Framework Independent Summarized Parser Output in XML and its Example-based Documentation

Tam Wai Lok
University of Tokyo

Miyao Yusuke
University of Tokyo

Tsujii Jun'ichi
University of Tokyo
University of Manchester
National Centre for Text Mining

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

We see a communication problem between the grammar engineering community and the NLP community. The information to be communicated is the results produced by a grammar. This paper is about our solution to the problem. Our solution has two components: an alternative output format and its documentation. Our alternative output format carries constituency information that parsers are built for computing, but in lesser quantity and a simpler form than the standard attribute-value matrix (AVM) output format. The documentation for it provides a shallow and static explanation different from the deep and dynamic explanation found in literature about grammar formalisms meant for grammar writers. The shallow and static explanation is meant to enable members of the NLP community to achieve a shallow level of understanding of the results produced by a grammar for the sake of developing NLP systems that interoperate with parsers.

1 Introduction

Grammar engineering presents an especially difficult tension between grammar writers who are predominantly interested in carrying research in the field forward and developers who build NLP systems that interoperate with deep parsers but are not very interested in the grammars behind them. The source of this tension is that the results produced by grammars are not designed and documented as a language resource for the wider NLP community. Members of the grammar engineering community can proceed with their research without documentation that explains the meaning of the results produced by a grammar. The common knowledge acquired from the literature about the formalism on which a grammar is based and shared among them in the computation of the results render such documentation unnecessary for the grammar writers. However, given that grammars are built for practical use in the development of larger NLP systems, the paucity of documentation for users who should not need to acquire the common knowledge shared among members of the grammar engineering community and the design of the output format of deep parsers which require such knowledge for deciphering the results are practical problems, if not theoretical ones. In this paper, we present a solution to these practical problems. It is our hope that our work can draw the attention of the grammar engineering community to the need of the wider NLP community for a simpler design of and documentation for the results produced by a grammar.

We are not being critical of the non-existence of documentation for grammar writers. There may not be a practical problem in that area as long as members of the grammar engineering community can carry on with their work by relying on common knowledge shared among them and on the literature about the grammar formalisms on which their work is based. Such documentation, while good to have for the sake of new members of the grammar engineering community, is not a solution to the practical problem in communicating the results produced by the grammar engineering community to the wider NLP community. Developers of NLP systems outside the grammar engineering community are not equipped with the background knowledge needed for understanding such documentation. The solution we present here is meant for these developers who share knowledge about linguistic concepts like POS, semantic representations and subcategorization with grammar writers but lack the knowledge in a specific framework required for finding information about these concepts from framework specific representations.

Our solution is built on top of ENJU (Miyao et al. [2004]). ENJU is built with a view to being a practical parser that accepts real text and forms a part of larger NLP systems. It includes a mostly induced, partly handcrafted grammar which keeps grammar engineering work at a minimum. This is in part why we are less concerned with meeting the needs of grammar writers but more concerned with providing support for

use in NLP system development. This support is provided by means of an alternative output format which carries information summarized from that carried in the standard AVM output format and documentation for the alternative output format.

To illustrate what is kept and what is left out in our summary of a complete feature structure representation, we give the representation of the relative pronoun ”who” in the AVM format and the representation of it in our alternative output format one after the other.

