Preprocessing and lexicon design for parsing technical text[1]

Robert P. Futrelle, Christopher E. Dunn, Debra S. Ellis and Maurice J. Pescitelli, Jr.

Biological Knowledge Laboratory
College of Computer Science 161CN
Northeastern University
360 Huntington Avenue
Boston, MA 02115

Internet: futrelle, chris, ellisds and mjp
all @corwin.ccs.northeastern.edu
Phone: (617) 437-2076 FAX: (617) 437-5121

ABSTRACT

Technical documents with complex structures and orthography present special difficulties for current parsing technology. These include technical notation such as subscripts, superscripts and numeric and algebraic expressions as well as Greek letters, italics, small capitals, brackets and punctuation marks. Structural elements such as references to figures, tables and bibliographic items also cause problems. We first hand-code documents in Standard Generalized Markup Language (SGML) to specify the document's logical structure (paragraphs, sentences, etc.) and capture significant orthography. Next, a regular expression analyzer produced by LEX is used to tokenize the SGML text. Then a token-based phrasal lexicon is used to identify the longest token sequences in the input that represent single lexical items. This lookup is efficient because limits on lookahead are precomputed for every item. After this, the Alvey Tools parser with specialized subgrammars is used to discover items such as floating-point numbers. The product of these preprocessing stages is a text that is acceptable to a full natural language parser. This work is directed towards automating the building of knowledge bases from research articles in the field of bacterial chemotaxis, but the techniques should be of wide applicability.

1. INTRODUCTION

The Biological Knowledge Laboratory focuses on the analysis of research articles in the field of bacterial chemotaxis (Futrelle, 1989, 1990b). We are building a corpus consisting of the 1000 or so articles that make up the published record of the field since its inception in 1965. As the corpus is built we are attempting to use syntactic and semantic analysis to convert the corpus to a knowledge base. But the texts are complex -- they have a superstructure that includes title, authors, abstract, sections, paragraphs, bibliography, etc. They also contain sub- and superscripts, italics, Greek letters, formulas, and references to figures, tables, and bibliographic items. Another major component of technical documents that has been ignored is graphics, which requires its own analysis; we have a separate project devoted to graphical analysis and understanding (Futrelle, 1990a).

[1] This work was supported by the Division of Instrumentation and Resources of the National Science Foundation, grant number DIR-88-14522.

In this paper we describe procedures we have implemented and resources we have developed for preprocessing these complex documents. The preprocessing produces text which retains all important details of the original but is in a form that a conventional natural language parser can use without major modifications.

The preprocessing software runs in part under Unix (for LEX) and in part under Symbolics Genera 8.0 using their *Statice* database system for the lexicon. The Alvey Natural Language Toolkit (Briscoe, et al, 1987) is used for the subgrammar analysis. We have used Alvey on the Symbolics, Suns and on Mac II's. The systems described here are sentence-oriented, leaving to other software the task of organizing the structures above the sentence level.

Most research on natural language processing is restricted to text which does not contain complex orthography or has had it stripped away. This has prevented the application of computational linguistics to most technical documents and technical documents are a huge and important repository of knowledge. Though our contribution is primarily a technical one, it is one that is sorely needed if progress is to be made.

2. THE PROBLEMS AND THEIR SOLUTION

To appreciate the type of problems that arise in text analysis, consider the various uses of a punctuation mark, the period. In the sentence, "Bacteria swim." the item "swim." that includes the period is not a word, it is the word "swim" followed by end-sentence punctuation. On the other hand, the period in "etc." is not (necessarily) a sentence end marker. The period in "7.3", however, is an integral part of the number. The comma is normally used to mark phrases and clauses, but it is used as an integral part of the number "32,768" or the chemical name "2,6-diaminohexanoic acid" (the essential amino acid, lysine). Superscripts can play the role of an isotopic indicator, "^{3}H" for tritium, or a footnote[2].

