T.A.L., 2000, vol. 41, n° 1, pp. 127–156

ANSWER EXTRACTION USING A DEPENDENCY GRAMMAR IN EXTRANS

Diego MOLLÁ **Gerold SCHNEIDER**
Rolf SCHWITTER **Michael HESS***

Résumé - Abstract

Nous exposons ici l'implémentation d'un système d'extraction de réponses, ExtrAns, qui utilise la sortie d'un analyseur et d'une grammaire basés sur les dépendances. Afin d'augmenter la vitesse de calcul, l'analyseur et la grammaire utilisés sacrifient le fonctionnalisme (dans le cadre des grammaires de dépendance) au profit de la projectivité. Nous avons découvert que les structures de dépendance résultantes, bien qu'elles soient difficiles à traiter, peuvent être utilisées par ExtrAns pour trouver les dépendances syntaxiques et sémantiques nécessaires dans plusieurs des étapes du traitement linguistique. En particulier, nous mettons l'accent sur la génération de formes logiques minimales.

We report on the implementation of an answer extraction system, ExtrAns, that uses the output of a dependency-based parser and grammar. In order to increase speed, the parser and grammar used sacrifice functionalism (in the framework of dependency theory) in favour of projectivity. We have found that the resulting dependency structures, although cumbersome to handle, can be used by ExtrAns to find the syntactic and semantic dependencies needed in several of the linguistic processing stages. In particular, we focus on the minimal logical form generation.

Mots Clefs - Keywords

Système automatique (extraction de réponses), interface syntaxe-sémantique, forme logique minimale, grammaire de dépendance, fonctionnalisme et projectivité.

Automatic system (answer extraction), syntax-semantics interface, minimal logical form, dependency grammar, functionalism and projectivity.

*Computational Linguistics Group, University of Zurich. E-mail: {molla, gschneid, schwitter, hess}@ifi.unizh.ch. This research is funded by the Swiss National Science Foundation, project No. 12–53704.98.

INTRODUCTION

Dependency grammar has long been considered of largely “academic interest” only, and it clearly was not a mainstream theory of grammar. However, a number of attempts have already been made to implement the central concepts of dependency grammar in different Natural Language Processing (NLP) systems (Hellwig P. 1986; Covington M. A. 1990; McCord M. *et al.* 1992; Järvinen T. & Tapanainen P. 1997; Romacker M. & Hahn U. 1999), and it is becoming clear that this theory can very well compete with more standard, constituent-based, theories of grammar.

The current article focuses on a particular dependency grammar in a specific answer extraction application, ExtrAns, that constructs and processes logical forms. The use of a dependency grammar in ExtrAns allowed us to gain some interesting insights into the properties of dependency grammar theory, such as functionalism and projectivity and their relation to the semantic representation, within the framework of a practical system. This paper is a case study and theoretical issues are only introduced when necessary.

We start with an informal discussion of answer extraction in Section 1. In Section 2 we present a short overview of ExtrAns and show the different linguistic modules at work, including Link Grammar — the dependency grammar we use. In Section 3 we introduce the concepts of functionalism, word order and projectivity within dependency grammar, and present Link Grammar as an implementation of a dependency grammar. Section 4 is devoted to logical form generation in ExtrAns, where a wide range of issues is discussed that occur in the translation from dependency structures into logical forms.

1. ANSWER EXTRACTION

The fundamental goal of Answer Extraction (AE) is to locate those exact phrases of unedited text-based documents that answer a user query worded in natural language. AE is not an altogether new concept, since it can be seen as a specific type of information retrieval (IR) that retrieves “answer-passages” (O’Connor J. 1975). Emphasis in IR systems, however, has been traditionally placed on retrieving full documents. To our knowledge, only recently AE has received attention by the IR community, as the newly created Question Answering track in TREC-8 demonstrates (TREC-8 1999).

The ideal scenario for the use of a typical IR system is the “essay-writing” scenario. The user would use the IR system to find all the documents that are related to a specific topic. Then, the user would take his or her time to read the documents in order to achieve a full understanding of the topic. In this scenario, the IR system is optimal when recall (relevant documents retrieved / total relevant documents) is high, even at the expense of precision (relevant documents retrieved / total documents retrieved). To achieve its task the IR system uses only the content words in documents and queries and treats them as “bags of words” to be matched against each other. Function words, mor-

phology, and syntax are all ignored in this approach. This makes, for instance, the concepts of *computer design* and *design computer* indistinguishable. The "bags of words" approach takes benefit of the use of fast statistical techniques that find the most important words in the documents and compare them with the words in the query. This approach is preferred over more linguistic ones. In fact, it is a common belief that linguistic analysis is useful in a IR system only if the linguistic analysis remains very localised and shallow in nature (Lewis D. D. & Sparck Jones K. 1996).

In contrast, the ideal scenario of an AE system is the "problem-solving" scenario. The user has a specific problem to solve and the solution must be found, often under time pressure. In this scenario, a short list of specific answers to particular questions would be more useful than a long list of complete documents. The use of "bags of words" techniques is more limited in AE, because the text fragments retrieved are much shorter and the connection with the query is much more restricted: the text must directly answer the query. To find out how an IR system behaves in an AE environment, ExtrAns' document collection was split into 3124 files, one per sentence. Subsequently, ZPRISE (Downey L. L. & Tice D. M. 1999) was used to process this information. The results of a small evaluation with 30 short English questions showed an average of 0.7 for recall and 0.04 for precision (only the first 100 sentences per query were considered for the evaluation). ZPRISE obtained acceptable recall by returning far more sentences than needed by the user.

We think that the use of deeper linguistic techniques would improve the quality of the results. Moreover, AE offers a convenient environment for systems that use these techniques. Typical applications of AE include interfaces to machine-readable technical manuals, on-line help systems for complex software, help desk systems in large organisations, and public enquiry systems accessible over the Internet. These applications use technical manuals with a well-defined formatting and a technical language — sometimes even controlled languages (Farrington G. 1996). The amount of data that these applications use is also smaller and therefore more manageable than in typical IR applications.

The domain of ExtrAns is the set of UNIX documentation files known as the "manpages". The manpages have a very clear formatting, and the domain is technical. ExtrAns takes advantage of this and uses deep linguistic analysis to generate the logical forms of the data and the queries. The logical forms are used by conventional proof procedures to find the answer to the queries (Mollá D. *et al.* 1998). In a small evaluation with the same queries and data as above, the average values obtained were 0.39 for recall and 0.39 for precision. These values are more adequate for the AE task than those of ZPRISE. Still, these are preliminary results, and a more exhaustive evaluation is under way. The original purpose of building ExtrAns was not to find out whether deep linguistic processing is needed in AE, but to find out whether it is possible at all with the technology available today.

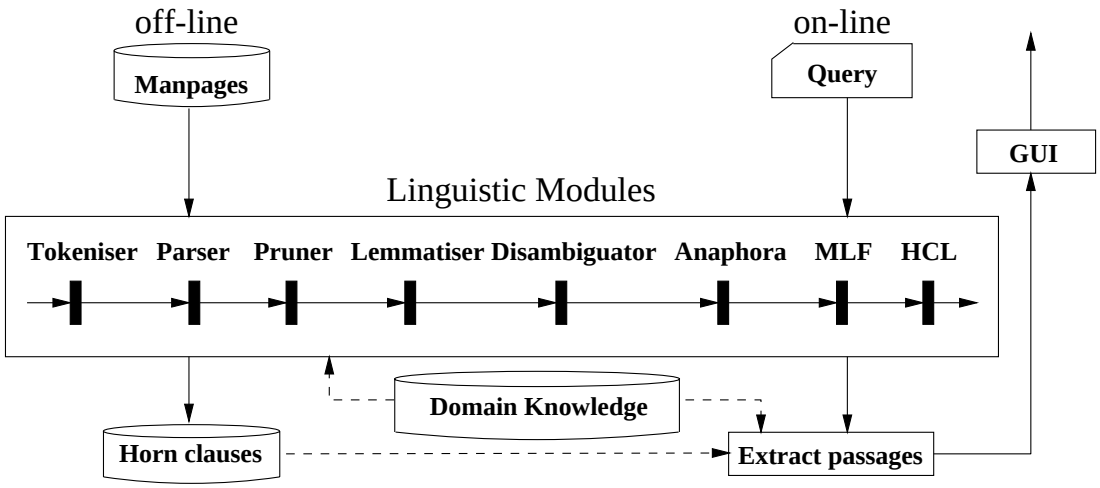


Figure 1. General architecture of ExtrAns

2. THE EXTRANS SYSTEM

2.1. Overview

In this section we present a short overview of ExtrAns, a Prolog implementation of a practical AE system (Figure 1). The current version of ExtrAns runs over 500 unedited manpages. Since ExtrAns has to cope with unedited documents, it needs a very reliable tokeniser that recognises and interprets — besides regular word forms and sentence boundaries — also formatting information. For the subsequent syntactic analysis of the manpages, ExtrAns relies on Link Grammar (LG) (Sleator D. D. & Temperley D. 1993). LG describes the structure of dependency relations between the words of a sentence by a set of labelled links which is called a linkage. The LG parser outputs the linkages for each sentence that it finds, showing the words that are linked together and the types of the links between them. For later processing steps ExtrAns converts these linkages into directed linkages by adding the dependency directions. After that, a pruner based on a set of hand-crafted heuristic rules eliminates the most obviously wrong directed linkages. As LG does not do any morphological analysis of the words in a sentence, ExtrAns' lemmatiser supplies the lemmas (root forms) of the inflected words. The lemmatiser module uses a third-party program called Morph (Humphreys K. *et al.* 1996) for this task. In the subsequent module, ambiguous attachments of prepositional phrases, gerund and infinitive constructions are disambiguated by a corpus-based approach (Brill E. & Resnik P. 1994). In the next step pronominal anaphoric references are resolved using exclusively syntactic information (Lappin S. & Leass H. J. 1994). From the resulting set of directed linkages ExtrAns constructs one or — in the case of multiple analyses — more minimal logical forms (MLFs) as semantic representation for each sentence. The MLFs are

finally converted into Horn clause logic (HCL) and asserted to the Prolog database.

