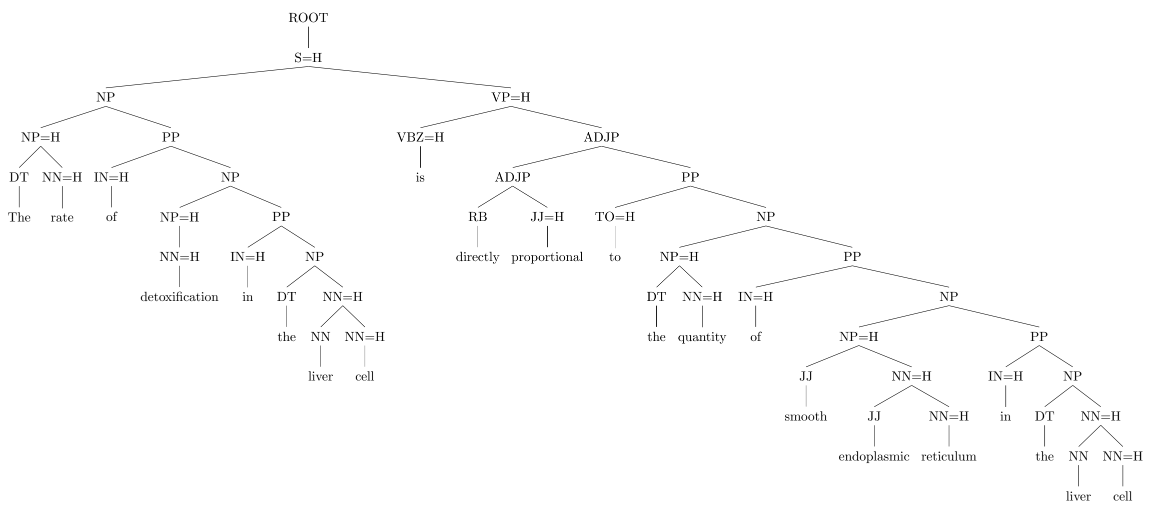
We will take sentence

(1) The rate of detoxification in the liver cell is directly proportional to the quantity of smooth endoplasmic reticulum in the liver cell.

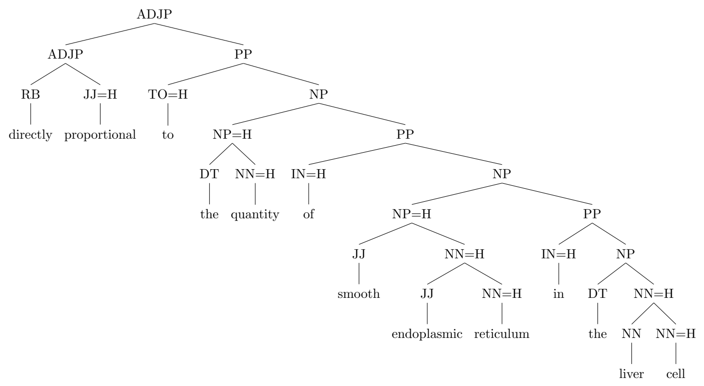
as example, the semantic group annotation is:

(2) [1 The rate of] [2 detoxification] [3 in] [4 the liver cell] [5 is directly proportional to] [6 the quantity of] [7 smooth endoplasmic reticulum] [8 in] [9 the liver cell] .

After parsing and preprocessing, we get the tree:



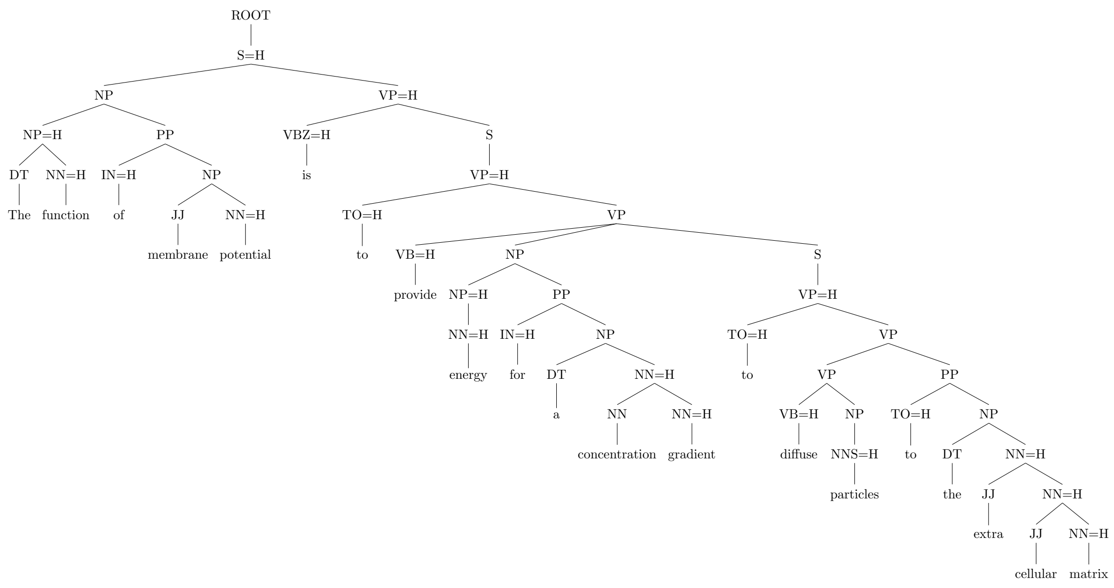
$S = H$ label means the syntax category is S and S is the head of its parent tree. We can see that stanford parser doesn't always provide head information. For example:



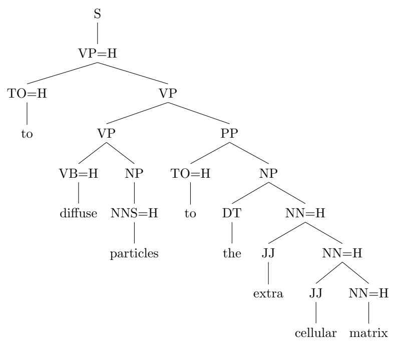
All of the children of root ADJP don't have any head information. This is a problem for our extraction algorithm simply because our algorithm relies on the head information to find the real group for the root. And quite often the head itself would be extracted out leaving a hole in the original tree. We need also to assign a head to the remaining siblings of it. Our work around is to assign the first child which has non-empty group informaiton as the head. This may seem to be problematic, but actually it works pretty well simply because the tree is almost binarized And stanford parser generate head information for the most cases. Of course, a heuristic rule may work better. We here leave it to future work. Check *extractor._get_head_group* for the details.

First the whole tree will be added to agenda. After it has been poped out for further examination, firstly it will be checked whether it contains any conjunctions. The answer is no in this case. Then we can check whether it contains any adjuctions.

A simple strategy is to only check the root and its leftmost and rightmost children. But a close look into the corpus show us this strategy is too simple and sometimes will not work. It will extrat some grammars without good linguistic interpretation if we postpone the adjuction checking of other subtrees. So in our algorithm, we exaustively check the adjunction for all of the subtrees. An example that the simple strategy doesn't work is shown below:



When



is popped out, if we don't identify the adjuction structure wrapped inside, the substitution procedure would break it and it would be impossible to find a proper adjuction tree.

Now come back to our example, the chuck "The rate of detoxification in the liver cell" looks like a adjuction structure for its first left child, but actually it's not simply because the adjunction node's group is not disjoint from other parts("of" is also in group 1). It's illustrated in Figure 1.

