LT TTT – A Flexible Tokenisation Tool

Claire Grover, Colin Matheson, Andrei Mikheev[†], Marc Moens

Language Technology Group
University of Edinburgh, 2 Buccleuch Place
Edinburgh EH8 9LW, Scotland
{grover, colin, mikheev, marc}@cogsci.ed.ac.uk

Abstract

We describe LT TTT, a recently developed software system which provides tools to perform text tokenisation and mark-up. The system includes ready-made components to segment text into paragraphs, sentences, words and other kinds of token but, crucially, it also allows users to tailor rule-sets to produce mark-up appropriate for particular applications. We present three case studies of our use of LT TTT: named-entity recognition (MUC-7), citation recognition and mark-up and the preparation of a corpus in the medical domain. We conclude with a discussion of the use of browsers to visualise marked-up text.

1. Introduction

The LTG's Text Tokenisation Toolkit (LT TTT, Grover et al., 1999) was developed within an XML processing paradigm whereby tools are combined together in a pipeline allowing each to add, modify or remove some piece of mark-up. The tools are compatible with the LT XML toolset (Thompson et al., 1997) and use the LT XML API to manipulate attribute values and character data in XML elements and to address particular elements in XML streams. Different combinations of the same tools can thus be used in pipelines for achieving different text processing tasks.

LT TTT provides components over and above programs available with LT XML. Some of these are rule-based while others utilise the statistical technique of maximum entropy modelling. Although a text is required to be in XML format for processing, pipelines can be built where the input text is in some other initial format. Indeed, LT TTT can be used simply to convert non-XML mark-up to XML. Conversely, after tokenisation, LT TTT, LT XML or XSL tools can be used to convert XML mark-up to other formats; we discuss formats for visualisation of mark-up in Section 6.

LT TTT is available for free to individuals, researchers and development teams, provided its usage is restricted to non-commercial purposes. It can be accessed at `http://www.ltg.ed.ac.uk/software/ttt/`.

2. System Overview

The core LT TTT program is `fsgmatch`, a general purpose transducer which processes an input stream and rewrites it using a set of rules provided in a grammar file. The rule files for `fsgmatch` operate at one of two levels: at the 'character-level' they operate over character data inside an XML element and can rewrite it in arbitrary ways; at the 'xml-level' they operate over XML elements and typically group such elements into larger XML elements. A pipeline normally involves some initial processing at the character-level to perform, for example, segmentation of the character data contents of paragraphs into word tokens. As an example, in a corpus of newspaper articles, we might mark up the string "April 20: the government collapsed this afternoon." as the following sequence of tokens (W=word).[1]

```
<W>April</W> <W>20</W><W>:</W>
<W>the</W> <W>government</W>
<W>collapsed</W> <W>this</W>
<W>afternoon</W><W>.</W>
```

Using the pipeline architecture, we typically build up mark-up using several calls to `fsgmatch` interspersed with calls to other tools. A typical pipeline will include a call to our maximum entropy sentence boundary disambiguator, `ltstop`, after the word tokens have been marked up. This component decides whether a full stop is an end-of-sentence marker or part of an abbreviation. Once the full stops are disambiguated, an xml-level call to `fsgmatch` can be used to mark up sentences as XML elements. Another typical part of a pipeline is a call to `ltpos`, a part-of-speech tagger, since part-of-speech information is often critical in identifying larger chunks. The output at this stage might be:

```
<SENT>
<W P='NNP'>April</W>
<W P='CD'>20</W><W P=':'>:</W>
<W P='DT'>the</W>
<W P='NN'>government</W>
<W P='VBD'>collapsed</W>
<W P='DT'>this</W>
<W P='NN'>afternoon</W><W P='.'>.</W>
</SENT>
```

At a later stage, xml-level processing might be needed to group word tokens of particular types into larger units. For example, dates and times may be marked up:

[†] Now also at Xanalys Inc.

[1] In most of our examples, we use LT TTT just to add mark-up and not to change the input in any other way. This means that existing whitespace and newline characters are unaffected. However, we are unable to preserve the actual format of the examples within narrow columns and so we introduce extra linebreaks here which do not exist in reality.

<SENT>
<TIMEX TYPE='DATE'>
<W P='NNP'>April</W> <W P='CD'>20</W>
</TIMEX><W P=':'>:</W>
<W P='DT'>the</W>
<W P='NN'>government</W>
<W P='VBD'>collapsed</W>
<TIMEX TYPE='TIME'>
<W P='DT'>this</W> <W P='NN'>afternoon</W>
</TIMEX><W P='.'>.</W>
</SENT>

Recognition of high-level units, such as dates in our example, is typically done using `fsgmatch` rules which consult a pre-prepared lexicon of expected words. It is, however, possible to incrementally build a lexicon while processing a corpus. We discuss such cases in Sections 3 and 4.