In contrast to document sentences that are processed off-line, the logical form of the user query is computed on-line and then proved by refutation over the database. The logical forms retrieved in the proof indicate which words in the document sentences answer the user query directly. These words are ranked and highlighted both in the context of the sentence and the document.

A fully detailed discussion of all the linguistic modules contributing to the answer extraction process is beyond the scope of this article; in the remainder of this section we will briefly discuss the most important modules.

2.2. Link Grammar

Our choice of the syntax analysis engine was motivated mainly by practical reasons. A recent evaluation of parsers for practical applications has shown that the majority of the evaluated parsers were dependency-based (Sutcliffe R. F. E. *et al.* 1996). ExtrAns uses Link Grammar (LG) as a particular implementation of a dependency grammar for the syntactic analysis of the document sentences and the user queries. We chose LG because the parser is fast enough for the on-line processing of user queries and because it is able to handle unknown words and to skip over unanalysable parts of a sentence.[1] Since LG is strongly lexicalist, each word in the dictionary is listed together with its grammatical requirements. The default grammar/dictionary (G/D) has about 60'000 word forms and covers a wide variety of syntactic constructions.

LG uses linkages to describe the syntactic structure of a sentence. In a linkage, links connect pairs of words in such a way that the requirements of each word described in the sentence are satisfied, that the links do not cross, and that the words form a connected graph (Sleator D. D. & Temperley D. 1993).

On account of ExtrAns' particular specification and the technical domain under investigation we had to modify the parser and the G/D. An obvious extension was the addition of domain-specific words to the G/D. Although LG can handle unknown words, results are always better when such words have been categorised in advance. This is done by adding a set of specific entries to the G/D that treat the words tagged by the tokeniser as special (command names, command arguments, file names, etc.). Another modification to the G/D was the re-categorisation of certain words because they are used differently in the UNIX domain. An example is the transitive verb *print* that had to be re-categorised as a transitive verb that may form a two-word verb such as *print out*. More substantial changes had to be done in the G/D so that LG can deal with some specific syntactic structures like post-nominal modifiers for command names (e.g. *while an ls on such a link . . .*) or imperatives with fronted openers (e.g. *to quit, type q*).

[1] Under <http://bobo.link.cs.cmu.edu/link/improvements.html> an evaluation of the LG parser can be found.

2.3. Disambiguator

ExtrAns' pruner filters out all those directed linkages that are obviously wrong. But there are still many directed linkages left where ExtrAns cannot decide which is the correct one unless we use some kind of domain knowledge. This occurs in the case of attachment ambiguity. ExtrAns uses a corpus-based approach (Brill E. & Resnik P. 1994) to find the correct attachment. Brill and Resnik's original approach was designed to solve the PP attachment ambiguity of sentences with a transitive verb and a prepositional phrase (e.g. *cp copies filename1 onto filename2*) and the program was trained for the Treebank Wall-Street Journal corpus (Marcus M. *et al.* 1993). ExtrAns takes this approach one step further (Mollá D. & Hess M. 2000) and includes all categories of verbs, multiple PP attachment (e.g. *cp copies the file from A to B*), gerund and infinitive constructions (e.g. *The grun function runs 'program', using the PATH variable to find it.*). Of course, we need to specialise the disambiguator in an adequate way — therefore, it was trained with a subset of the manpages for our domain. Due to this specialised corpus the results were more accurate (76,6% correct disambiguations) than those based on the Treebank corpus (72,8% correct disambiguations) although the number of training rules derived from the manpages was far smaller (116:1770 rules).[2]

2.4. Anaphora resolution

In the current version of ExtrAns, anaphora resolution is restricted to pronominal cases. The resolution algorithm is an adaptation of a purely syntactic approach (Lappin S. & Leass H. J. 1994). This approach was designed to identify the noun phrase antecedent of third person pronouns (e.g. *If filename2*$_1$ *already exists, it*$_1$ *is removed before filename1 is moved*) and lexical anaphors (e.g. *cp copies a file*$_1$ *onto itself*$_1$) and was applied to the syntactic representation generated by McCord's Slot Grammar (McCord M. *et al.* 1992). For every antecedent candidate, the algorithm derives from the syntactic representation a measure of discourse salience. A set of factors such as recency, being in the subject position, being in the object position or being contained in another noun phrase determine this measure. On account of this measure the coreference between a pronoun and the noun phrase antecedent is established by classifying both words in the same equivalence class. An equivalence class represents those words that refer to the same object in the domain.

Since Slot Grammar is dependency-based, the relevant relations (also called slots) that were assigned to a sentence can be emulated by checking the link types returned by the LG parser. The coherence of the syntactic factors is checked by a set of syntactic rules that rely on the direction of the dependencies represented by the links. A small evaluation was carried out with a subset of the manpages and — after manually resolving the pronominal references — 80% correct resolutions were found.

[2]The complete training set was split into two parts for evaluation purposes. One part was used for training in the evaluation, and the other was used to compute the figures.

2.5. Minimal logical form

The input of the logical form generator is a set of directed linkages extended with information from the tokeniser plus a list of equivalence classes provided by the anaphora resolution algorithm. For example, the sentence *cp does not copy filename1 onto itself* is modified by the tokeniser by tagging the UNIX command *cp* (*cp.com*) and the command argument *filename1* (*filename1.arg*). The anaphora resolution algorithm creates an equivalence class with the words *filename1.arg* and *itself*. The logical form generator consults a small domain knowledge base and categorises the types of words tagged by the tokeniser as command arguments. For example, *filename1.arg* is categorised as a file.

It is important that ExtrAns is fast and robust enough for the AE task. Therefore, the logical forms must be easy to derive from the directed linkages and easy to use in the proof procedure, and yet they must be expressive enough to cope with the relevant semantic facts of the data. This is why we resort to a simple notation that consists of a conjunction of predicates where all the variables are existentially closed. This notation is easy to build, as we will see in Section 4, and it is also easy to work with. To make it expressive enough, we need to resort to reification and to a particular interpretation of the logical operators and quantification.

By reification we mean that some "abstract" concepts introduced by predicates become "concrete". As opposed to Hobbs' ontologically promiscuous semantics (Hobbs J. R. 1985), where every predicate is reified, for the time being we apply reification to a very limited number of types of predicates:

Objects. A noun like *cp* introduces the predicate `object(cp,o1,x1)`, whose meaning is "`o1` is the concept that the object `x1` is `cp`". The new entity `o1` can be used in constructions with adjectives modifying nouns intensionally (e.g. *a possible error*), or in expressions of identity (e.g. *A file is the unit of storage in UNIX*).

Events. A verb like *copies* introduces the predicate `evt(copy,e1,[x1,x2])`, whose meaning is "`e1` is the concept that `x1` copies `x2`" — `x1` and `x2` represent the objects introduced by the arguments of the verb *copy*. Reification of events is the core of the Davidsonian semantics (Davidson D. 1967; Parsons T. 1985), and is useful to express the modification of events by means of adverbs (e.g. *copy quickly*), prepositional phrases (e.g. *copy onto the hard disk*), etc.

Properties. Adjectives and adverbs introduce properties. For example, an adjective such as *blue* introduces the predicate `prop(blue,p1,x1)`, whose meaning is "`p1` is the concept that `x1` is `blue`". Reification of properties is useful when we want to modify an adjective (e.g. *the house is pale blue*).

Non-reified predicates can be introduced, too. For example, the preposition *onto* would introduce a predicate like `onto(e1,x1)`. The names of the variables

in these examples do not have any semantic content — they are all of the same type.

This notation can be used to encode the minimal logical form (MLF) of a sentence, that is, a logical form that expresses the minimal information necessary for the AE task. The current MLFs, for example, do not encode complex quantification or logical operators (apart from conjunction). Further information that is ignored includes tense and aspect, temporal relations, plurality, and modality. We can see this in the following examples of sentences and their corresponding MLFs:

(1) a. *cp.com copies files*
`holds(e1), object(cp,o1,x1), object(command,o2,x1), evt(copy,e1,[x1,x2]), object(file,o3,x2)`

b. *cp.com does not copy a file onto itself*
`not(e1), object(cp,o1,x1), object(command,o2,x1), evt(copy,e1,[x1,x2]), object(file,o3,x2), onto(e1,x2)`

c. *cp.com refuses to copy a file onto itself*
`holds(e1), object(cp,o1,x1), object(command,o2,x1), evt(refuse,e1,[x1,e2]), evt(copy,e2,[x1,x2]), object(file,o3,x2), onto(e2,x2)`

d. *if the user types y, then cp.com copies the files*
`if(e1,e2), object(cp,o1,x1), object(command,o2,x1), evt(copy,e1,[x1,x2]), object(file,o3,x2), object(user,o4,x3), evt(type,e2,[x3,x4]), object(y,o5,x4)`

All these MLFs contain the predicate `object(command,o2,x1)` because the tokeniser has tagged *cp* as a command (*cp.com*).

The words *file* and *itself* in (1b) and (1c) co-refer and result in variables that belong to the same equivalence class. During the MLF generation these variables are converted into a unique variable denoting the same entity.

The negation in (1b) is represented as a predicate over the concept of a particular copying event, and the implication in (1d) is a predicate over the concepts of a particular typing and copying event each.

In these examples we can also see how existence is handled. By default only existential quantification is used, but some of the entities have a stronger sense of existence. For example, the copying event in (1b) exists only in the Platonic universe which contains everything one can think of. The copying event in (1a), on the other hand, also exists in the world of manpages, and that is explicitly asserted by the predicate `holds` (Hobbs J. R. 1985; Hobbs J. R. 1996). The minimal logical forms in (1c) and (1d) do not state whether the copying event holds. One could argue that the event in (1c) does not hold because of the lexical meaning of *refuse*. However, for the time being we do not decompose lexical meaning, and thus we cannot deduce the negation. The information is therefore left underspecified. We assert that the refusing event holds, but we do not say anything about the copying event. If needed, and

provided that we have enough knowledge to assess it, its negation or assertion can be inferred in a later stage.