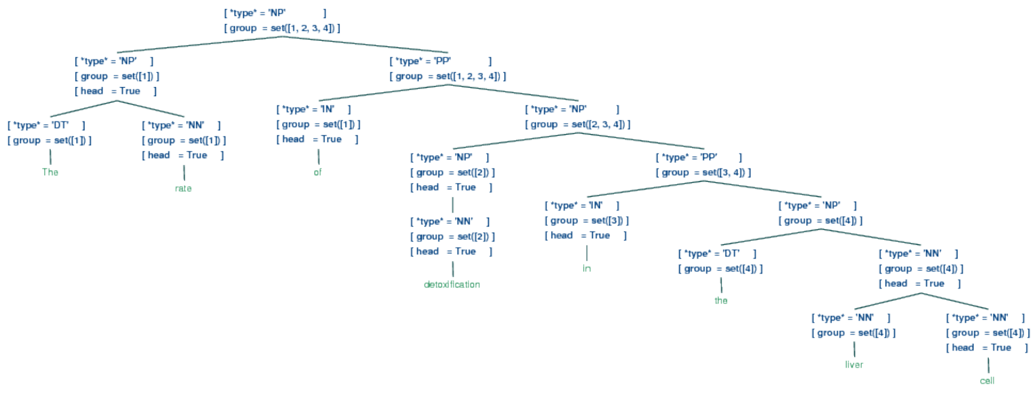

[*type* = 'NP'] [group = set([1, 2, 3, 4])]
[*type* = 'NP'] [group = set([1])] [head = True]
[*type* = 'PP'] [group = set([1, 2, 3, 4])]
[*type* = 'DT'] [group = set([1])]
The
[*type* = 'NN'] [group = set([1])] [head = True]
rate
[*type* = 'IN'] [group = set([1])] [head = True]
of
[*type* = 'NP'] [group = set([2, 3, 4])]
[*type* = 'NP'] [group = set([2])] [head = True]
[*type* = 'NN'] [group = set([2])] [head = True]
detoxification
[*type* = 'PP'] [group = set([3, 4])]
[*type* = 'IN'] [group = set([3])] [head = True]
in
[*type* = 'NP'] [group = set([4])]
[*type* = 'DT'] [group = set([4])]
the
[*type* = 'NN'] [group = set([4])] [head = True]
[*type* = 'NN'] [group = set([4])]
liver
[*type* = 'NN'] [group = set([4])] [head = True]
cell

Figure 1: "The rate" is not adjuction

Using the same algorithm, we can identify "the rate of " is a proper adjunction tree.

The third step is to extract the elementary tree. Actually after all of conjunctions and adjunctions are kicked out the leftover's root must be a root of elementary tree. Based on this observation, the algorithm for substitution tree extraction is relatively simple. First we identify the group of the root, then kick out all the parts which don't belong to this group and add them to the agenda. The left spine would be a "pure" elementary tree which can be directly added to the resulted grammar.

For this example, 9 grammar snippets will be extracted:

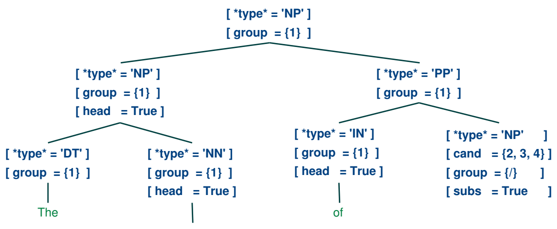

[*type* = 'NP'] [group = {1}]
[*type* = 'NP'] [group = {1}] [head = True]
[*type* = 'PP'] [group = {1}]
[*type* = 'DT'] [group = {1}]
The
[*type* = 'NN'] [group = {1}] [head = True]
[*type* = 'IN'] [group = {1}] [head = True]
of
[*type* = 'NP'] [cand = {2, 3, 4}] [group = {/}] [subs = True]

Figure 2: Grammar fragment 1

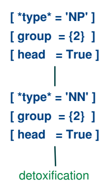


Figure 3: Grammar fragment 2

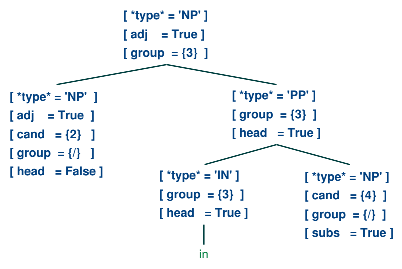


Figure 4: Grammar fragment 3

6 Semantic roles alignment

The extraction step extract grammar snippets for each sentence. This step intends to align the semantics to the grammar snippets and the syntax holes in the grammar snippets. The algorithm is showed in Algorithm 5.

Algorithm 5 Align semantic variables to syntax

```
 1: for all instance in alignment do
 2:   if it also occurs in triples then
 3:     continue
 4:   else
 5:     align the tree which has the same group to it
 6:   end if
 7: end for
 8: for all triple in alignment do
 9:   align the tree which has the same group to it
10:   for all hole in tree do
11:     assign the instance which is in the hole candidate set to it
12:   end for
13: end for
```

The algorithm seems to be simple, but due to the Stanford parser's output is not reliable and the inconsistency in the semantic alignment annotation, this algorithm could not alway succeed in aligning correspond semantics to syntax. In previous step, grammar fragments can always be extracted out successfully.

