FROM TREES TO PREDICATE-ARGUMENT STRUCTURES

Maria Liakata and **Stephen Pulman**
Centre for Linguistics and Philology, Oxford University
Walton Street, Oxford OX1 2HG
Firstname.Lastname@ling-phil.ox.ac.uk

Abstract

The Penn Treebank encodes valuable information such as grammatical function, semantic roles, and identification of traces. The addition of such information was intended to facilitate the process of predicate-argument extraction. However, even with the enriched annotation this task is far from trivial and, to our knowledge, no complete set of predicate argument structures derived from the Treebank exists. Our paper describes a method for retrieving predicate-argument structures that circumvents the complexity of the tree structures in the corpus, while employing few template rules. Our system operates on a flattened, morphologically enriched version of the corpus. This flattened representation allows access to all levels of the tree simultaneously and thus enables the detection of the main sentence constituents by means of simple template rules. A small number of rules apply to identify the head words of each constituent and the latter fill in the constituent templates, to build the logical forms representative of the predicate argument structure. The system is robust in the face of incomplete syntactic coverage.

The problem

In (Marcus et al., 1994) a revised version of the Penn Treebank was described which included various types of annotation in addition to the original constituent structure. These extra annotations included information about grammatical functions (SUBJECT, LOGICAL SUBJECT, etc), semantic roles (LOCATION, MANNER, etc), the identification of traces with moved or controlling phrases, and information about various types of ellipsis and null elements. As well as being interesting and valuable in its own right, this extra annotation was intended in part to facilitate the extraction of predicate argument structures from Treebank trees: e.g.

```
( (S
   (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
   ((VP (MD will)
      (VP (VB join)
      (NP (DET the) (NN board) )
      (PP-CLR (PREP as)
         (NP (DET a) (NN director) ))
      (NP-TMP (NNP Nov.) (CD 29) )))
```

corresponds to a (simplified) predicate-argument construction something like:

```
and(join(e1, 'Pierre_Vinken', the(board)),
    as(e1, a(director)), 'TMP'(e1, 'Nov.29'))
```

However, extracting such predicate argument structures automatically is by no means a trivial task, even with the extra level of annotation provided. Various groups have reported different approaches, which we discuss below, but to our knowledge no complete conversion of the Penn Treebank to predicate-argument structures exists.

Note that here we are concerned simply with building structures that reflect basic predicate-argument relations, in a traditional first-order-logic-like notation. We are not concerned with assigning the appropriate case or thematic roles to phrases in some relation to the predicate: this is partly done for some adjuncts in the Treebank already, but to do this for all arguments and adjuncts requires access to external lexical information about the verbs and adjectives involved. Nor are we concerned with sense disambiguation: we will simply map a word to a similar logical constant, ignoring possibilities of polysemy. We regard both of these steps as de-

sirable, but separable later additions.

In our case, the main reason for wanting such simple predicate-argument structures is in order to provide data to an inductive logic programming machine learning system aimed at constructing simple 'domain theories' - theories which say what is and what is not likely to happen in some domain. We need these theories for various kinds of disambiguation methods for syntax and reference resolution.

Other approaches

The canonical method of building logical forms from syntax trees is via a form of syntax-directed translation, presupposing a complete syntactic grammar. Typically, each syntax rule has associated with it a semantic operation which constructs the interpretation of the mother constituent from those of its daughters. This construction might be via function-argument application, if the interpretations are represented as terms of some higher order logic, or via feature instantiation in unification or constraint based theories of grammar.

