A grammatical formalism based on patterns of Part of Speech tags*

Pablo Gamallo Otero and Isaac González López
University of Santiago de Compostela

In this paper, we describe a grammatical formalism, called DepPattern, to write dependency grammars using patterns of *Part of Speech* (PoS) tags augmented with lexical and morphological information. The formalism inherits ideas from Sinclair's work and Pattern Grammar. To properly analyze semi-fixed idiomatic expressions, DepPattern distinguishes between open-choice and idiomatic rules. A grammar is defined as a set of lexical-syntactic rules at different levels of abstraction. In addition, a compiler was implemented so as to generate deterministic and robust parsers from DepPattern grammars. These parsers identify dependencies which can be used to improve corpus-based applications such as information extraction. At the end of this article, we describe an experiment which evaluates the efficiency of a dependency parser generated from a simple DepPattern grammar. In particular, we evaluated the precision of a semantic extraction method making use of a DepPattern-based parser.

Keywords: syntax, parsing, pattern grammar, dependency grammar, information extraction

1. Introduction

Within the tradition of corpus linguistics, we can find two assumptions that do not follow the main lines of most standard linguistic approaches. On the one hand, it is assumed that grammar and lexis cannot be separated (Sinclair 1991), that they "are one and the same thing" (Hunston & Francis 1999: 272). On the other hand, it is also assumed that sense and structure are associated (Sinclair 1991): "There is a strong association between meaning and pattern" (Hunston & Francis 1999: 272).

Similar ideas are also clearly stated in cognitive linguistics. In particular, Cognitive Grammar makes the two assumptions stated above, but uses its own terminology. On the one hand, grammar is seen as a structured inventory of units that

International Journal of Corpus Linguistics 16:1 (2011), **45–71**. DOI 10.1075/ijcl.16.1.03gam
ISSN 1384–6655 / E-ISSN 1569–9811

embody the regularities discernible at varying levels of abstraction. Continuous and discontinuous idioms, strong collocations, semi-productive rules, productive syntactic patterns, and so on are all represented and defined in the same way, namely as units of meaning. Hence, "lexicon, morphology, and syntax form a continuum of meaningful structures" (Langacker 1991: 3). On the other hand, it is also claimed that "grammar is inherently meaningful" (Langacker 2003: 251), and thus cannot be perceived as an autonomous module *vis-à-vis* semantics.

Furthermore, dependency-based grammars also share, to a certain extent, the same principles. Word Grammar states that there are no clear boundaries between different areas of knowledge — e.g. between lexicon and grammar. In more conclusive terms, Hudson claims that "the lexicon is not distinct from the grammar" (Hudson 1990). In addition, other researchers within the same framework assert that one of the main advantages of dependencies over phrase structure representations is "closeness to semantic representations such as predicate-argument structures" (Debusmann & Kuhlmann 2010: 365). Dependency links, which form abstract syntactic structures, are therefore directly associated with meaning.

Let's note that three very different linguistic frameworks — corpus linguistics, cognitive linguistics, and dependency grammars — proposed, at the end of the eighties, the same unconventional principles. This should not be taken as a coincidence, but as sound evidence of their reliability. However, in spite of the soundness of these two principles, most formal grammars elaborated so far whose aims are to improve natural language processing tools (e.g. syntactic analyzers) don't follow such ideas. In fact, they rely on more standard linguistic theories that assume a sharp separation between syntax and lexicon, as well as the autonomy of syntax *vis-à-vis* semantics. Two significant exceptions are Constraint Grammar (Karlsson 1990) and Lexicalized Tree Adjoining Grammar (Abeillé et al. 1989).

Since most formal grammars are based on linguistic frameworks that do not take into account the two principles introduced above, they have serious problems coping with those collocations and idioms that are seen as semi-fixed lexical units made up of several words. The importance of this type of composite expression has been largely neglected in the development of grammatical formalisms. There are many composite lexical units that do not necessarily appear as continuous strings in texts, that is, they are discontinuous structures, e.g. "*take* NP *into account*", "*turn* NP *on*", etc. In addition, many idioms allow for internal lexical and syntactic variation. For instance, the semi-fixed unit "*BE-NEG-in-POSS-nature*" can be elaborated by a great variety of specific expressions*: is not in his nature*, *was hardly in your nature*, *is not in the nature of the chairman*, *is not in Peter's nature*, etc. In fact, such a lexical unit contains at least five words, but only two of them — *in* and *nature* — are actually fixed. The remaining words have some kind of lexical and syntactic variation. In order to recognize semi-fixed lexical units, they should be

approached in a fashion similar to the way any syntactic structure would be analyzed. It is not realistic to treat them as static and predefined entries in the lexicon. In fact, a proper treatment of semi-fixed idioms constitutes a serious challenge for those formal grammars and natural language systems based on linguistic theories that conceive lexicon and syntax as being two clearly separated modules.

In this paper, we shall describe a rule-based formalism, called DepPattern, which is sensitive to the two principles introduced above. Its rules aim to identify dependencies by using unlimited patterns of Part-of-Speech (PoS) tags, morphological features, and lexical information. In addition, a set of related dependencies can produce either a (semi)-fixed lexical unit (idiom principle), or a standard syntactic unit (open-choice model). Thus, in DepPattern, lexis and syntax are not separated, and yet at the same time the natural association between dependencies and semantics allows DepPattern to be easily adapted for the task of semantic extraction. Since dependencies are semantically motivated, many researchers on Information Extraction use syntactic dependencies rather than constituency structures to automatically acquire word similarity from large corpora. This explains why most work on syntactic-based semantic extraction only makes use of word dependencies instead of phrase constituents (Grefensttete 1994, Lin 1998, Gamallo et al. 2005).

We have implemented a DepPattern compiler which generates deterministic and robust parsers based on regular expressions. Parsers generated from DepPattern can be used for five languages: English, French, Spanish, Portuguese, and Galician.[1] They parse text that was previously PoS tagged by either FreeLing (Carreras et al. 2004) or Tree-Tagger (Schmid 1994), and process naturally-occurring text to identify dependencies which can be used, in turn, to extract semantically related words.

This paper is organized as follows: In the following section (Section 2), we describe some of the linguistic ideas underlying the grammatical formalism. These ideas have been drawn from Sinclair's work, Pattern Grammar, and dependency-based linguistic theories. Section 3 will then briefly sketch two related grammatical formalisms, which also take into account the fuzzy boundaries between lexis and syntax. Section 4 depicts a general overview of the computational architecture we have implemented on the basis of the grammatical formalism: grammar compiler, parsers, PoS tags converter, etc. In Section 5, we give an informal description of the main elements of the formalism. Finally, in Section 6 a corpus-based experiment and its evaluation are presented. The aim of the experiment is to extract lexical semantic similarities using dependencies identified from a large English corpus. The input dependencies were recognized by a parser compiled from a small English DepPattern grammar.

2. Linguistic ideas underlying the formalism

To elaborate our formalism, we have taken into account ideas from different linguistic frameworks. In particular, we borrowed basic concepts from Sinclair's work, Pattern Grammar, and dependency-based linguistic theories.