There are several reasons why we do not want to encode all the information available in the data. First of all, we believe that the logical forms should remain simple. Further complications in the logical forms could prevent a practical application from being fast and robust enough. To give an example with the sentences introduced in (1), they are all informative answers to queries like:

(2) a. *which command can copy files?*

b. *which commands copy files?*

c. *which command can copy a file?*

d. *which command copies all my files?*

However, if one were to implement modality, plurality, and quantification, one would also need to add the right inference rules to be able to retrieve (1) from (2). In particular, even if the copying event is conditional on the user's typing 'y', sentence (1d) is an informative answer and it can be inferred straightforwardly with the current MLFs. If we had converted (1d) into a logical form with a logical implication, we would have had to decide whether the antecedent *the user types y* is true (which, incidentally, cannot be decided due to lack of contextual information). In ExtrAns we do not encode semantic information that is not going to be used later or that does not significantly improve the overall performance of the system.

MLFs aim at being incrementally extensible so that new (more specific) information can be added without destroying the given information. Given their simple flat conjunctive structure, further extensions are feasible, including complex quantification (Hobbs J. R. 1996). We only need to add more elements to the MLF.

The systematic construction of the MLFs from the directed linkages is discussed in Section 4.

2.6. Horn clause logic

After the MLFs have been generated they are translated into Horn Clause Logic (HCL) for processing reasons. In the current implementation of ExtrAns only Horn clauses consisting of facts build the database, but we are not excluding the full power of HCL for the future. Depending on whether the MLF corresponds to a document sentence or a user query, variables are treated differently during the translation to HCL. In the former case the variables are skolemised and in the latter case converted into Prolog variables. To be able to display a retrieved sentence later by selective highlighting, every predicate in the HCL representation of the MLF contains a pointer to the sentence (e.g. `sent1/1` for the first interpretation) together with a list of pointers to the words (e.g. `[1]` or `[1,2,3]`) that resulted in the creation of the predicate. Example (3) shows the HCL representation of a simple document sentence:

(3) *cp.com copies files*
`holds(e1)/sent1/1~[].  object(cp,o1,x1)/sent1/1~[1].`
`object(command,o2,x1)/sent1/1~[1].`
`evt(copy,e1,[x1,x2])/sent1/1~[1,2,3].`
`object(file,o3,x2)/sent1/1~[3].`

User queries are converted into negative Horn clauses and all solutions are searched by refutation over the database. This is done by using Prolog's default proof procedure (`findall`), e.g. (2) all generate:

(4)
```
?- findall([S,I,P1,P2,P3], ( object(command,_,X)/S/I~P1,
evt(copy,_,[X,Y])/S/I~P2, object(file,_,Y)/S/I~P3 ), Results).
```

The Horn clause generator deals with synonymy by converting each word into its synonym set identifier, with the help of a small domain-specific thesaurus based on WordNet (Fellbaum C. 1998). To ease readability, this conversion is not shown in (3) and (4).

2.7. Graphical user interface

The graphical user interface (GUI) of ExtrAns displays the retrieved sentences by selectively highlighting all relevant parts of a sentence that directly answer the query. This presentation technique was developed as document sentences may answer a user query in different ways for any of the following reasons: First, a document sentence may be syntactically ambiguous, e.g. because the disambiguator can only resolve attachment ambiguities. In particular, homographs belonging to different parts of speech may produce alternative parses (in the same way as in *time flies like an arrow*), and these cannot be disambiguated for the time being. In this case ExtrAns asserts all alternative MLFs for a sentence and includes all of them in the proof. Second, a document sentence can have multiple (i.e. not exclusive) interpretations. In such cases ExtrAns will send more than one sentence to the parser and will produce alternative MLFs. This is the case for the second sentence with a comma-delimited enumeration in Figure 2 (overleaf). Third, a MLF may provide multiple answers because different sets of facts of the same logical form can prove the query. This case occurs if e.g. a document sentence has a coordinated structure. And finally, a user query can be ambiguous, too, therefore different interpretations of the same query may be proved by different parts of one MLF.

Consider, for example, the result of the query *Which command removes a file?* in ExtrAns' main window (Figure 2). All words of the retrieved document sentences that answer the user query are highlighted. Since some sentences contain unresolvable ambiguities that lead to more than one solution, each word is ranked according to its frequency in all the solutions found. The more often a word occurs in a solution the more intense its colour is in the presentation — as the second sentence in Figure 2 shows. It is therefore very easy for

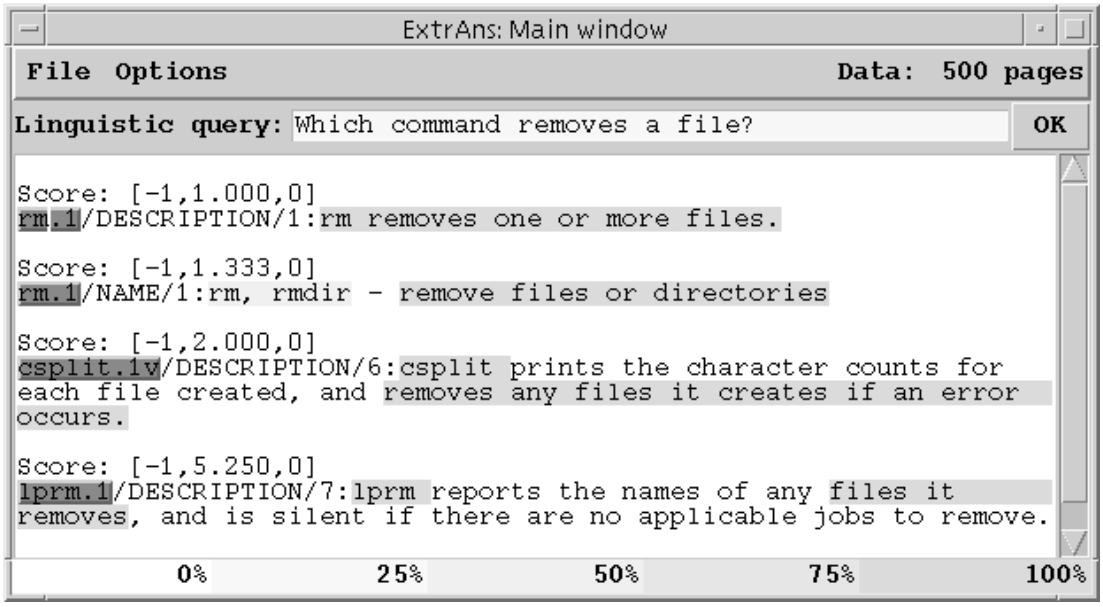


Figure 2. Reported answers in ExtrAns' main window after 3.2 secs on a Sun Ultra 10 workstation.

the user to see which parts of a highlighted sentence are particularly relevant. Relevance is seen here as a degree of unambiguity.

By clicking on the name on the left of a displayed sentence (e.g. *rm.1*), the user gets access to the full manpage. This makes it possible for the user to verify immediately in the context of the document whether the highlighted sentence contains in fact an answer to the question.

3. DEPENDENCY GRAMMAR

After this overview of the ExtrAns system, we will focus now on dependency grammar, the concepts of functionalism and projectivity, and how Link Grammar handles them.

3.1. Basic concepts of dependency grammar

There are a number of possible definitions of dependency grammar (DG). They all have in common that certain words (so-called governors) expect other words (so-called dependents), which are either compulsory (syntactic complements, semantic arguments) or optional (syntactic adjuncts, semantic modifiers). In its simplest version, DG is a constituent grammar which only knows lexical items. DG is the one grammar formalism that really takes government as its foundation (Covington M. A. 1992:4), whereas in Government & Binding (GB) no universal definition of government exists (Cook V. & Newson M. 1996:315). DG is essentially a valency grammar in which the valency concept is extended from verbs to nouns and adjectives and finally to all word classes.

On the syntactic dependency level, predicative adjectives for instance open valencies for verbs (*ready to go*), and prepositions open valencies for nouns.

Tesnière's original dependency concept (Tesnière L. 1959) aims at being a proto-semantic language-independent theory rather than merely a syntactic theory. Configurational considerations play at most a secondary role. Tesnière distinguishes between the *ordre linéaire* of the running text and the *ordre structural* which is expressed by a dependency structure. Such a dependency structure is a deep-syntactic structure, also called tectogrammatical (Sgall P. *et al.* 1986) or functional structure (Bröker N. 1998b; Järvinen T. & Tapanainen P. 1997). As word order plays no primary role, dependencies between words may also cross each other. Dependency structures without crossing dependencies and constituent structures without crossing tree branches are called projective or continuous, while crossing dependency structures or crossing tree branches are called non-projective or discontinuous. No extensions to the original DG conception of Tesnière are needed to express non-projectivity.

3.1.1. Levels of dependency

Many linguists distinguish at least between morphological, syntactic and logico-semantic dependency (Mel'čuk I. 1988; Helbig G. 1992). A difference between the syntactic and the semantic level is e.g. that while determiners depend on nouns on the syntactic level, nouns come to depend on determiners on the semantic level (Montague R. 1973).

The subdivision of the syntactic level into a surface-syntactic and a deep-syntactic level (Mel'čuk I. 1988) reflects Tesnière's distinction between *ordre linéaire* and *ordre structural.* But Tesnière was criticised because his conception of dependency is not consistently functional and sometimes resorts to surface-syntactic arguments. The mapping between surface-syntactic and deep-syntactic dependency is not isomorphic. First, in a functional approach it can be argued that prepositions are just relational markers like case (Pollard C. 1994:44-45), which allows linguists to deal with highly inflectional languages like Finnish in the same way as with isolating languages like English — the direction of the dependency is reversed between the surface-syntactic and the deep-syntactic level. Second, on the deep-syntactic level the verb forms one coherent unity, as opposed to the frequent separation of auxiliary and main verb on the surface-syntactic level. Tesnière himself addresses this problem by using his device of *translation*, which takes up all the verbal elements into a single nucleus behaving like a single word. Third, in many cases, a noun modified by a relative clause can be bound to the head verbs of the subordinate clauses on the deep-syntactic level (e.g. *rm removes the files*$_1$ *that the user chooses* t_1 *to delete* t_1).

