T.A.L., vol. 41, n° 2, pp. 1–25

EXTRANS, AN ANSWER EXTRACTION SYSTEM

Diego MOLLÁ **Rolf SCHWITTER** **Michael HESS**
Rachel FOURNIER*

Résumé - Abstract

Les systèmes d'Extraction de Réponses (ER) récupèrent dans des documents des expressions qui répondent directement à des questions du langage courant. L'ER pour des manuels techniques exige un haut niveau de rappel et de précision ; pourtant, ce sont de petites unités de texte qui doivent être récupérées. C'est pourquoi il est important d'effectuer une analyse linguistique détaillée. Nous présentons ici ExtrAns, un système d'ER pour des manuels Unix qui utilise une analyse syntaxique complète, une désambiguïsation partielle et un module de résolution d'anaphores pour générer les formes logiques minimales correspondant aux documents et à la requête. La procédure de recherche se sert d'un algorithme de démonstration de la requête sur la représentation en clauses de Horn des formes logiques minimales. Les ambiguïtés non résolues sont traitées à l'aide d'une mise en évidence graduelle.

Answer Extraction (AE) systems retrieve phrases in textual documents that directly answer natural language questions. AE over technical manuals requires very high recall and precision, and yet small text units must be retrieved. It is therefore important to perform linguistic analysis in detail. We present ExtrAns, an AE system over Unix manuals that uses full parsing, partial disambiguation, and anaphora resolution to generate the minimal logical forms of the documents and the query. The search procedure uses a proof algorithm of the user query over the Horn clause representation of the minimal logical forms. Remaining ambiguities in the retrieved sentences are dealt with by graded highlighting.

Mots Clefs - Keywords

Composants linguistiques en RI, extraction de réponses, formes logiques minimales.

Linguistic components in IR, answer extraction, minimal logical forms.

*Computational Linguistics Group, University of Zurich. E-mail: {molla, schwitt, hess, fournier}@ifi.unizh.ch.

INTRODUCTION

The need for systems capable of retrieving precise information from textual documents in an efficient way is becoming more obvious by the day. The fact that the known methods, such as document retrieval and information extraction, are increasingly inadequate or insufficiently powerful in the task of localising very precise information, has been recognised by the TREC programme committee with the formation of the Question Answering Track (Voorhees E. M. & Harman D. 1999). Since fully-fledged text-based question answering is still too ambitious for practical purposes, realistic compromise solutions are clearly needed. We suggest answer extraction as one such solution and introduce ExtrAns, an implementation of an answer extraction system[1].

Document Retrieval (DR) techniques are the prototypical form of information retrieval methods, to the point that textbooks about information retrieval typically cover almost exclusively DR (van Rijsbergen C. 1979; Salton G. & McGill M. J. 1983). They have the advantage of allowing arbitrary queries over very large document collections (many gigabytes in size) covering arbitrary domains. One of the disadvantages of DR is the very fact that such systems retrieve entire documents, which is unhelpful if documents are dozens, or hundreds, of pages long. Typically, DR systems (be they based on Boolean, vector space, or probabilistic principles) are keyword based, i.e. they take into consideration only the content words of documents and queries, discarding all the morphological and syntactic information including all function words. This is why these systems cannot distinguish *the command copies files* from *the command files copies* (lost ordering information), or *export from Germany to the UK* from *export to Germany from the UK* (lost function word information). True, some DR systems can use phrasal search terms (such as *computer design*), to be found as a whole in the documents, but then a number of relevant documents (such as those containing *design of computers*) will no longer be retrieved. Statistical methods are used occasionally to find inherent relations between words, such as synonymy (Deerwester S. *et al.* 1990), or to determine simple dependencies between the words in a sentence (Strzalkowski T. *et al.* 1997). Some DR systems use partial linguistic information and even ontologies to create more accurate indexing terms (Woods W. A. 1997; Strzalkowski T. *et al.* 1998). However, to the best of our knowledge, no DR system produces *full* syntactic parses of either documents or queries. In fact, the common belief holds that it does not pay off to use deep linguistic analysis in DR (Lewis D. D. & Sparck Jones K. 1996).

Information Extraction (IE) techniques are similar to DR techniques in that they, too, are suitable for processing text collections of basically unlimited size (normally, messages in a stream) covering a potentially wide range of topics. However, IE systems differ from DR systems in that they identify those messages in a stream that fall into a (usually very small) number of specific

[1]This research is funded by the Swiss National Science Foundation, project No. 12–53704.98.

topics, and extract from those a very limited amount of highly specific data. This information is placed into a frame-like database record with a fixed number of predefined role slots, with one type of frame for each type of report. Such systems typically use some kind of very shallow syntactic analysis because of run-time requirements (Appelt D. E. *et al.* 1993; Grisham R. & Sundheim B. 1996; Chinchor N. A. 1998). Clearly, the kind of information extracted by these systems is much more precise and specific than what is delivered by DR systems. On the other hand, IE systems do not allow for arbitrary questions.

Text-based *Question Answering* (QA) systems would be the ideal solution to the problem of finding precise and localised information. QA systems read texts, assimilate their content into knowledge bases, and generate answers to arbitrary questions phrased in unrestricted natural language. Two well-known examples of QA systems are Unix Consultant (UC), which performs QA over a (hand-crafted) data base of facts about Unix (Wilensky R. *et al.* 1994), and LILOG (Herzog O. & Rollinger C.-R. 1991), which performs QA over a small number of travel guide texts in German. QA systems must integrate very deep syntactic and semantic analysis of both documents and questions, search in knowledge bases, inference over textual and world knowledge, and generation of answers in natural language. Some systems (such as UC) even include a user modelling component that keeps track of previous interactions with the user and makes sure that the overall dialogue between the user and the machine sounds natural. Each of these tasks is in itself very difficult to implement with the current state of the art in technology. As a result, all past and existing fully-fledged QA systems work only over very narrow domains, for extremely small volumes of text, and with very high development costs. One may attempt to produce a reduced version of a QA system, for example, by using a very simple knowledge representation (Katz B. 1997). To our knowledge, however, it has not been possible to use such systems with real-world text in practical applications.

Answer Extraction (AE) lies between text-based QA on the one hand, and DR and IE on the other. AE systems allow questions in arbitrarily phrased, unrestricted, natural language, over a collection of texts in equally unrestricted natural language, but they merely pinpoint the exact phrases in the documents of the collection which contain the *explicit* answers to the specific question. AE systems do *not* try to perform inferences over the content of the documents (or world knowledge), and they do not generate answers either. In this respect they are much more modest than fully-fledged QA systems. It is interesting to see that the systems which participated in the QA track of the TREC-8 conference (Voorhees E. M. & Harman D. 1999) did not do QA in the classical sense but AE as defined in this paper. Other attempts to use DR techniques for AE are under way (Woods W. A. 1997). There are also attempts to produce AE over FAQs (Burke R. D. *et al.* 1997; Winiwarter W. 1999) by trying to match the user's query with an existing query in a FAQ, and returning the answer given in the FAQ.

