Università
Ca'Foscari
Venezia

Corso di Dottorato di Ricerca in Scienze del Linguaggio

Scuola di Dottorato in Lingue, Culture e Società Moderne e

Scienze del Linguaggio

ciclo 30°

anno 2018

UXWN - Analysis and Improvement of a Logical Form Resource for NLP

SSD: L-LIN / 01

Coordinatore del Dottorato
Chiar.mo Prof. Enric Bou Maqueda

Supervisore
Chiar.ma Prof.ssa Marina Buzzoni

Dottorando
Agata e Rotondi
Matricola 820514

UXWN - Analysis and Improvement of a Logical Form Resource for NLP

http://www.unive.it/UXWN

Agata Rotondi

Abstract

Logical Form is an exceptionally important linguistic representation for highly demanding semantically related tasks like Question Answering. In this work I present different types of Logical Form and in particular I investigate those resources that provide a Logical Form of the WordNet Glosses. I take a closer look to one of them, eXtended WordNet, and I analyse its weaknesses and strengths. After classifying the most common errors of this resource, I semi automatically correct them and the result is a new resource: United eXtended WordNet.

Acknowledgments

My deepest appreciation goes to my advisor, Prof. Rodolfo Delmonte, for his help and guidance during these years which made it possible to achieve this work.
I would also like to thank Prof. Marina Buzzoni for her advice and kindness.
I am very grateful to all the people I met during these path, in particular to the JRC people, and to everyone who gave me suggestions, support and encouragement of any kind.
Special thanks to my family & friends, Andrea, Fiamma's girls, Traineeland, Iyengar Yoga and (last but not least) Gozeto.

List of Abbreviations

AMR	Abstract Meaning Representation
CN	Compound Noun
LHS	Left - Hand Side
LF	Logical Form
LR	Lexical Resource
NC	Nominal Compound
NLP	Natural Language Processing
NL	Natural Language
PN	Proper Name
Pos	Part of Speech
POS	Possessive (Predicate)
Q/A	Questions Answering
RHS	Right - Hand Side
UXWN	United eXtended WordNet
XWN	eXtended WordNet
WN	WordNet
WSD	Word Sense Disambiguation

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Think of a very easy question.

Let's say: *how many fingers on one hand?*
Don't matter the subject, the question is easy because you know the answer. Even for easy questions we need world knowledge to answer.
Humans, during their lifetime, acquire an incalculable amount of diverse knowledge which they store and use in different ways. The abilities of human brain are amazing but the extraordinary results achieved in Artificial Intelligence are making computers closer to us.
Everyone knows that computers are unbeatable for what concerns complex mathematical calculations but can they answer to an easy question like the one above? They don't have hands, nor fingers, and they don't know what they are unless we don't give them this knowledge. But how can we teach them the things of this word? They don't speak (yet) our language and therefore we can't communicate with them in natural language. We need to represent

the meaning of sentences in a machine readable way and there is no a unique way to do this. Many efforts have been done so far to solve the problem, and different ideas have been advanced in the last decades.

The aim of this work is to investigate one kind of representation named Logical Form and in particular I envisage investigating a resource of world knowledge where information have been automatically translated into Logical Form - eXtended WordNet. How was the resource created? Is it well formed or are there errors? Can we effectively use Logical Form for NLP tasks? Are there other resources of this kind? These are some of the questions I will tray to answer.

Before going through the main topics, I will introduce the reader to the well known machine readable dictionary WordNet from which the Logical Form resource has been generated. I will also discuss some of the WordNet improvements which have been suggested during the years. In fact, despite its popularity, WordNet has some well known limitations that can affect the automatic systems that use it.

EXtended WordNet was developed with the purpose to solve some of the WordNet shortcomings and I envisage this work to be a further step forward in the field of knowledge base and meaning representation.

The work is organised as follow:

In chapter 2 I will briefly illustrate WordNet and some of its improvements, with the intention of introducing eXtended WordNet, the resource which is at the base of this work.

Chapter 3 is devoted to an in-depth description of Moldovan's eXtended WordNet, in particular to how WordNet glosses have been transformed into Logical Forms. Here, I will show some other examples of semantic representations and I will compare eXtended WordNet to the other two resources that provide Logical Form transformation of WordNet glosses (ILF and WN30-lfs).
Chapter 4 deals with an accurate analysis of the problems that affect the Logical Forms of eXtended WordNet. This resource was automatically produced by the University of Texas at Dallas and my intention is firstly, to determine its most common errors and then, try to automatically correct them.
In Chapter 5 I will describe the automatic and manual procedures I carried out for the Logical Forms correction and some interesting considerations that came out during the work.
I will show how the efforts made for the correction led to the creation of a new resource that I named United eXtended WordNet.
The final chapter regards further improvements and related topics I didn't have the time to study in depth and which will be interesting to investigate as future work.

The most important challenge I had to face with during this work regards the correction of the Logical Forms of eXtended WordNet. The resource is large and there was several kinds of errors to correct. I had do handle a big amount of data, produce different scripts and do manual corrections. This task was really demanding and time consuming and left me little time to examine interested related topics.

Chapter 2

WordNet Extensions

2.1 Introduction

This chapter is devoted to introducing the reader to WordNet (WN) and to illustrating some improvements of this famous and widely used resource. I will start with a brief introduction to WN and after that I will organise WN improvements into two main categories: terminology extension and relations enhancement. Terminology extension, and in particular section 2.2.2, will include my participation with H. Tanev to the task 14 of Semeval 2016. It is not the purpose of this thesis to investigate the field of word vectors and I won't go into details about that. This section aims to be just an example of how to cope with the Semantic Taxonomy Enrichment Task. In the relations enhancement section I will introduce different types of relations that can be added to WN and in particular I will show how the eXtended WordNet (XWN) project succeeded in adding word sense disambiguation to WN glosses. In the last part of this chapter I will present the application of the disambiguated glosses of XWN for building

lexical chains and the Senseval Task for which they have been taken as standard.

2.2 A Brief Introduction to WordNet

WN (Miller 1995, Fellbaum1998) is one of the most popular machine readable dictionaries; its popularity comes from the richness of semantic relations which encodes and it is also due to its free public availability and its broad coverage - it includes over 200K senses of 155K word forms.

It was originally conceived as a full-scale model of human semantic organisation but its growth and later design were subsequently guided by its success in the Natural Language Processing (NLP) community.

Unlike traditional dictionaries, which ignore a synchronic organisation of the lexicon, the structure of WN is based on psycholinguistic principles.

WN is divided into four parts, one for each of the main syntactical categories: noun, verb, adjective, and adverb.

Words like *right* or *back,* which can be interpreted in different syntactical ways according to the context, are entered separately. Thus, searching the database for a word, one finds the different senses of the word in every syntactical category and the different words with which each sense can be expressed.

Lexical items in WN are not listed alphabetically, they are rather encoded in sets of synonyms called **Synsets**. Each Synset has a unique number, named **synsetID**, that identifies it. Synsets are connected to other Synsets by pointers representing semantic

relations.

The main relation among words is synonymy but for each syntactical category different semantic relations (antonymy, hyponymy, meronymy, troponymy, entailment) play a major role see Table 2.1.

Semantic Relation	Syntactic Category	Examples
Synonymy (similar)	N, V, Aj, Av	pipe, tube rise, ascend sad, unhappy rapidly, speedily
Antonymy (opposite)	Aj, Av, (N, V)	wet, dry powerful, powerless friendly, unfriendly rapidly, slowly
Hyponymy (subordinate)	N	sugar maple, maple maple, tree tree, plant
Meronymy (part)	N	brim, hat gin, martini ship, fleet
Troponomy (manner)	V	march, walk whisper, speak
Entailment	V	drive, ride divorce, marry
Note: ***N*** *= Nouns* ***Aj*** *= Adjectives* ***V*** *= Verbs* ***Av*** *= Adverbs*		

Tab 2.1 Semantic Relations in WordNet

Each word sense identifies a single Synset. For instance, given *car* (with the sense of *a motor vehicle with four wheels)* the corresponding Synset {*car*, *auto*, *automobile*, *machine*, *motorcar*} is univocally determined. For each Synset WN provides a **gloss**, i.e. a definition plus optional comments and examples. E.g. the gloss of *car* is: *a motor vehicle with four wheels; usually propelled by an internal combustion engine; "he needs a car to get to work"*.

Searching WN[1] for the previous example of the term *back*, one finds that, as predicted, it belongs to several synsets in all the four syntactic categories. See a fragment of the output here below:

N {**back**, dorsum}: the posterior part of a human (or animal) body from the neck to the end of the spine, "his back was nicely tanned"
N {**back**, backrest}: a support that you can lean against while sitting, "the back of the dental chair was adjustable"
V {**back**} travel backward, "back into the driveway"; "The car backed up and hit the tree"
ADJ {**back**, hind, hinder}: located at or near the back of an animal, "back (or hind) legs"; "the hinder part of a carcass"
ADV {**back**} travel backward, "back into the driveway"; "The car backed up and hit the tree"

In Figure 2.1 (next page) an excerpt of the WN semantic network containing the *car* Synset, taken from Navigli 2009, which shows the richness and complexity of semantic relations between Synsets.

I won't go into more detail about WN structure and I refer the reader to WN website[2] and to Fellbaum 1998 for further information.

What is important for this thesis is how widely WN has been and is currently used with considerable success for different NLP tasks.
Its semantic relations can be exploited for Word Sense Disambiguation (WSD) (see for e.g. Agirre and Soroa 2009, Banerjee and Pedersen 2002, Resnik 1995), which has a crucial role in the development of information retrieval (see for e.g. Chai

[1] You can search WN at http://wordnetweb.princeton.edu/perl/webwn

[2] https://wordnet.princeton.edu

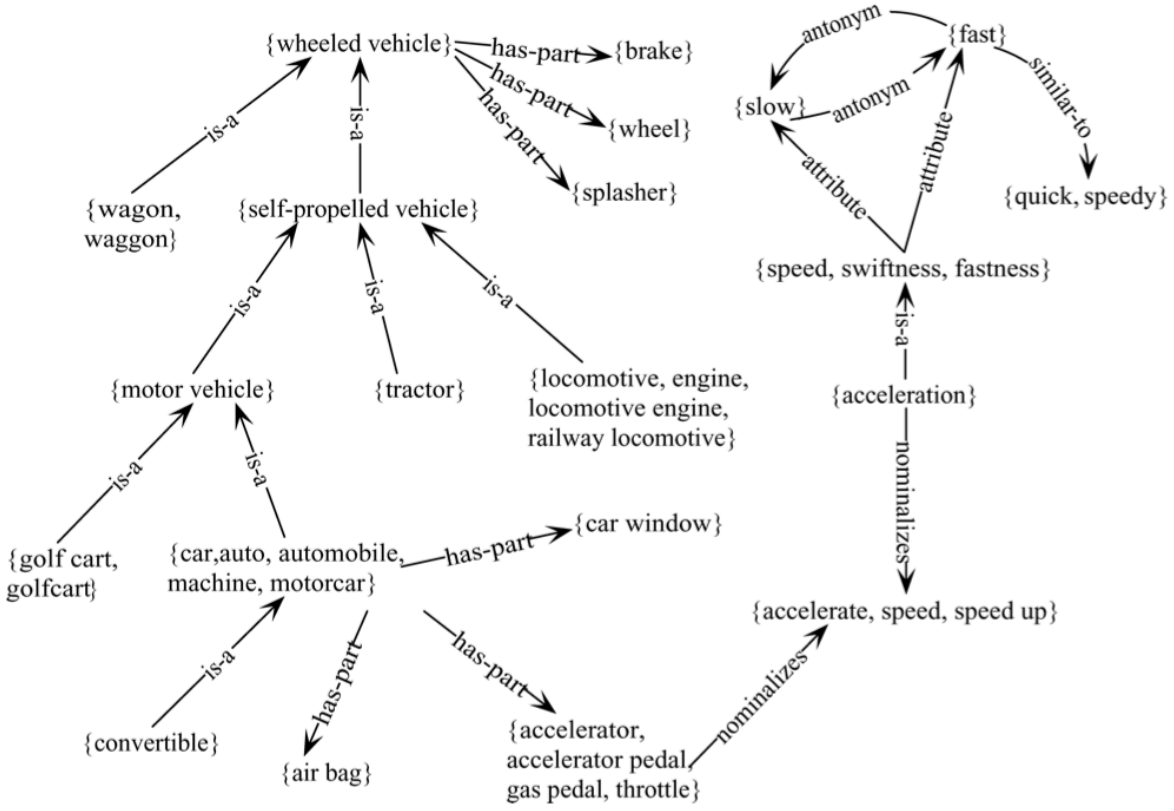


Fig 2.1 Excerpt of the WN Semantic Network

and Biermann 1997, Mandala et al. 1998, Rosso et al. 2004, Varelas et al. 2005), machine translation, summarisation, and language generation systems in addition to query expansion (Dipasree et al. 2014, Fang 2008) and cross-language applications.

Kilgariff notes that, for some NLP tasks, « *not using WN requires explanation and justification* » (Kilgariff 2000).

Another proof of its popularity is the existence of an active Global WordNet Association[3] which organises every two years a Global WordNet Conference.

Furthermore, almost 80 versions of WN have been developed for more than 50 languages (from Latin to Sanskrit) and the original

[3] http://globalwordnet.org

English WN has been mapped to several resources among which the Ontologies SUMO, OpenCyc, Dolce etc. and aligned to Wikipedia and Wikitionary (see for e.g. Miller and Gurevych 2014, Navigli and Ponzetto 2012, Niemann and Gurevych 2011, Pilehvar and Navigli 2014, Ruiz-Casado et al. 2005) . The different national versions have been linked to the original WN providing in this way a useful interlingua alignment (see the EuroWordNet[4]).
Besides its applications, WN has been investigated also in terms of its limitations and how to solve them (see for example Gangemi 2001). Beyond its coverage, also the quality of this resource is very important and, as shown by Neel and Garzon 2010, it affects the performance of the applications which employ it.

The efforts to improve WN may be divided into: **semantic relations enhancement** and **terminology extension**.

2.3 WordNet Improvements - Terminology Extension

Semantic knowledge bases of WN kind are expensive to produce and maintain. To be considered functional for NLP applications, they must include a large amount of words senses in a well structured hierarchy. Nevertheless, existing resources, despite their large coverage, have often limited scope and domains and they frequently omit lemmas and senses from specific fields, slang usages, and terminology emerged after their

[4] http://projects.illc.uva.nl/EuroWordNet/

construction.
The manual updating of WN is an expensive effort which requires a lot of time and as a result the resource is not updated frequently. There are plenty of scientific papers, which address the automatic taxonomy/ontology enrichment task and in particular the automatic enrichment of WN. See among the others Haridy et al. 2010, Navigli et al. 2004 and Nimb et al. 2013.
Existing works fall into one of the following categories:

1. Adding new senses for existing terms, e.g. Nimb et al. 2013;
2. Adding new terms, e.g. Jurgens and Pilehvar 2015.

The new terms which are added may belong to already existing terminology (Vujicic et al 2014), to a particular domain (e.g. biomedical: Poprat et al. 2008, medical: Smith and Fellbaum 2004, or architectural: Bentivogli et al. 2004), or they can belong to one well defined class like in Toral et al. 2008 who adds proper nouns to WN.
The new terms may be taken from dictionaries or extracted from a corpus. In several cases the exploited resource is Wikipedia, like Ponzetto and Navigli 2009 and Ruiz-Casado et al. 2005. The majority of the works based on Wikipedia are limited mainly to noun concepts because of its structure which mostly is organised as: noun+description. To overcome this limitation Jurgens and Pilehvar 2015 propose to extend WN with novel lemmas from Wiktionary managing to double the existing number of Synsets and attaching new ones to their appropriate hypernyms. With the excellent results achieved, they built the publicly available

resource CROWN[5].

For the WN enrichment task different resources have been exploited and different approaches have been experimented: distributional similarity techniques Snow et al. 2006, structured based approaches Ruiz-Casado et al. 2005, creation of a new ontology and its merging with the existing ones by alignment based methods (Pilehvar and Navigli 2014) or considering the attributes distribution (Reisinger and Pasca 2009).

The taxonomy enrichment task can be considered as a specific case of the ontology learning and population task, as in Buitelaar and Cimiano 2008, whose purpose is the automatic learning of semantic classes and relations.

As an example of how to expand the WN taxonomy, I will show in the next section a procedure used in a task of Semeval 2016.

2.3.1 SemEval 2016 - Task 14 - Semantic Taxonomy Enrichment

The enrichment of WN taxonomy is part of SemEval[6] 2016 and in particular the task 14[7] - Semantic Taxonomy Enrichment - « *provides an evaluation framework for automatic taxonomy enrichment techniques by measuring the placement of a new concept into an existing taxonomy: given a new word and its definition, systems were asked to attach or merge the concept into an existing WN concept* » (Jurgens and Pilehvar 2016).

---

[5] https://github.com/davidjurgens/crown

[6] SemEval is an ongoing series of evaluations of computational semantic analysis systems, it occurs annually with different tasks. SemEval 2016 website http://alt.qcri.org/semeval2016/

[7] http://alt.qcri.org/semeval2016/task14/

Already several works exist which try to automatically improve WN with new concepts, but there is no a standard evaluation framework to measure the quality of extension algorithms. The performance of existing systems can be easily measured (for e.g. removing terms from WN and verifying their reinsertion) whereas accuracy is more difficult to estimate (new terms may be very different from the ones already in WN).
Task 14 aims to evaluate systems for WN enrichment.
Words (not already stored in WN[8]) from Wiktionary together with their definitions and pos are provided to the participating systems which have to identify the Synsets to which the new terms should be merged as synonyms or attached as hyponyms. See e.g. *mudslide* and *changing_room* in Table 2.2:

Lemma	pos	Definition	Target Synset	Operation
mudslide	noun	A mixed drink consisting of vodka, Kailua and Bailey's	cocktail - a short mixed drink	ATTACH
changing_room	verb	A room, especially in a gym, designed for people to change their clothes	dressing_room - a room in which you can change clothes	MERGE

Tab 2.2 SemEval 2016 Task 14 - Examples

New terms to be added for the task are 1000, divided into training and test datasets. They belong to specific domains, slangs and neologisms and they have been manually annotated by the organizers (as gold-standard, to check the systems results).
Systems are evaluated according two criteria:

[8] Wordnet version 3.0

- the percentage of new terms added to WN
- the accuracy of the placement

Accuracy is judged with Wu&Palmer's similarity measure[9], recall is measured on the percentage of terms placed. Systems can decline to place difficult words for e.g. a gloss with many out of vocabulary words; words declined are not considered in the percentage.

2.3.1.2 Deftor at SemEval 2016 Task 14

Together with H. Tanev from the Joint Research Centre[10], I attempted the task with an algorithm which transforms each candidate definition into a term vector, where each dimension represents a term and whose value is calculated by Tf-idf[11].
We opted for a relatively simple method for searching relevant Synsets, which does not exploit any external dictionary or another semantic resource. We called our system Deftor (DEFinition vecTOR). Deftor is a system which represents the definitions (glosses) as lexical vectors and finds the most similar one for each new lemma.

---

[9] The Wu&P measure calculates similarity by considering the depths of the two concepts in the WN hierarchy and the depth of the LCS (least common subsumer)

[10] https://ec.europa.eu/jrc/en - Text and Data Mining Unit

[11] tf-idf (term frequency–inverse document frequency) is a statistical measure which increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Automatic enrichment of taxonomies and knowledge bases is very important especially for rapidly changing domain. The taxonomy enrichment task is quite challenging, mostly because of the many possibilities when attaching a new term to an existing taxonomy: first, a new word can be attached as a hyponym to different concepts, which describe it at different levels of abstraction. For example, in WN *hurricane* is a hyponym of *cyclone*, which is a hyponym of *windstorm*, which itself is a hyponym of *storm* and *storm* is a hyponym of *atmospheric phenomenon*. It is not always easy to decide where to attach a concept: in the above mentioned case the definition of *storm* and *windstorm* are not very different. In this case, it is also difficult to decide if a new concept should be merged with a similar concept from WN or it should be attached as a hyponym.
Another problem are the multiple aspects from which a concept can be perceived. For example, one can consider *hurricane* to be a *natural disaster*. It is also a *weather condition* or *cause of death*.

All these considerations unfortunately make taxonomy enrichment task quite ambiguous and difficult to tackle. In some cases, the right attachment of a new concept will be difficult also for a human expert (see the annotation process of gold standard data for the task in Jurgens and Pilehvar 2016).
Our approach to the taxonomy enrichment task represents each Synset from WN and the candidate new terms as word vectors from their definitions and then attaches each new term as a hyponym to the Synset for which the cosine similarity of its definition vector and the definition vector of the new term is the highest.

The algorithm which we propose transforms each candidate definition into a *definition vector*, a term vector, where each dimension represents a term and its weight is calculated by Tf-idf. In this way we represent each WN definition, as well as the definitions of the new terms for inclusion in WN. Moreover, we expanded each definition vector with the definitions of the words from this vector.

We then calculated the cosine similarity between each WN Synset definition and the definition of the candidate term whose place in the WN hierarchy is to be identified. Then, we attach the candidate term as a hyponym to the Synset with the most similar definition.

In order to create a definition vector for a word sense, we perform part-of-speech tagging of its gloss and we represent each gloss as a list of lemmata of its non-stop words. Words are downcased. After that, as a second step, each definition vector is being expanded with the lemmata from the glosses of its words, obtained on the first step. For example, if the WN definition for *computer* is *a machine for performing calculations automatically*, then our algorithm creates a first version of the definition vector with the non- stop lemmata *machine, perform, calculation* and the Tf-idf values of these words. Then, the algorithm takes the glosses of all the WN senses of the words in the first version of the vector.

In this particular case, we will add to the definition vector of *computer* the words from the glosses of all the senses of *machine*, *perform*, and *calculation*. Moreover, pos tags of these words are known, since we perform part of speech tagging of the glosses.

As an additional step of pre-processing we extract the genus from

the gloss - usually the first word which defines the more generic concept under which is the defined term (*a machine for …*).
We have processed all the WordNet synsets, where each Synset is represented as a definition vector. Then an inverted index was created for each definition vector, in which a word points to the definition vectors in which it appears.
For each new term t, we do the following:

1. Find the definition vector d of t.
2. For each word w from d we find via the inverted index all the synsets whose definition vectors contain w and whose part-of-speech is the same as the one of t. Let's denote the set of definition vectors of these synsets as D.
3. We find the similarity of d and each vector $d_i \in$ D. The similarity is being calculated as $d.d_i.cos(d, d_i)$, this formula was empirically derived from the training data.
4. If the part of speech of t is verb, we add to the above-calculated similarity score the similarity of the glosses of the genus of t and the genus of the Synset under consideration.
5. The Synset with highest similarity is taken and then the new term is attached as its hyponym. If the similarity of the best Synset is found to be under a certain threshold, then we do not attach the new term and we skip it.

The results of our method can be improved but are much above the baseline Random synset, which shows the feasibility of our

approach.
The main advantages of our strategy are: simplicity, independence from external resources, potentially multilingual applications.
For more details see Jurgens and Pilehvar 2016, Tanev and Rotondi 2016.

2.4 WordNet Improvements - Relations Enhancement

Despite the richness of its semantic relations, the network of WN is considered relatively sparse by several researchers (see for e.g. Moldovan and Novischi 2004, Verdezoto and Vieu 2011, Graber et al. 2006). Graber et al. 2006 subdivides this shortage into three fundamental limitations:

No cross pos links: as mentioned above, searching the WN database for a word it returns the different senses of the word in every syntactic category. This works for words which are semantically and morphologically related such as *operate*, *operator*, *operation*. Instead, semantically related words which do not share the same stem are not connected for e.g. [*traffic*, *congested*, *stop*].

No weighted relations: WN does not consider the difference of semantic distance between the members of hierarchically related pairs. For e.g. if *run* is a subordinate of *move* and *jog* a subordinate of *run*, it is intuitive that *run* and *jog* are semantically much closer than *run* and *move*.

Few relations: connections between Synsets may be increased and refined in different ways, improving exponentially the potential of this resource.

There are many projects in literature which aim to add new semantic relations or to reorganise the WN network.
Mel'cuk and Zholkovsky 1998 propose to add several different semantic and lexical relations such as *instrument* for *knife-cut* or *actor* for *book-writer*, but this kind of relation is not easy to formulate. Magnini and Cavaglia 2000 present a lexical resource where WN Synsets are linked considering topical domains, this strategy is able to connects words across pos but cannot account for the association of pairs like *Holland* and *tulip*. The goal of Graber et al. 2006 is to add quantified oriented arcs between pairs of Synset using the value of "*evocation*", i.e. how much a concept brings to mind another. This starting from the support of human annotators combined with existing similarity measures.
Other authors propose a manual (Gangemini et al. 2003, Guarino 1998) or semi-automatic (Verdezoto and Vieu 2011) reorganisation of the WN taxonomy and even the mapping between WN and other resources is often done manually, being in such way very costly.
As already said, the structure of WN is based on psycholinguistic principles and it was designed more as a dictionary than as a knowledge base (Miller 1995). The small numbers of semantic relations encoded was a authors choice with the purpose to make it generally applicable.
But in my opinion, enriching the semantic relations network is profitable and among the possible enhancements of the resource,

one is more valuable and needed than others: the semantic disambiguation of WN glosses (word sense disambiguation - WSD). We have seen that WN is widely exploited for word sense disambiguation, it is therefore essential that also words in the glosses be semantically disambiguated.

2.4.1 Word Sense Disambiguation in eXtended WordNet

The disambiguation of WN glosses has been studied in literature. Several projects aim to reach this goal (see for e.g. the WordNet Gloss Disambiguation Project[12]) and it is one of the achievements of the eXtended WordNet project (henceforth XWN). XWN is a project of the University of Texas at Dallas that, under the supervision of Prof D. Moldavan, intends to « *provides several important enhancements intended to remedy the present limitations of WordNet* »[13]. In XWN, WN glosses are syntactically parsed, transformed into Logical Forms and each content word is semantically disambiguated with high precision.
In different papers (Moldovan and Novischi 2004, Harabagiu and Moldovan 1998) Moldovan highlights the lack of connections between topically related words in WN. For e.g. there is no link between the verb *hungry* and the noun *refrigerator*:

hungry#1 - feeling a need or desire to eat ***food***
refrigerator#1 - home appliance in which ***food*** *can be*

[12] http://wordnet.princeton.edu/glosstag.shtml

[13] eXtended WordNet website: http://www.hlt.utdallas.edu/~xwn/about.html

even if the word ***food*** is common to both glosses.

In order to solve this limitation, Moldovan and Novischi 2004 built the XWN-WSD program, which has been used, together with a in-house system, to link each content word in a gloss to its corresponding WN concept, with a precision of almost 90%.

Due to the nature of glosses, XWN-WSD tool differs from ordinary systems of semantic disambiguation of open text. Glosses have a different structure compared to standard sentences, they are often grammatically incomplete and they may lack some words. Furthermore, several words appear rarely in WN glosses and consequently there is no sufficient and consistent training data to apply statistical and learning methods.

XWN-WSD tool employs a suite of methods based on heuristics. These methods are various and include: comparison of bigrams between glosses and SemCor corpus[14], common domain of the word to disambiguate and the Synset of the gloss, similarity between the words in the gloss of a word *w* and the words in the glosses of the possible senses of *w* etc. See Moldovan and Novischi 2004, pages 306-309.

Monosemous words don't need this procedure and they were directly linked to the appropriate sense. An example is the gloss of the word *abbey*: *a monastery ruled by an abbot*. The word *abbot* has only one sense in WN, it is not ambiguous and it has been tagged with the sense #1.

The results of the different procedures come to various level of

[14] Texts from the Brown Corpus semantically annotated with WordNet. Different versions are freely downloadable at: http://web.eecs.umich.edu/~mihalcea/downloads.html#semcor

precision and recall. Considering accuracy and coverage set, methods have been combined to reach the best performance.

Using these methods (system1) together with another in-house system (system2) of WSD of open text, Moldovan et Novischi succeeded, with an accuracy of 86%, in semantically disambiguating the words in WN glosses.
Furthermore, they provided the disambiguation of each content word in the glosses with a quality attribute. If system1 and system2 agreed on the disambiguation of a word, a *silver* tag was given to that word; if they disagreed the word was tagged as *normal*. A *gold* label was given to those words for which a manual intervention of a human annotator occurred (three kinds of manual annotation were applied; see detailed information about this in Litkowski 2004). With XWN-WSD, words senses have been assigned to more than 630K open class words. Only a 2.5% of these words has been tagged manually but there may be more than one gold disambiguation in a gloss (see Example 2). All the word forms corresponding to the verbs *to have* and *to be* were not disambiguated automatically.
The output of the XWN-WSD has been released in XML format. Here below the WSD for the gloss of *Sundanese* (I underlined the quality of the disambiguation and the disambiguation itself) :

Example 1 - XWN-WSD for *Sundanese* gloss:

```
<gloss pos="NOUN" synsetID="06515461">
  <synonymSet>Sundanese</synonymSet>
 <text>
   the Indonesian language spoken in the Lesser Sunda Islands
 </text>
```

```
  <wsd>
      <wf pos="DT" >the</wf>
      <wf pos="JJ" lemma="indonesian" quality="normal" wnsn="1"
     >indonesian</wf>
      <wf pos="NN" lemma="language" quality="silver" wnsn="1"
     >language</wf>
      <wf pos="VBN" lemma="speak" quality="gold" wnsn="1"
     >spoken</wf>
      <wf pos="IN" >in</wf>
      <wf pos="DT" >the</wf>
      <wf pos="NNPS" lemma="lesser_sunda_islands"
      quality="normal" wnsn="1" >lesser_sunda_islands</wf>
  </wsd>
</gloss>
```

Example 2 - XWN-WSD gold disambiguations:

```
<gloss pos="ADJ" synsetID="00128476">
  <synonymSet>subsequent</synonymSet>
 <text>
   following in time or order; "subsequent developments"
 </text>
  <wsd>
      <wf pos="VBG" lemma="follow" quality="gold" wnsn="2"
     >following</wf>
      <wf pos="IN" >in</wf>
      <wf pos="NN" lemma="time" quality="gold" wnsn="7" >time
      </wf>
      <wf pos="CC" >or</wf>
      <wf pos="NN" lemma="order" quality="gold" wnsn="4"
     >order</wf>
   </wsd> </gloss>
```

As we can see in example 1 and 2, the XML format makes the output easy to read. The element <wsd> includes the WSD output. For each term of the definition gloss, a sub element <wf> (word form) is generated (<punc> for punctuation marks). If the word is open class, the sub element <wf> contains the following attributes:

- **pos** for the part of speech

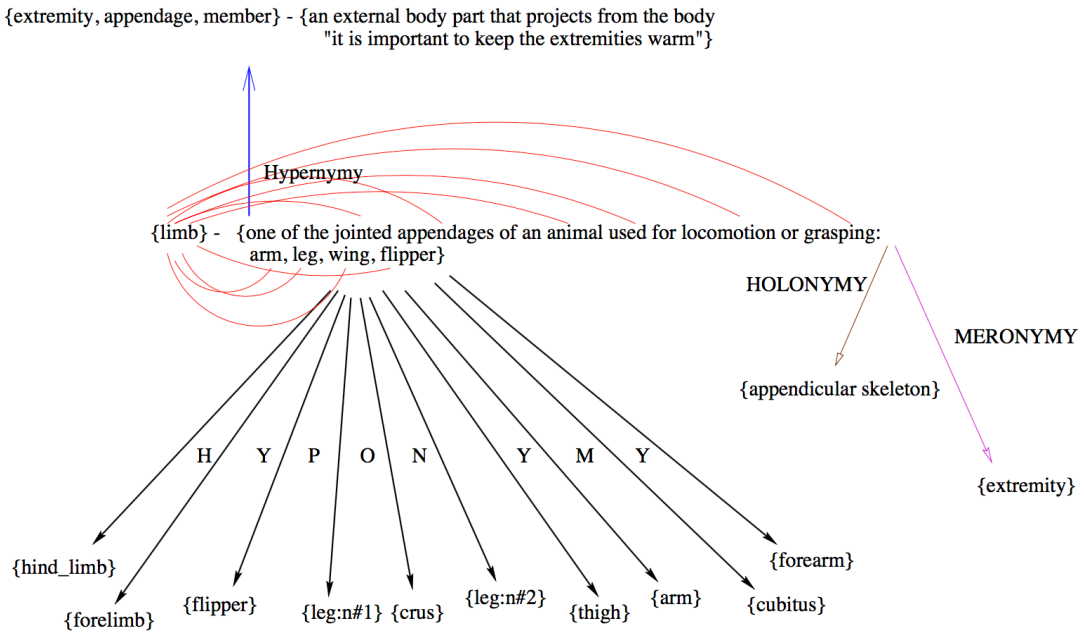


Fig 2.2 The Related Synsets of the *Limb* Synset of XWN

- **lemma** representing the stem of the word
- **quality** of the WSD
- **wnsn** representing the disambiguated sense (or senses)

otherwise, the system attributes a single feature to the <wf> tag, the pos.

As recognised by Rus in Rus 2002/1, XWN-WSD increases the connectivity among synsets by at least one order of magnitude.

As said before, each term has been linked to the Synsets of the words in its gloss. See for instance the example of the word *Limb* in Figure 2.2 where new connections are marked with red arches.

The new kind of relation which occurs between a Synset and any of the concepts in its gloss is called GLOSS(*x*,*y*); where *x* is the Synset and *y* a concept in its gloss. In the *Limb* example above we can denote this relation for e.g. as: GLOSS(*limb*, *animal*) or

GLOSS(*limb*, *grasp*) etc. These connections were not part of the semantic relations network of *limb* in WN.

For the WSD task, as for the others that we will see later (Logic Form transformation, pos tagging and parsing), examples in the glosses are removed. Therefore, the total number of open class words disambiguated (637,252) regards only glosses definitions and no examples and comments.
Table 2.3 shows the number of disambiguated open class words for each category, divided by pos (data refer to XWN2.0-1.1)[15].

Set of Glosses	Number of Glosses	Open Class Words	Monosemous Words	Gold Words	Silver Words	Normal Words
Noun Glosses	79689	505946	138274	10142	45015	296045
Verb Glosses	13508	48200	6903	2212	5193	30813
Adjective Glosses	18563	74108	14142	263	6599	50359
Adverb Glosses	3664	8998	1605	1829	385	4920
TOTAL	115424	637252	160924	14446	57192	382137

Tab 2.3 Disambiguated Open Class Words in XWN

Results show a higher percentage (27% against the average percentage of 17% for the other three pos classes) of monosemous words regarding noun glosses. In fact, as we will see later, noun definitions in WN are longer and with a richer and more specific vocabulary than those of the other pos.

[15] http://www.hlt.utdallas.edu/~xwn/wsd.html

The XWN projects achieve the disambiguation of a huge number of content words in the WN glosses and even though the manual correction of gold disambiguations is very costly and time consuming, more than 14K content words disambiguations have been manually checked.

The success of XWN WSD is proved in the next sections with its application in building lexical chains and its use as standard for a Senseval task.

2.4.2 XWN-WSD and Lexical Chains

Let's come back to the first example of the previous section, the glosses of *hungry (feeling a need or desire to eat food)* and *refrigerator* (*home appliance in which food can be stored at low temperature*). WSD of the glosses can now be exploited to build lexical chains and explain cohesion and intention of a simple text like:

S1) *Jim was hungry*
S2) *He opened the refrigerator*

Lexical chains (Hrist 1995) are semantically related words that link two concepts. They are built on resources that contain concepts and their relations.
In this case the word *food* is the key concept; thanks to XWN WSD a lexical chain between *hungry* and *refrigerator* can be constructed, explaining the connection between the two sentences.

A better lexical chain framework can be developed from the disambiguated glosses rather than relying only on the WN relations among synsets.
This idea has been investigated and implemented in the work of Prof Moldovan, in particular in Moldovan and Novischi 2002 lexical chains are tested in a Q/A task.

Another work for which a mention is needed but that I won't treat here, is the further effort to improve semantic connectivity of XWN concepts made by Erekhinskaya and Modovan 2013. Starting from the fact that GLOSS relation doesn't provide weighted connections[16], they transformed the glosses into semantic graph using the semantic parser Polaris (Moldovan and Blanco 2012), and replacing in this way the GLOSS relation with lexical chains.

2.4.3 Senseval-3: WSD of WN Glosses Task

XWN-WSD is not the only project which aims to disambiguate the WN glosses but it is, indeed, one of the most significant.
As seen earlier, in XWN WN glosses have been disambiguated combining human annotation and automated methods. The result is an excellent source of data which has been used as standard for

[16] For e.g. the gloss of the concept *notation* is "*a technical system of symbols used to represent special things*". It is obvious that GLOSS(*notation*,*system*) is stronger than GLOSS(*notation*,*special*).

one of the tasks of Senseval-3[17,18]: Word Sense Disambiguation of WN Glosses.

This task is analogous to other WSD tasks of texts of previous Senseval editions. What makes it different are the peculiarities of WN glosses: they often do not constitute standard sentences and might be incomplete (see page 29).

Participants have to face these characteristics to complete the task and to do so they are allowed to exploit other data provided by XWN: pos tags for each word in the glosses, parses and logical forms.

All the glosses from XWN in which at least one disambiguation was tagged as *gold* (more than 9k) constitute the test set provided to participants (no training data available) which have to replicate the hand-tagged results. The glosses were provided in XML format, exactly as they appear in XML file, structured as: synsetID + POS + gloss:

```
<gloss pos="ADJ" synsetID="00128476">
  <synonymSet>subsequent</synonymSet>
 <text>
   following in time or order; "subsequent developments"
 </text>
</gloss>
```

Seven teams participated to the task. They could investigate the disambiguations already available in XWN and then they had to develop their own systems.

After the tokenisation of the glosses, systems were asked to replicate the hand-tagged results. The expected output for each

[17] Senseval is the precursor of SemEval, it can be defined as a set of evaluation exercises for the semantic analysis of text. See Senseval website: http://www.senseval.org

[18] Senseval Task 3: WSD of WN glosses webpage: http://www.clres.com/SensWNDisamb.html

gloss was: pos of the gloss + its Synset number + a WN sense number to identify each content word in the gloss.
Due to the absence of significant context, participants employed WN semantic relations for the disambiguation of the glosses.
Results were evaluated using precision and recall and are reported in Litkowski 2004. They reveal good performances of the participating systems but lower than the XWN WSD gold standards.
The task was designed to encourage « *development of technology to make use of standard lexical resources* » (like WN and XWN) and to motivate « *the lexical resource community to take up the challenge of disambiguating dictionary definitions* » (Litkowski 2004).
Motivations and results of the task lead to the conclusions that WSD of WN glosses is an important improvement as well as a challenging task and that XWN is one of the projects with better results so far.

That said, I have to mention the fact that during the evaluation of the task's results it came out that gold disambiguation quality of XWN is not always synonym of correctness. It seems that sometimes human annotators did not consider WN semantic relations while disambiguating a word but they rather (probably) relied only on their personal judgment.
For instance, WN definitions have often the structure: genus+differentiae. Since most of WN Sysnsets have a hypernym, it should be easy to disambiguate the genus of this definitions by looking at the hypernym of a Synset's definition. This doesn't

alway happen and some gold disambiguations are incorrect. For e.g. the word *failure* has different senses in WN[19], see Fig 2.3:

- S: (n) **failure** (an act that fails) *"his failure to pass the test"*
- S: (n) **failure** (an event that does not accomplish its intended purpose) *"the surprise party was a complete failure"*
- S: (n) **failure** (lack of success) *"he felt that his entire life had been a failure"; "that year there was a crop failure"*
- S: (n) **failure**, loser, nonstarter, unsuccessful person (a person with a record of failing; someone who loses consistently)
- S: (n) **failure** (an unexpected omission) *"he resented my failure to return his call"; "the mechanic's failure to check the brakes"*
- S: (n) bankruptcy, **failure** (inability to discharge all your debts as they come due) *"the company had to declare bankruptcy"; "fraudulent loans led to the failure of many banks"*
- S: (n) **failure** (loss of ability to function normally) *"kidney failure"*

Fig 2.3 Different WN Senses of the Word *Failure*

While annotating the sense of *failure* in the definition of *naught* : *complete failure*, a human annotator (as well as an automatic system) might use the WN hypernym relation that connects the Synset of *naught* to the *failure* one:
therefore, *failure* in the definition of *naught* should be tagged with

naught = complete failure
↓ has direct hypernym
failure = an act that fails

its first sense in WN (*an act that fails*) while it is tagged in XWN with the second WN sense (*an event that does not accomplish its intended purpose*) revealing in this way no consideration of WN semantic relations by the human annotator.

[19] The output is taken from : http://wordnetweb.princeton.edu/perl/webwn

This fact reveals the importance of an accurate human annotation which is often needed in the creation and maintenance of resources for NLP. Manually annotating does still often mean better accuracy but this is true only if the annotation is meticulous and well organised.
I will show later that in other parts of the XWN resource, specifically in the Logical Forms, manually transformed definitions are not free of mistakes.

2.5 Conclusions

WN is an important resource for the NLP community. Despite its success, it has been proved that it can be improved in different ways, from the expansion of its terminology to the enrichment of its semantic relations.
Several projects aim to create an enhanced version of WN, and one in particular succeeded in this intention: XWN.
We have seen in this chapter how in XWN the content words of WN glosses have been disambiguated. From the analysis of the disambiguation results, some first differences came out between the definitions and dimensions of each pos file: in WN, and therefore in XWN, definitions of nouns are longer and more complex and they constitute more than half of the whole resource. Furthermore, WN definitions can't be considered and treated as normal text, their structure is different and might be sometimes incomplete. Manually checked results are usually accurate but the human annotation is not free of errors.

XWN WSD is an important achievement but is just a part of the whole project.

In the next chapter I will take a closer look to XWN ad in particular to another enhancement made with this project: the Logic Form transformation of WN glosses.

Chapter 3

eXtended WordNet and Logical Forms

3.1 Introduction

In this chapter I will describe the XWN resource, its structure and in particular how WN glosses have been transformed into Logical Forms (henceforth LF). I will then take a closer look to the LF as semantic representation and to several works about it. I will start by reviewing some examples from STEP 2008, providing in this way a brief overview on how to include more semantics in this kind of representation. I will then describe and comment on two resources that, like XWN, provide LF transformation for WN glosses: WN30-lfs and ILF. In the interest of proving LF, and in particular XWN LF, as an interesting topic of research and a valid support for NLP tasks, I will report the Senseval3 task on LF transformation and an example of its application in a Q/A system. Finally, conclusions will introduce the work of correction of XWN LFs.

3.2 eXtended WordNet

In the previous chapter I roughly introduced the eXtended WordNet (XWN) project and in particular I reported the efforts of UTD people in disambiguating the glosses of WN (XWN-WSD). This is just a little part of the whole project which we see in detail here below.

The XWN resource is apparently less famous than WN but as highlighted in different papers it is also very useful and appreciated in the NLP community. Litkowski 2004 affirms that « *The eXtended WordNet is used as Core Knowledge Base for applications such as Question Answering, Information Retrieval, Information Extraction, Summarization, Natural Language Generation, Inferences and other knowledge intensive applications* »

With the XWN project, the purpose of Prof Moldovan and his group of researchers is to semantically and morphologically enhance WN.

They envision an improvement of the rich information contained in WN glosses with the intent to «*increase the connectivity between synsets and provide computer access to a broader context for each concept*»[20].

WN was not designed to serve as lexical resource and, despite its success, it exhibits some well-known limitations when used for knowledge processing applications.

[20] http://www.hlt.utdallas.edu/~xwn/about.html

To reach their goal of enhancement, the UTD researchers built a tool which takes as input WN[21] and automatically generates an improved version named eXtending WordNet. The tool framework is organised as follow:

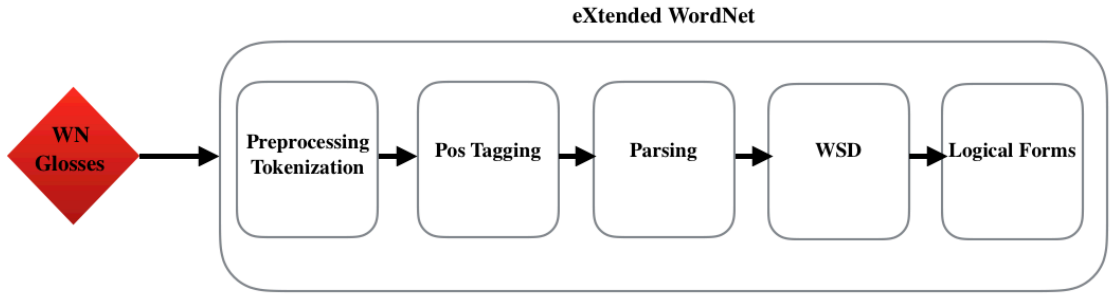


Fig. 3.1 eXtended WordNet Framework

- preprocessing and pos tagging of the glosses (it includes tokenisation and compound words detection)
- parsing
- WSD of content words in the glosses
- logic form transformation of WN glosses

Let's take a closer look to the different steps.
For the WSD part we refer the reader to the previous section 2.3.1

3.2.1 Preprocessing and Pos Tagging

The first stage of the process is the preprocessing and pos tagging of the glosses. It includes tokenisation, identification of compound words and exclusion of examples. The accuracy achieved in this step is crucial since it affects the rest of the process.

[21] The tool is designed to take as input current and future versions of WN

In the preprocessing stage, the glosses are parsed in order to exclude contents between parentheses and examples. For e.g. one of the senses of the word *blind* has the gloss: *a hiding place sometimes used by hunters (especially duck hunters); "he waited impatiently in the blind"*. Only the definition (*a hiding place sometimes used by hunters*) is processed in the next steps.