Some groups have tried the canonical syntax-directed translation route. (Frank et al., 2002), working within the framework of Lexical Functional Grammar, induce a Context Free PS grammar from the Treebank, and, less directly, from the Susanne corpus, and then associate with each PS rule regular expressions which allow f-structure representations to be built. Although the method is claimed to achieve high levels of precision and recall when evaluated against a hand-constructed gold standard, relatively small numbers of sentences have been processed, presumably due to the level of effort needed to enrich the PS rules, the large number of these implicit in the treebank, and the fact that for the syntax directed method to be successful, each constituent (LHS & RHS of a rule) in the whole tree must be matched by at least one template rule. Even though the regular expressions are more generalised than the PS rules (template/rule ratio of 0.36), the latter fact is crucial, for it means that for those trees containing any constituent for which no semantic rule has been written, nothing at all will work, at least, not without some further means of dealing with the incompleteness of coverage.[1]

(Palmer et al., 2001) have also used a template matching approach to map trees into predicate argument structures. Once a template is matched to a portion of tree, an appropriate corresponding feature bundle is selected according to the semantic class of the word instantiated as the predicate of the template. The templates are inspired by the kind of data structures used in Tree Adjoining Grammar, and mimic various TAG operations in that they can apply to non-contiguous stretches of tree. The point of the templates is to recognise tree configurations that signal particular grammatical relations. The large number of templates (c. 200) required is presumably motivated by the fact that there are many variations in tree configurations that need to be accommodated even though they are not relevant to the recognition of grammatical relations.

Theirs is a more ambitious aim than ours in that they are also trying to annotate arguments with thematic role information acquired from WordNet (Miller, 1995) and other resources, as well as carrying out sense disambiguation. Because of the intensive human effort needed, a relatively small subset of the Treebank has been analysed. Accuracy for the automatic part of predicate-argument assignment is reported to be around 80%.

Gildea and Jurafsky (Gildea and Jurafsky, 2000) are concerned with identifying semantic roles of sentence constituents within certain semantic frames. Since this approach, as is the case with Palmer's work, views predicate-argument structure with Information Extraction in perspective, the notion of a frame is seminal. Gildea and Jurafsky propose two methods for performing the frame labelling. In the first case, they train a parser on sentences from FrameNet (Baker et al., 1998) that contain frame annotation in order to derive constituent related features indicative of the application of a certain frame. They then use the learned features to identify frames in test data from FrameNet. The second approach is similar, the difference being that their aim is to extract features that act as frame boundaries without the initial training on the manually annotated data. The caveat in the previous approach is that FrameNet contains a small set of

[1] We would like to thank Josef van Genabith for making us aware of this work.

sentences for each frame and even then, they are not representative of the corpus but rather constitute ideal examples.

We have recently come across work by (Cahill et al., 2002) that attempts automatic annotation of the entire Penn Treebank (Marcus et al., 1994) with LFG f-structure information. An annotation algorithm traverses each tree top down and partitions CFG rules into head and left and right contexts. The above tripartition as well as subcategorization information for the most frequent rules is used to construct an annotation matrix. Once the annotation process is over, the result is input to a constraint solver that makes the final f-structure assignments. About 78% of the sentences in the Treebank were reported to have been annotated with a single f-structure whereas the rest received several fragmented f-structures or none. However, no results were available with respect to the quality of the f-structures generated and there was no provision for phenomena marked in terms of traces in the Penn-II annotation.

Our Approach

Our intention was to develop a method for extracting predicate argument structures that would avoid the problems posed by the classical technique i.e. requiring complete coverage, and that would also be economical in the number of templates required. We also required the process to be compatible with the use of external knowledge sources (dictionaries, thesauri, etc.) at a later stage without requiring any reimplementation.

The key idea in our method is to avoid the complexity entailed by the original tree representation, requiring a recursive application of semantic rules, or the complicated application of templates to non-contiguous portions of tree, and instead to represent the tree structures in a logically equivalent 'flat' representation which allows access to all levels of the tree simultaneously. This technique will apply to any hierarchically structured object like a tree or a logic term.