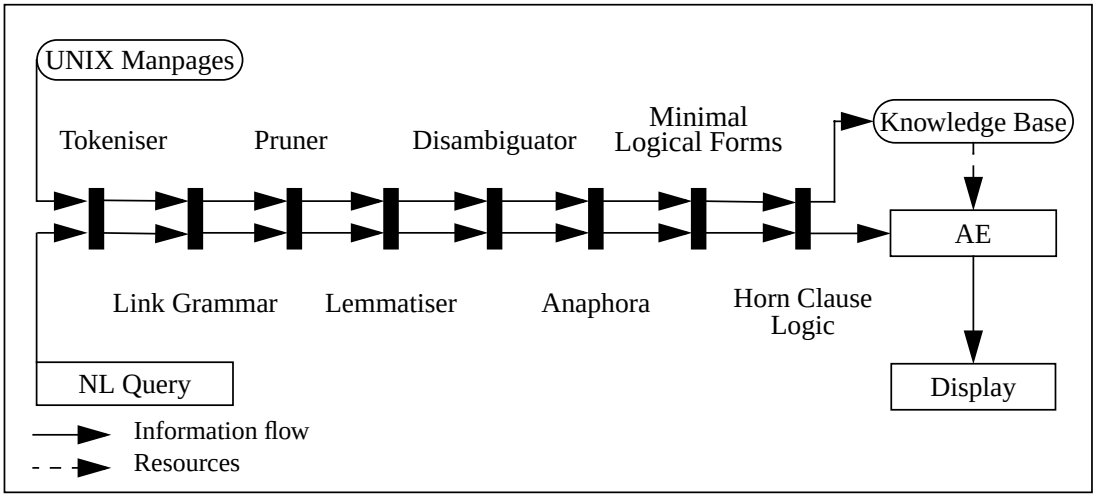


Figure 1: The architecture of ExtrAns

Examples for possible applications of AE methods are interfaces to machine-readable technical manuals or on-line help systems for complex software. In these applications very high retrieval precision on the level of individual phrases is mandatory (queries in these domains tend to be very specific), and high recall is equally vital (technical texts typically explain things only once). High recall and precision are best achieved if we determine *part of the meaning of sentences* (both of questions and texts) and locate relevant phrases in the documents on the basis of their meaning. This is computationally expensive but we can approach the final goal gradually. We can begin with a fairly simple yet useful system for technical manuals, since these are moderately sized document collections and they cover a very limited domain, and we can then refine the system in a stepwise manner to cover wider domains and larger volumes of data — for example, a recent follow-up of ExtrAns is WebExtrAns, an AE system over XML technical manuals that is using, for the time being, the maintenance manual of a commercial aircraft. Apart from extending the coverage, we may also attempt to gradually increase the depth of analysis, and ultimately we might even arrive at a fully-fledged text-based QA system. The main goal of this paper is to show that the current state of the art in NLP technologies makes it possible to implement useful AE systems over technical manuals.

1. THE EXTRANS SYSTEM

1.1. Overview

ExtrAns finds those exact phrases in a collection of technical documents that directly answer a user query. Figure 1 gives an overview of ExtrAns' general architecture. The current version of ExtrAns runs over 500 unedited Unix

manual pages (*manpages*). These highly technical documents are first pre-processed by the **tokeniser** of ExtrAns that exploits all formatting information and domain-specific typographic conventions. For the syntactic analysis of document sentences and user queries, ExtrAns uses "**Link Grammar**" (LG), which consists of a very fast parser and a grammar of English written in the spirit of dependency grammars. The LG parser outputs all the alternative dependency structures for a sentence, showing the words that are linked and the types of the links. From the output of LG, obviously wrong structures are filtered out by a **pruner** that relies on a set of hand-crafted rules for the Unix domain. Since LG does not carry out any morphological analysis, ExtrAns uses a third-party **lemmatiser** that generates the lemmas of the inflected word forms (Humphreys K. *et al.* 1996). In a next step, different forms of attachment ambiguities are resolved by a **disambiguator** trained with data from the man-pages. Following that, pronominal **anaphors** are resolved (on purely syntactic information, in contrast to the disambiguator that uses statistical knowledge). From these (partially disambiguated) dependency structures ExtrAns derives one or more **Minimal Logical Forms** (MLFs) as semantic representation for the core meaning of each sentence. MLFs have been designed to keep the balance between expressivity and processability for the AE task at hand. For processing reasons, MLFs are translated into **Horn clause logic** and asserted into the knowledge base. Unlike sentences in documents, which are processed off-line, MLFs of user queries are computed on-line and are proved by refutation over the documents. ExtrAns' AE procedure always finds all proofs for a user query and assumes that the more often a part of an extracted sentence was used for the proof the more adequate it is. Adequacy is displayed by highlighting the pertinent parts of the sentences found, both individually (see Figure 2 in Section 1.9) and in the context of the whole document. ExtrAns is particularly user-friendly not only because of this feature but also because it is very robust. It uses a keyword mechanism for unanalysable parts of sentences and a graceful fall-back strategy that relaxes the proof criteria in a stepwise manner if direct hits cannot be found.

1.2. Tokeniser

Since ExtrAns has to be able to cope with unrestricted text, it needs a very reliable tokeniser. ExtrAns' tokeniser processes the NAME, SYNOPSIS, and DESCRIPTION section of the manpages. Apart from identifying regular word forms and sentence boundaries, the tokeniser has to recognise a set of domain-specific words. Therefore, the tokeniser uses information from the SYNOPSIS section together with a set of heuristics to recognise these special words and represent them as normalised tokens. The Unix manpages are encoded in Troff format where command names are printed in boldface and arguments in italics. As a consequence a considerable amount of formatting information can be exploited and added to the representation of the tokens. For example, command names are distinguished from homographic word forms

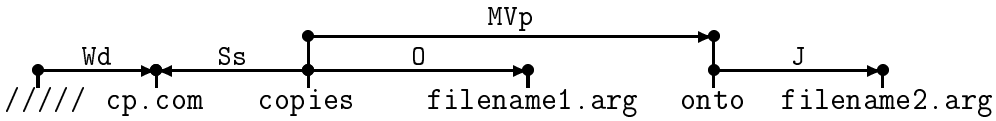
(like “eject” in ***eject*** *is used for those removable media devices that do not have a manual eject button*) and are represented by adding an unambiguous subscript to the token (e.g. ***eject*** is tagged and represented as *eject.com*). Such subscripts are also introduced for words that are used as arguments (e.g. *filename1.arg, device.arg, nickname.arg*), path names (e.g. *</usr/5bin/ls>.path, </etc/hostname.le>.path*) and command options (e.g. *<-c>.opt, <-kbd>.opt, <-3>.opt*).

The tokeniser supplies a unique sentence identifier, the tokenised data and information about the offset position of every token with respect to the input sentence and the Troff source file. This information is used later to highlight phrases in the retrieved sentences.

1.3. Link Grammar

The syntactic analysis module used by ExtrAns, Link Grammar (LG), consists of a very fast parser and a grammar/dictionary with about 60,000 word forms (Sleator D. D. & Temperley D. 1993). The coverage of the parser was tested using 2,781 sentences from the manpages and a percentage of 76% full parses was found. The remaining sentences are partially parsed by systematically ignoring words, as we will see below. LG returns dependency relations between pairs of words in a sentence by a set of labelled links called linkage. By default, the direction of the dependency is not given explicitly in the linkage. This is a serious shortcoming of LG since information about the direction of the dependency is indispensable for the anaphora resolution algorithm and for the construction of the MLFs. Therefore, the output of LG was extended in ExtrAns by adding the dependency direction to the linkage as the arrows in (1) indicate:

(1) *cp.com copies filename1.arg onto filename2.arg*



Such directed linkages are called dependency structures in this paper. In the example above, the link `Wd` connects the subject `cp.com` to the wall `/////`. The wall is a dummy word at the beginning of every sentence and has a linking requirement like any other word. The link `Ss` connects the transitive verb `copies` with the subject on the left. Thus, the verbal head is at the right hand side of the link. The transitive verb and its direct object `filename1.arg` that acts as the head of a noun phrase are connected by the link `O`. The link `MVp` connects the verb to the modifying prepositional phrase. Finally, the link `J` connects the preposition `onto` to its object `filename2.arg`.