We have found a way to deal with all of these problems. The documents are first encoded (marked up) as they are entered by a trained editor/typist using an editor which supports the Standard Generalized Markup Language (SGML) (Bryan, 1988; van Herwijnen, 1990). The complex items in the marked-up text are then broken into their constituent tokens and selectively reassembled so that every token or contiguous sequence of tokens is *resolved* in some way. The resolution of a token sequence is done by first looking for the sequence in a phrasal lexicon. If found, the sequence is replaced by its lexical item. If a token sequence is not in the lexicon, an attempt is made to parse it using specialized subgrammars. If this fails, the item is flagged for analysis by a human editor or lexicographer to see if it is an error or a new lexical item.

The word "salt" is a single token entry in the lexicon. The sequence, "sodium chloride" is a two token entry. The item "CO_2" which is represented by seven tokens is found as a single item in the lexicon. But it is not appropriate to represent most numbers in the lexicon, because they form an essentially *unbounded class*[3]. For example, the number "$3.4x10^{-8}$ " (made up of 17 tokens) is not in the lexicon. It is analyzed by a subgrammar and found to be a legally formed number in scientific notation. The number is replaced by a structure which includes the lexical item "num", a noun which the natural language parser can deal with. After preprocessing, the text is passed on to a full natural language parser for syntactic and semantic (logical form) analysis. Currently, we use the GPSG-based parser from the Alvey toolkit for both subgrammar analysis and full natural language parsing (Briscoe, et al, 1987; Ritchie, et al, 1987).

3. THE PROCESSING SEQUENCE

The processing sequence is outlined in Figure 1. Each stage can produce a file as output that can be the input to the next stage, so the analyses do not have to be synchronous. The *pre*processing stages are stages 1-6.

2 ...or a bibliographic reference, as in, "Smith found this effect earlier[7]."

3 Certain numbers such as cell strain designators or the familiar "Boeing 747" would be in the lexicon.

Stage 0: Obtain selected articles from primary biological literature, 1960-1990
Form 0: *word complex-orthographic-item word word floating-point-number punctuation ....*
Stage 1: SGML encoding (tagging) while typing in article using SGML-based editor
Form 1: *sentence-start-tag word tagged-complex-item word word tagged-number ....*
Stage 2: Tokenization using regular-expression analyzer generated by LEX
Form 2: *SGML-symbol string complex-item-token ... tokens-for-number SGML-symbol ....*
Stage 3: Lexicon lookup in token-based phrasal lexicon
Form 3: *found-item found-item found-item not-founds found-item not-founds ....*
Stage 4: Subgrammar analysis using Alvey syntactic and semantic tools
Form 4: *found-item found-item found-item analyzed-structure not-found*
Stage 5: Editor and lexicographer at the workbench resolve any remaining unknowns
Form 5: *found-item found-item found-item analyzed-structure added-to-lexicon*
Stage 6: Natural language parsing using Alvey GPSG-based tools
Form 6: *Parse trees and logical form structures*
Stage 7: Building knowledge frames . . . .

Figure 1. Schematic view of the successive stages of corpus processing. "Form *n*" lists typical items in the stream of text which result from the processing in Stage *n* and are the input to Stage *n+1*. There is not an absolutely tight correspondence between the items in successive forms in this figure, due to the complexity of the analysis. The underlined stages denote the preprocessing stages which are currently implemented and explained in some detail in this paper.

STAGE 0: Obtaining Selected Articles – In many cases, these articles are only available in bound journals. The originals are scanned for diagram entry, but the typing (with simultaneous markup) is done from photocopies when necessary.