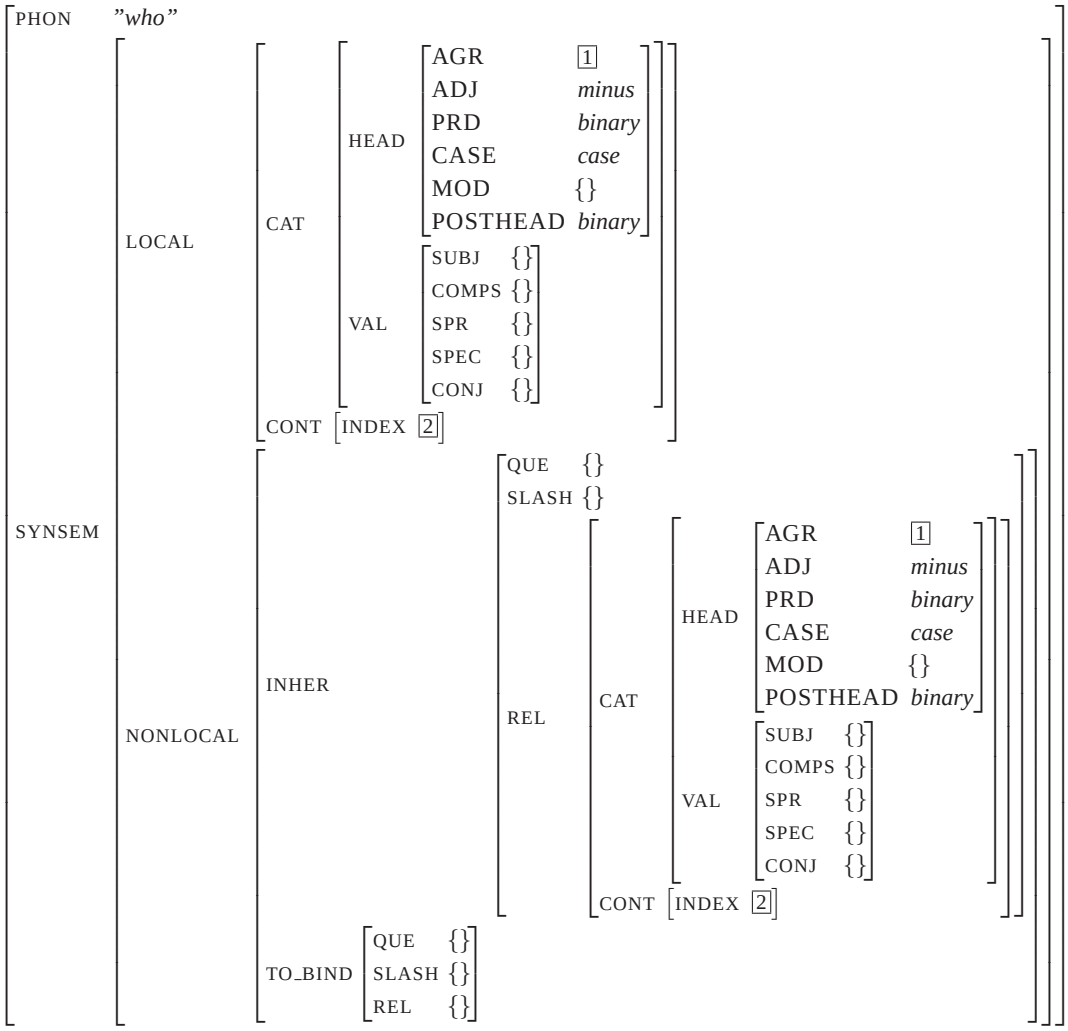

Figure 1: the complete AVM representation of ”who”

```
1 <tok id="t4" cat="N" pos="WP" base="who" lexentry="N.3sg/[&lt;NP.3sg&gt;]" pred
    ="relative_arg1" arg1="c8">
  who
3 </tok>
```

As an alternative to the standard AVM output format, our format is different from MRS (Copestake et al. [2005]), another alternative output format but similar to the bracketing style used in the Penn Treebank. While

both MRS and the Penn Treebank bracketing style are grounded in linguistic theories, the Penn Treebank bracketing style is meant to be understood (at a shallow level) without in-depth knowledge of the theories whereas MRS is meant to be understood with deep knowledge of them. This difference is obvious in the literature about the two formats. The Penn Treebank annotation manual (Bies [1995]) provides a large number of examples without explaining how they are computed with the transformationalist theories. MRS comes with a paper that describes in details the steps for computing a MRS representation and the theories on which the computation is grounded (Copestake et al. [2005]). For the Penn Treebank, the point of enabling users to achieve a shallow level of understanding of the analysis by annotators is to help developers to debug NLP systems that use the Penn Treebank as a language resource by giving them an idea how the analysis of a linguistic phenomenon looks, not to enable them to do the computation done by annotators. Likewise, we want to enable our users to achieve a shallow level of understanding of the results produced by our grammar for debugging NLP systems that use these results as a language resource: we do this by giving them an idea how the analysis of a linguistic phenomenon looks. Our goal is not to enable them to do the computation done by parsers and grammar writers.