2.1 Open-choice model *versus* idiom principle

Sinclair argues that there are two different ways of interpreting language expressions. On the one hand, the meaning of a composite expression is the result of a number of open choices made according to well-defined semantic compositionality. This is called the 'open-choice model', and is defined as follows (Sinclair 1991: 109–110):

> It is often called a "slot-and-filler" model, envisaging texts as a series of slots which have to be filled from a lexicon which satisfies local constraints. At each slot, virtually any word can occur. […] All grammars are constructed on the open-choice principle.

On the other hand, in many cases the meaning of a composite is not compositional, that is it cannot be derived from the meaning of its parts. These frozen and semi-fixed structures can only be interpreted by making use of the 'idiom principle' (Sinclair 1991: 110), which is defined as follows:

> The principle of idiom is that a language user has available to him or her a large number of semi-preconstructed phrases that constitute single choices, even though they might appear to be analyzable into segments.

According to Sinclair, these semi-preconstructed phrases are the general rule in language rather than the exception. So, the idiom principle should be incorporated into the basic organization of any (lexico-)grammar, together with the well-known open-choice model. The principle of idiom is at least as important as the open-choice model used to describe a particular lexico-grammar.

The problem is that semi-preconstructed phrases are much more complex to analyze than, for example, fixed expressions such as *of course* or *in spite of*, or standard syntactic units such as *red car* or *John is sleeping*. Fixed expressions are taken as standard lexical units stored in a static lexicon, whereas syntactic constructions can be analyzed by means of regular grammatical rules. By contrast, semi-fixed idioms, which should be defined as non-compositional lexical units, usually have complex syntactic properties similar to those treated by the open-choice model. For instance, many of them do not necessarily appear as continuous strings in texts:

(1) a. We certainly take the idea into account
b. How do I turn the radio on for my BlackBerry?

Many others allow internal syntactic variation, for instance:

(2) a. It is not in its nature to establish bilateral relations with the Member States
b. It is not in the nature of politics that the best men should be elected
c. To apologize is not in Bush's nature

In sum, semi-preconstructed phrases inherit properties from both fixed-expressions and "free" constructions.

To analyze such controversial expressions, the DepPattern formalism allows linguists to define grammatical rules aimed at identifying complex lexical units. These rules behave as standard syntactic rules, but instead of generating syntactic constructions, they build composite lexical units that will be used as "words" in further analyses. In fact, our formalism consists of two types of grammatical rules: both syntactic and lexical rules. The former follow the open-choice model, while the latter rely on the idiom principle.

2.2 Pattern Grammar

Pattern Grammar, such as it has been described in Hunston & Francis (2000), can be viewed as an "implementation of Sinclair's programme" (Teubert 2007: 230). This framework is focused on patterns of words, which are defined as follows (Hunston & Francis 2000: 37): "All the words and structures which are regularly associated with that word and which contribute to its meaning". A few examples of patterns are, for instance:

V n n	*I wrote him a letter*
V that	*We agreed that she was not to be told*
V n to-inf	*My advisers counselled me to do nothing*
N *on* n	*A decision on its German business*
ADJ to-inf	*The print was easy to read*

In the Pattern Grammar representation, **v** represents a verb group, **n** a noun group, **adj** an adjective group, **that** a clause introduced by *that*, **to-inf** a clause introduced by a to-infinitive form, and ***on*** a specific lexical item making up part of a pattern. The upper-case **V** (or **N**, **ADJ**) indicates that this is the word-class whose patterns we are focusing on. So, the upper-case part-of-speech represents the target word-class, while the lower-case part-of-speech and lexical items define a meaningful context of that word-class.

Pattern Grammar is a very interesting linguistic framework that has inspired Natural Language Processing tools. One of them, described in Mason & Hunston (2004), is aimed at automatically recognizing grammar patterns from corpora.

Patterns are syntactic surface structures allowing a description of language that is less abstract, more lexical, and surface in orientation. To describe a pattern, Hunston and Francis require only the names of lexical items and basic parts-of-speech (or word-classes); they attempt to dispense with other grammatical information such as constituency or functionality. In Pattern Grammar, it is assumed that the syntactic surface structure of a phrase is enough to map one-to-one with its meaning. In sum, Pattern Grammar deals with the direct association between surface grammatical structures (i.e. patterns) and meaning, without considering other levels of grammatical organization.

However, we claim that the patterns defined by Hunston and Francis are not only surface syntactic representations, but that they also implicitly contain some high-level grammatical information. In particular, the patterns introduced above are also provided with dependency information between a head and its modifiers (or dependent expressions). Let's take the pattern **N *on* n**. This structure only represents those cases were the ***on* n** complement depends on the nominal head of **N**. So, this pattern cannot be used to represent an expression such as *the paper on the table*, belonging to the sentence *he put the paper on the table*. Here, it is the verb *put* that subcategorizes the ***on* n** complement. In fact, within the pattern **N *on* n**, the head of the noun group **n** is syntactically related to the preposition ***on***, which in turn is syntactically related to the head of **N**. It follows that patterns do not only contain a surface sequence of lexical items and parts-of-speech, but that they are also organized as a complex syntactic structure containing head-dependent binary relationships among their internal elements.

Our DepPattern formalism uses part-of-speech surface chains, morphological features, and lexical items not in order to identify meaningful patterns of words, but syntactic dependencies instead. So, following the idea stated in Teubert (2007), the aim of our formalism is to complement surface patterns with more structured linguistic information such as syntactic dependencies. DepPattern can also be used to identify the meaningful patterns of a word, but this is only a side-effect of its main objective, namely, to recognize dependencies.

2.3 Dependency-Based Approach

As has been said in the two previous subsections, our formalism takes into account the distinction between the open-choice model and the idiom principle, as well as the idea that surface patterns are semantically motivated structures. In addition to this, we also take into account the underlying patterns of binary dependencies.

Dependencies have been traditionally considered to be syntactic objects. They are at the centre of many syntactic theories, known as 'dependency-based approaches', e.g. Dependency Grammar (Tesnière 1959), Mel'čuk's Meaning-Text Theory (Kahane 2003), or Word Grammar (Hudson 1990). In these theories, the two main properties of syntactic dependencies are the following: First, they are relations between individual words. Second, they are asymmetrical relations where one of these words is always subordinate (dependent) to the head.

Given these two basic properties, our formalism defines a pattern as a sequence of parts-of-speech, each one standing for an individual word, and representing an asymmetrical dependency. Every pattern contains at least two target parts-of-speech –the head and its dependent — and may contain an indefinite number of optional contextual tags (distinguished by square brackets). Let's take an example:

VERB [DT]? [ADJ]* NOUN

This pattern can be associated with a specific dependency, namely the direct object relationship between a verb, the head, and a noun, its dependent. It consists of two target parts-of-speech (VERB and NOUN) and two contextual ones: [DT]? and [ADJ]*. Such a pattern, which does not contain any explicit lexical information, will enable us to identify the direct object dependency between any verb, e.g. *to write*, and any noun, e.g. *letter*, within different types of constructions: *write a nice letter*, *write a letter*, *write nice letters*, *write letters*, etc. Notice that the wildcards "?" and "*" are the usual metacharacters to search for any sequence of the same string: [DT]? represents 0 or 1 determiners, while [ADJ]* 0 or more adjectives. In Section 5, we will give a more accurate description of the elements involved in patterns.

