Hydras in Hiding:

Using Wordnet dictionaries to automatiaclly determine headedness

By Matthew Lou-Magnuson

For Dr.Francis Bond

HG7030
NLP for Linguists

Assignment1
March 17, 2014

Contents

1 Headedness & Phrase Structure

General typological and descriptive studies typically employ a logical and theory neutral approach to linguistic structure.[1] The basic idea is that within a given linguistic frame (e.g. a minimally simple transitive sentence) certain items can be freely replaced with others without creating ungrammatical structural errors. Applying such observations broadly, one can define classes and their members based on this notion of substitutability. Moving beyond the scope of individual items, it is often the case that whole combinations of items may also be able to be interchanged in identical contexts, thus giving rise to both multiplex and simplex categories. Within general linguistic theory, such simplex categories are taken as grammatical classes, while the multiplex combinations made of them are called phrasal types.

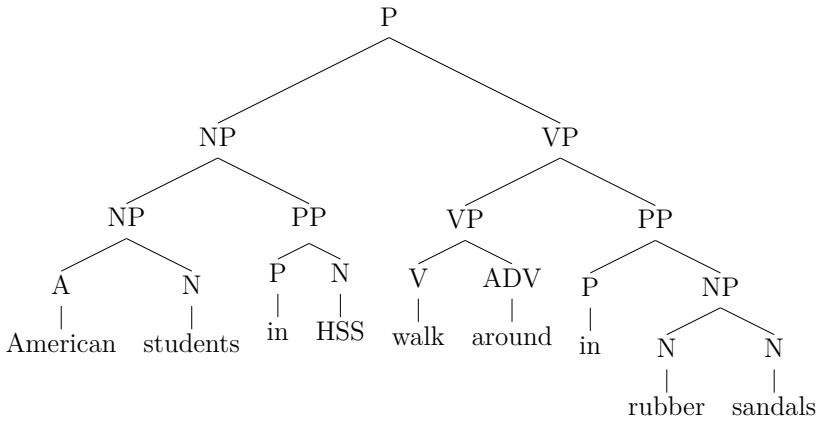

Figure 1: Phrase structure for 'American students in HSS walk around in rubber sandals.'

In the figure above, the bottom nodes of the tree each have a grammatical class label, and the higher intermediate nodes each represent a phrasal structure. The highest level is that of a complete predicate, which for English almost necessitates the union of an overt nominal and verbal phrase. For each phrase in the predicate there is a node somewhere below it in the tree that is of the same type, e.g. for the noun phrase that forms the subject of the above sentence there is the noun student below. This simplex element that roots the type of a given phrase above is referred to as the head of that phrase. Looking again to figure one, the heads of the top level noun and verb phrases are *students* and *walk*, respectively. Given that walk is an intransitive verb (requiring only the verb itself to make a complete verb phrase), everything but these two heads can be removed, yielding the essential structural core of the sentence - *students walk*.

Computers are well-suited to the semantically null but computationally intensive analysis required to produce a phrase structure grammar automatically. In the following sections and code, this paper shall investigate what information can be gleaned from a relatively compact and readily available data set found in Wordnet dictionaries.

[1]For a general overview see Tallerman (2011), and for more specific introductions, see Croft (2003) on typology and Payne (1997) on general description

2 Algorithm

Wordnets are small dictionaries, designed not only with lexical look-ups, but categorical information of both a semantic and syntactic nature. The specific Wordnets used in this project are described in the results section, and formatted and hosted by Dr. Francis Bond, Nanyang Technological University.[2] While entries often consist of single word lemmas, there are a number of multiword lemmas in the typical Wordnet also, typically serving to disambiguate a meaning where a single word entry cannot alone. Each lemma has a grammatical class assigned to it, and thus a multiword entry can serve as a phrase proxy; idiomatic constructions aside, the grammatical class of the lemma must match one of the words in the entry itself, just as a phrase in natural speech must have a head that gives it its syntactic type. One can then note the position of the word that matches the overall grammatical class of the lemma, namely, initial or final, and based on a running count estimate the headedness of a given phrase type.

When executing the above method, two general constraints were placed on the counting procedure. Firstly, unless one appeals to an outside dictionary of the language, each word in a multiword expression must have a counterpart in the single word entries in order to determine its grammatical class. Entries where either the initial or final word did not meet this criteria were excluded from the count as being insufficiently informative. Secondly, when encountering entries were both the initial and final word match the overall grammatical class of the lemma, one must appeal to semantics to determine which is the actual head. In these cases, rather than count each word once, the entry was excluded from the count so as to maintain a consistent one count per lemma ratio.