This paper is organized as follows: We start with giving more details on the communication problem we mention above. Then we describe our solution by highlighting some of its characteristics and providing some examples. Finally, we conclude this paper with a summary and some thoughts on the direction our work is heading.

2 Problem definition

In the beginning of this paper, we describe the problem we are addressing as one of communication between the grammar engineering community and the wider NLP community. This kind of communication problems between research communities is not uncommon in the academic world. But the problem involving the grammar engineering community and the NLP community is particularly serious for two reasons.

The first reason is that substitute for more canonical documentation that serves members of the grammar engineering community well does not function well for members of the wider NLP community. By substitute of documentation, we are referring to textbooks that introduce students to a formalism like (Sag et al. [2003]) or handbooks that cover everything essential about a formalism like (Pollard and Sag [1994]). Literature does not function well for members of the wider NLP communities because the parser results are different from those in papers. The former comes with much more information than the latter. This is because linguists who propose these formalisms omit feature-value pairs they consider irrelevant to the linguistic phenomena they are interested in when they present the computation in papers.

Let us illustrate the difference between the result produced by a parser and the analysis given on paper with an example. A sign of any POS carries the *MOD* feature in HPSG. A noun or a verb that does not function as an adjunct is assigned an empty value for this feature. When talking about control and raising, linguists know that there is not much point in specifying the *MOD* value of the control (raising) verb and its NP arguments on paper. However, parsers do not know this. They can only display all feature-value pairs or rely on users to choose which features to display. The knowledge required for filling in the gaps between the output of parsers and the output given on paper, like the common knowledge that members of the grammar engineering community rely on for carrying research in their field forward, is missing for members of the wider NLP community. This renders the literature about the grammar formalism on which a grammar is based less useful for members of the wider NLP community.

The second reason is that there is an explosion in the information being communicated between (the systems built by) the two communities. The grammar engineering community places little restriction on the introduction of new features for covering new phenomena in a grammar. Often the new features are included in the feature structure representations of all signs. So the introduction of new features for covering a new phenomenon does not only put more information in the feature structure representations of sentences related to the phenomenon for which the features are introduced. It also puts more information in the feature structure representations of sentences not related to the phenomenon. The result of this is an explosion of information. With wide-coverage being the pursuit of the grammar engineering community, we are witnessing such explosions in every well-known deep parser. For example, the features structure representation of example sentence 1 has more than 500 feature-value pairs in the output produced by ENJU, the deep parser we use.

(1) John is the man who Mary loves

Common current attempts at providing a solution to the communication problem we identify here are not satisfactory in two aspects:

- Reducing the information to be communicated to (the systems built by) the NLP community is recognized as a means of providing a solution to the problem. However, the information left to be communicated to the NLP community is very often packed in a new format which demand them to acquire new knowledge for the purpose of making sense of and using the packed information. One such new format is MRS. It may be true that the design of a new format is inevitable for the purpose of packing the information to be communicated. However, the reduced information is often in an unfamiliar format. If such a format is significantly different from what developers are familiar with it would create the same hurdle created by the original grammar frameworks.

- MRS and dependencies are two formats sometimes cited as a solution to the problem. Both formats carry no constituency information, which is the information parsers are supposed to compute according to the widely accepted definition of a parser as a program that identifies the phrase structure of an input sentence. As a result, many other research communities and systems built by them expect this information from parsers and the research community working on parsers. An example of NLP systems that needs constituency information from parsers is a speech synthesiser. It needs the phrase structure of an input sentence to determine the prosodic structure of it. Providing constituency information with other information would help to solve the communication problem between the grammar engineering community and the NLP community.