The grammar rule files for `fsgmatch` are themselves XML documents with each rule an instance of a `rule` element. The following are some simplified rules for recognising month-date units such as "April 20" in our example.

```
<!-- January, February, ...  -->
<RULE name = "month-name">
<REL match = "W[P='NNP']">
  <CONSTR
  check_in="LEX" check_tags="MONTH *">
  </CONSTR>
</REL></RULE>

<!-- a possible date (up to 31) -->
<RULE name="poss-date">
<REL
  match=
  "W[P='CD']/#~^([1-9]|[12][0-9]|3[01])$">
</REL></RULE>

<!-- February 4, ...  -->
<RULE name="month-date"
      targ_sg="TIMEX[TYPE='DATE']">
<REL type="REF" match="month-name"></REL>
<REL type="REF" match="poss-date"></REL>
</RULE>
```

The rule `month-name` matches a `W[P='NNP']` element if it occurs in a lexicon, `LEX`, with a category tag `MONTH`. The rule `poss-date` matches a `W[P='CD']` element whose contents match the regular expression which picks out the numbers 1–31. (The regular expression language is the standard Unix/Perl regular expression language.). The rule `month-date` looks for two consecutive strings as defined by the rules `month-name` and `poss-date` and when a successful match is found, the XML element `TIMEX[TYPE='DATE']` is wrapped around the entire string.

The rules for `fsgmatch` are deterministic, with the first matching rule being the one that succeeds. The potential disadvantages of such a system are overcome by allowing the user to place constraints on the left and right context of a string to be matched to ensure that a rule only applies when intended. For example, the month names "April" and "May" are also possible first names of a person. When marking up dates one therefore has to be careful that the surrounding context of an instance of "April" or "May" is truly indicative of a date. One can be sure that "12th April 2000" is a date because of the preceding ordinal and the following year identifier, but in examples where the month name occurs on its own, more care is needed. A preceding preposition such as "in" or "before" is a good indicator of a month, whereas a following proper name is not. In some cases the string will be truly ambiguous and annotation indicating the presence of ambiguity might be called for. In Section 3 we describe our named entity recognition system which implements a stepwise approach to resolving such cases of ambiguity.

While pipelines tend to incrementally add mark-up to a document, it may be desirable to shed some interim processing results in the final output. In this case a call to the program `sgdelmarkup` can be included in the pipeline. In our example, once the dates have been identified, it may be desirable to remove all other mark-up:

<TIMEX TYPE='DATE'>April 20 </TIMEX>: the government collapsed <TIMEX TYPE='TIME'>this afternoon</TIMEX>.

Similarly, the LT XML program, `sggrep`, can be used to extract the date and time elements:

<TIMEX TYPE='DATE'>April 20</TIMEX>
<TIMEX TYPE='TIME'>this afternoon</TIMEX>

In this section we have used a simplified example in order to give a brief overview of the types of processing that the LT TTT tools can be used for. In the next three sections we demonstrate the flexibility and reusability of our tools by focusing on three different tokenisation tasks that we have recently undertaken. The first example is our use of LT TTT to participate in the MUC-7 Named Entity Recognition Competition. Here the task was to recognise and mark up sequences of words which denote names of persons, organisations and places, temporal expressions and monetary amounts and percentages. The second example is the identification of citation and reference list elements in academic texts. Here we first mark up the reference list at the end of the paper, providing a list of author names to be used when searching the body of the text for citations. The third example is the conversion into XML of the OHSUMED corpus of Medline abstracts, its segmentation into sentences, part-of-speech tagging and general preparation for further linguistic processing. Finally, in Section 6 we discuss the use of LTG tools for visualising mark-up.

3. Named Entity Recognition

Named entity recognition involves processing a text and identifying certain occurrences of words or expressions as belonging to particular categories of named entities. Named entity recognition software serves as an important preprocessing tool for tasks such as information extraction, information retrieval and other text processing applications.

The LT TTT tools formed the core of the LTG's system (Mikheev et al., 1998) that was entered in the Named Entity task of the 7th Message Understanding Competition (MUC-7). This is a competition on information extraction from text, sponsored by the U.S. Defense Advanced Research Projects Agency (Chinchor, 1998). The

named entities our system recognises and the type of annotation it uses for the mark-up are therefore the ones stipulated by the MUC-7 competition rules. There are three kinds of named entity to be recognised: temporal expressions, numeric expressions and names of people, places and organisations. This must be marked using the SGML tags TIMEX, NUMEX and ENAMEX. TIMEX elements are either TYPE='DATE' or TYPE='TIME', NUMEX elements are either TYPE='MONEY' or TYPE='PERCENT' and ENAMEX elements are one of TYPE='PERSON', TYPE='LOCATION' or TYPE='ORGANIZATION'. The following are some examples:

- <TIMEX TYPE='DATE'>all of 1987</TIMEX>
- <TIMEX TYPE='TIME'>8:24 a.m. Chicago time</TIMEX>
- <NUMEX TYPE='MONEY'>several million New Pesos</NUMEX>
- more than <NUMEX TYPE='PERCENT'>95%</NUMEX>
- in <ENAMEX TYPE='LOCATION'>North and South America</ENAMEX>
- the <ENAMEX TYPE='ORGANIZATION'>U.S. Fish and Wildlife Service</ENAMEX>
- the <ENAMEX TYPE='PERSON'> Clinton</ENAMEX> government
- <ENAMEX TYPE='ORGANISATION'> Microsoft</ENAMEX> chairman <ENAMEX TYPE='PERSON'>Bill Gates</ENAMEX> said yesterday

The system that we built for the MUC-7 named entity task achieved a combined precision and recall score of 93.39. It is comprised of multiple processing layers linked together within the XML pipeline architecture with symbolic and statistical components interleaved. The first step converts the texts into XML from their original SGML format and performs initial segmentation into word tokens as sketched in the previous section. We found that the identification of NUMEX and TIMEX elements was relatively simple and could be performed at this stage with calls to `fsgmatch` using hand-crafted grammars for first NUMEX and then TIMEX elements.

The identification of ENAMEX elements is much harder and requires several passes through the text interleaving various kinds of processing. The initial step of the ENAMEX subtask is part-of-speech tagging with the maximum entropy tagger `ltpos` (Mikheev, 1997), discussed briefly in Section 2. Thereafter we proceed cautiously and make wide use of contextual information. A string of words like "Adam Kluver" has an internal structure which suggests that this is a person name; but we know that it can also be used as a shortcut for the name of an organization ("Adam Kluver Ltd.") or a location ("Adam Kluver Country Park"). Looking it up on a list will not necessarily help: the string may not be on a list, it may be on more than one list, or it may be on the wrong list. However, somewhere in the text, there is likely to be some contextual material which makes it clear what type of named entity it is. Our strategy is to only make a decision once we have identified this bit of contextual information.

We further assume that, once we have identified contextual material which makes it clear that "Adam Kluver" is (e.g.) the name of a company, then any other mention of "Adam Kluver" in that document is likely to refer to that company. If the author at some point in the same text also wants to refer to (e.g.) a *person* called "Adam Kluver", s/he will provide some extra context to make this clear, and this context will be picked up in the first step. The fact that at first it is only an assumption rather than a certainty that "Adam Kluver" is a company is represented explicitly, and later processing components try to resolve the uncertainty.

In our system, we implemented this approach as a staged combination of rule-based processing with probabilistic partial matching. Note that the rule-based processing is implemented using `fsgmatch` with hand-tailored grammars but that the partial matching software is not part of the LT TTT release. Nevertheless, the example is instructive since it demonstrates the utility of our incremental approach to adding mark-up whereby different tools can be brought in where necessary. Full details of our MUC-7 system can be found in (Mikheev et al., 1998, 1999a and 1999b) but we give a brief overview of the stages here.

Step 1. Sure-fire Rules

In the first step, the system makes a call to `fsgmatch` using a set of sure-fire grammar rules. These rules only fire when a possible candidate expression is surrounded by a certain context and they rely on known corporate designators (Ltd., Inc., etc.), person titles (Mr., Dr., Sen.), and other definite contexts. As already noted, part-of speech-tagging occurs prior to this stage as does a restricted stage of semantic tagging. Information from gazetteers is available at this stage but it is treated as *likely* rather than definite and is only utilised if the context is sufficiently suggestive. For example, names of possible locations found in our gazetteer of place names are marked as `LOCATION` only if they appear with a context that is suggestive of location. "Washington", for example, can just as easily be a surname or the name of an organization. Only in a suggestive context, like "in Washington", will it be marked up as a location.

Step 2. Partial Match 1

After the sure-fire rules have applied the system performs a probabilistic partial match of the entities identified so far. It collects all named entities already identified in the document and generates all possible partial orders of their composing words preserving their order, and marks them if found elsewhere in the text. For instance, if "Adam Kluver Ltd" had already been recognised as an organisation by the sure-fire rules, in this second step any occurrences of "Kluver Ltd", "Adam Ltd" and "Adam Kluver" are also tagged as *possible* organizations. This assignment, however, is not definite since some of these words (such as "Adam") could refer to a different entity. This information goes to a pretrained maximum entropy model (see Mikheev, 1998 for more details on this approach). The model takes into account contextual information for named entities, such as their position in the sentence, whether they exist in lower-

case in general, whether they were used in lowercase elsewhere in the same document, etc. These features are passed to the model as attributes of the partially matched words. If the model provides a positive answer for a partial match, the system makes a definite assignment.