The LG parser is able to handle unknown words by making guesses from context about syntactic categories. Nevertheless, the result is always better when the words have been categorised in advance. Therefore, we have added about 650 domain-specific words to the LG dictionary. Some words that were already classified in the default LG dictionary had to be moved to other

categories because they could be used differently in the Unix domain. An example is the verb *print* of the category *transitive verbs* that had to be moved to the category *transitive verbs that may form two-word verbs* to exclude ambiguity because it is often used as *print out* in the manpages.

In the original LG dictionary, words with multiple entries were distinguished by means of different subscripts. For ExtrAns, a subscript has been added to each single word and the subscript set was refined so that it can be used to tag the syntactic categories of the words and agreement information. To ease readability, these subscripts do not appear in the linkages, only the subscripts provided by the tokeniser are shown. Substantial changes had to be made in the grammar to deal with some specific syntactic structures like post-nominal modifiers for command names (e.g. . . . *an ls.com on such a link* . . .) or special forms of imperatives with openers (e.g. *to quit, type q*).

LG allows robust parsing by systematically ignoring words until a valid dependency structure is found. These words are represented in a special form, with no links attached to them (Grinberg D. *et al.* 1995). Such "null-linked" words are not lost, they can be used — as we will see later — by the retrieval procedure.

1.4. Disambiguator

ExtrAns' pruner uses heuristic rules to filter out all those dependency structures that are obviously wrong in an ambiguous sentence. In a subsequent stage, ExtrAns' disambiguator uses a corpus-based approach (Brill E. & Resnik P. 1994) to eliminate (some of) those ambiguities that require domain knowledge.

Brill & Resnik's original disambiguator was designed to solve prepositional attachment ambiguity of sentences with a transitive verb and a prepositional phrase (PP). The algorithm decides whether the PP should attach to the verb or to the direct object. The algorithm bases its decision on four-tuples (verb, object, preposition, and the object under the preposition) and a set of training rules that are automatically generated from a training corpus. For example, a sentence like *cp copies filename1 onto filename2* would lead to the template `n v copy file onto file`. This means that there exists a PP attachment in the sentence, and the main verb is `copy`, the head of the direct object is `file`, the preposition is `onto`, and the head of the noun phrase in the PP is `file`. There are two fields in front of these four-tuples: The first field indicates the attachment decision given by the disambiguator. It initially contains a default value `n` (attachment to the noun) that may be modified by the disambiguator. The second field is defined in the training corpus only, and it reflects the correct attachment during the training and the evaluation.

ExtrAns' requirements are different from the original disambiguator since the LG parser multiplies out all the possible attachment variations, giving as a result a list of alternative dependency structures for each sentence. ExtrAns uses Brill's disambiguator in the following form: for every dependency struc-

ture, the attachment information is translated into four-tuples plus two fields like above where the second field is now used to express the actual attachment decision in the dependency structure. After the disambiguator is run the first two fields are compared. Equal fields indicate a correct attachment decision in the dependency structure. Only the set of dependency structures that has the highest ratio of correct attachments passes the filter (Mollá D. & Hess M. 2000).

ExtrAns includes — besides transitive verbs — all categories of verbs, multiple PP attachments, gerund and infinitive constructions. First the training was done with the Treebank corpus (Marcus M. *et al.* 1993) (the accuracy is reported to be 81.8%) but it turned out that this corpus was not appropriate for the Unix domain (the accuracy was only 72,8%). Therefore, the training for ExtrAns was redone with a subset of 34 manpages (containing a total of 1475 attachment decisions). Because there is no automatic way to collect the relevant training data, it was necessary to encode them manually by reading the manpages and writing out the attachment decisions.

Not surprisingly, the rules generated from the manpages are more accurate in our domain (76.6% correct disambiguations), although the number of training rules (116 rules) for the test set is far smaller than those for the Treebank corpus (1770 rules). For the evaluation we had to split up the subset of 34 manpages into a training set of 17 manpages and a test set of the same size.

There are other types of ambiguities that cannot be treated by the disambiguator. As we will see, in such cases all competing readings of an ambiguous sentence are asserted into the knowledge base where they are fused, graded and presented in context during the retrieval procedure.

1.5. Anaphora resolution

Anaphora resolution in the current version of ExtrAns is restricted to pronominal cases. The anaphora resolution algorithm used by ExtrAns relies on purely syntactic information (Lappin S. & Leass H. J. 1994). Lappin & Leass' algorithm uses the syntactic representations generated by the Slot Grammar parser (McCord M. *et al.* 1992) which contain information about the head-argument and head-adjunct relations and how these relations are realised (e.g. as subject, agent, object, indirect object, or prepositional object). Since Slot Grammar is dependency-based, the relevant information can be emulated in ExtrAns by checking the link types returned by the LG parser. The implemented algorithm contains among others the following main components: an intrasentential syntactic filter, a morphological filter, an anaphora binding algorithm, and a salience weighting procedure, very much like the original (Lappin S. & Leass H. J. 1994).

The output of the anaphora resolution algorithm includes a list of equivalence classes. The equivalence classes group those words that refer to the same object.

In contrast to the original algorithm, ExtrAns' implementation restricts intersentential anaphora resolution only to the previous sentence. There are two reasons for this decision. First, in the Unix domain a pronoun corefers very rarely with a noun phrase that is not in the same or in the previous sentence. Second, computing the salience measure for all (theoretically) possible candidates takes a lot of time and space.

Lappin & Leass report an accuracy of 89% for intrasentential cases. We expect that ExtrAns' anaphora module has similar accuracy, since it is a direct implementation of the original algorithm.

1.6. The minimal logical forms

An important requirement of ExtrAns is that it must be fast and robust enough to be able to cope with the manpage sentences. This must apply not only to the processing of the manpages but also to the retrieval procedure to find the answers to the question. The format of the logical forms plays a crucial role in the latter. The logical forms must be simple so that they are easy to construct and to use in the retrieval stage. Still, they must remain expressive enough for the task at hand. For that reason, it is convenient that they allow the addition of more information in a monotonic way (incrementality), so that further refinements of the logical forms can be added without having to modify the notation or destroy any old information.

To fulfill these features, ExtrAns' logical forms consist merely of conjunctions of predicates where all the variables are existentially closed with wide scope. This simple notation is easy to build and easy to use. It is also incremental, since further extensions to the meaning can be done simply by adding more predicates to the conjunction. To make the logical forms expressive enough, we resort to reification and a particular interpretation of existence and of the logical operators.

1.6.1. Reification

By reification we mean that some "abstract" concepts introduced by predicates become "concrete". The effect of this extension is to provide handles that can be used to refer to these concepts later in the discourse. At the current stage of ExtrAns we reify predicates derived from open-class words. This is in contrast with (Hobbs J. R. 1985), who proposes to reify all the predicates available in a logical form — still, we do not rule out Hobbs' approach in applications that require a more detailed logical form representation. The predicates reified by ExtrAns are classified into three types:

Objects. A noun such as *cp* introduces the predicate `object(cp,o1,x1)`, and the 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, or in expressions of identity.

Events. A verb such as *copies* introduces `evt(copy,e1,[x1,x2])`, and the 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, prepositional phrases, 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, like in the sentence *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)`.