3 Solution

3.1 Our alternative output format: summarized parser output format

Our alternative output format has the following characteristics:

Fixed number of attributes In feature structure based grammar formalisms, the number of attributes of every sign increases proportionally with the coverage of a grammar. In our simplified output format, we define a fixed set of attributes for terminal nodes and a fixed set of attributes for non-terminal nodes.

Framework independent attribute names In feature structure based grammar formalisms, features may be embedded as the value of some other features. Path information, that is, the names of all the embedding features of a feature, is needed for identifying the embedded feature. Different feature structure based formalisms have different paths and names for features that carry similar information. In our simplified output format, attributes take atomic values and are given framework independent names based on the type of information they carry.

Hidden value-sharing In feature structure based grammar formalisms, unification of values occurs between features found in multiple locations, essentially repeating the same information. In our simplified output format, inheritance of attribute values from a daughter node to its mother is not shown. Only sharing of values between sisters and constituents in a long distance dependency relation is visible. The visibility of value-sharing of the later kind is enough for capturing a wide range of linguistic phenomena.

It is not difficult to see that these three characteristics deal with the following sources of complaints about the complete AVM output:

1. There are too many feature-value pairs in a feature structure representation of a constituent.
2. It is difficult to tell what kind of information is contained in an embedded feature with a long path name.
3. The sharing of values between features in a large feature structure is difficult to trace and make sense of.
4. The same piece of information appears in multiple locations.

These complaints are not only about the quantity of information represented in the complete output. Some of these complaints are about the way information is carried. HPSG allows phrase structure trees whose non-root nodes carry information produced by the parsing of large constituents. For example, the SYNSEM|LOCAL|CONT|LOVER feature of the root node ”loves” in a HPSG-style phrase structure tree of example sentence 1 is assigned the
SYNSEM|LOCAL|CONT|INDEX value of the root node ”Mary”. This information is produced by the parsing of the nonterminal embedded sentence node ”Mary loves”.

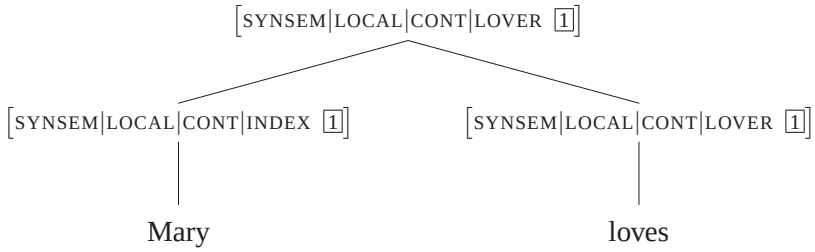

Figure 2: Mary loves

In lambda calculus based semantics found in other frameworks like LFG (Dalrymple [2001]) and CCG (Steedman [2000]), the agent role of the semantic representation of the root node ”loves” would not be filled by the reference marker of the root node ”Mary”. Instead, it would remain uninstantiated and the variable corresponding to the argument slot would be marked by a λ:

$$\lambda X.\lambda Y.love(X, Y)$$

Grammar writers who are familiar with the HPSG formalism have little problem in understanding why the agent role of the semantic representation of the root node "loves" is filled by the reference marker of the root node "Mary" in a HPSG-style phrase structure tree. However, developers in the NLP community may find it confusing. The problem is that the complete AVM output includes steps in the computation but these steps look different from the way they look during the computation. To address this problem, we remove these steps from our alternative output format by hiding the sharing of values.

Our alternative output format represents an attempt to simplify feature structure based grammar formalisms without sacrificing the power of deep processing in capturing linguistic phenomena like long distance dependencies and raising which proves difficult for shallow processing. Our approach is different from the approach of output formats like MRS and dependencies. We try to summarize the complete AVM output. They extract some specific information (e.g. semantics in the case of MRS) from the complete AVM output. For this reason, we name our output format **Summarized Parser Output** (SPO). In SPO, it is possible to distinguish between the output produced by parsing example sentence 1 and the output produced by parsing:

(2) John is a man and Mary loves John

Capturing this difference between different constructions is what we mean by capturing linguistic phenomena. This is important for the output of a parser. In other alternative output formats, it would be impossible to distinguish output produced by sentences with the same meaning or the same dependencies between constituents.

