A framework for structured knowledge extraction and representation from natural language via deep sentence analysis

Stefania Costantini[1], **Niva Florio**[1], and **Alessio Paolucci**[1]

Dip. di Informatica, Università di L'Aquila, Coppito 67100, L'Aquila, Italy
`stefania.costantini@univaq.it`
`niva.florio@univaq.it`
`alessio.paolucci@univaq.it`

Abstract. We present a framework that we are currently developing, that allows one to extract knowledge from natural language sentences using a deep analysis technique based on linguistic dependencies. The extracted knowledge is represented in OOLOT, an intermediate format that we have introduced, inspired by the Language of Thought (LOT) and based on Answer Set Programming (ASP). OOLOT uses an ontology-oriented lexicon and syntax. Therefore, it is possible to export the extracted knowledge into OWL and native ASP.

1 INTRODUCTION

Many intelligent systems have to deal with knowledge expressed in natural language, either extracted from books, web pages and documents in general, or expressed by human users. Knowledge acquisition from these sources is a challenging matter, and many attempts are presently under way towards automatically translating natural language sentences into an appropriate knowledge representation formalism [1]. Although this task is a classic Artificial Intelligence challenge (mainly related to Natural Language Processing and Knowledge Representation [2]), with the Semantic Web growth new interesting scenarios are opening. The Semantic Web aims at complementing the current text-based web with machine interpretable semantics; however, the manual population of ontologies is very tedious and time-consuming, and practically unrealistic at the web scale [3, 4]. Given the enormous amount of textual data that is available on the web, to overcome the knowledge acquisition bottleneck, the ontology population task must rely on the use of natural language processing techniques to extract relevant information from the Web and transforming it into a machine-processable representation.

In this paper we present a framework that we are currently developing. It allows one to extract knowledge from natural language sentences using a deep analysis technique based on linguistic dependencies and phrase syntactic structure. We also introduce OOLOT (Ontology-Oriented Language of Thought). It is an intermediate language based on ASP, specifically designed for the representation of the distinctive features of the knowledge extracted from natural language. Since OOLOT is based on an ontology-oriented lexicon, our framework can be easily integrated in the context of the Semantic Web.

It is important to emphasize that the choice of ASP is a key point and it is of fundamental relevance. In fact according to [5], this formalism is the most appropriate one to deal with normative statements, default statements, exceptions and many other characteristic aspects of knowledge encoded through Natural Language.

Though this is an ongoing work, we believe to be able to argue in favour of the usefulness and the potential of the proposed approach.

In particular, in Section 2 we introduce the framework architecture. In Section 3 we analyze the state of the art for parsing and extraction of dependencies taking into account our translation needs, taking into particular account three kinds of parsers. Section 4 describes the context disambiguation and lexical item resolution methods that we have devised. Section 5 introduces the intermediate format OOLOT, while Section 6.3 describes the translation methodology with the help of an example. Finally, Section 7 shows an exporting from OOLOT into OWL example, and in Section 8 we conclude with a brief resume of achieved goals and future works.

2 FRAMEWORK ARCHITECTURE

The proposed framework aims at allowing automatic knowledge extraction from plain text, like a web page, producing a structured representation in OWL or ASP as output. Thus, the framework can be seen as a standalone system, or can be part of wider workflow, e.g. a component of complex semantic web applications.

Starting from plain text written in natural language, as first step we process the sentence through a statistical parser (see Section 3). If we use a parser with embedded dependency extractor, we can perform a single step and have as output both the parse tree (constituents), and in the meantime the dependency graph. Otherwise, if we use two different components, the workflow is that of Fig.1. In this case, we use a simple algorithm for context disambiguation (see Section 4). Then, each token is resolved w.r.t. popular ontologies including DBPedia and OpenCYC and the context is used in case of multiple choices.