Step 3. Rule Relaxation

Once this has been done, the system again applies `fsgmatch` with grammar rules but with more relaxed contextual constraints and with a new lexicon of names in the text identified in the previous stages of processing (i.e. a lexicon built 'on the fly' and local to the document(s) being processed). At this stage the system will mark word sequences which look like person names taking into account the new lexicon of names. For example, in expressions like "Murdoch's News Corp", the string "Murdoch's" could be part of the name of the organisation, or could be a possessive. Further inspection of the text reveals that Rupert Murdoch is referred to in contexts which support a person interpretation; and "News Corp" occurs on its own, without the genitive. On the basis of evidence like this, the system decides that the name of the organisation is "News Corp", and that "Murdoch" should be tagged separately as a person. Finally, during this stage known organizations and locations from the gazetteers available to the system are marked in the text, without checking the context in which they occur.

Step 4. Partial Match 2

At this point, the system has exhausted its resources (grammar rules for named entities, as well as its gazetteers). The system then performs another partial match to annotate names like "White" when "James White" had already been recognised as a person, and to annotate company names like "Hughes" when "Hughes Communications Ltd." had already been identified as an organisation. As before, this process of partial matching is again followed by a probabilistic assignment supported by the maximum entropy model. One of the texts in the competition contained the string "U7ited States and Russia". Because of the typo in "U7ited States", it wasn't found in a gazetteer. But there was internal evidence that it could be a location (the fact that it contained the word "States"); and there was external evidence that it could be a location (the fact that it occurred in a conjunction with "Russia", a known location). These two facts in combination meant that the system correctly identified "U7ited States" as a location.

4. Mark-up of Bibliographic Material

In this section we describe how we use LT TTT and other LTG XML tools to identify and mark up bibliographical information in academic texts. We assume that there are two main types of bibliographic information in documents: the reference list (or 'bibliography') which usually appears at the end of the text, and the in-text citations which normally point to items in the reference list. Typically, then, a reference list looks something like this:[2]

> Abelson, D., (1990). Preferential, cooperative binding of topoisomerase II to scaffold associated regions. EMBO J. 8 3997-4006.
>
> Cabelli, H.F., 1990. Promoter occlusion: transcription through a promoter may inhibit its activity. Cell 29 939-944.
>
> van Dijk, D., (1990). Regulation of the higher-order structure of chromatin by histones H1 and H5. J. Cell Biol. 90 279-288.

The other form of bibliographic material, in-text citations, come in two main forms - 'syntactic' and 'parenthetic'. A syntactic citation is part of the sentence which contains it:

> This is argued by Abelson (1990) and others, and Jones (1987) further claims that .....

while parenthetic citations are in the form of parenthetic comments:

> This has often been claimed (Abelson [1990]; Jones [1987]), and the data suggest that ....

The distinction is useful in that publishers typically insist on different forms for the two types. Generally, the order and presentation of bibliographic information varies fairly widely, of course, depending on publishers' individual conventions. Nevertheless there are many common factors which make it viable to use grammars to describe the material, and we have produced two fairly extensive example grammars for processing bibliographies and another for identifying and structuring the citations. The grammars are not intended to provide comprehensive coverage, but we claim that the general approach is viable for large-coverage systems. Note that in this context we do not use probabilities at any stage, although there are obvious places where a more extensive system could usefully employ the same statistical methods outlined above.

We began by writing a new DTD for bibliographic information which contains general information on the structure of citations and reference list items. The first stage in the process then converts plain text to XML, after which a small character-level grammar performs basic 'chunking' into very simple tokens. To take an example at this point, the journal information in the first example reference list item above (EMBO J. 8 3997-4006.) is tokenised as:

```
<W C='W'>EMBO</W> <W C='W'>J</W>
<W C='FS'>.</W><W C='CD'>8</W>
<W C='CD'>3997</W><W C='DASH'>-</W>
<W C='CD'>4006</W><W C='FS'>.</W>
```

Here we are not using a part-of-speech tagger, and the word elements have a class attribute ('C') which identifies the tokens as numbers, as themselves (dash, full stop, and so on), and as 'everything else' ('W'). No attempt is made to capture abbreviations and word-internal punctuation is not handled – so a hyphenated word such as "Johnson-Laird" will be three tokens. These are not necessary decisions and it is up to the grammar writer to decide what the appropriate split between character-level and xml-level processes should be. However, we certainly want to retain the integrity of the parts of a range specification such as

[2] The examples used here are drawn largely from a real reference list which was provided for the BibEdit project (Matheson and Dale, 1993) by Harcourt Brace Jovanovich.

"3997-4006" above, and in this context it is arguably simpler to leave the interpretation of hyphens to higher-level processing.