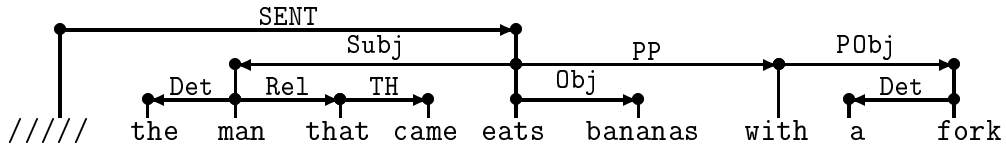
(Hudson R. 1996) suggests using additional deep-syntactic dependency links for the third case above. A dependent is allowed to have a surface-syntactic head and, in addition, a different deep-syntactic head. In such an approach, DG structures become graphs. The introduction of such extra links is a way of encoding a trace to a moving argument like in GB. Syntactic depend-

ency generally is equivalent to *government*, additional deep-syntactic links are part of the *binding* theory.

3.1.2. Constituency versus dependency

Constituent grammars are more widespread and known much better than DG, therefore a comparison between both approaches is pertinent. Any dependency structure is a coherent hierarchical structure of a whole sentence, because in dependency theory, valency is extended to all word classes. Even if constituents are not a primitive of the theory, they can easily and consistently be derived by recursively collecting all the dependents and sub-dependents of a head. The following dependency structure illustrates this:

(5) *the man that came eats bananas with a fork*



The subject of the sentence is the constituent whose head is *man*. Thus, the subject is *the man that came*. The prepositional phrase is *with a fork*, the direct object is *bananas*, and the relative clause is *that came*. If we look at (5), we can see that the directed links look like a tree, very much like in the case of the output of a constituency-based parser.

(Hays D. 1964) and (Gaifman H. 1965) have in fact proved that (at least a version of) DG is weakly equivalent to a context-free constituency grammar (CFG), and even strongly equivalent to a CFG without intermediate categories (Baumgärtner K. 1970). (Covington M. A. 1994) proves strong equivalence to X-bar syntax for those DGs whose dependency types can be matched to the X-bar primitives *specifier*, *adjunct* and *complement*. While these proofs are mathematically correct, they only apply to projective DGs. They neglect that at least Tesnière's original approach is completely non-configurational, i.e. word position is not primarily taken into consideration (Baumgärtner K. 1970; Järvinen T. & Tapanainen P. 1997; Schneider G. 1998). Tesnière's DG has free word order and is inherently non-projective. Many current DG implementations (including the dependency-based LG) still depart from Tesnière in this respect and remain projective and thus context-free and equivalent to constituency. A conversion between constituency and dependency is then possible, but some of DG's inherent functionalism is sacrificed, as we shall explain now.

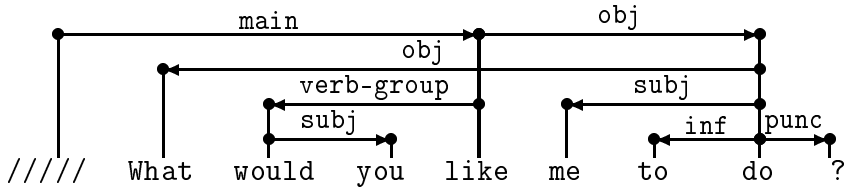
3.2. Functionalism, word order, and non-projectivity

In a functional grammar, related sentences should be assigned related or identical structures. A monostratal grammar with fixed word order will have to use different structures even for such obviously related cases as assertive and interrogative sentences, as in the latter the subject-verb order is reversed.

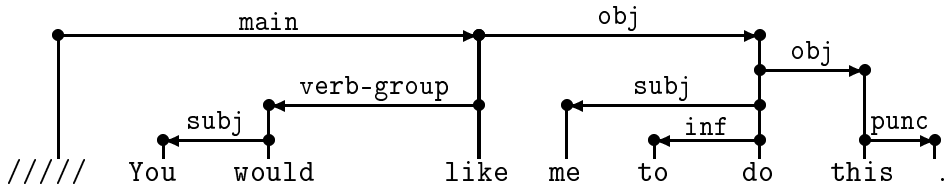
A monostratal grammar with free word order (e.g. a monostratal immediate dominance/linear precedence grammar) can already be much more functional, but it fails to be functional in unbounded dependencies, for which true non-projectivity or several levels of representation are needed. Most phrase structure grammars exploit the second alternative, so that they remain projective both at surface and deep level. The two levels are mapped to each other by movement, in which the surface and deep position of a word are co-indexed (Haegemann L. 1994). Co-indexing or movement is typically not available in DG, since there are no empty categories and no non-lexical nodes. Only by extending the DG formalism with non-lexical nodes it is possible to get a co-indexing version of DG (Lombardo V. & Lesmo L. 1998). (Neuhaus P. & Bröker N. 1997) state that unbounded dependencies can only be described in dependency theory by using non-projectivity. The only other alternative is to allow sentences with unbounded dependencies to receive analyses that are widely different from their functionally related counterparts, as we will see with LG's linkages.

Long-distance dependencies are an example of non-projectivity. The following output (6) of (Järvinen T. & Tapanainen P. 1997)'s functional *Dependency Parser for English* is a non-projective example of fronting, and (7) is its assertive counterpart[3]:

(6) *What would you like me to do?*



(7) *You would like me to do this.*



The following sections will compare two alternatives: either giving up projectivity and assigning closely related structures to functionally related sentences, or keeping projectivity and giving up functionalism, i.e. assign completely different structures to functionally related sentences.

[3]This is the actual output of their system; some dependency labels are thus different from our notation (main=SENT, obj=Obj, etc.). Note that while other linguists would take the auxiliary instead of the main verb as the surface-syntactic head (see 4.5), the analysis remains non-projective either way.

3.2.1. Giving up projectivity, keeping functionalism

While the DG characteristics of non-projectivity and free word order may be seen as a disadvantage, they also turn out to be one of the linguistically appealing strengths of DG: functionally related structures, most of them involving marked word order, some of them involving non-projectivity, can all be assigned closely related analyses:

- The same monostratal analysis can be assigned to sentences with unbounded dependencies as to their counterparts, as in (6) and (7).
- Inflectional languages such as German, Latin, Russian or Finnish (to name only a few) have a relatively free word order. In a non-projective grammar, all positional variants can be assigned the same structure (Bunt H. & van Horck A. 1996).
- Verbal particles can break up the verbal unity (e.g. *He called her up*).
- Active and passive sentences and constructions with or without dative shift can be assigned the same structure in a functional DG.
- In a non-projective grammar it is possible to add deep-syntactic long-distance dependencies, as we will see later.

In such a functional version of DG, a parser directly returns a deep-syntactic structure of the sentence which looks much like the f-structure in Lexical-Functional Grammar (LFG) (Bröker N. 1998b). For example, a simplified f-structure of (6) could be (8). Such a functional structure is an ideal candidate for an interface to semantics (Dalrymple M. *et al.* 1995:275).

(8)
$$\begin{bmatrix} SUBJ & [PRED\,'you'] & \\ OBJ & \begin{bmatrix} SUBJ & [PRED\,'me'] \\ OBJ & [PRED\,'what'] \\ PRED & \left['do\ \langle(\uparrow SUBJ)\,(\uparrow OBJ)\rangle'\right] \end{bmatrix} \\ PRED & 'like\ \langle(\uparrow SUBJ)\,(\uparrow OBJ)\rangle' \\ MOOD & COND \end{bmatrix}$$

While functional analyses are linguistically appealing, non-projectivity is computationally costly. In their discussion of non-projective DGs, (Neuhaus P. & Bröker N. 1997:337) prove that recognition (and thus, parsing) of linguistically adequate dependency grammars is NP-complete. In practical non-projective DG parsing, much of the parsing can be done by a context-free backbone (Bröker N. 1998a), but NP-completeness remains a serious threat in the background.[4]

[4]See (Kahane S. *et al.* 1998) for a DG which uses lifting rules, a device akin to GB move-alpha or the HPSG Slash feature. Because lifting rules correspond to a small well-defined amount of non-projectivity, such DGs are parsable in polynomial time.

3.2.2. Keeping projectivity, giving up functionalism

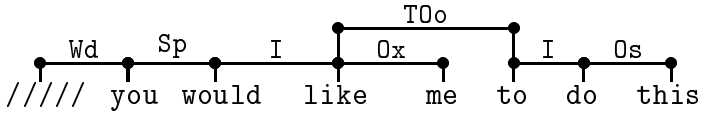
If we opt to keep projectivity, we can rely on a large number of established parsing algorithms with polynomial parsing times. In the case of Link Grammar, the parsing complexity is $O(n^3)$ (Sleator D. D. & Temperley D. 1993). But projective dependency parsers cannot deal with unbounded dependencies in an elegant way. Link Grammar is only able to parse such sentences because it gives up some standard dependency assumptions and because it employs additional link types, which are linguistically difficult to account for. We shall see in the following how Link Grammar remains projective at the expense of functionalism.

3.3. Link Grammar as a dependency grammar

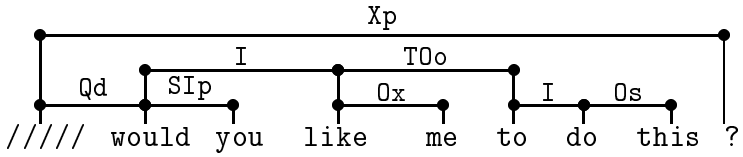
The syntactic structures which Link Grammar (LG) yields are called linkages. Like a DG, LG only knows lexical nodes and is strongly lexicalist. Valency is also the central idea in LG. It is hardly surprising, therefore, that the creators of LG themselves suggest that there is a very close relationship between dependency systems and Link Grammar (Sleator D. D. & Temperley D. 1993:12).