SPO is meant to be a format for making it easy to use the parser results in the development of NLP systems. During the development of an NLP system that interoperates with a parser, developers are not involved in the computation done by the parser but they are often required to check the results produced by the parser for debugging purpose. Textbooks and handbooks which explain how the computation is done do not meet their need. They need a large collection of examples in the style of the Penn Treebank manual against which they can check the results produced by the parser without doing the computation. Therefore, our documentation for SPO is modelled on the Penn Treebank manual.

3.1.1 Specifications of SPO

Nodes of a phrase structure tree in feature structure based formalisms are structured complexes of features and values. These nodes are represented by XML elements in SPO. The structure of a parse tree is determined by mother-daughter relations and sister relations between its nodes. The two relations are captured in the following way:

mother-daughter relations Two nodes in a mother-daughter relation are represented by an enclosure relation between two *cons* elements. The node represented by the enclosed element is the daughter. The node represented by the enclosing element is the mother.

sister relations Two nodes in a sister relation are represented by two non-mutually-enclosing *cons* elements which are both enclosed by the same *cons* element.

A *cons* element can represent the root node, a terminal node and a nonterminal node. The outermost *cons* element represents the root node. A leaf node of a parse tree is represented by a *tok* element.

To enable our readers to visualize what we have just described, we give the following empty template with all attributes except *id* removed .

```
<cons id="c1">
 <!--this is the root node -->
 <!--this is the mother of the constituents represented by c2 and c3 -->
        <cons id="c2">
          <!--this is the daughter of the constituent represented by c1 -->
          <!--this is the sister of the constituent represented by c3 -->
          <tok id="w1">
          <!--this is a leaf node -->
          </tok>
        </cons>
        <cons id="c3">
          <!--this is the daughter of the constituent represented by c1 -->
          <!--this is the sister of the constituent represented by c2 -->
         <tok id="w2">
         <!-- this is a leaf node-->
         </tok>
        </cons>
</cons>
```

As for the attributes carried by the *cons* and *tok* elements:

Attributes carried by both *cons* and *tok* elements POS information (*cat*), reference marker (*id*)

Attributes carried only by *cons* elements syntactic head (*head*), semantic head (*sem_head*), the rule responsible for rewriting an element as its daughter (s) (*schema*)

Attribute carried only by *tok* elements base form (*base*), references to lexical rules or lexical entries (*lex-entry*), tense (*tense*), aspect (*aspect*), verb type (*aux*), the argument variables to which the semantic representations of the corresponding nodes apply (*argn*), semantic representation (*pred*)

Attributes like *tense*, *aspect* and *voice*, which correspond to features whose values are passed up from the lexical entry of a verb to the terminal verb node and from a terminal verb node to a non-terminal verb phrase node in a phrase structure tree of feature structures, are not included as attributes of the *cons* elements which we use for representing terminal nodes and non-terminal nodes. This is what we mean by hiding values shared between a mother and a daughter.

3.1.2 Summarizing features

The attributes of *cons* and *tok* elements are summarized from features of the corresponding nodes. Some features of a node are captured by straightforward one-to-one conversion. Others are captured by generalizing over a few features of the node and producing one attribute in the corresponding XML representation for several features of the node. The rest are simply not represented in the XML form.

The case of one-to-one conversion and the case of neglecting a feature are trivial but the idea of generalizing over several features of a node requires some explanation. To illustrate, let us take the *cat* attribute of a *cons* element or a *tok* element as an example. Its value is determined by both the value of the SYNSEM|LOCAL|CAT|HEAD feature and the value of the SYNSEM|LOCAL|CAT|SUBCAT feature of the corresponding node. We say the *cat* attribute is a generalization over the HEAD feature and the SUBCAT feature. By neglecting some features and generalizing over others, we greatly reduce the number of attributes in SPO while keeping the number of elements the same as the number of nodes in the parse tree being represented by it.