As for the reference list information, we assume that this is typed and structured more or less as suggested in the BibEdit project (see Matheson and Dale, 1992 for the relevant types), and so the first item in our example bibliography above will have the following general structure:

```
<REF>
  <AUTHOR>Abelson, D.,</AUTHOR>
  <DATE>1990</DATE>.
  <TITLE>
  Preferential, cooperative binding of topoisomerase II
  to scaffold associated regions.
  </TITLE>
  <JOURNAL>EMBO J. 8 3997-4006</JOURNAL>
</REF>.
```

One problem in automatically detecting this structure is in determining where the title ends – as titles can contain abbreviations, there is no obvious way of identifying the correct span in something like:

> van Stump, D., 1990. Regulation of the higher-order structure of chromatin by histones H1 and H5. J. Cell Biol. 90 279-288.

Given that "Cell Biol" is a possible journal name, it is difficult to stop the "J" being included in the title. In the absence of a complete list of journal titles, we assume that the best solution to this problem is to stage the process of structuring the reference lists, attempting to identify the publication information first using one grammar and then applying a second to find everything else. In this way, assuming that we always include the largest amount of material possible in the mark-up, we will pick out "J. Cell Biol" as the journal name before we look for the title. Below is an example of a marked-up journal:

```
<JOURNAL>
  <JNAME>J. Cell. Biol</JNAME>
  <VOLUME>5</VOLUME>
  <RANGE>
    <START>2689</START>
    <SEPARATOR>-</SEPARATOR>
    <END>2696</END>
  </RANGE>
</JOURNAL>
```

The reference list grammars thus produce a fairly detailed analysis of the input, and one interesting aspect of this is that the information on author names can be used in processing the text to find citations. An example of a single author name from a reference list is shown below:

```
<NAME>
  <PRENAME>Van</PRENAME>
  <SURNAME>Outen</SURNAME>
  <INVERTED>D.</INVERTED>
</NAME>
```

The 'inverted' part of a name represents the field which is typically inverted over the others in some circumstances ("Jones, Peter", "Heath, Sir Edward", and so on). The surname field is clearly very useful in looking for in-text citations, and hence with a marked-up reference list in place it is possible to use this information to identify citations with a high degree of certainty. One method of doing this is to create an 'on the fly' lexicon from the names in the reference list. We extract the SURNAME elements of the reference list and convert them to the `fsgmatch` lexicon format using the program `xmlperl` (McKelvie, 1999). This is a rule based transformation language which allows the rules that manipulate XML elements and element contents to contain Perl code. This is the `xmlperl` rule for surnames:

```
<rule query=".*/SURNAME/#">
   s%'% ' %;
   s%-% - %;
   if   ($_=~/\-|\'/)
          {print "$_  :: SURNAME\n"}
   else {print "$_  SURNAME\n"}
</rule>
```

A call to `xmlperl` with a rule file containing just this rule will create an output where everything apart from SURNAME elements is ignored. The query part of the rule uses the LT XML query language to pick out just the character data contents of SURNAME elements and the Perl part of the rule specifies a transformation of the character data: it puts spaces around word-internal quote marks and hyphens and then it appends the look-up tag SURNAME to the entry. The main part of the rule looks to see if the name does in fact contain internal punctuation, and if so, it prints an appropriate 'phrasal' lexical entry (one with a double colon separator). If not, a simple lexical entry is output. When run over a marked-up reference list, this will create a lexicon of the form:

```
Abelson              SURNAME
Baader               SURNAME
Cabelli              SURNAME
O ' Brien          :: SURNAME
Stainton - Ellis   :: SURNAME
```

The grammar for in-text citations can now use this lexicon when searching texts, and this both increases the accuracy of the search and allows a wider range of citations to be identified. To illustrate this, note that syntactic citations can be of the form "Jones 1990" as well as "Jones (1990)" – in other words, brackets round the date are not required in some styles. However, if we simply assume that proper names are just capitalised items, then we will overgenerate with data such as:

> In 1990, it rained a lot.
> There was no sun in August 1998.
> Apparently 1940 was a good summer.

If we are looking for capitalised words plus dates, we will identify "In 1990", "August 1998", and "Apparently 1940" as citations. With a lexicon of surnames, however, the search can be broader (allowing unbracketed dates), and more accurate in that we will also avoid suggesting three authors in syntactic citations like:

Apparently, Chomsky and Halle (1968) argue that ..

Note that the full TTT documentation (Grover et al., 1999) contains a detailed tutorial on the process of extracting and subsequently using lexicons, along with an extended description of the bibliographic grammars.