Another example of a pattern used by our DepPattern formalism is the following more complex structure:

VERB<lemma:*turn*> [DT]? [ADJ]* [NOUN]? PRP<lemma:*on*>

This pattern was enriched with lexical information. VERB<lemma:*turn*> represents the verb *turn* while PRP<lemma:*on*> is the preposition *on*. It can be used to identify the dependency between *turn* and the particle *on*, regardless of its syntactic variation:

(3) a. This means you must always turn the light on when the carriage is at the right
    b. When it is plugged in it will turn the red light on
    c. You shouldn't turn on the light at night

We will see later, in Section 5.6, how the formalism allows us to take into account the distinction between semi-fixed idioms and standard syntactic units.

Furthermore, most dependency grammars assume the 'uniqueness principle'. This principle states that each word has only one head, i.e. a word plays the role of dependent only once. However, some frameworks like Word Grammar (Hudson 1990) do not assume such a principle. Hudson uses multiple heads to account for different linguistic phenomena. He considers that the uniqueness principle is too restrictive. Our formalism takes into account the uniqueness principle as the main strategy in searching for new dependencies. The search strategy is the following: if a dependent word is not a head of further dependencies, then this word can be removed from the search space. However, in order to take into account ideas of other frameworks such as Word Grammar, the uniqueness principle can be suspended, if required. We will provide more details on this in Section 5.

To summarize, DepPattern is a formal grammar that makes use of patterns enriched with lexical and morphological information in order to identify binary dependencies between words. In some cases, the identified dependencies represent open-choice syntactic structures, but they may also form part of semi-fixed idiomatic units. To make the process of dependency identification faster, the search strategy is based on the uniqueness principle, which however can be skipped under certain conditions (cf. Section 5.5).

3. Related Formal Grammars

In the literature, we find at least two different grammatical formalisms that do not clearly separate lexicon from syntax: Constraint Grammar and Lexicalized Tree Adjoining Grammar.

The constraint grammar formalism (Karlsson 1990, Bick 2006) generates context-dependent rules conceived as linguistic constraints. It is based on the notion of 'eliminative' parsing. Constraints are defined to discard as many alternatives (readings) as possible. The main idea underlying the formalism is to treat morphological, lexical, syntactic and semantic disambiguation by the mechanism of eliminating improper alternatives. Each rule (i.e. constraint) adds, removes, selects or replaces a tag or a pattern of grammatical tags in a given sentence context. In other words, rules rely on the same basic operations to identify either lexical units or morpho-syntactic structures. This is a very powerful and expressive formalism. The main problem has to do with its internal complexity. The formalism is constituted by a large set of instructions and commands acting as a high-level programming language. Each rule/constraint is a complex object consisting of a domain, an operator, a target and a context condition. Without specific training in this high-level language, the process of elaborating constraint-based rules appears to be quite cumbersome for linguists.

Lexicalized Tree Adjoining Grammar also seems to mitigate the gap between lexicon and grammar to a certain extent (Abeillé et al. 1989). Lexical items are associated with a finite set of syntactic structures by defining a domain of locality over which constraints are specified. The grammar is easily lexicalized because Tree Adjoining languages are not context-free grammars: they are mildly context sensitive. This way, the same context-sensitive grammar is used for idioms as for open-choice syntactic structures. The latter are what Abeillé & Schabes (1989: 1) call "free sentences".

The main problem of lexicalized Tree Adjoining Grammar is its too-restricted notion of context. A context is defined as a local structure associated with a lexical item, which is the head of such a structure. More precisely, following the principles of the X-bar theory, local contexts correspond to the maximal projection of the category X-bar of the head. For instance, a noun phrase is seen as the local context of the head noun. However, we claim that in order to treat many controversial cases, larger contexts are required. In Hunston & Francis (2000), we can find many examples which require contexts beyond local domains. Consider the noun *privilege* followed by a to-infinitive clause:

(4) a. All those who had the privilege to know him
    b. It was my privilege to watch the game

The to-infinitive clause is syntactically dependent on *privilege* only if this noun follows some specific verbs, in particular *to have* and *to be*. By contrast, when *privilege* follows other verbs, the noun does not select the to-infinitive clause. For instance, in the following sentence:

(5) He used the privilege to control custom functions

Here, it is the verb *to use* that subcategorizes the to-infinitive clause. So, in order to elaborate a syntactic rule linking the noun *privilege* to the to-infinitive clause (its complement), it is necessary to take into account information on an external verb, which does not belong to the maximal projection of the noun. In other words, we need to specify a verb that is not part of the local context of the noun *privilege*, given that the restricted context of a head noun is only constituted by its specifiers, complements, and modifiers (Hunston & Francis 2000). In short, local contexts are not enough to identify word dependencies. Larger contexts and therefore more specific grammatical patterns, are required.

The formalism we will describe in the following section tries to overcome the two shortcomings stated here, for it is easy to grasp by linguists, and at the same time it is not restricted to local contexts.

4. Overview of the system

Our formalism enables linguists to write grammars that will be compiled into dependency-based parsers. We have implemented a DepPattern compiler, called *Compi*, brought under the GNU General Public License (GPL), which can generate deterministic parsers, written in Perl, for five different languages: Spanish, English, French, Portuguese and Galician. DepPattern parsers are also robust since they are mainly based on regular expressions and take as input any text, PoS tagged by either Tree-Tagger (Schmid 1994) or FreeLing (Carreras et al. 2004). In the Sketch Engine (Kilgarriff et al. 2004), syntactic dependencies and collocations are identified using a similar robust technique: pattern-matching over PoS tags. According to Nivre & Nilson (2003: 2),

> deterministic dependency parsing can be viewed as an interesting compromise between deep and shallow processing. It is a kind of deep processing in that the goal is to build a complete syntactic analysis for the input string, not just identify basic constituents as in partial parsing. But it resembles shallow processing in being robust, efficient, and deterministic.

DepPattern parsers are robust, deterministic and efficient: they are able to parse about 10,000 words per second on a processor Core 2 Quad, 2.8 GHz.

At http://gramatica.usc.es/pln/tools/deppattern.html, it is possible to download a linguistic toolkit under the GPL licence, with the following modules: a tag converter from different tagsets used by Tree-Tagger and FreeLing, five by default DepPattern grammars (one grammar per language), their corresponding DepPattern parsers, a grammar compiler to generate parsers from new grammars written by users, and a generic command for putting all of these modules together. The whole system is similar to the Intex architecture (Silberztein 1994), a linguistic environment to parse corpora with Finite State Transducers. The toolkit installation includes Tree-Tagger but not FreeLing, which can be downloaded from http://garraf.epsevg.upc.es/freeling/ for free installation. The system can be run on any GNU/Linux distribution.