3 Issues

The simple match-and-count method described above yields reasonable results, but is not without issue. Chiefly, five problematic questions arrose when examining the data, and the changes to the basic procedure above are described below. While not all issues necessitated change in the data set presented here, areas likely relevant *a priori* are discussed as well.

3.1 Multiple Class Membership

The most problematic aspect with using a Wordnet dictionary is accounting for lexical items with multiple class memberships, e.g. English *bear* 'a ferocious woodlands creature' and *bear* 'to carry or to endure.' While the computer can of course tag each possible grammatical class, it is context and semantics that largely determine which is the correct choice for a given entry. Given a hypothetical entry such as *bear claws* even a human cannot determine if the correct reading is 'claws belonging to a bear' or 'the act of exposing claws' without context.

Thus as the computer is only able to readily use one grammatical class per single word

[2]Detailed information on the project is available online at `http://compling.hss.ntu.edu.sg/omw/` and in Bond and Foster (2013)

lemma, the problem becomes then how to select one option of the many that may exist for a given word, i.e. how to choose between the noun or verb usage of *bear*. The solution employed here is to randomize the choice for each word with multiple class membership. The analysis is run multiple times, and the results are taken cumulatively to average out the bias.

3.2 Open and Closed Classes

Another key observation came from initial testing with English, a strongly head initial language in verb phrases. The initial results were erratic, indicating mixed to moderate head final tendencies. Upon closer inspection of these supposedly head final lemmas, it became clear that a large number of the multiword verbal entries involve a verb and a particle, e.g. emph{open up}. If a handful of items from the class or prepositions (used here as verbal particles) are taken as their verbal counterparts (c.p. *I* upped *the anty*, *The hunter* downed *the deer*) then the data can become skewed. As such functional words form a closed lexical class, a decision was made to have all words that have a functional class membership never receive a chance to vary randomly in assignment; a word like English *up* can never be marked as a verb, adjective, or noun.

3.3 Duplex vs. Multiplex Lemmas

Across the multiple languages tested, the issue of phrases within phrases was found to be problematic, in particular prepositional phrases. As way of example, consider the following entry from the Italian Wordnet:

(1) *fatto* *a* *mano*
make.PP at hand
'handmade'

The expression involves a past participle, and while this is correctly classified as an adjective, it's derivation from a verb allows for a tighter integration of participle phrases. This leads to the situation where the adjective phrase is marked as head initial, despite the fact that Italian (and indeed all romance languages) are head final in such environments. Ideally the corpus would be rich enough to allow a mapping of such higher level combinations, but given the limited context of the Wordnet environment, the issue was resolved by limiting the multiword entries counted to only include two-member duplexes, rather than trying to extract higher level phrase patterns form the longer multiplex items. That is, only heads and one immediate dependent are allowed.

3.4 Partially or Non-Segmented Orthographies

When dealing with a language that relies heavily on agglutination, or a not-fully or not-at-all segmented orthography, the ability to use the single word entries of a Wordnet to split complex entries initially seems plausible. For example, consider the following example from the Norwegian Wordnet:

(2) *hovedkontor*
hoved- kontor
priciple- office
'main office'

(3) *forvente*
for- vente
ahead- wait.INF
'to wait for'

It seems that splitting a word into its subparts is procedurally straightforward, and so languages like Norwegian or Thai can be amenable to the same algorithm for headedness. In compounds, though, often the basic word form undergoes morphological change, or involves elements that are only used in compounds. As an example of the latter, the word *hoved* in number two is still listed in many dictionaries as an independent adjective, but in practice has become antiquated and restricted to compound forms only. So while *hoved* appears in numerous words in the Norwegian Wordnet, it has no single word entry. A second problem is when words that do have single word counterparts, are used as a different part of speech or an inseparable. The element *for-* in *forvente* in example three is a normal preposition in Norwegian, but a fused portion of the root in verbal derivations, and so using the single word entry would result in an incorrect split. While the Norwegian examples above deal with partially segmeted orthography (i.e. Norwegian, like English regularly permits a noun to modify another noun, but even for novel expressions will write them as a single orthographic entity), the same issues of using a singleword dictioanry to split mulitword expressions will apply to completly non-segmented orthographies like those used to write Chinese or Thai, unless information about the morphological split is written into the Wordnet before hand.