Definitions are then tokenized with an in-house system which complies with the Treebank tokenisation requirements. The tokenizer includes specific glosses-extensions and identifies a definite set of collocations (*up and down*, *to and from*, *in order to* etc.).

In order to obtain an accuracy of almost 100%, pos tagging has been carried out combining the Brill tagger[22] and the Mxpost tagger[23]; when the two taggers disagree, the agreement for word to tag is sought in WN. Eventually, a human check occurs if needed. For e.g. the outputs of the taggers for the gloss of *abbey#3* is:

```
Brill's: a/DT monastery/NN ruled/VBN by/IN an/DT abbot/NN
MXOST: a_DT monastery_NN ruled_VBN by_IN an_DT abbot_JJ
```

---

[22] Before applying the rules, Brill's tagger gives to each word the most likely tag, estimated by examining a large tagged corpus. If a word is unknown (not in the tagged corpus) it is considered proper noun if capitalised, otherwise it is tagged according to the most common tag for words ending in the same three letters. See 67

[23] Mxpost tagger uses a rich feature representation and for each word it generates a tag probability distribution reaching an accuracy of 96%. See 69, 66

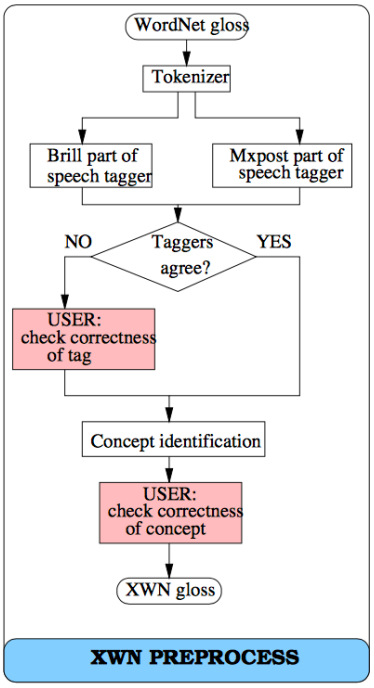


Fig 3.2 XWN Preprocessing Phase

In this case, the taggers disagree on the last word *abbot*. The system checks the pos of *abbot* in WN and assigns the right tag NN to the word.

The choice of the taggers was done after evaluating the combination of different taggers, with refer to the work of Mihalcea and Bunescu 2000. At the start of the project, three taggers were chosen considering public availability, accuracy and the set of tags used: the Mxpost tagger (a tagger based on the minimum entropy principle)[69], the Brill tagger (a rule based pos tagger, see Brill 1992), and the Qtag tagger (a probabilistic tagger, see Mason 1997). After evaluating the accuracy resulting from different combinations, the decision of using Brill + Mxpost was made also considering the agreement sets.

Results reported in Rus 2002/1 show an achieved pos accuracy of 98,93% resulting from an accuracy of 98,50% for 91,57% of the words, 100% accuracy for 0,94% of the words and a human

annotation of 7,52% of the words.
Compound words were also identified semi automatically with the intervention of a human when needed. Manual revision may be very time consuming but, as reported in Mihalcea and Moldovan 2001, in this first stage took only few hours.
The functions of the preprocessing and parsing stage are combined in the *xwnPreprocess tool* and summarised in Figure 3.2 from Mihalcea and Moldovan 2001 (see previous page).

3.2.2 Parsing

To improve the parsing accuracy, an extension was first applied to the glosses. Depending on the pos of the Synset, glosses were extended in different ways, see the following examples:

- Nouns: noun + *is* + gloss + period
  `eggshake is a milkshake with egg in it.`
- Verbs: *to* + verb + *is to* + gloss + period
  `to cut is to make an incision or separation.`
- Adjectives: adjective + *is something* + gloss + period
  `marine is something native to or inhabiting the sea.`
- Adverbs: adverb + is + gloss + period
  `syntactically is with respect to syntax.`

Two parsers were used to complete the task: the Charniak's parser and an in-house parser. The last one is a bottom-up chart parser for which the main source of errors comes from its tendency to structure inputs as sentences. This mainly happens for glosses containing relative clauses. The extensions of the glosses

compensate this tendency improving the accuracy from 62,67% to 85,25% (the parser was tested on 400 manually tagged glosses). The combination with the in-house parser and the Charniak's parser, plus a human intervention, led to an accuracy of 98,93% (see Rus 2002/1). Here below the output of the parse tree for the gloss of *eggshake*:

```
<gloss pos="NOUN" synsetID="07446232">
  <synonymSet>eggshake</synonymSet>
 <text>
   a milkshake with egg in it
 </text>
  <wsd>
       </wsd>
<parse quality="SILVER">
(TOP (S (NP (NN eggshake) )
        (VP (VBZ is)
            (NP (NP (DT a) (NN milkshake) )
          (PP (IN with)
                    (NP (NP (NN egg) )
                        (PP (IN in)
                            (NP (PRP it) ) ) ) ) ) ) )
</parse> <\gloss>
```

Also in this case, results have been classified into three quality categories: *Gold* for the manually checked parsered glosses; *Silver* for those parsered glosses for which the two parsers agreed but no human verification occurred; *Normal* was attributed to the rest of the glosses for which there was no agreement of the two parsers and no human verification. Table 3.1[24] illustrates the results arranged by pos and quality. Despite the high accuracy achieved by automatic parsers, the human annotation improves the final results. Data which have been manually checked are found to be more precise. Therefore, the complete absence of *gold*

[24] http://www.hlt.utdallas.edu/~xwn/parsing.html

glosses in the noun file reveals a lesser precision for the final parsered glosses of this particular section.

The high number of *normal glosses* highlighted in Table 3.1 is another important sign of this feature.

POS	Total Glosses	Gold Glosses	Silver Glosses	Normal Glosses
Noun	87,777	0	38,087	49,690
Verb	13,560	13,560	0	0
Adjective	20,229	14,334	5,895	0
Adverb	3,922	1,058	2,864	0

Tab 3.1 XWN Parsing Results

As one can see, the noun category results are quite different from those of other categories; they are the only ones with zero gold - manually checked - glosses and with a consistent number of normal glosses (i.e. the less accurate). I think that this is due mainly to the fact that the noun glosses are more complex than other pos categories. Definitions of nouns in WN are longer and more complex than those of other pos categories. Results of automatic systems reflect these characteristics.
I will come back on these differences in the next chapters.

3.2.3 Some Considerations

More than once I mentioned the fact that WN definitions have different lengths according to the pos file they belong to, but how different are they?

These considerations might be useful both for those who is interested in working on WN and for those who deal with XWN.
In order to calculate the average length of definitions for each pos file I built systemA in appendix1. For each XWN pos file SystemA gathers all the definitions and saves them in a txt file, one per line, taking care of deleting examples, content between parenthesis and recording separately multiple definitions. For e.g. the definition of *tease*: *the act of harassing someone playfully or maliciously (especially by ridicule); provoking someone with persistent annoyances; "he ignored their teases"; "his ribbing was gentle but persistent"* is recorder as two separated definitions:

the act of harassing someone playfully or maliciously
provoking someone with persistent annoyances

When all the definitions have been cleaned and properly recorded, SystemA calculates their average length. Results are shown in Table 3.2:

Pos File	Definitions Average Length
Noun	53
Verb	30
Adjective	37
Adverb	23

Tab 3.2 Average Lengths of Definitions

As mentioned, definitions of nouns are much longer than those of other pos, while definitions of adverbs are the shortest.

A second consideration regards the procedures and results of XWN described so far. The purpose of the first section of this chapter was to investigate the accuracy sought by the XWN researchers in order to better understand the results of the LF transformation of the glosses and its errors.
I will show that the research and achievement of a good level of correctness in the first phases of XWN (pre-processing, pos tagging and parsing) are not enough to guarantee the impeccability of the LF transformation of the WN definitions.

3.2.4 Logical Form Transformation

Starting from the output of the syntactic parser, transformations and heuristics were applied to create LF of the glosses. LF is an intermediate semantic representation which stands between the syntactic parser and the deep semantic form.
To generate LFs, grammar rules were extracted automatically from the parse tree and for each of them one or more transformation rules were manually developed. Since the number of grammar rules was large (more than 5K for nouns WN glosses, more than 10K for all the pos), a set of most common grammar rules were derived from a representative corpus of glosses. The performance was then adjusted by selecting other valuable rules and by considering some syntactic and lexical information. The most common grammar rules were enough to cover most of the occurrences; this was possible especially for the nature of the definitions which are (most of the times) of the form: *genus* + *differentia*.

Before applying the transformation rules, the output of the parser was simplified in order to facilitate the LF transformation process: determiners, modals and auxiliaries were eliminated, plurals and negations ignored (NNS -› NN), proper nouns treated as common nouns (NNP -› NN), verb tenses ignored (but passive information kept) (VBG, VBP, VBZ, VBN -› VB) etc.
Furthermore, some of the most complex structures were simplified (see for instance Figure 3.3).

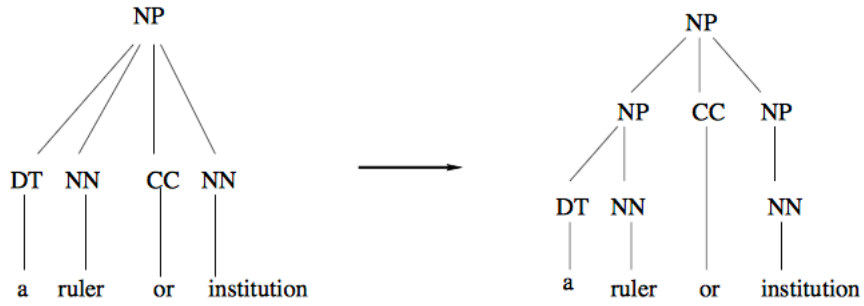


Fig 3.3 Example of Structure Simplification

Transformation rules were of two types:

- *intra-phrase*: produce predicates for every noun, verb, adjective or adverb and assign them the first variables. E.g.:

Phrase	Grammar Rule	Transformation Rule
(NP(a/DT short/JJ sleep/ NN))	NP -> DT JJ NN	*adj/JJ noun/NN -> noun(x1) & adj(x1)*

- *inter-phrase*: assign arguments to verb predicates, proposition predicates and conjunctions. E.g.:

Phrase	Grammar Rule	Transformation Rule
(PP (by/IN (NP an abbot))	PP -> IN NP	*prep/IN noun/NP(x) -> prep(-, x) & noun(x)*

The strategy was *bottom-up*: rules were applied starting from the leaves of the tree and then climbing to the top.
In the first stage of the process nouns and verbs were identified and an unique argument was assigned to each of them. The argument was of the type *x* for nouns and *e* for verbs.
During this phase, complex nominals were identified and recognised as single nominal. This was done with the *nn* predicate (first introduced in the TACITUS project, see Hobbs 1986) which can have a variable number of arguments, with the first one representing the result of the aggregation of the nouns corresponding to the rest of the arguments. Examples of LF with *nn* predicates:

```
NN(x1, x2, x3) animal:NN(x2) life:NN(x3) in:IN(x1, x4)
particular:JJ(x4) region:NN(x4)

local:JJ(x1) NN(x1, x2, x3) church:NN(x2) community:NN(x3)

club:NN(x1) for:IN(x1, x2) player:NN(x2) of:IN(x2, x3) NN(x3,
x4, x5) racket:NN(x4) sport:NN(x5)
```

Modifiers, adjectives and adverbs take the argument of the head of the phrase in which they are included. For *nn* predicates the argument is the first one:

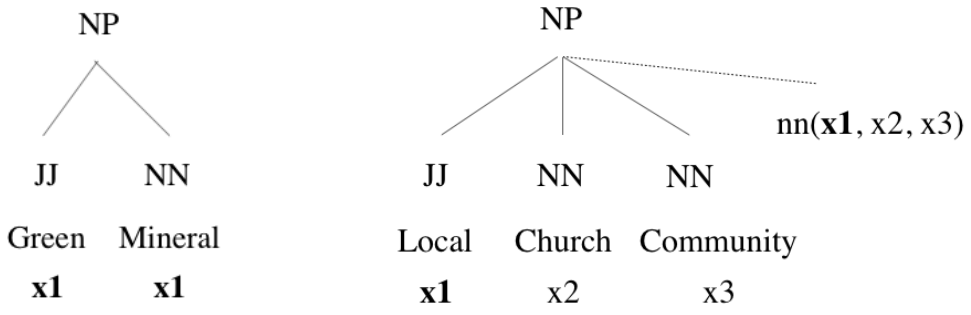


Fig 3.4 *nn*-Predicate

Predicates were then propagated to upper levels of the tree, trying to connect arguments from one predicate to another. Arguments which still remain unassigned need the intervention of heuristics. For example, the *subject* of a verb is identified as the phrase head argument preceding the verb or the prepositional object argument of the following *by* preposition (if the verb is in passive voice); the *prepositional argument* is identified in the following phrase head argument etc. (see all the heuristics in Rus 2002/1). It is important to remember for this work that when the heuristics failed, and an argument slot remained empty, a new argument was generated.

In Figure 3.5 (next page) a representation of the process.

A predicate is generate for every noun, verb, adjective or adverb in the gloss. Following the Davidsonian treatment of action predicates (Davidson 1967), events are reified and each verb is transformed in a three arguments predicates (e1,x1,x2) where: *e1* represents the action, state or event stated by the verb, *x1* the syntactic subject and *x2* the syntactic direct object. E.g.:

```
writer:NN(x1) compose:VB(e1, x1, x2) rhyme:NN(x2)

monastery:NN(x1) rule:VB(e1, x2, x1) abbot:NN(x2)
```

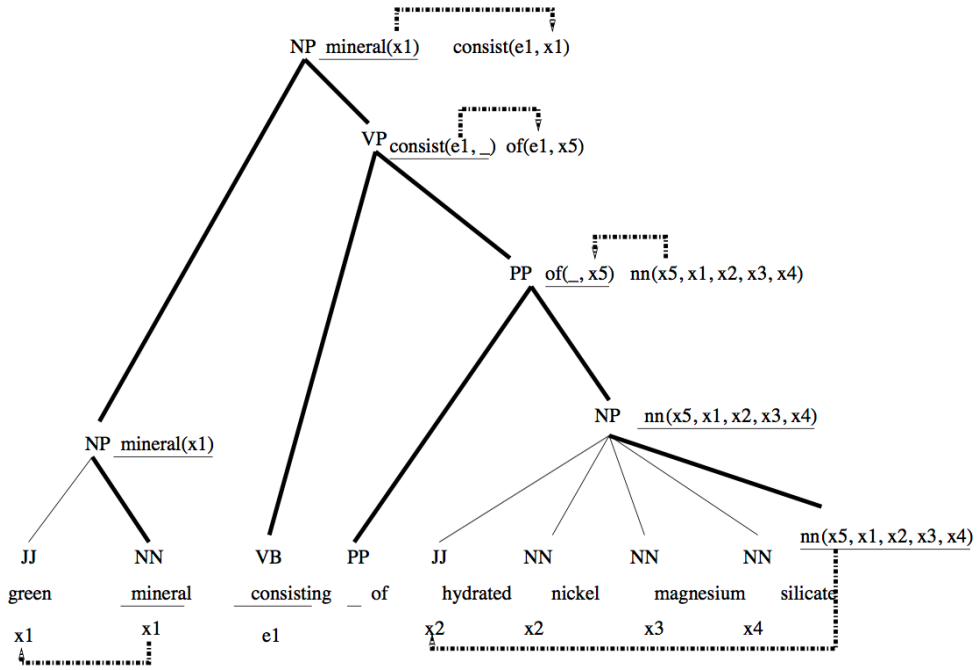


Fig 3.5 Logical Form Transformation Process

In case of ditransitive verbs a fourth argument is added. E.g.:

```
professor:NN(x1) give:VB(e1, x1, x2, x3) grade:NN(x2)
                                    student:NN(x3)
```

The arguments are in the fixed order: subject, direct object, indirect object. While the first arguments are always present in the verb predicate, the fourth argument is added only when necessary. If one of the syntactic roles is missing in the gloss, the argument can be dummy, i.e. it is assigned to the verb predicate but it is not associated to anything else in the gloss. This is the case of intransitive verbs, e.g.:

```
someone:NN(x1) arrive:VB(e1, x1, x2) late:RB(e1)
```

As shown by *late:RB(e1)* in the previous example, predicates generated from modifiers share the same arguments with the predicates corresponding to the phrase heads. Adjectives share the

arguments with the nouns they modify and Adverbs share the arguments with the verb/adjective they refer to. See for e.g. the gloss of the word *ballroom*:

```
large:JJ(x1) room:NN(x1) use:VB(e1, x3, x1) mainly:RB(e1)
              for:IN(x1, x2) dancing:NN(x2)
```

Conjunctions and Prepositions are also transformed into predicates. While conjunctions have a variable number of arguments, of which the first one represents the result of the aggregation, prepositions have two fixed arguments; the first argument corresponds to the predicate of the head of the phrase to which prepositional phrase is attached, whereas the second argument corresponds to the prepositional object. E.g.:

```
apprehend:VB(e2, x1, x2) and:CC(e1, e2, e3) reproduce:VB(e3, x1,
                    x2) accurately:RB(e1)

     bed:NN(x1) on:IN(x1, x4) ship:NN(x2) or:CC(x4, x2, x3)
                          train:NN(x3)

  expose:VB(e2, x1, x2) to:IN(e2, x7) ray:NN(x7) of:IN(x7, x3)
sun:NN(x3) or:CC(e1, e2, e3) affect:VB(e3, x1, x2) by:IN(e3, x4)
                exposure:NN(x4) to:IN(x4, x3)
```

As already said, XWN is divided in four files, one for each pos. LFs are structured in different manners, one for each of these files.

- Noun: the argument *x1* is assigned to the first word representing the Synset and in the gloss it refers to the same entity. E.g.:

```
frappe:NN(x1)      ->      thick:JJ(x1)      milkshake:NN(x1)
contain:VB(e1,x1, x2) ice_cream:NN(x2)
```

- Verb: the argument *e1* is assigned to the first word representing the Synset, its subject is the argument *x1* and its object the argument *x2*.[25] E.g.:

```
prologize:VB(e1, x1, x2) -> write:VB(e2, x1, x3) or:CC(e1, e2,
          e3) speak:VB(e3, x1, x3) prologue:NN(x3)
```

- Adjective: as for nouns, the argument x1 is assigned to the first word representing the Synset and in the gloss it refers to the same entity. E.g.:

```
ascetic:JJ(x1) -> practice:VB(e1, x1, x2) great:JJ(x2) self-
                        denial:NN(x2)
```

- Adverbs: the argument *e1* is assigned to the first word representing the Synset and in the gloss it refers to the same action or modification of the same action. E.g.:

```
syntactically:RB(e1) -> with:IN(e1, x1) respect:NN(x1) to:IN(x1,
                        x2) syntax:NN(x2)
```

A quality attribute was assigned also to the LF. They were tagged as *gold* when a human checked the output; when there was no human supervision, they were tagged as *silver* if the parse trees obtained from the two different parsers agreed, *normal* if they did not. Thus, as the labels themselves suggest, in terms of correctness the LFs can be arranged as:

[25] l'unico VB in LHS con 4 arguments è give:VB(e1, x1, x2, x3) -> allow:VB(e1, x1, x3) to:IN(e1, e4) have:VB(e2, x3, x2) or:CC(e4, e2, e3) take:VB(e3, x3, x2)

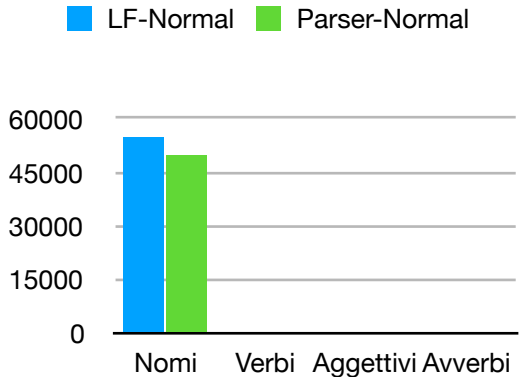
gold > silver > normal

Tab 3.3 gives an overview of the quality of LFs for each pos:

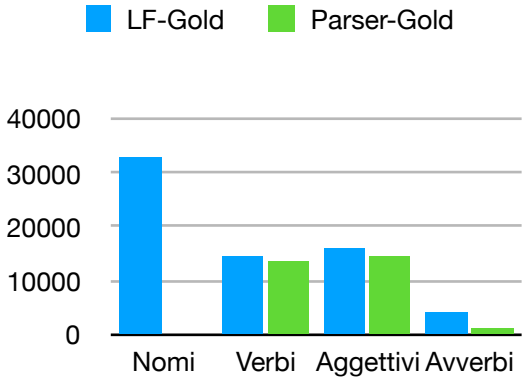
POS	Total LF	Gold LF	Silver LF	Normal LF
Noun	94868	32844	7228	54796
Verb	14441	14441	0	0
Adjective	20380	16059	4321	0
Adverb	3994	3994	0	0
TOTAL	133683	67338	11549	54796

Tab 3.3 Quality of LFs

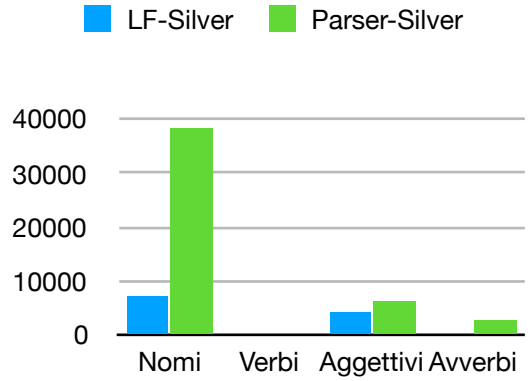
Comparing the quality of the LFs with that of the syntactic parser (see page 39), we can see that a high number of normal parsing results with a similar high number of normal LFs in the noun file (graph 1), which is more than 50% of the whole file (just like the *normal* parsered glosses). The gold qualities are not connected since they derive from a manual annotation more than from an automatic process (graph 2). Despite the high number of gold LFs in the noun file the average quality of this file is lower than the other ones, so it is likely that it is more subject to errors. Verb and adverb LFs have all been checked manually.
The graphs and tab 3.3 give also an overview of the size of the whole resource in terms of LFs: the noun file is the largest and at the same time the less accurate. More than 133K definitions have been translated into LF in XWN. It's a huge amount of data I will investigate in the following chapters.



Graph 1



Graph 2



Graph 3

Results have been evaluated by XWN people by comparing the automatic derived LFs with a set of 1Kmanually transformed LFs. The measure used for the evaluation was *exactLFaccurancy: number of correct LF over number of attempted LF transformations*. The accuracy achieved was almost **90%**. See Rus 2002/1 and 2.

3.2.5 XWN Format

In order to provide a flexible and scalable resource, the final output of XWN was released in XML format. This choice allows

for future incorporation of new information without modifying the existing structure.

Here below the final XWN output for the verb *breathe*.

The first two attributes of the root element *xwn* identify the version of XWN (ver) and the version of WN from which come the glosses (wnver); xmlns provides the link to the XWN project website.

For each Synset in WN, the element *gloss* was generated. It has two attributes: *pos* for the pos of the Synset (a verb here) and *synsetID* for the WN ID of the synset. Gloss has five child elements: *synonymSet*, *text*, *wsd*, *parse*, and *lft*. They contain respectively: the words of the Synset (they can be one or n)[26], the text of the gloss (examples will be removed later in the process), wsd of the content words in the gloss, the syntactic parse tree and the LF.

```
<xwn ver="2.0-1" wnver="2.0" xmlns="http://
xwn.hlt.utdallas.edu">

<gloss pos="VERB" synsetID="00001740">
    <synonymSet>breathe, take_a_breath, respire, suspire
    </synonymSet>
    <text>
    draw air into, and expel out of, the lungs; "I can breathe
better when the air is clean"; "The patient is respiring"
    </text>
    <wsd>
     <wf pos="VB" lemma="draw" quality="gold" wnsn="11" >draw
     </wf>
     <wf pos="NN" lemma="air" quality="gold" wnsn="1" >air</wf>
     <wf pos="IN" >into</wf>
     <punc>,</punc>
```

[26] I calculated the max number of words in the synsets for each XWN pos file. See Appendix 2 for the code and results. The longest Synset is in the noun file consists of 28 words: {buttocks, nates, arse, butt, backside, bum, buns, can, fundament, hindquarters, hind_end, keister, posterior, prat, rear, rear_end, rump, stern, seat, tail, tail_end, tooshie, tush, bottom, behind, derriere, fanny, ass}

```
      <wf pos="CC" >and</wf>
      <wf pos="VB" lemma="expel" quality="gold" wnsn="5" >expel
      </wf>
      <wf pos="IN" >out</wf>
      <wf pos="IN" >of</wf>
      <punc>,</punc>
      <wf pos="DT" >the</wf>
      <wf pos="NNS" lemma="lung" quality="gold" wnsn="1" >lungs
      </wf>
    </wsd>
    <parse quality="GOLD">
    (TOP (S (S (VP (TO to)
               (VP (VB breathe) ) ) )
        (VP (VBZ is)
            (S (VP (TO to)
                   (VP (VP (VB draw)
                           (NP (NN air) )
                           (PP (IN into) ) )
                       (, ,) (CC and)
                       (VP (VB expel)
                           (PP (IN out)
                               (PP (IN of) (, ,)
                                   (NP (DT the) (NNS  lungs) ) )
    ) ) ) ) ) ) )
        (. .) ) )
    </parse>
    <lft quality="GOLD">
    breathe:VB(e1, x1, x2) -> draw:VB(e2, x1, x3) air:NN(x3)
    into:IN(e2, x4) and:CC(e1, e2, e3) expel:VB(e3, x1, x3)
out_of:IN(e3, x4) lung:NN(x4)
    </lft>
</gloss>
. . .
</xwn>
```

3.3 Logical Forms

The logical forms of XWN are a kind of semantic representation which stands between the syntactic parse and the deep semantic form.

Following Hobbs' guidelines and his successful representation used in TACITUS (Hobbs 1986 and 1985) , XWN LF are thought

to be as close as possible to natural language and syntactically simple. Hobbs affirms that «*for many linguistic application it is acceptable to relax ontological scruples, intricate syntactic explanations, and the desire for efficient deductions in favour of a simpler notation closer to English*» (in Moldovan and Rus 2001).
XWN LFs share with Hobbs' representation also the Davidsonian treatment of action sentences (Davidson 1967) in which events are treated as individuals.
Hobbs et al. 1993 postulated that LFs are a first step towards the interpretation of a sentence.
XWN LFs fall in the category of natural language based knowledge representation, a class of representations which aims to be *computer-friendly* and *human-friendly* at the same time.
Indeed, they are undoubtedly easy to understand even for a non-trained reader but still easy to compute.
Advantages of using LFs in NLP are manifold. As highlighted by Anthony and Patrick 2004 «*advantages specifically related to the utilisation of logical forms in language processing include a simplified interface between syntax and semantics, a natural and easily exploitable representation of syntactic arguments, and the potential for formation of conceptual predicates*»
They are defined by Altaf et al. 2004 as «*simple yet highly effective*» and their utility in Q/A systems has been proved in several works (See Moldovan and Rus 2001/1 e 2, Rus 2002/2).

In the following sections I review several works on LF: the STEP2008 workshop, the Senseval3 Task, two different projects that map WN glosses into LF (ILF and) and applications of LF to Q/A.

3.3.1 LF and STEP2008

XWN is not the only project that aims to produce LFs for English sentences and, as pointed out in Delmonte and Rotondi 2015, there are different ways of computing and building them.
For instance, LF can include more semantics than XWN LF do; some valuable attempts of this kind are illustrated in the Proceedings of the ACL Sigsem workshop on Semantic in Text Processing (STEP 2008)[27] (Bos and Delmonte 2008).
For example Bos proposed Boxer, a software for computing and reasoning with semantic representations. This tool produces a semantic representation named Discourse Representation Structure (DRS) and translates it to LF in order to perform inference. As we can see in figure 3.6 from Bos 2008, variables in the Boxer LF representation are all of the same kind and there is no distinction for events. Semantic/thematic roles are reified, and head the variables of both argument and events.

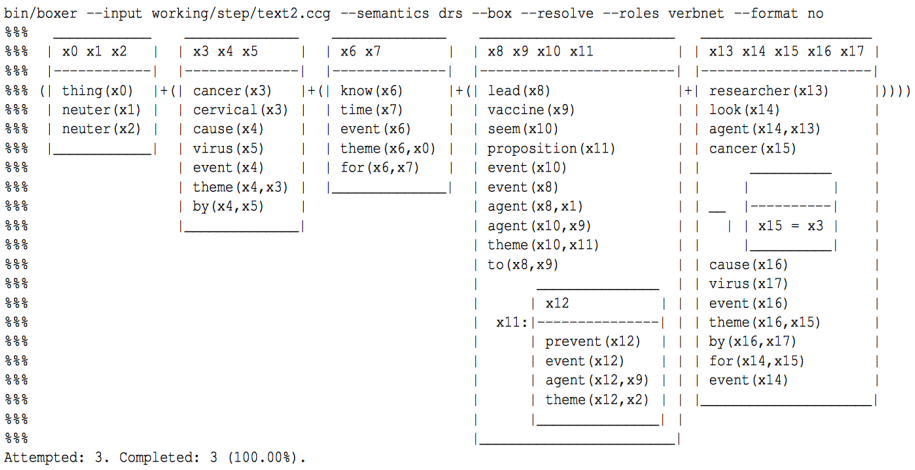


Fig 3.6 Example of Boxer Output for the Text: *Cervical cancer is caused by a virus. That has been known for some time and it has led to a vaccine that seems to prevent it. Researchers have been looking for other cancers that may be caused by viruses.*

[27] See the website of STEP2008 workshop: http://project.cgm.unive.it/events/STEP2008/index.htm and Sigsem website: http://www.sigsem.org/w/index.php?title=SIGSEM:About

Another approach was suggested by Clark with the BLUE system. BLUE comprises a parser, a LF generator, an initial logic generator and subsequent processing modules. During the parsing the system generates also a simplified LF which includes, among others, plurals, tense and polarity. An example of this LF from Clark et al. 2008 is shown below.

```
;;; LF for "A soldier was killed in a gun battle."
(DECL ((VAR _X1 "a" "soldier")
       (VAR _X2 "a" "battle" (NN "gun" "battle")))
       (S (PAST) NIL "kill" _X1 (PP "in" _X2)))
```

This mixed structure is then used to «*generate ground logical assertions of the form r(x,y), containing Skolem instances (denoting existentially quantified variables) by applying a set of simple, syntactic rewrite rules recursively to it. Verbs are reified as individuals, Davidsonian-style*» (Clark and Harrison 2008):

```
;;; logic for "A soldier was killed in a gun battle."
object(kill01,soldier01)
in(kill01,battle01)
modifier(battle01,gun01)
```

As commented in Delmonte and Rotondi 2015 «*predicates used in this representation are just syntactic relations of the type SUBJect_of, OBJect_of, and MODifier_of and all prepositions, which typically take two variables related to the individuals they are bound to. In particular, in this representation Skolem instances are associated with its corresponding input word. Syntactic relations represent deep relations: the surface subject of the passive sentence Sent.3 is turned into an OBJect*».

The STEP2008 proceedings include also another reach way of representing meaning in LF, a component of the GETARUNS

system for text understanding (Delmonte 2008). Here below an example of this representation where two semantic elements appear: DEFINITENESS and TENSE (associated to the Reference Time location variable T2):

```
LOGICAL FORM
wff(situation,
    wff(go,
        < entity : sn4 : wff(isa, sn4,  John) >,
        < indefinite : sn5 : wff(isa, sn5, restaurant) >,
        < event : f1 :
            wff(and, wff(isa,  f1,  ev),
                     wff(time, f1, < definite : t2 :
                         wff(and, wff(isa, t2, tloc),
                                  wff(pres, t2)) >)) >))
```

3.3.2 LF Resources - ILF and WN30-lfs

As seen in the previous section, XWN LFs are not the only existing LFs. In additions to the representations of STEP 2008 there are other attempts in producing LFs, like for instance, the LFToolkit[28] by N. Rathod and J. Hobbs, the experiment reported by Alshawi et al. 2011 and the work of Aoife et al. 2007. See also Wilks 1993 for more work on the topic.
What is more interesting for this work are the other two available resources of LF derived from WN glosses: ILF and WN30-lfs.

[28] See LFToolkit webpage: http://www.isi.edu/~hobbs/LFToolkit/

In the next two sections I will describe and comment these two resources highlighting their qualities and especially their defects, some of which affect XWN as well (more on this in Chapter 4).

3.3.2.1 WN30-lfs

In order to exploit the considerable amounts of world knowledge contained in WN words sense definitions, USC/ISI[29] people translated WN glosses into LF. Their results are available and freely downloadable online[30]. As clearly stated in the project 's webpage, LF provides further semantic information to supplement the WN 3.0 release.
The resulting LFs are divided into two files, one for most of the WN glosses and a second one regarding the Core WordNet[31]. As authors comment, LFs for the Core WN are generally of higher quality then the other ones.
The WN30-lfs transformation pipeline works as follow:

- each gloss is converted into a sentence of the form "word is gloss"
- the processed definitions are parsered using the Charniak parser

---

[29] Information Science Institute of the University of California

[30] https://wordnet.princeton.edu/wordnet/download/standoff

[31] WN contains thousands of Synset referring to highly specific concepts that are less relevant for NLP. Core WN has been semi-automatically compiled/populated? with 5 thousands synsets that express frequent and salient concepts.

- the parse tree is converted into a logical syntax (shallow logical form, see Hobbs 1985) by the LFToolkit that translates lexical items into logical fragments involving variables
- after the identification of syntactic relations, variables of the constituents are matched
- predicates are assigned word sense using the WN semantically annotated corpus[32] (partially achieved)

Thus, each lexical semantic head is transformed into logical fragments involving variables; for e.g. *John works* is translated into *John(x1)&work(e,x2)&present(e)*. At the beginning object variables are differentiated, and then *John* is recognised as the subject of *works* and the two variables x1 and x2 are set equal to each other. When the system fails, the constituents are translated into logic anyway; only the equalities between variables are lost lacking in this way the connections between constituents.

WN30-lfs has been released in XML format for the whole WN glosses (104K entries, half marked as *partial* half as *complete*) and in plain text for the CoreWN glosses (3K entries).
In the following example the LF transformation for the gloss of *butter* from the whole WN file and for the CoreWN respectively:

```
<entry word="butter#n#1" status="partial">
    <gloss>an edible emulsion of fat globules made by churning
milk or cream; for cooking and table use</gloss>
    <lf>butter#n#1'(e0,x0) -> edible'(e9,x1) +
emulsion#n#1'(e1,x1) + of'(e6,x1,x12) + fat#n#1'(e15,x17) +
nn'(e14,x17,x12) + globule#n#1'(e10,x12) +
dset(s5,x12,e10+e14) + make#v#15'(e2,x4,x3,x2) +
```

[32] http://wordnet.princeton.edu/glosstag.shtml

```
by'(e3,x5,e7) + churn#v#1'(e7,x10,x14) + milk/
cream#n#2'(e11,x14) + for'(e4,x6,x11) + cooking'(e12,x16) +
table'(e13,x15)</lf>
    <sublf>milk'(e11,x14) -> milk/cream#n#2'(e,x14)</sublf>
    <sublf>cream#n#2'(e11,x14) -> milk/cream#n#2'(e,x14)</
    sublf>
    </entry>


butter'(e4,x1) -> an'(e8,x1,e5) & edible'(e9,x1) & of'(e6,x1,x6)
& emulsion'(e5,x1) & fat'(e15,x6) & globule'(e10,x6) &
typelt'(e18,x6,s2) & make'(e11,x10,x6) & by'(e14,e11,e17) &
Progressive'(e21,e17) & churn'(e17,x13,x15) &
orn'(e22,x15,x17,x18) & milk'(e24,x17) & cream'(e23,x18)
```

It is evident that the two LFs are different. While the XML format allows for pos tags of (most of) the content words the LF in plain text lack them. The last don't include the text after semicolon in the gloss (which usually is not part of the sentence-definiton) as well as contents between brackets:

```
<entry word="getaway#n#2" status="complete">
    <gloss>a rapid escape (as by criminals)</gloss>
    <lf>getaway#n#2'(e0,x0) -> rapid#a#1'(e2,x0) +
escape#n#1'(e0,x0) + as'(e3,e1) + by'(e1,x0,x1) +
criminal#n#1'(e4,x1) + dset(s2,x1,e4)</lf>
    </entry>
```

There are several other differences between the LFs in the two formats (for e.g. inclusion/exclusion of determiners, representation of conjunctions, arguments and features of verbs etc.) that I do not refer here. Comparing WN30-lfs to XWN is more interesting, and to do so I will evaluate WN30-lfs of the whole WN glosses file.

Contrary to XWN, WN30-lfs is not divided into different pos files and a unique file contains all the LFs. The syntactic category of the word whose gloss is translated into LF appears only in the LHS of the LF:

```
<lf>apple_juice#n#1'(e0,x0) -> juice'(e0,x0) + of'(e1,x0,x1) +
apple#n#1'(e2,x1) + dset(s2,x1,e2)</lf>
<lf>cheer#v#4'(e0,x0) -> become'(e1,x1,e2) +
cheerful#a#1’(e3,x2)</lf>
<lf>agile#a#1'(e0,x0) -> moving#a#1'(e1,x1) +
quickly#r#1'(e3,e2) + lightly'(e5,e4)</lf>
<lf>artistically#r#1'(e0,x0) -> in'(e0,x0,x1) +
artistic#a#2'(e2,x1) + manner’(e1,x1)</lf>
```

Every word in the LHS of the LF is provided with its WSD tag but not with the SynsetID which I think to be the best straightforward link to a WN sense. Furthermore, no manual checking occurred for none of the LFs.
As highlighted in Table 3.4 the size of the two resources is different and XWN includes 20K LFs more than WN30-lfs. The difference is more evident for nouns and adjectives.

POS	WN30-lfs	XWN
noun	71391	94868
verb	13156	14441
adjective	15743	20380
adverb	3502	3994
total	103792	133683

Tab 3.4 Number of LFs per Pos

Both resources include WSD; WN30-lfs directly in the LF, XWN in the dedicated element of the XML tree. While WSD in WN30-lfs may be missing, in XWN is systematic.
The simple syntax of XWN LF is easier to follow compared to the eventuality notation chosen for WN30-lfs, where all predicates have an event variable *e* associated to them . Delmonte and Rotondi 2015 find this «*representation in eventuality notation too cluttered with additional event variables, which makes the LF entry too heavy to read*». They also points out that LF of WN30-lfs, especially the partial ones, «*contains a lot of unbound or ungrounded variables*» see for instance make(e2,x4,x3,x2) in the LF of *butter* above «*where none of the object variables have an individual ground object linked to them*». The problem of unbound variables is pointed out also by Agerri and Peñas 2010: «*…it is difficult to understand the fact that the logical forms of WN30-lfs often contain free variables and/or predicates without any relation with any other predicates in the definition*». They agree with Delmonte and Rotondi 2015 also about the complexity of WN30-lfs representation: «*Another issue is the apparent complexity of the logical forms themselves, containing predicates of an unclear number of arguments, or making decisions (such as collapsing the coordinating disjunction 'or' into the two predicates that it links) with no explained benefits*».

3.3.2.2 Intermediate Logic Forms - ILF

Inspired by XWN and WN30-lfs Agerri and Peñas 2010 «*believe that there is still some need for providing lexical and/or*

knowledge resources suitable for computational semantics tasks that required formalized knowledge». They propose a LF of WN glosses named Intermediate Logic Form (ILF) which includes «*neo-Davidsonian reification in a simple and flat syntax close to natural language*».
Agirre and Peñas don't use first-order-logic operators and aim to provide a formal representation close to NL and suitable for semantic inference tasks. The pipeline of their system performs the following operations:

- Pre-processing of the gloss inspired by XWN (some contents are deleted and definitions are extended depending on POS)
- tokenization using the C&C tokenizer *tokkie* (Clark and Curran 2007)
- POS tagging using the CRFTtagger[33])
- Syntactic analysis using the Stanford Dependency Parser[34]
- ILF generation directly from the dependency structure using an in-house system

The final output is a well structured XML tree whose elements are briefly described here below.
Considering Example 3.7, which illustrates the output for the gloss of the adjective *Bigheaded*, we can se that the ILF output is structured in Synsets. Every *sense* element has three attributes: *offset* (a unique numeric identifier whose first number identifies the pos), its *pos* category and the Synset *name* (word+pos+sense number). Every *sense* element has some sub-elements, two of

[33] http://sourceforge.net/projects/crftagger/

[34] https://nlp.stanford.edu/software/lex-parser.shtml

```
<sense offset="301890382" pos="s" synset_name="bigheaded.s.01">
    <gloss>
      <text>Something overly conceited or arrogant.</text>
      <parse>
        <s id="1">
          <words pos="true">
            <word ind="1" pos="NN">something</word>
            <word ind="2" pos="RB">overly</word>
            <word ind="3" pos="JJ">conceited</word>
            <word ind="4" pos="CC">or</word>
            <word ind="5" pos="JJ">arrogant</word>
            <word ind="6" pos=".">.</word>
          </words>
          <dependencies style="typed">
            <dep type="advmod">
              <governor idx="3">conceited</governor>
              <dependent idx="2">overly</dependent>
            </dep>
            <dep type="amod">
              <governor idx="1">something</governor>
              <dependent idx="3">conceited</dependent>
            </dep>
            <dep type="amod">
              <governor idx="1">something</governor>
              <dependent idx="5">arrogant</dependent>
            </dep>
            <dep type="conj_or">
              <governor idx="3">conceited</governor>
              <dependent idx="5">arrogant</dependent>
            </dep>
          </dependencies>
        </s>
      </parse>
      <ilf>[rel(1,3,2,'advmod',G1_3,G1_2), rel(1,1,3,'amod',G1_1,G1_3),
      rel(1,1,5,'amod',G1_1,G1_5), rel(1,3,5,'conj_or',G1_3,G1_5), e(1,2,G1_2),
      w(1,2,'overly','r','rb'), e(1,3,G1_3), w(1,3,'conceited','a','jj'),
      syn(1,3,301891773), e(1,1,G1_1), w(1,1,'something','n','nn'), e(1,5,G1_5),
      w(1,5,'arrogant','a','jj'), syn(1,5,301889819)]</ilf>
      <pretty-ilf>something(x1) amod(x1,x3) amod(x1,x5) overly(x2) conceited(x3)
      advmod(x3,x2) conj_or(x3,x5) arrogant(x5)</pretty-ilf>
    </gloss>
    <lemma id="0">bigheaded</lemma>
    <lemma id="1">persnickety</lemma>
    <lemma id="2">snooty</lemma>
    <lemma id="3">snot-nosed</lemma>
    <lemma id="4">snotty</lemma>
    <lemma id="5">stuck-up</lemma>
    <lemma id="6">too_big_for_one's_breeches</lemma>
    <lemma id="7">uppish</lemma>
    <example id="0">a snotty little scion of a degenerate family"-Laurent Le
      Sage</example>
    <example id="1">they're snobs--stuck-up and uppity and persnickety</example>
  </sense>
```

Fig 3.7 ILF Output

them are required: *gloss* and one or more *lemma*, which contain the different words by which a sense is expressed, while *example* is optional. The nested elements of *gloss* contain the definition of the word, the pos tagging, the dependency structure and the logical form.

The *pretty-ilf* element provides a better readability version of the LF.

The authors «*performed extreme neo-davidsonian reification aiming to reduce the number of free-variables in the resultant logical form*» and «*every relation between discourse referents is expressed by a predicate*» without using logic boolean connectives. The resulting flat and simple syntax is similar to XML LF.

Inspired by WN30-lfs and XWN, ILF aims to provide lexical knowledge improving the weakness of the previous resources.

To do so they start by strengthening the preprocessing phase. They modify the definitions by extending them depending on the pos[35] and by removing «*any redundant and superfluous information*».