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 task at hand. Several examples of MLFs are:

(2) *cp copies long files*
holds(e1)[2]*, object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2), prop(long,p2,x2)*

(3) *cp copies possible files*
holds(e1), object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2), prop(possible,p2,o2)

(4) *cp copies very long files*
holds(e1), object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2), prop(long,p2,x2), prop(very,p3,p2)

(5) *cp copies files quickly*
holds(e1), object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2), prop(quick,p3,e1)

Here we can see reification at work. The adjective *long* in (2) modifies the noun *files*, and accordingly the predicate introduced by the adjective modifies the object `x2`. However, the adjective *possible* in (3) is used intensionally. What is possible is the fileness of `x2`, that is, `o2`. Accordingly, the predicate introduced by the adjective modifies `o2`. Finally, the adverb *very* in (4) modifies the adjective *long* (whose reified concept is `p2`), whereas *quickly* in (5) modifies the verb *copies* (that is, `e1`).

[2]We will discuss the predicate `holds(.)` in Section 1.6.2.

1.6.2. Existence and logical operators

Reification can also be used to encode existence of concepts. Reified concepts may or may not exist in the real world. Existential quantification alone only guarantees existence in a Platonic universe of possible entities. To express that an event `e` actually exists in the world of manpages, ExtrAns uses a specific predicate (Hobbs J. R. 1996), thus giving `holds(e)`. In those cases where there is not enough information for ExtrAns to conclude that an event exists, nothing is said about it. For example, the copying event in (6) clearly holds in the world of Unix manpages, whereas one cannot say the same in (7) and (8), and it is not clear in (9):

(6) *cp copies files*
`holds(e1), object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2)`

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

(8) *cp does not copy a file onto itself*
`not(e1), object(cp,o1,x1), evt(copy,e1,[x1,x2]), object(file,o2,x2), onto(e1,x2)`

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

What holds in (7) is the refusing event `e2`, but we do not say anything about the status of the copying event `e1`. One could argue that `e1` does not hold because of the lexical meaning of *refuse*, and therefore we should also add `not(e1)`. However, for the time being ExtrAns does not decompose lexical meaning and as a consequence the negation cannot be deduced. The information is therefore left underspecified. If needed, and provided that we have enough knowledge to assess negation or assertion, this information can be inferred and added in a later stage.

In the same way as with existence of concepts, all logical operators are translated as regular predicates over reified concepts. For example, the negation in (8) is represented as a predicate over the concept of this particular copying event, and the implication in (9) is a predicate over the concepts of a particular typing and a particular copying event.

The translation of existence and logical operators into regular predicates is a means to convert embedded structures into flat structures, allowing the MLFs to be simple conjunctions of predicates. There are some restrictions on the expressivity of the MLFs (for example, the current MLF of *cp is not the command that removes files* would be equivalent to that of *cp is not a command*

but it removes files), but they still remain accurate enough for the task of AE. In fact, attempting a more complex and exhaustive representation may represent a hindrance later. For example, (9) is an informative answer to the question *which command copies files?*, and it can be inferred straightforwardly with the current logical forms. If we had converted (9) into a logical form with a logical implication and we wanted to check if it is a good answer to the question, we would face the impossible task of trying to decide whether the antecedent *the user types y* is true.

1.6.3. Incrementality

Since ExtrAns returns logical forms that are minimal, there is missing information. For example, tense, aspect, and temporal relations are all ignored. This is a direct consequence of the domain of Unix manpages for which the current work is conceived, where temporal relations are of little relevance. Other clusters of information are also ignored, such as plurality, modality, and quantification. Each of these topics deserves a wealth of research well outside the scope of this paper, such as Hobbs' work on quantification (Hobbs J. R. 1996).

The main reason why we do not want to encode all the information available in the data is 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 above, (2) to (9) are all informative answers to queries like:

(10) *which command can copy files?*
which commands copy files?
which command can copy a file?
which command copies all my files?

If one were to implement modality, plurality, and quantification, one would need to add also the right inference rules to be able to retrieve (2) to (9). In ExtrAns we spare the effort of trying to encode semantic information that is not going to be used later or that does not significantly improve the overall performance of the system. The use of incrementally extensible MLFs allow us to find the right balance between processability and expressivity.

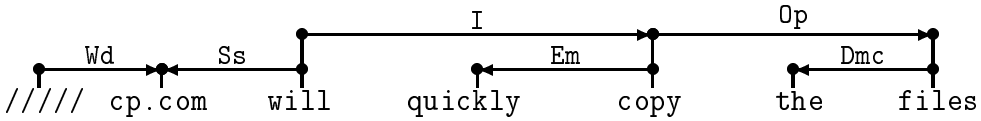
Using flat forms with reification thus enables us to underspecify and to construct only the part of the semantic information that we need (the MLFs). Different tasks require different degrees of detail in the MLFs. If, in a subsequent stage in the ExtrAns project, we decide to perform more ambitious AE, or even attempt QA, we can enrich the semantic information by adding more predicates to the MLFs. Another advantage of using this notation is the possibility of computing the semantic closeness between the question and the potential answer, as we will see later when we discuss approximate matching.

1.6.4. Implementation

The input of the MLF generator is a set of dependency structures complemented with the list of equivalence classes that are computed by the anaphora

resolution algorithm. The MLF generator converts all of this information into the MLFs of the sentences. The general procedure is a top-down algorithm that starts from the head of the sentence and follows the dependencies until they all are covered. Take, for example, (11):

(11) *cp will quickly copy the files*

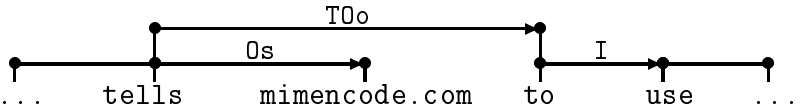


The general algorithm is:

1. Starting from the main verb (*copy*), find the leftmost auxiliary verb (*will*);
2. Starting from the auxiliary verb, find the head of the subject (*cp.com*) and build its logical form, `object(cp,oa1,x1)`;
3. Build the logical forms `object(file,oa3,x6)` of the objects (only one object in (11));
4. Create an entity for the main eventuality, `e4`;
5. Build the logical forms of other modifiers `prop(quickly,p3,e4)`;
6. Add the logical form of the main event. The final MLF becomes (new information in boxes):
   `holds(e4)`, `object(cp,oa1,x1)`, `evt(copy,e4,[x1,x6])`,
   `object(file,oa3,x6)`, `prop(quickly,p3,e4)`.

However, several complications in the syntactic structures oblige us to add particular cases to the general procedure, depending on which part of the sentence we are processing. For example, we need to traverse `Wd`, `Ss`, and `I` before finding the main verb *copy* in (11). A general difficulty is the existence of several types of dependency structures that should produce the same MLF. Examples are the use of a passive versus an active form, different variations of a sentence with a verb that can take a ditransitive form, or the analysis of questions. Another difficulty is that sometimes the output of LG does not show all the deep-syntactic dependencies that exist in the sentences, and these must be inferred by the MLF generator. For example, the dependency structure in (12) represents *mimencode* as the direct object. However, the MLF must show that *mimencode* is the subject of *use*:

(12) *The "-b" option tells mimencode to use the "base64" encoding*



..., object(option,o1,x3), evt(tell,e4,[x3,e7]),
object(mimencode,o7,x5), *evt(use,e7,[x5,x10])*,
object(encoding,o4,x10), ...

The resulting algorithm becomes very complex and a detailed explanation is beyond the scope of this paper. The interested reader is recommended to consult (Mollá D. 1998; Mollá D. *et al.* forth.).