The other problem we encountered in the alignment is that quite often several triples are aligned to the same syntax. A example is that for sentences with structure "the function of . . . is for . . . to . . . ". This type of sentence would have

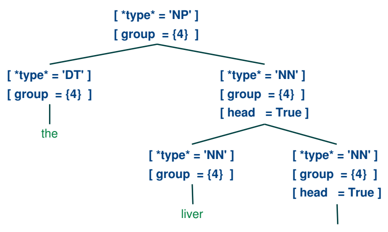


Figure 5: Grammar fragment 4

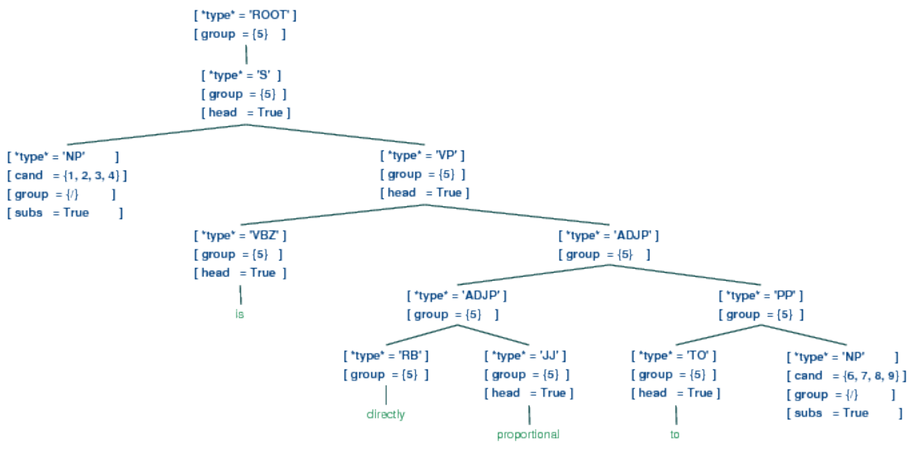


Figure 6: Grammar fragment 5

3 syntax holes in the elementary tree. And there are 3 triples related to it. We decided to extract 3 elementary trees for each there are two holes aligned to semantic variables. Since we haven't try out the generator, we don't know whether this treatment which we can consider it as a type of underspecification is appropriate or not here.

To filter out the same grammar fragments extracted from different sentences, we maintain a global hashtable.

For the example provided in section 5, we got the final grammar below:

```
|directly-proportional|
(ROOT[]
  (S[+head]
    (NP[sem=0, +subs] )
    (VP[+head]
      (VBZ[+head] is)
      (ADJP[]
        (ADJP[] (RB[] directly) (JJ[+head] proportional))
        (PP[] (TO[+head] to) (NP[sem=1, +subs] ))))))

|Liver-Cell|
(DT[] the)
```

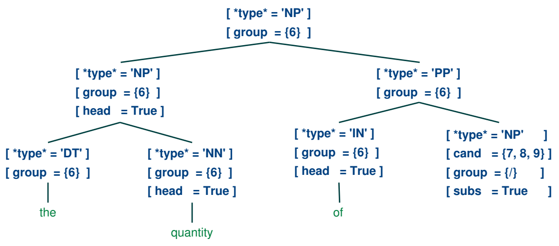


Figure 7: Grammar fragment 6

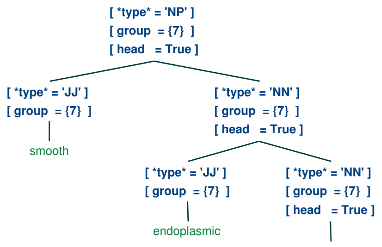


Figure 8: Grammar fragment 7

```
(NN[+head] (NN[] liver) (NN[+head] cell))

|quantity|
(NP[]
  (NP[+head] (DT[] the) (NN[+head] quantity))
  (PP[] (IN[+head] of) (NP[sem=0, +subs] )))

|has-part|
(NP[+adj]
  (NP[+adj, -head, sem=1] )
  (PP[+head] (IN[+head] in) (NP[sem=0, +subs] )))

|Smooth-Endoplasmic-Reticulum|
(NN[+head] (JJ[] endoplasmic) (NN[+head] reticulum))
(JJ[] smooth)

|Detoxification|
(NN[+head] detoxification)

|base|
(NP[+adj]
  (NP[+adj, -head, sem=0] )
  (PP[+head] (IN[+head] in) (NP[sem=1, +subs] )))

|rate|
(NP[]
```