The gloss of the previous example, bigheaded:

```
used colloquially of one who is overly conceited or arrogant; "a
snotty little scion of a degenerate family"-Laurent Le Sage;
"they're snobs--stuck-up and uppity and persnickety"
```

results in the ILF LF :

```
something(x1) amod(x1,x3) amod(x1,x5) overly(x2) conceited(x3)
advmod(x3,x2) conj_or(x3,x5) arrogant(x5)
```

as authors comments, the most relevant concepts of the definition result to be in a prominent position and not buried among other

[35] NOUN and ADV glosses were extended with a period at the end of it, VERB glosses were extended with "to" in front of the gloss and a period at the end of it, ADJ glosses were extended with "Something" in front of the gloss and a period at the end of it.

not so relevant information. I think that by removing part of the definition the risk of loosing information is high.
Furthermore, during the pre-processing phase, if a semicolon is present in the gloss, they delete everything after it. This works well when a gloss contains one definition + 1 or more examples, but sometimes after a semicolon there is still some knowledge it is worth considering. See the following examples (pos+word+gloss), one for each pos category:

N position = the act of positing; a**n assumption taken as a postulate or axiom**

V beam = smile radiantly; **express joy through one's facial expression**

A clean = free from clumsiness; **precisely or deftly executed;** "he landed a clean left on his opponent's cheek"; "a clean throw"; "the neat exactness of the surgeon's knife"

R well = without unusual distress or resentment; **with good humor;** "took the joke well"; "took the tragic news well"

In XWN examples are excluded from LF but the content after semicolon is represented as additional LF. Thus, for instance, the previous gloss of *beam* is represented in XWN with a double LF:

```
<lft quality="GOLD">
beam:VB(e1,x1,x2) -> smile:VB(e1,x1,x3) radiantly:RB(e1)
</lft>
<lft quality="GOLD">
beam:VB(e1,x1,x2) -> express:VB(e1,x1,x3) joy:NN(x3)
through:IN(e1,x4) facial:JJ(x4) expression:NN(x4)
</lft>
```

The result of this choice is that even though both XWN and ILF translate all the glosses of WN into LF, only XWN contains all their world knowledge.
Moreover, in the pre-processing phase, text between brackets should be removed but this sometimes doesn't happen in ILF:

```
<sense offset="114709791" pos="n"
synset_name="absolute_alcohol.n.01">
    <gloss>
      <text>Pure ethyl alcohol (containing no more than 1%
water).</text>
      <parse>...</parse>
      <ilf>..</ilf>
      <pretty-ilf>pure(x1) ethyl(x2) alcohol(x3) amod(x3,x1)
    nn(x3,x2) ((x4) nsubj(x4,x3) dep(x4,x5) contain(x5)
    advmod(x5,x6) dobj(x5,x12) no(x6) dep(x6,x10) more(x7)
than(x8) advmod(x8,x7) 1(x9) quantmod(x9,x8) %(x10)
num(x10,x9) water(x11) )(x12) nn(x12,x11)</pretty-ilf>
    </gloss>
    <lemma id="0">absolute_alcohol</lemma>
</sense>
```

The use of the predicate *nn* for compound nouns has been improved in comparison with XWN, this in particular if we check for the particular case of *world_war_II* (more on this in the next chapter); but taking a closer look we found many unneeded uses of the predicate *nn* as for instance in *coarse tobacco*:

```
<sense offset="114715356" pos="n" synset_name="shag.n.01">
    <gloss>
      <text>A strong coarse tobacco that has been shredded.
      </text>
      <parse>
        <s id="1">
          <words pos="true">
            <word ind="1" pos="DT">a</word>
```

```
            <word ind="2" pos="JJ">strong</word>
            <word ind="3" pos="NN">coarse</word>
            <word ind="4" pos="NN">tobacco</word>
            ...
      </parse>
      <ilf>...</ilf>
      <pretty-ilf>a(x1) strong(x2) coarse(x3) tobacco(x4)
    det(x4,x1) amod(x4,x2) nn(x4,x3) rcmod(x4,x8) that(x5)
have(x6) be(x7) shred(x8) nsubjpass(x8,x4) rel(x8,x5)
aux(x8,x6) auxpass(x8,x7)</pretty-ilf>
    </gloss>
    <lemma id="0">shag</lemma>
</sense>
```

where it might have been a tagging error. More tagging errors occur with colour, nouns and past participles.
Other mistakes concern wrong cases of pp attachment as in the following example:

```
  <sense offset="107312829" pos="n" synset_name="migration.n.
03">
    <gloss>
      <text>The nonrandom movement of an atom or radical from
    one place to another within a molecule.</text>
      <parse>
        <s id="1">
          <words pos="true">
            <word ind="1" pos="DT">the</word>
            <word ind="2" pos="JJ">nonrandom</word>
            <word ind="3" pos="NN">movement</word>
            <word ind="4" pos="IN">of</word>
            <word ind="5" pos="DT">a</word>
            <word ind="6" pos="NN">atom</word>
            <word ind="7" pos="CC">or</word>
            <word ind="8" pos="JJ">radical</word>
            ...
      </parse>
      <ilf>...</ilf>
```

```
      <pretty-ilf>the(x1) nonrandom(x2) movement(x3) det(x3,x1)
     amod(x3,x2) prep_of(x3,x6) prep_of(x3,x11) prep_to(x3,x13)
     a(x5) atom(x6) det(x6,x5) conj_or(x6,x11) radical(x8)
     prep_from(x8,x10) one(x10) place(x11) amod(x11,x8)
     another(x13) prep_within(x13,x16) a(x15) molecule(x16)
det(x16,x15)</pretty-ilf>
    </gloss>
    <lemma id="0">migration</lemma>
  </sense>
```

Here again the errors might be caused by the wrong tag associated to *radical*. It would be interesting to know the error rate due to the LF-transformation algorithm compared to tagging errors, but for our purposes it is enough to know that some kind of errors affect both ILF and XWN.

Last but not least, contrary to XWN and WN30-lfs, ILF doesn't perform WSD. WSD is an important feature and its lack in ILF is seen as a required improvement by the authors themselves.

Considering positive features of ILF, it is important to point out that, contrary to XWN and WN30-lfs, in ILF there are no unbound variables. This is a remarkable improvement and we will face the problem of unbound variables in XWN in the next chapter.

3.3.3 XWN LF and Senseval3

The work of Moldovan and Rus, i.e. XWN LF, inspired one of the tasks of the Third International Workshop on the Evaluation of Systems for the Semantic Analysis (Senseval-3): Identification of

Logic Forms in English[36]. The goal of the task is the evaluation of systems that transform English sentences into LF. The motivations that stand behind this further research on LF are their advantages over similar representations (Montague-style recursive semantics and Description Grammars). According to the coordinator of the task (see Rus 2004) LFs:

- have a simple syntax/semantics interface
- are user-friendly even for no trained users
- have positional syntactic arguments that ease other NLP tasks such as textual interpretation
- are easy customisable
- predicates might be disambiguated and turned into concept predicates (see Rus 2009)

For the Senseval task a gold standard of 300 LF was provided. The gold LF were automatically produced by applying an extended version of the LF derivation engine developed by Rus for XWN to English sentences. The output of Rus' engine was checked manually (by different annotators) before the release.
The outputs of participant systems were compared to the gold standard LF, evaluating precision and recall.
The target LF of the task are very similar to XWN LF. They share most of the features: they are both flat, scope-free first order logic representations formed by the conjunction of individual predicates related via shared arguments.

---

[36] See the task web page: http://www.cs.memphis.edu/~vrus/logic/indexLF.html and Senseval-3 web site: http://web.eecs.umich.edu/~mihalcea/senseval/senseval3/

Predicates are generated for content words, prepositions and conjunctions. The name of the predicate is formed by the lemmatised word and the pos category. The types of argument are two: *e* for events, *x* for entities and their order is fixed. Simplifications similar to XWN LF are adopted: determiners, punctuation, plurals, auxiliaries and verb tenses are ignored. There is no quantification or negation. Few features differentiate the LF of Senseval task from XWN LF. The most important one is the distinction between complements and adjuncts which is not considered at all in WXN LF[37]. See the LF for the sentence: *The earth provides the food humans eat every day*.

Senseval LF: `earth:n_(x1) provide:v_(e1, x1, x2) food:n_(x2) human:n_(x3) eat:v_(e2, x3, x2; x4) day:n_(x4)`

In the example above, we can see that the verb *eat* has two arguments separated by comma, whereas the semicolon separates the adjuncts (in this case x4). More examples in appendix 3 and 4 (trial data).
Only 4 out of 27 participating teams completed the assignment and produced a valid output:

- The University of Amsterdam (Ahn et al. 2004, AMS): the core of the dutch system is a syntactic analysis module with the following framework: pos tagging, syntactic parsing (Collins), conversion of the parser output into a dependency structure, improvement of the dependency structure with non local

[37] From [83] it seems that the distinction between complements and adjuncts was not required in the task.

dependencies and functional tags obtained from the Wall Street Journal corpus. LF are derived directly from the dependency structure. The main sources of errors identified for the dutch LF are: errors in the dependency parser, not identified multi-word compounds and inconsistencies in the provided LFs.

- The University of Sidney (Anthony and Patrick 2004, SYD): Anthony and Patrick used a functional dependency parser which includes properties associated to each token (e.g. main element, syntactic object position etc.). Examples of dependency functions employed are: agent, object complement, subject etc. The output of the parser is transformed into a linear data structure sorted by word position and passed to the filter module that removes elements like determiners and auxiliaries. The logical form processor, fed with the remaining tokens, builds an inverted index with grammatical dependencies and constructs the LF representation. Most of the errors in the final LF comes from a poor handling of nominal group complexes and coordinating conjunctions (*and*, *or* etc.)
- The MITRE (Bayer et al. 2004, MITRE): the MITRE's contribution is a system whose first component is a link grammar parser which produces labeled undirected links among pairs of words. A link interpretation language made of rules is used to convert the link parser output into a dependency graph. LFs are not derived directly from the dependency structure and before the LF transformation step, extra information is added from external resources and argument networks are constructed for each dependency object.

- The Language Computer Corporation (Altaf et al. 2004, LCC): LCC's approach is similar to the one used for the generation of XWN LFs. LCC LFs are derived directly from the output of the syntactic parser with a bottom-up procedure that generates independent arguments first, and then propagates up the parse tree marking heads of phrases and identifying dependent arguments.

The four participating teams agree about the lack of complete specification for different special cases in the target LF (e.g. little guidance about collocation as pointed out by Bayer et al. 2004); the details of the task seem to be vague in different cases and this might have affected the results. Beside that, the Dutch team found inconsistencies in the provided target LF (trial corpus); in particular they found discrepancies in the LF representations of verbs with particles (sometimes represented as combinations, sometimes not), missing arguments and verbs not reduced to base form.

Results of the task were evaluated considering precision and recall at argument and predicate level; they estimated also the number of sentences correctly transformed over the total number of sentences attempted.

The measures of the task evaluation are described in Rus 2004: « *precision at argument level: number of correct identified arguments divided by the number of all identified arguments; recall at argument level: number of correctly identified arguments divided by the number of arguments that were supposed to be identified. Precision at predicate level: number of correctly and fully identified predicates (with all arguments correctly identified)*

divided by the number of all attempted predicates; recall at predicate level: number of correctly and fully identified predicates (with all arguments correctly identified) divided by the number of all predicates that were supposed to be identified»;
and in Anthony and Patrick 2004: «*Sentence-argument is defined as the number of sentences that have all arguments correctly identified divided by the number of sentences attempted. Sentence-predicate is similar except conditioned on predicates. Sentence-argument-predicate is defined to be the number of sentences that have all arguments correctly identified divided by the number of sentences which have all predicates correctly identified. Sentence-argument-predicate-sentences refers to the number of sentences that have all arguments and all predicates correctly identified divided by the number of sentences attempted*».
In Table 3.5 (next page) a comparative view of the results form Rus 2004, the section regarding Sentence-argument-predicate-sentences (Sent-APSent) results, i.e. LF entirely correctly derived, is highlighted. Tab8 is organised in alphabetic order considering the names of the teams (and not considering the results).
At a glance it is clear that results are not satisfactory, especially if we envisage using this LFs for a NLP application whose success relies on the accuracy achieved by its components.
This is even more clear if we consider Sent-AP Sent value.
Discrepancies in the trial corpus and the results of the task are a clear sign of how inconsistencies affect LF transformation (especially when fully automatic). In addition to this, the fact that only 4 teams over the 24 registered produced a valid output proves the difficulty of the task and of the LF automatic transformation in general.

TEAM	Argument Level		Predicate Level	
	Precision	Recall	Precision	Recall
AMS	0,729	0,691	0,819	0,783
LCC	0,776	0,777	0,876	0,908
MITRE	0,734	0,659	0,839	0,781
SYD	0,763	0,655	0,839	0,849
	Sent-A	Sent-P	Sent-AP	Sent-AP Sent
AMS	0,256	0,320	0,510	0,163
LCC	0,236	0,516	0,419	0,216
MITRE	0,266	0,213	0,406	0,086
SYD	0,160	0,353	0,386	0,136

Tab 3.5 Senseval-3 Results

3.3.4 LF and Question Answering

LF have been applied in several challenging tasks such as question answering and learning by reading (see for e.g Barker et al. 2007).

In particular, the XWN logic representation was developed in order to «*enable reasoning mechanisms for many practical applications*» (Moldovan and Rus 2001/2).

I will briefly show here below an example of how to use LFs in Q/A.

Q/A is a very challenging and widely studied task for which several approaches have been proposed. The employment of LFs has been proved to be a valid support for its accomplishment in numerous papers (see Harabagiu et al. 2000, Moldovan et al. 2002, Moldovan et al. 2003, Moldovan and Rus 2001/ 1 and 2, Rus 2002/2)

It is well known that, in order to solve common sense reasoning problems, it is necessary to have world knowledge. A human being can add to a sentence many information that are not explicitly stated and that come from his knowledge of the world. This part of information is extremely important and affects the results of automatic inference systems. Grasser 1981 affirms that the ratio of explicit to implicit facts is 1 to 8. Let's see some examples.

Given the sentence:

A soldier was killed in gun battle

a reader may infer that:

There was a war; The soldier was shot; The soldier died

These additional information might seem obvious, but, in order to get them, an automatic system needs a large resource to consult. WN Glosses encode a lot of knowledge that can be exploited for this purpose. Considering the above example, the gloss of *battle:N#1* (first sense of noun *battle*) is *a hostile meeting of opposing military forces in the course of a war,* the gloss of *kill:V#1* (first sense of the verb *kill*) is *cause to die*, and the gloss

of gun:V#1 (first sense of the verb *gun*) is *shoot with a gun*. Given these extra information is now possible for an automatic system to infer that a battle can be part of a war where soldiers are shot with guns and die.
Another example is from TREC 2000[38]. Given the question:

How did Socrates die?

The answer appears in the text "… *Socrates's death came when he chose to drink poisoned wine*…". To prove that this sentence contains the answer to the previous question, a system needs to know that *poison* can be used to *kill*. This knowledge is found in WN glosses and in particular in the gloss of *poison:V#2* (second sense of the verb *poison*) *kill with poison* plus the already seen gloss of *kill:V#1* (first sense of the verb *kill*) *cause to die*.
All this precious extra knowledge needs to be represented in a suitable way in order to be understood by a system. LF has been proved to be fit for the purpose thanks to its unambiguous and simple syntax.
The automatic Q/A systems proposed in the aforementioned papers (in particular: Moldovan et al. 2002, Moldovan and Rus 2001/1 and 2, Rus 2002/2) follow a similar pipeline:

- first of all, for each question, a set of candidates paragraph that may contain the answer is retrieved

[38] TREC is the Text REtrieval Conference whose purpose is «*to support research within the information retrieval community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies*». Workshops and exercises are provided every year; resulting data and evaluation software «are available to the retrieval research community at large, so organizations can evaluate their own retrieval systems at any time». See TREC Website: http://trec.nist.gov

- evaluating pairs of question(Q)-candidate answer(A) in LF, if all the keywords of Q are found in A the system checks the preservation of syntactic relations considering their LFs
- if some keywords are missing the system tries to establish lexical chains (LC) between pairs of concepts in Q and A favouring short path and hypernym relations. A chain is established when two words have a WN concept in common.
- Once LC are established, the resulting LFs are used to perform agreement unification. In a successful unification the arguments of a question predicate are bound to the arguments of an answer predicate.

During the LC phase LFs are expanded with axioms obtained from WN and then predicates and arguments from Q are matched with predicates and arguments from A in a recursive way. A Q is successfully proven when all its predicates and arguments are matched with the ones of A.
I use another example from TREC to show how the system works. Considering the question:

Who shot Billy the Kid?

The system retrieves two paragraphs that contain all the keywords from Q and A:

P1- *The scene called for Philips' character to be saved from a lynching when Billy the Kid (Emilio Estevez) shot the rope in half just as he was about to be hanged*.

P2- *In 1881, outlaw William H. Bonney Jr., alias Billy the Kid, was shot and killed by Sheriff Pat Garrett in Fort Sumner, N.M.*

Question and candidate answers are translated into LF as follow (only the relevant parts are shown):

```
Q- PERSON(x1) & shoot(e1,x1,x2) & Billy_the_Kid(x2)
P1- Billy_the_Kid(x1) & shoot(e1,x1,x2) & rope(x2)
P2- Billy_the_Kid(x2) & shoot(e1,x1,x2) &
Sheriff_Pat_Garrett(x1)
```

The system tries to prove the question starting from the paragraphs: it fails for P1 because *Billy_the_Kid* is the agent and not the object as requested in the question and then it succeeds with P2 by unifying *Sheriff_Pat_Garrett* with PERSON. As pointed out by Moldovan et al. 2002 the LF fixed slot allocation of verb predicate plays here a crucial role.

If not all the keywords are found in Q and A, the system uses LC. Considering the following question:

Q- When did Lucelly Garcia, former ambassador of Columbia to Honduras, die?

whose answer is found in the paragraph:

P- *Several gunmen on a highway leading to the Colombian city of Ibague murdered Colombian Ambassador to Honduras Lucelly Garcia today*

here only some of the Q-keywords are found in P (*Lucelly_Garcia*, *ambassador*, TIME-STAMP), and lexical chains are built from *Colombian* to *Columbia* and from *murder* to *die* before the system succeeds in identifying the answer (more details in Moldovan and Rus 2001/2).

3.4 Conclusions

Considering the proved usefulness of LF and in particular the need of LF resources for computational semantics tasks, I decided to give my contribution by creating an accurate LF resource.
Delmonte and Rotondi 2012 points out the need of a precise LF representation and affirms that «*Logical Forms are useful as long as they are consistent, otherwise they would be useless if not harmful*».
As highlighted by the different works on LF, it seems that this kind of representation cannot be produced fully automatically and need a lot of additional work. The automatic production of LF is still error prone and the already existing resources of LF suffer from lack of an accurate manual checking phase; WN30-lfs and ILF don't make use of manual checking and only the 34% of XWN LF has been manually checked.
Thus, I thought it is worthwhile to correct existing LF resources rather than producing a new one from scratch.
I considered in particular those resources that translate WN glosses into LF because of the importance of WN for the NLP community and the huge amount of world knowledge and semantic relations that it encodes.

After comparing the three existing resources of LF, I chose to improve XWN. The reasons of my choice are manifold.
As seen above, XWN is the most extended freely downloadable resource of LFs. Its bigger size, compared to the other two resources, is due to the transformation of all the glosses of WN without loosing part of them.
XWN is divided into 4 files, one for each pos, and this makes its application easier; furthermore the XML format allows a simple access to the data encoded.
XWN provides WSD and its LF has a simple syntax close to NL.
As pointed out in different papers, XWN is not devoid of errors. In the next chapter I will examine XWN showing which kinds of errors affect the resource (in particular the LFs).

Chapter 4

Errors Detection

4.1 Introduction

LF is an exceptionally important linguistic representation for highly demanding semantically related tasks like QA. We have seen in the previous chapter how its automatic production at runtime is error-prone and therefore I decided to improve one of the existing resources by correcting its errors.
I chose to work on XWN for the above mentioned reasons and I will start the work by classifying the mistakes I found in XWN LFs in order to facilitate the revision of the resource.
This chapter concerns the errors I found in XWN LFs; the most interesting common mistakes will be shown, grouped into classes, from section 4.2.1 to section 4.2.7.
As I will illustrate, errors affect all the qualities of LFs, even those that have been manually checked. Moreover, we will see that a good syntactic parsing is necessary but doesn't guarantee the correctness of the LF.

4.2 Errors Classification

Any resource that aims at providing a meaning representation for NLP tasks must be as more accurate as possible as its errors would damage possible applications.
LF transformation still requires a big effort in manual checking which is often missing in exiting LF resources. The number of errors in XWN, ILF and WN30-lfs is too high to make these resources usable as they are.
Some papers mention different problems that affect the LFs of these resources and therefore I will start my analysis from the already-known mistakes of XWN before pursuing with the detection of new cases. The investigation will be done by manually checking the resource with the help of regular expressions and some in-house Python systems.
I will group the most interesting common mistakes into classes and I will present them in the next sections.

I won't accurately investigate the reasons why these errors affect LFs or what went wrong during the LF transformation because the purpose here is to correct LFs and not to improve the accuracy of the WXN LF transformation system. Therefore, I won't systematically compare LFs errors with the parse trees from which they have been derived. I will rather show the most common errors I found in the LFs in order to correct them.
The purpose of the work is twofold:

- providing the NLP community with a consistent and usable LF resource

- classifying the most important mistakes of LFs with the intention of helping future implementations and new works

4.2.1 Free Variables

Indexed variables are fundamental elements of the LF and are used to indicate relations intervening between event and arguments or modifiers. As we have seen in section 3.2.3, number and order of arguments of an event are fixed[39] (e_n, x_n/subject, x_n/ direct object) and are generated for every verb even when it doesn't have these syntactic roles (e.g. one place-predicate). In these cases arguments are dummy and their variables free:

```
laugh:NN(x1) -> sound:NN(x1) of:IN(x1, e1) laugh:VB(e1, x1, x26)
```

```
hibernate:VB(e1, x1, x2) -> sleep:VB(e1, x1, x9) during:IN(e1,
x3) winter:NN(x3)
```

```
basket:NN(x1) -> score:NN(x1) in:IN(x1, x2) basketball:NN(x2)
make:VB(e1, x5, x1) by:IN(e1, e2) throw:VB(e2, x5, x3)
ball:NN(x3) through:IN(e2, x4) hoop:NN(x4)
```

However, when the arguments are actually present they should be coindexed with the event, with particular attention to intransitive or passivised structures. Very often this does not happen, as in the following case:

```
able:JJ(x1) -> have:VB(e1, x1, x8) necessary:JJ(x8) means:NN(x2)
skill:NN(x3) know-how:NN(x4) or:CC(x8, x2, x3, x4, x5)
authority:NN(x5) to:IN(x8, e2) do:VB(e2, x8, x6)
something:NN(x6)
```

[39] Actually, for ditransitive verbs the number of arguments should change (one more argument for indirect object).

In the latter example, the subject of *do* should be x1 (and not x8) that is the person that *is able*, subject of predication of *have* and also head of the adjective modifier.
The problem of free variables was previously identified by Agerri and Peñas 2010 who point out that in XWN LF «*there are variables that do not belong to anything, and others that are left free (not related in any way with the rest of the formula)*».
Unbound variables don't allow relations indicated by predicate-arguments associations, which are thus disconnected. Variables associated to predicates need to be equated with those of the arguments of the predicate in order to acquire semantic consistency.
This is the main source of errors of XWN and from my analysis more than half of all LFs suffer from that problem.
The results of the analysis are shown, divided by pos, in Table 4.1:

POS FIle	LFs with Disconnected Variables	Total Number of examined LFs	%
Noun	49178	87819	56.00
Verb	9021	13373	67.46
Adjective	8895	20337	43.74
Adverb	479	3922	12.23
TOTAL	67573	125451	54.05

Tab 4.1 LFs with Disconnected Variables

The higher percentage of disconnected variables in the verb file is mainly due to the nature of its definitions which very often consist of verbs without arguments:

```
scour:VB(e1, x1, x2) -> examine:VB(e1, x1, x3) minutely:RB(e1)
```

```
darken:VB(e1, x1, x2) -> tarnish:VB(e2, x1, x3) or:CC(e1, e2,
e3) stain:VB(e3, x1, x4)
```

while adverb definitions have very few verbs and this explains the low percentage of disconnected variables in the adverb file; here below an example of on an entry from the adverb file:

```
convivially:RB(e1) -> in:IN(e1,x1) convivial:JJ(x1)manner:NN(x1)
```

4.2.2 Compound Nouns

As previously described, the detection of compound words during the LF transformation is a tricky task which might require human intervention and whose achievement is often error-prone. This is proved in particular by difficulties observed in ILF and in Senseval task in the previous chapter.
In XWN, as well as in ILF, special attention has been given to the detection of compound words and in particular of Compound Nouns (CN)[40] which are mapped into LF by means of the *nn*-predicate. E.g (from XWN):

---

[40] I will use *Nominal Compounds* to refer to those nouns that are the result of the union of words from different syntactic categories and *Compound Nouns* for those nouns that arise from the union of one or more nouns.

```
jab:NN(x1) -> sharp:JJ(x1) nn(x1,x2,x3) hand:NN(x2) gesture:NN(x3)
```

The *nn*-predicate transformation doesn't occur methodically in XWN and even when a CN has already been detected in the resource, it can appear in LF also as disconnected words:

```
ball:NN(x1) -> pitch:NN(x1) be:VB(e1, x1, x5) not:RB(e1)
in:IN(e1, x4) nn(x4, x2, x3) strike:NN(x2) zone:NN(x3)

strike:NN(x1) -> pitch:NN(x1) that:IN(e4, e1) be:VB(e1, x1, x26)
in:IN(e1, x2) strike:NN(x2) zone:NN(x3) and:CC(e4, e1)
that:IN(e4, x4) batter:NN(x4) do:VB(e2, x4, e3) not:RB(e2)
hit:VB(e3, x5, x4)
```

Other CNs are transformed in LF without the *nn*-predicate but as a single words, see for e.g.:

```
baseball_league:NN(x1) -> league:NN(x1) of:IN(x1, x2)
baseball_team:NN(x2)
```

But these options don't seem to be subject to any particular rule, and some CNs are detected but then transformed differently in different contexts. This is the case of *world war* which appears 159 times in the noun file and mapped in several ways:

```
snafu:NN(x1) -> acronym:NN(x1) often:RB(e0) use:VB(e1,x2,x1) by:IN(e1,
x2) soldier:NN(x2) in:IN(e1,x3) world:NN(x3) war:NN(x4) ii:JJ(x3)
situation:NN(x5) normal:JJ(x6) all:JJ(x6) fucked:NN(x6) up:IN(e1,x6)

battle_of_the_ardennes_bulge:NN(x1) -> battle:NN(x1) during:IN(x1, x2)
world:NN(x2) war:JJ(x2) ii:NN(x3)

coral_sea:NN(x1) -> japanese:JJ(x1) defeat:NN(x1) in:IN(x1,x2)
world_war_ii:NN(x2)
```

```
wac:NN(x1) -> member:NN(x1) of:IN(x1,x2) women's:NN(x2) army:NN(x3)
corp:NN(x4) be:VB(e1,x2,e2) organize:VB(e2,x9,x2) during:IN(e2,x5)
world:NN(x5) war:NN(x6) ii:NN(x7) but:CC(e4,e0,e3) be:VB(e3,x1,x8)
no:RB(e3) longer:RB(e3) separate:JJ(x8) branch:NN(x8)
```

Other words are sometimes considered as compounds:

```
hampton_roads:NN(x1) -> naval_battle:NN(x1) of:IN(x1,x2)
american_civil_war:NN(x2)
```

while in other cases they are not:

```
philippine_sea:NN(x1) -> naval:JJ(x1) battle:NN(x1) in:IN(e0, x2)
world:NN(x2) war:JJ(x2) ii:NN(x3)
```

One could affirm that both the above transformations of *naval battle* can be considered correct, which is true, but a consistent resource should provide a unique way to transform those words which are considered to be compounds.
Moreover, there are cases of CNs that have never been detected in the whole resource, see for e.g. *waste-product*:

```
reclamation:NN(x1) -> recovery:NN(x1) of:IN(x1, x2) useful:JJ(x2)
substance:NN(x2) from:IN(x2, x3) waste:NN(x3) product:NN(x4)
```

Sometimes, CNs are not identified even if they previously appear as subject of a definition:

```
healthcare:NN(x1) -> preservation:NN(x1) of:IN(x1, x2) mental:JJ(x2)
physical:JJ(x2) health:NN(x2) by:IN(x2, e4) preventing:VB(e1, x2, x26)
or:CC(e4, e1, e2) treating:VB(e2, x2, x26) illness:NN(x3)
through:IN(x3, x4) services:NN(x4) offer:VB(e3, x5, x4) by:IN(e3, x5)
health:NN(x5) profession:NN(x6)
```

```
healthcare_delivery:NN(x1) -> provision:NN(x1) of:IN(x1,x2)
health:NN(x2) care:NN(x3)
```

Not-identified CNs are wrongly transformed into two or more different predicates damaging in this way the consistency of LFs. In fact, in case a CN is not detected, two or more predicates are generated in the LF and their variables result improperly mapped, losing in this way the whole sense of the sentence.

4.2.3 Conjunctions and Prepositions

As described at page 53, also conjunctions and prepositions are transformed into predicates. Conjunctions are turned into predicates with a variable number of arguments, of which the first one represents the result of the aggregation, while prepositions have two fixed arguments. From my analysis it turns out that several conjunctions and prepositions are missing in the LFs. The problem doesn't seem to depend on the quality of the LF as it occurs even in gold - manually checked - LFs.
See for e.g. the LF of *seedcake*:

```
<gloss pos="NOUN" synsetID="07164600">
  <synonymSet>seedcake, seed_cake</synonymSet>
 <text>a sweet cake flavored with sesame or caraway seeds and lemon
 </text>
 …
 <lft quality="GOLD">
  seedcake:NN(x1) -> sweet:JJ(x1) cake:NN(x1) flavor:VB(e1, x7, x1)
with:IN(e1, x6) sesame:NN(x2) caraway:JJ(x5) seed:NN(x3) and:CC(x30,
x31, x32) lemon:NN(x4)
 </lft>
</gloss>
```

Without dwelling on other errors of the previous LF (free variables and CNs), we can see that the first conjunction *or* is missing and the coordinating conjunction *and* is assigned variables which do not have any correspondence in the representation.
In order to estimate the amount of missing conjunctions, for each entry in XWN I automatically compared[41] each definition with its LF, if the definition includes a conjunction that is missing in the LF I counted this case as a missing-conjunction-error. Results are shown in Table 4.2 (next page) divided by *and*/*or* missing cases; in the last column of Table 4.2 the number of LFs with conjunctions errors (I counted them by considering the synsetID of the missing cases, avoiding in this way double counts for those LFs that have both a missing *or* and a missing *and*).
Here again, results depend on the nature of the glosses. Definitions of adjectives and adverbs include more conjunctions than those of other pos category. At the same time they are usually less complex than for e.g. noun glosses (shorter, less subordinate clauses etc.). This means that the LF transformation failed for conjunctions independently from the complexity of the structure they occur in.

Most frequent prepositions appearing in the database are: *on, in, to, by, for, with, at, of, from, as, out*. Some of them are not transformed into LF especially if they appear at the end of the

[41] In Appendix5 the system I built for the comparison (and some examples). The system takes as input each pos XWN files (one by one) and, climbing the XML tree, for each entry it gathers the definition and the corresponding LF. If there is more than one definition in the gloss, the system consider only the first one. Errors have been counted for missing AND and missing OR as separate cases, as well as as number of LFs.

POS file	LF examined	missing AND	missing OR	LFs with conjunctions errors (num and %):
noun	79.689	3.156	2.981	5.964 - 7,5%
verb	13.507	320	1.017	1.322 - 9,8%
adjective	18.561	752	2.950	3.658 - 15,9%
adverb	3.664	148	331	476 - 13%
TOTAL	115.421	4376	7279	11.420 - 9,9%

Tab 4.2 Missing Conjunctions

gloss. The problem is the incoherence that affects LF transformation of prepositions in XWN: sometimes they are transformed, sometimes they are not, showing in this way no logic. This happens also for *gold* LFs; see the following examples where *to* is first preserved in the LF of *powder* and then erased from the LF of *talc*:

```
<gloss pos="VERB" synsetID="00040699">
  <synonymSet>powder</synonymSet>
. . .
 <text> apply powder to; "She powdered her nose"; "The King
wears a powdered wig"</text>
<lft quality="GOLD"> powder:VB(e1, x1, x2) -> apply:VB(e1, x1,
x3) powder:NN(x3) to:IN(e1, x2) </lft>
</gloss>


<gloss pos="VERB" synsetID="00040890">
  <synonymSet>talc</synonymSet>
. . .
 <text> apply talcum powder to (one's body) </text>
<lft quality="GOLD"> talc:VB(e1, x1, x2) -> apply:VB(e1, x1, x3)
talcum_powder:NN(x3) </lft>
</gloss>
```

When prepositions are part of phrasal verbs the treatment is again not homogeneous and sometimes the verb particle may be simply erased. In the following examples it appears attached to the verb in *work_out*, as a separate entry in *set up* and erased in *line (up)*:

```
<gloss pos="VERB" synsetID="00243111">
  <synonymSet>elaborate, work_out</synonymSet>
 <text> work out in detail; "elaborate a plan" </text>
  . . .
 <lft quality="GOLD">
 elaborate:VB(e1, x1, x2) -> work_out:VB(e1, x1, x4) in:IN(e1,
x3) detail:NN(x3) </lft>
</gloss>


<gloss pos="NOUN" synsetID="03977417">
  <synonymSet>sampling_station, sampler</synonymSet>
 <text> an observation station that is set up to make sample
observations of something  </text>
. . .
 <lft quality="GOLD">
  sampling_station:NN(x1) -> observation_station:NN(x1)
be:VB(e1, x1, e2) set:VB(e2, x6, x1) up:IN(e2, x5) to:IN(e2, e3)
make:VB(e3, x1, x2) sample:NN(x2) observation:NN(x3) of:IN(x2,
x4) something:NN(x4) </lft>
</gloss>


<gloss pos="NOUN" synsetID="07918617">
  <synonymSet>secondary</synonymSet>
 <text>
   the defensive football players who line up behind the linemen
 </text>
. . .
 <lft quality="NORMAL">
  secondary:JJ(x4) -> defensive:JJ(x1) football:NN(x1)
player:NN(x1) line:VB(e1, x1, x26) behind:IN(e1, x2)
linemen:NN(x2) </lft>
</gloss>
```

4.2.4 Relative Adverbs

As described in Rus 2002/1 when relative adverbs such as *where*, *when*, *how*, *why* introduce a relative clause they should be represented in LF as «*predicates with two arguments referring to the argument of the relative clause or phrase, respectively argument of the main clause or phrase*»; see the following LF:

```
arena:NN(x1) -> playing_field:NN(x1) where:IN(x1, e1) nn(x4, x2,
x3) sport:NN(x2) event:NN(x3) take_place:VB(e1, x4, x5)
```

In this LF we can see also two cases of compound words, one treated as single word (*playing_field*) and another one transformed with a *nn* predicate (*sport event*); X5 is a free variable.

Relative adverbs are not always correctly transformed into LF and sometimes they are erased.

In order to automatically check for this type of mistake, I modified the previous system[42] and I searched for missing relative adverbs in the LFs. The results of the analysis are shown in Table 4.3 (next page).

As we already know, definitions of nouns are longer and much more complex than those of other POS. Relative clauses and relative adverbs are more frequent in the noun file and the most part of errors has been detected here.

[42] In Appendix 6 the system employed for searching missing relative adverbs.

POS file	LFs examined	LFs with missing rel advs
noun	79.689	1.252
verb	13.507	12
adjective	18.561	25
adverb	3.664	0
TOTAL	115.421	1.289

Tab 4.3 Cases of Missing Relative Adverbs

4.2.5 Pos Tagging Errors

Another frequent type of error resulting from the analysis concerns the wrong pos label assigned to predicates of LFs. Sometimes tagging errors are just the result of a wrong syntactic parsing, see the following example of *Hebrew:*

```
<gloss pos="NOUN" synsetID="10304135">
  <synonymSet>Habakkuk</synonymSet>
 <text> a Hebrew minor prophet</text>
  <wsd>
      <wf pos="DT" >a</wf>
      <wf pos="NNP" lemma="hebrew" >hebrew</wf>
      <wf pos="JJ" lemma="minor" quality="normal" wnsn="1"
>minor</wf>
     <wf pos="NN" lemma="prophet" quality="normal" wnsn="2"
>prophet</wf>
  </wsd>
<parse quality="SILVER">
(TOP (S (NP (NN Habakkuk) )
        (VP (VBZ is)
            (NP (DT a) (NNP Hebrew) (JJ minor) (NN prophet) ) )
        (. .) ) )
</parse>
```

```
 <lft quality="SILVER">
  habakkuk:NN(x1) -> hebrew:NN(x1) minor:JJ(x1) prophet:NN(x2)
 </lft>
</gloss>
```

but errors of this type doesn't always result from mistakes in the syntactic parsing; as we can see in the following example, the adjective *English* has been correctly tagged in the parse but it results with a wrong pos in the LF:

```
<gloss pos="NOUN" synsetID="02319609">
  <synonymSet>Durham, shorthorn</synonymSet>
 <text> English breed of short-horned cattle </text>
  <wsd>
     <wf pos="JJ" lemma="english" quality="normal" wnsn="1" >english</
wf>
      <wf pos="NN" lemma="breed" quality="silver" wnsn="2" >breed</wf>
      <wf pos="IN" >of</wf>
      <wf pos="JJ" lemma="short" quality="normal" wnsn="6" >short</wf>
      <punc>-</punc>
      <wf pos="JJ" lemma="horned" quality="silver" wnsn="1" >horned</
wf>
      <wf pos="NNS" lemma="cattle" quality="silver" wnsn="1" >cattle</
wf>
  </wsd>
<parse quality="NORMAL">
(TOP (S (NP (NNP Durham) )
        (VP (VBZ is)
            (NP (NP (JJ English) (NN breed) )
                (PP (IN of)
                    (NP (JJ short-horned) (NNS cattle) ) ) ) )
        (. .) ) )
</parse>
 <lft quality="NORMAL">
  durham:NN(x1) -> english:NN(x1) breed:NN(x2) of:IN(x1, x3) short-
horned:JJ(x3) cattle:NN(x3)
 </lft>
</gloss>
```

Moreover, tagging errors can occur also in gold LFs, as in the following example of *English* tagged as NN:

```
<lft quality="GOLD">
```

```
wilkes:NN(x1) -> english:NN(x1) reformer:JJ(x1) publish:VB(e1,
x1, x2) attack:NN(x2) on:IN(x2, x3) george_iii:NN(x3) and:CC(e3,
e1, e2) support:VB(e2, x1, x4) right:NN(x4) of:IN(x4, x5)
american:JJ(x5) colonist:NN(x5)
<\lft>
```

Most of the tagging errors occur for those words that can belong to different syntactic categories according to the context. This happens in XWN in particular for colours, numbers, nationality adjectives (especially when compounds), participles and gerundives.
Let's see some examples:

colour - *black and grey*:

```
jackdaw:NN(x1) -> common:JJ(x1) black-and-gray:NN(x1)
eurasian:JJ(x1) bird:NN(x1) note:VB(e1, x3, x1) for:IN(e1, x2)
thievery:NN(x2)
```

number - *ten*:

```
large_integer:NN(x1) -> integer:NN(x1) equal:JJ(x1) to:IN(x1, x3)
greater:JJ(x2) than:IN(x1, x2) ten:JJ(x2)
```

nationality adjective - *North American*:

```
solitaire:NN(x1) -> dull:JJ(x5) gray:JJ(x1) north:NN(x1)
american:NN(x2) thrush:NN(x3) note:VB(e1, x6, x1) for:IN(e1, x4)
beautiful:JJ(x4) song:NN(x4)
```

Some participles, present and past, can be used as adjectives and should be recognised and correctly transformed into LF. This often does not happen as in the definition of *chance-medley* (***unpremeditated*** *killing of a human being in self defense)*:

```
chance-medley:NN(x1) -> unpremeditated:VB(e1, x5, x1)
killing:NN(x1) of:IN(x1, x2) human:NN(x2) in:IN(x2, x3)
self:NN(x3) defense:NN(x4)
```

Furthermore, I found the wrong interpretation even for some words that end with *-ed*, like regular past participles, but that can't be participles, whatever the context is:

```
esthetician:NN(x1) -> worker:NN(x1) skilled:VB(e1, x4, x1)
in:IN(e1, e2) give:VB(e2, x1, x2) beauty:NN(x2) treatment:NN(x3)
```

Gerundives and present participles, when appearing at the beginning of a definition, are mapped into the verb base form preceded by *act of*, as in the definition of *going (**advancing** toward a goal)*:

```
going:NN(x1) -> act:NN(x1) of:IN(x1, e1) advance:VB(e1, x2, x26)
toward:IN(e1, x2) goal:NN(x2)
```

this should be avoided when the *-ing* form is used as nominalised verb as in the definition of *notepaper (**writing paper** intended for writing short notes or letters)*:

```
notepaper:NN(x1) -> act:NN(x1) of:IN(x1, e1) write:VB(e1, x2,
x2) paper:NN(x2) intend:VB(e2, x6, x2) for:IN(e2, e3)
write:VB(e3, x2, x5) short:JJ(x5) note:NN(x3) or:CC(x5, x3, x4)
letters:NN(x4)
```

The total number of tagging errors is hard to measure.
Comparing pos labels of LF predicates with those of the parse trees is not an option as we have seen that they may be both

wrong. Furthermore, there seems to be no coherence in how the tagging errors have been generated, some words may be both correctly and wrongly labeled.

In order to estimate the number of errors, I manually judged samples of LFs. For each Pos file I took 150 LFs: 50 normal, 50 silver, 50 gold for the noun file, 75 gold and 75 silver for the adjective file (no normal LFs in this file), and 150 gold for the adverb and the verb file (no normal or silver LFs in this files). I did not choose the first LFs of the files for the evaluation but I randomly selected them. This was done in order to avoid groups of particularly well or bad formed LFs. In fact, the entities in XWN files are not alphabetically ordered, and one can find groups of simple and short definitions (for e.g. definitions of cities) whose LF transformations are simpler and less error-prone than other of more complex definitions. Moreover, some groups of gold LFs can be more accurate than others due to a more careful human annotator (more than one worked on the project). The system I built for the random selection is in Appendix 7 and results of the manual evaluation are shown in Table 4.4. For the evaluation I marked as wrong those LFs with at least one wrong pos label.

POS File	normal LF		silver LF		gold LF	
	correct	wrong	correct	wrong	correct	wrong
noun	31 / 50	19 / 50	45 / 50	5 / 50	47 / 50	3 / 50
verb	X		X		148 / 150	2 / 150
adjective	X		62 / 75	13 / 75	71 / 75	4 / 75
adverb	X		X		147 / 150	3 / 150

Tab 4.4 Pos Tagging Errors

Even though I previously showed that wrong pos labels occur also in manually-checked LFs, results show that the number of errors is directly connected with the quality of LFs. Even thought the evaluated data are limited, they are enough to show us the inaccuracy of not-manually checked LFs and in particular of normal LFs for which I estimate a tagging error rate of almost 40%.

4.2.5.1 Tagging Errors in LHS

Pos tagging errors may occur even in the LHS of the LF which is quite curious if we consider that XWN is divided into pos files with rigorous LHS structures.
As we have seen in the previous chapter, LFs have specific LHS structures according to the pos category of the lemma (and therefore according to the pos file they belong to):

noun file = noun :**NN(x1)** -> definition in LF

```
chocolate_cake:NN(x1) -> cake:NN(x1) contain:VB(e1, x1, x2)
                         chocolate:NN(x2)
```

verb file = verb :**VB(e1,x1,x2)** -> definition in LF

```
bake:VB(e1, x1, x2) -> prepare:VB(e1, x1, x2) with:IN(e1, x3)
       dry:JJ(x3) heat:NN(x3) in:IN(e1, x4) oven:NN(x4)
```

adj file = adjective :**JJ(x1)** -> definition in LF

```
home-baked:JJ(x1) -> baked:JJ(x1) at:IN(x1, x2) home:NN(x2)
```

adv file = adverb :**RB(e1)** -> definiton in LF

```
dolce:RB(e1) -> gently:RB(e1) sweetly:RB(e1)
```

Therefore, for the XWN LF transformation system it should be enough to consider the pos of the file that it is processing to correctly generate the LHS of the LFs.
These parameters seem to be not enough and wrong pos labels have been detected in several LHS. See the following examples:

burn - noun as verb

```
<gloss pos="NOUN" synsetID="00386296">
  <synonymSet>burn</synonymSet>
 <text> damage inflicted by burning </text>
  <wsd>
      <wf pos="NN" lemma="damage" quality="normal" wnsn="3" >damage</
wf>
      <wf pos="VBN" lemma="inflict" quality="silver" wnsn="1"
. . .
  </wsd>
<parse quality="NORMAL">
(TOP (S (NP (NN burn) )
        (VP (VBZ is)
            (NP (NP (NN damage) )
              . . .
</parse>
 <lft quality="NORMAL">
  burn:VB(e1, x3) -> damage:NN(x1) inflict:VB(e3, x2, x1) by:IN(e3,
x2) burning:NN(x2)
 </lft>
</gloss>
```

flip - noun as adjective

```
<gloss pos="NOUN" synsetID="01176224">
  <synonymSet>flip, toss</synonymSet>
 <text> the act of flipping a coin </text>
  <wsd>
      <wf pos="DT" >the</wf>
      <wf pos="NN" lemma="act" quality="gold" wnsn="2" >act</wf>
      <wf pos="IN" >of</wf>
```

```
      <wf pos="VBG" lemma="flip" quality="normal" wnsn="4" >flipping</
wf>
      <wf pos="DT" >a</wf>
      <wf pos="NN" lemma="coin" quality="silver" wnsn="1" >coin</wf>
  </wsd>
<parse quality="NORMAL">
(TOP (S (NP (NN flip) )
        (VP (VBZ is)
            (NP (NP (DT the) (NN act) )
                (PP (IN of)
                    (S (VP (VBG flipping)
                           (NP (DT a) (NN coin) ) ) ) ) ) ) )
        (. .) ) )
</parse>
 <lft quality="NORMAL">
  flip:JJ(x3) -> act:NN(x1) of:IN(x1, e2) flip:VB(e2, x1, x2)
coin:NN(x2)
 </lft>
</gloss>
```

I found the LHS tagging error only in the noun file and I counted the mistakes for each wrong pos assigned with SystemF in Appendix8. Results are shown in Table 4.5 (for e.g. the case of *flip* here above is counted as JJ in LHS, *burn* as VB in LHS).