At this point we have enough information to translate the knowledge extracted from a natural language sentence into our intermediate OOLOT format. OOLOT stands for "Ontological Oriented Language Of Though", a language mainly inspired by [6], that we have introduced as an intermediate representation language for the extracted knowledge in a way that is totally independent from the original lexical items, and therefore, from the original language. OOLOT is itself a language, but its lexicon is ontology-based; it uses Answer Set Programming as basic host environment that allows us to compose a native, high expressive knowledge representation and reasoning environment. For the

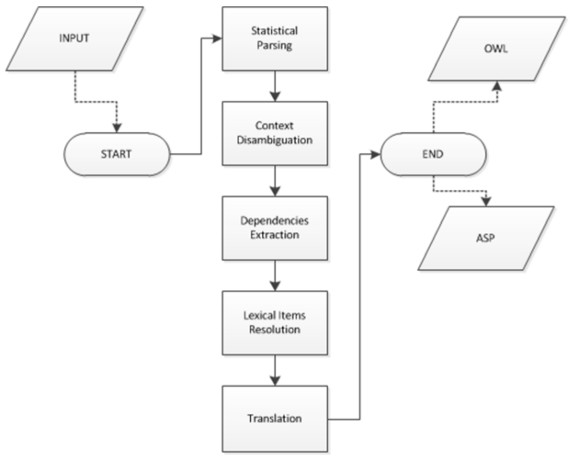


Fig. 1. The framework architecture

translation process described in Section 6, we employ a λ-calculus engine that drives the translation into OOLOT, using information about the deep structure of the sentence extracted in the previous steps.

From the ontology-based Language of Thought it is possible to directly translate the encoded knowledge into OWL. In addition, it is also possible to export the knowledge base in pure ASP.

3 PARSING

3.1 Background

In informatics and linguistics, parsing is the process that can determine the morpho-syntactic structure of a sentence; parsing associates a sentence expressed in a natural language with a structure (e.g. a parse tree structure), that analyses the sentence by a certain point of view; thus there are morphological parsing, syntactic parsing, semantic parsing, etc.

With regard to the syntactic parsing, analysis consists of a definition of the phrases building up the sentence in their hierarchical order, likewise the constituent analysis proposed by Chomsky [7]. In the 1950s Noam Chomsky said natural language sentences can be generated by a formal grammar [8, 7]; this is the so-called generative approach, motivated by the fact that people are able to generate sentences that are syntactically correct and totally new (i.e., that have never been heard before). Syntactic parser decomposes a text into a sequence of tokens (for example, words), and attributes them their grammatical functions and thematic or logical roles, with respect to a given formal grammar. The task of syntactic parser is to say if the sentence can be generated by the grammar and, if so, it gives the appropriate sentence syntactic representation (called parse tree), showing also the relations between the various elements of the sentence [8, 7].

Most of today's syntactic parsers are mainly statistical [9–12]. They are based on a corpus of training data that have been previously annotated (parsed) by hand. This approach allows the system to gather information on the frequency with which the various constructs are needed in specific contexts. A statistical parser can use a search procedure on the space of all candidates, and it would provide the probability of each candidate and makes it possible to derive the most probable parse of a sentence.

In the '90, Collins proposes a conditional and generative model which describes a straightforward decomposition of a lexicalized parse tree [11, 12] based on a Probabilistic Context Free Grammar (PCFG). Charniak and Johnson's parser [10, 13] is based on a parsing algorithm for PCFG, but it is a lexicalized N-Best PCFG parser: it is a generative and discriminative reranking parser which uses the MaxEnt reranker to select the best among the possible parses. The Berkeley parser uses an automatically induced PCFG, parsing sentences with a hierarchical state-splitting. Only statistics is not enough to determine when to split each symbol in sub-symbols [14]; thus [14, 15] present an automatic approach for obtaining annotation trees through a split-merge method. At a first stage, this parser considers a simple PCFG derived from a treebank, but then it iteratively refines this grammar, in order to sub-categorize basic symbols (like NP and VP) into sub-symbols. So non-terminal basic symbols are split and merged in order to maximize the training treebank and to add a larger number of annotations to the previous grammar. This parser can learn automatically the type of linguistic distinction showed in the manually annotated treebank and then it can create annotation trees thanks to a more complex and complete grammar.