4 Results

The following results are from the Danish, English, Italian, French, Portuguese, Norwegian (Bokmål) and Spanish Wordnets provided online.[3] As explained above, all percentages are based on two-word multiword lemmas, and reflect the changes to the algorithm discussed in the issues section. The table below records the results of the final analysis of ten attempts, with the percentage and number of total responses displayed for each phrasal category. An asterisk indicates a false prediction of headedness, and a question mark indicates that while correct in the table, at least one previous analysis was incorrect.

[3]All files and metadata are taken from `http://compling.hss.ntu.edu.sg/omw/` (accessed March 10th, 2014), where further information can be seen in Sagot and Fišer (2008) for French, Gonzalez-Agirre et al. (2012) for Spanish, Pianta et al. (2002) for Italian, Fellbaum (1998) for English, *DanNet — det danske wordnet* (2014) for Danish, and Satre (2014) for Norwegian. Headedness was checked in the following grammars: Bradley and Mackenzie (2004) for Spanish, Crocker (2009) for French, Peyronel and Higgins (2006) for Italian, Hutchinson and Lloyd (2003) for Portuguese, Bratveit et al. (1995) for Norwegian, and Allan et al. (2000) for Danish

	Adjective	Noun	Verb
Danish	N/A	Final 100% N = 1	N/A
Norwegian	N/A	Final 100% N = 2	Initial 100 N = 1
English	* Initial 52% N = 141	Final 75% N = 17,081	Initial 98% N = 2562
Italian	? Final 58% N = 12	Initial 90% N = 751	Initial 95% N = 44
French	? Final 78% N = 37	Initial 81% N = 1600	Initial 58% N = 167
Portuguese	? Final 79% N = 53	Initial 95% N = 1085	Initial 99% N = 139
Spanish	Final 97% N = 88	Initial 96% N = 903	Initial 100% N = 202

Figure 2: Final resuls from 10 analyses

The null results for Danish and Norwegian were due to a lack of multiword entries, and inability to split compounds. The analysis for verb phrases in French was constantly between 50-60%, and so this result has a low confidence level beyond the statistics presented.

5 Discussion

The decision to include three Germanic and four Romance languages was two fold: 1) to compare quality of procedure by using languages for which the author has some knowledge, and 2) to contrast results within language families that share properties. Unfortunately, the Danish and Norwegian data sets were extremely small (roughly 4-5% that of English) and did not contain many multi-word entries. As for other Germanic languages, the Dutch Wordnet is not publicly available, and I was unable to receive access permission to the German Wordnet by time of analysis. With regard to the Romance languages, the results seem robust across the family as represented in Wordnet form.

In general the greatest problems seemed to occur with the adjective phrases, a failure attributable to the structural properties of many Indo-European languages. As was discussed earlier, the ability of verbal particles and prepositional elements to form close-knit constructions with past participles is responsible for the incorrect results for the English adjective phrases, as well as the inconsistent results in Italian (as shown in example one), French, and Portuguese. Conversely, with the possible exception of the French data set, the confidence and accuracy of the method applied to noun and verb phrases is quite high. It would be interesting to continue this study using languages that lack an adjective-like participle form, and test if indeed the problem is language family specific, or an issue of adjectival typology or simply Wordnet entries in general.