Figure 1 depicts an overview of the whole system. Raw text is processed by one of the available PoS taggers (e.g. Tree-Tagger-English if the input text is in English), then the appropriate tag converter changes the tagger output into a new layout readable by the DepPattern parser, which was previously compiled from an English grammar. The final output of the parser is a syntactic analysis of the input text. Such an analysis is the set of dependency triplets identified by the rules of the grammar. To write a DepPattern grammar, it is necessary to know the type of information provided by the input of the parser, i.e. the output of the PoS converter. This is a plain text file with as many lines as tokens in the corpus. Each line consists

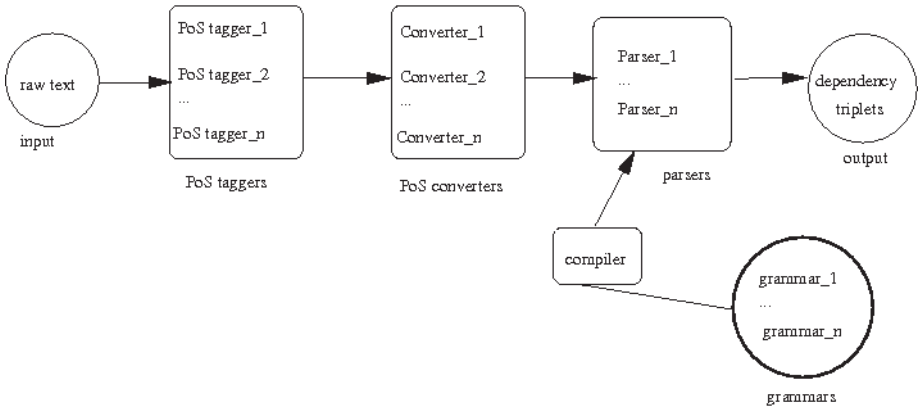

Figure 1. Computational architecture of DepPattern

of two columns: the first one contains a token and the second one all linguistic information (morphological, lexical and syntactic) associated with that token. For instance, the PoS converter transforms the expression *Mary reads good books* into these 4 lines:

```
Mary gender:0|lemma:Mary|number:S|person:3|tag:NOUN|token:Mary|type:P|
reads gender:0|lemma:read|mode:0|number:0|person:3|tag:VERB|tense:P|token:reads|type:0|
good degree:0|function:0|gender:0|lemma:good|number:0|tag:ADJ|token:good|type:0|
books gender:0|lemma:book|number:P|person:3|tag:NOUN|token:books|type:C|
```

The second column is an attribute:value structure following the EAGLES recommendations for the morphosyntactic tagging of text (EAGLES, 1996).[2] This information structure must be taken into account by the linguist when writing a DepPattern grammar. Notice the PoS tag information (in bold) is only one of the different attribute:value pairs of the structure.

In the following section, we will pay attention to the description of the formalism used to write a compilable grammar.

5. A brief introduction to the DepPattern formalism

DepPattern is a formal grammar based on context-dependent rules, augmented with morphological and lexical features, which seeks to identify the dependency structure of sentences.

In this section, we will briefly introduce some of the main properties of our formalism. More details are given in the tutorial and specific documentation.[3]

5.1 Basic description of rules

A specific DepPattern grammar is constituted by a set of context-dependent rules. Every rule aims at identifying a specific dependent-head relation by means of a pattern of part-of-speech (PoS) tags. A pattern of PoS tags is defined as a sequence of PoS tags containing, at least, two tags related by a syntactic dependency. A rule is always constituted by two elements:

- a pattern of PoS tags, which can also be enriched with lexical and morphological information;
- the name of a dependent-head relation found within the pattern.

Let's see an example:

(6) DobjR : VERB [DT]? [ADJ]* NOUN
%

The colon separates the pattern of PoS tags (at the right) from the name of the dependency: DobjR. Symbol "%" signifies the end of the rule. In this example, lexical and morphological information are not taken into account. The names of both PoS tags and dependencies must be declared in two configuration files: "tagset.conf" and "dependencies.conf", respectively. In "tagset.conf", all tags provided by the tag converter are declared. As has been said above, the goal of the converter is to unify all tagsets inherited from the different PoS taggers used by DepPattern. The linguist can modify the name of any tag declared in "tagset.conf". As far as the names of dependencies are concerned, the linguist can declare all those he/she needs to write the grammar. In the file "dependency.conf", the name of each dependency must be assigned a type. For instance, the line:

DobjR HeadDep

means that the dependency name DobjR is assigned the type "HeadDep". DepPattern defines two basic types, "DepHead" and "HeadDep", according to the position of the dependent with regard to the head. "DepHead" type is instantiated by those dependencies containing a dependent node appearing to the left of the head. "HeadDep" type is instantiated by those containing a dependent appearing to the right. There can be an indefinite number of contextual tags between the head and the dependent. Since DobjR was declared to be of type "HeadDep", it can be used to identify nouns appearing to the right of verbal heads ("Dobj" stands for Direct Object and "R" for right). In the rule in (6) above, DobjR permits identifying a dependency relation between the two PoS tags that are not distinguished by square brackets: VERB and NOUN. Since DobjR is assigned the "HeadDep" type, it classifies the verb as the head and the noun as the dependent. The remaining tags,

which are distinguished by square brackets, represent the context of the relation. In particular, [DT]? means that there can be none or one determiner, and [ADJ]* none or several adjectives, all of them between the verb and the noun.

5.2 Output: Dependency triplets

As has been said, compiled rules (i.e. the parser) take as input any text previously PoS tagged with Tree-Tagger or FreeLing and transformed by a tag converter into an ordered list of tokens with attribute-value structures, such as those shown in Section 4. The output of the parser is a list of dependency triplets. Let's suppose the system has correctly pre-processed the expression *reads a good book*. Considering the rule in (5), described above, the parser would yield as output the following triplet:

(DobjR; read_VERB; book_NOUN)

The first element, DobjR, is the name of the dependency, the second one is the head and the third one is the dependent unit. Both the head and the dependent are provided here with their corresponding PoS tag. This representation is only an approximation of that returned by the system. In order to simplify the description of the triplets, we leave the remaining linguistic information (token, number, gender, tense, etc.) out of this simplified representation. Information on token positions within the sentence is also removed. In sum, to make the reading easier, each analyzed word will be described within the triplets with only two elements: a lemma and a PoS tag.

5.3 Attribute-value information and operations on attributes and values

Further elements can be used to elaborate different aspects of a rule, namely morphological features, specific lemmas, lexical classes, and operations such as agreement, recursivity, inheritance, change of values, addition of new attribute-value pairs, etc. Let's take the following examples:

(7)
```
AdjnL : ADV<type:Q> ADJ
%
AdjnL : ADJ NOUN
Agr: number, gender
%
```

In the first rule of (7), the attribute-value <type:Q> elaborates the information about the adverb tag. It is filled by quantifier (Q) adverbs such as *very* or *quite*. In the second rule above, "Agr" stands for the operation of agreement, where

"number, gender" identify the names of the attributes whose values must be shared by both the head and the dependent. Besides, DepPattern permits the use of other operations such as "Inherit" or "Add". The former allows the linguist to select some values of the dependent in order to assign them to the corresponding head attributes. The latter can be used to either modify selected values or to add new attribute-value pairs (e.g. semantic features) to the information structure of the head.

Finally, lexical classes containing lists of lemmas can be declared in a configuration file: "lexical_classes.conf". The variables associated with those lists will be used within the rules. For instance, we can define the extensional class of verbs requiring "human" subjects by creating a specific variable, $Human_Verbs, associated with the list of such verbs. Then, the linguist can use that variable to write a lexical restriction within a rule aimed to identify the subject of verbs.