LFs checked	VB in LHS	JJ in LHS	RB in LHS	Total mistakes in LHS
94.868	572	512	227	1311

Tab 4.5 Mistakes in LHS - Noun File

4.2.6 Possessives

As Moldovan and Novischi 2004 explain, possessives pronouns are transformed into LF with the predicate POS. This predicate reifies the relation of ownership between two entities. Number and order of arguments are fixed. See e.g. below:

```
<gloss pos="NOUN" synsetID="06193120">
  <synonymSet>topic_sentence</synonymSet>
 <text> a sentence that states the topic of its paragraph </text>
. . .
 <lft quality="GOLD"> topic_sentence:NN(x1) -> sentence:NN(x1)
state:VB(e1, x1, x2) topic:NN(x2) of:IN(x2, x3) its:POS(x3, x1)
paragraph:NN(x3) </lft>
</gloss>
```

This structure seems to be more a suggestion of the author than a real implementation because the number of possessive pronouns transformed with the POS predicate is really small in the resource and they appear only in *gold* LFs (added manually by a human annotator?). In the whole XWN I found 1249 POS predicates over 133K LFs; in fact many possessive pronouns are erased from the LF as in the following example:

```
<gloss pos="NOUN" synsetID="04287654">
  <synonymSet>tower</synonymSet>
 <text> a structure taller than its diameter; . . . </text>
. . .
 <lft quality="GOLD"> tower:NN(x1) -> structure:NN(x1) tall:JJ(x1)
than:IN(x1, x2) diameter:NN(x2) </lft>
. . .
</gloss>
```

Like possessive pronouns, also the genitive marking should be represented in LF with the POS predicate; see the following example:

```
<gloss pos="VERB" synsetID="00038132">
  <synonymSet>marcel</synonymSet>
 <text> make a marcel in a woman's hair </text>
. . .
 <lft quality="GOLD"> marcel:VB(e1, x1, x2) -> make:VB(e1, x1, x3)
marcel:NN(x3) in:IN(e1, x5) woman:NN(x4) 's:POS(x5, x4) hair:NN(x5)
 </lft>
</gloss>
```

However, in the noun file several genitive markings are not identified and they are interpreted in different ways both as part of the word they standby and as a single element. See the following examples:

```
<gloss pos="NOUN" synsetID="08026917">
  <synonymSet>culmination</synonymSet>
 <text> (astronomy) a heavenly body's highest celestial point
above an observer's horizon </text>
. . .
 <lft quality="NORMAL"> culmination:NN(x1) -> heavenly:JJ(x1)
body's:NN(x1) highest:JJ(x2) celestial:JJ(x2) point:NN(x2)
above:IN(e0, x2) observer's:NN(x3) horizon:NN(x4)
 </lft>
</gloss>
```

```
<gloss pos="NOUN" synsetID="00157666">
  <synonymSet>capture</synonymSet>
 <text> the removal of an opponent's piece from the chess board
 </text>
. . .
  <lft quality="NORMAL"> capture:NN(x1) -> removal:NN(x1)
of:IN(x1, x2) opponent:NN(x2) 's:VB(e1, x2, x3) piece:NN(x3)
from:IN(x3, x4) chess:NN(x4) board:NN(x5)
 </lft>
</gloss>
```

In summary, the problem of possessives regards those possessive pronouns which have not been transformed into LF and those genitive markings which have not been identified as such.
In order to estimate the problem of possessives in XWN, I built two systems: SystemG in Appendix9 compare each definition with its LF looking for missing possessive pronouns, SystemH in

Appendix10 finds those genitive markings which have been wrongly transformed into LF. Results are shown in Table 4.6 and in Table 4.7 respectively.
The numbers of LFs inspected by the systems differ because SystemG checks definition-LF pairs and to do so it considers only the first definition and first LF of each gloss; conversely, SystemH analyses all the LFs of XWN.
As results show, possessive and genitive markings mistakes affect almost exclusively noun glosses and in particular those with *normal* quality.

pos XWN file	LFs checked	LFs with missing possessive pronouns
noun	79689	1900
verb	13507	39
adjective	18561	82
adverb	3664	1

Tab 4.6 Missing Possessive Pronouns

pos XWN file	LFs checked	LFs with genitive marking error
noun	94868	415
verb	14463	0
adjective	20377	0
adverb	3994	0

Tab 4.7 Genitive Marking Mistakes

4.2.7 Negation

The last notable type of error I found during the analysis of XWN LFs regards negations.
Unlikely what is said in Rus 2002/2 and reported in section 3.2.3, negations are transformed in LF and, apart from *nothing* and *none*, they are turned into RB - adverbial predicates. See for e.g.:

```
inactive:JJ(x1) -> not:RB(x1) engaged:JJ(x1) in:IN(x1, x2) full-
time:JJ(x2) work:NN(x2)
```

Negations are quite frequent in the resource and considering different negation marks (*not*, *nor*, *no*, *none*, *nothing*, *never*) I counted more than 3200 occurrences in XWN definitions, most of which in the adjective file[43]; verb and adverb files have very few negations in their definitions. See Table 4.8 for the exact number of negations.

	Noun file	Verb file	Adjective file	Adverb file
Negations	1040	62	2074	82

Tab 4.8 Negations in XWN Definitions

The problem this time doesn't concern with the lack of negation markers[44] in the LF but rather with how negation are scoped.
Negation can receive different scope according to its semantic role:

[43] I built SystemH in Appendix 10 for counting the number of negation marks in XWN definitions.

[44] I searched for missing negation markers in the LFs with SystemI in Appendix11 and the few cases I found are irrelevant.

- wide scope when it negates the main verb or modifiers of the verb like adverbials
- narrow scope when it negates specific arguments or adjuncts

As pointed out by Delmonte and Rotondi 2012, most of the cases of narrow scope in XWN LFs (1095 vs 901 of wide scope) are in the adjective file and they are all correctly marked, see for e.g.:

```
absolute:JJ(x1) -> not:RB(x1) limited:JJ(x1) by:IN(x1, x2)
law:NN(x2)
```

An example of correctly marked wide scope is the following, where the negation has wide scope on the coordination of two verbs:

```
alien:JJ(x1) -> not:RB(e3) contain:VB(e1, x7, x1) in:IN(e1, x5)
or:CC(e3, e1, e2) derive:VB(e2, x1) from:IN(e2, x2)
essential:JJ(x2) nature:NN(x2) of:IN(x2, x3) something:NN(x3)
```

Most errors occur when the negation is wrongly attached to an auxiliary verb (*be*, *do*, *have*). In these cases the LF transformation creates two event variables, one for the auxiliary and one for the main verb; and then the negation is assigned narrow scope over the event variable of the auxiliary. See for e.g.:

```
absentee_rate:NN(x1) -> percentage:NN(x1) of:IN(x1, x2)
worker:NN(x2) do:VB(e1, x2, e2) not:RB(e1) report:VB(e2, x2,
x26) to:IN(e2, e3) work:VB(e3, x2, x26)
```

In several cases the scope is marked correctly on the main verb as in:

```
lowbrow:JJ(x1) -> characteristic:JJ(x2) of:IN(x1, x2)
person:NN(x2) be:VB(e1, x2) not:RB(x5) cultivated:JJ(x5)
or:CC(e4, e1, e2) do:VB(e2, x2, e3) not:RB(e3) have:VB(e3, x2,
x3) intellectual:JJ(x3) taste:NN(x3)
```

Furthermore, sometimes negations are associated to a free variable, as shown here below:

```
acquit:VB(e1, x1, x2) -> pronounce:VB(e1, x1, x3) not:RB(e2)
guilty:JJ(x3) of:IN(x3, x4) criminal:JJ(x4) charge:NN(x4)
```

Because of the nature of this kind of error, it is not easy to automatically estimate the number of wrongly scoped negation markings in the whole resource. In fact, this is not a problem of missing particles for which a comparison between definitions and LFs is enough to identify the mistakes.

4.3 Errors in other LF Resources

I previously clarified the reasons why I chose to work on XWN rather than on other resources that transform the glosses of WN in LFs (ILF and WN30-lfs). After seeing the several errors that affect XWN, one might fairly ask if these errors affect also the other resources or if the LFs of XWN are particularly incorrect. The answer is straightforward: similar errors occur in other resources (and in particular in WN30-lfs); proving in this way the difficulty of automatically transform sentences, more or less complex, in LFs.

The problem of free-variables is even more serious in WN30-lfs than in XWN (see again Agerri and Peñas 2010 about the topic of free-variables), while in ILF the problem is solved mainly thanks to the simple *pretty ilf* syntax which, however, is probably too simple since it excludes even pos of words.
NCs are most of the time identified in ILF but then sometimes wrongly mapped in LF, as in the following example where the *nn-predicates* don't exactly clarify the relations among the three nouns:

```
<text>The professional relation between a health care
professional and a patient.</text>
<pretty-ilf>the(x1) professional(x2) relation(x3) det(x3,x1)
amod(x3,x2) prep_between(x3,x8) prep_between(x3,x11) a(x5)
health(x6) care(x7) professional(x8) det(x8,x5) nn(x8,x6)
nn(x8,x7) conj_and(x8,x11) a(x10) patient(x11) det(x11,x10)</
pretty-ilf>
```

In WN30-lfs the problem of NCs is often solved by erasing part of the compound:

```
<gloss>of materials from waste products</gloss>
<lf>reclaim#v#2'(e0,x0) -> of'( e2 , x0 , x1 ) + material'( e3 ,
x1 ) + dset( s0 , x1 , e3 ) + from'( e1 , x0 , x2 ) +
waste#n#1'( e4 , x2 )</lf>
```

WN30-lfs identifies possessives but, often, it maps them wrongly in the LFs. As shown here below, the arguments of the possessive predicate don't support the relation between *tower* and *diameter*:

```
<gloss>a structure taller than its diameter; can stand alone or
be attached to a larger building</gloss
<lf>tower#n#1'(e0,x0) -> structure'( e1 , x1 ) + tall'( e2 ,
x2 ) + poss( s4 , x4 ) + diameter#n#2'( e4 , x4 ) + can'( e5 ,
```

```
x7 , e12 ) + stand'( e12 , x9 , x5 ) + alone#r#2'( e7 , e6 ) +
or'( e9 , e10 , e8 ) + be'( e11 , x8 , e15 ) + attach'( e15 ,
x12 , x11 ) + to'( e13 , e11 , x13 ) + large'( e17 , x13 ) +
building'( e16 , x13 )</lf>
```

the previous example proves also that different definitions are wrongly transformed as a unique LF in WN30-lfs.
Also conjunctions predicates are wrongly mapped into LF in WN30-lfs, see for instance the following predicate of *or* and its free variables :

```
<gloss>a slice of meat cut from the fleshy part of an animal or
large fish</gloss>
<lf>steak#n#1'(e0,x0) -> slice'( e0 , x0 ) + of'( e3 , x1 , e2 )
+ meat'( e5 , x3 ) + cut#a#1'( e6 , x4 ) + from'( e7 , x4 , x5 )
+ fleshy#a#2'( e10 , x5 ) + part#n#1'( e8 , x5 ) + of'( e9 ,
x5 , x6 ) + animal'( e12 , x7 ) + or'( e14 , e15 , e13 ) +
large'( e16 , x8 ) + fish'( e17 , x9 )</lf>
```

Negations appear wrongly scoped in several WN30-lfs, see the following example where *not* is mapped to *major_league* rather than to the main verb:

```
<gloss>a league of teams that do not belong to a major league
(especially baseball)</gloss>
<lf>minor_league#n#1'(e0,x0) -> league#n#1'( e6 , x0 )+nn'( e3 ,
x0 , x1 ) + of'( e2 , x1 , x2 ) + team'( e4 , x2 ) + dset( s6 ,
x2 , e4 ) + not#r#1'( e10 , e9 ) + belong_to#v#1'( e11 , x8 , x6
) + major_league#n#1'( e12 , x9 ) + especially'( e5 , e0 ) +
baseball#n#1'( e0 , x0 )</lf>
```

ILF LFs are generally less error prone but they completely lack primary information like pos labels while keeping unnecessary parts like articles and determiners.

4.4 Conclusions

In this chapter I described the most common errors of LFs discovered with an accurate analysis of the XWN resource.
Tu sum up, XWN LFs errors concern:

- Free/unbound variables of arguments
- no identification of Nominal Compounds
- missing Conjunctions and Prepositions
- missing Relative Adverbs
- wrong pos labels of predicates both in the LHS and in the RHS of LFs
- Possessives Pronouns and Genitive Marking
- Scope of Negations

As shown, the number of some errors varies substantially from one file to the other and their distribution depends mainly on:

- the quality of LF

Even though some errors affect also manually-checked LFs (e.g. missing conjunctions and prepositions), they occur especially in *normal* LFs (in particular pos tagging errors and possessives mistakes) proving in this way the importance of a careful human supervision of LF transformation.

- the nature of the sentence transformed into LF

Free variables are most frequent in verb definitions because they are generally made of a verb without arguments. The complexity of syntactic structures doesn't influence the frequency of missing

conjunctions and prepositions: they are simply related to the number of conjunctions and prepositions in the definitions (and therefore they affect especially adjective LFs). Relative adverbs are most frequent in complex definitions of nouns and this is why their lack affects especially the noun file.

Reflecting on how to improve the LFs of XWN, different ideas came out.
One solution could be refining the syntactic parsing. As we have seen, LFs are directly derived from the parse trees, and even thought the high accuracy achieved in this first phase, it might be further improved. However, we have seen also that exact parse trees are sometimes wrongly transformed into LFs and this lead me to the conclusion that a perfect parse tree is not enough to prevent errors in the LF.
A second idea was to rethink the LF transformation system. This is a costly operation and considering the work done so far by the different authors who worked on the project I eventually decided to work directly on the already existing LFs by semi automatically correcting the detected mistakes described in this chapter.

In Chapter5 I will describe the correction of XWN LFs which we will see it will bring to a new LF resource: the UXWN (United eXtended WordNet).

Chapter 5

Errors Correction

5.1 Introduction

In this chapter I will describe the work carried out for the correction of XWN LFs. Thanks to a in house parser, I corrected the high number of free variables of the resource and the work is organised following what I did before and after the application of the Parser.
During the correction, some interesting considerations came out, in particular regarding Instances vs Classes, Proper Names and Nominal Compounds.

5.2 Conjunctions and Prepositions - Correction

As previously shown in section 4.2.3, conjunctions and prepositions are sometimes erased from LFs.
I already mentioned the system[45] I built to search for missing conjunctions in the resource. The system compares each definition with its LF, both divided into tokens, and records in a txt file those synsetIDs for which a missing conjunction is found in the LF.
I then used the list of synsetIDs to manually check and correct the LFs in the resource when needed.
For e.g. the synsetID 07187330 is on the list. It corresponds to the gloss of *steak*:

```
<gloss pos="NOUN" synsetID="07187330">
  <synonymSet>steak</synonymSet>
 <text>
   a slice of meat cut from the fleshy part of an animal or large fish
 </text>
. . .
 <lft quality="NORMAL">
  steak:NN(x1) -> slice:NN(x1) of:IN(x1, x2) meat:NN(x2) cut:VB(e1,
x5, x1) from:IN(e1, x3) fleshy:JJ(x3) part:NN(x3) of:IN(x3, x4)
animal:JJ(x4) large:JJ(x4) fish:NN(x4)
 </lft>
</gloss>
```

Here we can see that *or* is missing in the LF and maybe because of the wrong pos label of *animal*, which is wrongly considered as adjective. I corrected the LF by adding the *or* predicate, its corresponding arguments and by changing the pos label of animal:

[45] System C in Appendix 5

```
steak:NN(x1) -> slice:NN(x1) of:IN(x1, x2) meat:NN(x2) cut:VB(e1, x5,
x1) from:IN(e1, x3) fleshy:JJ(x3) part:NN(x3) of:IN(x3, x4)
animal:NN(x5) or:IN(x4, x5, x6) large:JJ(x6) fish:NN(x6)
```

I manually checked all the synsetIDs of missing conjunctions retrieved by the SystemC (more than 10K) but I did not correct all of them. I judged some cases adequate without the conjunctions as for example coordinations of adjectives:

```
<gloss pos="NOUN" synsetID="00016236">
  <synonymSet>object, physical_object</synonymSet>
 <text>
   a tangible and visible entity; an entity that can cast a shadow;
"it was full of rackets, balls and other objects"
 </text>
. . .
 <lft quality="GOLD">
  object:NN(x1) -> tangible:JJ(x1) visible:JJ(x1) entity:NN(x1)
 </lft>
. . .
</gloss>
```

In section 4.2.3 I explained also the problems regarding prepositions and phrasal verbs. These particles are sometimes erased from LFs and I searched for the missing cases with SystemL in Appendix12. SystemL, similar to SystemC for conjunctions, compares the first definition-LF pair of each XWN entry and if it finds a preposition in the definition that is missing in the LF it marks this case as a missing preposition. The system records on a txt file all the missing cases using the following structure:

preposition missing + SynsetID + Definition + Logical Form

so for instance, SystemL detect the missing *out* in the LF of *weed* and records this entry as:

```
2022 Missing Preposition: out
SynsetID: 12334577
Definition:
   any plant that crowds out cultivated plants
Logical Form:
 any:JJ(x1) plant:NN(x1) crowd:VB(e1, x1, e2) cultivate:VB(e2,
x3, x1) plant:NN(x2)
```

I used the output file to check the cases recorded and to judge if each case was or not a missing case. If yes I proceeded by correcting the corresponding LF. For e.g. I corrected the previous case in:

```
any:JJ(x1) plant:NN(x1) crowd_out:VB(e1, x1, x2) cultivated_JJ:
(x2) plant:NN(x2)
```

Just like for conjunctions, I did not correct all the cases detected by the system because I judged some LFs well formed even without prepositions, see for example:

```
<gloss pos="NOUN" synsetID="00081044">
  <synonymSet>award, awarding</synonymSet>
 <text>
   a grant made by a law court; "he criticized the awarding of
compensation by the court"
 </text>
. . .
 <lft quality="GOLD">
  award:NN(x1) -> grant:NN(x1) make:VB(e1, x2, x1)
law_court:NN(x2)
 </lft>
</gloss>
```

Here variables are enough to bear the relations of the passive form, the preposition can be omitted from the LF.

5.3 Possessives - Correction

As pointed out in section 4.2.6, errors regarding possessives can be divided into two categories: possessive pronouns which have not been transformed into LF and genitive markings which have not been identified as such.

5.3.1 Genitive Marking

As mentioned before, I used SystemH in Appendix10 to identify wrongly transformed genitive markings. The system investigates each XWN pos file searching for those predicates whose structure is: word+'+s+:NN/JJ/VB/RB e.g. *employee's:NN* in:

employee's:NN(x2) salary:NN(x3)

To do so, for each LF the system builds a list of tuples word+pos (predicate without arguments); for e.g. the LF of *withholding*:

```
<gloss pos="NOUN" synsetID="00344768">
  <synonymSet>withholding</synonymSet>
 <text> the act of deducting from an employee's salary </text>
. . .
```

```
 <lft quality=“NORMAL"> withholding:NN(x1) -> act:NN(x1)
of:IN(x1, e1) deducting:VB(e1, x1, x26) from:IN(e1, x2)
employee's:NN(x2) salary:NN(x3) </lft>
</gloss>
```

is transformed into:

```
[['act', 'NN'], ['of', 'IN'], ['deducting', 'VB'], ['from',
'IN'], ["employee's", 'NN'], ['salary', ‘NN']]
```

after this, SystemD searches for those words that end with *’s* and have no POS predicate as second element. The output of the system is a txt file with all these cases, each of them proceeded by the quality of the LF, for e.g.:

```
[NORMAL] withholding:NN(x1) -> act:NN(x1) of:IN(x1, e1)
deducting:VB(e1, x1, x26) from:IN(e1, x2) employee's:NN(x2)
salary:NN(x3)
```

I used the output file as starting point to manually correct the resource.
For each retrieved case, I checked whether the *’s* particle was a genitive marking or not and in the first case I proceeded by creating the corresponding POS predicate. E.g. the above case of *employee’s:NN* was corrected into:

employee:NN(x2) ’s:POS(x3,x2) salary:NN(x3)

Thanks to this procedure I found that the problem of not identified genitive markings appears only in the noun file and in particular in *normal* LFs. I checked and corrected 423 not identified genitive markings.

5.3.2 Missing Possessive Pronouns

I identified missing possessive pronouns with SystemG in Appendix9. Just like SystemH, SystemG takes as input the XML structure of XWN and for each LF it builds a list of tuples word+pos (predicate without arguments). It then checks each definition in XWN: if a possessive pronoun is found in the definition and not in the corresponding LF the missing possessive pronoun case is recorded in a txt file with the structure:

synsetID + missing possessive pronoun+ definition + LF

So, for e.g., for the already seen gloss of *tower* (see page 114), the system knows that there is a possessive pronoun in the definition

```
a structure taller than its diameter
```

that is missing in the LF

```
tower:NN(x1) -> structure:NN(x1) tall:JJ(x1) than:IN(x1, x2)
diameter:NN(x2)
```

and it records this case in the output as:

```
04287654 its
a structure taller than its diameter
tower:NN(x1) -> structure:NN(x1) tall:JJ(x1) than:IN(x1, x2)
diameter:NN(x2)
```

I then used the output of SystemG to manually check the XWN files and to correct them when needed. Sometimes I judged the

lack of a possessive pronoun passable as it doesn't change too much the meaning of the LF and I didn't do any modification, see for e.g.:

```
10145344 his
French composer best known for his operas
  bizet:NN(x1) -> french:JJ(x1) composer:NN(x1) best:RB(e1)
know:VB(e1, x3, x1) for:IN(e1, x2) opera:NN(x2)
```

I checked almost 2K cases of missing possessive pronouns most of which in the noun file.

5.4 Free Variables - The Parser

Together with Prof. Delmonte we conceived a LF Parser[46] that counts the number of disconnected variables and corrects most of them. The data regarding disconnected variables before the correction are shown in Tab 4.1, section 4.2.1, and they highlight an error rate of 54% which means that more than half of LFs contain some disconnected variables. This result is due also to the fact that we considered also dummy variables generated for those verbs that don't have all the arguments in the sentence (see section 3.2.3 and 4.2.1).

Our purpose with the Parser is not just to count and correct free variables, we aimed at something more: providing a new consistent resource of LFs.

For this reason, the Parser returns as output a new file where, for each gloss of XWN, the SynsetID, the Lemma, the Synset and the

[46]In Appendix13 the Parser code

LF(s) are joined. Each line in the output file is an entry and the whole resource provides a direct and easy access to the LFs without loosing the connection with WN and XWN (thanks to the SynsetIDs). For instance, the first entry of the verb file in our new resource is:

```
synset(200665984,about-face_VB(e1,x1,x2),[about-face])-
[change_VB(e2,x1,x3),mind_NN(x3),and_CC(e1,e2,e3),assume_VB(e3,x
1,x4),opposite_JJ(x4),viewpoint_NN(x4)]
```

SynsetID Lemma Synset LF

We named the new resource United eXtended WordNet and I will describe the Parser pipeline in the following pages.

5.4.1 Parser Pipeline

The Parser works on one XWN pos file at a time. It takes as input two files, one containing the list of LFs and another one containing the SynsetIDs following by the Synsets[47].
Each LF starts necessarily with the same lemma which corresponds to the first lemma in the Sysnet, see for e.g :

```
<gloss pos="VERB" synsetID="00628816">
    <synonymSet>distinguish, separate, differentiate, secern,
secernate, severalize, severalise, tell, tell apart</synonymSet>
 <text>
   mark as different; "We distinguish several kinds of maple"
 </text>
. . .
 <lft quality="GOLD">
```

[47] Excerpts of the input files for the noun file are shown in Appendix 14 e 15.

```
 distinguish:VB(e1, x1, x2) -> mark:VB(e1, x1, x2) as:IN(e1, x3)
different:JJ(x3)
 </lft>
</gloss>
```

and since each gloss can contain one or more definitions, it can contain also one or more LFs, see for e.g.:

```
<gloss pos="VERB" synsetID="02587500">
  <synonymSet>clash, jar, collide</synonymSet>
 <text>
    be incompatible; be or come into conflict; "These colors
clash"
 </text>
. . .
 <lft quality="GOLD">
 clash:VB(e1, x1, x2) -> be:VB(e1, x1, x3) incompatible:JJ(x3)
 </lft>
 <lft quality="GOLD">
  clash:VB(e1, x1, x2) -> be:VB(e2, x1, x4) in:IN(e2, x3)
or:CC(e1, e2, e3) come:VB(e3, x1, x5) into:IN(e3, x3)
conflict:NN(x3)
 </lft>
</gloss>
```

There may be one or more LFs associated to the same SynsetID and the parser takes care of this during the process.

To build the UXWN, the Parser has to: connect each SynsetID+Synset with the corresponding LF(s) and detect and correct the free variables of each LF.

As to the first point, the connection is done by matching the first lemma in the Synset with the lemma heading the LF. After the correction of a matched LF, the Parser checks the rest of the LF input file (see Appendix15) looking for another occurrence of the current lemma and in that case it keeps the same

SysnetID+Sysnet. Therefore, the above gloss of *clash* is transformed in UXWN as two entries, one for each LF, with same SysnetID, Lemma and Synset and different LF:

```
synset(202587500,clash_VB(e1,x1,x2),[clash,jar,collide])-
[be_VB(e1,x1,x3),incompatible_JJ(x3)]

synset(202587500,clash_VB(e1,x1,x2),[clash,jar,collide])-[be_VB-
[e2,x1],in_IN-[e2,x3],or_CC-[e1,e2,e3],come_VB-[e3,x1],into_IN-
[e3,x3],conflict_NN(x3)])
```

As we can see, free variables of *be* and *come* have been removed. Moreover, we added a number in front of the SynsetID (here number 2) which represents the pos of the lemma. We assigned : 1 to nouns, 2 to verbs, 3 to adjectives and 4 to adverbs. In this way the pos of each lemma can be smoothly identified by automatic systems even if an entry is treated individually (without knowing the file to which it belongs).
We decided to remove dummy arguments of verb predicates to give more consistency to the LF. We came to this conclusion following Copestake 2009 who suggests that missing arguments should be treated appropriately in LF.

The Parser is divided into two modules: the first one tries to match variables in predicates with their object counterpart, the second module does the opposite: it tries to match variables in object formulas with their predicate counterparts.
The Parser takes into account different logical structures, in particular:

- predicate argument structures governed by verbs, where (as we have already seen) number and order of argument are fixed, and the event variable might or might not have higher level binders or meta level formula, e.g. *close:VB(e1,x1,x2);*
- prepositions, conjunctions and other similar two place relation markers (e.g. *of:IN(x2,x3)*), where variables bind objects, prepositions (when introducing gerundives, e.g. *by:IN(e1,e2)*) and conjunctions treated as relation markers (e.g. *since:IN(e4,e5)*);
- object formulae, which include simple one place predication with just one variable associated to an entity, a property or an attribute (e.g. *bag:NN(x3)*). The same specification is used also for adjectives and adverbs (e.g. *happy:JJ(x1)*, *especially:RB(e1)*);
- meta-level formulae: coordinating conjunctions, which allow to refer to sets of objects or predicates (e.g. *and:CC(x4,x1,x2,x3)*, *or:IN(x7,x4,x5,x6)*) and complex nominal compounds with the *nn-predicate* (e.g. *nn(x7, x3, x4) truck:NN(x3) trailer:NN(x4)*).

Meta level formulae and negations are transformed by reification in order to simplify the matching procedures. Coordinations are turned into one place predicates e.g.:

and_CC(x6,x1,x2,x3) -> and_CC(xc), coord(xc,x6), coord(xc,x1), coord(xc,x2), coord(xc,x3)

while negations are reified:

not_RB(e2) -> neg(xn,e2),not(xn)

We eliminated all *do* auxiliaries and associated negation predicate directly to the main verb variable in order to eliminate unwanted auxiliary information and to associate the negation operator to the main verb meaning. A similar procedure was followed for *cannot*, which in XWN if often transformed without decomposition and wrongly tagged as noun, as in:

```
insomniac:NN(x1) -> someone:NN(x1) cannot:NN(x2) sleep:VB(e1,
x2, x26)
```

In this way, part of the wrongly scoped negations has been automatically fixed by the Parser.
The Parser provides two types of correction. The first correction is made following lexical information and is addressed to all those predicates that contain a dummy variable for an argument which does not exist in reality. This was done considering unergative verbs, unaccusative verbs, impersonal and weather verbs, verbs which induce intransitive structures and cases of verbs which allow the object to be left unexpressed. The Parser checks also for passivised past participles in order to correct deep subjects omissions. For e.g. the Parser automatically removes the dummy argument *x26* in the LF of *growth*:

```
<gloss pos="NOUN" synsetID="12726302">
  <synonymSet>growth, growing, maturation, development,
ontogeny, ontogenesis</synonymSet>
 <text> (biology) the process of an individual organism growing
organically; . . .  </text>
. . .
```

```
 <lft quality="NORMAL"> growth:NN(x1) -> process:NN(x1)
of:IN(x1, x2) individual:JJ(x2) organism:NN(x2) grow:VB(e1, x2,
x26) organically:RB(e1) </lft>
 . . .
</gloss>
```

which results in UXWN as:

```
synset(112726302, growth_NN(x1), [growth, growing, maturation,
development, ontogeny, ontogenesis])-[process_NN(x1), of_IN(x1,
x2), individual_JJ(x2), organism_NN(x2), grow_VB-[e1, x2],
organically_RB(e1)]
```

The second type of correction is done considering structural information. The Parser collects variables related to object formula separately from those related to predicate formula and then it does a simple intersection. Intersections are used to find ungrounded variables. In case no intersection intervenes, the output is marked with the label *no intersection* which is used by the correction module. This is the case of the following example, where the intersection of relevant variables is empty:

```
INPUT
lf(approved_JJ(x1),
[generally_RB(e1),especially_RB(e1),officially_RB(e1),judge_VB(e
1,x5,x1),acceptable_JJ(x3),satisfactory_JJ(x3)]

OUTPUT - correction module
approved_JJ(x1) 6 [x5,x3] no intersection
lf(approved_JJ(x1),
[generally_RB(e2,e1),especially_RB(e3,e1),officially_RB( e4,e1),
judge_VB(e1,e5,u,x1),acceptable_JJ(e6,e5),satisfactory_JJ(e7,e5)
]).
```

We used the two procedures in a sequence – at first we found ungrounded variables and then looked for predicates with unneeded variables that coincided with the ones found in the previous procedures – and eliminated them.
The Parser succeeded in automatically correcting most of the LFs for which some inconsistency was found, reducing the error rate form 56% to 24%, see Table 5.1.
I worked then by manually correcting those remaining LFs marked with *no intersection* by the Parser (more than 5K LFs) reducing in this way the the number of errors to the reasonable percentage of 15%.

POS FIle	LFs with Disconnected Variables - After	Total Number of LFs	% Before	% After
Noun	15122	87819	56.00	30.75
Verb	513	13373	67.46	5.69
Adjective	8895	20337	43.74	7.87
Adverb	702	3922	12.23	6.28
TOTAL	25232	125451	54.05	23.82

Tab 5.1 Disconnected Variables after Parser Correction

From this point forward, further corrections are made directly on the output of the Parser (UXWN) and not on XWN.

5.5 The Case of Proper Names

The Parser works smoothly with three of the four XWN pos files: Adjectives, Adverbs and Verbs, but when it started the file containing the Nouns a problem arose. WN, and therefore XWN, contains definitions for some 18K lemmata starting with a capital letter which can be computed as proper names or named entities, i.e. person names, organisation names, famous events names, institution names, location names etc. This particular entities need to be represented univocally since they denote single referents in the world and not classes of individuals like common nouns. Each entry in WN that is a proper name or a named entity should therefore has a Synset that univocally identifies it. For instance if we are referring to *Johannes Gutenberg*, the inventor of the printing press, his Synset in WN must contain those elements that univocally identify him, such as: *Johannes_Gutenberg* or *Johann_Gutenberg* or *J_Gutenberg* or simply *Gutenberg* since without any additional specification his surname is universally associated to the inventor, but it can't contain for e.g. only *Johannes*. *Johannes* is a generic proper name that doesn't identify any person in particular and certainly not *Johannes Gutenberg*.

While working with the parser, it came out that proper names in WN are wrongly treated like common nouns and this required a further effort of correction.
In the next sections I will take a closer look to the problem of proper names.

5.5.1 Classes and Instances

Talking about proper names it is crucial to firstly clarify the difference between classes and instances.
Let's start with an example:

a) Women are numerous
b) Rosa Parks is numerous

A doesn't mean that every particular woman is numerous but it rather intends that it exists a class of women which contains several instances.
B, on the other hand, results as a non sense. This is because women denotes a class whereas Rosa Park is an instance of this class. Some nouns are conceived as classes and the membership in these classes denotes the relations of hyponymy which are the base of the WN hierarchy of nouns.
In the first versions of WN this distinction was not considered and both concepts were drawn with the *is-a* relation and therefore encoded in the same way:

- Rosa Park is a woman
- A heroine is a women

Ontologists highlighted the need of representing this important distinction between classes and instances. In particular Gangemi et al. 2001 and Oltramani et al. 2002 complain about the lack of two different representations for individuals and concepts in WN.

In his work Oltramani points out the confusion between concepts and individuals as one of the critical point of the early WN versions. This problem is considered to be the result of an expressivity lack which impedes the distinction between concept-to-concept relations (subsumption) and individual-to-concept ones (instantiation). Examples of this deficit are the hyponyms of *composer* which consist of classes such as *contrapuntist* or *songwriter* as well as of individuals (e.g. *Beethoven*). Under *organisation* there are the conceptual hyponyms *alliance*, *federation*, *company*, together with instances like *Irish_Republican_Army*, *Red Cross*, and so on, without any particular distinction.

Miller and Hristea 2006 propose their approach to fulfil the gap. They start from considering the features of instances: they are nouns, precisely proper names (therefore they should be capitalised) and most important, they should refer to a unique entity.

The identification is easy for cases such as persons or cities but unfortunately the aforementioned properties don't identify univocally instances because they are shared by other no-instance words. For this reason the authors have to proceed with a manual inspection of the candidates nouns.

In fact, there are some particular terms which can be considered both as instances and as classes. This is the case of *Beethoven*:

c) Beethoven was born in Germany, on December 16, 1770 - class

d) She loved to listen to Beethoven - instance

In *c* it is clear that Beethoven has a unique referent, the german composer, while in *d* it is used to refer to the composer's music.

When a word has two distinct referents they are both tagged as instances by the authors. For example, *Bethlehem* in Pennsylvania and the one near Jerusalem.
Furthermore, when an instance has more than one hypernym it is labelled as an instance of all of them: *Venus* is an instance of *terrestrial planet* ("*a planet having a compact rocky surface like the Earth's*") and also of *inferior planet* ("*any of the planets whose orbit lies inside the earth's orbit*"). But identifying instances is not always a straightforward task. There might be conflicts based on subjective interpretation of words. Miller and Hristea report their difficulties in classifying for example geographical regions that do not have well-defined political boundaries (such as the *Antarctic Zone* or *Barbary Coast*), or sacred texts especially the *Christian Bible* which can be treated as an instance or as a class with several hyponyms (e.g. *American Revised Version*, *Douay*, *Vulgate* etc.).
Miller and Hristea tagged more than 7,000 words using the annotation @ vs @i. Where @ means class as in: {*peach,drupe,*@}, *a peach is a drupe* or *all peaches are drupes*, and @i means instance: {*Berlin,city,*@i}, Berlin is an instance of a city.
Miller and Hristea's classification has been included in WN starting from version 2.1, so unfortunately XWN doesn't hold it.

5.5.2 Proper Names

But what is a proper name? Is there any difference between proper names and proper nouns?

Huddleston in his outline of the English Grammar (Payne and Huddleston 2002) asserts that there are two reasons to define a distinction between proper names and proper nouns:

« a) Although a proper name may have the form of a proper noun, as in the case of *John* or *London*, it need not have. Thus *The Open University* is a proper name but not a proper noun: what distinguishes it from, say, *the older university* is precisely that it is the official name of a particular institution.

b) Proper nouns do not always function as the head of NPs serving as proper names. Thus in *They weren't talking about the same Jones* the proper noun *Jones* is head of the NP *the same Jones* but this is clearly not a proper name. Similarly in *He likes to think of himself as another Einstein*, *The Smiths are coming round tonight* etc. »

The distinction is not universally assumed (see Chalker 1992) and the Huddlestonian theory has been criticised by Anderson 2007 who considers it vague and unnecessary. The latter affirms the distinction is acceptable only if proper names cannot be considered as complex proper nouns.

I won't follow the splitting theory and I will use the two terms as synonyms and I will refer to them using PN. The no-distinction option is already employed for example by Toral et al. 2008.

Following the idea of instance, what it is important here is the concept of a term which refers to a unique entity. In the following mentioned lexical resources the encoded PNs are words or compound words which denote singular individuals (and places,

organisations etc.). Thus, for example, *Jean Baptiste Poquelin* is a PN as much as *Molière* and they both refer to the same entity: *the french author of the famous theatrical comedy Tartuffe*.

The category of PNs was often ignored or neglected in the first e-dictionaries and it is still roughly encoded in some resources.
Identifying PNs is essential considering that they occur very frequently in natural language and constitute a significant part of many texts. For instance, they represent a considerable part of unknown words in a corpus and more than 10% of newspaper texts. A resource which includes PNs has to cope with the richness of their forms and variants and with their temporary nature. Depending on the task, old PNs may not be useful anymore while new PNs should be constantly included in the resource. PNs are the fastest growing syntactic category and each PN can have spelling variations. For example the name *Gaddhafi* is a good example of a name with multiple variants, see: *Qaddhafi*, *Qaddafi*, *Gaddafi*, *Kaddafi*, *Khadafy*, *Qadhafi*, *Qadaffi* and *Gadaffi*. This richness could be not only a matter of spelling variation. One can refer to the same individual with his whole name, or just with a part of it, or adding a title e.g. *Jefferson*, *Thomas Jefferson*, *President Jefferson* or *Jesus*, *Jesus of Nazareth*, *the Nazarene*, *Jesus Christ*, *Christ*, *Savior*, *Saviour*, *Good Shepherd*, *Redeemer*, *Deliverer* (from WN).
This becomes much more complex if we think about PNs in a multilingual environment; for instance, the name of the pope John Paul II changes in Italian - *Giovanni Paolo II*, in French - *Jean Paul II*, in Croatian - *Ivan Pavao II*, in Spanish - *Juan Pablo II* etc. Furthermore, PNs may have common abbreviations (e.g *Edw*. for *Edward* or *Eliz*. for *Elizabeth*).

5.5.3 Proper Names - Resources

Despite the huge amount of information encoded in WN, there has not been made any special effort to include PNs and only 1% of the almost 80000 nominal entries in WN consists of PNs.
This fact makes WN inadequate for treating texts with PNs and points out the need of the creation of specific resources of PNs or the enrichment of generic lexical resources with PNs.
Both approaches have been developed and there exist several works regarding the creation of specific resources for PNs and the enrichment of generic resources with them.

Most of the new resources of PNs are multilingual. See for instance: the Multilingual Ontology of Proper Names (Krstev er al. 2005), the Multilingual Onomasticon (Sheremetyeva 1998), ProlexBase 2 (Maurel 2008) and its extension ProlexFeeder (Savary et al. 2013), and GeoNames for toponyms. See also The Fine-Grained Proper Noun Ontology (Mann 2002) for an example of their applicability to QA tasks. This kind of resources is structured, organised in hierarchies and sometimes mapped to WN synsets and/or to Wikipedia entries. More advanced projects aim to automatically solve the problem of the fast growing feature of PNs. This is the case of JRC-Names, a daily updated resource containing multi-word entities, their acronyms and their variants. The JRC researchers exploit the amount of data collected by the European Media Monitor (EMM)[48] which automatically gathers and analyses up to 220,000 news articles per day in about 70 different languages from up to 7,000 news sites (Steinberger et al. 2009). Named Entity

[48] http://emm.newsbrief.eu/NewsBrief/clusteredition/it/latest.html

Recognition and classification of entity types (person, organisation, location) are performed on this data for 21 languages. JRC-Names is a good example of a solid resource of multilingual open-domain acronyms and abbreviations of PNs. See more at Ehrmann et al. 2017 and Jacquet et al. 2016.

5.5.3.1 WN PNs Extensions

Regarding the enrichment of generic resources with PNs, and in particular the enrichment of WN, there are some interesting projects which aim to cope with the problem: Proper Noun Thesaurus (De Loupy et al. 2004), Named Entity WordNet (Toral et al. 2008) and the linkage of a gazetteer to WN made by Sundheim (Sundheim et al. 2006).

With Named Entity WordNet Total and colleagues attempt to extend WN with PNs automatically extracted from one of the most widely exploited source of dynamic and structured information: Wikipedia. They are aware of the importance of employing solid PN resources in several NLP tasks and they admit the inadequacy of WN on this subject. As mentioned before, WN distinguishes between common nouns (classes) and instances (PN) only from version 2.1, where they constitute less than 8 thousand entries (synsets). This shortage is closely related to the difficulty of creating a lexical resource for a class of words which grows rapidly, i.e. PNs. At the start, WN was created manually and it is almost unfeasible or definitely expensive to manually populate LRs with PNs. Toral's approach essentially consists of two phases: in the first one the synsets-classes of WN are linked to Wikipedia categories, subsequently the entries of

Wikipedia are checked in order to identify PNs. The identified PNs are encoded as new Synsets and linked to the input Synsets with *instance of* relations. The identification of PNs is done following different capitalisation norms, limiting in this way the approach to only those languages which share such norms (in this case: Catalan, Dutch, French, Italian, Norwegian, Portuguese, Romanian, Spanish and Swedish). Some statistical evaluations are made to exclude unusual capitalisation in order to evaluate as PNs only those terms which occur capitalised more than 91% of the times. To avoid mis-classification of nouns which can be both classes and instances, entry occurrences are examined in the body article of the entry and not in the Web which may be often irregular. On this way, for e.g. the entry *Chldren's Machine* (name for a particular model of laptop) may occur in a sentence where *children* and *machine* refer to common nouns. Looking for this entry only in the body of its Wikipedia article avoids mistakes. In order to work on a bigger portion of text and achieve more reliable results, for each Wikipedia article the research of PNs has been spread on the different languages which follow the same capitalisation norms. Moreover, Toral and colleagues do not consider only monosemous words from WN and for each polysemous word mapped to a Wiki-category they consider the instances for each of its senses. For example the word *Obelisk* has two senses in WN: 1) *stone pillar* and 2) *character used in printing*. *Obelisk* is mapped to the Wiki-category *Obelisks* where they found the instance *Washington Monument*. In this case, the disambiguation is straightforward and the sense chosen is the first one. Named Entity WordNet contains more than 310 thousand PN and almost 400 thousand *instance of* relation and is publicly

available. It represents an interesting and valid answer to the shortage of PNs in WN.

5.5.4 PNs in WN

The previous considerations and examples should have cleared enough the concept of instance and the need of well structured resources which have to take into account the distinction between classes and instances. The matter of correctly including PNs in the resources, and in particular in WN, has been widely studied and it is an indispensable point to consider while building new resources.
XWN doesn't include terminology extensions of PNs like Named Entity WN, and this is something I won't address here, but the previous considerations made me clear the idea that the new resource UXWN needs to correctly store instances and in particular names of people, which are about 4K in XWN.

The reflexion about PNs comes out from the analysis of XWN and the discovery of its inadequacy about this topic. It is true that the distinction between instances and classes was introduced only from WN 2.1, and therefore it is missing in XWN which is based on WN 2.0, but this distinction didn't solve the problem of how Sysnets of PNs are wrongly structured; ergo this is a problem that affects previous and later versions of WN as well as XWN independently from the version of WN which it is based on.
To understand the problem one needs to take a closer look to how PNs are encoded in WN. I have already said that instances are included in WN starting from the version 2.1, but how? If one

searches the database for a proper name the result will clearly show the tag *instance*, as in:

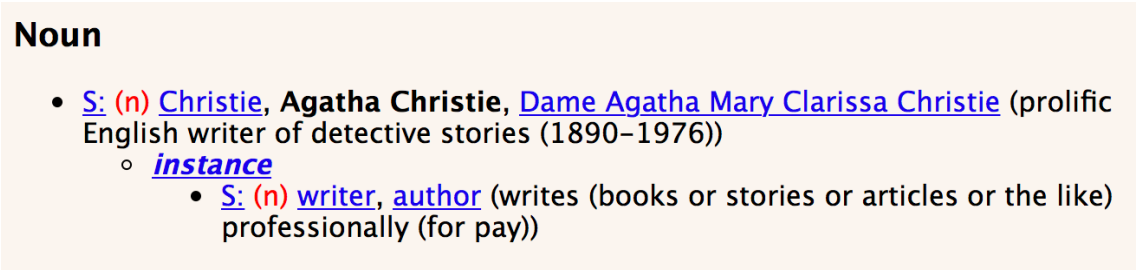


Fig 5.1 Example of WN Output

but the problem still remains in how the Synsets of instances are structured.

Contrary to expectations, the structure for this kind of entities in the WN database resembles the one used for common nouns, which, as we know, are used to denote classes of individuals.

Each PN needs a structure that can accurately identify its referent. PNs are used to individuate uniquely a single referent in the world - they are rigid designators according to Kripke (Kripke 1980). This means that each PN designates the same object in all possible worlds in which that object exists and never designates anything else.

Let's consider Synsets: Synsets are a collection or set of synonym lemmata which may constitute a single concept in a specific language. Lexicons of different languages may vary a lot and a Synset made of several words for one concept, translated in another language, may turn up to be uniquely denoted by one single lemma.