5. Medical Corpus Preparation

We have recently started a project called 'Data Intensive Semantics and Pragmatics' (DISP) which will apply a hybrid combination of statistical and symbolic processing at the lexical semantic level. The project is primarily a computational linguistics one where the aim is to investigate the semantic relations between nouns in complex nominals. The medical domain has been chosen because the field of medical informatics provides a relative abundance of pre-existing knowledge bases and ontologies. While the focus of the project is on semantic issues, a prerequisite is a large, reliably annotated corpus and a level of syntactic processing that can support the computation of predicate-argument relations. Since we need to compute semantic information, current approaches to 'shallow parsing' or 'chunking' are of little use to us (though they may be useful as a stage in overall processing) and we have therefore chosen to use the grammar development environment and wide-coverage syntactic and semantic grammar provided by the Alvey Natural Language Tools (ANLT) system (Carroll et al., 1991, Grover et al., 1993). Our chosen corpus is the OHSUMED Corpus (Hersh et al., 1994) which is a collection of MEDLINE abstracts of medical journal papers from the years 1987 to 1991 (see `http://www.ncbi.nlm.nih.gov/PubMed/`).

We are in the process of tuning the grammar to the language found in the corpus and when this is complete, we plan to use Briscoe and Carroll's (1993) extension of the ANLT software which uses probabilities to rank parse results so as to return the most probable syntactic analyses. Since we are using the full ANLT grammar, any parses found automatically provide an underspecified logical form computed compositionally from the parse tree. We hope to be able to parse and compute logical forms for a large proportion of the corpus and to further annotate the corpus with syntactic and semantic information in order to discover regularities in complex nominals.

A significant part of our effort so far has centered on the conversion of the OHSUMED corpus into XML annotated format and we have completed various stages including segmentation into word tokens, part-of-speech tagging and lemmatisation, using the LT TTT tools in combination with other programs. While the low-level tokenisation is much like our simplified description in Section 1, what is interesting in this task is the extent to which processing prior to parsing can be used to reduce the burden on the grammar and parser and the LT TTT toolkit has proved itself invaluable at this stage.

5.1. Conversion to XML

The OHSUMED corpus consists of several large ASCII files (one for each year) and contains just over 355,000 records of medical journal abstracts (though not all records actually contain the text of the abstract). An idiosyncratic coding scheme marks up the different parts of an abstract using line initial full stops paired with specific capitalised letters as identifiers of specific parts, starting with a unique identifier. Although we are primarily interested in the text of the abstract (encoded in a `.W` field), we have developed a pipeline for converting each entire record to XML where each field is packaged as a separate XML element. This conversion is achieved using two calls to `fsgmatch` with specialised grammars for this corpus. The following is an actual example after initial conversion to XML.

<TEXT>
<RECORD>
<ID>464</ID>
<MEDLINE-ID>87052753</MEDLINE-ID>
<SOURCE>
Contact Dermatitis 8703; 15(3):178-82
</SOURCE>
<MESH>
Adult; Aged; Aged, 80 and over; Dermatitis, Atopic/*DI; Dermatitis, Contact/*DI; Eyelid Diseases/*ET; Female; Hand Dermatoses/CO; Human; Male; Middle Age; Patch Tests.
</MESH>
<TITLE>
Eyelid dermatitis: the role of atopy and contact allergy.
</TITLE>
<PTYPE>JOURNAL ARTICLE.</PTYPE>
<ABSTRACT>
Patients with eyelid dermatitis were studied with patch tests and a clinical point method for the diagnosis of atopic skin disease. In 38 patients, contact allergy was found in 11. The dermatitis was an expression of atopic dermatitis in 15 patients.
</ABSTRACT>
<AUTHOR>Svensson A; Moller H.</AUTHOR>
</RECORD>

Once the texts of the abstracts are encoded as XML ABSTRACT elements, they can be extracted (using `sggrep`) and prepared as input to the parser. As sketched in Section 1, tokenisation into word units is performed and full stops are disambiguated. Since the parser expects sentences as input, it is crucial that SENTENCE element mark-up should be added at this stage.

5.2. Partial Tagging

The ANLT grammar is a unification grammar based on the GPSG formalism (Gazdar et al., 1985) which is a precursor of more recent 'lexicalised' grammar formalisms such as HPSG (Pollard and Sag, 1994). As such, lexical entries carry a significant amount of information including information about the subcategorisation properties of content words. Thus the practical parse success of any grammar is significantly dependent on the quality of the lexicon. The ANLT grammar is distributed with a large lexicon of varying quality: the function words such as complementizers, prepositions, determiners and quantifiers are all reliably hand-coded but content words are less reliable. Verbs are generally coded to a high standard but the noun lexi-

con is full of redundancies and duplications. If we try to parse OHSUMED sentences using the ANLT lexicon and no other resources, we will achieve poor results, mainly because many of the medical domain words are simply not in the lexicon but also partly because the coding of words which are in the lexicon is not always adequate. At later stages of development we hope to have medical domain specific lexicons integrated with the system but at this early stage we are exploring ways of combining ANLT lexicon information with the part-of-speech tags assigned by our tagger, `ltpos`. The idea is to ignore function word tags, since they are less reliable and less informative than the ANLT hand-coded lexical entries, but to retain content word tags. The system attempts to look up the word_tag pair by treating the tag as a novel kind of affix which constrains the category of the lexical entry it attaches to. Thus the string "block_NN" will be associated with the noun entry for "block" but not with any verb entry. If the word is not in the lexicon then the system falls back on default entries for the tag.