STAGE 1: SGML Markup – Markup languages such as SGML allow us to add *markup* to a text of a document to specify its logical structure. Thus, in SGML, one would specify, using *tags*, that certain words formed a *section heading* without committing to stylistic details such as font, font size, or the positioning of the heading with respect to the margin. For example, the text that begins the subsection you are reading would be encoded in SGML as:

(1) <SS1><ST> STAGE 1: SGML Markup </ST> <P><U.S>Markup languages such as SGML add <E1>markup</E1> to a text of a document to specify its structure. </U.S>

In (1) the SGML *tags* enclosed by braces have the following meanings:

(1a)	<SS1>	=	subsection start-tag
	<ST>	=	section title start-tag
	</ST>	=	end of section title
	<P>	=	paragraph start tag
	<E1>	=	emphasis start tag
	</E1>	=	emphasis end
	<U.S>	=	sentence start tag
	</U.S>	=	sentence end tag

The SGML encoding of (1) is, in turn,

(2) &lt;SS1&gt;&lt;ST&gt;STAGE1: SGML Markup – ... &lt;&sol;U.S&gt;

which shows that we can satisfy *Becker's Criterion* (Becker, 1975) that states that any technique that claims to be useful and generally applicable should be able to analyze the very text which explains the technique!

In (2) items such as "&lt;" are SGML *entities*; this one denoting the reserved character, "<" (less than). The tags used here are drawn from the American Association of Publishers' (AAP) set, the Electronic Manuscript Standard (EMS), with the addition of our own user-defined tags such as the sentence tags, <U.S> and </U.S>. SGML is an ISO standard (#8879). SGML specifies a system in which tags and entities can be defined and used so that an arbitrarily complex text can be translated to a standard form which uses only the ASCII character set so it can be disseminated widely and dealt with uniformly by a variety of systems.

The encoding (markup) of the text is done using an SGML editor that makes the process efficient and checks that the text complies with our SGML syntax specifications, e.g., no sentence-start tag can be entered until the previous sentence-end tag has been entered. The particular system we use is Author/Editor (Softquad, Toronto, Canada) running on Mac II's .

The example sentence – Here is the example sentence we will use to illustrate our preprocessing strategy. It is first presented as it might appear in a research article source, but laid out for easy comparison with the SGML form which follows:

(3a) Cells were suspended in medium containing

(3b) 3.05x10^{-2} μM

(3c) L-[*methyl* -^{3}H]-methionine,

(3d) α-methylaspartate

(3e) and AIBU^8.

Here is the SGML encoding of the example sentence:

(4a) <U.S>Cells were suspended in medium containing

(4b) 3.05&times;10<SUP> &minus;2</SUP>&micro;M

(4c) <SCP>L</SCP>-[<IT>methyl</IT>-<SUP>3</SUP>H]-methionine,

(4d) <GK>a</GK>-methylaspartate

(4e) and AIBU <RB>8</RB>.</U.S>

The "&micro;" entity stands for the Greek letter mu. "<SCP>" indicates small caps, "<IT>" indicates italics and "<RB>" is a bibliographic reference tag. Note that small caps and italics are encoded because they are standard typographical conventions used in chemical names; otherwise the *appearance* of items is not encoded.

STAGE 2: Tokenization – We use an analyzer generated by LEX (Aho, Sethi and Ullman 1986) to tokenize the input. It uses a regular expression grammar to identify the primitive elements of the SGML encoded text. The six classes of tokens produced by this stage are shown in Table 1. Note that "token" as we use it here includes a parenthesized pair (for numbers), not just a contiguous sequence of non-blank characters.

Table 1. The input and output forms for the tokenization stage, Stage 2.

Input Class	Output Format	Example Output
ASCII text strings	*string*	"Cells"
numbers	(num *string*)	(num "05")
special characters	(*string*)	(".") (",") ("(")
SGML tag	*symbol*	<U.S>
SGML entity	*symbol*	|&micro;|
no-white-space	nws	nws

For each class, the original ASCII representation has been preserved, either by including the string itself or using a Lisp symbol whose print representation is the ASCII representation. As an example, the outputs from tokenizing (4a) and (4b) are the 7 token sequence (5a) and the 20 token sequence (5b):

(5a)<U.S> "Cells" "were" "suspended" "in"
"medium" "containing"

(5b) (num "3") nws (".") nws (num "05") nws
|&times;| nws (num "10") nws
<SUP> nws |&minus;| nws
(num "2") nws </SUP> |&micro;|
nws "M"

The white spaces in the original text have been complemented to yield the *nws* symbol to indicate that the tokenized elements were originally abutted. This is necessary for disambiguation of complex sequences, and it makes normal prose easier to read at this stage.