3.1.3 An example

The specifications and the methods of summarization produce less expressive power in exchange for reducing complexity. But they can be used creatively for capturing a wide range of linguistic phenomena in a deep but simple way. Let us illustrate how this can be done with our analysis of the relative clause contained in example sentence (1).

```
<cons id="c36" cat="NX" xcat="" head="c37" sem_head="c37" schema="
    head_relative">
  <cons id="c37" cat="NX" xcat="" head="t16" sem_head="t16">
    <tok id="t16" cat="N" pos="NN" base="man" lexentry="[D&lt;N.3sg&gt;]_lxm"
        pred="noun_arg0">
      man</tok></cons>
  <cons id="c38" cat="S" xcat="REL" head="c40" sem_head="c40" schema="
      filler_head">
    <cons id="c39" cat="NP" xcat="REL" head="t17" sem_head="t17">
      <tok id="t17" cat="N" pos="WP" base="who" lexentry="N.3sg/[&lt;NP.3sg&gt
          ;]" pred="relative_arg1" arg1="c37">
        who</tok></cons>
    <cons id="c40" cat="S" xcat="TRACE" head="c43" sem_head="c43" schema="
        subj_head">
      <cons id="c41" cat="NP" xcat="" head="c42" sem_head="c42" schema="
          empty_spec_head">
        <cons id="c42" cat="NX" xcat="" head="t18" sem_head="t18">
          <tok id="t18" cat="N" pos="NNP" base="mary" lexentry="[D&lt;N.3sg&gt
              ;]_lxm" pred="noun_arg0">
            Mary</tok></cons></cons>
      <cons id="c43" cat="VP" xcat="TRACE" head="t19" sem_head="t19">
        <tok id="t19" cat="V" pos="VBZ" base="love" tense="present" aspect="
            none" voice="active" aux="minus" lexentry="[NP.nom&lt;V.bse&gt;NP.
            acc]_lxm-movement_rule-singular3rd_verb_rule" pred="verb_arg12" arg2
            ="c37" arg1="c41">
          loves</tok></cons></cons></cons></cons>
```

We make use of the idea of gaps in our analysis of relative clauses. In HPSG, this is done by introducing the SYNSEM|NONLOCAL|INHER|SLASH feature, the SYNSEM|NONLOCAL|INHER|REL feature and the SYNSEM|NONLOCAL|TO-BIND|SLASH feature. We try to do this without introducing any new attribute. A gap is formed when the relativized argument (object) of the embedded verb (”loves”) is removed from the subcategorization frame temporarily in the phrasal projection of the verb is formed without the argument being

sister to the verb. The phrasal projection of the verb formed as a result is gapped. A gapped verb phrase is simply marked by being assigned a *cat* value which says something different from the XML representation of the phrase structure of the sentence in question about the subcategorization frame of the verb. In the XML representation of example sentence (1), the lexical entry of the transitive verb ”love” (t19) is dominated by a verb phrase node (c43) whose subcategorization frame contains only a subject. This is indicated by its *cat* value VP. But we cannot find any other element that is enclosed by the element representing the verb phrase node. This is what we mean by having the *cat* value of a verb phrase saying something different from the phrase structure.

The semantic representation of the embedded verb ”loves” is given as the value of the *pred* attribute of the lexical entry of ”loves” (t19). Its theme role is represented by the *arg2* attribute of t19, which is assigned the *id* value c37 of the nonterminal head noun node. *id* can be understood as the entity a constituent refers to. Two different *id*s refer to the same entity if they come from two elements one of whose *id* value is assigned as the *sem_head* value of the other. So the *id* value c37 of the terminal noun node and *id* value t16 of the root node ”man” refers to the same entity.

3.2 Shallow documentation for parser output

In this section, we first provide a more detailed description of our example-based documentation and give an excerpt of it to illustrate the difference between theory-centred literature and example-based documentation. Then we offer more explanation as to why the latter is better suited for developers in the NLP community.