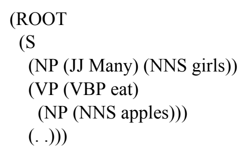
Statistical parsing is useful to solve problems like ambiguity and efficiency, but with this kind of parsing we lose part of the semantic information; this aspect is recovered thanks to dependency representation [16].

Dependency grammars (DGs) were proposed by the French linguist Tesnière [17] and have recently received renewed attention (cfr. [18] and the references therein). In Dependency Grammars, words in a sentence are connected by means of binary, asymmetrical governor-dependent relationships. In fact, Tesnière assumes that each syntactic connection corresponds to a semantic relation. In a sentence, the verb is seen as the highest level word, governing a set of complements, which govern their own complements themselves. Opposed to the notion of the sentence division into a subject and predicate, the grammatical subject in Tesnière's work is also considered subordinate to the verb. The valence of a verb (its property of requiring certain elements in a sentence) determines the structure of the sentence it occurs in. Tesnière distinguishes between actants, which are required by the valence of the verb, and circonstants which are optional.

3.2 Parser analysis

It is difficult to evaluate parsers; we can compare them in many ways, such as the speed with which they examine a sentence or their accuracy in the analysis (e.g. [23]). The task based evaluation seems to be the best one [16, ?]: we must choose whether to use a parser rather than another simply basing on our needs. At this stage of our ongoing research, we use the Stanford parser because it is more suited to our requirements, both for the analysis of the constituents and for that of the dependencies.

Stanford parser performs a dependency and constituent analysis [19, 20]. This parser provides us with different types of parsing. In fact, it can be used as an unlexicalized PCFG parser [19] to analyse sentences, or it can be used as a lexicalized probabilistic parser [20]. Thanks to an A* algorithm, the second version combines the PCFG analysis with the lexical dependency analysis. At the moment the Stanford parser provides us a typed dependency and a phrase structure tree. The Stanford typed dependencies (cfr. [16]) describe the grammatical relations in a sentence. The relations are binary and are arranged hierarchically; as Tesnière suggested, Stanford dependency relations have a head and its dependent but, unlike Tesnière, the head of a dependency can be any content words, not only verbs. Thanks to rules [21] applied on phrase structure trees (also created by the Stanford parser), typed dependencies are generated.
In particular, for constituent analysis, we choose to analyse the sentence *"Many girls eat apples."*. Seeing Fig.2, we can notice that the parser attributes to each token its syntactic roles, and it provides us also the grammatical function of each word. In order to better understand the hierarchy of the syntactic structure is useful to represent it as a tree (Fig.3).



```
(ROOT
  (S
    (NP (JJ Many) (NNS girls))
    (VP (VBP eat)
      (NP (NNS apples)))
    (. .)))
```

Fig. 2. Phrase structure produced by the Stanford parser for the sentence "Many girls eat apples".

With regard to dependency analysis, the Stanford parser gives us two versions of this analysis: the typed dependency structure (Fig.4) and the collapsed typed dependency structure (Fig.5). In the first, each node of the sentence is a node connected with a binary relation to another node; in the second, prepositions are turned into relations (unfortunately, in this example, you may not notice the difference). Thus, Fig.4 and Fig.5 show us that *girls* and *many* are connected with an *amod* relation, that means an adjective phrase modifies the meaning of the noun phrase; *eat* is connected to *girls* with a *nsubj* relation, where the noun phrase is the syntactical subject of the verb; *eat* and *apple* are connected with a *dobj* relation because the direct object of the verb phrase is the object of the verb [16].

As reference for the following steps, we use the Stanford syntactic phrase structure (Fig.2) and the Stanford collapsed typed dependencies structure (Fig.5).