5.4 The uniqueness principle

Most dependency grammars presuppose the 'uniqueness principle', which states that each word has only one head, i.e. a word plays the role of dependent only once. Default rules are applied by taking into account such a principle. Accordingly, a rule not only identifies a dependency between two words, but also removes the dependent word from the input of the next rule to be applied. The fact of removing the dependent each time a rule is applied enables the linguist to simplify the definition of patterns within the rules. In other words, the removal of dependent nodes reduces the search space since it shortens the number of possible combinations of tags. This algorithm is inspired by the "shift-reduce" transition used by some deterministic dependency-based parsers (Nivre 2005). Let's see an example in (8). Let's suppose that we build a simple grammar with the following two rules:

(8) SpecL : DT NOUN
%
AdjunctL : ADJ NOUN
%

These rules can be used to analyze an expression such as *a beautiful mountain*. The input of the rules (output of the tag converter) is the following sequence of tokens and attribute-value structures:

```
a          gender:0|lemma:a|number:0|person:0|possessor:0|tag:DT|token:a|type:0|
beautiful  degree:0|function:0|gender:0|lemma:beatiful|number:0|tag:ADJ|token:beatiful|
           type:0|
mountain   gender:0|lemma:mountain|number:S|person:3|tag:NOUN|token:mountain|type:C|
```

The PoS tags assigned to the three tokens are in bold: DT, ADJ and NOUN. The first rule of (8) to be applied is AdjunctL, which identifies the relation between the adjective *beautiful* (dependent) and the noun *mountain* (head). Once this dependence is identified, the dependent token (i.e. *beautiful*) and its attribute-value structure are removed from the search space, i.e. from the input sequence. This gives rise to a reduced tagged text:

```
a          gender:0|lemma:a|number:0|person:0|possessor:0|tag:DT|token:a|type:0|
mountain   gender:0|lemma:mountain|number:S|person:3|tag:NOUN|token:mountain|type:C|
```

The new input no longer contains the interpolated adjective. This situation enables the other rule in (8), SpecL, to be applied. Notice that SpecL only succeeds when all adjunct adjectives have been removed. Now, SpecL both identifies the determiner-noun dependency and removes the dependent determiner from the input. The removal of dependent tags from the input tagged text leads to a systematic reduction of context information within the definition of rules. Otherwise, a rule such as SpecL, aimed at identifying a determiner-noun dependency, would require a pattern with, at the very least, a contextual adjective between both tags. Therefore, it follows that the strategy based on the uniqueness principle helps to simplify the definition of generic rules; it narrows the use of context tags only to restrictive rules coping with more irregular cases. Generic rules with abstract patterns such as those described in our example (8) do not need contextual tags, as they are systematically removed by previous rule applications. The main drawback of such a strategy derives from the fact that grammar is not fully declarative. The order in which rules are applied is significant. However, this situation has a clear advantage. Rules are provided with a search control strategy long utilized in procedurally-oriented grammars, making parsing robust and deterministic.

The analysis of a sentence is an iterative process that stops when there are no more rules to apply. As a result, the parser generates a set of triplets representing those dependencies identified by the rules. The output generated by the two rules described in (8) is the following two triplets:

(SpecL; mountain_NOUN; a_DT)
(AdjunctL; mountain_NOUN; beautiful_ADJ)

5.5 Environments without the Uniqueness Principle

In many cases, however, ruling out uniqueness allows us to yield a richer dependency analysis. We use local environments where the removal of dependents is not allowed, in order to deal with syntactic ambiguity, words with more than one head, semi-fixed idioms, etc. For instance, let us regard a case of one word likely

to have two heads. Objective complements are functions that can be described as adjectives somehow dependent on both the verb and the direct object. Take the sentence:

(9) Such experiences make life worthwhile

It is possible to propose an analysis that links the adjective *worthwhile* to both the verb *make* and the noun *life* (and not to the subject). This analysis is only possible if the uniqueness principle is locally suspended. For this purpose, we use a local environment, described as a NEXT structure of rules:

(10) AdjunctR: [VERB] NOUN ADJ
NEXT
AdjunctR: VERB [NOUN] ADJ
%

This NEXT structure allows us to apply an indefinite sequence of rules without removing the dependent tags from the search space. This results in a fully declarative grammatical environment. Thus, given the sentence in (9), we are able to grasp the two different dependent relationships held by the adjective: its relation to both the noun and the verb. The NEXT structure identifies the following two dependency triplets:

(AdjunctR; life_NOUN; worthwhile_ADJ)
(AdjunctR; make_VERB; worthwhile_ADJ)

As we will observe in the next section, extraction information systems are more interested in syntactic-semantic dependencies than in purely syntactic links. The link between *worthwhile* and the verb *make* is provided with more generic syntactic information than that between *worthwhile* and *life*, which is semantically motivated. Therefore, the latter is more significant for the task of semantic extraction.

The NEXT environment can also be used to deal with other linguistic phenomena such as syntactic ambiguity. For instance, the ambiguous attachment of a prepositional phrase to either a noun or a verb (known as "PP attachment") can be easily introduced within a NEXT environment, which allows both dependencies to be identified. This is a similar case to that described above in (10). The ambiguous preposition is linked to two different heads before being removed from the input chain. This way, the two readings of the ambiguous expression *buy the books for children* in (11) can be treated as shown in (12):

(11) a. The foundation helps donors to buy books for children in South Africa
b. Buy books for children online

(12) PrepCompR : [VERB] NOUN PRP
NEXT
PrepCompR : VERB [NOUN] PRP
%

PrepCompR stands for prepositional complements appearing to the (R)ight of the head. In (12), the preposition is attached both to the head noun by the first rule, and to the head verb by the second one. It gives rise to two dependency triplets representing the two possible prepositional attachments:

(PrepCompR; buy_VERB; for_PRP)
(PrepCompR; book_NOUN; for_PRP)

As we will show in the next section, the NEXT environment is also useful in defining semi-fixed idiomatic expressions.

Finally, DepPattern also allows us to define *global* environments where the uniqueness principle is not operative at all. This is performed using the "NoUniq" command, which prevents rules from removing dependents. An extensive use of this command makes the grammar fully declarative, but forces the linguist to define rules with long patterns, since he/she is required to take into account all possible contextual tags between the dependent and the head.

5.6 Types of dependencies

As has been said before in Section 5.1, our formalism allows the linguist to define the dependencies he/she considers necessary to build the grammar. If a new dependency is required, he/she must declare it in the file "dependencies.conf" with a name and an assigned type. DepPattern distinguishes two types of dependencies: open-choice and idiomatic. Open-choice dependencies yield syntactic relations between lexical units. Idiomatic dependencies also produce syntactic relations between lexical units, but in addition they generate a new lexical unit by modifying the lemma of the head. So far, all examples were described using only open-choice dependencies.