Stage 3: Lexicon Lookup – At this point, a lexicon is consulted for each sequence of tokens contained in a title, section heading, sentence, etc. For our example, the token sequence generated from the full sentence (4) is handed to the lexicon lookup routine as the 73 token list,

(6) ("Cells" "were" "suspended" ... <GK>
nws "a" nws </GK> nws ("-") nws
"methylaspartate" ... (num "8") nws
</RB> nws ("."))

(notice our ellipsis). The lexicon lookup stage attempts to match sequences of tokens from the input to items found in the lexicon. The lexicon is an extended phrasal lexicon, in which each lexical entry is a sequence of one or more tokens. Five typical lexical items are

"cells"

"sodium chloride"

"<GK>a</GK>-methylaspartate"

"<GK>"

"."

Note that in the lexicon, the *nws* (no-white-space) tokens are removed by concatenation for both storage and lookup. A lexical item *L* (one or more tokens) is a *prefix* if there are longer items in the lexicon (more tokens) with the same initial items as *L*. The first token of all items in the lexicon is listed as a separate entry. But some of these and some multiple token entries never function as independent *stand-alone* items and are noted as such in the lexicon. For example the SGML tag tokens "<GK>" and "<IT>"indicating that Greek and italicized characters follow never function as separate items.

To efficiently and reliably find multi-token items, certain information is precomputed and stored in the lexicon. For example, the items "sodium", "chloride", "sodium chloride", "sodium bromide" "sodium iodide" might all appear in the lexicon. When "sodium chloride" appears in the source text, it is that two-item entry that we want identified, not the two separate words. To assure that this happens the prefix list ((3 2)) is computed and attached to "sodium". This says that there are 3 items of length 2 that begin with "sodium", so the next item in the source, "chloride" is attached and the two-word item is found and returned by the lexicon lookup. Prefix lists can be complex, forming trees rooted at the initial item. The prefix lists prevent the search for a single item from continuing to the end of the sentence, because they put explicit bounds on the lengths of all items that could possibly match, given any prefix.

The output from the lexicon lookup stage for (6) is the list

(7a) ("Cells" "were" "suspended" "in"
"medium" "containing"

(7b) (?? ((num "3") nws))

"."

(?? (nws (num "05") nws |&times;|
nws (num "10") nws <SUP> nws
|&minus;| nws (num "2") nws
</SUP>))

"&micro;M"

(7c) "<SCP>L</SCP>-[<IT>methyl</IT>-<SUP>3</SUP>H]-methionine" ","

(7d) "<GK>a</GK>-methylaspartate"

(7e) "and" "AIBU"

(?? (<RB> nws (num "8") nws </RB> nws))

".")

There are three unknown item sequences here, shown broken out in (7b) and (7e) as (??....) forms. The first two are parts of the number 3.05x10^{-2}. The third is a bibliographic reference. The "." in the number in (7b) and the "." at the end of the sentence in (7e) are recognized since "." is a stand-alone item. <SUP> it is a prefix for entries such as "<SUP>3</SUP>H-ethanol" but it is not stand-alone, so it is included in the unknown in (7b). Note that the strings which are the lexicon identifiers for complex items such as the chemical name in (7d) retain their original SGML markup, without the no-white-space symbols introduced by tokenization. In an interactive system, these items could be presented on a screen by interpreting the markup according to a style specification and producing the indicated orthography, e.g., α-methylaspartate.

Stage 4: Subgrammar analysis – The reason that the three unknown items were unrecognized in the previous step is that they were parts of lexical items that belong to two of the unbounded classes of lexemes. The job of the subgrammar is to analyze this type of unknown which can include numbers, number ranges, simple ratios, references and page numbers. Each class has an associated structure for representing its instances. In our previous example we had two unknown token sequences and one lexical item, which when taken together correspond to the number 3.05x10^{-2}:

(8) (?? ((num "3") nws))

"."