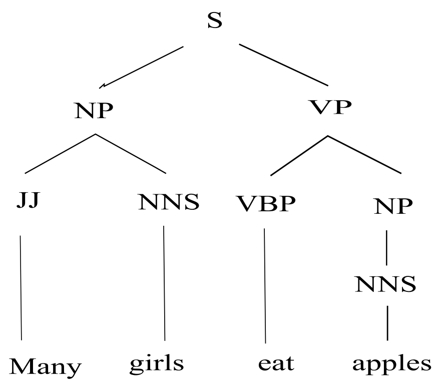


Fig. 3. Parse tree of the sentence "Many girls eat apples." for the analysis done by the Stanford parser.

amod(girls-2, Many-1)
nsubj(eat-3, girls-2)
dobj(eat-3, apples-4)

Fig. 4. Typed dependency structure produced by the Stanford parser for the sentence "Many girls eat apples.".

amod(girls-2, Many-1)
nsubj(eat-3, girls-2)
dobj(eat-3, apples-4)

Fig. 5. Collapsed typed dependency structure produced by the Stanford parser for the sentence "Many girls eat apples.".

4 CONTEXT DISAMBIGUATION AND LEXICAL ITEM RESOLUTION

The context disambiguation task is a very important step in our work flow, as we need to assign each lexical unit to the correct meaning, and this is particularly hard due to the polysemy. For this task, we use a simple algorithm: we have a finite set of contexts (political, technology, sport, ...), and as first step we built a corpus of web pages for each context, and then we used each set as a training set to build a simple lexical model. Basically we build a matrix where for each row we have a lexical item, and for each column we have a context. The relation (lexical item, context) is the normalized frequency of each lexical item into the given context. The model is then used to assign the correct context to a given sentence. We use a $n \times m$ matrix, where n is the number of lexical tokens (or items), and m is the number of contexts. In other words, we give a score for each lexical token in relation to each context. To obtain the final score we perform a simple sum of the local values to obtain the global score, and thus assign the final context to the sentence. Our method for context disambiguation can certainly be improved, in particular [25], seems to be a good development direction.

The context is crucial to choose the correct reference when a lexical item has multiple meanings, and thus, in an ontology can be resolved in multiple references. The context becomes the key factor for the resolution of each lexical item to the relative reference.

We perform a lookup in the ontology for each token, or a set of them (using a sliding window of length k). For example, using DBPedia, for each token (or a set of tokens of length k), we perform a SPARQL query, assuming the existence of a reference to the lexical item: if this is true, we've found the reference, otherwise we go forward. If we found multiple references, we use the context to choose the most appropriate one.

Given the sentence *Many girls eat apples* and it's syntactic phrase structure (Fig.2) and dependencies structure (Fig.5), as first step we tokenize the sentence, obtaining:

Many, *girls*, *eat*, *apples*.

Before the lookup, we use the part of speech tagging from the parse tree to group the consecutive tokens that belong to the same class. In this case, such peculiar aspect of natural language is not present and thus the result is simply the following:

(*Many*), (*girls*), (*eat*), (*apples*).

Excluding the lexicon for which we have a direct form, for each other lexicon the reference ontology is resolved through a full text lookup; thus we obtain the lexical item resolution in Table 1.

Table 1. Lexical item resolution example

Lexicon	*Ontology*	*URI*
girls	DBPedia	http://dbpedia.org/resource/Girl
eat	DBPedia	http://dbpedia.org/class/Eating
apples	DBPedia	http://dbpedia.org/resource/Apple

5 FROM THE LANGUAGE OF THOUGHT TO OOLOT

The Language of Thought (LOT) is an intermediate format mainly inspired by [6]. It has been introduced to represent the extracted knowledge in a way that is totally independent from original lexical items and, therefore, from original language.

Our LOT is itself a language, but its lexicon is ontology oriented, so we adopted the acronym OOLOT (Ontology-Oriented Language Of Thought). This is a very important aspect: OOLOT is used to represent the knowledge extracted from natural language sentences, so basically the bricks of OOLOT (lexicons) are ontological identifier related to concepts (in the ontology), and they are not a translation at lexical level.

In [26] a translation methodology from natural language sentences into ASP that takes into accounts all words of the sentence is presented. In this method, the final representation is itself dependent from the original lexical structure, and this is sub-optimal if we want to export our knowledge base into a more general formalism like, e.g., OWL.

In the next section we present the translation process.