5.6.1 *Open-choice dependencies*

The difference between "open-choice" and "idiomatic" is not based on the degree of freedom. The degree of freedom in open-choice dependencies is directly associated with the number of restrictions involved in the corresponding rule. Thus, a rule restricted by many lexical and morphological features has a lower degree of freedom than a rule using only part-of-speech tags. For instance, the three rules defined below in (13) represent different degrees of freedom:

(13) AdjunctL : ADV VERB
%
AdjunctL : ADV<type:Q> ADJ
%
PrepCompR : VERB<lemma:*focus*> [NOUN]? PRP<lemma:*on*>
%

The first rule is not restricted by any morphological or lexical features. Any adverb or verb may fill the generic part-of-speech constraints. The second rule, however, contains a specific morphological constraint, which makes the syntactic choice less open. Only those adverbs belonging to the class of Q(uantifiers) fill the ADV condition. The third rule contains even more specific lexical restrictions. It is only filled if a specific verb, *focus*, and a specific preposition, *on*, co-occur in the same expression. Notice that this rule was defined as open-choice even though it conveys very specific lexical information. Lexicalized rules have a very low degree of freedom (like idiomatic rules), but they are not defined as idiomatic if they do not generate a new lexical unit. In this case, the combination of *focus* and *on* does not generate a phrasal verb *focus&on.* Lexicalized rules with open-choice dependencies are useful to avoid ambiguity in PP-attachment. Let's take, for instance, the analysis of the expression *focus the topic on education*, using the following four rules:

(14) PrepCompR : VERB<lemma:*focus*> [NOUN]? PRP<lemma:*on*>
%
DobjR : VERB NOUN
%
PrepTermR : PRP NOUN
%
SpecL : DT NOUN
Agr: number
%

The first rules that will be applied are PrepTermR and SpecL. The application of the latter removes the determiner from the input and allows, in a second iteration, for the remaining two rules to be applied. The application of these four rules gives rise to the following four dependencies:

(SpecL; topic_NOUN; the_DT)
(DobjR; focus_VERB; topic_NOUN)
(PrepCompR; topic_NOUN; on_PRP)
(PrepTermR; on_PRP; education_NOUN)

As has been said before, DepPattern allows one to work not only with single lemmas (such as <lemma:*focus*>), but also with classes of lexical words. For instance, it is possible to previously define a lexical variable containing the verbs that subcategorize the preposition *on* and use that variable instead of a specific lemma. Lexical classes are defined in the configuration file "lexical_classes.conf".

5.6.2 *Idiomatic dependencies*

Idioms are lexical units constituted by syntactically related words. We use idiomatic dependencies to link all words within idioms. The dependencies identified as idiomatic are the same as those identified as open-choice (e.g. subject, direct object, prepositional complement, adjunct, specifier, etc.). So, every open-choice dependency should have its idiomatic counterpart. The idiomatic versions of SubjL, RobjR, AdjunctL, or PrepCompR are noted as SubjL.lex, RobjR.lex, AdjunctL.lex, and PrepCompR.lex. We use by convention the "**.lex**" extension to mark dependencies as idiomatic.

The only difference between open-choice and idiomatic dependencies is the new lexical unit generated by the latter. Let's take the following rules (one idiomatic and two open-choice), defining the use of a specific phrasal verb:

(15) PrepCompR.lex : VERB<lemma:*turn*> [NOUN]? PRP<lemma:*on*>
%
DobjR : VERB NOUN
%
SpecL : DT NOUN
Agr: number
%

There is only one significant difference between the idiomatic rule PrepCompR.lex and its open-choice counterpart, PrepCompR, defined above in (14) for *focus on*. The idiomatic rule generates a higher-order lexical unit, the phrasal verb *turn&on*, which will be integrated in this way in all open-choice dependencies that the verb *turn* is involved in. Given the expression *turn on the radio*, the three rules described in (15) produce the following three dependencies:

(SpecL; radio_NOUN; the_DT)
(DobjR; turn&on_VERB; radio_NOUN)
(PrepCompR.lex; turn_NOUN; on_PRP)

As has been said at the end of Section 5.6.1 it would be possible to define more generic rules by declaring in the corresponding configuration file a class of transitive phrasal verbs sharing the particle *on*. To create and update classes of phrasal

verbs, nothing prevents us from using automatic strategies based on information extraction, such as that described in Fazly et al. (2009).

Let's take a moment to note that the idioms generated by the formalism represent semi-fixed and discontinuous constructions since they were built using rules possessing the same syntactic properties as those used for the standard open-choice constructions. DepPattern blurs the difference between syntax and lexis. Idiomatic rules are defined as lexicalized rules, but the formalism also allows for a definition of lexicalized rules that are not idiomatic.

To deal with more complex semi-fixed idioms, we can make use of hybrid sequences of idiomatic and open-choice rules within NEXT environments. For instance, the lexical unit associated with expressions such as *is not in its nature* or *was not in my nature* will be built by means of a NEXT structure of rules, which contains three idiomatic dependencies (AdjunctR.lex, CompR.lex and PrepCompR.lex) and one open-choice dependency (SpecL), which relates any possessive determiner to the noun *nature*:

```
AdjunctR.lex : VERB<lemma:be> ADV<lemma:not> [PRP<lemma:in>] [DT<type:P>]
[NOUN<lemma:nature>]
NEXT
CompR.lex : [VERB<lemma:be>] [ADV<lemma:not>] PRP<lemma:in> [DT<type:P>]
NOUN<lemma:nature>
NEXT
SpecL : [VERB<lemma:be>] [ADV<lemma:not>] [PRP<lemma:in>] DT<type:P> NOUN<lemma:nature>
NEXT
PrepCompR.lex : VERB<lemma:be> [ADV<lemma:not>] PRP<lemma:in> [DT<type:P>]
[NOUN<lemma:nature>]
%
```

The application of the three idiomatic rules generates a new verbal unit whose lemma is: *be&not&in&nature*. The possessive determiner does not take part in the lemma since its realization is open to many choices. This determiner is integrated into the lexical unit, not as part of the lemma, but as an element of the expression's internal syntactic structure. To deal with other possessive realizations of the same unit (e.g. *is not in the nature of the president*, *is not in the president's nature*, etc.), it would be necessary to define a slightly more complex NEXT structure.

DepGrammar is merely a linguistic tool. It does not solve theoretical problems such as how to decide whether an expression has or does not have an idiomatic interpretation. Linguists, and not the system, must decide what is treated as idiomatic. In particular, it is the linguist who interprets that *turn on* is idiomatic and not *focus on*. Likewise, it is she or he who makes *is not in her nature* a unit of meaning instead of *it is not in her nature to*. These linguistic decisions should be made taking into account the type of application the grammar was made for. A linguistic decision is good if it helps improve the application system or specific task

(information extraction, machine translation, summarization, etc.) that makes use of the syntactic analysis generated by the grammar. In the next section, we will describe a task-based evaluation of a small grammar written with DepPattern.

6. Information extraction from a syntactically annotated corpus

According to Kilgarriff (2003: 12), "it is of greatest interest to evaluate a system or resource according to how well it performs a task which we really want it to perform". In this section, we will describe a task-based evaluation of a DepPattern grammar. More precisely, we will perform an (indirect) evaluation of the grammar by assessing the accuracy of an application, namely the extraction of similar words, which uses a parser compiled from that grammar. Although the dependency-based parsing can be useful for any extraction task, the evaluation will be focused on word similarity extraction.

The specific objective of this section is to compare different semantic extraction methods, namely two window-based methods and one dependency-based strategy. The window-based ones extract semantically related words using simple word co-occurrences. The latter performs the same extraction by making use of syntactic dependencies. To identify dependencies, we used a parser built from a small DepPattern English grammar. This is a very generic grammar which only contains about 25 dependency-based rules.

In sum, this experiment will allow us to check whether parsers generated from DepPattern grammars may improve semantic extraction. For this purpose, we will compare the scores obtained using a dependency parser to those obtained by raw co-occurrence-based methods without syntactic information.