In a flat representation of a term, the main functor is indexed, each of its arguments is replaced by a new index, and this process is applied recursively, in a depth-first, typically left-to-right fashion throughout the term. Thus a structure like:

likes(father-of(mary),john)

would be represented as

0:likes(1,3), 1:father-of(2), 2:mary, 3:john

This representational technique goes back at least as far as the various 'path-indexing' schemes for terms developed in the logic programming and theorem proving community (Stickel, 1989) and has been used several times within computational linguistics, e.g. (Whitelock, 1992; Copestake et al., 1995).

In applying this technique to the Treebank, we first 'Prologised' the trees, so that they took the form of embedded lists. We further 'Featurised' the lists to separate POS tags from grammatical function and semantic role tags and allow for a clean cut distinction between the POS label of a node, its features and its value, or constituents in the case of non-terminal nodes. The features correspond to the information encoded in the tags of the Treebank and are separated by the POS information by means of hashes. They include grammatical function and semantic role, type of trace where applicable, a flag for gap threading and indices to the nodes providing the information for the resolution of gaps and traces. We also used the morphological component of WordNet 1.6 to add morphological stem information that is included in the features as 'root=*'. (We have also slightly changed the tagset, incorporating distinctions between different types of determiners and prepositions).
A sentence from the featurised version of the Treebank can be found below:

```
[['S':[],
 ['NP':[fn= 'SBJ'],
  ['DET':[wh= n],'The'],
  ['NN':[root= turf],turf]],
 ['VP':[],
  ['VBZ':[root= have],has],
    ['VP':[],
      ['VBN':[root= range],ranged],
      ['PP':[fn= 'CLR'],
       ['PP':[],
        ['PREP':[],from],
         ['NP':[],['NNP':[],'Chile']]],
       ['PP':[],
```

```
['TO':[],to],
 ['NP':[],['NNP':[],'Austria']]]]]]]
```

Each file now contains sentences in the form of Prolog predicates such as the above. Next we flatten the structure to give:

```
0 : 'S'([],[1,4]),
1 : 'NP'([fn= 'SBJ'],[2,3]),
2 : 'DET'([wh= n],['The']
3 : 'NN'([root= turf],[turf]),
4 : 'VP'([],[5,6]),
5 : 'VBZ'([root= have],[has]),
6 : 'VP'([],[7,8]),
7 : 'VBN'([root= range],[ranged]),
8 : 'PP'([fn= 'CLR'],[9,13]),
9 : 'PP'([],[10,11]),
10 : 'PREP'([],[from]),
11 : 'NP'([],[12]),
12 : 'NNP'([],['Chile']),
13 : 'PP'([],[14,15]),
14 : 'TO'([],[to]),
15 : 'NP'([],[16]),
16 : 'NNP'([],['Austria'])
```

Retrieving proposition components

We build predicate-argument structures in three stages. Firstly, we process moved elements and gaps so as to produce a kind of canonicalised version of the corpus in which movements are undone and gaps are filled. This is described later. Secondly, we pull out of the flattened tree the constituents that supply the components of the predication. Thirdly, we convert those constituents into logical forms (actually, quasi-logical forms (QLF), since further contextual processing would be needed to have something directly evaluable for truth).

Having the sentence tree structure available in a flattened list format makes it possible to define, in a clean declarative way, various predicates on trees using traditional linguistic notions like 'dominates', 'precedes', 'c-commands', etc. Given definitions of these predicates it is easy now to write declarative rules which will pick out various structural configurations, such as the following:

Rule for locating the subject of a sentence:

"An NP containing the feature 'SBJ' is the subject of a sentence S which most immediately dominates it" (The extra Treebank annotation does most of the work here).

Rule for finding the main verb:

"The main verb of a sentence S is the V node most directly dominated by the VP which is dominated by the S node. The V cannot have a VP sister unless the latter dominates an infinitive".

Rule for finding a direct object:

"An NP sister to a main verb, that is not marked 'TMP', and where the verb immediately precedes the NP, is the direct object of that verb".