Linkages are not very intuitive at first sight. An LG link label consists of upper case letters indicating the major link type and lower case letters to indicate subtypes. An `Os` link, e.g., stands for an object link in which the object is singular:

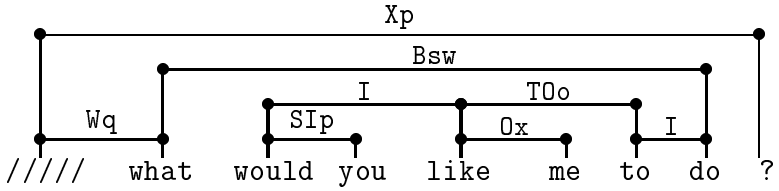
(9) *You would like me to do this*



(10) *Would you like me to do this?*



(11) *What would you like me to do?*



Sentences (9), (10), and (11) illustrate that the verb-subject link has different link labels for unmarked and reversed word order. The `S` link indicates that the subject appears to the left, while an `SI` link type is used for reversed order, i.e. if the subject appears on the right of the verb. These two link labels

need to be explicitly mapped to each other if we want to build up a functional structure.

In (9) (like in all assertive sentences), LG takes the subject to be the head of the sentence, a clear deviation from standard DG assumptions. The root link `Wd` directly connects the root to the subject, which is non-standard in dependency theory and which makes it non-trivial to find the head of the sentence. If the subject was still taken to be the main head of sentences (10) and (11), they would need a non-projective analysis. LG can only remain projective by taking the frontmost element of a sentence to be its top head, which is only sometimes the verbal head. For the generation of ExtrAns' minimal logical forms, the additional link types used by LG need to be explicitly mapped to their functional correspondences. In (11), for instance, `Bsw` needs to be mapped to the functional `Os` (object) we find in (9) and (10).

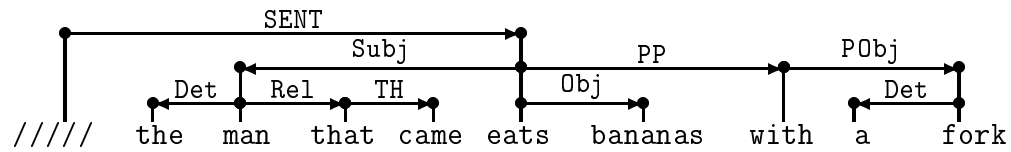
The following is an overview of the differences between linkages and DG structures. Some of these differences can be seen by comparing the dependency structures (6) and (7) to the corresponding linkages (9) and (11):

- Projectivity: LG is projective and cannot directly analyse non-projective structures.
- Links are undirected: Unlike dependencies, LG links do not state which participant in a link is head and which is dependent. However, as we shall see below, this problem can be overcome.
- Word order: LG rules define word order, i.e. whether a word links to another word on the left or on the right. Fixed word order seriously restricts the ability of a monostratal grammar to be functional.
- Top-head word: LG takes the frontmost element to be the top head of the sentence rather than the main verb. This allows LG to remain projective.
- Some linkages contain cycles: LG is enriched with deep-syntactic links for e.g. relative clauses, as we shall see now.

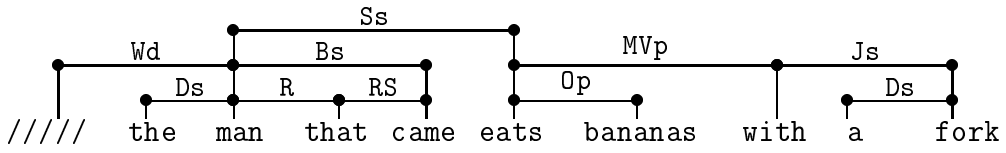
4. LOGICAL FORM GENERATION IN EXTRANS

ExtrAns produces minimal logical forms (MLFs) of sentences from linkages returned by LG. As we have seen above, these linkages differ from the most common types of dependency structures in several respects. These differences make linkages more difficult to handle. Let us compare the dependency structure (5), repeated as (12), with the linkage (13):

(12) *the man that came eats bananas with a fork*



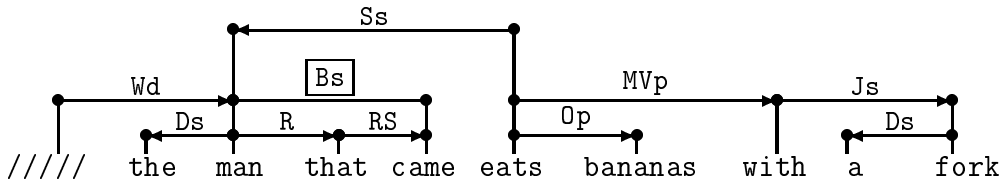
(13) *the man that came eats bananas with a fork*



There are three obvious differences that we have discussed already. First, some of the link labels in the linkage are not linguistically intuitive. Second, the link `Wd` in (13) connects the root with the subject *man*, while the corresponding dependency (`SENT`) connects the root with the verb *eats*. These two differences complicate the MLF generation. Third, links lack direction of dependency. This information, however, can be recovered. Because of its fixed word order, LG uses different link types for head-right links and head-left links of the same kind. The direction of the dependency depends mostly on the link type and therefore a simple table suffices in most cases (there are exceptions, such as the link `B`, as we shall see in Section 4.3).

We can also see that the links `B`, `R`, and `RS` form a cycle. This is the result of adding deep-syntactic links such as `B`. To simplify the syntactic tests in the anaphora resolution algorithm, only surface-syntactic links are extended with the direction of the dependency. The `B` link should therefore *not* be extended:

(14) *the man that came eats bananas with a fork*



A directed linkage such as (14) is part of the input of ExtrAns' MLF generator. The linkages in the following sections will be displayed in this format by default. Bear in mind that the directed linkage (14) is still different from the dependency structure (12), though. Converting (14) into (12) and deriving the MLF from it would be at least as time-consuming and complex a task as creating the MLFs directly from (14). We have chosen the latter method for reasons of simplicity and in order to keep processing times minimal. In the remaining sections we will focus on specific types of sentence structures, and how MLFs can be constructed from the corresponding directed linkages without having to convert the linkages into more standard dependency structures.

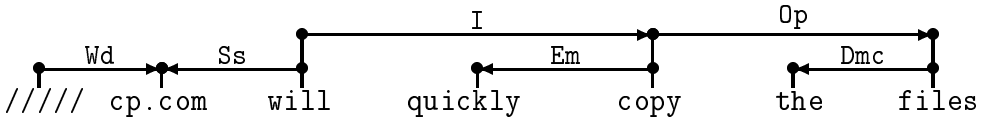
Unless otherwise stated in particular examples, all the directed linkages and MLFs shown from now on are the direct output of ExtrAns, slightly edited to improve readability. As a consequence, the MLFs use Prolog syntax, and the variable names and indices do not directly correspond to those in the handmade examples earlier in this paper.

4.1. Overall sentence structure

The first problem that appears when trying to analyse a sentence or clause is that there are many different structures that must be treated in their own particular way. Coordination aside (Section 4.2), there are conditionals and other complex sentences that complicate the processing.

Given that the directed linkage does not consider the verb as the head of the sentence, we need to find the verb by exploring the sentence. An additional complexity is that the verb may be composed of a main verb and one or more auxiliary verbs. In the search of these verbs we may traverse several links, as in (15):

(15) *cp will quickly copy the files*



We need to traverse `Wd` and `Ss` to find the auxiliary verb *will*, plus `I` to find the main verb *copy.* Once we have found the main and auxiliary verbs, the algorithm can basically follow the subject dependency to find the subject, and process the verb phrase. The task is not so easy, though. The general algorithm is:

1. Starting from the leftmost auxiliary verb (or the main verb if there is no auxiliary verb), find the head of the subject (*cp.com*) and build its logical form `object(cp,oa1,x1), object(command,oa2,x1).`

2. Build the logical forms `object(file,oa3,x6)` of the objects (only one object in (15)).

3. Create an entity for the main eventuality, `e4`.

4. Build the logical forms of other modifiers `prop(quickly,p3,e4).`

5. Add the logical form of the main event. The final MLF becomes (new information in boxes):
   `holds(e4)`, `object(cp,oa1,x1), object(command,oa2,x1),`
   `evt(copy,e4,[x1,x6])`, `object(file,oa3,x6),`
   `prop(quickly,p3,e4).`

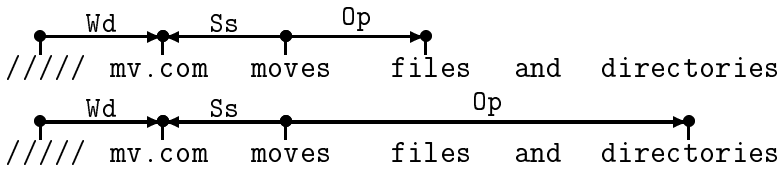
This procedure becomes extremely complicated when one considers the various idiosyncrasies of the sentence structures and the way they are handled by LG. We will explore some of these idiosyncrasies in the rest of this section.

4.2. Coordination

Coordination generally poses a problem for a dependency-based system. Even if one chooses one of the conjuncts of *files and directories* as the head of the whole coordination on syntactic grounds (Mel'čuk I. 1988), this does not help in the task of computing the MLF because the resulting structure would become asymmetrical, making one of the conjuncts depend on the other. Strictly speaking, the connective *and* cannot serve as the head because, among other things, the verbs in a sentence like *the user compiles and executes the program* have the same dependent subject and object.