Our documentation is indexed by linguistic phenomena. It is organized into sections, each of which includes:

1. a section title that describes a linguistic phenomenon
2. an example sentence that illustrates the linguistic phenomenon
3. the translation of the result produced by parsing the example sentence with our parser to a format based on the Penn Treebank bracketing style
4. explanation for our analysis of the linguistic phenomenon

Here is an excerpted section broken into the mentioned elements:

Section title non-subject wh-relatives

Example sentence (3) John is the man who Mary loves

Simplified output

```
     (S  (NP  (NX John))
        (VP  (VX is)
            (NP  (DP the)
                (NX  (NX man[id=c37]))
                (S-REL  (NP-REL who[pred=relative_arg1,arg1=c37])
                       (S-TRACE  (NP  (NX Mary[id=c41]))
                                 (VP-TRACE
    loves[pred=verb_arg12,arg1=c41,arg2=c37))))))
```

Explanation

Syntax

- The relative pronoun ”who” is assigned the POS label (cat) NP .
- The embedded transitive verb ”loves” forms a gapped verb phrase, which is assigned the POS label VP, with no daughters.
- The gapped verb phrase is sister to the subject noun phrase ”Mary”. Together they form the gapped sentence ”Mary loves’, which is assigned the POS label S.
- The gapped sentence is sister to the relative pronoun. Together they form the relative clause ”who Mary loves”, which is assigned the POS label S.
- The relative clause is sister to the head noun ”man”.

Semantics

- The object position (arg2) of the embedded transitive verb ”loves” is relativized. It is assigned the reference marker (id) of the head noun ”man” (c37).

Note the similarity in style to the Penn Treebank annotation manual. Our explanation and the explanation offered in the Penn Treebank annotation manual are shallow and static. A shallow explanation does not give the readers the reason for a certain output. For example, we do not tell our readers the reason that a certain attribute is assigned a certain value is because a particular feature structure unifies with another feature structure and some values of the features carried by them are shared. The Penn Treebank annotation manual does not account for the existence of a trace in a specific position in terms of transformations. A static explanation does not include the steps taken to compute the result. Such steps are transformations in a transformationalist framework and unifications in a feature structure based framework. Our explanation does not mention unification . Likewise, transformations are hardly mentioned in the Penn Treebank annotation manual.

Also note the difference in style between our explanation and the explanation offered for the analysis of linguistic phenomenon in textbooks like (Sag et al. [2003]), handbooks like Pollard and Sag [1994] and literature like Copestake et al. [2005]. The explanation offered in these textbooks, handbooks and literature meant for members of the grammar engineering community and hence is deep and dynamic. A deep explanation gives the readers the reason for a certain output. A dynamic explanation goes through the steps taken to compute the result meant to be explained.

The importance of deep and dynamic explanation for grammars is obvious. (A deep and dynamic explanation for the results necessarily becomes a holistic explanation for the grammar.) In order to understand how a grammar works, grammar writers have to know which feature structure unifies with which and what values are shared between them. It is the unification and the sharing that enable a grammar to rule out ungrammatical sentences and construct the meaning of a sentence from its parts. The existence of such explanations, which are so useful to grammar engineering, obviates the task of creating documentation that provides the same kind of explanation.

However, it is easy to underestimate the importance of shallow and static explanation for results produced by a grammar. Such documentation is a major means of communication between the producers and the consumers of the parser, less often the means of communication among the producers. The need of the consumers is determined by the purpose for which they use the parser results: in our case, this purpose is the development of NLP systems that interoperate with parsers. What is needed is a shallow understanding of the results produced by a grammar.

What is a shallow level of understanding of the results produced by a grammar? It is some ideas about what the correct analysis of a linguistic phenomenon looks like. We provide examples in our shallow explanation to allow developers to check their results. Likewise, the shallow explanation offered in the Penn Treebank annotation manual comes with examples that allows developers to check their results they get from systems trained with the treebank against the examples directly. There is no question about the usefulness of shallow explanation to developers because it is simply designed to meet their needs during development.