The ANLT parser expects plain ASCII input, not XML, and we are currently working with a system where the parser accepts either simple words or word_tag pairs. Once our SENTENCE elements have been tokenised and part-of-speech tagged, we append the tag to the word separated by an underscore, we adopt a format of one line per sentence and we remove all XML mark-up. Additionally, for the reason explained above, we dispense with all non-content word tags, i.e. all tags except for nouns (NN), verb (VB), adjective (JJ) and adverb (RB). We also split up possessive marked nouns ("patient's" becomes "patient 's") and separate the end of sentence full stop. Thus the first sentence of the example above is converted to the following format so that it can be input to the parser (line breaks inserted for presentation):

```
Patients_NNS with eyelid_NN
dermatitis_NN were studied_VBN with
patch_NN tests_NNS and a clinical_JJ
point_NN method_NN for the diagnosis_NN
of atopic_JJ skin_NN disease_NN .
```

We use `xmlperl` to perform this conversion, as describd in the previous section. For example, the rule for conversion of a `W[P='NNS']` element specifies that an underscore and the value of the attribute `P` should be appended to the character data content of the element and that the XML tag should be discarded. For an element such as `W[P='IN']` (a preposition), the XML mark-up is simply discarded.

5.3. Pre-empting Parser Choices

The process of preparing OHSUMED sentences for parsing described so far has involved no more than than standard tokenisation and part-of-speech tagging. However, it is possible to use LT TTT to perform other tasks which can substantially affect the behaviour of the parser. In the ANLT grammar, nouns are classified according to their subcategorisation properties in much the same way as verbs are. Thus one of the entries for the noun "quandary" specifies that it subcategorises for a PP complement headed by the preposition "about" (e.g. "the government's quandary about reform"). When the word_tag pair `quandary_NN` is looked up, all of the noun entries for "quandary" are returned including the one subcategorising a PP[about]. However, there is a class of nouns which are inadequately coded in the ANLT lexicon, namely deverbal nominalisations such as "management", "formation", "insertion". These typically occur in the ANLT lexicon as simple nouns even though they have a similar complementation pattern to the verbs from which they derive. Many of these nominalisations occur in the corpus immediately followed by a PP headed by "of" and it is clear that this PP is a complement of the noun corresponding to the direct object of the original verb. Thus we have "insertion of the needle" corresponding to "insert the needle" but no lexical entry for "insertion" with a PP[of] subcategorisation. Therefore, an attempt to parse a sentence containing "insertion_NN of the needle_NN" will either fail or yield an incorrect result. Since the problem is wide-scale and pervasive but also systematic, our solution is to refine the tag assigned to a noun wherever it is immediately followed by "of". We do this after the tagger has applied using a call to `fsgmatch` with a simple grammar which looks for a `W[P='NN']` or `W[P='NNS']` in a context where it is followed by a `W` element whose content is the string "of". When such an element is matched, the rule specifies that its `P` value is changed to `NNOF` or `NNSOF` appropriately. Thus the real input to the parser is "insertion_NNOF of the needle_NN" where the only category that results from look-up of "insertion_NNOF" is one where a subcategorised PP[of] is required. This effectively forces the parser to attach the PP[of] as a complement of the noun and prevents it from considering any other potential parse options.

The previous example constitutes a linguistically motivated, relatively high-level, intervention prior to parsing but we also use LT TTT to perform a number of low-level transformations of the input string which simplify the task of the parser. For example, at this early stage of grammar development, we wish to ignore parenthetical material (though in future we do intend to accommodate it), and we can therefore use LT TTT to either remove parenthesised strings or to mark-up them up in some way so that the parser need not try to analyse them. Our current choice is to remove the contents but keep a marker of where the parenthetical was located. Thus we convert the string "patients presenting to the emergency department (ED) are routinely admitted to intensive care" to this: "patients_NNS presenting_VBG to the emergency_NN department_NN ()_PAR are routinely_RB admitted_VBN to intensive_JJ care_NN", where the word_tag pair "()_PAR" has a lexical entry similar to a comma. Since parentheticals usually occur at major phrase boundaries, retaining a record of its location can prevent the parser from considering analyses which do not have a major boundary at that location.

6. Visualisation

One of the benefits of using XML to annotate data is that it is comparatively simple to use a browser to visualise the annotations. Thus, although heavily annotated data can be hard to view in its raw XML form, there are a number of

different ways to convert it so that it can be easily viewed. For those with up-to-date browsers, XML documents can be viewed directly and the user can control how different elements are rendered using style-sheet commands. Thus the same document may be viewed using different style-sheets in order to highlight or suppress particular parts of the mark-up.