After computing the MLFs, the information about word equivalences that results from the anaphora resolution is added to the formulas by making sure that all equivalent words introduce the same object variable, such as in (7). To achieve this, only a simple variable replacement is needed.

To enhance robustness, the "null-linked" words produced by LG are converted into keywords by adding a predicate `keyw(Word)` to the MLF. New keywords are also introduced by the MLF generator whenever it cannot process a specific link of a dependency structure.

1.7. Horn clause logic

For processing reasons we use Horn Clause Logic (HCL) to retrieve the answers. In particular, the logical form of the query is converted into a Prolog query, which is run against the logical forms of the manpage sentences stored as Prolog facts. For example, the query *does cp copy files?* translates into (13), which succeeds over any of the translations of (2) to (9) above.

(13) `?- object(cp,_,X), evt(copy,_,[X,Y]), object(file,_,Y).`

To be able to display the sentences with the selective highlighting, every predicate in the logical form contains a pointer to the sentence identifier which is provided by the tokeniser, an index indicating the interpretation number, and a list of pointers to the words that are involved in the predicate. Thus, a more accurate description of *cp copies files* (6) is:

(14) `holds(e1)/sent1/1~[].  object(cp,o1,x1)/sent1/1~[1].`
`evt(copy,e1,[x1,x2])/sent1/1~[1,2,3].`
`object(file,o2,x2)/sent1/1~[3].`

As we see, the logical form is encoded as a set of independent Prolog facts, one per predicate. The variables in the logical form are converted into skolem constants. This has been done by adding the sentence identifier and the interpretation number, since all the variable identifiers are always unique within a sentence interpretation. Thus, the variable `x1` in the logical form is converted into the Prolog constant `x1/sent1/1` in the database, and so forth. To ease readability, skolemisation is not shown in the examples of this paper.

To find all the solutions of the query, ExtrAns uses the built-in Prolog predicate `findall`. Thus, a more accurate representation of (13) is:

(15) `?- findall([S,I,R1,R2,R3], (object(cp,_,X)/S/I~R1,`
`evt(copy,_,[X,Y])/S/I~R2, object(file,_,Y)/S/I~R3), Res).`

After running (15) the Prolog variable `Res` is instantiated to:

```
[..., [sent1,1,[1],[1,2,3],[3]], ...]
```

With this information we know that the first interpretation (`1`) of sentence `sent1` (14) answers the query, and the words in position 1, 2, and 3 (the union of `[1]`, `[1,2,3]`, and `[3]`) are to be highlighted.

To increase efficiency, the result of inferences that can be done in the off-line stage are also added to the databases. We have seen already that the tokeniser can recognise command names, arguments, path names, etc. Given the information provided by the tokeniser, predicates are added accordingly. For example, the predicate `object(command,o2,x1)/sent1/1~[1]` is inferred from *cp.com* by the logical form generator and it is added accordingly (these predicates were not shown in the previous examples for exposition purposes). Other simple inferences can also be made. For example, if a logical form represents the sentence *cp copies files and directories*, extra predicates are added to make sure that queries about copying only files (or only directories) are successful too.

Finally, the logical forms of the query and the data also contain information about noun and verb synonymy. Synonymy information is encoded in a small WordNet-like (Fellbaum C. 1998) hand-crafted thesaurus that uses a synset identifier for each set of synonyms. This synset identifier (of the form `s_word` in this paper) appears in the Horn clauses that are stored in the databases, and also in the Prolog queries[3]:

(16) *cp copies files*

```
holds(e1)/sent1/1~[].  object(cp,o1,x1)/sent1/1~[1].
object(s_command,o2,x1)/sent1/1~[1].
evt(s_copy,e1,[x1,x2])/sent1/1~[1,2,3].
object(s_file,o2,x2)/sent1/1~[3].
```

(17) *which command copies files?*

```
?- findall([S,I,R1,R2,R3], (object(s_command,_,X)/S/I~R1,
evt(s_copy,_,[X,Y])/S/I~R2, object(s_file,_,Y)/S/I~R3),
Res).
```

Using MLFs with synonymy information allows us to generate the Prolog query (17) from any of the queries in (10) above, and many other variations with different synonyms. The query would retrieve any of the sentences (2) to (9) and variations.

1.8. Answer extraction

Various enhancements over the general retrieval procedure have been made in order to allow ExtrAns to work with larger volumes of data (Mollá D. &

[3]If a word has sense ambiguity, its synset corresponds to the first sense defined in the thesaurus. We have not tried to implement another approach because, in the restricted domain where we are working, very few words have sense ambiguity.

Hess M. 1999). Several databases are used, one per manpage. When running the query, only those manpages that are likely to have the answer are preselected. The preselection criterion is based on whether the manpage contains all the predicates of the query. Thus, a general operation of set-intersection is performed on the sets of preselected manpages for each predicate in the query. This basic criterion must be refined to accommodate the possibility of disjunctions in the query, such as when the query has unresolved ambiguities (or hyponyms, as we will see later). Disjunction in the query translates into performing set-union in the sets of preselected manpages of the members of the disjunction.

To increase robustness in the results, a fall-back search strategy is added to the general retrieval procedure. When there are not enough answers with the default search, the user can try hyponyms. If there are not yet enough answers, the user can try approximate matching. Finally, if there are still not enough answers, the user can try keyword search. The user is always asked if (s)he wants to try the next stage in the fall-back search or stop. In all cases, the threshold beyond which the user is asked is 5 hits.

Hyponym search is attempted after the standard search with synonyms. As opposed to synonymy, there is no "hyponym set identifier" that can be used to replace the synset, and therefore all the hyponyms must be introduced by means of disjunctions. For example, let us assume that the thesaurus contains the information that *text* is a hyponym of *file*, and *download* is a hyponym of *copy*. Then, the query (17) can be converted into:

(18)
```
?- findall([S,I,R1,R2,R3], (object(s_command,_,X)/S/I~R1,
(evt(s_copy,_,[X,Y])/S/I~R2; evt(s_download,_,[X,Y])/S/I~R2),
(object(s_file,_,Y)/S/I~R3; object(s_text,_,Y)/S/I~R3)), Res).
```

Approximate matching is attempted after the hyponym search. Approximate matching is particularly useful when there is no real answer to the user query because the query is too specific. In this case, a matching algorithm is performed that computes, for every sentence from the manpages, how well it fits the user query. Only the best fits are returned to the user.

The fitness degree of a particular sentence is computed according to the number of predicates in the query that can be satisfied at once. To give an illustrative example, let us assume that the query *which command copies big files?* does not give any direct answer. The predicates used in the query are:

(19) `object(s_command,_,X)/S/I~R1`. `evt(s_copy,_,[X,Y])/S/I~R2`.
`object(s_file,_,Y)/S/I~R3`. `prop(big,_,Y)/S/I~R4`.

The set of boxes shows the largest range of predicates that can be jointly used to retrieve the sentence *cp copies files* (6). Since three predicates are used, (6) can be retrieved with score 0.75, that is, three out of four predicates from the query. If this is the highest score found across all the sentences in the

Sentence	Syn.	Hypo.	Approx.
which command copies files?	3,260		
is there a command that creates directories?	22,670		
how can I move files?	2,380		
how can I read my mail?	8,300		
how can I establish a link to a file?	110	180	117,860
which command prints files?	53,040		
how do I do to eject a floppy?	93,200		
which command brings me to my home directory?	100	160	56,200
how can I stop a process?	880		
how do I compile programs?	4,160		

Table 1: Response times against selected queries over 500 manpages, showing only the stages necessary for ExtrAns to find an answer. Times in milliseconds on a Solaris (Ultra-10) workstation.

manpages, (6) will be retrieved[4]. This mechanism is somewhat more complex because the query may also include disjunctions, and therefore a logical form may generate several scores. Only the best score would be used.