(?? (nws (num "05") nws |&times;| nws (num "10") nws <SUP> nws |&minus;| nws (num "2") nws </SUP>))

We have written a context-free grammar to recognize this token stream as a number in scientific notation and place a structure in the output stream of the general form

(9) ("$num$" *SGML-string Lisp-num-form*)

For our example (8) this would result in:

(10) ("$num$"
"3.05&times;10<SUP>&minus;
2</SUP>" 3.05E-2)

The number structure consists of three fields. The first, "$num$", is a lexical item, the noun which represents all numbers. The parser for doing the later syntactic analysis of this sentence will access the feature-value list associated this noun. The second field contains the SGML encoding of the number. This can be used for displaying the number on the screen. The third field contains a Lisp-readable form of the number.

Another structure recognized by subgrammar analysis is the bibliographic reference, (7e). The structure produced by the analysis has the form:

(11) ("$bibref$" *SGML-string List-of-contents*)

When the token sequence from (7e) is recursively analyzed, the result is

(12) ("$bibref$"
"<RB>8</RB>"
(("$num$" "8" 8)))

In this example, the bibliographic reference structure contains a number structure. In general, any sequence of lexical items, structures and unrecognized token streams can be placed in the *List-of-contents* for bibliographic references.

Subgrammar analysis of expressions such as (8) involves first creating a stream without the "??" tokens and without the actual integers ("3", "05", "10" and "2") and with the "ordinary" words replaced by simple placeholders, e.g., "$word$". Critical elements such as nws, <SUP> | &minus; |, etc. are retained.

Once this simplified stream is available, the parse is done according to the subgrammar specialized for numbers, bibliographic references, etc. But the output of the subgrammar analysis must produce a new stream which includes forms such as in (10) and (12) as well as all of the original words. To do this we take advantage of the compositional semantics built into the Alvey parser. The semantic attachment facilities in Alvey allow references to daughter nodes by number and the inclusion of simple lambda forms. But in addition, arbitrary lisp forms can be included. We define semantic rules with lisp forms included. The Alvey semantics then works compositionally by walking up the parse tree. This allows the semantic interpretation to generate the Common Lisp source code for a translator of the original stream, e.g., of (7a-e). When this translator is applied to the original stream, all "??" items which parse are replaced by forms such as (10) and (12) and all words such as "Cells" "were", etc. are simply copied to the output. All "??" items that remain are either ill-formed or are items not yet in the lexicon. Note that a separate translator is built for each sentence. But the construction is simple and deterministic and therefore rapid. Lisp's ability to treat code as data is what we're exploiting here.

The syntactic role of some of the special forms found by the subgrammar is subtle. Thus, in

(13) "This was discovered by Smith when he was working at the MBL^{19}."

the bibliographic reference does not act like any familiar syntactic constituent. But in the following form the reference functions as a noun,

(14) "Commonsense knowledge is discussed in (Davis, 1990)."

In the full natural language parsing (Stage 6) there will be additional categories and grammar rules to allow such structures to be treated properly.

When the translator generated by the semantic interpretation of the subgrammar parse is applied to (7a-e), the final form which results is

```
(15a)  ( "Cells" "were" "suspended""in"
         "medium" "containing"

(15b)  ("$num$"
           "3.05&times;10<SUP>minus;2
           </SUP>" 3.05E-2)
         "&micro;M"

(15c)  "<SCP>L</SCP>-[<IT>methyl</IT>-
         <SUP>3</SUP>H]-methionine" ","

(15d)  "<GK>a</GK>-methylaspartate"

(15e)  "and" "AIBU"
         ("$bibref$" "<RB>8</RB>"
            (("$num$"  "8"  8))) "."  )
```

This preserves all of the details of the original text. Every form is an item or contains an item that can be found in the lexicon and one that will allow a proper screen display (cf. (16) below). Lisp forms of numbers and citation information are also available.

The subgrammars are simple and deterministic so the parses are fast compared to the later full natural language parses.