6 TRANSLATING INTO OOLOT

6.1 Background

[26] describes a technique for extracting knowledge from natural language and automatically translate it into ASP. To achieve this result we built an extension of λ-calculus and we have introduced meta expressions to fully automate the translation process, originally inspired by [5].

This is a key point in the process of representing knowledge extracted from natural language, and thus for using it into other contexts, e.g. the Semantic Web. The selection of a suitable formalism plays an important role: in fact, though under many aspects first-order logic would represent a natural choice, it is actually not appropriate for expressing various kinds of knowledge, i.e. for dealing with default statements, normative statements with exceptions, etc. Recent work has investigated the usability of non-monotonic logics, like ASP [5] with encouraging results in terms of dealing with the kind of knowledge represented through natural language.

OOLOT allows us to have native reasoning capabilities (using ASP) to support the syntactic and semantic analysis tasks. Embedded reasoning is of fundamental importance for the correct analysis of complex sentences, as shown in [27]. The advantages of adopting an ASP-based host language is not limited to the previous aspects: in fact, the integration of ASP and the Semantic Web is not limited to the Natural Language Processing side. Answer Set Programming fits very well with Semantic Web as demonstrated by the recent research efforts of integrating rule-based inference methods with current knowledge representation formalisms in the Semantic Web [28, 29].

Ontology languages such as OWL and RDF Schema are widely accepted and successfully used for semantically enriching knowledge on the Web. However, these languages have a restricted expressivity if we have to infer new knowledge from existing one. Semantic Web needs a powerful rule language complementing its ontology formalisms in order to facilitate sophisticated reasoning tasks. To overcome this gap, different approaches have been presented on how to combine Description Logics with rules, like in [29].

6.2 Lambda Calculus

λ-calculus is a formal system designed to investigate function definition, function application and recursion. Any computable function can be expressed and evaluated via this formalism [30]. In [26] we extended the λ-calculus introducing the λ-ASP-$Expression_T$ that allows a native support for ASP, and, at the same time, permits to formally instantiated to λ-ASP-Expression [5, 26]. For the purpose of our running example, the set of λ-ASP-$Expression_T$ is available in Table 2.

In the following subsection, we illustrate the translation process based on λ-calculus. It is important to note that the choice of the lambda calculus was made because it fully matches the specifications of the formal tool we need to drive the execution of the steps in the right way.

6.3 Lambda-based translation

According to the workflow in Fig.1, the translation from plain text to the OOLOT intermediate format makes use of the information extracted in several steps. The information on the deep structure of the sentence is now used to drive the translation using the λ-calculus according to the λ- expression definitions in Table2.

Table 2. The λ-ASP-expression template

Lexicon	SemClass	$\lambda - ASP - expression\ T$
-	noun	$\lambda x.\langle noun\rangle(x)$
-	verb	$\lambda y.\langle verb\rangle(y)$
-	transVerb	$\lambda y.\lambda w.\langle verb\rangle(y, w)$
many	det	$\lambda u \lambda v.($ $v@X \leftarrow u@X,$ $not\ \neg v@X,$ $possible(v@X, u@X),$ $usual(v@X, u@X)$ $)$

For each lexicon, we use the phrase structure in Fig.2 to determine the semantic class to which it belongs. In this way, we are able to fetch the correct *lambda*-ASP-expression template from the Table 2.

For the running example, as result we have the λ-ASP-expressions of Table 3.

Table 3. The λ-ASP-expressions

Lexicon	$\lambda - ASP - expression$
apples	$\lambda x.dbpedia : Apple(x)$
eat	$\lambda y \lambda w.dbpedia : Eating(y, w)$
girls	$\lambda z.dbpedia : Girl(x)$
many	$\lambda u \lambda v.($ $v@X \leftarrow u@X,$ $not\ \neg v@X,$ $possible(v@X, u@X),$ $usual(v@X, u@X)$ $)$

Now, differently from [26], we use the dependencies, that is, we use the deep structure information, to drive the translation.