Approximate matching is a way to compute the semantic closeness between the query and the potential answer. The use of flat forms for representing the MLFs makes the algorithm for approximate matching fairly straightforward. If we had used embedded forms, the algorithm would have had to be far more complicated, since information embedded deeply in the logical forms would not have been readily available for comparison.

The preselection of manpages in the approximate matching stage must be less restrictive than in the synonym and hyponym stages, since a manpage may have the best answer available in the whole set of manpages, without satisfying all the predicates in the query. Thus, set-union of the partial sets of manpages is used instead of set-intersection. An obvious result is that the number of preselected manpages is much higher, and therefore the response time is longer. Table 1 shows the response times for different queries.

Keyword search is the last resort. The keywords selected are those words in the query that produce object, event, and property predicates (that is, all the open-class words). From these, those words that are too frequent in the manpages are ignored, and the rest are used in a “linguistic” query where no interdependencies are expressed, targeting only at the objects, events, and properties in the manpages. In particular, the keyword query of *which command copies files?* becomes:

[4]There is no score threshold. The sentences with the best score are retrieved, even if the score turns out to be rather low.

(20)
```
?- findall([S,I,R1,R2,R3], ( (object(s_command,_,_)/S/I~R1;
evt(s_command,_,_)/S/I~R1; prop(s_command,_,_)/S/I~R1),
(object(s_copy,_,_)/S/I~R2; evt(s_copy,_,_)/S/I~R2;
prop(s_copy,_,_)/S/I~R1),
(object(s_file,_,_)/S/I~R3; evt(s_file,_,_)/S/I~R3;
prop(s_file,_,_)/S/I~R3) ), Res).
```

The use of keywords is not only restricted to keyword search. Words that were neither processed by the parser nor by the MLF generator are converted into keywords by adding a conjunction (not a disjunction) with an object, an event, and a property, to the Horn clause representation. In this fashion we ensure that, even if some words of the sentence have not been analysed, they are still considered in the retrieval procedure.

We are planning to introduce DR techniques in the retrieval procedure. A clear case where DR is desirable is in the manpage preselection, especially when approximate matching is needed to find a near miss. Another clear possibility of integrating DR techniques is in the keyword search which is, after all, a “bag of words” approach.

1.9. Graphical user interface

All extracted sentences are displayed by the Graphical User Interface (GUI) of ExtrAns. The GUI can produce three types of interfaces: a no-frills text interface, an HTML interface for accessing ExtrAns through the Internet[5], and a Tcl/Tk-based window interface. Figure 2 shows the window interface displaying the first hits for the user query *how can I create a directory?*.

The specific answers are highlighted and the hits are scored according to their adequacy. The scoring system is still preliminary and object of our research. The user can access the complete manpage that contains the sentences by clicking on the manpage name at the left of each sentence. It is now very easy for the user to figure out by inspecting the context whether an extracted sentence contains in fact the answer to the question.

The degree of highlight — expressed in a percentage scale in Figure 2 — indicates how confident ExtrAns is that a part of a sentence answers a user query. This presentation technique for search results was developed to deal with unresolvable ambiguities and multiple interpretations. In brief, document sentences may answer a user query in different ways for any of the following reasons: (i) A document sentence may contain unreducible ambiguities. Although ExtrAns tries to eliminate as many ambiguities as possible, some of them will always survive. In this case ExtrAns asserts all competing logical forms for a document sentence and includes all of them in the proof. (ii) A document sentence can have multiple interpretations. In the case of a string like *bin-mail, binmail — an early program for processing mail messages* ExtrAns sends two different sentences to the parser; one for *bin-mail* and one for

[5] http://www.ifi.unizh.ch/cl/extrans

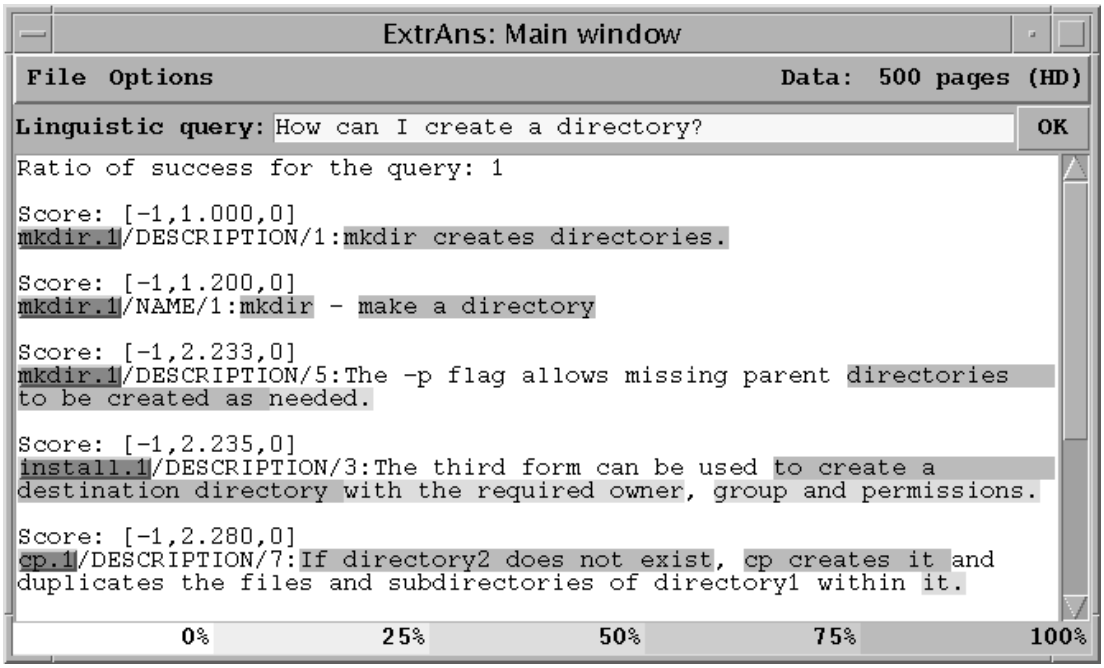


Figure 2: Retrieved sentences for the query *how can I create a directory?*

binmail. Both interpretations are stored in the knowledge base as independent logical forms and can lead to more than one correct proof. (iii) A single logical form can provide multiple answers. It is possible that different sets of facts of the same logical form can independently answer a user query. This occurs, for instance, if a sentence has a coordinated structure. For example, the document sentence *mv moves files and directories* can be proved in three different ways for the query *what does the command mv move?*: once for *file*, once for *directories,* and finally once for the whole coordinated noun phrase *files and directories*. (iv) A user query can be ambiguous, too. When a user query is ambiguous, different interpretations of the same query may be proved by the logical forms of a particular sentence.

On the whole, all words of an extracted document sentence that answer a user query are highlighted. Since the same sentence may answer the query several times in a different way, each word is ranked according to its frequency in all the solutions found. Those words that occur in more solutions are highlighted with a brighter colour on account of a graded colouring scheme. By this kind of visualisation, adequacy is seen as a minimal degree of ambiguity in the extracted answer. This has the big advantage that the user does not need to perform any interactive disambiguation.