We do not fully distinguish complements from adjuncts at this stage since the distinction is not made in the Treebank. Since the rules are purely declarative we anticipate that it will be an easy matter to interface them with appropriate lexical information at this stage of processing later. Currently we use just an initial heuristic that a non-temporal NP immediately after a verb is a direct object, and anything else we treat as an adjunct, except for predicative adjective phrases and sentential complements. There are a total of just 8 structural rules and predicates which are very robust (i.e. they yield some kind of result for every non- fragmented sentence in the corpus) and together serve to extract the main sentence components, i.e. subject, verb, and complements/adjuncts.

Thus for the above sentence we get the following information about the sentence components:

```
VERB:  7 : VBN([root=range],[ranged])
SUBJECT: 1 : NP([fn=SBJ],[2,3])
COMPLEMENTS & ADJUNCTS:
[8 : 'PP'([fn= 'CLR'],[9,13])]
```

The rule for the extraction of proposition constituents will apply recursively into any complements until all main verbs and their arguments are retrieved from a sentence.

Treatment of Null Elements

An issue discussed extensively in (Marcus et al., 1994) is the representation of null elements, originating from WH-movement, passive constructions, infinitival clauses and topicalised ar-

guments. The former are co-indexed with the nodes that lend them their meaning.

In order to obtain viable logical forms for the predicate argument structures, one needs to resolve such null elements. Rather than leaving this task for the stage of the actual logical form construction (qlf), we decided to apply a pre-processing trace resolution stage to the Treebank and deal with empty elements originating in infinitives, wh-movement and passives while creating a new flattened version of the corpus. We refrained from treating more configurations than the previously mentioned as the latter seem to amount to a high percentage of cases and resolution is more straightforward than in the case e.g. of extraposed sentences. The rules involved in this first stage are exactly the same kind of thing as we saw above, using predicates on trees, along with the Treebank indexing system, to de-transform transformed structures. (Two exceptions to this are ellipsis and logical subject detection in passives, which we deal with instead by the third stage.)

In the null element phase there are 4 rules for infinitives, 4 for cases of wh-movement and 1 for passives. Below is an example of a flattened passive sentence before and after the application of trace resolution.

Before trace resolution:

```
0 : 'S'([],[1,5]),
1 : 'NP'([fn= 'SBJ',idx= 47],[2,3,4]),
2 : 'JJS'([],[most]),
3 : 'NN'([root= country],[country]),
4 : 'NNS'([root= fund],[funds]),
5 : 'VP'([],[6,7]),
6 : 'VBD'([root= be],[were]),
7 : 'VP'([],[8,9,10]),
8 : 'VBN'([root= clobber],[clobbered]),
9 : 'NP'([trace=*,idx=47],['**trace**']),
10 : 'PP'([fn= 'TMP'],[11,12]),
11 : 'PREP'([],[after]),
12 : 'NP'([],[13,14,15]),
13 : 'DET'([wh= n],[the]),
14 : 'CD'([],['1987']),
15 : 'NN'([root= crash],[crash])
```

After trace resolution:

```
0 : 'S'([],[1,5]),
1 : 'NP'([fn= 'SBJ',idx= 47],[2,3,4]),
2 : 'JJS'([],[most]),
3 : 'NN'([root= country],[country]),
4 : 'NNS'([root= fund],[funds]),
5 : 'VP'([],[6,7]),
6 : 'VBD'([root= be],[were]),
7 : 'VP'([],[8,9,10]),
8 : 'VBN'([root= clobber],[clobbered]),
9 : 'NP'([trace= *,idx= 47],[2,3,4]),
10 : 'PP'([fn= 'TMP'],[11,12]),
11 : 'PREP'([],[after]),
12 : 'NP'([],[13,14,15]),
13 : 'DET'([wh= n],[the]),
14 : 'CD'([],['1987']),
15 : 'NN'([root= crash],[crash]),
```

In the latter case the value of the empty 'NP', 9 :'NP' has been replaced by the value of the co-indexed 'NP', 2:'NP' while the trace information of the empty 'NP' is maintained in the features. For instances of unindexed traces or omitted logical subjects the value of the corresponding 'NP' is set to 'unspecified'.