The first lemma of a Synset or the only present lemma (in case the Synset is a singleton), represents the Synset and in case of polysemous common nouns it can appear as head of different

Synsets, marked with different SynsetIDs, as for instance *plant*, which is associated to the following four SynsetIDs:

```
00014510        plant, flora, plant_life
03806817        plant, works, industrial_plant plant
05562308        plant
09760967        plant
```

From a lexicographic point of view these four entries instantiate totally different senses and are associated with different glosses:

```
00014510    a living organism lacking the power of
       locomotion
03806817    buildings for carrying on industrial labor
05562308    something planted secretly for discovery by
       another
09760967    an actor situated in the audience whose acting
       is rehearsed but seems spontaneous to the
  audience
```

As can be seen, the **offset indices** are very far from one another, thus indicating the distance in meaning involved in each of the different lemma forms. It would be incorrect to have same lemmata in adjacency within the same semantic lexical field. Polysemous words in WN are not many, and their presence in distant and different semantic lexical fields is an indication of the high frequency of usage of the word in the language.

The problem is that in WN PNs are stored like polysemous common nouns, and the first lemma of the Synset is shared by different Synsets. This may sound quite strange, seeing that the only meaning associated to a proper name is the referent which they should designate. This usually happens for person names which share the

name or the surname. As an example, here is the list of different entries associated to *John*, first lemma:

```
06043175        John, Gospel_According_to_John
10364758        John, Saint_John, St_John,
                Saint_John_the_Apostle,
                St_John_the_Apostle,
                John_the_Evangelist,
                John_the_Divine
10365110        John, King_John, John_Lackland
```

John first appear at SynsetID 06043175 but then the two following mentions appear one adjacent to the other - thus belonging to the same semantic field. Furthermore, it is well known that to identify a person usually the name is not sufficient and the title or the surname is needed. So it is obvious that first names are ambiguous but they don't have to be regarded polysemous for this reason.

In the case of *John* we are dealing with three totally different referents: the Gospel, the Apostle and the King, So why use *John* as first lemma and not one of the following more distinctive lemmata (e.g. *Saint John* or *King John*)?

This is totally misleading from a semantic point of view, because here we are not dealing with polysemous words as was the case with *plant*, but rather with referential identity. Besides, the word *John* by itself can have additional uses. Consider for instance the corresponding lower case word *john* which is used with ambiguous meanings:

```
10076833        whoremaster, whoremonger, john
04274300        toilet, lavatory, lav, can, john, privy,
                bathroom
```

Here *john* is not the first member of the Synset but the difference in meaning is clearly understood by a native speaker, and is testified again by the distance in terms of offset index values. This two examples are an additional proof to the fact that using a single name of person as referent in a Synset might be deceptive.

In WN there are only sparse cases of first names as first lemma in adjacent Synsets before reaching the section of the Noun file where all proper names are collected. Here, the choice to use a proper name or a surname as referent of the Synset becomes very common in the more restricted list of person names made up of some 4K entries that start around SynsetID 110102000. Here are some examples:

```
110102151       Aaron
110102325       Aaron, Henry_Louis_Aaron, Hank_Aaron

110105319       Agrippina, Agrippina_the_Elder
110105487       Agrippina, Agrippina_the_Younger
```

The situation is even more complex when the shared lemma is a common english surname such as *Anderson* or *Robinson*:

```
110112423       Anderson, Carl_Anderson,
                Carl_David_Anderson
110112636       Anderson, Marian_Anderson
110112784       Anderson, Maxwell_Anderson
```

```
110112893       Anderson, Philip_Anderson,
                Philip_Warren_Anderson, Phil_Anderson
110113110       Anderson, Sherwood_Anderson


110534424       Robinson, Edward_G._Robinson,
                Edward_Goldenberg_Robinson
110534598       Robinson, Edwin_Arlington_Robinson
110534737       Robinson, Jackie_Robinson,
                Jack_Roosevelt_Robinson
110534919       Robinson, James_Harvey_Robinson
110535121       Robinson, Lennox_Robinson,
                Esme_Stuart_Lennox_Robinson
110535282       Robinson, Ray_Robinson,
                Sugar_Ray_Robinson,Walker_Smith
110535526       Robinson, Robert_Robinson,
          Sir_Robert_Robinson
```

Instances are clearly treated as common nouns, when instead the first lemma of the Synset should be the distinctive trait of the entry and not something shared with others, furthermore in the same semantic field.
It is clear that this unmotivated choice of representing PNs is completely useless and needs to be reorganised.

5.5.5 PNs reorganisation

Regarding PNs, the purpose here is to provide the new resource UXWN with the correct organisation of these entities.
To build the resource, the Parser takes as input two files, one containing the list of LFs and and another one containing the

SynsetIDs following by the Synsets. For the reorganisation of PNs Synsets I worked on the input files.

5.5.5.1 PN Errors Identification

The first step in the reorganisation of PNs is identifying those PNs with different Synset IDs and **same first member of the synset**:

```
110442065      Mayer, Louis_B_Mayer, Louis_Burt_Mayer
110442275      Mayer, Marie_Goeppert_Mayer
```

To save time, I decided to parse automatically the 4K entries of person names in order to highlight the errors.
SystemM in Appendix16 takes as input the LFs file and the SynsetIDs+Sysnet file and returns the same files with the errors marked by a *.
The SynsetIDs+Sysnet file is taken as input as Synsets.txt and each entry is converted into a list. Since the file is already in alphabetic order, I can proceed comparing the entries two at time. I start with the first entry which is compared with the second one, then the second one is compared with the third one and so on… If the two entries have different Synset IDs but same first lemma they are marked with *. For example, part of the input file (see Appendix 17):

```
gloss_synsetID(110442065,
[Mayer,Louis_B_Mayer,Louis_Burt_Mayer])
gloss_synsetID(110442275,[Mayer,Marie_Goeppert_Mayer])
```

```
gloss_synsetID(110442455,
[Mays,Willie_Mays,Willie_Howard_Mays_Jr_,the_Say_Hey_Kid])
gloss_synsetID(110442607,[Mazzini,Giuseppe_Mazzini])
gloss_synsetID(110442783,
[McCarthy,Joseph_McCarthy,Joseph_Raymond_McCarthy])
gloss_synsetID(110442979,
[McCarthy,Mary_McCarthy,Mary_Therese_McCarthy])
```

has the following output (see Appendix 18):

```
*gloss_synsetID(110442065,
[Mayer,Louis_B_Mayer,Louis_Burt_Mayer])
gloss_synsetID(110442275,[Mayer,Marie_Goeppert_Mayer])
gloss_synsetID(110442455,
[Mays,Willie_Mays,Willie_Howard_Mays_Jr_,the_Say_Hey_Kid])
gloss_synsetID(110442607,[Mazzini,Giuseppe_Mazzini])
*gloss_synsetID(110442783,
[McCarthy,Joseph_McCarthy,Joseph_Raymond_McCarthy])
gloss_synsetID(110442979,
[McCarthy,Mary_McCarthy,Mary_Therese_McCarthy])
```

When an error is identified, the first lemma of the entry is stored in a list (see Appendix 19):

```
 […, mayer, mccarthy, mccormick, mead, meade, meissner,
menninger, meredith,…]
```

which is used to analyse the second input file, the LF file (see Appendix 20):

```
lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),filmmaker_NN(x3),found_VB(e1,x1,x4),
own_JJ(x4),film_NN(x4),company_NN(x5),and_CC(e3,e1,e2),later_RB(
e2),merge_VB(e2,x8,x1),with_IN(e2,x6),samuel_NN(x6),goldwyn_NN(x
7)])

lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),physicist_NN(x3),note_VB(e1,x7,x1),f
```

```
or_IN(e1,x4),research_NN(x4),on_IN(x4,x5),structure_NN(x5),of_IN
(x5,x6),atom_NN(x6)])

lf(mays_NN(x1),
[united_NN(x1),state_NN(x2),baseball_NN(x3),player_NN(x4)])
```

If the first lemma of the LF is in the list, the LF is marked with * (see Appendix 21):

```
*lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),filmmaker_NN(x3),found_VB(e1,x1,x4),
own_JJ(x4),film_NN(x4),company_NN(x5),and_CC(e3,e1,e2),later_RB(
e2),merge_VB(e2,x8,x1),with_IN(e2,x6),samuel_NN(x6),goldwyn_NN(x
7)])

*lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),physicist_NN(x3),note_VB(e1,x7,x1),f
or_IN(e1,x4),research_NN(x4),on_IN(x4,x5),structure_NN(x5),of_IN
(x5,x6),atom_NN(x6)])

lf(mays_NN(x1),
[united_NN(x1),state_NN(x2),baseball_NN(x3),player_NN(x4)])
```

5.5.5.2 PN Errors - Correction

Now that errors have been identified and marked, I can proceed with their correction. This second task of the process is done manually due to the diversity of PN forms.
For the SynsetID+Synset file it is often sufficient to add first names to a shared surname (or vice-versa) as in:

```
     gloss_synsetID(110305694,
[Haldane,Elizabeth_Haldane,Elizabeth_Sanderson_Haldane])

     gloss_synsetID(110305872,
[Haldane,John_Haldane,John_Scott_Haldane])
```

which have been corrected in:

```
gloss_synsetID(110305694,
[Haldane_Elizabeth,Elizabeth_Haldane,Elizabeth_Sanderson_Haldane
])

gloss_synsetID(110305872,
[Haldane_John,John_Haldane, John_Scott_Haldane])
```

but in several cases a human interpretation is needed. This is the case for e.g. of *Henry_II*:

```
gloss_synsetID(110323405,[Henry_II])

gloss_synsetID(110323655,[Henry_II])
```

here there is no any surname to add for identifying the individuals and I have to decide how to proceed. I solved the ambiguity by adding the feature *of_England* and *of_France* but it is clear that occurrences of this kind must be considered case by case:

```
gloss_synsetID(110323405,[Henry_II_of_England])

gloss_synsetID(110323655,[Henry_II_of_France])
```

Human interpretation is even more required for the disambiguation of LFs. For instance, we need to know which *Rousseau* was the *French philosopher* and which one the *French painter* between *Jean Jacques* and *Henri* to disambiguate the corresponding LF:

```
    gloss_synsetID(110541545,
    [Rousseau,Jean-Jacques_Rousseau,]).

    gloss_synsetID(110541808,
[Rousseau,Henri_Rousseau,Le_Douanier_Rousseau,])

lf(rousseau_NN(x1),
[french_JJ(x1),philosopher_NN(x2),and_CC(x1,x2,x3),writer_NN(x3)
,born_VB(e1,x1),in_IN(e1,x4),switzerland_NN(x4)])
```

```
lf(rousseau_NN(x1),
[french_JJ(x1),primitive_JJ(x1),painter_NN(x1)])
```

After completing the disambiguation of PNs Synsets and PNs LFs in the two input files, I started again the Parser in order to finish the correction of free variables and complete the creation of UXWN.

5.6 Duplicate LFs

Every sense of a word needs to be identified with an unequivocal definition. If this doesn't happen, and two LFs are identical of each other, the meaning associated to one Synset would be interchangeable with the meaning associated to another Synset, which is clearly a sign of inconsistency.
I checked all the LFs in UXWN and I found duplicate LFs in all the four pos categories. There are two cases of duplicates:

1. Same synsetID, same lemma, same LF. In this case LFs are copies and need to be removed, e.g.:

```
synset(201238255, line_VB(e1, x1, x2), [line])-[mark_VB-[e1, x1],
with_IN-[e1, x3], line_NN(x3)]).
synset(201238255, line_VB(e1, x1, x2), [line])-[mark_VB-[e1, x1],
with_IN-[e1, x3], line_NN(x3)]).
```

This is due to the fact that in XWN some entries have a double LF, e.g.:

```
<gloss pos="VERB" synsetID="01238255">
  <synonymSet>line</synonymSet>
```

```
 <text>mark with lines; "sorrow had lined his face"</text>
. . .
 <lft quality="GOLD">
 line:VB(e1, x1, x2) -> mark:VB(e1, x1, x4) with:IN(e1, x3)
line:NN(x3)
 </lft>
 <lft quality="GOLD">
 line:VB(e1, x1, x2) -> mark:VB(e1, x1, x5) with:IN(e1, x3)
line:NN(x3)
 </lft>
</gloss>
```

In the above case the Parser links the SynsetID with the two LFs, and after correcting them (free variables x4 and x5 are removed) the result in UXWN are two entries with same synsetID and LF.

2. Different synsetID, same lemma, same LF. In this case two (ore more) senses of a word have the same definition, e.g.:

```
synset(200621145, multiply_VB(e1, x1, x2), [multiply])-[combine_VB-
[e1, x1], by_IN-[e1, x3], multiplication_NN(x3)]).
synset(200239103, multiply_VB(e1, x1, x2), [multiply, manifold])-
[combine_VB-[e1, x1], by_IN-[e1, x3], multiplication_NN(x3)]).
```

Also here, the problem comes from some entries in XWN where a lemma is associated with one LF plus the one(s) of the other senses of the lemma, e.g.:

```
<gloss pos="VERB" synsetID="00621145">
  <synonymSet>multiply</synonymSet>
 <text>combine by multiplication; "multiply 10 by 15"</text>
 . . .
 <lft quality="GOLD">
 multiply:VB(e1, x1, x2) -> combine:VB(e1, x1, x2) by:IN(e1, x3)
multiplication:NN(x3)
 </lft>
</gloss>


<gloss pos="VERB" synsetID="00239103">
  <synonymSet>multiply, manifold</synonymSet>
```

```
 <text>combine or increase by multiplication; "He managed to multiply
his profits"</text>
  …
 <lft quality="GOLD">
 multiply:VB(e1, x1, x2) -> combine:VB(e2, x1, x4) or:CC(e1, e2, e3)
increase:VB(e3, x1, x5) by:IN(e1, x3) multiplication:NN(x3)
 </lft>
 <lft quality="GOLD">
 multiply:VB(e1, x1, x2) -> combine:VB(e1, x1, x4) by:IN(e1, x3)
multiplication:NN(x3)
 </lft>
</gloss>
```

The Parser produces three entries for the above example:

```
synset(200621145, multiply_VB(e1, x1, x2), [multiply])-[combine_VB-
[e1, x1], by_IN-[e1, x3], multiplication_NN(x3)]).
synset(200239103, multiply_VB(e1, x1, x2), [multiply, manifold])-
[combine_VB-[e2, x1], or_CC-[e1, e2, e3], increase_VB-[e3, x1], by_IN-
[e1, x3], multiplication_NN(x3)]).
synset(200239103, multiply_VB(e1, x1, x2), [multiply, manifold])-
[combine_VB-[e1, x1], by_IN-[e1, x3], multiplication_NN(x3)]).
```

and the last one must be removed.
Here below I illustrate in Table 5.2 the situation I found checking for these mistakes.

POS FIle	LFs	Duplicate LFs - SameSynsetID	Duplicate LFs - DifferentSynset ID	Duplicate LFs - TOTAL
Noun	91,927	253	318	3571
Verb	14,447	8	98	106
Adjective	20,314	52	54	106
Adverb	3982	2	20	22
TOTAL	130,670	315	490	3805

Tab 5.2 Duplicate LFs

We can see that this kind of errors doesn't seriously compromise the resource as other errors do, and that the most affected category is again that of nouns.
I automatically inspected UXWN by building a system[49](System L in Appendix 22) that takes as input the resource and returns as output two files with the duplicate LFs, one file for each of the above mentioned cases. The output files contain SynsetID, lemma and LF of each duplicate LF.
To clean UXWN from unneeded LF duplicates, I used the output files of systemN to manually retrieve and remove the unwanted LFs.

5.7 Pos Tagging Errors - Correction

As previously described in section 4.2.5, the problem of wrong pos labels is quite common in XWN LFs. Unlike other types of errors, pos tagging errors occur both in the LHS and in the RHS of the LFs. Errors in the RHS are various and their total number is difficult to estimate. On the contrary, errors in the LHS have been measured and they are 1311 (only in the noun file).
I treated LHS errors and RHS errors separately:

LHS errors have been adjusted directly by the Parser previously described. Each LF taken by the Parser for the correction is then returned in UXWN with the correct LHS pos label. So for e.g. the LF of flip which was wrongly transformed as adjective in XWN:

[49] In appendix 22 and 23 the SystemL and its output for the adjective section (as example)

```
<gloss pos="NOUN" synsetID="01176224">
  <synonymSet>flip, toss</synonymSet>
 <text> the act of flipping a coin </text>
. . .
 <lft quality="NORMAL">
   flip:JJ(x3) -> act:NN(x1) of:IN(x1, e2) flip:VB(e2, x1, x2)
coin:NN(x2)
 </lft>
</gloss>
```

results with the correct pos label in UXWN:

```
synset(101176224,flip_NN(x1),[flip,toss])-
[act_NN(x1),of_IN(x1,e2),flip_VB-[e2,x2],coin_NN(x2)]
```

RHS errors have been manually corrected. Since their detection was not systematic due to the multiform nature of the errors and the incongruities with the parse trees, I proceeded in two ways: firstly, using regex I searched for the most common pos tagging mistakes (colours, numbers, nationality adjectives) and I corrected them; secondly, for each LF I manually inspected for whatever reason during the whole work of correction, I checked every pos labels and corrected the errors. Here below some typical examples regarding nationality adjectives and colours; in the LF of *dulse* a second wrong pos label has been assigned to the noun *seaweed*:

```
      XWN
orpington:NN(x1) -> english:NN(x1) breed:NN(x2) of:IN(x1, x3)
large:JJ(x3) white-skinned:JJ(x3) chicken:NN(x3)
      UXWN
synset(101713098,orpington_NN(x1),['Orpington'])-
[english_JJ(x1),breed_NN(x1),of_IN(x1,x3),large_JJ(x3),'white-
skinned_JJ'(x3),chicken_NN(x3)]

      XWN
```

```
dulse:NN(x1) -> coarse:JJ(x1) edible:JJ(x1) red:NN(x1)
seaweed:VB(e1, x2, x1)
      UXWN
synset(101339113,dulse_NN(x1),[dulse,'Rhodymenia_palmata'])-
[coarse_JJ(x1),edible_JJ(x1),red_JJ(x1),seaweed_NN(x1)]
```

5.8 Compound Nouns - Correction

As seen in section 4.2.2, the detection of nominal compounds is a tricky task which leads to different representations in XWN LFs.

In particular, CNs are transformed sometimes by means of the *nn*-predicate (`nn(x4, x2, x3) strike:NN(x2) zone:NN(x3)`) and can also occur as single words (`baseball_team:NN(x2)`). The most serious problem arises when CNs are not detected and appear in the LF as different predicates, with disconnected variables:

```
      XWN
malevolent_program:NN(x1)  ->  computer:NN(x1)  program:NN(x2)
design:VB(e1,  x4,  x1)  to:IN(e1,  e2)  have:VB(e2,  x1,  x3)
undesirable:JJ(x3) harmful:JJ(x3) effects:NN(x3)
```

The sense of this LF results to be that *malevolent program* is a *computer* designed to have undesirable and harmful effect while it is instead a *computer program*, a software not an hardware.

To cope with this problem I considered the fact that many CNs are stored in WN and therefore they appear in the Synsets. I decided to automatically compare the sequences of nouns in the LFs with the nouns in the Synsets. With this purpose I built SystemP in Appendix 25. I decided to treat only CNs - and not nominal compounds in general - mostly because they are

numerous in the resource and also because their misdetection always bring to a wrong LF while instead, for instance, LF of nominal compounds made of adjective+noun are often acceptable although not considered as compounds, e.g.:

```
synset(100093905, playing_NN(x1), [playing])-[act_NN(x1),
of_IN(x1, e1), play_VB-[e1, x2], musical_JJ(x2),
instrument_NN(x2)]
```

Moreover, I decided to work only on simple CNs and not on coordinated CNs (e.g. *goat and camel hair*) for which particular attention was provided by Rus in Rus 2002/2.

The first step is to gather all the lemmata from the Synsets of nouns. This is done with SystemO in Appendix24, which takes care of splitting Synsets with multiple lemmata and returns a txt file with one lemma per line.

Now that the list of nouns is available, for each LF, the system compares possible sequences of nouns with the lemmata in the list. It works directly on UXWN. Every time two nouns appear one next to the other in the LF, and there is no *nn*-predicate, it tries to find their union in the list by connecting them in different ways. So for e.g. in the following entry the SystemO finds the two consecutive nouns *health* and *care* and tries to connect them in different ways (*healthcare*, *health-car*e, *health_care*):

```
synset(100999111, healthcare_delivery_NN(x1), [healthcare_delivery,
health_care_delivery, care_delivery])-[provision_NN(x1), of_IN(x1, x2),
health_NN(x2), care_NN(x3)]
```

The results are compared with the list of nouns and one of them has a match in the list: *healthcare*.

While replacing the two predicates with the CN one, the system needs to take care also of variables. They have to be correctly bound and with this purpose it keeps the first one. The resulting LF in UXWN is:

synset(100999111, healthcare_delivery_NN(x1), [healthcare_delivery, health_care_delivery, care_delivery])-[provision_NN(x1), of_IN(x1, x2), **healthcare_NN(x2)**]

The LF results to be well structured, variables of arguments are correctly bound among them and the whole sense is preserved.
The first run of the system was done looking for CNs made of three words, in this way it identifies CNs like: *little_bighorn_river*, *roman_catholic_church*, *sun_myung_moon*, *posterior_cardinal_vein* etc. Also in this case the system keeps the first variables and succeed in correcting LFs such as:

```
     UXWN - Before SystemP
synset(101225246, tarawa_NN(x1), ['Tarawa', 'Makin', 'Tarawa-
Makin'])-[battle_NN(x1), in_IN(x1, x2), world_NN(x2),
war_NN(x3), ii_NN(x4), in_IN(x2, x5), pacific_NN(x5)]

     UXWN - After SystemP
synset(101225246, tarawa_NN(x1), ['Tarawa', 'Makin', 'Tarawa-
Makin'])-[battle_NN(x1), in_IN(x1, x2), world_war_ii_NN(x2),
in_IN(x2, x5), pacific_NN(x5)]
```

With the second run the system looks for CNs made of two words and corrects them in the same way. So for example, in the following case, the system firstly identifies the three-words CN *new_york_bay* and then the two-words CN *liberty_island*:

```
     UXWN - Before SystemP
```

```
synset(104141426, statue_of_liberty_NN(x1),
['Statue_of_Liberty'])-[large_JJ(x1), monumental_JJ(x1),
statue_NN(x1), symbolize_VB-[e1, x1, x2], liberty_NN(x2),
on_IN(x2, x3), liberty_NN(x3), island_NN(x4), in_IN(x3, x5),
new_NN(x5), york_NN(x6), bay_NN(x7)]
```

```
     UXWN - After SystemP - First Run
synset(104141426, statue_of_liberty_NN(x1),
['Statue_of_Liberty'])-[large_JJ(x1), monumental_JJ(x1),
statue_NN(x1), symbolize_VB-[e1, x1, x2], liberty_NN(x2),
on_IN(x2, x3), liberty_NN(x3), island_NN(x4), in_IN(x3, x5),
new_york_bay_NN(x5)]
```

```
     UXWN - After SystemP - Second Run
synset(104141426, statue_of_liberty_NN(x1),
['Statue_of_Liberty'])-[large_JJ(x1), monumental_JJ(x1),
statue_NN(x1), symbolize_VB-[e1, x1, x2], liberty_NN(x2),
on_IN(x2, x3), liberty_island_NN(x3), in_IN(x3, x5),
new_york_bay_NN(x5)]
```

As last run, the systems looks again for two-words CNs in order to find multiple cases of two words CNs in a same LF, as in:

```
     UXWN - Before SystemP
synset(100509974, rumba_NN(x1), [rumba])-[ballroom_NN(x1),
dance_NN(x2), base_VB-[e1, x1], on_IN-[e1, x3], cuban_JJ(x3),
folk_NN(x3), dance_NN(x4)]
```

```
    UXWN - After the last run of SystemP
synset(100509974, rumba_NN(x1), [rumba])-[ballroom_dance_NN(x1),
base_VB-[e1, x1], on_IN-[e1, x3], cuban_JJ(x3),
folk_dance_NN(x3)]
```

Each LF modified by SystemP is marked with * in order to allow for manual check.

The majority of corrections concerns those LFs that are definitions of nouns having them the most number of CNs.

Thanks to SystemP I succeeded in correcting more than 9K two-words CNs and about 150 three-words CNs.

5.9 Relative Adverbs - Correction

As shown in section 4.2.4, relative adverbs are sometimes erased from the LFs. SystemD in Appendix6 takes care of detecting all the cases of missing relative adverbs in the LFs; it takes as input each XWN pos file at time and records the synsetIDs of those LFs with missing relative adverbs in a txt file.
For the correction I worked directly on UXWN using the previously mentioned txt file with the recorded SynsetIDs.
I manually checked those LFs with missing relative adverbs, adding the relative adverb when needed. For picking out the affected LFs in UXWN I built SystemQ that compares each UXWN entry with the SynsetIDs stored in the file and marks the matched ones. Let's consider as example the definition of *Reign_of_Terror*: *the historic period (1793-94) during the French Revolution when thousands were executed*. The corresponding LF in XWN is:

```
reign_of_terror:NN(x1) -> historic:JJ(x1) period:NN(x1)
during:IN(x1, x2) french:NN(x2) revolution:NN(x3)
thousand:NN(x4) be:VB(e1, x4, e2) execute:VB(e2, x5, x4)
```

which results in UXWN, after the Parser correction, as:

```
synset(114397271, reign_of_terror_NN(x1), ['Reign_of_Terror'])-
[historic_JJ(x1), period_NN(x1), during_IN(x1, x2),
french_NN(x2), revolution_NN(x3), thousand_NN(x4), be_VB-[e1,
x4, e2], execute_VB-[e2, x4]]
```

the free variable x5 has been removed. After the automatic CNs correction, *french:NN(x2) revolution:NN(x3)* is correctly transformed into a CN with one bound variable:

```
synset(114397271, reign_of_terror_NN(x1), ['Reign_of_Terror'])-
[historic_JJ(x1), period_NN(x1), during_IN(x1, x2),
french_revolution_NN(x2), thousand_NN(x4), be_VB-[e1, x4, e2],
execute_VB-[e2, x4]]
```

Now, SystemQ identifies the SynsetID of this LF as a possible case of missing relative adverbs and this is true considering that *when* is missing in the LF. I eventually corrected the LF adding the relative adverb and the final result is:

```
synset(114397271, reign_of_terror_NN(x1), ['Reign_of_Terror'])-
[historic_JJ(x1), period_NN(x1), during_IN(x1, x2),
french_revolution_NN(x2), when_IN(x1,e1), thousand_NN(x4),
be_VB-[e1, x4, e2], execute_VB-[e2, x4]]
```

Some LFs, even though correctly identified by the system, don't need the insertion of the missing relative adverb as the sense of the LF is preserved anyway, see for e.g:

*Edmund_I : king of the English **who** succeeded Athelstan*

UXWN

```
synset(110236213,edmund_i_NN(x1),['Edmund_I'])-
[king_NN(x1),of_IN(x1,x2),english_NN(x2),succeed_VB-
[e1,x1,x3],athelstan_NN(x3)]
```

*Opkins_Sir_Frederick_Gowland: English biochemist **who** did pioneering work **that** led to the discovery of vitamins (1861-1947)*

UXWN

```
synset(110340955, hopkins_sir_frederick_gowland_NN(x1),
['Hopkins_Sir_Frederick_Gowland',
```

```
'Sir_Frederick_Gowland_Hopkins'])-[english_JJ(x1),
biochemist_NN(x1), do_VB-[e1, x1, x2], pioneering_JJ(x2),
work_NN(x2), lead_VB-[e2, x2], to_IN-[e2, x3], discovery_NN(x3),
of_IN(x3, x4), vitamin_NN(x4)]
```

The work was fast and easy for definitions of verbs and adjectives where LFs with errors were a few, it took longer for definitions of nouns where the LFs to check where more than 1K.

5.10 UXWN Release

In order to guarantee an easy access to the resource, UXWN has been released in XML format and it is freely downloadable at : http://www.unive.it/UXWN .
Due to the large size of whole resource, each pos file of UXWN is downloadable separately.
The output of the Parser is a txt file that, after the corrections, has been converted in a XML graph. Each entry of the txt file e.g:

```
synset(100002560,nothing_NN(x1),[nothing,nonentity])-
[nonexistent_JJ(x1),thing_NN(x1)]
```

is represented in the XML tree as a *synset* element where the attribute *ID* contains the WN synsetID (preceded by the number representing the pos), its sub-elements *s*, *w* and *lf* contain respectively: Synset, Lemma and LF as follow:

```
<uxwn>
. . .
 <synset ID="100002560">
  <s>nothing, nonentity</s>
```

```
  <w>nothing_NN(x1)</w>
  <lf>nonexistent_JJ(x1), thing_NN(x1)</lf>
 </synset>
. . .
</uxwn>
```

The XML format allows systems to easily access the information stored, it is extendable, readable and understandable (even by novices).

5.11 Conclusions

The precise number of corrected LFs is hard to measure. Some LFs had different kinds of errors and some cases detected by the automatic systems have not been considered as errors. For these reasons, to count the corrected LFs it is not proper to consider the results of automatic systems.
Even thought I can't provide a final percentage, it is fair to affirm that LFs have been substantially improved. Free variables have been almost removed (error rate from 54% to 15%), conjunctions and prepositions are no longer missing in the LFs as well as possessive pronouns and relative adverbs. More than 400 genitive markings have been manually corrected and the quality of pos labels has been refined. Many CNs (9K) have been finally identified and correctly transformed into LF and the almost 4K names of person are provided with Synsets that respect their instance nature. The manual checking have been costly and time consuming but it brought to really good results.

The improved LFs are provided in a new resource that, thanks to its clear structure and to the XML format, aims to be smoothly queryable by automatic systems.

Chapter 6

Future Work and Conclusions

6.1 Introduction

The efforts made so far to automatically and manually correct the LFs of XWN have led to the creation of the improved resource UXWN. The most common errors have been detected and corrected but are there more improvements it is worth considering? One of the advantages of this kind of LF is definitely its simple structure, augmenting the semantics encoded or adding more features might ruin this quality. The only profitable structural modification I envisage, regards how the adjectives are transformed in LF. I will discuss adjectives transformation in this final chapter where I will also ponder on how to test the new resource and on a possible alignment between UXWN and AMR (Abstract Meaning Representation), topics that, due to a lack of time, I couldn't study in depth.

6.2 Adjectives - a Further Improvement

As previously seen, object formulae include simple one place predication with just one variable associated to an entity. This formula is used for entities (e.g. *cat_NN(x1)*) as well as for modifiers like adjectives (e.g. *angry_JJ(x1)*).
It is well known that adjectives have different proprieties both syntactic and semantic and therefore a simple object formula might be inadequate to represent their relations with nouns.
Considering Larson's analysis of adjectives (Larson 1998), I will suggest here to add an event variable to adjectives in order to adjust they LF representation.
Form a syntactic point of view, English adjectives can occur **predicatively** when they are the main predicate in a clause or in a clause-like structure, as in:

- the stone is *weighty*

or they can occur **attributively** when they function as modifiers in a nominal, e.g.:

- two *small* elephants

Introducing an event variable for adjectives can allow to differentiate cases in which the same adjectival word plays the role of predicate or of attribute, as *red* in the following sentences:

a) The stiff hat was *red*

b) The *red* hat was stiff

The two sentences could be differentiated as follows, in a) x1 is associated to the subject of predication, *hat*, and the predication itself is constituted by a different property identified by variable e3; In b) the attribute is associated to the nominal head object variable x1 and is specified with event variable e2, assuming in this way that the property of being *stiff* is independent of the property of being *red*, but they are both associated to the entity x1:

a) be_VB(e1,x1), hat_NN(x1), stiff_JJ(e2,x1), **red_JJ(e3,e1)**

b) be_VB(e1,x1), hat_NN(x1), **red_JJ(e2,x1)**, stiff_JJ(e3,e1)

Considering inferential proprieties of adjectives used attributively, different interpretations are possible.
As discussed by Gal et al. 1991, adjectives can be categorised as **restrictive** and **non restrictive**, the latter are very few (e.g. *false*, *artificial* etc.). Restrictive adjectives are used to limit the number of items matching a given description and can be further divided into **scalar** adjectives and **descriptive** adjectives. Scalar adjectives like *small* or *tall* describe qualities that are measurable but in general the property is not fixed, it rather depends on the noun. For example, the scalar adjective *short* can refer to different measures according to the context: *a short person* (height), *a short journey* (temporal or distance). Furthermore the scalar-property is often comparative, a *big ant* is still a *small* living being. On the contrary, descriptive adjectives add some information to the noun they qualify (rather than referring to one of its properties). Descriptive adjectives can be represented

directly as predicates bound to the noun they qualify (a simple conjunction of predicates), a representation that is unsatisfactory for scalar adjectives that are generally implicit comparisons. In fact saying that a person is short means that he is shorter than the average person. But how can be represented in LF this difference between descriptive and scalar adjectives?
The need for a further research arises also from the well known difference between intersective and non intersective adjectives. Considering the following sentences:

- Mary is Italian
- Mary is a surgeon

we can think about Mary as a member of two different sets: the Italians and the surgeons. She is also a member of the intersection of these sets, therefore:

- Mary is an Italian surgeon

This reasoning doesn't work for other adjectives, for example the following sentences:

- Mary is skilful
- Mary is a painter

don't necessarily entail that:

- Mary is a skilful painter

In fact Mary can be skilful in general or as a surgeon but not as a painter.
The intersective non-intersective reading has been widely discuss in literature, often considering the following Larson's famous example:

- Olga is a beautiful dancer

Where Olga can be considered beautiful even if her dancing is awkward (in an intersective reading) or she can just be seen as someone who dances beautifully (in a non intersective reading).
Moreover, as illustrated by Cinque 2014, the interpretative proprieties of adjectives appear to be related to the two types of syntactic modification: abdominal adjectives as direct modifiers of the NP or as predicates of a reduced relative clause that modifies the NP. This is something to take into account while formulating a new representation.
Also the interpretation of adjacent adjectives needs to bee considered. For example, with the current LF, the sentence:

- the invisible visible stars

would have the following representation:

invisible_JJ(x1) visible_JJ(x1) star_NN(x1)

where *invisible* and *visible* modify the name in the same way without representing the real meaning of the sentence, i.e. those stars that are usually visible and are invisible in this moment.

- the invisible visible stars
- the visible invisible stars

} invisible_JJ(x1) visible_JJ(x1) star_NN(x1)

Furthermore, if we invert the order of the adjectives, the meaning of the sentence changes but this LF can't be used to represent this difference, as both adjectives are represented in the same way:
These considerations lead to the conclusion that it is worth to reconsider and reorganise the LF transformation of adjectives and this is one of the envisaged future works.

6.3 Testing the Resource

Another important topic I din't have the time to develop and that is among the first points of future works, regards the use of UXWN. How much the new resource improve the results of NLP systems in comparison to XWN? This is something it is worth considering for a thorough evaluation of the resource.
We have seen that XWN, and LFs in general, have been used in Q/A. As results in Moldovan and Rus 2001/1 show, such application is satisfactory but still improvable. It is right to expect that a better resource can influence positively the results of its applications.
Let's consider an example:

- Is Mick Jagger a singer?

Answering this question is easy for every one, but a computer needs some world knowledge to choose the right answer. As already seen, WN Glosses encode a lot of knowledge that can be exploited for this purpose. A system can reply this question simply by querying the database. In fact, thanks to WN, we know that:

1. Jagger, Mick Jagger, Michael Philip Jagger : English rock star (born in 1943)
2. rock star : a famous singer of rock music

the LFs of these glosses in XWN are (respectively):

1. jagger:NN(x1) -> english:NN(x1) rock:NN(x2) star:NN(x3)
2. rock_star:NN(x1) -> famous:JJ(x1) singer:NN(x1) of:IN(x1, x2) rock_music:NN(x2)

the two LFs contain the knowledge a system needs to answer the question, the problem is that it is wrongly encoded. According to the first LF, *Jagger* is an *English* but there is no relation to the fact that he is also a *rock star*. In fact *rock* and *star* are represented as two different entities, two different predicates with different variables. This doesn't allow a possible system to build the connection between *rock star* and *singer*.
Let's now consider the two corresponding representations in UXWN:

1. synset(110357697, jagger_NN(x1), ['Jagger', 'Mick_Jagger', 'Michael_Philip_Jagger'])-[**english_JJ(x1), rock_star_NN(x1)**]
2. synset(109850134,rock_star_NN(x1),[rock_star])-[famous_JJ(x1),singer_NN(x1),of_IN(x1,x2),rock_music_NN(x2)]

Here the encoding problem has been solved: *English* has been correctly labeled as adjective and *rock star* has been transformed as CN. The automatic inference is now allowed.
As shown in chapters 4 and 5, various kinds of error have been detected and corrected during for the creation of UXWN. In the previous example, we have seen how the correction of pos labels and the proper transformation of CNs can significantly improve the result of a simple Q/A task. The intention is to test the resource with an automatic system and collect a sufficient amount of data to estimate its application in comparison to other similar resources.

6.4 UXWN and AMR

The abstract meaning representation (AMR) is a graph-based semantic representation of a natural language sentence that embeds annotations related to traditional tasks such as semantic role labelling, wsd, named entity recognition etc. One of the goals of AMR is to include in a single dataset different basic disambiguation information which are usually encoded in different datasets. In fact, semantic annotation today is divided

into different annotations associated to different evaluations and training data, split across many resources.

There exist different corpora that have been manually transformed in AMR and nowadays, the growing interest of the NLP community for this compact, readable, whole-sentence semantic annotation, is reflected in a number of different works that aim at automatically transforming English sentences into AMR graphs. Both in SemEval-2016[50] and in SemEval-2017[51] one task was dedicated to the automatic generation of this representation and different parsers have been built with this purpose (see for e.g. Pust et al. 2015, Vanderwende et al. 2015, Wang et al. 2016).

Just like the LF subject of this thesis, following a neo-Davidsonian fashion, AMR introduces variables for entity and events and in addition also for properties and states. In the AMR graph, not all the individual words in a sentence are annotated, leaves are rather labeled with concepts and relations link entities. AMR concepts are either English words ("boy"), PropBank framesets ("want-01"), or special keywords. AMR uses approximately 100 relations: frame arguments, following PropBank conventions (e.g. :arg0), general semantic relations (e.g. :cause), relations for quantities (e.g. :quant), relations for date-entities (e.g. :day), relations for lists (e.g. op1). For a detailed description, the reader is referred to the AMR guidelines[52].

Below, an example of AMR for the sentence *the boy wants the girl to believe him* :

---

[50] http://alt.qcri.org/semeval2016/task8/

[51] http://alt.qcri.org/semeval2017/task9/

[52] https://www.isi.edu/~ulf/amr/help/amr-guidelines.pdf

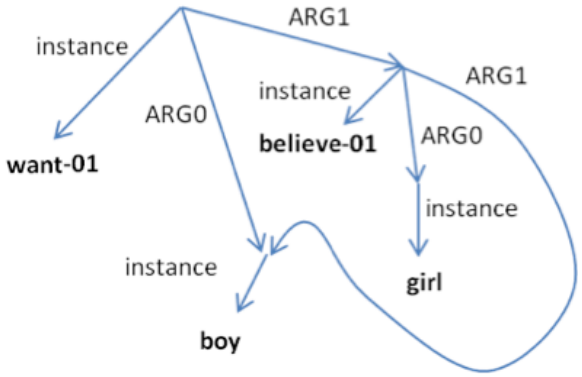


Fig 6.1 Example of AMR

another way to represent the same sentence with AMR:

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (b2 / believe-01
             :ARG0 (g / girl)
             :ARG1 b))
```

Despite some similarities with our LF, AMR is quite different. Both the representations attempt to provide the same structure for sentences that have the same basic meaning. The difference is that our LF relies more on the single words of a sentence than what AMR does. For example, if it is true that the following sentences

a) the boy was killed by the girl

b) the girl killed the boy

have the same LF

$$\text{boy\_NN}(x2), \text{kill}(e1, x1, x2), \text{girl}(x1)$$

it is also true that the following sentences

c) The girl made adjustments to the machine.

d) The girl adjusted the machine.

e) The machine was adjusted by the girl.

have different LFs but a unique AMR:

```
(a / adjust-01
   :ARG0 (b / girl)
   :ARG1 (m / machine))
```

Obviously, this is not the only difference between the two representations. Our LF is definitely more simple and less rich in semantics features, and this is considered its greatest strength.
But considering the growing interest of the NLP community towards AMR, I'd like to investigate a possible alignment between the simple LF and this new representation. This might be done in two ways: or making the LF more close to AMR, for example by adding some of those semantic features which are missing or by making the structure more close to the AMR format, or by providing each entry of UXWN with the corresponding AMR representation.
As proved also by the results of SemEval (see May 2016), the automatic generation of AMR is still a difficult and competitive task which results are sometimes quite far from the gold standards (manually annotated AMR). Therefore, using one of the existing AMR parsers might be risky for the whole correctness of the resource. It is reasonable to consider other solutions and one idea in this sense would be to take a closer look to the work of J. Bos 2016 who proposes a systematic translation from AMR to first order logic formulas.

Furthermore, a comparative analysis between the two representations might be useful to discover strengths and weaknesses of these structures.

6.5 Conclusions

WordNet is one of the most famous resources for NLP. It is widely used and studied and, during the last decades, many researchers have investigated and implemented different improvements and extensions.
My work began from here, with a summary of different improvements divided into two categories: Terminology Extension and Relations Enhancement. The first category includes my participation to the Task 14 of SemEval 2016 (Semantic Taxonomy Enrichment), while the second one concerns in particular WSD of WN glosses. Chapter 2 is a first introduction to the real topics of this dissertation: XWN and LF.
In fact, the idea of this thesis was to analyse XWN, a resource that aims at providing several important enhancements to WN. In XWN the WN glosses are syntactically parsed, transformed into LF and content words are semantically disambiguated. Therefore, XWN brings mainly two important improvements to the WN glosses: WSD and the LF.
LF is a kind of semantic representation which stands between the syntactic parse and the deep semantic form. It is a simple and highly effective representation which is used in several NLP systems. There are different types of LF and 3 different resources

that provides the LF of WN glosses. In Chapter 3 I showed different representations and I compared the three LF resources justifying the choice to work on XWN.
In Chapter 4 and 5 I described respectively the work of analysis of XWN and the improvements made, which led to a new resource that I named UXWN and which is freely downloadable from the dedicated webpage: http://www.unive.it/UXWN .
During the analysis, it comes out that even if a correct LF transformation relies on the quality of the syntactic parsing, this is necessary but not sufficient to guarantee the correctness of the representation. Furthermore, comparing the different existing LF resources, I observed that some types of error are recurring when the LF is automatically derived. Taking into account these considerations might be useful for father implementations.

At the beginning of this work my aim was to study the LF, to analyse XWN, its errors and possible applications. But the more I went ahead with the project, the more I understood that my contribution to the NLP community could be something more. I eventually produced a proper and usable resource where world knowledge is represented as consistent LFs.

As discussed in the last pages, other improvements are possible and the ongoing UXWN project will take care of implementing them.

Appendices

APPENDIX 1

SYSTEM A - AVERAGE LENGHT OF DEFINITIONS

```
import xml.etree.ElementTree as ET
import re
tree=ET.parse("FilePOS.xml")
root=tree.getroot()
lista_definizioni=[]
definizioni=open("definizioni_POS.txt","a")
for w in root.iter(tag="text"):
      glossa=w.text
      newglossa1= re.sub(r"\(.*?\)", "", glossa)
      if re.search(r"\; \".*?\"", newglossa1):
         newglossa2= re.sub(r"\; \".*?\"", "", newglossa1)
      else:
         newglossa2=newglossa1
      if re.search(r"\; [a-z]*", newglossa2):
         temp=newglossa2.split(";")
         for x in temp:
             newdefinition=x.lstrip()
             definition=newdefinition.rstrip()
             definizioni.write(definition+"\n")
             lista_definizioni.append(definition)
      else:
         newdefinition=newglossa2.lstrip()
         definition=newdefinition.rstrip()
         definizioni.write(definition+"\n")
         lista_definizioni.append(definition)
lunghezzadefinizioni=[]
for w in lista_definizioni:
         x=len(w)
         lunghezzadefinizioni.append(x)
tot=0
for x in lunghezzadefinizioni:
         tot=tot+x
averagelenght= tot/numerodefinizioni

definizioni=open("definizioni_POS.txt","a")
definizioni.write("\n"+"the average length of the definitions in
the pos file is : "+str(averagelenght)+"\n")
definizioni.close()
```

\- - - - - - - - - -

Excerpt from the file output - verb pos file:

```
...
drench or submerge or be drenched or submerged
become empty of water
get foggy
```

```
burn to charcoal
become hazy, dull, or cloudy
cause to burn rapidly and with great intensity

the average length of the definitions in the noun file is : 53
```

APPENDIX 2

SYSTEM B - MAX NUMBER OF WORDS IN THE SYNSETS

```
import xml.etree.ElementTree as ET
import re
tree=ET.parse("POS.xml")
root=tree.getroot()
count=0
lista_count=[]
lista_nomi=[]
for w in root.iter(tag="synonymSet"):
      synset=w.text
      contasynset=open("synsets_POS.txt","a")
      contasynset.write(synset+"\n")
      contasynset.close()

s=open("synsets_POS.txt", "r").readlines()
for x in s:
    if "," in x:
        temp=x.split(",")
        for w in temp:
            count=count+1
        lista_count.append(count)
        lista_nomi.append(x)
        count=0
    else:
        count=1
        lista_count.append(count)
        lista_nomi.append(x)
        count=0

listacheck=zip(lista_count, lista_nomi)
#listacheck is a list of tuples integer+string
c=listacheck[0]
d=c[0]
syn=c[1]
for x in listacheck:
    if x[0]>d:
        syn=x[1]
        d=x[0]
    else:
        pass
print "the lenght of the longest synset is %d" %d
print syn

for x in listacheck:
#if instead of d we insert a positive integer y we find all the
synsets
#with lenght y
    if x[0]==d:
```