According to the dependency in Fig.5, the first relation that we use is $amod(girls-2,\ many-1)$. Thus, for the λ-calculus definition, we apply the λ-ASP-expression for *girls* to the λ-ASP-expression for *many*:

$$\begin{array}{l}
\lambda u \lambda v.(\\
\quad v@X \leftarrow u@X, \\
\quad not\ \neg v@X, \\
\quad possible(v@X, u@X), \\
\quad usual(v@X, u@X) \\
)@@(\lambda x.(dbpedia : Girl(x))
\end{array}$$

obtaining:

$$\begin{array}{l}
\lambda v.(\\
\quad v@X \leftarrow dbpedia : Girl(X), \\
\quad not\ \neg v@X, \\
\quad possible(v@X, dbpedia : Girl(X)), \\
\quad usual(v@X, dbpedia : Girl(X)) \\
)
\end{array}$$

The second relation, $nsubj(eat-3,\ girls-2)$, drives the application of the λ-expression for *eat* to the expression for *girls* that we obtained in the previous step:

$$\begin{aligned}
&\lambda v.(\\
&\quad v@X \leftarrow dbpedia : Girl(X),\\
&\quad not\ \neg v@X,\\
&\quad possible(v@X, dbpedia : Girl(X)),\\
&\quad usual(v@X, dbpedia : Girl(X))\\
&)@@(\lambda z \lambda w.(dbpedia : Eating(z, w))
\end{aligned}$$

that reduces to:

$$\begin{aligned}
&dbpedia : Eating(X, W) \leftarrow dbpedia : Girl(X),\\
&\quad not\ \neg dbpedia : Eating(X, W),\\
&\quad possible(dbpedia : Eating(X, W), dbpedia : Girl(X)),\\
&\quad usual(dbpedia : Eating(X, W), dbpedia : Girl(X))
\end{aligned}$$

Then, we apply *apple* to the expression we have seen, obtaining the final result:

$$\begin{aligned}
&dbpedia : Eating(X, dbpedia : Apple) \leftarrow dbpedia : Girl(X),\\
&\quad not\ \neg dbpedia : Eating(X, dbpedia : Apple),\\
&\quad possible(dbpedia : Eating(X, dbpedia : Apple), dbpedia : Girl(X)),\\
&\quad usual(dbpedia : Eating(X, dbpedia : Apple), dbpedia : Girl(X))
\end{aligned}$$

7 EXPORTING INTO OWL

Our framework has been designed to export the knowledge base from OOLOT into a target formalism. For now, we are working on a pure ASP and OWL exporter.

Exporting into OWL is a very important feature, because it allows endless possibilities due to its native Semantic Web integration. In this way, the framework as a whole becomes a power tool that starting from plain text produces the RDF/OWL representation of the sentences; through ASP it takes care of special reasoning and representation features of natural language.

To complete the example, the resulting RDF representation is:

```
< http://dbpedia.org/resource/Girl,
  http://dbpedia.org/ontology/Eating,
  http://dbpedia.org/resource/Apple >
```

Fig. 6. RDF

The export into OWL has, at the present stage, some drawbacks, including loosing of some aspects of natural language that instead are perfectly managed in OOLOT. The export procedure is ongoing work, so there is room for improvement. Due to the nature of the problem, that is very complex, these aspects will be the subject of a future work.

8 CONCLUSION

In this paper, we have proposed a comprehensive framework for extracting knowledge from natural language and representing the extracted knowledge in suitable formalisms so as to be able to reason about it and to enrich existing knowledge bases. The proposed framework is being developed and an implementation is under way and will be fully available in short time. The proposed approach incorporates the best aspects and results from previous related works and, although in the early stages, it exhibits a good potential and its prospects for future development are in our opinion really interesting.

Future improvements concern many aspects of the framework. First of all, we have to choose the best parser or establish how to combine the best aspects of all them. On the OOLOT side, there is the need to better formalize the language itself, and better investigate the reasoning capabilities that it allows, and how to take the best advantage from them.

The ontology-oriented integration is at a very early stage, and there is room for substantial improvements, including a better usage of the current reference ontologies, and the evaluation study about using an upper level ontology, in order to have a more homogeneous translation.