Stage 5: The Lexicographer's Workbench
Natural language parsing cannot be done until all items are resolved by the lexicon, so unknown items are passed on to the editor and the lexicographer (humans). Errors in the original source and errors in our own re-entry can be caught at this stage. What remain are items that need to be added to the lexicon. These additions are made using the Lexicographer's Workbench which is currently under development. In the Workbench a collection of analytical tools and heuristic procedures are used to tentatively classify new items which are then presented to the lexicographer for simple approval or more rarely for special treatment. Morphological

analysis is useful, e.g., certain classes of enzyme names have the suffixes "tase" or "ase" as in "phosphatase" or "nuclease". This means that new words can be analyzed and suggestions made as to their classification. Alvey has a sophisticated morphological analysis package which we are experimenting with in which the rules are user definable (Ritchie, et al 1987).

One difficult task is the identification of new *phrasal* items, a difficulty emphasized by Amsler (Amsler, 1989). For example, consider the case in which "sodium", "chloride", "bromide" and "sodium chloride" are in the lexicon but "sodium bromide" is not. If "sodium bromide" appeared in the input it would not even be flagged as an unknown. Nevertheless, we would want the Workbench to be provided with the heuristic that chemical name sequences are most likely chemical names themselves. Thus the workbench would make the decision itself and insert "sodium bromide" in the lexicon with the proper feature/value specs. This decision would, as all others, be subject to review by the lexicographer or application field specialist.

Stage 6: Natural language parsing – When the lexical items are extracted from (15), the result is

(16a) ("Cells" "were" ""suspended" "in" "medium" "containing"

(16b) "$num$" "&micro;M"

(16c) "<SCP>L</SCP>-[<IT>methyl</IT>-<SUP>3</SUP>H]-methionine" ","

(16d) "<GK>a</GK>-methylaspartate"

(16e) "and" "AIBU" "$bibref$" ".")

This is the input to the natural language parser. The grammar furnished with the Alvey tools is large and covers a wide variety of constructions. Nevertheless, it will take further extensions to get acceptable coverage of the scientific prose in our corpus. This is work in progress. A semantics for this large grammar is under development (C. Grover, personal communication). In addition, a more efficient, LR(1) parser is being built to improve the performance over the chart parser currently available in the Alvey Toolkit (J. Carroll, personal communication).

Stage 7: Building Knowledge Frames – We have studied papers in our corpus in an effort to identify all of the major semantic constructions. One type deals with the experimental details themselves such as the techniques used and the results seen. The other deals with scientific argumentation – how models are used to suggest experiments and how results reinforce or weaken various hypotheses that might explain them. Our goal is to design knowledge frames for the different semantic structures we have found. Then the logical forms produced by parsing would be used as input to a system which generates instances of the appropriate knowledge frames representing the sentences. (This is also work in progress.) Furthermore, these knowledge frames can be connected together into superstructures representing coherent arguments for or against a given proposition. Taken together, these frame instances and their connecting frames compose the *knowledge base* which would underlie our "Scientist's Assistant" system, a system for answering both general and specific queries about the contents and arguments that are to be found in our corpus.

4. DISCUSSION

Because of the complexities of technical text notation and the availability of a comprehensive standard, we decided to use SGML for text markup. Then we designed a token-based phrasal lexicon for resolving the complex items generated by the markup. This lexicon is robust because it handles everything from simple words to complex multi-word chemical names containing Greek letters, commas, superscripts and more. In addition, our subgrammar analysis handles unbounded class items that cannot be accommodated in the lexicon such as numbers in scientific notation and bibliographic references.