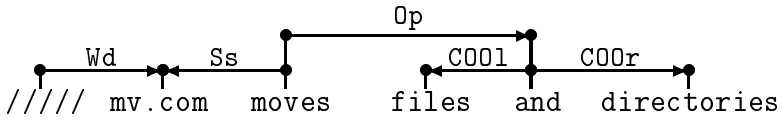
LG lists all the possible combinations that result from distributing the components of the coordination:

(16) *mv moves files and directories*



However, this approach can create a combinatorial explosion, and it fails to account for those cases where collectivity is implied, like in the sentence *John and Mary met*. An additional problem is that we need to recover the connective *and*. A better approach is to group the components of the coordination:

(17) *mv moves files and directories*



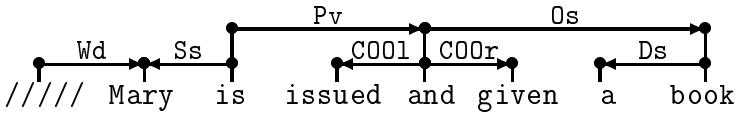
holds(e2), object(mv,oa1,x1), object(command,oa2,x1),
evt(move,e2,[x1, $\boxed{x4}$ *]),* $\boxed{x3 \sqsubseteq x4}$, $\boxed{x5 \sqsubseteq x4}$, $\boxed{object(file,oa3,x3)}$,
$\boxed{object(directory,oa4,x5)}$

In (17), the components are grouped together (this has been done by slightly modifying LG's parser), making the connective *and* dominate the other components of the coordination. The resulting MLF introduces a lattice part-of operator $\sqsubseteq$ (Landman F. 1991). The MLF of the NP can be computed easily by following the direction of dependency:

- The word *and* introduces an entity `x4` that represents the coordination.
- The word *files* and the dependency with *and* introduce:
  `x3`$\sqsubseteq$`x4, object(file,oa3,x3).`
- The word *directories* and the dependency with *and* introduce:
  `x5`$\sqsubseteq$`x4, object(directory,oa4,x5).`

There are different types of coordinated elements (nouns, verbs, NPs, adverbs, adjectives, etc). All of these different types of coordination must be treated in a different way in every case. Some of them may be quite complex, like the case of a passive with a coordination of ditransitive verbs:

(18) *Mary is issued and given a book*

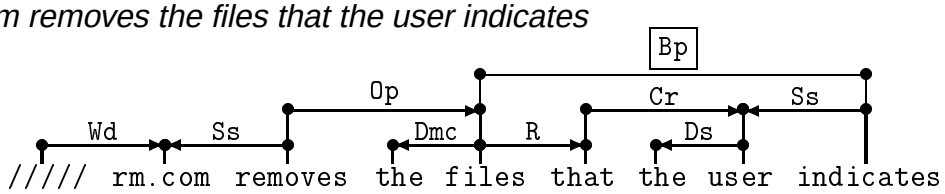


holds(e4), object('Mary'[5],oa1,x1), e3⊑e4,
evt(issue,e3,[a3,x7,x1]), e5⊑e4, evt(give,e5,[a3,x7,x1]),
object(book,oa2,x7)

4.3. Embedded clauses

An embedded clause adds complexity to the sentence, introducing recursivity. A further complexity is that, on the deep-syntactic level, the embedded clause typically has dependencies that cross the boundary of the clause itself. These dependencies are always backward dependencies (link type B and its subtypes). An example is the relative clause:

(19) *rm removes the files that the user indicates*



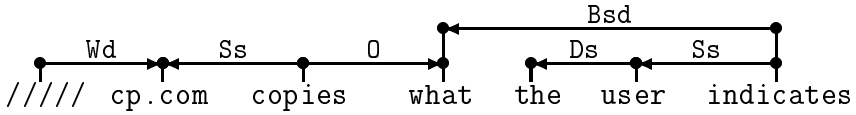
holds(e2), object(rm,oa1,x1), object(command,oa2,x1),
evt(remove,e2,[x1,x4]), object(file,oa3,x4),
object(user,oa4,x7), evt(indicate,e8,[x7,x4])

The link type B (with any subtype, such as Bp above) does not obey the syntactic structure of the sentence, and its introduction may create a cycle in the linkage, as we can see above. Still, this link is important, since it leads to the missing constituent in the main sentence. The presence of such a link simplifies the construction of the MLF since we only need to follow the link (always a B link) to find the constituent that is missing in the relative clause. In fact, the B link is a functional link that encodes a deep-syntactic dependency — in this case, a predicate-argument relation. This is the only purely deep-syntactic link that LG introduces, although, as we will see later, deep-syntactic links that risk violating projectivity are avoided by LG.

In other embedded clauses, the B link is not an extra link but the only connection with the sentence. This is the case with embedded wh-clauses:

[5]The quotes in 'Mary' are a result of Prolog notation, to distinguish the word from a Prolog variable.

(20) *cp copies what the user indicates*



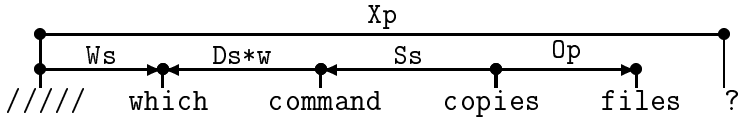
holds(e2), object(cp,oa1,x1), object(command,oa2,x1),
evt(copy,e2,[x1,x3]), object(user,oa4,x5),
evt(indicate,e6,[x5,x3])

This presents the problem that, on the surface-syntactic level, the verb of the clause is dependent of the wh-word, but on the deep-syntactic level, it is the wh-word that is dependent of the clause — this is the option shown in (20). It would be more convenient if the linkages included a new link parallel to `B` but directed the opposite way which shows the surface-syntactic dependency. In that fashion, the treatment of `B` would be more uniform. As it is now, ExtrAns must use `B` for both senses.

4.4. Questions

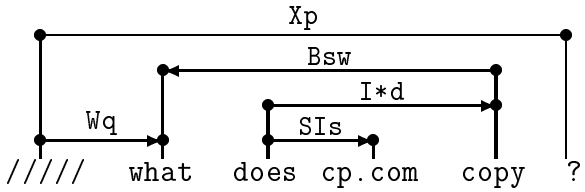
Interrogative sentences present the problem that some of the components appear in reversed order or that they are in a position different from the normal position in a declarative sentence. This may lead to non-projectivity, as we have seen in sections 3.2 and 3.3. In LG, this problem is solved by resorting to different link labels:[6]

(21) *which command copies files?*



object(command,OA1,X2), evt(copy,E3,[X2,X4]),
object(file,OA2,X4)

(22) *what does cp copy?*



object(cp,OA2,X3), object(command,OA3,X3),
evt(copy,E4,[X3,X1])

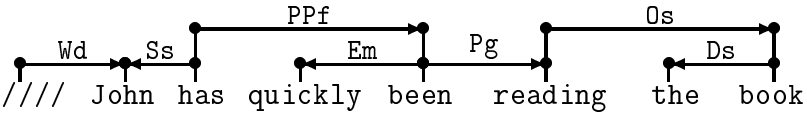
As a result, all the different types of interrogative sentences must be treated as particular cases in the several stages of the logical form generation, especially for finding the sentence head, the subject, and the objects.

[6]Since these examples correspond to questions, the variables are represented as Prolog variables and there is no `holds` predicate. The MLF of a question eventually converts into a Prolog query (Section 2.6).

4.5. The verbal group

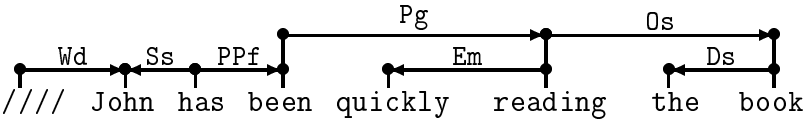
LG uses several links to connect the words that make up the verbal group. The subject attaches to the first of the words, and the objects and modifiers attach to any of the words in the verbal group. Consider the following examples:

(23) *John has quickly been reading the book*




```
holds(e5), object('John',oa1,x1), evt(read,e5,[x1,x7]),
object(book,oa2,x7), prop(quickly,p2,e5)
```

(24) *John has been quickly reading the book*



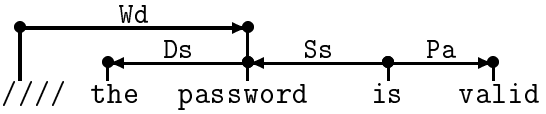

```
... MLF as in (23) ...
```

The attachment of the subject in (23) and (24) is to the auxiliary verb *has*, whereas the object attaches to the main verb *reading*, and the adverb *quickly* attaches to the verbal group member on its right. Because of the different attachments to different parts of the verbal group, one would say that there is no real head in the verbal group. For processing reasons, we decided to give a direction of dependency from the auxiliary on the left (the one to which the subject attaches) to the main verb. We have done this so that the anaphora resolution module encounters fewer difficulties in its syntactic tests. Still, for computing the logical form, we effectively consider all the components of the whole verbal group as possible attachment places for the dependents. This treatment is therefore akin to Tesnière's consideration of the whole verbal group as a single nucleus (Section 3.1.1).

4.6. Predicative adjectives

Predicative adjectives are linked to the copular verb by the link `Pa`:

(25) *the password is valid*



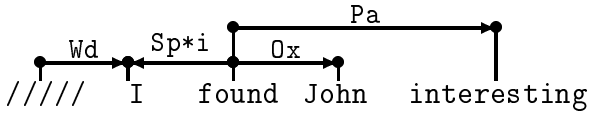

```
holds(p4), object(password,oa1,x2), prop(valid,p4,x2)
```

Here, the adjective generates the main predicate of the MLF, and as such it is asserted as a property of the subject, and this property holds. This is done by treating the link `Pa` as one more of the links that bind the components of

the verbal group. Predicative adjectives are therefore integrated into the verbal group, like in Tesnière's approach.