4 Conclusion and future work

We have outlined and illustrated with examples our solution to the communication problems between the grammar engineering community and the wider NLP community. We attempt to solve these problems by simplifying the output of a deep parser in an alternative output format and providing documentation for that format.

Our alternative output format SPO is different from alternative output formats proposed for other deep parsers in our concern with preserving the syntactic information in the AVM format. We preserve this information so that the output of our parser in its simple form can be used for a wide range of NLP applications. (Chun et al. [2006], Miyao et al. [2006], Yakushiji et al. [2006])

The documentation for SPO provides shallow and static explanations to developers in the NLP community. This differs from the deep and dynamic explanation found in literature that serves grammar writers well as documentation. Though not very useful to grammar engineering, shallow and static explanation is needed by developers in the NLP community for the purpose of building NLP systems that interoperate with parsers. In showing that there is no substitute for documentation meant for developers, we argue that documentation targeted at the NLP community is an urgent task for those developing parser for NLP applications.

Our idea of a simplified but information-rich output format and documentation of it for members of the NLP community presented here are tested on a partly-handcrafted grammar should help development of applications fed on the output of more handcrafted grammars like LKB/ERG Copestake and Flickinger [2000]) as well.

We create the summarized output format described in this paper by summarizing the output of a deep parser which do HPSG-based parsing. Our simplified output format can be used for summarizing the output of other deep parsers which use other grammar formalisms. In fact, our simplified output format has some similarities to LFG. For example, subcategorization information is implicitly represented by the POS label and argument slots of the semantic representation in our simplified output and in LFG. The idea of leaving information that can be read off the phrase structure tree (in XML format) unrepresented in attribute-value pairs is also similar to the idea of separating constituency information from the functional-structure. Currently, we are in talks with groups working on parsing in LFG to explore using the same output format for summarizing output of deep parsers based on different formalisms. Our next step would be to extend the use of our summarized output to parsers built on feature based CCG. A common output format between deep parsers based on different formalisms would be very useful for parser evaluation, if accompanied by documentation created in the manner described in this paper for the output produced by each of the deep parsers.

Acknowledgments

This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and Grant-in-Aid for Young Scientists (MEXT, Japan).

References

Ann Bies. Bracketing guidelines for treebank II style Penn treebank project, 1995. URL `citeseer.ist.psu.edu/bies95bracketing.html`.

Hong-Woo Chun, Yoshimasa Tsuruoka, Jin-Dong Kim, Rie Shiba, Naoki Nagata, Teruyoshi Hishiki, and Jun'ichi Tsujii. Extraction of gene-disease relations from Medline using domain dictionaries and machine learning. In *Proceedings of the Pacific Symposium on Biocomputing 2006*, pages 4–15, Maui, 2006.

Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage English grammar using hpsg. In *Proceedings of the Second conference on Language Resources and Evaluation*, Athens, Greece, 2000.

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. Minimal recursion semantics: An introduction. In *Journal of Research on Language and Computation*, volume 3, pages 281–332. Springer, 2005.

Mary Dalrymple. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press, 2001.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 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*, pages 684–693, Hong Kong, 2004.

Yusuke Miyao, Tomoko Ohta, Katsuya Masuda, Yoshimasa Tsuruoka, Kazuhiro Yoshida, Takashi Ninomiya Takashi, and Jun'ichi Tsujii. Semantic retrieval for the accurate identification of relational concepts in massive textbases. In *Proceedings of COLING-ACL 2006*, pages 1017–1024, Sydney, 2006.

Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1994.

Ivan A. Sag, Tom Wasow, and Emily M. Bender. *Syntactic Theory: A Formal Introduction*. CSLI Publications, second edition, 2003.

Mark Steedman. *The Syntactic Process*. MIT PRess, 2000.

Akane Yakushiji, Ysuke Miyao, Tomoko Ohta Tomoko, Yuka Tateisi, and Jun'ichi Tsujii. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 284–292, Sydney, 2006.