6 Apendix: Code

```
# The following is a collection of classes to encapsultate Wordnet dictionary
    data. The classes also contain additional methods for preforming automated
    headedness analysis. If this file is run from the terminal it will
    automatically detect all Wordnet files in the same directory, and print out a
     prliminary report of headedness by Wordnet and by length of multiword
    expressions. If not run directly, i.e. imported or run in an interpreter, the
     automatic reporting as been dissabled, and a function has been added to find
     the Wordnet files in the current directory at call.

#Begin by importing required modules. NOTE- One global variable to hold Wordnet
    file names is generated dynamically either when run or called from the <
    find_files()> function, and not declared here. All other user created
    variables are encapsulated in the classes defined and generated below.
import os
import re
import random
import codecs

def find_files():
    ''' The following fuction uses the OS library to search the current
       directory, and create a list of valid Wordnet filenames.
    '''

    dir_list = os.listdir(".")
    global file_names
    file_names = []
    for name in dir_list:
        match = re.match(r'.*tab',name)
        if match:
            file_names.append(match.group())

class library:
    ''' The library class is designed to hold all processed Wordnet dictionary
       objects <dictionary()> for group execution.
    '''

    def __init__(self):
        ''' The class initializes a list of dictionary objects using the list of
             file names generated at run, or in the <find_files()> function.
        '''
        self.dictionaries = []
        for name in file_names:
            self.dictionaries.append(dictionary(name))

    def report(self):
        ''' The <report()> method itterates throught the dictionary objects
           stored, and prints the Wordnet file name, two head analyses based on
```

```
            multiword entries of all lengths and thoses of pairs alone, and
            finally a horizontal seperator for ease of reading.
        '''

        for dict in self.dictionaries:
            print dict.language
            print "All Multiplex:", dict.heads_all
            print "Pairs Only:", dict.heads_pair
            print "-" * 60

class dictionary:
    ''' The following class encapsulates a single Wordnet dictionary. It is also
        designed to contain methods for analyzing headedness of the langauge
       contained in the dictionary.
    '''

    def __init__(self, name):
        ''' The dictionary reads the Wordnet file and passes each non-comment
            line to an <entry()> object for processing. Then it randomizes and
            classifies the entries, building a dictionary of singleword entries
            and their grammatical class and list of multiword entries. It then
            searches the syntactic classes contained in the entries, and creates
            individual dictionaries to record "initial" and "final" counts for
            each, over all multiword entries and those that are only pairs.
            Finally, it searches through the multiword entries and counts the
            position of heach potential head and records it in the appropriate
            count list.
        '''

        self.language = [str(name)]
        self.f = codecs.open(name, encoding='utf-8', mode='r')
        self.entries = [ entry(line.strip().split('\t')) for line in self.f.
            readlines()[1:]]
        self.f.close()
        random.shuffle(self.entries) #Randomization is used to prevent the
            ordering of Wordnet entries dictate grammatical class.
        self.multiword_entries = []
        self.singleword_dict = {}
        self.lex_types = set()
        self.heads_pair = dict()
        self.heads_all = dict()
        self.sort_entries()
        self.countheads_pairs()
        self.countheads_all()

    def sort_entries(self):
        '''This method reads through the processed <entry()> objects uses the
            singleword entries into a dictionary of words and grammatical classes
            , and puts the multiword entries into a seperate list for easy access
```

```
        . Finally, it checks that any singleword that appears with a
        functiona class (i.e. is listed as 'r' in the offset) is marked as
        such, and thus not allowed to vary.
    '''

    for entry in self.entries:
        if entry.lemma_type == 'mwe':
            self.multiword_entries.append(entry)
        elif entry.lemma_type == 'swe':
            self.singleword_dict[entry.lemma] = entry.part_of_speech
        self.lex_types.add(entry.part_of_speech)
    for entry in self.entries:
        if entry.lemma_type == 'swe' and entry.part_of_speech == 'r':
            self.singleword_dict[entry.lemma] = entry.part_of_speech

def countheads_pairs(self):
    ''' This method first creates a dictionary to count "initial" and "final
        " entries for use by two-word multiword entries. It then sorts out
        those pair-only lemmas. In order to count the heads properly, it
        skips any word without an entry in the singleword dictionary, and any
         lemma where the initial and final word have identical parts of
        speech. If these coditions are met, it updates the approriate count
        dictionary for "I"nitial if the correct wordtype is in the initial
        word, or "F"inal if the correct word type is in the final word.
    '''

    for lex_type in self.lex_types:
        self.heads_pair[lex_type] = { 'I':0 , 'F':0 }
    mwe_pairs = [ entry for entry in self.multiword_entries if len(entry.
        lemma)==2]
    for entry in mwe_pairs:
        if entry.lemma[0] not in self.singleword_dict or entry.lemma[-1] not
            in self.singleword_dict:
            continue
        if self.singleword_dict[entry.lemma[0]] == self.singleword_dict[
            entry.lemma[-1]]:
            continue
        if self.singleword_dict[entry.lemma[0]] == entry.part_of_speech:
            self.heads_pair[entry.part_of_speech]['I']+=1
        if self.singleword_dict[entry.lemma[-1]] == entry.part_of_speech:
            self.heads_pair[entry.part_of_speech]['F']+=1

def countheads_all(self):
    ''' This method first creates a dictionary to count "initial" and "final
        " entries for use by all multiword entries. In order to count the
        heads properly, it skips any word without an entry in the singleword
        dictionary, and any lemma where the initial and final word have
        identical parts of speech. If these coditions are met, it updates the
         approriate count dictionary for "I"nitial if the correct wordtype is
```

```
            in the initial word, or "F"inal if the correct word type is in the
            final word.
        '''

        for lex_type in self.lex_types:
            self.heads_all[lex_type] = { 'I':0 , 'F':0 }
        for entry in self.multiword_entries:
            if entry.lemma[0] not in self.singleword_dict or entry.lemma[-1] not
                in self.singleword_dict:
                continue
            if self.singleword_dict[entry.lemma[0]] == self.singleword_dict[
                entry.lemma[-1]]:
                continue
            if self.singleword_dict[entry.lemma[0]] == entry.part_of_speech:
                self.heads_all[entry.part_of_speech]['I']+=1
            if self.singleword_dict[entry.lemma[-1]] == entry.part_of_speech:
                self.heads_all[entry.part_of_speech]['F']+=1


class entry:
    ''' The entry class takes a list of split strings, representing a tab-
        delineated line of text from a Wordnet file, and stores the relevent
        information in accessible instance variables. As part of the process, it
        also splits space-seperated lemmas if possible and uses the status of the
        lemmas as a unicode string or a list to determine if the entry is of the
        multiword or singleword type.
    '''

    def __init__(self, raw_entry):
        self.offset = raw_entry[0]
        self.lemma_type = ''
        if ' ' in raw_entry[2]:
            raw_entry[2] = raw_entry[2].split()
        if type(raw_entry[2]) is list:
            self.lemma_type = 'mwe'
        if type(raw_entry[2]) is unicode:
            self.lemma_type = 'swe'
        self.lemma = raw_entry[2]
        self.part_of_speech = raw_entry[0][-1]

if __name__ == "__main__":
    '''The code here checks to see if the module was run as the main program or
        imported as a module. If run as the main, then the library is built from
        Wordnet files in the current directory, and the basic analsyis is run and
        printed.
    '''

    dir_list = os.listdir(".")
    file_names = []
```

```
for name in dir_list:
    match = re.match(r'.*tab',name)
    if match:
        file_names.append(match.group())

lib = library()
lib.report()
```