There is another construction containing a predicative adjective, this time without using a copular verb, where the predication refers to the object of the verb. Still, the logical form must express the correct entity that is predicated over. An example is:

(26) *I found John interesting*



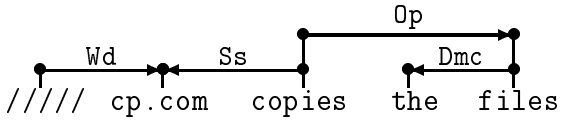

```
holds(e2), object('I',oa1,x1), evt(find,e2,[x1,p4]),
prop(interesting,p4,x3), object('John',oa2,x3)
```

As we can see, the linkage does not show a dependency between *John* and the adjective *interesting*. It is remarkable that LG introduces a deep-syntactic link in some embedded clauses (Section 4.3), but not in the example above. The MLF generator must therefore find the argument *John* by exploring the object of the verb *found*.

4.7. Passive and ditransitivity

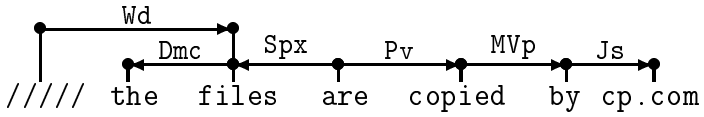
The passive is a typical case where the surface structure does not match the functional structure. For example, the two following sentences create the same MLF:

(27) *cp copies the files*




```
holds(e2), object(cp,oa1,x1), object(command,oa2,x1),
evt(copy,e2,[x1,x4]), object(file,oa3,x4)
```

(28) *the files are copied by cp*



. . . MLF as in (27) . . .

If there is a PP headed by *by*, then the PP object fills the first verb argument. If there is no such PP, the first verb argument remains anonymous:

(29) *the files are copied*

```
holds(e4), object(file,oa1,x2), evt(copy,e4,[a2,x2])
```

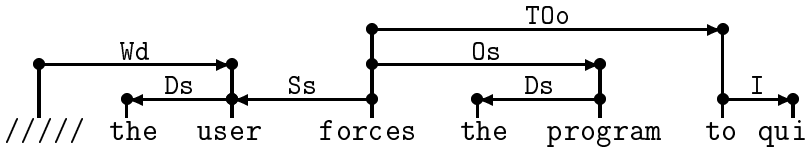
We can see in (27) and (28) that the directed linkages keep the surface structure. We must therefore consider the passive as a particular case where the links `S`, `O`, and `MV` must be dealt with differently from the active form.

Ditransitivity is another typical example where different surface structures result in the same logical form. As in the passive, LG generates completely different linkages. ExtrAns must consider these different possibilities when building the MLF.

4.8. Non-finite clauses

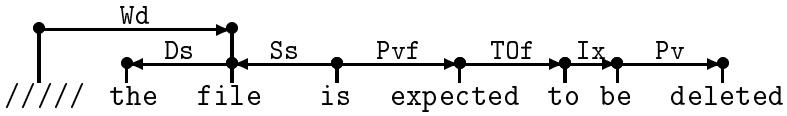
Non-finite clauses present the problem that one needs to find the correct arguments of the verb in the embedded clause, since the deep-syntactic links are not expressed by LG for these cases. For example, here are two control and raising sentences with their MLFs:

(30) *the user forces the program to quit*



holds(e3), object(user,oa1,x2), evt(force,e3,[x2,e7]), evt(quit,e7,[x5]), object(program,oa2,x5)

(31) *the file is expected to be deleted*

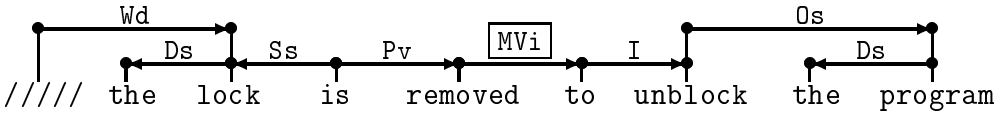


holds(e4), object(file,oa1,x2), evt(expect,e4,[a4,e7]), evt(delete,e7,[a2,x2])

Generally, the subject of the main clause is the subject of the non-finite clause, except when the subject of the embedded clause is determined by other means. For example, (30) specifies that the subject of the non-finite clause is *the program*. However, the resulting directed linkage may lead to misunderstandings, since *the program* is treated as an object of the main verb *forces*. The link `TOo` (note the subtype `o`) specifies that the object of the main verb is actually the functional subject of the non-finite verb *quit*. LG tries to indirectly express the deep-syntactic link without risking breaking the projectivity principle. The logical form generator must therefore follow the links back until the subject is found, or store the potential subjects in temporary variables. We found it more convenient to do the latter, so that we do not have to trace back the links. Note that the logical form needs to place the agent, not the subject, in the first argument of the verb. Therefore, all the appropriate rules for the passive must apply to both the main clause and the non-finite clause (31).

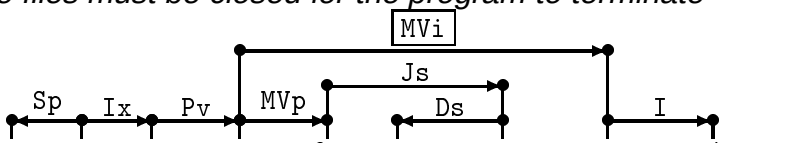
The following is a list of sentences with intentional PPs. Intentional PPs can be recognised on the basis of the syntactic structure (subtype `i` in `MVi` in these examples). Consequently, their MLFs contain a predicate `purpose`:

(32) *the lock is removed to unblock the program*



holds(e4), object(lock,oa1,x2), evt(remove,e4,[a3,x2]),
purpose(e4,e6), evt(unblock,e6,[a3,x8]),
object(program,oa2,x8)

(33) *the files must be closed for the program to terminate*



holds(e5), object(file,oa1,x2), evt(close,e5,[a3,x2]),
purpose(e5,e10), evt(terminate,e10,[x8]),
object(program,oa2,x8)

Now, it is the *agent* of the main clause which becomes the subject of the non-finite clause, as we can see in the case of a passive such as (32), unless, of course, the subject is specified (33). Note again that one needs to find the subject in (33) outside the embedded clause. The reason is that the linkage is kept at a surface level, and therefore the PP *for the program* is left attached to the main clause, very much like in the case of the agent in a passive.

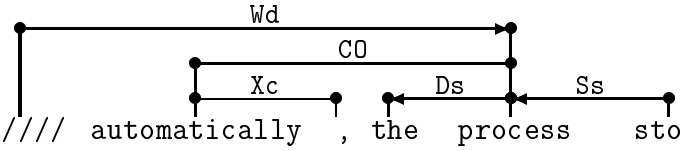
4.9. Sentence openers

Sentence openers present one more case where LG fails to provide the correct dependency. Sentence openers attach to the head of the sentence, but LG typically designates the subject as the head of the sentence. As a result, the openers attach to the subject, not to the main verb. The current version of ExtrAns does not correct the attachment in the directed linkage. Instead, the event introduced by the main verb is specified in the parameter list of the routine that processes the openers.

The link used to mark a sentence opener (CO) does not show a direction of dependency. This is so because, given that the link connects the wrong words, other modules that rely on the direction of the dependency (such as the anaphora resolution module) can become confused.

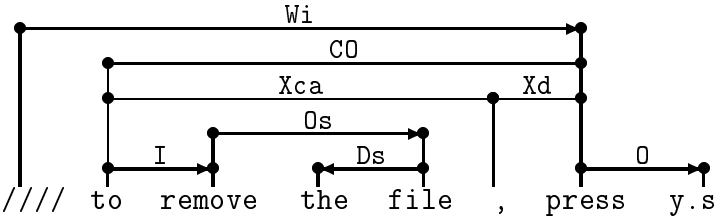
An additional problem is that there is one link type only, namely CO, to mark a sentence opener, but there are many types of openers. The various types of openers can sometimes be distinguished by means of the link subtype (the subindices, if there are any) but in some occasions the link subtype is not known or it is not specific enough, and we need to look at the structure of the opener itself to find out what type of opener it is. Some examples where no link subtypes are provided follow:

(34) *automatically, the process stops*



holds(e5), object(process,oa1,x4), evt(stop,e5,[x4]), prop(automatically,p1,e5)

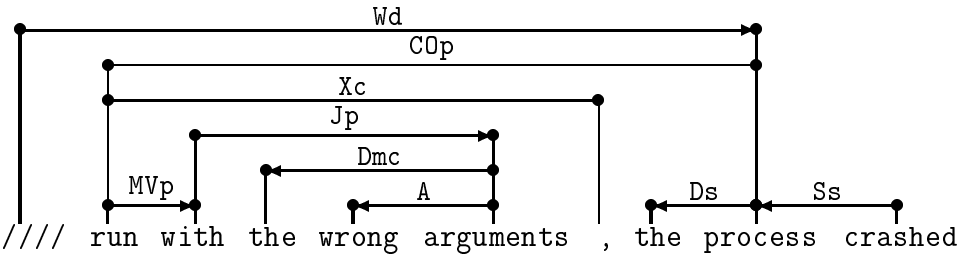
(35) *to remove the file, press y*



holds(e6), object(hearer,oa1,h), evt(press,e6,[h,x7]), object(y,oa2,x7), object(symbol,oa3,x7), purpose(e6,e2), evt(remove,e2,[h,x4]), object(file,oa4,x4)

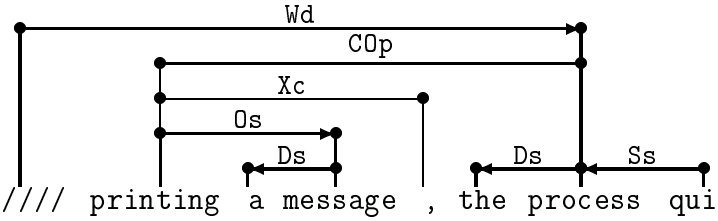
The following examples show the use of non-finite clauses as sentence openers:

(36) *run with the wrong arguments, the process crashed*



holds(e9), object(process,oa1,x8), evt(crash,e9,[x8]), evt(run,e1,[a3,x8]), with(e1,x5), object(argument,oa2,x5), prop(wrong,p4,x5)

(37) *printing a message, the process quits*



holds(e7), object(process,oa1,x6), evt(quit,e7,[x6]), evt(print,e1,[x6,x3]), object(message,oa2,x3)

As in any non-finite clause, one must find the arguments of the verb. The general rule is, “the subject of the main clause is also the subject of the

opener". One must also keep in mind that a past participle generally — though not always (Quirk R. *et al.* 1972) — marks a passive (36), and that a present participle marks an active (37).