```
        maxsyn=open("maxsynset_POS.txt","a")
        maxsyn.write(x[1]+"\n")
        maxsyn.close()
    else:
        pass
```

System Description

For each element in XWN, the system gathers the synset and saves it in a txt file. Then, for each synset saved, it counts the number of words it is made of. Finally, it finds the longest synset and it prints it out.
The system works on one XWN pos file at time and therefore it returns 4 different outputs.

Results are shown here below:

POS	MAX LENGHT	SYNSET
NOUN	28	buttocks, nates, arse, butt, backside, bum, buns, can, fundament, hindquarters, hind_end, keister, posterior, prat, rear, rear_end, rump, stern, seat, tail, tail_end, tooshie, tush, bottom, behind, derriere, fanny, ass
VERB	24	roll_in_the_hay, love, make_out, make_love, sleep_with, get_laid, have_sex, know, do_it, be_intimate, have_intercourse, have_it_away, have_it_off, screw, fuck, jazz, eff, hump, lie_with, bed, have_a_go_at_it, bang, get_it_on, bonk
ADJECTIVE	25	besotted, blind_drunk, blotto, crocked, cockeyed, fuddled, loaded, pie-eyed, pissed, pixilated, plastered, potty, slopped, sloshed, smashed, soaked, soused, sozzled, squiffy, stiff, tiddly, tiddley, tight, tipsy, wet
ADVERB	11	immediately, instantly, straightaway, straight_off, directly, now, right_away, at_once, forthwith, in_real_time, like_a_shot

APPENDIX 3

SENSEVAL3 - IDENTIFICATION OF LOGIC FORMS IN ENGLISH

TRIAL DATA - ENGLIS SENTENCES

Some students like to study in the mornings.
Juan and Arturo play football every afternoon.
Alicia goes to the library and studies every day.
I tried to speak Spanish, and my friend tried to speak English.
Alejandro played football, so Maria went shopping.
Alejandro played football, for Maria went shopping.
When he handed in his homework, he forgot to give the teacher the last page.
The teacher returned the homework after she noticed the error.
The students who are on the bus to the United States are studying English.
After they finished studying, Juan and Maria went to the movies.
Juan and Maria went to the movies after they finished studying.
You can look up my number in the telephone directory.
I look forward to meeting you.
If you heat ice it melts.
If I am late for work my boss gets angry.
She acts as if she were Queen.
She will be delighted to see you.
Mary is a teacher.
Tara is beautiful.
That sounds interesting.
The sky became dark.
The bread has gone bad.
Mary seemed able to win the race, but she became fatigued near the finish line.
Greg is kicking the ball now.
The wind blows constantly in Chicago.
He accepted my apology.
The cake smells good!
Ellen smells the cake.
The woman grew silent.
The gardener grew some flowers.
Jane appeared uninjured after the accident.
Before I could leave, Jane appeared.
The dog was sick.
Fred felt funny.
Mad dogs and Englishmen go out in the midday sun.
They are jealous.
What she said is untrue.
Fred bit his thumb.
The chimpanzees groomed each other.
Jane gave the gorilla a kiss.
Jane gave a kiss to the gorilla.
Sunshine makes me very happy.
The voters elected Clinton president of the USA.
They ran quickly.
He went home twice nightly.

We walked on the playground.
My friend phoned me this morning.
I was happy when I saw her again.
The bus was full. However, Fred found a seat.

APPENDIX 4

SENSEVAL3 - IDENTIFICATION OF LOGIC FORMS IN ENGLISH

TRIAL DATA - LOGIC FORMS

```
student:n_ (x1) like:v_ (e4, x1, e5) to (e4, e5) study:v_ (e5, x1, x2)
in (e5, x2) morning:n_ (x2) .
Juan:n_ (x1) and (x7, x1, x2) Arturo:n_ (x2) play:v_ (e6, x7, x3, x4)
football:n_ (x3) afternoon:n_ (x4) .
Alicia:n_ (x1) go:v_ (e6, x1, x7) to (e6, x2) library:n_ (x2) and (e7,
e6, e3) study:v_ (e3, x1, x4) day:n_ (x4) .
I (x3) try:v_ (e6, x3, e7) to (e6, e7) speak:v_ (e7, x3, x11)
Spanish:n_ (x11) and (e10, e6, e8) my (x1) friend:n_ (x1) try:v_ (e8,
x1, e9) to (e8, e9) speak:v_ (e9, x1, x2) English:n_ (x2) .
Alejandro:n_ (x1) play:v_ (e7, x1, x2) football:n_ (x2) so (e9, e7,
e8) Maria:n_ (x3) go:v_ (e8, x3, x4) shopping:n_ (x4) .
Alejandro:n_ (x1) play:v_ (e7, x1, x2) football:n_ (x2) for(e2, e7,
e8) Maria:n_ (x3) go:v_ (e8, x3, x4) shopping:n_ (x4) .
When (e11, e10) he (x4) hand:v_ (e10, x4, x1) in (e10) his (x1)
homework:n_ (x1) he (x6) forget:v_ (e11, x6, e12) to (e11, e12)
give:v_ (e12, x6, x3, x2) teacher:n_ (x2) last:a_ (x3) page:n_ (x3) .
teacher:n_ (x1) return:v_ (e6, x1, x2) homework:n_ (x2) after (e6, e7)
she (x4) notice:v_ (e7, x4, x3) error:n_ (x3) .
student:n_ (x1) be:v_ (e8, x1, x2) on (e8, x2) bus:n_ (x2) to (x2, x4)
United_States:n_ (x4) study:v_ (e9, x1, x6) English:n_ (x6) .
After (e8, e6) they (x4) finish:v_ (e6, x4, e7) study:v_ (e7, x4)
Juan:n_ (x1) and (x9, x1, x2) Maria:n_ (x2) go:v_ (e8, x9, x3) to (e8,
x3) movie:n_ (x3) .
Juan:n_ (x1) and (x8, x1, x2) Maria:n_ (x2) go:v_ (e5, x8, x3) to (e5,
x3) movie:n_ (x3) after (e8, e6) they (x4) finish:v_ (e6, x4, e7)
study:v_ (e7, x4) .
You (x5) look_up:v_ (e8, x5, x1, x4) my (x1) number:n_ (x1) in (e8,
x4) telephone:n_ (x2) directory:n_ (x3) nn (x4, x2, x3) .
I (x1) look_forward_to:v_ (e4, x1, e5) meet:v_ (e5, x1, x2) you (x2) .
If (e5, e4) you (x2) heat:v_ (e4, x2, x1) ice:n_ (x1) it (x3) melt:v_
(e5, x3) .
If (e6, e5) I (x3) be:v_ (e5, x3, x1) late:a_ (x3) for (x3, x1)
work:n_ (x1) my (x2) boss:n_ (x2) get:v_ (e6, x2) angry:a_ (x2) .
She (x2) act:v_ (e5, x2) as_if (e5, e6) she (x3) be:v_ (e6, x3, x1)
Queen:n_ (x1) .
She (x1) be;v_ (e4, x1) delighted(x1) to (e4, e5) see:v_ (e5, x1, x2)
you (x2) .
Mary:n_ (x1) be:v_ (e4, x1, x2) teacher:n_ (x2) .
Tara:n_ (x1) be:v_ (e2, x1) beautiful:a_ (x1) .
That (x1) sound:v_ (e1, x1) interesting:a_ (x1) .
sky:n_ (x1) become:v_ (e2, x1) dark:a_ (x1) .
bread:n_ (x1) go:v_ (e2, x1) bad:a_ (x1) .
Mary:n_ (x1) seem:v_ (e9, x1) able:a_ (x1) to (e9, e10) win:v_ (e10,
x1, x2) race:n_ (x2) but (e13, e9, e11) she (x6) become:v_ (e11, x6)
fatigued:a_ (x6) near (e11, x5) finish:v_ (x3) line:n_ (x4) nn (x5,
x3, x4) .
Greg:n_ (x1) kick:v_ (e3, x1) ball:n (x2) now:r_ (e3) .
```

```
wind:n_ (x1) blow:v_ (e2, x1, x3) constantly:r_ (e2) in (e2, x3)
Chicago:n_ (x3) .
He (x2) accept:v_ (e5, x2, x1) my (x3) apology:n_ (x1) .
cake:n_ (x1) smell:v_ (e2, x1) good:a_ (x1) !
Ellen:n_ (x1) smell:v_ (e4, x1, x2) cake:n_ (x2) .
woman:n_ (x1) grow:v_ (e2, x1) silent:a_ (x1) .
gardener:n_ (x1) grow:v_ (e4, x1, x2) flower:n_ (x2) .
Jane:n_ (x1) appear:v_ (e4, x1, x2) uninjured:a_ (x1) after (e4, x2)
accident:n_ (x2) .
Before (e4, e3) I (x2) leave:v_ (e3, x2) Jane:n_ (x1) appear:v_ (e4,
x1) .
dog:n_ (x1) be:v_ (e2, x1) sick:a_ (x1) .
Fred:n_ (x1) felt:v_ (e2, x1) funny:a_ (x1) .
Mad:a_ (x1) dog:n_ (x1) and (x9, x1, x2) englishman:n_ (x2) go:v_ (e8,
x9, x5) out:r_ (e8) in (e8, x5) midday:n_ (x3) sun:n_ (x4) nn (x5, x3,
x4) .
They (x1) be:v_ (e2, x1) jealous:a_ (x1) .
What (x4) she (x1) say:v_ (e2, x1, x4) be:v_ (e3, e2) untrue:r_ (e2) .
Fred:n_ (x1) bit:v_ (e4, x1, x2) his (x2) thumb:n_ (x2) .
chimpanzee:n_ (x1) groom:v_ (e4, x1, x2) each_other:n_ (x2) .
Jane:n_ (x1) give:v_ (e5, x1, x3, x2) gorilla:n_ (x2) kiss:n_ (x3) .
Jane:n_ (x1) give:v_ (e5, x1, x2, x3) kiss:n_ (x2) to (e5, x3)
gorilla:n_ (x3) .
Sunshine:n_ (x1) make:v_ (e3, x1, x2) me (x2) very:r_ (x2) happy:a_
(x2) .
voter:n_ (x1) elect:v_ (e7, x1, x2) Clinton:n_ (x2) president:n_ (x2)
of (x2, x5) USA:n_ (x5) .
They (x1) run:v_ (e2, x1) quickly:r_ (e2) .
He (x1) go:v_ (e2, x1, x3) home:n_ (x3) twice:r (e2) nightly:r (e2) .
We (x2) walk:v_ (e4, x2, x1) on (e4, x1) playground:n_ (x1) .
My (x1) friend:n_ (x1) phone:v_ (e6, x1, x4, x2) me (x4) morning:n_
(x2) .
I (x1) be:v_ (e4, x1) happy:a_ (x1) when (e4, e5) I (x2) saw:v_ (e5,
x2, x3) her (x3) again:r_ (e5) .
bus:n_ (x1) be:v_ (e2, x1) full:a_ (x1) .
However:r_ (e4) Fred:n_ (x1) found:v_ (e4, x1, x2) seat:n_ (x2) .
```

APPENDIX 5

SYSTEM C - MISSING CONJUNCTIONS

```
import xml.etree.ElementTree as ET
import re
tree=ET.parse("POSfile.xml")
synset=open("SynsetsPOS.txt","a")
root=tree.getroot()
countnoncontrollate=0
missingand=0
missingor=0
listatext=[]
listalf=[]
count=0
tx=[]
countcontrollate=0
listasynset=[]

for element in root.iter(tag="gloss"):
      count=count+1
lenght=count
x=0

while x<lenght:
      text=root[x].find("text").text
      if ";" in text:#it takes only the first definition, no
examples
         tx=text.split(";")
         text1=tx[0]
      else:
         text1=text
      if "(" in text1:#deletes content between brackets
         text2=re.sub("\(.*?\)", "", text1)
         listatext=text2.split()
      else:
         listatext=text1.split()
      for n in range(len(listatext)):#deletes commas from tokens
         if "," in listatext[n]:
             y=re.sub(",", "", listatext[n])
             listatext[n]=y
         else:
             pass
      lf=root[x].find("lft").text
      countcontrollate=countcontrollate+1
      lft=re.sub("\:[A-Z]*\(.*?\)", "", lf)#deletes predicates
and        arguments
      lfts=re.sub(".*?\-\>", "", lft)#deletes word to define and
->
      listalf=lfts.split()
      if "and" in listatext:
```

```
            if "and" not in listalf:
                  missingand=missingand+1
                  syn=root[x].get("synsetID")#check the synsetID
                  if syn not in listasynset:
                        listasynset.append(syn)
                  else:
                        pass
            else:
                  pass
      else:
            pass
      if "or" in listatext:
            if "or" not in listalf:
                  missingor=missingor+1
                  syn=root[x].get("synsetID")#check the synsetID
                  if syn not in listasynset:
                        listasynset.append(syn)
                  else:
                        pass
            else:
                  pass
      else:
            pass
    x=x+1
for s in listasynset:
      synset.write(s+"\n")
synset.close()

print "I checked %d text-lf pairs" %countcontrollate
print "AND is missing in %d cases" %missingand
print "OR is missing in %d cases" %missingor
print "There are %d LFs with conjunctions errors" %missingor
```

System Description

System C takes as input XWN (one pos file at time) and for each entry it gathers the definition (first one if there are more than one) and its LF. Definitions are split into tokens as well as LFs. The system deletes the first part of LFs (word+"->"), commas, predicates and arguments. So for e.g. for the gloss of *benthos*:

```
<gloss pos="NOUN" synsetID="00004358">
  <synonymSet>benthos</synonymSet>
 <text>
   organisms (plants and animals) that live at or near the bottom of a
sea
 </text>
 . . .
```

```
 <lft quality="SILVER">
  benthos:NN(x1) -> organism:NN(x1) live:VB(e1, x1, x26) at:IN(e1, x4)
near:IN(e1, x5) bottom:NN(x2) of:IN(x2, x3) sea:NN(x3)
 </lft>
</gloss>
```

the system compares:

```
text ['organisms', 'that', 'live', 'at', 'or', 'near', 'the',
'bottom', 'of', 'a', 'sea']
```

with

```
lf ['organism', 'live', 'at', 'near', 'bottom', 'of', 'sea']
```

and checks if *and* or *or* are in the definition and not in the LF.
In this case it founds that *or* appears in the definition but is missing in the LF and counts this case as a missing-or case. Furthermore, the system records the SynsetIds of those LFs with missing conjunctions in order to count them (if a LF has tow cases of missing conjunctions it is only one time).
The final output of the system has the structure:

```
I checked n text-lf pairs
AND is missing in n cases
 OR is missing in n cases
There are n LFs with conjunctions errors
```

Final output in the Python Shell:

```
>>> ================================ RESTART =====
>>>
noun
I checked 79689 text-lf pairs
AND is missing in 3156 cases
OR is missing in 2981 cases
There are 2981 LFs with conjunctions errors
>>> ================================ RESTART =====
>>>
verb
I checked 13507 text-lf pairs
AND is missing in 320 cases
OR is missing in 1017 cases
There are 1017 LFs with conjunctions errors
>>> ================================ RESTART =====
>>>
adjective
I checked 18561 text-lf pairs
AND is missing in 752 cases
OR is missing in 2950 cases
There are 2950 LFs with conjunctions errors
>>> ================================ RESTART =====
>>>
adverb
I checked 3664 text-lf pairs
AND is missing in 148 cases
OR is missing in 331 cases
There are 331 LFs with conjunctions errors
>>>
```

APPENDIX 6

SYSTEM D - RELATIVE ADVERBS

```
import xml.etree.ElementTree as ET
import re
tree=ET.parse("POS.xml")
root=tree.getroot()
count1=0
count2=0
listatext=[]
listalf=[]
listasynset=[]
count=0
tx=[]
reladvs=["where","when","why","how","why"]
tag=0
gat=0

for element in root.iter(tag="gloss"):
    count=count+1
lenght=count
x=0

while x<lenght:
    text=root[x].find("text").text
    if ";" in text:#it takes only the firt definition, no examples
        tx=text.split(";")
        text1=tx[0]
    else:
        text1=text
    if "(" in text1:#delete content between brackets
        text2=re.sub("\(.*?\)", "", text1)
        listatext=text2.split()
    else:
        listatext=text1.split()
    for n in range(len(listatext)):
        if "," in listatext[n]:
            y=re.sub(",", "", listatext[n])
            listatext[n]=y
        else:
            pass
   lf=root[x].find("lft").text
   count1=count1+1
    lft=re.sub(".*?\-\>", "", lf)#delete word to define and ->
    lfts=re.sub("\:[A-Z]*\(.*?\)", "", lft)#delete predicates and arguments
    listalf=lfts.split()
   for adv in reladvs: #check if one adverb is in the definition but is missing in the LF
        if adv in listatext and adv not in listalf:
            for element in listalf:
                if "_" in element:
```

```
                check=element.split("_")
                for token in check:
                    if adv==token: #e.g. as_when
                        tag=tag+1
                    else:
                        pass
            if tag<1:
                count2=count2+1
                syn=root[x].get("synsetID")
                if syn not in listasynset:
                        listasynset.append(syn)
                else:
                        pass
                print "\n MISSING ADVERB \n %s" %syn, "\n listatext %s" %listatext, "\n
listalf %s"                             %listalf
            else:
                pass
        else:
            pass
        tag=0

    x=x+1

l=open("synsets_missingradverbs_POS.txt", "a")
for syn in listasynset:
        l.write(syn+"\n")
l.close()

print "I checked %d" %count1, " lf-text pairs there are %d LFs with missing relative
adverbs" %count2
```

System Description

For each entry in XWN, the system compares the definition and its LF both divided into tokens. In order to do not count as possible errors those adverbs transformed into LF attached to other particles e.g.:

```
definition of halloo: shout `halloo', as when greeting someone
or attracting attention

LF: halloo:VB(e1, x1, x2) -> shout:VB(e1, x1, x2) halloo:NN(x2)
as_when:IN(e1, e5) greet:VB(e3, x1, x3) someone:NN(x3) or:CC(e5,
e3, e4) attract:VB(e4, x1, x4) attention:NN(x4)
```

the system checks both parts of compounds-tokens (*as* and *when* in this case).

The output is structured as following:

For each LF with missing relative adverb(s):

an alert message saying MISSING ADVERB + synsetID of the gloss where missing adverb(s) is/are found in the LF + definition tokens + LF tokens

at the end:

number of checked defintion-LF pairs + numebr of LFs with missing relative adverbs

The synsetsID of LFs with missing relative adverbs are recorder in a txt file I will use for the correction.

```
>>> ================================ RESTART ==========
>>>
I found 1040 negations in the noun file
>>> ================================ RESTART ==========
>>>
I found 62 negations in the verb file
>>> ================================ RESTART ==========
>>>
I found 2074 negations in the adjective file
>>> ================================ RESTART ==========
>>>
I found 82 negations in the adverb file
>>>
```

Example of the output in the Python Shell - noun file:

```
 MISSING ADVERB
 14430040
 listatext ['a', 'time', 'period', 'when', 'some', 'activity', 'or', 'skill', 'wa
s', 'at', 'its', 'peak']
 listalf ['time', 'period', 'some', 'activity', 'or', 'skill', 'be', 'at', 'peak'
]

 MISSING ADVERB
 14430375
 listatext ['the', 'period', 'from', '1811-1820', 'when', 'the', 'Prince', 'of',
'Wales', 'was', 'regent', 'during', 'George', "III's", 'periods', 'of', 'insanity
']
 listalf ['period', 'from', '1811-1820', 'prince', 'of', 'wales', 'be', 'regent',
'during', 'george', "iii's", 'period', 'of', 'insanity']

 MISSING ADVERB
 14433212
 listatext ['a', 'period', 'during', 'a', 'parliamentary', 'session', 'when', 'me
mbers', 'of', 'Parliament', 'may', 'ask', 'questions', 'of', 'the', 'ministers']
 listalf ['period', 'during', 'parliamentary', 'session', 'member', 'of', 'parlia
ment', 'ask', 'question', 'of', 'minister']

 MISSING ADVERB
 14434730
 listatext ['the', 'day', 'in', '2001', 'when', 'Arab', 'suicide', 'bombers', 'hi
jacked', 'United', 'States', 'airliners', 'and', 'used', 'them', 'as', 'bombs']
 listalf ['day', 'in', '2001', 'arab', 'suicide', 'bomber', 'hijack', 'united', '
state', 'airliner', 'and', 'use', 'as', 'bomb']

 I checked 79689  lf-text pairs there are 1252 LFs with missing relative adverbs
```

APPENDIX 7

SYSTEM E - RANDOM SELECTION OF LFs

```
import xml.etree.ElementTree as ET
import re
from random import randint

tree=ET.parse("POS.xml")
root=tree.getroot()
lf=open("formelogichePOS.txt","a")
for element in root.iter(tag="lft"):
    x=element.text
    quality=element.attrib["quality"]
    lf.write("["+quality+"]"+x)
lf.close()



lf=open("formelogicheNN.txt").readlines()
listarandom=[]
begin=0
end=len(lf)-1
countgold=0
countsilver=0
countnormal=0
while countgold<=50 and countsilver<=50 and countnormal<=50:
    x=randint(begin,end)
    if x not in listarandom: #be sure the random number is
different
        if "[GOLD]" in lf[x] and countgold!=50:
            fiftygold=open("50goldNN.txt", "a")
            fiftygold.write(lf[x])
            fiftygold.close()
            countgold=countgold+1
        elif "[SILVER]" in lf[x] and countsilver!=50:
            fiftysilver=open("50silverNN.txt", "a")
            fiftysilver.write(lf[x])
            fiftysilver.close()
            countsilver=countsilver+1
        elif "[NORMAL]" in lf[x] and countnormal!=50:
            fiftynormal=open("50normalNN.txt", "a")
            fiftynormal.write(lf[x])
            fiftynormal.close()
            countnormal=countnormal+1
        else:
            pass
    else:
        pass
```

System Description

As first thing, the system takes as input each XWN pos file (one per time) and considers each LF and its quality.
It creates 4 pos files (.txt) where each line is structured as: quality of the LF + LF. E.g. from the noun file:

[NORMAL] entity:NN(x1) -> that:IN(e1, e2) be:VB(e2, x1, e8) perceive:VB(e3, x3, x1) or:CC(e7, e3, e4) know:VB(e4, x4, x1) or:CC(e8, e7, e5) infer:VB(e5, x5, x1) to:IN(e5, e6) have:VB(e6, x1, x2) own:JJ(x2) distinct:JJ(x2) existence:NN(x2)
[NORMAL] thing:NN(x1) -> separate:JJ(x1) self-contained:JJ(x1) entity:NN(x1)
[GOLD] anything:NN(x1) -> thing:NN(x1) of:IN(x1, x2) any:JJ(x2) kind:NN(x2)
[GOLD] something:NN(x1) -> thing:NN(x1) of:IN(x1, x2) some:JJ(x2) kind:NN(x2)
[SILVER] nothing:NN(x1) -> nonexistent:JJ(x1) thing:NN(x1)

Then, the system randomly selects different LFs to estimate and saves them in different .txt files which I used for the manual evaluation. During the random selection the system considers the qualities of the LFs.
Here above the code for the selection of 50 gold 50 silver 50 normal LFs from the noun file.

APPENDIX 8

SYSTEM F - LHS TAGGING MISTAKES IN THE NOUN FILE

```
import xml.etree.ElementTree as ET
import re

tree=ET.parse("noun.xml")
root=tree.getroot()
countVB=0
countJJ=0
countRB=0
countLF=0

for element in root.iter(tag="lft"):
    countLF=countLF+1
    lf=element.text
    y=lf.split("->")
    lhs=y[0]
    lhspos=re.search(r"\:.*\(", lhs).group()
    pos=lhspos[1:-1]
    if pos == "VB":
        countVB=countVB+1
    elif pos == "JJ":
        countJJ=countJJ+1
    elif pos == "RB":
        countRB=countRB+1
    else:
        pass

print "I checked %d LFs in the noun file and I found: \n %d
wrong VB LHS" %(countLF, countVB), "\n %d wrong JJ LHS"
%countJJ, "\n %d wrong RB LHS" %countRB
```

The output system in the Python Shell:

```
>>>
I checked 94868 LFs in the noun file and I found:
 572 wrong VB LHS
 512 wrong JJ LHS
 227 wrong RB LHS
>>>
```

APPENDIX 9

SYSTEM G - MISSING POSSESSIVE PRONOUNS

```
import xml.etree.ElementTree as ET
import re
root=tree.getroot()
count1=0
listatext=[]
listalf=[]
listatemp=[]
listasynset=[]
count=0
tx=[]
possessives=["its","their","my","our","mine","her","his"]
tag=0
gat=0

for element in root.iter(tag="gloss"):
    count=count+1
lenght=count
x=0

missingpp=open("missingppPOS.txt","a")
while x<lenght:
    text=root[x].find("text").text
    if ";" in text: #it takes only the first definition, no
examples
        tx=text.split(";")
        text1=tx[0]
    else:
        text1=text
    if "(" in text1: #delete content between brackets
        text2=re.sub("\(.*?\)", "", text1)
        listatext=text2.split()
    else:
        listatext=text1.split()
    for n in range(len(listatext)):
        if "," in listatext[n]:
            y=re.sub(",", "", listatext[n])
            listatext[n]=y
        else:
            pass
   lf=root[x].find("lft").text
   count1=count1+1
   lft=re.sub(".*?\-\>", "", lf) #delete word to define and ->
   lfts=re.sub(r"\(.*?\)", "", lft) #delete arguments
   splitlf=lfts.split()
   for eachelement in splitlf:
        wordpos=eachelement.split(":")
        listalf.append(wordpos)
```

```
    for possessive in possessives:
        z=[possessive, 'POS']
        if possessive in listatext and z not in listalf:
            syn=root[x].get("synsetID")
            if syn not in listasynset:
                listasynset.append(syn)
                missingpp.write(syn+" "+possessive+lf[1:]+"\n")
            else:
                pass
   x=x+1
   listalf=[]

missingpp.close()


print "\n I checked %d lf-text pairs there are %d LFs with
missing POS predicates" %(count1, len(listasynset))
```

System Description

Similarly to SystemC for missing conjunctions and SystemD for relative adverbs, SystemG takes as input the XML structure of XWN and for each LF it builds a list of tuples word+pos (predicate without arguments). It then checks each definition (divided in tokens) in XWN: if a possessive pronoun is found in the definition and not in the corresponding LF the missing possessive pronoun case is recorded in a txt file with the structure:

synsetID + missing possessive pronoun+ definition + LF

So, for e.g., for the gloss of *tower*, the system knows that there is a possessive pronoun in the definition

`a structure taller than` **`its`** `diameter`

that is missing in the LF

```
tower:NN(x1) -> structure:NN(x1) tall:JJ(x1) than:IN(x1, x2)
diameter:NN(x2)
```

comparing the tokens of the definitions:

```
['a', 'structure', 'taller', 'than', 'its', 'diameter']
```

with the tuples of the corresponding LF:

```
[['structure', 'NN'], ['tall', 'JJ'], ['than', 'IN'],
                    ['diameter', 'NN']]
```

The system finds the missing case and records in the output file this entry as:

```
04287654 its
a structure taller than its diameter
tower:NN(x1) -> structure:NN(x1) tall:JJ(x1) than:IN(x1, x2)
diameter:NN(x2)
```

At the end of the process SystemG prints the number of checked LFs and the number of LFs with missing possessive pronouns for each pos file.

The final output in the Python Shell:

```
>>>
Noun file

 I checked 79689 lf-text pairs there are 1900 LFs with missing POS predicates
>>> ================================ RESTART ================================
>>>
Verb file

 I checked 13507 lf-text pairs there are 39 LFs with missing POS predicates
>>> ================================ RESTART ================================
>>>
Adjective file

 I checked 18561 lf-text pairs there are 82 LFs with missing POS predicates
>>> ================================ RESTART ================================
>>>
Adverb file

 I checked 3664 lf-text pairs there are 1 LFs with missing POS predicates
```

Excerpt from the output file - adjective file:

00005114 its
being the most comprehensive of its class
comprehensive:JJ(x1) -> be:VB(e1, x1, x2) most:RB(x4) comprehensive:JJ(x4) of:IN(e1, x2)
class:NN(x2)

00051022 its
being or pertaining to something added to a product to increase its value or price
value-added:JJ(x1) -> be:VB(e1, x1) or:CC(e5, e1, e2) pertain:VB(e2, x1) to:IN(e2, x2)
something:NN(x2) add:VB(e3, x9, x2) to:IN(e3, x3) product:NN(x3) to:IN(e3, e4)
increase:VB(e4, x2, x7) value:NN(x4) or:CC(x7, x4, x5) price:NN(x5)

00063893 its
not decorated with something to increase its beauty or distinction

unadorned:JJ(x1) -> not:RB(e1) decorate:VB(e1, x8, x1) with:IN(e1, x2) something:NN(x2) to:IN(x2, e2) increase:VB(e2, x2, x6) beauty:NN(x3) or:CC(x6, x3, x4) distinction:NN(x4)

00160893 their
used of persons or their behavior
unashamed:JJ(x1) -> feel:VB(e1, x1, x2) no:JJ(x2) shame:NN(x2)

00218538 its
supporting no vertical weight other than its own
nonbearing:JJ(x1) -> support:VB(e1, x1, x2) no:JJ(x2) vertical:JJ(x2) weight:NN(x2) other_than:IN(x2, x4) own:JJ(x4)

00430582 their
used of British soldiers during the Revolutionary War because of their red coats
red-coated:JJ(x1) -> use:VB(e1, x8, x1) of:IN(e1, x2) british:JJ(x2) soldier:NN(x2) during:IN(e1, x3) revolutionary:NN(x3) war:NN(x4) because:IN(e1, x5) of:IN(e1, x5) red:JJ(x5) coat:NN(x5)

APPENDIX 10

SYTEM H - COUNTING NEGATIONS (IN DEFINITIONS)

```
import xml.etree.ElementTree as ET
import re

tree=ET.parse("POS.xml")
root=tree.getroot()
countnot=0
count=0
listatext=[]
listalf=[]
tx=[]
negations=["no","not","none","nothing","never","nor"]

for element in root.iter(tag="gloss"):
    count=count+1
lenght=count
x=0

while x<lenght:
    text=root[x].find("text").text
    if ";" in text: #it takes only the firt definition, no
examples
        tx=text.split(";")
        text1=tx[0]
    else:
        text1=text
    if "(" in text1: #delete content between brackets
        text2=re.sub("\(.*?\)", "", text1)
        listatext=text2.split()
    else:
        listatext=text1.split()
    for n in range(len(listatext)):
        if "," in listatext[n]:
            y=re.sub(",", "", listatext[n])
            listatext[n]=y
        else:
            pass
    for negation in negations:
        if negation in listatext:
            countnot=countnot+1
        else:
            pass
    x=x+1


print "I found %d negations in the noun file" %countnot
```

System Description

SystemG divides each definition in tokens and search for negation markings. Eg for the gloss of *ball* :

a pitch that is not in the strike zone; "he threw nine straight balls before the manager yanked him"

the system takes only the definition (*a pitch that is not in the strike zone*) and splits it into tokens:

['a', 'pitch', 'that', 'is', **'not'**, 'in', 'the', 'strike', ‘zone']

It counts *not* as negation marking.
I run the system for each pos XWN file and results are shown (how they appear in the Python Shell) here below:

APPENDIX 11

SYSTEM I - COUNTING MISSING NEGATION MARKINGS

```
import xml.etree.ElementTree as ET
import re

tree=ET.parse("POS.xml")
root=tree.getroot()
count1=0
listatext=[]
listalf=[]
listasynset=[]
count=0
tx=[]
negations=["no","not","none","nothing","never","nor"]
tag=0

for element in root.iter(tag="gloss"):
    count=count+1
lenght=count
x=0

while x<lenght:
    text=root[x].find("text").text
    if ";" in text: #it takes only the firt definition, no
examples
        tx=text.split(";")
        text1=tx[0]
    else:
        text1=text
    if "(" in text1: #delete content between brackets
        text2=re.sub("\(.*?\)", "", text1)
        listatext=text2.split()
    else:
        listatext=text1.split()
    for n in range(len(listatext)):
        if "," in listatext[n]:
            y=re.sub(",", "", listatext[n])
            listatext[n]=y
        else:
            pass
   lf=root[x].find("lft").text
   count1=count1+1
   lft=re.sub(".*?\-\>", "", lf) #delete word to define and ->
   lfts=re.sub("\:[A-Z]*\(.*?\)", "", lft) #delete predic and
arguments
   listalf=lfts.split()
   for negation in negations:
        if negation in listatext and negation not in listalf:
            syn=root[x].get("synsetID")
```

```
            if syn not in listasynset:
                listasynset.append(syn)
            else:
                pass
            print "\n MISSING negation  \n %s" %syn,  "\n
listatext %s"         %listatext, "\n listalf %s" %listalf
        else:
            pass
    x=x+1


print "\n I checked %d lf-text pairs there are %d LFs with
missing negation" %(count1,len(listasynset))
```

System Description

Just like SystemD for relative adverbs, SystemH compares each definition with its corresponding LF. If a Negation Marking is found in the definition and not in the LF the system counts it as missing negation marking and prints out SynsetID of the gloss, definition and LF. E.g. SystemH's output for the gloss of *leakproof*:

MISSING negation
02019151
listatext ['having', 'no', 'leaks']
listalf ['have', 'leak']

This time I didn't ask the system to record the missing cases in a txt file since they are a few and I did not consider them as a common error to correct.
The system counts missing negation marking for each pos XWN file and **the total number is < 50**.
This result proves that XWN LFs are not affected by missing negations.

APPENDIX 12

SYSTEM L - MISSING PREPOSITIONS

```
import xml.etree.ElementTree as ET
import re
tree=ET.parse("POS.xml")
root=tree.getroot()
countnoncontrollate=0
missingprep=0
listatext=[]
listalf=[]
synsetIDlist=[]
count=0
c=0
p=0
tx=[]
cont=0
countcontrollate=0
prepositions=["on","in","to","by","for","with","at","of","from",
"as","out"]
for element in root.iter(tag="gloss"):
    count=count+1
lenght=count
x=0

missprep=open("missingprepPOS.txt","a")

while x<lenght:
    text=root[x].find("text").text
    if ";" in text:#it takes only the first definition, no
examples
        tx=text.split(";")
        text1=tx[0]
    else:
        text1=text
    if "(" in text1:#delete content between brackets
        text2=re.sub("\(.*?\)", "", text1)
        listatext=text2.split()
    else:
        listatext=text1.split()
    for n in range(len(listatext)):
        if "," in listatext[n]:
            y=re.sub(",", "", listatext[n])
            listatext[n]=y
        else:
            pass
    lf=root[x].find("lft").text
    countcontrollate=countcontrollate+1
    lft=re.sub(".*?\-\>", "", lf)#delete word to define and ->
```

```
    lfts=re.sub("\:[A-Z]*\(.*?\)", "", lft)#delete predicates
and arguments
    listalf=lfts.split()
    for token in listatext:
        if token in prepositions:
            p=p+1
            prep=token
            for tok in listalf:
                if tok == prep:
                    cont=cont+1
                elif "_" in tok:
                    z=tok.split("_")
                    for s in z:
                        if s==prep:
                            cont=cont+1
                        else:
                            pass
                elif "-" in tok:
                    z=tok.split("-")
                    for s in z:
                        if s==prep:
                            cont=cont+1
                        else:
                            pass
                else:
                    pass
        else:
            pass
    if p>0 and cont==0:
        syn=root[x].get("synsetID")
        if syn not in synsetIDlist:
            synsetIDlist.append(syn)
            c=c+1
            missprep.write(str(c)+" Missing Preposition: " +
prep + "\n"          + "SynsetID: " + syn + "\n" + "Definition:
          " + text1 +          "\n" + "Logical Form: " + lft +
"\n\n")
        else:
            pass
    else:
        pass
    prep=''
    cont=0
    p=0
    x=x+1
missprep.close()
```

System Description

Just like SystemC for conjunctions, For each XWN pos file, SystemL compares the first defintion-LF pair of each gloss (divided into tokens).
If it founds a preposition in the definition which is not present in the LF, it considers this case as a missing preposition and save the synsetID of the gloss in a txt file (if not already in it).
During the comparison definition-LF the system takes care of those compound words with hyphens or underscores. It splits these words in different tokens and search for the preposition. This is done in order to avoid false positives of missing prepositions which have been transformed into LF as part of a phrasal verb, for e.g for the gloss of *elaborate*.:

```
<gloss pos="VERB" synsetID="00243111">
  <synonymSet>elaborate, work_out</synonymSet>
 <text> work out in detail; "elaborate a plan" </text>
  . . .
 <lft quality="GOLD">
 elaborate:VB(e1, x1, x2) -> work_out:VB(e1, x1, x4) in:IN(e1,
x3) detail:NN(x3) </lft>
</gloss>
```

SystemL checks both part of the word *work_out*.
The system records the results of the analysis in different txt files, one for each XWN pos file, and for each missing case detected, it saves:

preposition missing + SynsetID + Definition + Logical Form

each entry is preceded by a growing number in order to count the cases.
Excerpt from the adjective output file:

220 Missing Preposition: by
SynsetID: 01326135
Definition:
  showing or motivated by sympathy and understanding and generosity
Logical Form:
 show:VB(e1, x1, x6) or:CC(e4, e1, e2) motivate:VB(e2, x6, x1) sympathy:NN(x2) understanding:NN(x3) and:CC(x6, x2, x3, x4) generosity:NN(x4)

…

232 Missing Preposition: out
SynsetID: 01383285
Definition:
  spread out irregularly
Logical Form:
 spread:VB(e1, x4, x1) irregularly:RB(e1)

233 Missing Preposition: out
SynsetID: 01383937
Definition:
  not extended or stretched out
Logical Form:
 not:RB(e3) extend:VB(e1, x4, x1) or:CC(e3, e1, e2) stretch:VB(e2, x5, x1)

APPENDIX 13

THE PARSER

```
:- style_check(-singleton).


% or_CC(x8,x2,x3,x4,x5) - coordination of properties/entities
% or_CC(x6,x2,x3,x4) - coordination of properties/entities
% or_CC(x5,x2,x3) - coordination of properties/entities
% and_CC(e3,e1,e2) - coordination of predicates
% by_IN(e1,x2) - a preposition modifying a predicate and a
property/entity
% of_IN(x2,x3) - a preposition modifying two properties/entities
% to_IN(x6,e5) - a preposition modifying a property/entity and a
predicate (tendency to)
% to_TO(e2,e3) - a particle linking two predicates

% exist_VB(e1,x1), borrow_VB(e2,x1)

%xwn_lookup(File,NewXWn):-
%   consult(File),
sxwn_lookup(File,WN,NewXWn):-
   consult(File),
   readrecursive(Wffs,Preds),
%   tell(errs),
   consult(WN),
   readrecursiveWN(Syns),
   tell(new_xwn),
%   newreadrecursiveWN(Syns),
%   infnewreadrecursiveWN(Syns),
   createnewxwn(Wffs,Syns,Preds,NewXWn),
%   told,
   writeall(NewXWn),
   told,
   !.

writeall([]):-!.
writeall([N-D|Diss]):-
   writenl(_,N-D),nl,
   writeall(Diss),
   !.

readrecursive(Sents,Feats):-
   findall(Pred,lf(Pred,Sent),Sents),
   findall(Sent,lf(Pred,Sent),Feats),
   !.
readrecursiveWN(Sents):-
   findall(ID-Syn,gloss_synsetID(ID,Syn),Sents),
   !.
newreadrecursiveWN(Sents):-
```

```
   findall(Syn-ID,gloss_synsetID(ID,Syn),Sentss),
   sort(Sentss,Sents),
   tell(wnsort),
   writeall(Sents),
   told,
   !.
infnewreadrecursiveWN(Sents):-
   findall(Syn-ID,hypv(ID,Syn),Sentss),
   sort(Sentss,Sents),
   !.

% newcreatexwn(Sent,Feat,Vars,Predss,Out):-


createnewxwn([],Syns,[],[]):-!.
createnewxwn([Ind|Sents],Syns,[Feat|Feats],[Id-LF|NewVit]):-
   length(Feat,L),
%   newcreaxwn(L,Feat,Feat,New),
   newcreaxwns(L,Feat,Feat,New),
   checklexicalpredicates(L,New,Feat,Lexs,Errs),
   transformnew(Feat,Lexs,LF),
%   infnewappendsynset(Ind,Sents,Syns,Rest,Id),
%   newappendsynset(Ind,Sents,Syns,Rest,Id),
   appendsynset(Ind,Sents,Syns,Rest,Id),
%   hyfappendsynset(Ind,Sents,Syns,Rest,Id),
%   creanewxwns(L,Feat,Feat,New),
%   write(Ind),write('  '), write(L), write('  '),
%   write(Errs),nl,
   writeqnl(_,Id-LF),nl,
   createnewxwn(Sents,Rest,Feats,NewVit),
   !.
createnewxwn([Ind|Sents],Syns,Feats,NewVit):-
   createnewxwn(Sents,Syns,Feats,NewVit),
   !.

infnewappendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   (Sy=[[Pre|_]|_]
     ;
    Sy=[[_,Pre|_]|_]
     ;
    Sy=[[PreUp|_]|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    (Sents=[LF|_],Syns=[Sec-_|_],Sec=[[Pre1|_]|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns];Rest=Syns),
    !.
infnewappendsynset(LF,Sents,Syns,Rest,LF1):-
   reconstr1(LF,Pre),
```

```
   infgetsys(Pre,Sents,Syns,ID,Sy,Rest),
   LF1=..[synset,ID,LF,Sy],
    !.
infnewappendsynset(LF,Sents,Syns,Rest,LF1):-
   reconstr1(LF,Pre),
   mcon(Pre,ing,Pre1),
   infgetsys(Pre1,Sents,Syns,ID,Sy,Rest),
   LF1=..[synset,ID,LF,Sy],
    !.
infnewappendsynset(LF,Sents,Syns,Rest,LF1):-
   reconstr1(LF,Pre),
   mcon(Pre,s,Pre1),
   infgetsys(Pre1,Sents,Syns,ID,Sy,Rest),
   LF1=..[synset,ID,LF,Sy],
    !.


infgetsys(Pre,Sents,Syns,ID,Sy,Rest):-
   remove(Sy-ID,Syns,Res),
   (Sy=[[Pre|_]|_]
     ;
    Sy=[[_,Pre|_]|_]
     ;
    Sy=[[PreUp|_]|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    (Sents=[LF|_],Syns=[Sec-_|_],Sec=[[Pre1|_]|_],Pre1\=Pre,
      Rest=[Sy-ID|Res];Rest=Res),
    !.

newappendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   (Sy=[Pre|_]
     ;
    Sy=[_,Pre|_]
     ;
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    (Sents=[LF|_],Syns=[Sec-_|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns];Rest=Syns),
    !.
newappendsynset(LF,Sents,Syns,Rest,LF1):-
   reconstr1(LF,Pre),
   getsys(Pre,Sents,Syns,ID,Sy,Rest),
   LF1=..[synset,ID,LF,Sy],
    !.
```

```
getsys(Pre,Sents,Syns,ID,Sy,Rest):-
   remove(Sy-ID,Syns,Res),
   (Sy=[Pre|_]
     ;
    Sy=[_,Pre|_]
     ;
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    (Sents=[LF|_],Syns=[Sec-_|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Res];Rest=Res),
    !.

hyfappendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   (Sy=[Pre|_]
     ;
    Sy=[_,Pre|_]
     ;
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    LF=..[Pref|_],
    Sents=[LF2|_],
    LF2=..[Pref2|_],
    (LF=LF2,
%      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns]
      ;
     reconstr1(LF2,Pre2),
     tolower(Pre,LowP),
     LowP=Pre2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns]
     ;
     Pref=Pref2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns]
      ;
      Rest=Syns),
    !.
hyfappendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   modifyending(Pre,Pre3),
   (Sy=[Pre3|_]
     ;
    Sy=[_,Pre|_]
     ;
```

```
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    LF=..[Pref|_],
    Sents=[LF2|_],
    LF2=..[Pref2|_],
    (LF=LF2,
%      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
     Rest=[Sy-ID|Syns]
     ;
     reconstr1(LF2,Pre2),
     tolower(Pre,LowP),
     LowP=Pre2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns]
     ;
     Pref=Pref2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[Sy-ID|Syns]
      ;
      Rest=Syns),
    !.
hyfappendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   append(Syns,[Sy-ID],Synss),
   hyfappendsynset(LF,Sents,Synss,Rest,LF1),
   !.

appendsynset(LF,Sents,[ID-Sy|Syns],Rest,LF1):-
%appendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   (Sy=[Pre|_]
     ;
    Sy=[_,Pre|_]
     ;
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    LF=..[Pref|_],
    Sents=[LF2|_],
    LF2=..[Pref2|_],
    (LF=LF2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
      ;
     reconstr1(LF2,Pre2),
     tolower(Pre,LowP),
     LowP=Pre2,
```

```
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
     ;
     Pref=Pref2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
      ;
      Rest=Syns),
    !.
appendsynset(LF,Sents,[ID-Sy|Syns],Rest,LF1):-
%appendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
   reconstr1(LF,Pre),
   modifyending(Pre,Pre3),
   (Sy=[Pre3|_]
     ;
    Sy=[_,Pre|_]
     ;
    Sy=[PreUp|_],
    atomic(PreUp),
    tolower(PreUp,Low),
    Low=Pre),
    LF1=..[synset,ID,LF,Sy],
    LF=..[Pref|_],
    Sents=[LF2|_],
    LF2=..[Pref2|_],
    (LF=LF2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
      ;
     reconstr1(LF2,Pre2),
     tolower(Pre,LowP),
     LowP=Pre2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
     ;
     Pref=Pref2,
      Syns=[_-Sec|_],Sec=[Pre1|_],Pre1\=Pre,
      Rest=[ID-Sy|Syns]
      ;
      Rest=Syns),
    !.
appendsynset(LF,Sents,[ID-Sy|Syns],Rest,LF1):-
%appendsynset(LF,Sents,[Sy-ID|Syns],Rest,LF1):-
%   write(ID),write('  '),write(Sy),nl,nl,
   append(Syns,[Sy-ID],Synss),
   appendsynset(LF,Sents,Synss,Rest,LF1),
   !.

modifyending(Pre,Pre3):-
     stringof(List,Pre),
     reverse(List,[e|Rest]),
```

```
     reverse([g,n,i|Rest],Pre3);
     mcon(Pre,ing,Pre3);
     mcon(Pre,s,Pre3),
     !.

reconstr1(Pre,Pred):-
     Pre=..[Pref|_],
     stringof(List,Pref),
     reverse(List,[A,B,'_'|Rev]),
     reverse(Rev,Lis),
     stringof(Lis,Pred),
     !.

transformnew([],[],[]):-!.
transformnew([LF|Feat],[Pred|Lexs],[NewLF|LFs]):-
            LF=..[Pre|C],
            Pred=Pre-C1,
            NewLF=Pre-C1,
   transformnew(Feat,Lexs,LFs),
   !.
transformnew([LF|Feat],Lexs,[LF|LFs]):-
   transformnew(Feat,Lexs,LFs),
   !.

checklexicalpredicates(L,[],Feat,[],[]):-!.
checklexicalpredicates(L,New,Feat,Out,Errs):-
     findall(Pre-D,(
            member(Pred,Feat),
            Pred=..[Pre,A,B|C],
            stringof(Lis,A),Lis=[e|_],
            D=[A,B|C]),Predd),
            Predd\=[],
     matchverbss(New,Feat,Predd,Out,Errs),
     !.
checklexicalpredicates(L,New,Feat,[],[]):-
     !.

matchverbss(New,Feat,Predd,Out,Errs):-
     matchverbs(New,Predd,Out,Err,Pass),
     append(Err,Pass,All),
     removeall(All,New,Errs),
     !.

removeall(Sent,[],[]):-!.
removeall(Sent,Tops,Rest):-
    remove(W,Sent,ResS),
    remove(W,Tops,ResT),
    removeall(ResS,ResT,Rest),
    !.
removeall(Sent,Tops,Tops):-
    !.
```

```
matchverbs(New,[],[],[],[]):-!.
matchverbs(New,[Pre-Args|Predd],[Pre-Argg|Out],[Var|
Rest],Ergs):-
     reconstr(Pre,Pred),
     v(Pred,Cats),
     member(Var,New),
     member(Var,Args),
     stringof(Lis,Var),Lis=[E|_],E\=e,
     remove(Var,Args,Argg),
     matchverbs(New,Predd,Out,Rest,Ergs),
     !.
matchverbs(New,[Pre-Args|Predd],[Pre-Argg|Out],Ergs,[Var|
Rest]):-
     reconstr(Pre,Pred),
     \+ v(Pred,Cats),
     member(Var,New),
     member(Var,Args),
     stringof(Lis,Var),Lis=[E|_],E\=e,
     remove(Var,Args,Argg),
     matchverbs(New,Predd,Out,Ergs,Rest),
     !.
matchverbs(New,[Pre-Args|Predd],[Pre-Args|Out],Ergs,Rest):-
     matchverbs(New,Predd,Out,Ergs,Rest),
     !.
reconstr(Pre,Pred):-
     stringof(List,Pre),
     reverse(List,[A,B,'_'|Rev]),
     reverse(Rev,Lis),
     stringof(Lis,Pred),
     !.
newcreaxwns(L,Sent,Feat,Out):-
   findall(B,(
            member(Pred,Sent),
            Pred=..['NN'|B]),Predd1),
   findall(B,(
            member(Pred,Sent),
             (Pred=..[or_CC|B];
             Pred=..[and_CC|B])),Predd2),
   findall(D,(
            member(Pred,Sent),
             (Pred=..[Pre,A,B|C],
                        stringof(Lis,A),Lis=[e|_],(A=e1,D=[B|
C];A\=e1,D=[A,B|C]);
             Pred=..[Pre,A,B|C],stringof(Lis,A),Lis=[x|_],
                                          (A=x1,D=[B|
C];A\=x1,D=[A,B|C]))),
             Predd3),
   findall(D,(
            member(Pred,Sent),
             Pred=..[Pre,A,B],D=[A,B]),Predd4),
```

```
   appiattisci(Predd4,Pre4),
   sort(Pre4,Pre44),
   (member(e1,Pre44),remove(e1,Pre44,Pred4);
      Pred4=Pre44),
   append(Pre44,Predd1,Pred11),
   append(Pred11,Predd2,Predd),
   appiattisci(Predd,Preddas),
   sort(Preddas,Predda),
   append(Predd,Predd3,Preds1),
   append(Preds1,Pred4,Preds),
   appiattisci(Preds,Pres),
   sort(Pres,Predss),
   findall(B,(
            (member(Pred,Sent),
             Pred=..[Pre,B],B\=x1,B\=e1,B\=e0)
             ),Varss),
   sort(Varss,Vars),
   newcreatexwn(Sent,Predda,Vars,Predss,Out),
   !.