6.1 The task: Word similarity extraction

A well-known task in Information Extraction is the use of word/feature co-occurrences to acquire semantic information such as word similarity (Grefenstette1994, Lin 1998). Each word in the corpus is associated with a set of features (or linguistic contexts). To extract the words most similar to a target word, all strategies take into account their shared features. This relies on Harris' distributional hypothesis (Harris 1985). According to this assumption, words occurring in similar contexts (i.e. sharing many features) are considered as semantically similar. Two different strategies, windows-based and syntactic-based, can be distinguished, depending on the definition of "feature". Windows-based techniques define a feature of a target word as any word with which it co-occurs within a window of size N, where N is the number of words appearing both to the right and to the left of the target

word. This means the features of a word are its neighbors in a corpus. For instance, given the expression *Mary reads interesting books and loves cinema*, and a window of size 3 (looking at three words to the right and three words to the left of the target word) the features of *books* would be the following six words:

(*Mary, reads, interesting, and, loves, cinema)*

Syntactic-based strategies, on the other hand, define a feature of a (previously lemmatized) target word as a triplet consisting of: a dependency name, a lemma, and the syntactic position occupied by the target word (Gamallo et al. 2005). For instance, given the same expression cited above along with a dependency-based analysis, the features of *book* would be the following two triplets:

(DobjR; read; X)
(AdjunctL; X; interesting)

Where X stands for any lemmatized noun (e.g. *book*), appearing in that syntactic position. Note that features based on syntactic dependencies are more informative and precise (even if smaller in number) than those selected from windowing techniques. It is assumed that since linguistic dependencies involve specific semantic relationships, they should be considered as fine-grained clues for identifying semantically related words.

In our experiments, three different strategies were evaluated: two windowing techniques and one syntax-based method. The first of these defines features using large windows (a string between two full-stops) of tagged words. The second strategy makes use of a smaller window (size 2), but considers word order. In both cases, function words were previously removed. The third strategy, which is syntax-based, uses dependencies to define features. The DepPattern parser used to identify dependencies was compiled from a small English grammar.

6.2 The corpus

Experiments were performed on the British National Corpus (BNC), which contains about 100 million word tokens.[4] The corpus was PoS tagged with Tree-Tagger and syntactically analyzed with a DepPattern parser. This task took less than 3 hours. Then, the 10,000 most frequent nouns were selected as target words for evaluation.

6.3 The evaluation protocol

Each one of the three strategies was tested on the list of target words, namely the 10,000 most frequent nouns of BNC. For each target word of the list and for each strategy, a ranking of the 10 most similar words was obtained, using 3 different

similarity coefficients, *cosine*, *dice* and *jaccard*, which made a total of 3x3 different experiments.

To evaluate the quality of the word similarity extraction, the synsets of WordNet (Fellbaum 1998) were selected as gold standard. The automatic evaluation consisted of measuring the quality (in terms of *precision*) of the 10 most similar word candidates for each noun. For this purpose, given each evaluated noun and its 10 similar candidates, we were able to automatically check whether the candidates are semantically related in WordNet to the evaluated noun. Besides related words by synonymy (same synset), the co-hyponymy relation was also taken into account, assuming that nouns sharing a direct hyponym are co-hyponyms. In summary, for each experiment, we evaluated 100,000 (10,000 x 10) semantically related candidates. Precision was defined as the number of candidates found in WordNet (synonyms and co-hyponyms) divided by the total number of candidates (i.e. 100,000).

Let's see an example. Table 1 shows the list of the 10 most similar words obtained by each of the three strategies for the noun *textbook*. Underlined words (in bold) are those considered to have been correctly extracted, since they were found to be semantically related to *textbook* in the gold standard, i.e. in WordNet. As the table shows, the dependency-based method extracted 4 out of 10 correct similar words. The windowing techniques were less precise: 2 or less correct words. Notice that this automatic evaluation is far from being perfect. WordNet does not take into account some possible correct co-hyponyms of *textbook*, e.g. *handbook*, *essay*, *dictionary*, etc.

Table 1. Results of three strategies for the noun textbook

	Similar words to *textbook*
Window (large)	*topic*, *mathematics*, *literature*, *subject*, *library*, *syllabus*, *teaching*, ***text***, *reading*, *essay*
Window (2+word order)	*creation*, *dictionary*, ***journal***, *comfort*, *conception*, *slope*, *edition*, *interview*, *lord*, ***catalogue***
Dependency-based	***journal***, *dictionary*, *handbook*, *essay*, ***brochure***, ***leaflet***, ***booklet***, *discourse*, *manuscript*, *accounting*, *equation*

6.4 Results

Table 2 depicts the quantitative results obtained for each strategy and each similarity measure. The best scores were achieved using the syntactic dependencies identified with a DepPattern parser: it reached a precision rate of more than 15%, compared to 11.5% and 8.7% obtained by the two window-based methods. Bordag (2008), performed a very similar experiment also using the BNC corpus and

Table 2. Precision of the three evaluated strategies

	Cosine Precision (%)	Jaccard Precision (%)	Dice Precision (%)
Window (large)	8.74	8.11	8.11
Window (2+word order)	11.50	10.14	10.14
Dependency-based	***15.18***	***12.97***	***12.97***

WordNet. The author evaluated a window-based strategy whose best scores were also very similar to those we obtained with the same strategy: about 8% precision. This seems to prove that our grammatical formalism is suitable for generating useful dependency-based parsers, that is, parsers improving NLP applications such as semantic extraction.

However, there is still room for improvement. The dependency-based parser was generated from a very generic DepPattern grammar, which contained only some open-choice rules. There were neither lexicalized nor idiomatic rules. In future work, we will elaborate different grammars at different levels of abstraction (only open-choice rules, open-choice + lexicalized rules, open-choice + lexicalized + idiomatic rules, etc.), in order to evaluate their efficiency for the specific task of semantic extraction. As has been pointed out earlier, we consider that indirectly evaluating a parser against an NLP application can be more informative than evaluating it directly against a treebank. The task-based evaluation allows us to know whether the underlying grammar is or is not useful for a specific NLP application.

7. Conclusions

This paper sketched some properties of an expressive rule language, DepPattern, aimed at defining dependency grammars using patterns of PoS tags enriched with morphological and lexical features. Unlike most similar formalisms, our proposal relies on the main assumptions underlying traditional work on corpus linguistics: (i) that lexis and grammar are not clearly separated, and (ii) that surface syntactic patterns embody relevant semantic information. In addition, the formalism is based on a simple language including regular expressions and is easy to grasp by linguists without a particular or laborious training.

One of the main contributions of DepPattern is the distinction between open-choice and idiomatic rules. The objective is to deal with semi-fixed expressions in an appropriate way, that is to take into account their syntactic variation, as well as to consider them as lexical units of meaning.

It is generally assumed that a richer set of syntactic dependencies improves semantic extraction. The output of the parsers compiled from DepPattern grammars is easily adapted for use in semantic extraction. This will allow us to properly evaluate the efficiency of the rules defined in the grammar. Those rules giving rise to the best precision scores should be considered to be the most semantically motivated, and therefore the most useful for the extraction task at hand. In fact, we consider that the feedback provided by the extraction task will help to modify and improve the definition of the grammar, which in turn, after having been modified, should also make the extraction process better. In future work, we aim to develop a bootstrapping architecture based on the intercommunication among different modules such as grammar construction, parser generation, semantic extraction, and automatic evaluation.