2. EVALUATION OF EXTRANS

We present in this section a small evaluation of ExtrAns based on a set of 30 queries over 500 manpages. Table 1 in Section 1.8 shows the first ten of them. The queries have been selected according to the following criteria:

- There has to be at least one answer in the manpage collection;
- The query asks about how to perform a particular action (e.g. *how do I compile programs?*) or how a particular command works (e.g. *can rm delete nonempty directories?*);
- The query is simple, i.e. it asks only one question (for example, *how can I move or rename files?* is not valid).

Since AE lies between DR and QA, we had to decide what kind of evaluation procedure we could use to take its peculiarities into account. In the QA area, evaluations tend to compare the output of a system with an ideal answer, either by determining this answer a priori and measuring the distance of each candidate answer with it (Hirschman L. *et al.* 1999), or by considering the retrieved items until one correct answer was found and measuring the score of the answer according to its rank (Voorhees E. M. & Tice D. M. 1999). But AE is not meant to restrict its output to a unique answer, which is often nearly impossible to determine anyway. If the user has several possible answers at his/her disposal, (s)he will be able to choose which one corresponds to the present information need, and the choice is optimal if all answers are listed in the output. This criterion corresponds to the standard DR measure of recall, i.e. the ratio between the relevant hits and the total number of relevant sentences in the collection. To determine whether the system is able to filter out the non-relevant sentences, we also decided to compute precision, i.e. the ratio between the total number of relevant sentences and the total number of hits.

2.1. Method

In a first step, we searched our collection for sentences that provided satisfactory answers to our queries. To do this, we defined for each query a set of keywords enhanced with all the related terms we could think of (synonyms, hyponyms, truncated forms, nominalisations etc.). These keywords were then searched for in the set of sentences available to ExtrAns. When a relevant sentence was found in a manpage, all the sentences in the manpage were also checked. With this procedure, we believe to have retrieved a reasonably high amount of answers to each query: it is known that if we provide enough appropriate keywords to a search engine, recall will be high. Since the retrieved sentences were then selected manually, we could also expect high precision. The relevance criterion used for the selection was defined pragmatically. First of all, the retrieved sentence must contain the element that has been asked for. Second, the user must be able to *recognise* this element as being the answer

to the query, otherwise (s)he will not be able to do anything with it. In other words, a sentence is relevant if it is informative and explicit enough to satisfy most users. In practice, this principle had to be converted into more specific assessment rules, which lead to an average of 7 answers found for each query.

We are conscious that such a method cannot ensure the retrieval of all and only the relevant sentences. Nevertheless, this approximation can be useful for a comparison with another retrieval system. The system chosen for the comparison is Prise, a DR application developed by NIST (Harman D. K. & Candela G. T. 1989). Since Prise returns full documents, we used ExtrAns' tokeniser to find the sentence boundaries and to create independent documents, one per sentence in the manpages. Then Prise was run with our set of queries, which lead to an average of 908 hits per query, that is, roughly 30% of the initial set of sentences in the manpages. This high number of hits is explained by the fact that Prise retrieves all sentences that contain at least one of the (truncated) query terms, as long as these are not stop words. But a DR system like Prise compensates its low precision by ranking the hits that are assumed more relevant at the top of the list. The sentences with the highest scores are the ones that contain a sufficient proportion of query terms with respect to their total number of words, be it several times the same query term or several different query terms (or both)[6]. Thus if we want our comparison to be fair, we have to take the sentence ranking into consideration. We did this by reducing the number of hits considered for the computation of recall and precision to 100, which enhanced precision without too much penalising recall (in our evaluation, 73% of the relevant hits belong to the top 100 hits). In order to refine the analysis of the ranking, we also computed recall and precision with respect to the n first hits, where $n = 10, 20, \ldots, 100$.

For ExtrAns, we still had to determine which stage was to be taken into account in the evaluation (see Section 1.8), because the evaluation was done in batch mode, and no user could assess whether (s)he had got enough answers or (s)he wanted to try the next stage. We set the minimal number of hits in a stage to one, i.e. if at least one sentence was found in a stage, ExtrAns stopped and we counted the relevant hits in that stage. With this method, we got about 15 hits per query.

2.2. Results

Due to the small number of queries and hits returned by ExtrAns, we found it more convenient not to compute average curves of the recall and precision pairs, but just to plot these pairs, with an indication of the stage where ExtrAns stopped. The combined plot of pairs computed for each n did not

[6]The first parameter (frequency of a term in the document) is less significant when we consider sentences instead of documents, because the variation of the term frequency in a sentence is much smaller than in longer documents. The second parameter (number of terms in the document), however, is more relevant for sentences, because it is more likely that words that appear in the same sentence are related with each other.

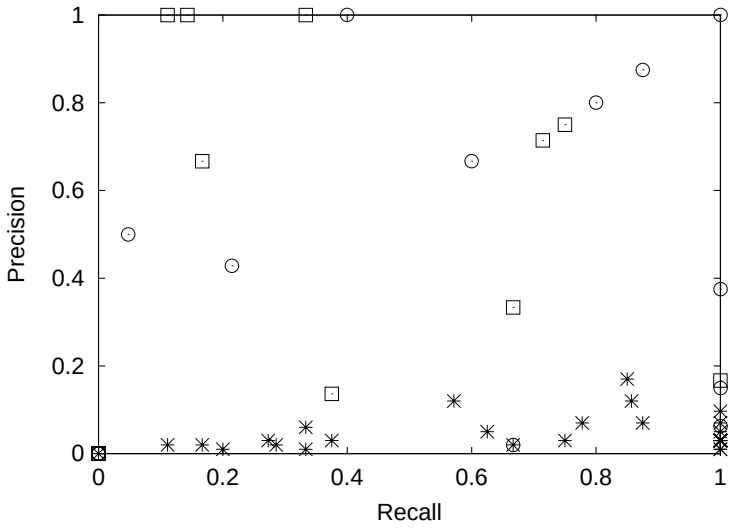


Figure 3: Recall against precision for 30 queries and the top 100 hits per query. Prise's results are displayed with a star ($*$), and ExtrAns' results with circles ($\odot$) for the default search and with squares ($\Box$) for the approximate matching.

show significant differences with the plot for $n = 100$: the values for ExtrAns were nearly the same, and for Prise, the number of recall and precision pairs increased but the area with the highest density of points remains the same. We will therefore concentrate on the plot for $n = 100$.

Figure 3 shows that precision is in general higher for ExtrAns than for Prise, and that Prise has the best recall values. In the upper right corner, we can see a higher density of ExtrAns' values which is likely to shift to the left if we use a less restricted set of queries. The fact that ExtrAns never stopped at the hyponym and keyword search is also related to the actual query set. If the queries were more complex, we would have some recall and precision pairs corresponding to the keyword search, and this would probably cause a lower overall precision. We are still improving the first stages, in order to avoid the keyword search in as many cases as possible. We expect that ExtrAns' precision will still remain higher than Prise's in most cases and that recall will depend on the performance of the first stages.

CONCLUSION

ExtrAns is an implementation of an AE system over the Unix manpages. The manpages and the user queries are converted into Horn clause representations of their MLFs and the retrieval procedure consists in trying to prove the query over the manpage sentences. The construction of the MLFs relies on full parsing, disambiguation, and anaphora resolution, among other linguistic procedures, which makes the process relatively costly in terms of time

and memory. Still, an adequate combination of these linguistic modules with keyword introduction, document preselection, and a fall-back strategy makes the system fast and robust enough for moderate numbers of technical documents such as the Unix manpages. The use of adequate displaying techniques such as graded highlighting and pointers to the source manpages helps the user to assess whether the results are good answers to the query. Although the domain of ExtrAns is that of Unix manpages, many of its modules can be reused for other domains — as we are currently doing with WebExtrAns. In principle, only the tokeniser would need major modifications (because it is based on the specific format of the manpages), plus a list of domain-dependent terms and the corresponding thesaurus. We have shown that currently available NLP technologies can be used to implement practical AE systems over real-world domains in real-world situations. ExtrAns is such a system, and a web-based interface can be found on the Internet.