%evaluateintersect(Rest,Vars,[],Vars):-
%   append(Rest,Vars,Out),
evaluateintersect([],[],[],[]):-!.
evaluateintersect(Rest,Vars,[],Out):-
   write('no intersection'),nl,
   append(Rest,Vars,Out),
   !.
evaluateintersect(Rest,Vars,Outs,Out):-
   Outs\=[],
%   eliminateshared(Vars,Outs,Out),
   eliminateshared(Vars,Outs,Out1),
   eliminateshared(Rest,Outs,Out2),
   append(Out1,Out2,Out),
   !.
eliminateshared(Vars,Outs,Out):-
   member(X,Vars),
   member(X,Outs),
   remove(X,Outs,Rest),
   remove(X,Vars,Var),
   eliminateshared(Var,Rest,Out),
   !.
eliminateshared(Vars,[],Vars):-
   !.
eliminateshared(Vars,Outs,Out):-
   append(Outs,Vars,Out),
   !.
newcreatexwn(Sent,Feat,Vars,Predss,Out):-
   (remove(x1,Predss,Restt);Restt=Predss),
   (remove(e1,Restt,Rest);Rest=Restt),
   append(Feat,Vars,Varss),
   sort(Varss,Varrs),
```

```
   (remove(x1,Varrs,Varr1);Varr1=Varrs),
   (remove(e1,Varr1,Varr);Varr=Varrs),
   intersection(Rest,Varr,Outs),
   evaluateintersect(Rest,Varr,Outs,Out),
   !.
newcreatexwn(Sent,Feat,[x1],Predss,[]):-!.
newcreatexwn(Sent,Feat,Vars,Predss,Out):-
   intersection(Predss,Vars,Outs),
   evaluateintersect(Predss,Vars,Outs,Out),
   !.
newcreatexwn(Sent,Feat,Vars,Predss,Vars):-
   Predss\=[],!.
newcreatexwn(Sent,Feat,Vars,Predss,Vars):-
   Predss=[],\+ member(x1,Vars),!.
newcreatexwn(Sent,Feat,Vars,[],[]):-!.


creanewxwns(L,Sent,Feat,Preds):-
   findall(Pred,(
            member(Pred,Sent),
            Pred=..['NN'|_]),Predd),
   Predd\=[], member(Pred,Predd),
   remove(Pred,Feat,Feat1),
   Pred=..['NN',A|B],
   reifycoord(xm,B,Predss),
   Coord=[coord(xm,A)],
   append(Predss,Coord,NewP),
   append(NewP,Feat1,New),
   creanewxwns(L,Sent,New,Preds),
   !.
creanewxwns(L,Sent,Feat,Preds):-
   findall(Pred,(
            member(Pred,Sent),
             (Pred=..[or_CC,A|B],Pre=or_CC;
             Pred=..[and_CC,A|B],Pre=or_CC)),Predd),
   Predd\=[], member(Pred,Predd),
   remove(Pred,Feat,Feat1),
   (Pred=..[or_CC,A|B],Pre=or_CC;
    Pred=..[and_CC,A|B],Pre=or_CC),
   reifycoord(xc,B,Predss),
   Pres=..[Pre,xc],
   Coord=[coord(xc,A),Pres],
   append(Predss,Coord,NewP),
   append(NewP,Feat1,New),
   creanewxwns(L,Sent,New,Preds),
   !.
/*
creanewxwns(L,[Pred|Sent],Feat,Preds):-
   creanewxwns(L,Sent,Feat,Preds),
   !.
*/
```

```
creanewxwns(L,Sent,Feat,Preds):-
   creanewxwn(L,Feat,Preds),
   !.

reifycoord(First,[],[]):-!.
reifycoord(First,[Var|Pred],[New|Preds]):-
   New=coord(First,Var),
   reifycoord(First,Pred,Preds),
   !.


creanewxwn(L,[],[]):-!.

creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[not_RB,B],
   Pred1=[neg(B,xn),not(xn)],
   append(Pred1,Sent,Sent1),
   !,
   creanewxwn(L,Sent1,Preds),
   !.
creanewxwn(L,[Pred|Sent],[]):-
   Pred=..[A,e1,C],
   checkvariabless([C],Sent,New),
   New=[],
   !.
creanewxwn(L,[Pred|Sent],[]):-
   Pred=..[A,e1,C],
   checkvariabless([C],Sent,New),
   findall(W,(member(Pre,New),Pre=..[W,D1],D=D1),Ws),
   Ws=[],
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,C],
   checkvariabless([C],Sent,New),
   findall(W,(member(Pre,New),Pre=..[W,D1],D=D1),Ws),
   Ws\=[],
   !,
   creanewxwn(L,[Pred|New],Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,C],
   checkvariabless([C],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],[]):-
   Pred=..[A,e1,x1,D],
   checkvariabless([D],Sent,New),
   New=[],
```

```
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,x1,D],
   checkvariabless([D],Sent,New),
   findall(W,(member(Pre,New),Pre=..[W,D1],D=D1),Ws),
   findall(W1,(member(Pre1,New),Pre1=..[W1,B,D1],D=D1),Ws1),
   findall(W2,(member(Pre2,New),Pre2=..[W2,D1,B],D=D1),Ws2),
   Ws=[],Ws1=[],Ws2=[],
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,x1,D],
   checkvariabless([D],Sent,New),
   findall(W,(member(Pre,New),Pre=..[W,D]),Ws),
   findall(W1,(member(Pre1,New),Pre1=..[W1,B,D]),Ws1),
   findall(W2,(member(Pre2,New),Pre2=..[W2,D1,B],D=D1),Ws2),
   append(Ws,Ws1,Wss),
   (Wss\=[];Ws2\=[]),
   !,
   creanewxwn(L,[Pred|New],Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,C,D],
   checkvariabless([C,D],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,x1,D,E],
   checkvariabless([D,E],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,e1,C,D,E],
   checkvariabless([C,D,E],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C],
   checkvariabless([B,C],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C,D],
   checkvariabless([B,C,D],Sent,New),
   !,
   creanewxwn(L,New,Preds),
```

```
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C,D,E],
   checkvariabless([B,C,D,E],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C,D,E,F],
   checkvariables([B,C,D,E,F],Sent,New),
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C,D,E,F,G],
   checkvariabless([B,C,D,E,F,G],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   Pred=..[A,B,C,D,E,F,G,H],
   checkvariabless([B,C,D,E,F,G,H],Sent,New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,[Pred|Sent],[Pred|New]):-
   atomic(Pred),
   creanewxwn(L,Sent,New),
   !.
creanewxwn(L,[Pred|Sent],New):-
   Pred=..[Attr,x1],
   creanewxwn(L,Sent,New),
   !.
creanewxwn(L,[Pred|Sent],Preds):-
   append(Sent,[Pred],New),
   !,
   creanewxwn(L,New,Preds),
   !.
creanewxwn(L,Preds,Preds):-!.


checkvariabless(Vars,Sent,[]):-
   collectallvarss(Sent,AllV),
   findall(Var,(member(Var,Vars),
                \+ member(Var,AllV)),Varr),
   Varr=[],
%   checkvariables(Vars,Sent,Rest),
   !.
checkvariabless(Vars,Sent,Varr):-
   collectallvarss(Sent,AllV),
   findall(Var,(member(Var,Vars),
                \+ member(Var,AllV)),Varr),
```

```
   Varr\=[],
   !.

collectallvarss(Sent,AllVs):-
   collectallvars(Sent,AllV),
   appiattisci(AllV,Alss),
   sort(Alss,AllVs),
   !.

collectallvars([],[]):-!.
collectallvars([Pred|Sent],[Vars|AllV]):-
   Pred=..[Ent|Vars],
   collectallvars(Sent,AllV),
   !.

checkvariable([],Sent,[]):-!.
checkvariable([B],[Pred|Sent],[]):-
   Pred=..[A,B],
   !.
checkvariable([B],[Pred|Sent],[]):-
   Pred=..[A,B|Vars],
   !.
checkvariable([B],[Pred|Sent],[]):-
   Pred=..[A,C,B|Vars],
   !.
checkvariable([B|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B],
   !.
checkvariable([B|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B|Vars],
   !.
checkvariable([C|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B,C|Vars],
   !.
checkvariable([D|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B,C,D|Vars],
   !.
checkvariable([E|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B,C,D,E|Vars],
   !.
checkvariable([F|Rest],[Pred|Sent],Rest):-
   Pred=..[A,B,C,D,E,F|Vars],
   !.


checkvariables([],Sent,[]):-!.
checkvariables([B],[Pred|Sent],Sent):-
   Pred=..[A,B],
   !.
checkvariables([B],[Pred|Sent],Sent):-
   Pred=..[A,B|Vars],
```

```
   !.
checkvariables([B],[Pred|Sent],Sent):-
   Pred=..[A,C,B|Vars],
   !.
checkvariables([B|Rest],[Pred|Sent],New):-
   Pred=..[A,B],
   !,
   checkvariables(Rest,Sent,New),
   !.
checkvariables([B|Rest],[Pred|Sent],New):-
   Pred=..[A,B|Vars],
%   Pred1=..[A|Vars],
   !,
   checkvariables(Rest,[Pred|Sent],New),
   !.
checkvariables([C|Rest],[Pred|Sent],New):-
   Pred=..[A,B,C|Vars],
%   Pred1=..[A,B|Vars],
   !,
   checkvariables(Rest,[Pred|Sent],New),
   !.
checkvariables([D|Rest],[Pred|Sent],New):-
   Pred=..[A,B,C,D|Vars],
%   Pred1=..[A,B,C|Vars],
   !,
   checkvariables(Rest,[Pred|Sent],New),
   !.
checkvariables([E|Rest],[Pred|Sent],New):-
   Pred=..[A,B,C,D,E|Vars],
%   Pred1=..[A,B,C,D|Vars],
   !,
   checkvariables(Rest,[Pred|Sent],New),
   !.
checkvariables([F|Rest],[Pred|Sent],New):-
   Pred=..[A,B,C,D,E,F|Vars],
%   Pred1=..[A,B,C,D,E|Vars],
   !,
   checkvariables(Rest,[Pred|Sent],New),
   !.
checkvariables(Rest,[Pred|Sent],New):-
   Sent=[Pred1|Sent1],
   (Pred=..[A,B],
    checkvariable(Rest,Sent,Vars),
     !,
   checkvariables(Vars,Sent,New)
   ;
%   \+ Pred=..[A,B],
   !,
   append(Sent1,[Pred],Preds),
   checkvariables(Rest,[Pred1|Preds],New)
   ),
```

```
   !.
checkvariables(Rest,Sent,Rest):-
   !.

/*
lf(independent_JJ(x1),
[not_RB(e2),dependent_on_JJ(x4),condition_VB(e1,x5,x1),by_IN(e1,
x5),or_CC(e2,e1),relative_JJ(x2),to_IN(e2,x2),anything_NN(x2),el
se_JJ(x2)]).
lf(independent_JJ(x1),
[not_RB(e3),dependent_on_JJ(e1,x1,x2),condition_NN(x2),by_IN(x2,
x4),or_CC(e3,e1,e2),relative_JJ(e2,x1,e4),to_IN(e4,x4),anything_
NN(x4),else_JJ(x4)]).

*/
```

APPENDIX 14

PARSER - INPUT FILE - NOUN SYNSETID+SYNSET

```
gloss_synsetID(100001740,[entity]).
gloss_synsetID(100002056,[thing]).
gloss_synsetID(100002342,[anything]).
gloss_synsetID(100002452,[something]).
gloss_synsetID(100002560,[nothing,nonentity]).
gloss_synsetID(100002645,[whole,whole_thing,unit]).
gloss_synsetID(100003009,[living_thing,animate_thing]).
gloss_synsetID(100003226,[organism,being]).
gloss_synsetID(100004358,[benthos]).
gloss_synsetID(100004483,[heterotroph]).
gloss_synsetID(100004609,[life]).
gloss_synsetID(100004740,[biont]).
gloss_synsetID(100004824,[cell]).
gloss_synsetID(100005598,[causal_agent,cause,causal_agency]).
gloss_synsetID(100006026,
[person,individual,someone,somebody,mortal,human,soul]).
gloss_synsetID(100012748,
[animal,animate_being,beast,brute,creature,fauna]).
gloss_synsetID(100014510,[plant,flora,plant_life]).
gloss_synsetID(100016236,[object,physical_object]).
gloss_synsetID(100017087,[natural_object]).
gloss_synsetID(100017572,[substance,matter]).
gloss_synsetID(100018827,[food,nutrient]).
gloss_synsetID(100019244,[artifact,artefact]).
gloss_synsetID(100020136,[article]).
gloss_synsetID(100020333,[psychological_feature]).
gloss_synsetID(100020486,[abstraction]).
gloss_synsetID(100020729,[cognition,knowledge,noesis]).
gloss_synsetID(100021213,[motivation,motive,need]).
gloss_synsetID(100021668,[feeling]).
gloss_synsetID(100022625,[location]).
gloss_synsetID(100023103,[shape,form]).
gloss_synsetID(100023548,[time]).
gloss_synsetID(100023929,[space]).
gloss_synsetID(100024197,[absolute_space]).
gloss_synsetID(100024304,[phase_space]).
gloss_synsetID(100024568,[state]).
gloss_synsetID(100025950,[event]).
gloss_synsetID(100026194,[act,human_action,human_activity]).
gloss_synsetID(100026769,[group,grouping]).
gloss_synsetID(100027371,[possession]).
gloss_synsetID(100027563,[attribute]).
gloss_synsetID(100027929,[relation]).
gloss_synsetID(100028549,[social_relation]).
gloss_synsetID(100028764,[communication]).
gloss_synsetID(100029305,[measure,quantity,amount]).
gloss_synsetID(100029881,[phenomenon]).
```

APPENDIX 15

PARSER - INPUT FILE - LOGICAL FORMS

```
entity:NN(x1) -> that:IN(e1, e2) be:VB(e2, x1, e8)
perceive:VB(e3, x3, x1) or:CC(e7, e3, e4) know:VB(e4, x4, x1)
or:CC(e8, e7, e5) infer:VB(e5, x5, x1) to:IN(e5, e6) have:VB(e6,
x1, x2) own:JJ(x2) distinct:JJ(x2) existence:NN(x2)
thing:NN(x1) -> separate:JJ(x1) self-contained:JJ(x1)
entity:NN(x1)
anything:NN(x1) -> thing:NN(x1) of:IN(x1, x2) any:JJ(x2)
kind:NN(x2)
something:NN(x1) -> thing:NN(x1) of:IN(x1, x2) some:JJ(x2)
kind:NN(x2)
nothing:NN(x1) -> nonexistent:JJ(x1) thing:NN(x1)
whole:NN(x1) -> assemblage:NN(x1) of:IN(x1, x2) parts:NN(x2)
be:VB(e1, x2, e2) regard:VB(e2, x4, x2) as:IN(e2, x3)
single:JJ(x3) entity:NN(x3)
living_thing:NN(x1) -> live:VB(e1, x1, x1) entity:NN(x1)
organism:NN(x1) -> live:VB(e1, x1, x1) thing:NN(x1) have:VB(e2,
x1, x2) ability:NN(x2) to:IN(x2, e5) act:VB(e3, x2, x26)
or:CC(e5, e3, e4) function:VB(e4, x2, x26) independently:RB(e4)
benthos:NN(x1) -> organism:NN(x1) live:VB(e1, x1, x26) at:IN(e1,
x4) near:IN(e1, x5) bottom:NN(x2) of:IN(x2, x3) sea:NN(x3)
heterotroph:NN(x1) -> organism:NN(x1) depend:VB(e1, x1, x26)
on:IN(e1, x2) complex:JJ(x2) organic:JJ(x2) substance:NN(x2)
for:IN(x2, x3) nutrition:NN(x3)
life:NN(x1) -> living:JJ(x1) thing:NN(x1) collectively:RB(e1)
biont:NN(x1) -> discrete:JJ(x1) unit:NN(x1) of:IN(x1, x2)
living:JJ(x2) matter:NN(x2)
cell:NN(x1) -> basic:JJ(x1) structural:JJ(x1) functional:JJ(x1)
unit:NN(x1) of:IN(x1, x2) all:JJ(x2) organism:NN(x2)
cell:NN(x1) -> cell:NN(x1) exist:VB(e1, x1, x26) as:IN(e1, x2)
independent:JJ(x2) unit:NN(x2) of:IN(x2, x3) life:NN(x3)
or:CC(e3, e1, e2) form:VB(e2, x1, x8) colony:NN(x4) or:CC(x8,
x4, x5) tissue:NN(x5) as:IN(x8, x9) in:IN(x8, x9) higher:JJ(x9)
plant:NN(x6) and:CC(x9, x6, x7) animal:NN(x7)
causal_agent:NN(x1) -> any:JJ(x1) entity:NN(x1) cause:VB(e1, x1,
x2) event:NN(x2) to:IN(x2, e2) happen:VB(e2, x2, x26)
person:NN(x1) -> human_being:NN(x1)
animal:NN(x1) -> living:JJ(x1) organism:NN(x1) be:VB(e1, x1, x2)
characterized:JJ(x2) by:IN(e1, x3) voluntary:JJ(x3)
movement:NN(x3)
plant:NN(x1) -> living:JJ(x1) organism:NN(x1) lack:VB(e1, x1,
x2) power:NN(x2) of:IN(x2, x3) locomotion:NN(x3)
object:NN(x1) -> tangible:JJ(x1) visible:JJ(x1) entity:NN(x1)
object:NN(x1) -> entity:NN(x1) cast:VB(e1, x1, x2) shadow:NN(x2)
natural_object:NN(x1) -> object:NN(x1) occur:VB(e1, x1, x2)
naturally:RB(e1)
natural_object:NN(x1) -> not:RB(e1) make:VB(e1, x2, x1)
man:NN(x2)
```

APPENDIX 16

SYSTEM M - PNs ERRORS

```
fstr = open("Synsets.txt","r").read()
frasi=fstr.split()
lunghezza=len(frasi)
prova=""
x=0
y=1
lista=[]
termini=[]
while y<lunghezza:
    frase=frasi[x].split(",")
    check=frasi[y].split(",")
    if frase[0]!=check[0] and frase[1]==check[1]:
        errori=open("SynsetsSegnati.txt","a")
        errori.write("*"+frasi[x]+"\n")
        errori.close()
        if frase[1].lower() not in lista: #the list is
lowercased
            lista.append(frase[1].lower())
        else:
            lista=lista
    else:
        errori=open("SynsetsSegnati.txt","a")
        errori.write(frasi[x]+"\n")
        errori.close()
    x=x+1
    y=y+1
for i in lista:
    termini.append(i[2:-1]) #cleaning the list from brackets
for w in range(len(termini)):
    if "']" in termini[w]:
        d=list(termini[w])
        d.remove("'")
        d.remove("]")
        c=''.join(d)
        termini[w]=c
    else:
        pass
for w in termini:
    l=open("ListaTermini.txt","a")
    l.write(w+"\n")
    l.close()
#now the LFs
f = open("LF.txt","r").read()
fr=f.split()
lenght=len(fr)
a=0
count=0
```

```
while a<lenght:
    s=fr[a].split("(")
    d=s[1]
    if d.count("_")>1:
        #es meyer_guggenheim_NN
        test=d.split("_")
        for w in test:
            if w in termini:
                count=count+1
            else:
                count=count
        if count>0:
            lfsegnate=open("LFPROVA.txt","a")
            lfsegnate.write("*"+fr[a]+"\n")
            lfsegnate.close()
        else:
            lfsegnate=open("LFPROVA.txt","a")
            lfsegnate.write(fr[a]+"\n")
            lfsegnate.close()
    else:
        #es guarneri_NN
        if d[:-3] in termini:
            lfsegnate=open("LFPROVA.txt","a")
            lfsegnate.write("*"+fr[a]+"\n")
            lfsegnate.close()
        else:
            lfsegnate=open("LFPROVA.txt","a")
            lfsegnate.write(fr[a]+"\n")
            lfsegnate.close()
    count=0
    a=a+1
```

APPENDIX 17

SYNSETID+SYNSET INPUT FILE - PNs

```
gloss_synsetID(110441921,
['Mayakovski','Vladimir_Vladimirovich_Mayakovski'])
gloss_synsetID(110442065,
['Mayer','Louis_B_Mayer','Louis_Burt_Mayer'])
gloss_synsetID(110442275,['Mayer','Marie_Goeppert_Mayer'])
gloss_synsetID(110442455,
['Mays','Willie_Mays','Willie_Howard_Mays_Jr_','the_Say_Hey_Kid'
])
gloss_synsetID(110442607,['Mazzini','Giuseppe_Mazzini'])
gloss_synsetID(110442783,
['McCarthy','Joseph_McCarthy','Joseph_Raymond_McCarthy'])
gloss_synsetID(110442979,
['McCarthy','Mary_McCarthy','Mary_Therese_McCarthy'])
gloss_synsetID(110443138,
['McCartney','Paul_McCartney','Sir_James_Paul_McCartney'])
gloss_synsetID(110443402,
['McCauley','Mary_McCauley','Mary_Ludwig_Hays_McCauley','Molly_P
itcher'])
gloss_synsetID(110443696,['McCormick','John_McCormick'])
gloss_synsetID(110443827,
['McCormick','Cyrus_McCormick','Cyrus_Hall_McCormick'])
gloss_synsetID(110444019,
['McCullers','Carson_McCullers','Carson_Smith_McCullers'])
gloss_synsetID(110444153,
['McGraw','John_McGraw','John_Joseph_McGraw'])
gloss_synsetID(110444312,['McGuffey','William_Holmes_McGuffey'])
gloss_synsetID(110444470,['McKim','Charles_Follen_McKim'])
gloss_synsetID(110444593,
['McKinley','William_McKinley','President_McKinley'])
gloss_synsetID(110444769,
['McLuhan','Marshall_McLuhan','Herbert_Marshall_McLuhan'])
gloss_synsetID(110444937,['McMaster','John_Bach_McMaster'])
gloss_synsetID(110445115,['McPherson','Aimee_Semple_McPherson'])
gloss_synsetID(110445294,['Mead','George_Herbert_Mead'])
gloss_synsetID(110445418,['Mead','Margaret_Mead'])
gloss_synsetID(110445607,['Meade','George_Gordon_Meade'])
gloss_synsetID(110445772,['Meade','James_Edward_Meade'])
gloss_synsetID(110445934,['Meany','George_Meany'])
gloss_synsetID(110446082,
['Medawar','Peter_Medawar','Sir_Peter_Brian_Medawar'])
gloss_synsetID(110446325,['Meiji_Tenno','Mutsuhito'])
gloss_synsetID(110446465,['Meir','Golda_Meir'])
gloss_synsetID(110446575,['Meissner','Fritz_W_Meissner'])
gloss_synsetID(110446678,['Meissner','Georg_Meissner'])
gloss_synsetID(110446778,['Meitner','Lise_Meitner'])
gloss_synsetID(110447014,
['Melanchthon','Philipp_Melanchthon','Philipp_Schwarzerd'])
```

```
gloss_synsetID(110447207,
[‘Melba','Dame_Nellie_Melba','Helen_Porter_Mitchell'])
gloss_synsetID(110447342,['Melchior'])
gloss_synsetID(110447571,
['Melchior','Lauritz_Melchior','Lauritz_Lebrecht_Hommel_Melchior
'])
gloss_synsetID(110447768,
['Mellon','Andrew_Mellon','Andrew_W_Mellon','Andrew_William_Mell
on'])
gloss_synsetID(110447952,['Melville','Herman_Melville'])
gloss_synsetID(110448085,['Menander'])
gloss_synsetID(110448186,
['Mencken','H_L_Mencken','Henry_Louis_Mencken'])
gloss_synsetID(110448352,
['Mendel','Gregor_Mendel','Johann_Mendel'])
gloss_synsetID(110448608,
['Mendeleyev','Mendeleev','Dmitri_Mendeleyev','Dmitri_Mendeleev'
,'Dmitri_Ivanovich_Mendeleyev','Dmitri_Ivanovich_Mendeleev'])
gloss_synsetID(110448910,['Mendelsohn','Erich_Mendelsohn'])
gloss_synsetID(110449048,
['Mendelssohn','Felix_Mendelssohn','Jakob_Ludwig_Felix_Mendelsso
hn-Bartholdy'])
gloss_synsetID(110449249,['Meniere','Prosper_Meniere'])
gloss_synsetID(110449483,
['Menninger','Charles_Menninger','Charles_Frederick_Menninger'])
gloss_synsetID(110449691,
['Menninger','Karl_Menninger','Karl_Augustus_Menninger'])
gloss_synsetID(110449857,
['Menninger','William_Menninger','William_Claire_Menninger'])
gloss_synsetID(110450027,['Menotti','Gian_Carlo_Menotti'])
gloss_synsetID(110450165,
['Menuhin','Yehudi_Menuhin','Sir_Yehudi_Menuhin'])
gloss_synsetID(110450367,
['Mercator','Gerardus_Mercator','Gerhard_Kremer'])
gloss_synsetID(110450568,['Mercer','John_Mercer'])
gloss_synsetID(110450727,['Merckx','Eddy_Merckx'])
gloss_synsetID(110450869,
['Mercouri','Melina_Mercouri','Anna_Amalia_Mercouri'])
gloss_synsetID(110450995,['Meredith','George_Meredith'])
gloss_synsetID(110451123,
['Meredith','James_Meredith','James_Howard_Meredith'])
gloss_synsetID(110451347,['Mergenthaler','Ottmar_Mergenthaler'])
gloss_synsetID(110451504,[‘Merlin'])
gloss_synsetID(110451642,['Merman','Ethel_Merman'])
gloss_synsetID(110451783,
['Merton','Robert_Merton','Robert_King_Merton'])
gloss_synsetID(110451910,['Merton','Thomas_Merton'])
gloss_synsetID(110452043,
['Mesmer','Franz_Anton_Mesmer','Friedrich_Anton_Mesmer'])
```

APPENDIX 18

SYNSETID+SYNSET OUTPUT FILE - PNs

```
gloss_synsetID(110441921,
['Mayakovski','Vladimir_Vladimirovich_Mayakovski'])
*gloss_synsetID(110442065,
['Mayer','Louis_B_Mayer','Louis_Burt_Mayer'])
gloss_synsetID(110442275,['Mayer','Marie_Goeppert_Mayer'])
gloss_synsetID(110442455,
['Mays','Willie_Mays','Willie_Howard_Mays_Jr_','the_Say_Hey_Kid'
])
gloss_synsetID(110442607,['Mazzini','Giuseppe_Mazzini'])
*gloss_synsetID(110442783,
['McCarthy','Joseph_McCarthy','Joseph_Raymond_McCarthy'])
gloss_synsetID(110442979,
['McCarthy','Mary_McCarthy','Mary_Therese_McCarthy'])
gloss_synsetID(110443138,
['McCartney','Paul_McCartney','Sir_James_Paul_McCartney'])
gloss_synsetID(110443402,
['McCauley','Mary_McCauley','Mary_Ludwig_Hays_McCauley','Molly_P
itcher'])
*gloss_synsetID(110443696,['McCormick','John_McCormick'])
gloss_synsetID(110443827,
['McCormick','Cyrus_McCormick','Cyrus_Hall_McCormick'])
gloss_synsetID(110444019,
['McCullers','Carson_McCullers','Carson_Smith_McCullers'])
gloss_synsetID(110444153,
['McGraw','John_McGraw','John_Joseph_McGraw'])
gloss_synsetID(110444312,['McGuffey','William_Holmes_McGuffey'])
gloss_synsetID(110444470,['McKim','Charles_Follen_McKim'])
gloss_synsetID(110444593,
['McKinley','William_McKinley','President_McKinley'])
gloss_synsetID(110444769,
['McLuhan','Marshall_McLuhan','Herbert_Marshall_McLuhan'])
gloss_synsetID(110444937,['McMaster','John_Bach_McMaster'])
gloss_synsetID(110445115,['McPherson','Aimee_Semple_McPherson'])
*gloss_synsetID(110445294,['Mead','George_Herbert_Mead'])
gloss_synsetID(110445418,['Mead','Margaret_Mead'])
*gloss_synsetID(110445607,['Meade','George_Gordon_Meade'])
gloss_synsetID(110445772,['Meade','James_Edward_Meade'])
gloss_synsetID(110445934,['Meany','George_Meany'])
gloss_synsetID(110446082,
['Medawar','Peter_Medawar','Sir_Peter_Brian_Medawar'])
gloss_synsetID(110446325,['Meiji_Tenno','Mutsuhito'])
gloss_synsetID(110446465,['Meir','Golda_Meir'])
*gloss_synsetID(110446575,['Meissner','Fritz_W_Meissner'])
gloss_synsetID(110446678,['Meissner','Georg_Meissner'])
gloss_synsetID(110446778,['Meitner','Lise_Meitner'])
```

```
gloss_synsetID(110447014,
['Melanchthon','Philipp_Melanchthon','Philipp_Schwarzerd'])
gloss_synsetID(110447207,
['Melba','Dame_Nellie_Melba','Helen_Porter_Mitchell'])
gloss_synsetID(110447342,['Melchior'])
gloss_synsetID(110447571,
['Melchior','Lauritz_Melchior','Lauritz_Lebrecht_Hommel_Melchior
'])
gloss_synsetID(110447768,
['Mellon','Andrew_Mellon','Andrew_W_Mellon','Andrew_William_Mell
on'])
gloss_synsetID(110447952,['Melville','Herman_Melville'])
gloss_synsetID(110448085,['Menander'])
gloss_synsetID(110448186,
['Mencken','H_L_Mencken','Henry_Louis_Mencken'])
gloss_synsetID(110448352,
['Mendel','Gregor_Mendel','Johann_Mendel'])
gloss_synsetID(110448608,
['Mendeleyev','Mendeleev','Dmitri_Mendeleyev','Dmitri_Mendeleev'
,'Dmitri_Ivanovich_Mendeleyev','Dmitri_Ivanovich_Mendeleev'])
gloss_synsetID(110448910,['Mendelsohn','Erich_Mendelsohn'])
gloss_synsetID(110449048,
['Mendelssohn','Felix_Mendelssohn','Jakob_Ludwig_Felix_Mendelsso
hn-Bartholdy'])
gloss_synsetID(110449249,['Meniere','Prosper_Meniere'])
*gloss_synsetID(110449483,
['Menninger','Charles_Menninger','Charles_Frederick_Menninger'])
*gloss_synsetID(110449691,
['Menninger','Karl_Menninger','Karl_Augustus_Menninger'])
gloss_synsetID(110449857,
['Menninger','William_Menninger','William_Claire_Menninger'])
gloss_synsetID(110450027,['Menotti','Gian_Carlo_Menotti'])
gloss_synsetID(110450165,
['Menuhin','Yehudi_Menuhin','Sir_Yehudi_Menuhin'])
gloss_synsetID(110450367,
['Mercator','Gerardus_Mercator','Gerhard_Kremer'])
gloss_synsetID(110450568,['Mercer','John_Mercer'])
gloss_synsetID(110450727,['Merckx','Eddy_Merckx'])
gloss_synsetID(110450869,
['Mercouri','Melina_Mercouri','Anna_Amalia_Mercouri'])
*gloss_synsetID(110450995,['Meredith','George_Meredith'])
gloss_synsetID(110451123,
['Meredith','James_Meredith','James_Howard_Meredith'])
gloss_synsetID(110451347,['Mergenthaler','Ottmar_Mergenthaler'])
gloss_synsetID(110451504,['Merlin'])
gloss_synsetID(110451642,['Merman','Ethel_Merman'])
*gloss_synsetID(110451783,
['Merton','Robert_Merton','Robert_King_Merton'])
gloss_synsetID(110451910,['Merton','Thomas_Merton'])
```

APPENDIX 19

EXTRACT FROM THE OUTPUT - LIST OF PNs

```
[…]
marx
mason
mayer
mccarthy
mccormick
mead
meade
meissner
menninger
meredith
merton
mill
miller
mitchell
mitford
monroe
montgolfier
montgomery
moody
moore
morgan
morris
morrison
muller
murdoch
murray
newman
niebuhr
noguchi
norman
oates
owen
page
paine
parker
parkinson
paul
peirce
percy
perry
philemon
philip_ii
pitt
pliny
porter
[…]
```

APPENDIX 20

LF INPUT FILE - PNs

```
lf(mayakovski_NN(x1),[soviet_JJ(x1),poet_NN(x1)])
lf(mayakovski_NN(x1),
[leader_NN(x1),of_IN(x1,x2),russian_JJ(x2),futurism_NN(x2)])
lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),filmmaker_NN(x3),found_VB(e1,x1,x4),
own_JJ(x4),film_NN(x4),company_NN(x5),and_CC(e3,e1,e2),later_RB(
e2),merge_VB(e2,x8,x1),with_IN(e2,x6),samuel_NN(x6),goldwyn_NN(x
7)])
lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),physicist_NN(x3),note_VB(e1,x7,x1),f
or_IN(e1,x4),research_NN(x4),on_IN(x4,x5),structure_NN(x5),of_IN
(x5,x6),atom_NN(x6)])
lf(mays_NN(x1),
[united_NN(x1),state_NN(x2),baseball_NN(x3),player_NN(x4)])
lf(mazzini_NN(x1),
[italian_JJ(x4),nationalist_JJ(x5),writings_NN(x1),spur_VB(e1,x6
,x1),movement_NN(x2),for_IN(x2,x3),unified_JJ(x3),independent_JJ
(x3),italy_NN(x3)])
lf(mccarthy_NN(x1),
[united_NN(x1),state_NN(x2),politician_NN(x3),unscrupulously_RB(
e1),accuse_VB(e1,x1,x4),many_JJ(x4),citizen_NN(x4),of_IN(e1,x5),
communist_NN(x5)])
lf(mccarthy_NN(x1),
[united_NN(x2),state_NN(x3),satirical_JJ(x1),novelist_NN(x4),and
_CC(x1,x2,x3,x4,x5),literary_JJ(x1),critic_NN(x5)])
lf(mccartney_NN(x1),
[english_NN(x2),rock_NN(x3),star_NN(x4),bass_NN(x5),guitarist_NN
(x6),and_CC(x1,x2,x3,x4,x5,x6,x7),songwriter_NN(x7),with_IN(x1,x
8),john_NN(x8),lennon_NN(x9),write_VB(e1,x8,x10),most_JJ(x12),of
_IN(e1,x10),music_NN(x10),for_IN(x10,x11),beatles_NN(x11)])
lf(mccauley_NN(x1),
[heroine_NN(x1),of_IN(x1,x2),american_NN(x2),revolution_NN(x3),c
arry_VB(e1,x2,x4),water_NN(x4),to_IN(e1,x5),soldier_NN(x5),durin
g_IN(e1,x6),battle_NN(x6),of_IN(x6,x7),monmouth_NN(x7),court_NN(
x8),house_NN(x9),and_CC(e5,e1,e2),take_VB(e2,x2,e3),over_IN(e2,x
10),'husband-
s_NN'(x10),gun_NN(x11),be_VB(e3,x2,e4),overcome_VB(e4,x12,x2),by
_IN(e4,x12),heat_NN(x12)])
lf(mccormick_NN(x1),
[united_NN(x1),state_NN(x2),operatic_JJ(x1),tenor_NN(x3)])
lf(mccormick_NN(x1),
[united_NN(x2),state_NN(x3),inventor_NN(x4),and_CC(x1,x2,x5),man
ufacturer_NN(x5),of_IN(x1,x6),mechanical_JJ(x6),harvester_NN(x6)
])
lf(mccullers_NN(x1),
[united_NN(x1),state_NN(x2),novelist_NN(x3)])
```

```
lf(mcgraw_NN(x1),
[united_NN(x2),state_NN(x3),baseball_NN(x4),player_NN(x5),and_CC
(x1,x2,x6),manager_NN(x6)])
lf(mcguffey_NN(x1),
[united_NN(x1),state_NN(x2),educator_NN(x3),compile_VB(e1,x1,x4)
,mcguffey_NN(x4),eclectic_JJ(x4),reader_NN(x5)])
lf(mckim_NN(x1),
[united_NN(x1),state_NN(x2),neoclassical_JJ(x1),architect_NN(x3)
])
lf(mckinley_NN(x1),
['25th_JJ'(x1),president_of_the_united_states_NN(x1)])
lf(mckinley_NN(x1),[assassinate_VB(e1,x2,x1),anarchist_NN(x2)])
lf(mcluhan_NN(x1),
[canadian_JJ(x1),writer_NN(x1),note_VB(e1,x4,x1),for_IN(e1,x2),h
is_POS(x2,x1),analysis_NN(x2),of_IN(x2,x3),mass_media_NN(x3)])
lf(mcmaster_NN(x1),
[united_NN(x1),state_NN(x2),historian_NN(x3),write_VB(e1,x1,x4),
nine_JJ(x4),volume_NN(x4),history_NN(x5),of_IN(x4,x6),people_NN(
x6),of_IN(x4,x7),united_NN(x7),state_NN(x8)])
lf(mcpherson_NN(x1),
[united_NN(x1),state_NN(x2),evangelist_NN(x3),note_VB(e1,x5,x1),
for_IN(e1,x4),extravagant_JJ(x4),religious_JJ(x4),services_NN(x4
)])
lf(mead_NN(x1),
[united_NN(x1),state_NN(x2),philosopher_NN(x3),of_IN(x1,x4),prag
matism_NN(x4)])
lf(mead_NN(x1),
[be_VB(e1,x1,e2),milkweed_VB(e2,x4,x1),of_IN(e2,x1),central_NN(x
1),north_NN(x2),america_NN(x3)])
lf(mead_NN(x1),
[be_VB(e1,x1,e2),threaten_VB(e2,x2,x1),species_NN(x1)])
lf(mead_NN(x1),
[united_NN(x1),state_NN(x2),anthropologist_NN(x3),note_VB(e1,x10
,x1),for_IN(e1,x4),claim_NN(x4),about_IN(x1,x9),adolescence_NN(x
5),and_CC(x9,x5,x6),sexual_JJ(x9),behavior_NN(x6),in_IN(x9,x7),p
olynesian_NN(x7),culture_NN(x8)])
lf(meade_NN(x1),
[united_NN(x1),state_NN(x2),general_JJ(x7),in_IN(x1,x3),charge_N
N(x3),of_IN(x3,x4),union_NN(x4),troops_NN(x5),at_IN(x4,x6),battl
e_of_gettysburg_NN(x6)])
lf(meade_NN(x1),
[english_NN(x1),economist_NN(x2),note_VB(e1,x7,x1),for_IN(e1,x3)
,study_NN(x3),of_IN(x3,x6),international_JJ(x6),trade_NN(x4),and
_CC(x6,x4,x5),finance_NN(x5)])
lf(meany_NN(x1),
[united_NN(x1),state_NN(x2),labor_NN(x3),leader_NN(x4),be_VB(e1,
x1,x5),first_JJ(x5),president_NN(x5),of_IN(x5,x6),'afl-
cio_NN'(x6)])
```

APPENDIX 21

LF OUTPUT FILE - PNs