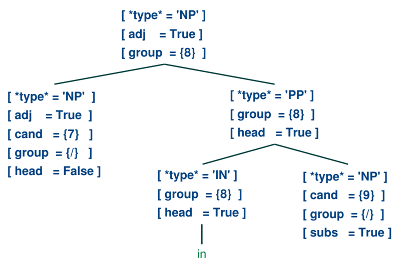


Figure 9: Grammar fragment 8

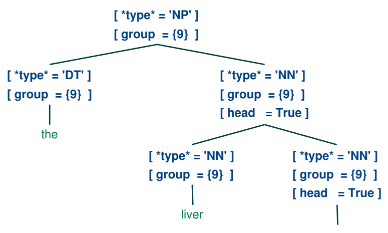


Figure 10: Grammar fragment 9

```
(NP[+head] (DT[] The) (NN[+head] rate))
(PP[] (IN[+head] of) (NP[sem=0, +subs] )))
```

Where sem represents the index of semantic variable.

7 Implementation and result

For implementation we use clojure and python. Clojure showed big advantages when we were trying to investigate the linguistic phenomena in the corpus. REPL really simplify the process. We use ParentedTree and FeatStruct from NLTK to model parse tree and feature structure. We found some small bugs in the deep copy facility provided by ParentedTree. Check our code for details.

The result is presented in directory "./result/". 'final.grm' contains the whole bunch of grammar fragments extracted out. 'grammr-verbose' contains the raw grammar which means semantics is not aligned for each sentence. Our algorithm works pretty well for correctly parsed sentences, but it would fail for those sentences which contain grammar errors. Here the errors don't need to be interpreted from linguistic perspective, but any inconsistency between the syntax tree and the semantic alignment can be considered as mistakes. To make our grammar not contain any mistakes, if one grammar fragment couldn't be aligned successfully, all other fragments from the same sentence will be deprecated. However, even we do this, there are still quite a lot mistakes in the final grammar especially for small span for noun phrase. It's due to that for noun phrase there are no holes in the syntax to align then the mistakes are transparent in the aligning step.

To work around this problem, a filter can be built to filter out the illegal grammar fragments, especially if there are alternatives syntax for the same semantic underline. In parsing step, we can use other parsers which could produce K-best result(Stanford lexicalized/unlexicalized parser can't produce K-best result, but its PCFG parser can), then we can use the semantic alignment to guide us to pick out the consistent parsing tree.

Application based evaluation can be carried after we integrate our module into a surface realizer.

8 Conclusion & Future work

In this project, we designed and implemented a algorithm to extract LTAG from parsed corpus and align the grammar to corresponding semantics. The biggest difference between our approach and DeVault, Traum, and Artstein (2008a)'s approach is that our result has better linguistic meaning. At the same time, our approach requires a bit more expert knowledge in the semantic alignment annotation step.

We still need to integrate our module with a surface realizer to build an end to end NLG system. Evaluation of our grammar needs to be carried. For the extraction algorithm self, possible improvements include making the semantic alignment annotation automatic, make the algorithm more robust to work with parsing errors.

References

Banik, Eva et al. (2012). "KBGen-Text Generation from Knowledge Bases as a New Shared Task". In: *INLG 2012*.

Chen, John and VK Shanker (2005). "Automated extraction of TAGs from the Penn Treebank". In: *New developments in parsing technology.*

DeVault, D, David Traum, and Ron Artstein (2008a). "Making grammar-based generation easier to deploy in dialogue systems". In: *Ninth SIGdial Workshop on Discourse and Dialogue (SIGdial)* June, pp. 198–207.

— (2008b). "Practical grammar-based NLG from examples". In: *Proceedings of the Fifth International Natural Language Generation Conference.*

Xia, F (2001). "Automatic grammar generation from two different perspectives". PhD thesis.

Xia, Fei (1999). "Extracting tree adjoining grammars from bracketed corpora". In: *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium.*

Xia, Fei and M Palmer (2006). "From Treebanks to Tree-Adjoining Grammars". In: *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*, pp. 1–39.