Thus, the proposition components in the latter case will be identified as:

```
VERB: 8 : VBN([root=clobber],[clobbered])
SUBJECT: 1 : NP([fn=SBJ,idx=47],[unspecified])
COMPLEMENTS & ADJUNCTS:
[9 : NP([trace= *,idx=47],[2,3,4]),
 10 : PP([fn= 'TMP'],[11,12])]
```

Constituent to QLF Rules

Once the principal sentence components have been identified, a set of rules apply that detect the head word of each component and then use these head words to construct the QLF of the sentence. The core of each sentence is the main verb, which is always represented as a two, three or four place predicate, with the root of the verb as the functor, an event index individual to each verb in the sentence as its 1st argument and the head word of the subject as its 2nd argument. A check is performed to determine whether there exist a direct object, adjective phrase, or sentence in the list of verb complements and if so, their head words, or the results of a recursion, are added as arguments to the verb predicate.

The rest of the members of the complements-adjuncts list (see previous section) are considered to be verb modifiers and are represented as predications of the preposition over the verb event index and the head noun (in the case of a PP adjunct) or, in the case of an adjunct NP, as a predicate of its semantic role over the event index and the head noun.

There is a relatively small number of head word extraction rules, namely 20 for NPs, 6 for

ADJPs and 5 for PPs. Notice that since we are at this stage ignoring many modifiers and going just for the main components of a predication this number will increase if we want our logical forms to be more fine-grained.

We now give some examples of output from the system:

```
['Apple','II',owners,',',for,example,',',
 had,to,use,their,television,sets,
 as,screens,and,stored,data,on,audiocassettes]
Qlf:
 have(e1,owner, (use(e3,owner,set),
                   and  as(e3,screen)),
                 and
                 (store(e2,owner,datum),
                   and on(e2,audiocassette))))

['At',the,end,of,'World','War','II',',',
'Germany',surrendered,before,'Japan']

Qlf: surrender(e1,'Germany')
     and before(e1,'Japan')
     and 'At'(e1,of(end,'World_War_II')
```

As one can observe from the above examples, the qlfs consist strictly of head words, so that most modifiers of NPs have been omitted, with the exception of "NP of NP" expressions. Modifier qlfs could be added at any time, by including a few more rules but our initial aim was simply to get the method working quickly on a wide range of sentences. (Note that the ellipsis 'before Japan' is not fully resolved: even with the extended Treebank annotation there is not enough information to resolve this type of ellipsis).

1 Evaluation

It is impossible to measure fully the accuracy of our procedure in terms of the usual precision and recall ratios, because we have no independent gold standard to measure against. A raw count of the percentage of sentences in the Treebank that we get useful logical forms from gives a misleadingly high picture: we get something useful from almost every sentence, but neither our recall nor our precision are perfect.

To attempt a more realistic evaluation we took 100 sentences at random from the Treebank and annotated them by hand, counting the number of distinct predicates that should be found, the number of arguments expected for each predicate, and the level of embedding of the predicates. The latter is necessary in order not to double count errors: for example, if the correct QLF should be:

```
['The',organization,is,scheduled,to,meet,
 in,'Vienna',beginning,'Nov.','25','.']

schedule(e1,
         unspecified,
         organization,
         (meet(e2,organization)
           and in(e2,vienna)
           and temporal(e2,'Nov._25')))
```

then if the system makes an error in the QLF of the level 2 embedded clause `(meet(e2,organisation) and ....)` we want that to count only as an error for the lower clause and not also for the level 1 root clause, even though in a sense the root clause now has an inaccurate or incomplete argument.