References

Allan, R., Holmes, P., and Lundskar-Nielsen, T. (2000). *Danish: An essential grammar*. New York: Routeledge.

Bond, F. and Foster, R. (2013). Linking and extending an open multilingual wordnet. In *51st Annual Meeting of the Association for Computational Linguistics: ACL-2013*, Sofia.

Bradley, P. and Mackenzie, I. (2004). *Spanish: an essential grammar*. New York: Routeledge.

Bratveit, K., Jones, G., and Gade, K. (1995). *Colloquial Norwegian: A complete language course*. New York: Routeledge.

Crocker, M. (2009). *French Grammar (Shaums Outlines)*. Chicago: McGraw-Hill.

Croft, W. (2003). *Typology and Universals*. London: Cambridge University Press.

DanNet — det danske wordnet (2014). Dannet – det danske wordnet. `http://wordnet.dk/`.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Massachusetts: MIT Press.

Gonzalez-Agirre, A., Laparra, E., and Rigau, G. (2012). Multilingual central repository version 3.0: upgrading a very large lexical knowledge base. In *In Proceedings of the 6th Global WordNet Conference (GWC 2012)*, Matsue, Japan.

Hutchinson, A. and Lloyd, J. (2003). *Portuguese: An essential grammar*. New York: Routeledge.

Payne, T. (1997). *Describing Morphosyntax: A guide for field linguists*. London: Cambridge University Press.

Peyronel, S. and Higgins, I. (2006). *Basic Italian: A grammar and workbook*. New York: Routeledge.

Pianta, E., Bentivogli, L., and Girardi, C. (2002). Multiwordnet: Developing an aligned multilingual database. In *Proceedings of the First International Conference on Global WordNet*, Mysore, India.

Sagot, B. and Fišer, D. (2008). Building a free french wordnet from multilingual resources. In *Proceedings of the Sixth International Language Resources and Evaluation*, Marrakech, Morocco.

Satre, R. (2014). About wordnet in norwegian. `https://www.ntnu.no/wiki/display/pastas/About+WordNet+in+Norwegian`.

Tallerman, M. (2011). *Understanding Syntax*. New York: Hodder Education, 3rd edition.