References

1. Bos, J., Markert, K.: Recognising textual entailment with logical inference. In: HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2005) 628–635
2. Pereira, F., Shieber, S.: Prolog and natural-language analysis. Microtome Publishing (2002)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. The Semantic Web (2007) 722–735
4. Kasneci, G., Ramanath, M., Suchanek, F., Weikum, G.: The YAGO-NAGA approach to knowledge discovery. SIGMOD Record **37**(4) (2008) 41–47
5. Baral, C., Dzifcak, J., Son, T.C.: Using answer set programming and lambda calculus to characterize natural language sentences with normatives and exceptions. In: Proceedings of the 23rd national conference on Artificial intelligence - Volume 2, AAAI Press (2008) 818–823
6. Kowalski, R.: Computational Logic and Human Thinking: How to be Artificially Intelligent - In Press
7. Chomsky, N.: Syntactic Structures. The MIT Press (1957)
8. Chomsky, N.: Three models for the description of language. IEEE Transactions on Information Theory **2**(3) (1956) 113–124

9. Charniak, E.: Tree-bank grammars. In: Proceedings of the National Conference on Artificial Intelligence. (1996) 1031–1036
10. Charniak, E., Johnson, M.: Coarse-to-fine n-best parsing and maxent discriminative reranking. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2005) 173–180
11. Collins, M.: A new statistical parser based on bigram lexical dependencies. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics, Association for Computational Linguistics (1996) 184–191
12. Collins, M.: Three generative, lexicalised models for statistical parsing. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics (1997) 16–23
13. McClosky, D., Charniak, E., Johnson, M.: Effective self-training for parsing. In: Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Association for Computational Linguistics (2006) 152–159
14. Petrov, S., Barrett, L., Thibaux, R., Klein, D.: Learning accurate, compact, and interpretable tree annotation. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Association for Computational Linguistics (2006) 433–440
15. Petrov, S., Klein, D.: Improved inference for unlexicalized parsing. In: Proceedings of NAACL HLT 2007. (2007) 404–411
16. de Marneffe, M., Manning, C.: The stanford typed dependencies representation. In: Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, Association for Computational Linguistics (2008) 1–8
17. Tesnière, L.: Elèments de syntaxe structurale. Klincksieck, Paris (1959) ISBN 2252018615.
18. Neuhaus, P., Bröker, N.: The complexity of recognition of linguistically adequate dependency grammars. In: Proc. of ACL-97/EACL-97. (1997)
19. Klein, D., Manning, C.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1, Association for Computational Linguistics (2003) 423–430
20. Klein, D., Manning, C.: Fast exact inference with a factored model for natural language parsing. Advances in neural information processing systems (2003) 3–10
21. De Marneffe, M., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC. Volume 6., Citeseer (2006) 449–454
22. Sleator, D., Temperley, D.: Parsing english with a link grammar. Arxiv preprint cmp-lg/9508004 (1995)
23. Cer, D., de Marneffe, M., Jurafsky, D., Manning, C.: Parsing to stanford dependencies: Trade-offs between speed and accuracy. LREC 2010 (2010)
24. : Index to link grammar documentation `http://www.link.cs.cmu.edu/link/dict/`.
25. Banerjee, S., Pedersen, T.: An adapted lesk algorithm for word sense disambiguation using wordnet. Computational Linguistics and Intelligent Text Processing (2002) 117–171
26. Costantini, S., Paolucci, A.: Towards translating natural language sentences into asp. In: Proc. of the Intl. Worksh. on Answer Set Programming and Other Computing Paradigms (ASPOCP), Edimburgh. (2010)

27. Costantini, S., Paolucci, A.: Semantically augmented DCG analysis for next-generation search engine. CILC (July 2008) (2008)
28. Eiter, T.: Answer set programming for the semantic web. Logic Programming (2010) 23–26
29. Schindlauer, R.: Answer-set programming for the Semantic Web (2006)
30. Church, A.: A set of postulates for the foundation of logic. The Annals of Mathematics **33**(2) (1932) 346–366