For older browsers, XML documents can be converted to HTML, again with appropriate style-sheet commands. In our work we have used James Clark's XT program (Clark, 1999) as well as the LTG's `xmlperl` program to convert XML documents to HTML with XSL or CSS style-sheets. As a simple example, if we have used LT TTT to recognise and mark up dates as described in Section 1, then we can convert the resulting document to HTML where all mark up is discarded except for paragraphs which become <P> elements and TIMEX elements which are converted to <SPAN CLASS='DATE'> or <SPAN CLASS='TIME'>.

```
<HTML><HEAD>
<STYLE>
SPAN.DATE {background:pink}
SPAN.TIME {background:green}
</STYLE></HEAD>
<BODY>
<P>
<SPAN CLASS='DATE'>April 20</SPAN>: the
government collapsed <SPAN CLASS='TIME'>
this afternoon</SPAN>.
</P></BODY></HTML>
```

Notice that the conversion process has explicitly added CSS commands in the preamble to make the date elements be highlighted in pink and the time elements in green. Thus one can view large bodies of data in a browser and have specific parts rendered in a way which makes them easily visible.

Acknowledgements

The work reported in this paper was supported in part by grant GR/L21952 (Text Tokenisation Tool) from the Engineering and Physical Sciences Research Council, UK. The corpus preparation work described in Section 5 is supported by grant R00023777 (Data Intensive Semantics and Pragmatics) from the Economic and Social Research Council, UK.

7. References

Briscoe, Ted and John Carroll, 1993. Generalised probabilistic LR parsing of natural language (corpora) with unification grammars. *Computational Linguistics*, 19(1):25–60.

Carroll, John, Ted Briscoe, and Claire Grover, 1991. A development environment for large natural language grammars. Technical Report 233, Computer Laboratory, University of Cambridge.

Chinchor, Nancy A., 1998. Overview of MUC-7/MET-2. In *Seventh Message Understanding Conference (MUC–7): Proceedings of a Conference held in Fairfax, Virginia, 29 April–1 May, 1998*. `http://www.muc.saic.com/proceedings/muc_7_toc.html`.

Clark, James, 1999. XT Version 19991105. `http://www.jclark.com/xml/xt.html`.

Gazdar, Gerald, Ewan Klein, Geoff Pullum, and Ivan Sag, 1985. *Generalized Phrase Structure Grammar*. London: Basil Blackwell.

Grover, Claire, John Carroll, and Ted Briscoe, 1993. The Alvey Natural Language Tools grammar (4th release). Technical Report 284, Computer Laboratory, University of Cambridge.

Grover, Claire, Andrei Mikheev, and Colin Matheson, 1999. LT TTT version 1.0: text tokenisation software. `http://www.ltg.ed.ac.uk/software/ttt/`.

Hersh, William, Chris Buckley, TJ Leone, and David Hickam, 1994. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In W. Bruce Croft and C. J. van Rijsbergen (eds.), *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval.* Dublin, Ireland.

Matheson, Colin and Robert Dale, 1992. BibEdit Deliverable 3.1: A representation for bibliographic information.

Matheson, Colin and Robert Dale, 1993. BibEdit: A knowledge-based copy editing tool for bibliographic information. In E. S. Atwell (ed.), *Knowledge at Work in Universities: Proceedings of the Second Annual Conference of the Higher Education Funding Councils' Knowledge Based Systems Initiative*. Cambridge.

McKelvie, David, 1999. XMLPERL 1.0.4. XML processing software. `http://www.cogsci.ed.ac.uk/~dmck/xmlperl`.

Mikheev, Andrei, 1997. Automatic rule induction for unknown word guessing. *Computational Linguistics*, 23(3):405–423.

Mikheev, Andrei, 1998. Feature lattices for maximum entropy modelling. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and Proceedings of the 17th International Conference on Computational Linguistics*. Montreal, Quebec.

Mikheev, Andrei, Claire Grover, and Marc Moens, 1998. Description of the LTG system used for MUC-7. In *Seventh Message Understanding Conference (MUC–7): Proceedings of a Conference held in Fairfax, Virginia, 29 April–1 May, 1998*. `http://www.muc.saic.com/proceedings/muc_7_toc.html`.

Mikheev, Andrei, Claire Grover, and Marc Moens, 1999a. XML tools and architecture for named entity recognition. *Markup Languages: Theory and Practice*, 1(1).

Mikheev, Andrei, Marc Moens, and Claire Grover, 1999b. Named entity recognition without gazetteers. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*. Bergen.

Pollard, Carl and Ivan A. Sag, 1994. *Head-Driven Phrase Structure Grammar*. Stanford, Ca. and Chicago, Ill.: CSLI and University of Chicago Press.

Thompson, Henry S., Richard Tobin, David McKelvie, and Chris Brew, 1997. LT XML. Software API and toolkit for XML processing. `http://www.ltg.ed.ac.uk/software/`.