```
lf(mayakovski_NN(x1),[soviet_JJ(x1),poet_NN(x1)])
lf(mayakovski_NN(x1),
[leader_NN(x1),of_IN(x1,x2),russian_JJ(x2),futurism_NN(x2)])
*lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),filmmaker_NN(x3),found_VB(e1,x1,x4),
own_JJ(x4),film_NN(x4),company_NN(x5),and_CC(e3,e1,e2),later_RB(
e2),merge_VB(e2,x8,x1),with_IN(e2,x6),samuel_NN(x6),goldwyn_NN(x
7)])
*lf(mayer_NN(x1),
[united_NN(x1),state_NN(x2),physicist_NN(x3),note_VB(e1,x7,x1),f
or_IN(e1,x4),research_NN(x4),on_IN(x4,x5),structure_NN(x5),of_IN
(x5,x6),atom_NN(x6)])
lf(mays_NN(x1),
[united_NN(x1),state_NN(x2),baseball_NN(x3),player_NN(x4)])
lf(mazzini_NN(x1),
[italian_JJ(x4),nationalist_JJ(x5),writings_NN(x1),spur_VB(e1,x6
,x1),movement_NN(x2),for_IN(x2,x3),unified_JJ(x3),independent_JJ
(x3),italy_NN(x3)])
*lf(mccarthy_NN(x1),
[united_NN(x1),state_NN(x2),politician_NN(x3),unscrupulously_RB(
e1),accuse_VB(e1,x1,x4),many_JJ(x4),citizen_NN(x4),of_IN(e1,x5),
communist_NN(x5)])
*lf(mccarthy_NN(x1),
[united_NN(x2),state_NN(x3),satirical_JJ(x1),novelist_NN(x4),and
_CC(x1,x2,x3,x4,x5),literary_JJ(x1),critic_NN(x5)])
lf(mccartney_NN(x1),
[english_NN(x2),rock_NN(x3),star_NN(x4),bass_NN(x5),guitarist_NN
(x6),and_CC(x1,x2,x3,x4,x5,x6,x7),songwriter_NN(x7),with_IN(x1,x
8),john_NN(x8),lennon_NN(x9),write_VB(e1,x8,x10),most_JJ(x12),of
_IN(e1,x10),music_NN(x10),for_IN(x10,x11),beatles_NN(x11)])
lf(mccauley_NN(x1),
[heroine_NN(x1),of_IN(x1,x2),american_NN(x2),revolution_NN(x3),c
arry_VB(e1,x2,x4),water_NN(x4),to_IN(e1,x5),soldier_NN(x5),durin
g_IN(e1,x6),battle_NN(x6),of_IN(x6,x7),monmouth_NN(x7),court_NN(
x8),house_NN(x9),and_CC(e5,e1,e2),take_VB(e2,x2,e3),over_IN(e2,x
10),'husband-
s_NN'(x10),gun_NN(x11),be_VB(e3,x2,e4),overcome_VB(e4,x12,x2),by
_IN(e4,x12),heat_NN(x12)])
*lf(mccormick_NN(x1),
[united_NN(x1),state_NN(x2),operatic_JJ(x1),tenor_NN(x3)])
*lf(mccormick_NN(x1),
[united_NN(x2),state_NN(x3),inventor_NN(x4),and_CC(x1,x2,x5),man
ufacturer_NN(x5),of_IN(x1,x6),mechanical_JJ(x6),harvester_NN(x6)
])
lf(mccullers_NN(x1),
[united_NN(x1),state_NN(x2),novelist_NN(x3)])
```

```
lf(mcgraw_NN(x1),
[united_NN(x2),state_NN(x3),baseball_NN(x4),player_NN(x5),and_CC
(x1,x2,x6),manager_NN(x6)])
lf(mcguffey_NN(x1),
[united_NN(x1),state_NN(x2),educator_NN(x3),compile_VB(e1,x1,x4)
,mcguffey_NN(x4),eclectic_JJ(x4),reader_NN(x5)])
lf(mckim_NN(x1),
[united_NN(x1),state_NN(x2),neoclassical_JJ(x1),architect_NN(x3)
])
lf(mckinley_NN(x1),
['25th_JJ'(x1),president_of_the_united_states_NN(x1)])
lf(mckinley_NN(x1),[assassinate_VB(e1,x2,x1),anarchist_NN(x2)])
lf(mcluhan_NN(x1),
[canadian_JJ(x1),writer_NN(x1),note_VB(e1,x4,x1),for_IN(e1,x2),h
is_POS(x2,x1),analysis_NN(x2),of_IN(x2,x3),mass_media_NN(x3)])
lf(mcmaster_NN(x1),
[united_NN(x1),state_NN(x2),historian_NN(x3),write_VB(e1,x1,x4),
nine_JJ(x4),volume_NN(x4),history_NN(x5),of_IN(x4,x6),people_NN(
x6),of_IN(x4,x7),united_NN(x7),state_NN(x8)])
lf(mcpherson_NN(x1),
[united_NN(x1),state_NN(x2),evangelist_NN(x3),note_VB(e1,x5,x1),
for_IN(e1,x4),extravagant_JJ(x4),religious_JJ(x4),services_NN(x4
)])
*lf(mead_NN(x1),
[united_NN(x1),state_NN(x2),philosopher_NN(x3),of_IN(x1,x4),prag
matism_NN(x4)])
*lf(mead_NN(x1),
[be_VB(e1,x1,e2),milkweed_VB(e2,x4,x1),of_IN(e2,x1),central_NN(x
1),north_NN(x2),america_NN(x3)])
*lf(mead_NN(x1),
[be_VB(e1,x1,e2),threaten_VB(e2,x2,x1),species_NN(x1)])
*lf(mead_NN(x1),
[united_NN(x1),state_NN(x2),anthropologist_NN(x3),note_VB(e1,x10
,x1),for_IN(e1,x4),claim_NN(x4),about_IN(x1,x9),adolescence_NN(x
5),and_CC(x9,x5,x6),sexual_JJ(x9),behavior_NN(x6),in_IN(x9,x7),p
olynesian_NN(x7),culture_NN(x8)])
*lf(meade_NN(x1),
[united_NN(x1),state_NN(x2),general_JJ(x7),in_IN(x1,x3),charge_N
N(x3),of_IN(x3,x4),union_NN(x4),troops_NN(x5),at_IN(x4,x6),battl
e_of_gettysburg_NN(x6)])
*lf(meade_NN(x1),
[english_NN(x1),economist_NN(x2),note_VB(e1,x7,x1),for_IN(e1,x3)
,study_NN(x3),of_IN(x3,x6),international_JJ(x6),trade_NN(x4),and
_CC(x6,x4,x5),finance_NN(x5)])
lf(meany_NN(x1),
[united_NN(x1),state_NN(x2),labor_NN(x3),leader_NN(x4),be_VB(e1,
x1,x5),first_JJ(x5),president_NN(x5),of_IN(x5,x6),'afl-
cio_NN'(x6)])
```

APPENDIX 22

SYSTEM N - DUPLICATES LFs

```
import re
output=open("output_POS.txt").readlines()
daeliminare=[]
listatuple=[]
listadoppie=[]
count1=0
count2=0

for x in output:
    temp=x.split(")-[")
    y=temp[0]
    sy=re.search(r"\([0-9]*\,", y)
    synum=sy.group()
    num=synum[1:-1]
    w=re.search(num+r"\, .*\, \[", y)
    wo=w.group()
    word=wo[11:-3]
    z=temp[1]
    lf=z[:-2]
#make a list of tuples synsID+word+logicform
    listatuple.append([num,word,lf])
for i in range(len(listatuple)):
    a=listatuple[i]
    for n in range(len(listatuple)):
        b=listatuple[n]
        if i==n:
            pass
        else:
            if a[0]==b[0] and a[1]==b[1] and a[2]==b[2]:
                #Case1
                daeliminare.append(b)
                count1=count1+1
            elif a[0]!=b[0] and a[1]==b[1] and a[2]==b[2]:
                #Case2
                listadoppie.append(b)
                count2=count2+1

sorted_listadoppie=sorted(listadoppie, key=lambda tup: tup[1])
sorted_daeliminare=sorted(daeliminare, key=lambda tup: tup[1])
lfdoppie=open("POS_Case1.txt", "a")
for x in sorted_listadoppie:
    resul=' / '.join(x)
    lfdoppie.write(resul+"\n")
lfdoppie.close()
copie=open("POS_Case2.txt", "a")
for x in sorted_daeliminare:
    resul=' / '.join(x)
```

```
    copie.write(resul+"\n")
copie.close()

print "In POS I found \n %d of Case1 \n" %count1
print "In POS I found \n %d of Case2 \n" %count2
```

APPENDIX 23

OUTPUT OF SYSTEM L - ADJECTIVES

CASE 1 (Same SynsetID, same lemma, same LF)

302269787 / based_JJ(x1) / have_VB(e1, x1, x2), base_NN(x2)
302061319 / based_JJ(x1) / have_VB(e1, x1, x2), base_NN(x2)
301845387 / clean_JJ(x1) / free_JJ(x1), from_IN(x1, x2), impurity_NN(x2)
300392688 / clean_JJ(x1) / free_JJ(x1), from_IN(x1, x2), impurity_NN(x2)
300928036 / common_JJ(x1) / commonly_RB(e1), encounter_VB(e1, x1)
300458805 / common_JJ(x1) / commonly_RB(e1), encounter_VB(e1, x1)
302600938 / cytotoxic_JJ(x1) / of_IN(x1, x2), relate_VB-[e1, x1], to_IN-[e1, x2], substance_NN(x2), be_VB-[e2, x2, x3], toxic_JJ(x3), to_IN(x3, x4), cell_NN(x4)
302361205 / cytotoxic_JJ(x1) / of_IN(x1, x2), relate_VB-[e1, x1], to_IN-[e1, x2], substance_NN(x2), be_VB-[e2, x2, x3], toxic_JJ(x3), to_IN(x3, x4), cell_NN(x4)
301835744 / disarming_JJ(x1) / capable_JJ(x1), of_IN(x1, e1), allay_VB(e1, x1, x2), hostility_NN(x2)
301747662 / disarming_JJ(x1) / capable_JJ(x1), of_IN(x1, e1), allay_VB(e1, x1, x2), hostility_NN(x2)
301506903 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301506903 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301139129 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301139129 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301795613 / grassroots_JJ(x1) / fundamental_JJ(x1)
300460273 / grassroots_JJ(x1) / fundamental_JJ(x1)
302353520 / lidded_JJ(x1) / have_VB(e1, x1, x2), lid_NN(x2)
301397581 / lidded_JJ(x1) / have_VB(e1, x1, x2), lid_NN(x2)
301151103 / light-armed_JJ(x1) / armed_JJ(x1), with_IN(x1, x2), light_JJ(x2), weapon_NN(x2)
300148032 / light-armed_JJ(x1) / armed_JJ(x1), with_IN(x1, x2), light_JJ(x2), weapon_NN(x2)
300786396 / new_JJ(x1) / in_IN(x1, x2), use_NN(x2), after_IN(x2, x3), medieval_JJ(x3), time_NN(x3)
301589642 / new_JJ(x1) / in_IN(x1, x2), use_NN(x2), after_IN(x2, x3), medieval_JJ(x3), time_NN(x3)
301707826 / nonindulgent_JJ(x1) / not_RB(x1), indulgent_JJ(x1)
301707826 / nonindulgent_JJ(x1) / not_RB(x1), indulgent_JJ(x1)
301252370 / nonindulgent_JJ(x1) / not_RB(x1), indulgent_JJ(x1)
301252370 / nonindulgent_JJ(x1) / not_RB(x1), indulgent_JJ(x1)
302496645 / noteworthy_JJ(x1) / worthy_JJ(x1), of_IN(x1, x2), notice_NN(x2)
302496645 / noteworthy_JJ(x1) / worthy_JJ(x1), of_IN(x1, x2), notice_NN(x2)
302090576 / noteworthy_JJ(x1) / worthy_JJ(x1), of_IN(x1, x2), notice_NN(x2)
302090576 / noteworthy_JJ(x1) / worthy_JJ(x1), of_IN(x1, x2), notice_NN(x2)
302642589 / premedical_JJ(x1) / prepare_VB-[e1, x1], for_IN-[e1, x2], study_NN(x2), of_IN(x2, x3), medicine_NN(x3)
300131871 / premedical_JJ(x1) / prepare_VB-[e1, x1], for_IN-[e1, x2], study_NN(x2), of_IN(x2, x3), medicine_NN(x3)
300622713 / uncrossed_JJ(x1) / not_RB(x1), crossed_JJ(x1)
300622476 / uncrossed_JJ(x1) / not_RB(x1), crossed_JJ(x1)
302236474 / uncurled_JJ(x1) / not_RB(x1), curled_JJ(x1)
300986826 / uncurled_JJ(x1) / not_RB(x1), curled_JJ(x1)
301272981 / uncut_JJ(x1) / not_RB(x1), cut_JJ(x1)
300631802 / uncut_JJ(x1) / not_RB(x1), cut_JJ(x1)
301559958 / undiscovered_JJ(x1) / not_RB(x1), discovered_JJ(x1)
301330201 / undiscovered_JJ(x1) / not_RB(x1), discovered_JJ(x1)
300671373 / uninflected_JJ(x1) / not_RB(x1), inflected_JJ(x1)

300671373 / uninflected_JJ(x1) / not_RB(x1), inflected_JJ(x1)
300670937 / uninflected_JJ(x1) / not_RB(x1), inflected_JJ(x1)
300670937 / uninflected_JJ(x1) / not_RB(x1), inflected_JJ(x1)
302497933 / unmerited_JJ(x1) / not_RB(x1), merited_JJ(x1)
301324701 / unmerited_JJ(x1) / not_RB(x1), merited_JJ(x1)
301560093 / unobserved_JJ(x1) / not_RB(x1), observed_JJ(x1)
301558829 / unobserved_JJ(x1) / not_RB(x1), observed_JJ(x1)
302042650 / unresponsive_JJ(x1) / not_RB(x1), responsive_JJ(x1)
302042650 / unresponsive_JJ(x1) / not_RB(x1), responsive_JJ(x1)
301934504 / unresponsive_JJ(x1) / not_RB(x1), responsive_JJ(x1)
301934504 / unresponsive_JJ(x1) / not_RB(x1), responsive_JJ(x1)
302082526 / unsheared_JJ(x1) / not_RB(x1), sheared_JJ(x1)
300633432 / unsheared_JJ(x1) / not_RB(x1), sheared_JJ(x1)

CASE 2 (Different SynsetID, same lemma , same LF)

300646823 / acyclic_JJ(x1) / not_RB(x1), cyclic_JJ(x1)
300646823 / acyclic_JJ(x1) / not_RB(x1), cyclic_JJ(x1)
300224255 / beneficent_JJ(x1) / do_VB(e1, x1, x2), or_CC(e3, e1, e2),
produce_VB(e2, x1, x2), good_NN(x2)
300224255 / beneficent_JJ(x1) / do_VB(e1, x1, x2), or_CC(e3, e1, e2),
produce_VB(e2, x1, x2), good_NN(x2)
302891484 / bladed_JJ(x1) / often_RB(e1), use_VB-[e1, x1], in_IN-[e1, x2],
combination_NN(x2)
302891484 / bladed_JJ(x1) / often_RB(e1), use_VB-[e1, x1], in_IN-[e1, x2],
combination_NN(x2)
300819602 / cathartic_JJ(x1) / emotionally_RB(e1), purge_VB(e1, x1)
300819602 / cathartic_JJ(x1) / emotionally_RB(e1), purge_VB(e1, x1)
300624265 / comate_JJ(x1) / bear_VB(e1, x1, x2), coma_NN(x2)
300624265 / comate_JJ(x1) / bear_VB(e1, x1, x2), coma_NN(x2)
302205396 / consonantal_JJ(x1) / of_IN(x1, x5), liquid_NN(x2), and_CC(x5, x2,
x3), nasal_NN(x3)
302205396 / consonantal_JJ(x1) / of_IN(x1, x5), liquid_NN(x2), and_CC(x5, x2,
x3), nasal_NN(x3)
300385058 / dark_JJ(x1) / have_VB(e1, x1, x2), dark_JJ(x2), hue_NN(x2)
300385058 / dark_JJ(x1) / have_VB(e1, x1, x2), dark_JJ(x2), hue_NN(x2)
301139129 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301139129 / earthborn_JJ(x1) / of_IN(x1, x2), earthly_JJ(x2), origin_NN(x2)
301078036 / embezzled_JJ(x1) / take_VB-[e1, x1], for_IN-[e1, x2], own_JJ(x2),
use_NN(x2), in_IN-[e1, x3], violation_NN(x3), of_IN(x3, x4), trust_NN(x4)
301078036 / embezzled_JJ(x1) / take_VB-[e1, x1], for_IN-[e1, x2], own_JJ(x2),
use_NN(x2), in_IN-[e1, x3], violation_NN(x3), of_IN(x3, x4), trust_NN(x4)
300883871 / exhaustible_JJ(x1) / capable_JJ(x1), of_IN(x1, e1), use_up_VB(e1,
x1)
300883871 / exhaustible_JJ(x1) / capable_JJ(x1), of_IN(x1, e1), use_up_VB(e1,
x1)
301899595 / first-string_JJ(x1) / not_RB(x1), substitute_NN(x1)
301899595 / first-string_JJ(x1) / not_RB(x1), substitute_NN(x1)
301899595 / first-string_JJ(x1) / of_IN(x1, x2), member_NN(x2), of_IN(x2, x3),
team_NN(x3)
301899595 / first-string_JJ(x1) / of_IN(x1, x2), member_NN(x2), of_IN(x2, x3),
team_NN(x3)
300593230 / incorrupt_JJ(x1) / free_JJ(x1), of_IN(x1, x5), corruption_NN(x2),
or_CC(x5, x2, x3), immorality_NN(x3)
300593230 / incorrupt_JJ(x1) / free_JJ(x1), of_IN(x1, x5), corruption_NN(x2),
or_CC(x5, x2, x3), immorality_NN(x3)
302466704 / industrial_JJ(x1) / employed_JJ(x1), in_IN(x1, x2),
industry_NN(x2)

APPENDIX 24

SYSTEM O - GATHERING NOUNS FROM SYNSETS

```
import xml.etree.ElementTree as ET
import re

tree=ET.parse("Noun.xml")
root=tree.getroot()
lista_synset=[]

for w in root.iter(tag="synonymSet"):
    synset=w.text
    if "," in synset:
        temp=synset.split(",")
        for x in temp:
            if " " in x:
                s=x.lstrip()
                lista=open("listaSynsetNomi.txt","a")
                lista.write(s+"\n")
                lista.close()
            else:
                lista=open("listaSynsetNomi.txt","a")
                lista.write(x+"\n")
                lista.close()
    else:
        lista=open("listaSynsetNomi.txt","a")
        lista.write(synset+"\n")
        lista.close()
```

- - - - - - - - - -

The output is a txt file with one lemma per line:

entity
thing
anything
something
nothing
nonentity
whole
whole_thing
unit
living_thing
animate_thing
organism
being
benthos
heterotroph ….

APPENDIX 25

SYSTEM P - COMPOUND NOUNS CORRECTION

```
import re
l=open("listaSynsetNomi.txt","r").read()
lt=l.split(",")
ls=[]
listacorrezioni=[]
for w in lt:
    x=w.lower()
    ls.append(x) #list of lemmas in the synsets lowercase

f=open("output_nomi_TerzaParte.txt", "r").readlines()
count=0
for w in f:
    new=open("output1TS.txt","a")
    a=re.search(r"[synset].*\)\-\[",w).group(0)
    new.write(a)
    new.close()
    lf=re.search(r"\-\[.*$",w).group(0)
    if re.search(r"[a-z]*\_NN\([x]*[0-9]*\)\, [a-z]*\_NN\
([x]*[0-9]*\),              [a-z]*\_NN\([x]*[0-9]*\)",lf):
        x=re.search(r"[a-z]*\_NN\([x]*[0-9]*\)\, [a-z]*\_NN\
                    ([x]*[0-9]*\), [a-z]*\_NN\
([x]*[0-9]*\)",lf).group()
        tovar=x.split(",")
        var=tovar[0]
        variabile=var[-6:] #it keeps the first variable
        y=x.split(",")
        a=y[0]
        b=y[1][1:]
        c=y[2][1:]
        ao=a[:-7]
        bo=b[:-7]
        co=c[:-7]
        d=ao+"_"+bo+"_"+co
        e=ao+"-"+bo+"-"+co
        g=ao+"-"+bo+"_"+co
        h=ao+"_"+bo+"-"+co
        if d in ls and "  nn" not in lf:
            m=lf.replace(x,d+"_"+variabile)
            new=open("output1TS.txt","a")
            new.write("*"+m[2:]+"\n")
            new.close()
            count=count+1
            listacorrezioni.append(d)
        elif e in ls and " nn" not in lf:
            m=lf.replace(x,e+"_"+variabile)
            new=open("output1TS.txt","a")
            new.write("*"+m[2:]+"\n")
```

```
                new.close()
                count=count+1
                listacorrezioni.append(e)
            elif g in ls and " nn" not in lf:
                m=lf.replace(x,g+"_"+variabile)
                new=open("output1TS.txt","a")
                new.write("*"+m[2:]+"\n")
                new.close()
                count=count+1
                listacorrezioni.append(g)
            elif h in ls and " nn" not in lf:
                m=lf.replace(x,h+"_"+variabile)
                new=open("output1TS.txt","a")
                new.write("*"+m[2:]+"\n")
                new.close()
                count=count+1
                listacorrezioni.append(h)
            else:
                new=open("output1TS.txt","a")
                new.write(lf[2:]+"\n")
                new.close()
        else:
            new=open("output1TS.txt","a")
            new.write(lf[2:]+"\n")
            new.close()

    print "I corrected %d compound nouns" %count
    print "The corrected compound nouns are the following: ",
    listacorrezioni
```

- - - - - - - - - -

The System returns the UXWN file with corrected and marked (*) CNs; it also prints out the CNs and the number of CNs corrected, see the following example regarding part of the UXWN noun definitions:

```
>>> ================================ RESTART ================================
>>>
I corrected 45 compound nouns
The corrected compound nouns are the following:  ['holy_roman_emperor', 'roman_ca
tholic_church', 'roman_catholic_church', 'little_bighorn_river', 'world_war_ii',
'world_war_ii', 'west_nile_encephalitis', 'helper_t_cell', 'new_world_goldfinch',
'new_zealand_wren', 'old_world_vulture', 'old_world_vulture', 'old_english_sheepd
og', 'thomas_hunt_morgan', 'new_world_mouse', 'new_world_anteater', 'new_world_mo
nkey', 'organophosphate_nerve_agent', 'roman_catholic_church', 'new_york_city', '
reverse_transcriptase_inhibitor', 'reverse_transcriptase_inhibitor', 'hormone_rep
lacement_therapy', 'guy_fawkes_day', 'type_ii_diabetes', 'reverse_transcriptase_i
nhibitor', 'motion-picture_film', 'new_york_bay', 'criminal_investigation_command
', 'new_york_city', 'posterior_cardinal_vein', 'helper_t_cell', 'high_anglican_ch
urch', 'roman_catholic_church', 'soviet_socialist_republic', 'soviet_socialist_re
public', 'american-indian_language', 'san_joaquin_valley', 'soviet_socialist_repu
blic', 'law_enforcement_agency', 'ceylon_cinnamon_tree', 'ku_klux_klan', 'congreg
ational_christian_church', 'wesleyan_methodist_church', 'sun_myung_moon']
>>>
```

APPENDIX 26

SYSTEM Q - MARKING LFs WITH MISSING RELATIVE ADVERBS

```
listasynsets=[]

#the system opens the file where synsetIDs have been previously
stored
synsets=open("synsets_missingradverbs_POS.txt").readlines()

for synset in synsets:
    listasynsets.append(synset[:-1])

check=open("output_POS.txt").readlines()
#the system opens the part of UXWN we want to check

for element in check:
    splitted=element.split(",")
    s=splitted[0]
    synsetID=s[7:]
    if synsetID[1:] in listasynsets:#check if the synsetID
corresponds
        tocheck=open("tocheck_verb.txt","a")
        tocheck.write("*"+element)
        tocheck.close()
    else:
        tocheck=open("tocheck_verb.txt","a")
        tocheck.write(element)
        tocheck.close()
```

Bibliography

- Agerri R. and Peñas A., 2010, *On the Automatic Generation of Intermediate Logic Forms for WordNet Glosses*, in Proceedings of the 11th International Conference on Intelligent Text Processing and Computational Linguistics (CICILing 2010), Iasi, Romania;
- Agirre E. and Soroa A., 2009, *Personalizing PageRank for Word Sense Disambiguation*, in Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009), Athens, Greece;
- Ahn D., Fissaha S., Jijkoun V. and De Rijke M., 2004, *The University of Amsterdam at Senseval3: Semantic Roles and Logic Forms*, in Proceedings of the 3rd Workshop on Sense Evaluation, in the 42th Annual Meeting of the Association for Computational Linguistics (ACL SENSEVAL 2004), Barcelona, Spain;
- Alshawi H., Chang P. and Ringgaard M., 2011, *Deterministic Statistical Mapping of Sentences to Underspecified Semantics*, in Proceedings of the 9th International Conference on Computational Semantics (IWCS 2011), Oxford, UK;
- Altaf M., Moldovan D. and Parker P., 2004, *Sensevale3 Logic Forms: a System and Possible Improvements*, in Proceedings of the 3rd Workshop on Sense Evaluation, in the 42th Annual Meeting of the Association for Computational Linguistics (ACL SENSEVAL 2004), Barcelona, Spain;
- Anderson J.M., 2007, *The Grammar of Names*, Oxford University Press;
- Anthony S. and Patrick J., 2004, *Dependency Based Logial Form Transformation*, in Proceedings of the 3rd Workshop on Sense Evaluation, in the 42th Annual Meeting of the Association for Computational Linguistics (ACL SENSEVAL 2004), Barcelona, Spain;

- Aoife C., Mccarthy M., Burke M., Van Genabith J. and Way A., 2007, *Deriving Quasi-Logical Forms From F-Structures For The Penn Treebank*, in Studies in Linguistics and Philosophy, vol. 83, pages 33-53;
- Banerjee S. and Pedersen T., 2002, *An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet*, in Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2002), Mexico City, Mexico;
- Barker K., Agashe B., Chaw S., Fan J., Friedland N., Glass M., Hobbs J., Hovy E., Israel D., Kim D., Mulkar-Mehta R., Patwardhan S., Porter B., Tecuci D., Yeh P., 2007, *Learning by Reading : A Prototype System, Performance Baseline and Lessons Learned*, in Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007), Vancouver, Canada;
- Bayer S., Burger J., Greiff W. and Wellen B., 2004, *The MITRE Logical Form Generation System*, in Proceedings of the 3rd Workshop on Sense Evaluation, in the 42th Annual Meeting of the Association for Computational Linguistics (ACL SENSEVAL 2004), Barcelona, Spain;
- Bentivogli L., Bocco A. and Pianta E., 2004, *Archiwordnet: Integrating WordNet with Domain-Specific Knowledge*, in Proceedings of the 2nd International Global WordNet Conference (GWC 2004), Brno, Czech Republic;
- Bos J., 2008, *Wide-Coverage Semantic Analysis with Boxer*, in Proceedings of Semantics in Text Processing Conference (STEP 2008), Venice, Italy;
- Bos J., 2016, *Expressive Power of Abstract Meaning Representations*, Computational Linguistics, vol. 42 (3), MIT Press Cambridge, MA;
- Bos J. and Delmonte R. eds., 2008, *Semantics in Text Processing* (STEP 2008), Research in Computational Semantics, vol.1, College Publications, London;
- Brill E., 1992, *A Simple Rule-Based Art of Speech Tagger*, in Proceeding of the 3rd Conference on Applied Natural Language Processing (ANLC 1992), Trento, Italy;
- Buitelaar P. and Cimiano P., 2008, *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, Frontiers in Artificial

Intelligence and Applications, vol. 167, IOS Press, Amsterdam, The Netherlands;

- Carreras X. and Marquez L., 2004, *Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling,* in Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL 2004), Boston, MA;
- Chai J.Y. and Biermann A., 1997, *The Use of Lexical Semantics in Information Extraction*, In Proceedings of the Workshop in Automatic Information Extraction and Building of Lexical Semantic Resources (ACL EACL 1997), Madrid, Spain;
- Chalker S., 1992, *Proper Noun*, in McArthur T. eds., The Oxford Companion to the English Language., Oxford University Press;
- Cinque G., 2014, *The Semantic Classification of Adjectives, a View from Syntax*, in Studies in Chinese Linguistics, vol. 35, pages 3 -32;
- Clark P., Fellbaum C., Hobbs J. R., Harrison P., Murray W. R. and Thompson J., 2008, *Augmenting WordNet for Deep Understanding of Text*, in Proceedings of Semantics in Text Processing Conference (STEP 2008), Venice, Italy;
- Clark P. and Harrison P., 2008, *Boeing's NLP System and the Challenges of Semantic Representation*, in Proceedings of Semantics in Text Processing Conference (STEP 2008), Venice, Italy;
- Clark S. and Curran, J., 2007, *Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models*, Computational Linguistics, vol. 33, pages 493–553;
- Coates-Stephens S., 1992, *The Analysis and Acquisition of Proper Names for the Understanding of Free Text*, in Computers and the Humanities, vol. 26(5/6), pages 285-296;
- Copestake A., 2009, *Slacker Semantics: Why Superficiality, Dependency and Avoidance of Commitment can be the Right Way to Go*, Invited Talk in Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), Athens, Greece;
- Davidson D., 1967, *The Logical Form of Action Sentences*, in Rescher N. eds. The Logic of Decision and Action, University of Pittsburgh Press, pages 81-95, Pittsburgh, PA;

- Davies, M., 1991, *Acts and Scenes*, in Cooper N. and Engel P. eds New Enquiries into Meaning and Truth, pages 41-82, Hemel Hempstead: Harvester Wheatsheaf;
- De Loupy C., Crestan E. and Lemaire E., 2004, *Proper Nouns Thesaurus for Document Retrieval and Question Answering*, in Atelier Question-Réponse, Traitement Automatique des Langues Naturelles *(TALN)*;
- Delmonte R., 2008, *Semantic and Pragmatic Computing with GETARUNS*, in Proceedings of Semantics in Text Processing Conference (STEP 2008), Venice, Italy;
- Delmonte R. and Rotondi A., 2012, *Treebanks of Logical Forms: They Are Useful Only if Consistent*, in Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey;
- Delmonte R. and Rotondi A., 2015, *A Logical Form Parser for Correction and Consistency Checking of LF Resources*, in Proceedings of the 12th International Workshop on Natural Language Processing and Cognitive Science (NLPCS 2015), Krakow, Poland;
- Dipasree P., Mandar M. and Kalyankumar D., 2014, *Improving Query Expansion Using WordNet*, in Proceedings of the Journal of the Association for Information Science & Technology, vol. 65(12), pages 2469-2478;
- Ehrmann M., Jacquet G. and Steinberger R., 2017, *JRC-Names: Multilingual entity name variants and titles as Linked Data*, in Semantic Web, vol. 2, pages 283-295;
- Erekhinskaya T. and Moldovan D., 2013, *Lexical Chains on WordNet and Extensions*, in Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2013), St. Pete Beach, FL;
- Fang, H., 2008, *A Re-Examination of Query Expansion Using Lexical Resources*, in Proceedings of the Human Language Technology Conference (ACL/HLT 2008), Stroudsburg, PA;
- Fellbaum C., 1998, WordNet: an Electronic Lexical Database, Cambridge, MA: MIT Press;

- Gal A., Lapalme G., St-Dizier P. and Somers H., 1991, *Prolog for Natural Language Processing*, Wiley eds.;
- Gangemi A, Guarino N, Masolo C. and Oltramari A., 2003, *Sweetening WordNet with DOLCE*, in AI Magazine 24(3), pages 13-24;
- Gangemi A., Guarino N. and Oltramari A., 2001, *Conceptual Analysis of Lexical Taxonomies: the Case of WordNet Top-Level*, in Formal Ontology in Information Systems: Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS 2001), Ogunquit, ME;
- Gildea D. and Jurafsky D., 2002, *Automatic Labeling of Semantic Roles*, in Computational Linguistics, vol. 28(3), pages 245–288;
- Graber, J., Fellbaum, C., Osherson, D. and Schapire, R., 2006, *Adding Dense, Weighted Connections to WordNet*, in Proceedings of the 3rd International WordNet Conference, Jeju Island, Korea;
- Grasser, A. C., 1981, *Prose Comprehension Beyond the Word*, New York: Spring-Verlag;
- Guarino N., 1998, *Some Ontological Principles for Designing Upper Level Lexical Resources*, in Proceedings of the First International Conference on Language Resources and Evaluation (LREC 1998), Grenada, Spain;
- Harabagiu S., Miller A. and Moldovan D., 1999, *WordNet 2 - a Morphologically and Semantically Enhanced Resource*, in Proceedings of the ACL SIGLEX Workshop on Standardizing Lexical Resources, College Park, MD;
- Harabagiu S. and Moldovan D., 1998, *Knowledge Processing on an Extended WordNet*, in: Fellbaum C. eds. WordNet - an Electronic Lexical Database, MIT press, pages 379-406;
- Harabagiu S., Moldovan D., Pasca M., Mihalcea R., Surdeanu M., Bunescu R., Girju R., Rus V and Morarescu P., 2000, *FALCON: Boosting Knowledge for Answer Engines*, in proceedings of 9th Text Retrieval Conference (TREC 9), Gaithersburg, MD;
- Haridy S., Badr N., Karam O. and Gharib T., 2010, *Enriching Ontologies Using Coarse-Grained Word Senses in the Case of: Wordnet*, Egyptian Computer Science Journal, 34(2);

- Hirst G. and St-Onge D., 1995, *Lexical Chains as Representations of Context for the Detection and Correction of Malapropism*;
- Hobbs J. R., 1985, *Ontological Promiscuity*, in Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL 1985), Chicago, IL;
- Hobbs J. R., 1986, *Overview of the TACITUS project*, in Proceedings of the Workshop on Strategic Computing - Natural Language, Marina del Rey, CA;
- Hobbs J. R., Stickel M., Martin P. and Edwards D., 1993, *Interpretation as Abduction*, Artificial Intelligence, vol. 63, pages 69-142;
- Jacquet G., Ehrmann M., Steinberger R. and Väyrynen J., 2016, *Cross-Lingual Linking of Multi-Word Entities and their Corresponding Acronyms*, in Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016), Portorož, Slovenia;
- Jurgens D. and Pilehvar M., 2015, *Reserating the Awesometastic: An Automatic Extension of the WordNet Taxonomy for Novel Terms*, in Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2015), Denver, CO;
- Jurgens D. and Pilehvar M., 2016, *SemEval 2016: Task 14 Semantic Taxonomy Enrichment*, in Proceeindgs of the 10th International Workshop on Semantic Evaluation (ACL SemEval 2016), San Diego, CA;
- Kilgarriff A., 2000, *Review of WordNet : An electronic lexical database,* in Language, vol. 76, pages 706–708;
- Kripke S.A., 1980, *Naming and Necessity*, Harvard University Press;
- Krstev C., Vitas D., Maurel D. and Tran M., 2005, *Multilingual Ontology of Proper Names*, in Proceedings of the 2nd Language & Technology Conference (LTC 2005), Poznań, Poland;
- Larson R. K., 1998, *Events and Modification in Nominals*, in Proceedings of the 8th Semantics and Linguistic Theory Conference (SALT 8), Massachusetts Institute of Technology, MA;
- Litkowski K. C., 2004, *Senseval-3 Task Word-Sense Disambiguation of WordNet Glosses*, in Proceedings of the 3rd Workshop on Sense

Evaluation, in the 42th Annual Meeting of the Association for Computational Linguistics (ACL SENSEVAL 2004), Barcelona, Spain;

- Magnini B. and Cavaglia G., 2000, *Integrating Subject Field Codes into WordNet*, in Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2000), Athens, Greece;
- Mandala R., Takenobu T. and Hozumi T., 1998, *The Use of WordNet in Information Retrieval*, in Proceedings of the Workshop Usage of WordNet in Natural Language Processing Systems (Coling ACL), Montréal, Canada;
- Mann G., 2002, *Fine-Grained Proper Noun Ontologies for Question Answering*, in Proceedings of the SemaNet Workshop, Taipei, Taiwan;
- Mason O., 1997, *Q-TAG a Portable Probabilistic Tagger*, Corpus Research, University of Birmingham;
- Maurel D., 2008, *Prolexbase: a Multilingual Relational Lexical Database of Proper Names*, in Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008), Marrakech, Morocco;
- May J., 2016, *SemEval 2016 Task 8 : Meaning Representation Parsing*, in Proceedings of the 10th International Workshop on Semantic Evaluation, (ACL SemEval 2016), San Diego, CA;
- Mel'cuk I. and Zholkovsky A., 1998, *The Explanatory Combinatorial Dictionary*, in Relational Model of the Lexicon, Cambridge University Press;
- Mihalcea R. and Bunescu R., 2000, *Levels of Confidence in Building a Tagged Corpus*, Technical Report, SMU;
- Mihalcea R. and Moldovan D., 2001, *eXtended WordNet: Progress Report*, in Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001), Pittsburgh, PA;
- Miller G. A., 1995, *WordNet: A Lexical Database for English*, Communications of the ACM vol. 38(11), pages 39-41;
- Miller G. A. and Hristea F., 2006, *WordNet Nouns: Classes and Instances*, in Computational Linguistics, vol. 32(1), pages 1-3;

- Miller T. and Gurevych I., 2014, *WordNet Wikipedia-Wikitionary: Construction of a Three-Way Alignment*, in Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014), Maui, HI;
- Moldova D. and Blanco E., 2012, *Polaris: Lymba's Semantic Parser*, in Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey;
- Moldovan D., Clark C., Harabagiu S. and Maiorano S., 2003, *COGEX: A Logic Prover for Question Answering*, in Proceedings of the North American Chapter of the Association for Computational Linguistics (HLTC NAACL 2003), Edmonton, Canada;
- Moldovan D., Harabagiu S., Girju R., Morarescu P., Lacatusu V. F., Novischi A., Badulescu A., Bolohan O., 2002, *LCC Tools for Question Answering*, in Proceedings of the 11th Text Retrieval Conference (TREC 2002), Gaithersburg, MD;
- Moldovan D. and Novischi A., 2002, *Lexical Chain for Question Answering*, in Proceedings of the 19th International Conference on Computational Linguistics (ACL COLING 2002), Taipei, Taiwan;
- Moldovan D. and Novischi A., 2004, *Word Sense Disambiguation of WordNet Glosses*, in Computer Speech and Language, vol. 18, pages 297-313;
- Moldovan D. and Rus V., 2001, *Explaining Answers with Extended WordNet*, in Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL 2001), Toulouse, France;
- Moldovan D. and Rus V., 2001, *Logic Form Transformation of WordNet and its Applicability to Question Answering*, in Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL 2001), Toulouse, France;
- Mooney R. J., 2007, *Learning for Semantic Parsing*, in Proceedings of the 8th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2007), Mexico City, Mexico;
- Morzycki M., 2015, *The Lexical Semantics of Adjectives: More than Just Scales*, in Modification, pages 13–87, Cambridge University Press;

- Navigli R., 2009, *Word Sense Disambiguation: A Survey*, in Journal ACM Computing Surveys (CSUR), vol. 41(2), February;
- Navigli R. and Ponzetto S.P., 2012, *BabelNet: the Automatic Construction, Evaluation and Application of a Wide Coverage Multilingual Semantic Network*, in Artificial Intelligence (193), pages 217-250;
- Navigli R., Velardi P., Cucchiarelli A., Neri F. and Cucchiarelli R., 2004, *Extending and Enriching WordNet with Ontolearn*, in Proceedings of the 2nd Global WordNet Conference (GWC 2004), Brno, Czech Republic;
- Neel A. and Garzon M., 2010, *Semantic Methods for Textual Entailment: How Much World Knowledge is Enough?* In Proceedings of the 23rd International Florida Artificial Intelligence Research Society Conference(FLAIRS 2010), Daytona Beach, FL;
- Niemann E. and Gurevych I., 2011, *The People's Web Meets linguistic Knowledge: Automatic Sense Alignment of Wikipedia and WordNet*, in Proceedings of the 9th International Conference on Computational Semantics (IWCS 2011), Oxford, UK;
- Nimb S., Pedersen B., Braasch A., Sørensen N. and Troelsgard T., 2013, *Enriching a WordNet from a Thesaurus*, Lexical Semantic Resources for NLP, page 36;
- Oltramari A., Gangemi A., Guarino N. and Masolo C., 2002, *Restructuring WordNet's Top-Level: the OntoClean Approach*, in Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002), (OntoLex workshop), Las Palmas, Spain;
- Payne J. and Huddleston R., 2002, *Nouns and Noun Phrases*, in Pullum G. and Huddleston R. eds., The Cambridge Grammar of the English Language, Cambridge University Press;
- Pilehvar M. T. and Navigli R., 2014, *A Robust Approach to Align Heterogenous Lexical Resources*, in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Baltimore MD;
- Ponzetto S. P. and Navigli R., 2009, *Large-Scale Taxonomy Mapping for Restructuring and Integrating Wikipedia*, in Proceedings of the 21st

International Joint Conference on Artificial Intelligence (IJCAI 09), Pasadena, CA;

- Poprat M., Beisswanger E. and Hahn U., 2008, *Building a BioWordNet by Using WordNets Data Formats and WordNets Software Infrastructurea Failure Story*, in Software Engineering, Testing, and Quality As- surance for Natural Language, page 31;
- Pust M., Hermjakob U., Knight K., Marcu D. and May J., 2015, *Using Syntax-Based Machine Translation to Parse English into Abstract Meaning Representation*, in Proceedings of the Empirical Methods in Natural Language Processing Conference (EMNLP 2015), Lisbon, Portugal;
- Ratnaparkhi A., 1996, *A Maximum-Entropy Part of Speech Tagger*, in Proceedings of the 1st Empirical Methods in Natural Language Processing Conference (EMNLP 1996), Philadelphia, PA;
- Reisinger J. and Pasca M., 2009, *Latent Variable Models of Concept-Attribute Attachment*, In Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Singapore;
- Resnik P., ,1995, *Disambiguating Noun Grouping with Respect to WordNet Senses*, in Proceedings of the 3rd Workshop on Very Large Corpora, Cambridge, MA;
- Rosso P., Molina A., Pla F., Jiménez D. and Vidal V., 2004, T*ext Categorization and Information Retrieval Using WordNet Senses*, in Proceedings of the 5th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2004), Seul, Korea;
- Rus V., 2002, *High Precision Logic Form Transformation*, International Journal on Artificial Intelligence Tools, vol. 11(3);
- Rus V., 2002, *Logic Form for WordNet Glosses*, PhD Thesis, Computer Science Department, School of Engineering, Southern Methodist University, Dallas, TX;
- Rus V., 2004, *A First Evaluation of Logic Form Identification Systems*, in Proceedings of the 3rd International Workshop on the Evaluation of

Systems for the Semantic Analysis of Text (ACL SENSEVAL 2004), Barcelona, Spain;

- Rus V., 2009, *What Next in Knowledge Representation?*, in Proceedings of The Knowledge Engineering: Principles and Techniques International Conference, Cluj, Romania;
- Ruiz-Casado M., Alfonseca E. and Castells P., 2005, *Automatic Assignment of Wikipedia Encyclopaedic Entries to WN Senses*, in Proceedings of the 3rd International Conference on Advances in Web Intelligence (AWIC 2005), Lodz, Poland;
- Savary, A., Manicki, L., and Baron, M., 2013, *Populating a Multilingual Ontology of Proper Names from Open Sources,* in Journal of Language Modelling, Vol 2(2), pages189-225;
- Schafer U., 2007, *Integrating deep and shallow natural language processing components – representations and hybrid architectures,* Ph.D. thesis, Faculty of Mathematics and Computer Science, Saarland University, Saarbrucken, Germany;
- Sheremetyeva S., Cowie J., Nirenburg S. and Zajac R., 1998, *Multilingual Onomasticon as a Multipurpose NLP Resource*, in Proceedings of the First International Conference on Language Resources and Evaluation (LREC 1998), Grenada, Spain;
- Smith B. and Fellbaum C., 2004, *Medical WordNet: a New Methodology for the Construction and Validation of Information Resources for Consumer Health*, in Proceedings of the 20th International Conference on Computational Linguistics, Association for Computational Linguistics(ACL COLING 2004), Geneva, Switzerland;
- Snow R., Jurafsky D. and Y Ng A., 2006, *Semantic Taxonomy Induction from Heterogenous Evidence*, in Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (ACL COLING 2006), Sydney, Australia;
- Steinberger R., Pouliquen B. and Van Der Goot E., 2009, *An Introduction to the Europe Media Monitor Family of Applications*, in Proceedings of the 32nd International ACM SIGIR conference on research and development in Information Retrieval (SIGIR 2009), Boston, MA;

- Sundheim B.M., Mardis S., and Burger J., 2006, *Gazetteer Linkage to WordNet*, in Proceedings of the 3rd International WordNet Conference, Jeju Island, Korea;
- Tanev H. and Rotondi A., 2016, *Deftor at SemEval-2016 Task 14: Taxonomy Enrichment Using Definition Vectors*, in Proceedings of the 10th International Workshop on Semantic Evaluation, (ACL SemEval 2016), San Diego, CA;
- Toral A., Muñoz R. and Monachini M., 2008, *Named Entity WordNet*, in Proceedings of the 6th Conference on Language Resources and Evaluation (LREC 2008), Marrakech, Morocco;
- Vanderwende L., Menezes A. and Quirk C., 2015, *An AMR Parser for English, French, German, Spanish and Japanese and a New AMR-Annotated Corpus*, in Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2015), Denver, CO;
- Varelas G., Voutsakis E., Raftopoulou P., Petrakis E.G.M. and Milio E., 2005, *Semantic Similarity Methods in WordNet and their Application to Information Retrieval on the Web*, in Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management, Bremen, Germany;
- Verdezoto N. and Vieu L., 2011, *Towards Semi-Automatic Methods for Improving WordNet*, in Proceedings of the 9th International Conference on Computational Semantics (IWCS 2011), Oxford, United Kingdom;
- Vujicic Stankovic S., Krstev C. and Vitas D., 2014, *Enriching Serbian WordNet and Electronic Dictionaries with Terms from the Culinary Domain*, in Proceedings of the 7th Global WordNet Conference (GWC 2014), Tartu, Estonia;
- Wang C., Pradhan S., Xue N., Pan X. and Ji H., 2016, *CAMR at SemEval-2016 Task 8: an Extended Transition-Based AMR Parser*, in Proceedings of the 10th International Workshop on Semantic Evaluation, (ACL SemEval 2016), San Diego, CA.
- Wilks Y., Guthrie, L., Guthrie, J., Cowie, J., Farwell, D., Slator, B., Bruce, R., 1993, *A Research Program on Machine-Tractable Dictionaries and*

Their Application to Text Analysis, in Literary and Linguistic Computing, 8.4. special issue, (eds. Ostler & Zampolli).