Finally, we claim our formalism to be useful for writing not only general-purpose grammars, but also local grammars (Gross 1993, Silberztein 1994, Mason 2004). Local grammars are suited to make low-level descriptions of many grammatical phenomena (semi-fixed idioms, specific patterns in controlled languages, etc.) that escape a systematic description in terms of abstract syntactic rules. In sum, DepPattern allows us to write both coarse-grained grammars aimed to deal with abstract and systematic phenomena, and fine-grained rules organized in local grammars coping with very specific and irregular cases.

Notes

* this work has been supported by the MICINN, within the project FF12010:14986.

1. The compiler and five compiled parsers are freely available (GPL license) at: http://gramatica.usc.es/pln/tools/deppattern.html

2. EAGLES: http://www.ilc.cnr.it/EAGLES/home.html

3. A tutorial on the formalism is available at http://gramatica.usc.es/pln/tools/tutorialGrammar.pdf. The user guide of the system is at http://gramatica.usc.es/pln/tools/user_guide.pdf

4. http://www.natcorp.ox.ac.uk/

References

Abeillé, A. & Schabes, Y. 1989. "Parsing idioms in lexicalized TAGs". In H. Somers & M. M. Wood (Eds.), *Proceedings of the fourth conference on European chapter of the Association for Computational Linguistics*, *Manchester, England, 10–12 April*, 1–9.

Abeillé, A., Bishop, K., Cote, S., Joshi, A. & Schabes, Y. 1989. "Lexicalized TAGs, parsing and lexicons". In L. Hirshman (Ed.), *Proceedings of the Workshop on Speech and Natural Language, Philadelphia, Pennsylvania, 15–18 October*, 210–215.

Bick, E. 2006. "A constraint grammar-based parser for Spanish". *Proceedings of the 4th Workshop on Information and Human Language Technology, Ribeirão Preto, Brazil, 27–28 October*, 127–138.

Bordag, S. 2008. "A comparison of co-occurrence and similarity measures as simulations of context". In A. Gelbukh (Ed.), *Proceedings of the 9th international conference on Computational Linguistics and Intelligent Text Processing, Haifa, Israel, 17–23 February*, 52–63.

Carreras, X., Chao, I., Padró, L. & Padró, M. 2004. "An open-source suite of language analyzers". In N. Calzolari (Ed.), *Proceedings of the 4th International Conference on Language Resources and Evaluation. Lisbon, Portugal, 26–28 May*, 239–242.

Debusmann, R. & Kuhlmann, M. 2010. "Dependency Grammar: Classification and Exploration". In M. W. Crocker & J. Siekmann (Eds.), *Resource-Adaptive Cognitive Processes*. Berlin/Heidelberg: Springer-Verlag, 365–388.

Fazly, A., Stevenson, S. & Cook, P. 2009. "Unsupervised type and token identification of idiomatic expressions". *Computational Linguistics*, 35 (1), 61–103.

Fellbaum, C. 1998. "A semantic network of English: The mother of all WordNets". *Computers and Humanities*, 32 (2–3), 209–220.

Gamallo, P., Alexandre, A. & Lopes, G. P. 2005. "Clustering syntactic positions with similar semantic requirements". *Computational Linguistics*, 31 (1), 107–146.

Grefensttete, G. 1994. *Explorations in Automatic Thesaurus Discovery*. Boston: Kluwer Academic Publishers.

Gross, M. 1993. "Local grammars and their representation by finite automata". In M. Hoey (Ed.), *Data, Description, Discourse*. London: HarperCollins, 26–38.

Harris, Z. 1985. "Distributional Structure". In J. J. Katz (Ed.), *The Philosophy of Linguistics*. Abingdon/New York: Oxford University Press, 26–47.

Hudson, R. 1990. *English Word Grammar*. Oxford: Blackwell.

Hunston, S. & Francis, G. 2000. *Pattern Grammar*. Amsterdam/Philadelphia: John Benjamins.

Kahane, S. 2003. "Meaning-Text Theory". In V. Agel, L. Eichinger, H.-W. Eroms, P. Hellwig, H. J. Heringer & H. Lobin (Eds.), *Dependency and Valency. An International Handbook of Contemporary Research,* Vol. 1, Berlin/New York: de Gruyter, 546–569.

Karlsson, F. 1990. "Constraint Grammar as a framework for parsing running text". *Proceedings of the 13th Conference on Computational Linguistics, Helsinki, Finland, 20–25 August*, 168–173.

Kilgarriff, A. 2003. "Thesauruses for Natural Language Processing". In C. Zong (Ed.), *Proceedings of Natural Language Processing and Knowledge Engineering* (NLPKE). *Beijing, 25–27 October*, 5–13

Kilgarrif, A., Rychly, P., Smrz, P. & Tugwell, D. 2004. "The Sketch Engine". In G. Williams & S. Vessier (Eds.), *Proceedings of the Eleventh EURALEX International Congress*, *Lorient, France, 6–10 July*, 105–116.

Langacker, R. W. 1991. *Foundations of Cognitive Grammar, Vol II: Descriptive Application*. Stanford: Stanford University Press.

Langacker, R. W. 2003. "Constructional integration, grammaticalization, and serial verb constructions". *Languages and Lingusitcs*, 4 (2), 251–278.

Lin, D. 1998. "Automatic retrieval and clustering of similar words". *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, *Montreal, Canada*, *10–14 August*, 768–774.

Mason, O. 2004. "Automatic processing of local grammar patterns". In M. Lee (Ed.), *Proceedings of the 7th Annual Colloquium for the UK Special Interest Group for Computational Linguistics, Birmingham, UK, 6–7 January*, 166–171.

Mason, O. & Hunston, S. 2004. "The automatic recognition of verb patterns: A feasibility study". *International Journal of Corpus Linguistics*, 9 (2), 253–270.

Nivre, J. 2005. "Dependency grammar and dependency parsing". *MSI report 05133*. Växjö University: School of Mathematics and Systems Engineering. Also available at: http://stp.lingfil.uu.se/~nivre/docs/05133.pdf (accessed November 2010).

Nivre, J. & Nilsson, J. 2003. "Three algorithms for deterministic dependency parsing". *Proceedings of 14th Nordic Conference of Computational Linguistics, Reykjavik, Iceland, 30–31 May*, 1–8.

Schmid, H. 1994. "Probabilistic Part-of-Speech tagging using decision trees". In D. Jones (Ed.), *Proceedings of the International Conference on New Methods in Language Processing*, *Manchester, UK, 14–16 September*, 44–49.

Silberztein, M. D. 1994. Intex: A Corpus Processing System. *Proceedings of the 15th International Conference on Computational Linguistics*, *Kyoto, Japan*, *5–9 August*, 579–583.

Sinclair, J. McH. 1991. *Corpus, Concordance, Collocation*. Oxford: Oxford University Press.

Tesnière, L. 1959. *Eléments de Syntaxe Structurale*. Paris: Klincksieck.

Teubert, W. 2007. "Sinclair, pattern grammar and the question of hatred". *International Journal of Corpus Linguistics*, 12 (2), 223–248.

Authors' addresses

Pablo Gamallo Otero
Department of Spanish
University of Santiago de Compostela
15782 Santiago de Compostela
Galiza, Spain

pablo.gamallo@usc.es

Isaac González López
Department of Spanish
University of Santiago de Compostela
15782 Santiago de Compostela
Galiza, Spain

isaacjgonzalez@gmail.com