CONCLUSION

The current implementation of ExtrAns converts LG's syntactic analysis of unedited document sentences and user queries into the MLFs needed for the AE task. LG being a dependency-based system, we have shown that it is possible to build a practical system that relies on a dependency structure to create a logical form. The structure given by LG, however, fails to be functional (as DGs could be), and some link attachments and labels are difficult to interpret. This is the consequence of taking full advantage of fast projective parsers, and considering only the surface-syntactic structure as opposed to a deeper functional structure. As a result, the translation of LG structures into logical forms becomes more complex than it would be for purely functional structures, but remains feasible.

REFERENCES

BAUMGÄRTNER K. (1970) : "Konstituenz und Dependenz. Zur Integration der beiden grammatischen Prinzipien.", *in Vorschläge für eine strukturale Grammatik des Deutschen,* H. Steger (ed.), Darmstadt, Wissenschaftliche Buchgesellschaft.

BRILL Eric & RESNIK Philip (1994) : "A rule-based approach to prepositional phrase attachment disambiguation", *in Proc. COLING '94*, Kyoto, pp. 998-1004.

BRÖKER Norbert (1998a) : "How to define a context-free backbone for DGs: Implementing a DG in the LFG formalism", *in Proc. of the COLING-ACL'98 workshop "Processing of Dependency-based Grammars",* S. Kahane & A. Polguère (eds.), pp. 29-38, Montreal.

BRÖKER Norbert (1998b) : "A projection architecture for dependency grammars and how it compares to LFG", *in Proc. LFG 98,* M. Butt & T. H. King (eds.), University of Queensland, Brisbane.

BUNT Harry & VAN HORCK Arthur (eds.) (1996) : *Discontinuous Constituency*, Berlin, Mouton de Gruyer.

COOK Vivian & NEWSON Mark (1996) : *Chomsky's Universal Grammar*, Oxford, Blackwell, 2nd édition.

COVINGTON Michael A. (1990) : "Parsing discontinuous constituents in dependency grammar", *Computational Linguistics*, vol. 16, pp. 234-237.

COVINGTON Michael A. (1992) : *GB Theory as Dependency Grammar*, Rapport technique n^{o} AI-1992-03, Athens, University of Georgia.

COVINGTON Michael A. (1994) : *An Empirically Motivated Reinterpretation of Dependency Grammar*, Rapport technique n^{o} AI-1994-01, The University of Georgia.

DALRYMPLE Mary, KAPLAN Ronald M., MAXWELL John T. & ZAENEN Annie (eds.) (1995) : *Formal Issues in Lexical-Functional Grammar*, Stanford, CSLI.

DAVIDSON Donald (1967) : “The logical form of action sentences”, *in The Logic of Decision and Action,* N. Rescher (ed.), Univ. of Pittsburgh Press, pp. 81-120.

DOWNEY Laura L. & TICE Dawn M. (1999) : “A usability case study using TREC and ZPRISE”, *Information Processing and Management,*, vol. 35, n° 5, pp. 589-603.

FARRINGTON Gordon (1996) : “AECMA simplified English. An overview of the international aerospace maintenance language”, *in Proc. CLAW96*, Katholieke Universiteit Leuven, Leuven, pp. 1-21.

FELLBAUM Christiane (ed.) (1998) : *WordNet: an electronic lexical database*, Cambrige, MIT Press, *Language, Speech, and Communication*.

GAIFMAN H. (1965) : “Dependency systems and phrase-structure systems”, *Information and Control*, vol. 8, pp. 304-337.

HAEGEMANN Lilian (1994) : *Introduction to Government & Binding*, Oxford, Basil Blackwell.

HAYS David (1964) : “Dependency theory: A formalism and some observations”, *Language*, vol. 40, pp. 511-525.

HELBIG Gerhard (1992) : *Probleme der Valenz- und Kasustheorie. Konzepte der Sprach- und Literaturwissenschaft*, Tübingen, Niemeyer.

HELLWIG Peter (1986) : “Dependency unification grammar”, *in Proc. COLING*, University of Bonn, Bonn, pp. 195-198.

HOBBS Jerry R. (1985) : “Ontological promiscuity”, *in Proc. ACL’85*, University of Chicago, Association for Computational Linguistics, pp. 61-69.

HOBBS Jerry R. (1996) : “Monotone decreasing quantifiers in a scope-free logical form”, *in Semantic Ambiguity and Underspecification,* K. van Deemter & S. Peters (eds.), Stanford, CSLI Publications, chap. 3, pp. 55-76.

HUDSON Richard (1996) : “Word grammar”, *in Concise Encyclopedia of Syntactic Theories,* K. Brown & J. Miller (eds.), Oxford, Elsevier, pp. 368-372.

HUMPHREYS Kevin, GAIZAUSKAS Rob, CUNNINGHAM Hamish & AZZAM Saliha (1996) : *GATE: VIE Technical Specifications*, Rapport technique, University of Sheffield, ILASH, Included in the documentation of GATE 1.0.0.

JÄRVINEN Timo & TAPANAINEN Pasi (1997) : *A Dependency Parser for English*, Rapport technique n° TR-1, Helsinki, Department of Linguistics, University of Helsinki.

KAHANE Sylvain, NASR Alexis & RAMBOW Owen (1998) : “Pseudo-projectivity: A polynomially parsable non-projective dependency grammar”, *in Proc. COLING-ACL’98*, Université de Montréal.

LANDMAN Fred (1991) : *Structures for Semantics*, Dordrecht, Kluwer.

LAPPIN Shalom & LEASS Herbert J. (1994) : “An algorithm for pronominal anaphora resolution”, *Computational Linguistics*, vol. 20, n° 4, pp. 535-561.

LEWIS David D. & SPARCK JONES Karen (1996) : “Natural language processing for information retrieval”, *Communications of the ACM*, vol. 39, n° 1, pp. 92-101.

LOMBARDO Vincenzo & LESMO Leonardo (1998) : “Unit coordination and gapping in dependency theory”, *in Processing of Dependency-Based Grammars; proceedings of the workshop. COLING-ACL,* S. Kahane & A. Polguère (eds.), Montreal.

MARCUS M., SANTORINI B. & MARCINKIEWICZ M. (1993) : “Building a large annotated corpus of English: the Penn Treebank”, *Computational Linguistics*, vol. 19, n° 2, pp. 313-330.

MCCORD Michael, BERNTH Arendse, LAPPIN Shalom & ZADROZNY Wlodek (1992) : "Natural language processing within a slot grammar framework", *International Journal on Artificial Intelligence Tools*, vol. 1, n° 2, pp. 229-277.

MEL'ČUK Igor (1988) : *Dependency Syntax: Theory and Practice*, State University of New York Press.

MOLLÁ Diego & HESS Michael (2000) : "Dealing with ambiguities in an answer extraction system", *in Workshop on Representation and Treatment of Syntactic Ambiguity in Natural Language Processing*, Paris, pp. 21-24.

MOLLÁ Diego, BERRI Jawad & HESS Michael (1998) : "A real world implementation of answer extraction", *in Proc. of the 9th International Conference and Workshop on Database and Expert Systems. Workshop "Natural Language and Information Systems" (NLIS'98)*, Vienna, pp. 143-148.

MONTAGUE Richard (1973) : "The proper treatment of quantification in ordinary English", *in Approaches to Natural Language,* K. J. J. Hintikka, J. Moravcsic & P. Suppes (eds.), Dordrecht, Reidel, pp. 221-242.

NEUHAUS P. & BRÖKER N. (1997) : "The complexity of recognition of linguistically adequate dependency grammars", *in Proc. ACL/EACL'97*, Madrid.

O'CONNOR John (1975) : "Retrieval of answer sentences and answer-figures from papers by text searching", *Information Processing & Management*, vol. 11, n° 5/7, pp. 155-164.

PARSONS Terence (1985) : "Underlying events in the logical analysis of English", *in Actions and Events: Perspectives on the philosophy of Donald Davidson,* E. Lepore & B. P. McLaughlin (eds.), Oxford, Blackwell, pp. 235-267.

POLLARD Carl (1994) : *Head-Driven Phrase Structure Grammar*, Chicago, Chicago University Press.

QUIRK Randolph, GREENBAUM Sidney, LEECH Geoffrey & SVARTVIK Jan (1972) : *A Grammar of Contemporary English*, Harlow, Longman.

ROMACKER Martin & HAHN Udo (1999) : "On the semantic linkage of dependency relations and conceptual relations", *in Proc. NLDB'99,* G. Friedl & H. C. Mayr (eds.), Klagenfurt, pp. 77-90.

SCHNEIDER Gerold (1998) : A Linguistic Comparison of Constituency, Dependency and Link Grammar, Master's thesis, University of Zurich, Unpublished.

SGALL Petr, HAJIČOVÁ Eva & PANEVOVÁ Jarmilla (1986) : *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*, Dordrecht, Reidel.

SLEATOR Daniel D. & TEMPERLEY Davy (1993) : "Parsing English with a link grammar", *in Proc. Third International Workshop on Parsing Technologies*, pp. 277-292.

SUTCLIFFE Richard F. E., KOCH Heinz-Detlev & MCELLIGOTT Annette (eds.) (1996) : *Industrial Parsing of Software Manuals*, Amsterdam, Rodopi.

TESNIÈRE Lucien (1959) : *Eléments de Syntaxe Structurale*, Paris, Klincksieck.

TREC-8 (1999), "Call for participation Text REtrieval Conference 1999 (TREC-8)", `http://trec.nist.gov/cfp.html`.