The main advantage of ExtrAns against a “bag of words” approach is that crucial information about the semantic information is kept in the MLFs. This allows ExtrAns to retrieve fairly accurate answers. ExtrAns, however, is not a full-blown QA system, and it makes very shallow inferences only. Thus, queries that require planning and world knowledge (such as *why?*) would not give satisfactory answers.

Current research is focusing on the linguistic and the processing side. The former includes a revision of the MLF notation, the analysis of nominalisations, and other ways of expressing the same information with different structures. The latter includes coping with larger documents over more varied domains, possibly by integrating DR techniques. A more exhaustive evaluation of ExtrAns is also under consideration.

REFERENCES

APPELT, Douglas E. ; HOBBS, Jerry R. ; BEAR, John ; ISRAEL, David ; KAMEYAMA, Megumi ; TYSON, Mabry (1993) : “SRI: description of the JV-FASTUS system used for MUC-5”, *in Proc. MUC-5*, pp. 221–235, Los Altos, CA.

BRILL, Eric ; RESNIK, Philip (1994) : “A rule-based approach to prepositional phrase attachment disambiguation”, *in Proc. COLING ’94*, pp. 998–1004, Kyoto, Japan.

BURKE, Robin D. ; HAMMOND, Kristian J. ; KULYUKIN, Vladimir A. ; LYTINEN, Steven L. ; TOMURO, Noriko ; SCHOENBERG, Scott (1997) : *Question Answering from Frequently-Asked Question Files: Experiences with the FAQ Finder System*, Rapport technique n TR-97-05, University of Chicago, Dept. of Computer Science.

CHINCHOR, Nancy A. (1998) : “Overview of MUC-7/MET-2”, *in Proc. MUC-7*, SAIC, San Diego, CA, `http://www.muc.saic.com`.

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.

DEERWESTER, Scott ; DUMAIS, Susan T. ; FURNAS, George W. ; LANDAUER, Thomas K. ; HARSHMAN, Richard (1990) : “Indexing by latent semantic ana-

lysis", *Journal of the American Society for Information Science*, vol. 41, n 6, pp. 391–407.

FELLBAUM, Christiane (1998) : "Wordnet: Introduction", *in WordNet: an electronic lexical database,* C. Fellbaum (ed.), Cambrige, MA, MIT Press, pp. 1–19.

FRIEDL, Günter ; MAYR, Heinrich C. (eds.) (1999) : *Proc. NLDB'99 – 4th Int. Conf. on Applications of Natural Language to Information Systems*, Klagenfurt, Austria.

GRINBERG, Dennis ; LAFFERTY, John ; SLEATOR, Daniel (1995) : *A Robust Parsing Algorithm for Link Grammars*, Rapport technique n CMU-CS-95-125, Carnegie Mellon University, Available at `http://bobo.link.cs.cmu.edu/grammar/html/bibliography.html`.

GRISHAM, R. ; SUNDHEIM, B. (1996) : "Message understanding conference - 6: a brief history", *in Proc. MUC-6*, ARPA, Los Altos, CA.

HARMAN, Donna K. ; CANDELA, Gerald T. (1989) : "A very fast prototype retrieval using statistical ranking", *SIGIR Forum*, vol. 23, n 3/4, pp. 100–110.

HERZOG, Otthein ; ROLLINGER, Claus-Rainer (eds.) (1991) : *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence - final report on the IBM Germany LILOG project*, Berlin, Springer-Verlag, *Lecture Notes in Computer Science*, volume 546.

HIRSCHMAN, Lynette ; LIGHT, Marc ; BRECK, Eric ; BURGER, John D. (1999) : "Deep Read: A reading comprehension system", *in Proc. ACL'99*, University of Maryland.

HOBBS, Jerry R. (1985) : "Ontological promiscuity", *in Proc. ACL'85*, University of Chicago, pp. 61–69, Association for Computational Linguistics.

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, CA, CSLI Publications, chap. 3, pp. 55–76.

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.

KATZ, Boris (1997) : "From sentence processing to information access on the world wide web", *in AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, Stanford University, Stanford CA.

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.

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.

MOLLÁ, Diego ; HESS, Michael (1999) : "On the scalability of the answer extraction system "ExtrAns"", In Friedl ; Mayr (Friedl G. & Mayr H. C. 1999), pp. 219–224.

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*, pp. 21–24, Paris.

MOLLÁ, Diego ; SCHNEIDER, Gerold ; SCHWITTER, Rolf ; HESS, Michael (forthcoming) : “Answer extraction using a dependency grammar in ExtrAns”, *T.A.L.*, vol. 41, n 1.

MOLLÁ, Diego (1998) : *ExtrAns: An Answer Extraction System for Unix Manpages – On-line Manual*, Rapport technique, Computational Linguistics, University of Zurich, `http://www.ifi.unizh.ch/CL/`.

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.

SALTON, Gerard ; MCGILL, Michael J. (1983) : *Introduction to Modern Information Retrieval. International Student Edition.*, Auckland, AU, McGraw-Hill, *Computer Science Series*.

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

STRZALKOWSKI, Tomek ; LIN, Fang ; PEREZ-CARBALLO, Jose (1997) : “Natural language information retrieval: TREC-6 report”, *in Proc. TREC-6,* E. M. Voorhees ; D. Harman (eds.), NIST-DARPA, pp. 347–366, Government Printing Office.

STRZALKOWSKI, Tomek ; STEIN, Gees ; WISE, G. Bowden ; PEREZ-CARBALLO, Jose ; TAPANAINEN, Pasi ; JARVINEN, Timo ; VOUTILAINEN, Atro ; KARLGREN, Jussi (1998) : “Natural language information retrieval: TREC-7 report”, *in Proc. TREC-7,* E. M. Voorhees ; D. Harman (eds.), NIST-DARPA, pp. 217–226, Government Printing Office.

VAN RIJSBERGEN, C.J. (1979) : *Information Retrieval*, London, Butterworths, 2 édition.

VOORHEES, Ellen M. ; HARMAN, Donna (eds.) (1999) : *The Eighth Text REtrieval Conference (TREC-8)*, NIST, `http://trec.nist.gov/pubs.html`.

VOORHEES, Ellen M. ; TICE, Dawn M. (1999) : “The TREC-8 question answering track evaluation”, In Voorhees ; Harman (Voorhees E. M. & Harman D. 1999), `http://trec.nist.gov/pubs.html`.

WILENSKY, Robert ; CHIN, David N. ; LURIA, Marc ; MARTIN, James ; MAYFIELD, James ; WU, Dekai (1994) : “The Berkeley Unix Consultant project”, *Computational Linguistics*, vol. 14, n 4, pp. 35–84.

WINIWARTER, Werner (1999) : “An adaptive natural language interface arquitecture to access FAQ knowledge bases”, In Friedl ; Mayr (Friedl G. & Mayr H. C. 1999), pp. 127–136.

WOODS, William A. (1997) : *Conceptual Indexing: A Better Way to Organize Knowledge*, Rapport technique, Sun Microsystems, Inc.