Then we looked at what the system had actually given for these sentences. The raw recall scores are:

Level	Predicates	Arguments
1	90.3	90
2	87.76	95.7
3	80.9	93.78
Total:	86.32	93.16

There is a 4th level of predicate embedding as well, but there weren't enough instances to yield results comparable to the first three levels. We have not calculated a score for precision, although one can argue that the score for arguments provides a measure for predicate precision. With a very few systematic exceptions, errors always consist in missing a predicate or argument, not in getting the wrong one.

Most errors fall into one of the following categories: (1) errors in Treebank annotation (2) incomplete treatment of negation (3) VPs in nominalisations are not always captured (4) we omitted to write a rule to cope with sentential subjects (5) our morphology procedure makes errors on 's and some other contractions (6) incorrect embedding of sentential verb modifiers (7) incomplete numeric expressions (8) inadequacy of the rule for possessives. Categories (3) and (6) account for the most errors though the amount of error due to inadequate trace information in the Treebank is not negligeable.

Conclusion and Further Work

We have described a method for extracting predicate-argument structures from Treebanks which is flexible, robust, and seems to work well in practice. It has several advantages, in our view:

- The rules needed are declarative, non-deterministic and relatively few in number (although this clearly depends on how fine-grained we require the results to be).
- Multiple external information sources (lexicons, thesauri, statistical data) can be cleanly integrated.
- The system does not require complete syntactic coverage to produce useful results: even on our first run we managed to get something useful from practically every sentence in the Treebank.

Our next step, after developing the current rule set further, will be to integrate some lexical information so that we can do a better job with verb subcategorisation information. We will also try to do some sense disambiguation of a limited type.

For somewhat different purposes, we also aim to export the flattened version of the Treebank into a relational database format: it is intuitively clear that the literals of the flat representation could be encoded as database records allowing us, with the development of a suitable set of SQL macros, to perform complex searches like 'find all instances of an embedded verb modified by 'only' and with a sentential complement' very efficiently, without needing custom-built tree searching tools like 'tgrep'.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley Framenet project. In *Proceedings of the COLING-ACL*, Montreal, Canada.

Aoife Cahill, Mairead McCarthy, Josef van Genabith, and Andy Way. 2002. Automatic Annotation of the Penn-Treebank with LFG F-Structure Information. In Simonetta Montemagni Allesandro Lenci and Vito Pirelli, editors, *Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data, Proceedings of the Workshop, The 3rd International Conference of Language Resources and Evaluation (LREC)*, pages 8–15, Las Palmas, Spain.

Anne Copestake, Dan Flickinger, Rob Malouf, Riehemann Susanne, and Ivan Sag. 1995. Translation using minimal recursion semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-6)*, Leuven, Belgium.

Anette Frank, Luisa Sadler, Josef van Genabith, and Andy Way. 2002. From Treebank Resources to LFG F-Structures. In Anne Abeille, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Press, Dordrecht, The Netherlands.

Daniel Gildea and Daniel Jurafsky. 2000. Automatic Labeling of Semantic Roles. In *Proceedings of ACL 2000*, Hong Kong.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

George A. Miller. 1995. "Wordnet: a lexical database for English.". In *Communications of the ACM*, volume 38 (11), pages 39 –41.

Martha Palmer, Hoa Dang Trang, and Joseph Rosenzweig. 2000. Semantic Tagging for the Penn Treebank. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, Athens, Greece.

Martha Palmer, Joseph Rosenzweig, and Sam Cotton. 2001. Automatic Predicate Argument Analysis of the Penn Treebank. In *Proceedings of Human Language Technology Conference*, San Diego, California.

Mark E. Stickel. 1989. The path-indexing method for indexing terms. Technical Report Technical Report 473, SRI International, Menlo Park, CA.

P. Whitelock. 1992. Shake and Bake translation. In *Proceedings of the 14th International Conference on Computational Linguistics*, Nantes, France.