The work closest to ours is the preprocessing done for the LOB corpus (Booth, 1987). Unfortunately, the SGML standard was not available to that project at the time, so they had to invent their own orthographic coding schemes and a pre-editing phase similar to ours to break the text into taggable units. There are many differences between the projects. One of these is in the design of the lexicon. The LOB group decided to develop a compact lexicon which includes only the base forms. Possessives or contracted forms such as "Smith's" or "it's" are not included. Because secondary storage is rapidly becoming less expensive and because modern database and file structure designs allow very rapid access to large lexicons we have opted for a very "flat" lexicon in which *every* variant form encountered in the corpus is stored as a separate entry. This includes capitalized words appearing at the beginning of sentences, etc. We add the variants of the base forms to the lexicon *only* when they are found in our corpus. Our own statistical analysis of large corpora such as the Brown Corpus show that the inclusion of these variant forms will probably add no more than 50% to the lexicon size over a lexicon that has only the base forms.

If we had only included base forms then other difficulties would crop up in attempting to map between found entities and the base forms. We avoid these difficulties by including the variant forms and flagging them to indicate their usage restrictions. We would flag "There" as a form only expected as a sentence initial (and fully equivalent to "there") whereas "DNA" would only be expected in fully capitalized form.

Another major activity in text encoding is the Text Encoding Initiative or TEI (Sperberg-McQueen and Burnard, 1990). They have been focusing on text in the humanities so they have been concerned with a different set of problems such as encoding verse, stage directions, foreign language quotations, etc. Neither the TEI not the LOB groups seemed to have directly faced the issues of how to interface the marked up text with the available parsing technology as we have.

SGML allows the user to design their own set of tags, entities and rules so we had to make some design decisions. Our design is constructed pragmatically to make it usable by an editor/typist who is not a scientist. For instance, we have used a special tag "<RB>" for a bibliographic reference which might be represented by a superscript or by the conventional "(Shepard, 1978)". And we have opted to use the simple superscript tag "<SUP>" for both algebraic exponents as in "3.05×10^{-2}" and isotope indicators as in "^{3}H". The subgrammar and lexicon lookup, respectively, resolve these latter two items. This allows the typist to encode source text primarily on the basis of its appearance, rather than its semantic (scientific) content.

We are constantly asked why we do not use OCR techniques (optical character recognition) or go directly to publishers for electronic versions of the papers in our corpus. Again, these are pragmatic decisions, peculiar to this point in time. Because OCR error rates are still relatively high, especially for technical text, and because OCR systems do little or no markup, we can produce accurate transcriptions and markup more cost-effectively by having a skilled typist/editor rekey the text. Most of our corpus (covering 30 years) does not exist anywhere in electronic form, and the wide variety of proprietary schemes used by printing firms for electronic typesetting is a nightmare to untangle.

In the future, technical word processing systems will be developed that will allow scientist authors to enter their text with the proper logical tagging but without the system obtruding on their work. The systems we are developing will be able to take advantage of such electronic documents as they become available.

Many authors have argued cogently and at length that multi-word items, idioms, punctuation and other complexities of real text require a comprehensive approach (Becker, 1975; Besemer and Jacobs, 1987; Amsler, 1989; Nunberg, 1988, 1990). The methods described here can serve as a foundation for any comprehensive system that must deal with the lexical, syntactic and semantic aspects of real-world technical text.

ACKNOWLEDGEMENTS

We thank John Carroll and Claire Grover for discussions of the Alvey tools, including the semantic component.

REFERENCES

Aho, A.; Sethi, R. and Ullman, J. 1986. Compilers; Principles, Techniques, and Tools. Addison-Wesley Publishing Company, Inc., Reading, MA.

Amsler, Robert A. 1989. Research Toward the Development of a Lexical Knowledge Base for Natural Language Processing. *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval.* Cambridge, MA : 242-249.

Becker, J.D. 1975. The Phrasal Lexicon. In *Proceedings Interdisciplinary Workshop on Theoretical Issues in Natural Language Processing.* Cambridge MA : 70 - 73.

Besemer, David J. and Jacobs, Paul S. 1987. FLUSH: A Flexible Lexicon Design. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics..* Stanford University, Stanford, CA : 186 - 192.

Booth, Barbara 1987. Text input and pre-processing: Dealing with orthographic form of texts. In *The Computational Analysis of English. A Corpus-Based Approach.* (Longman, London).

Briscoe, E.; Grover, C.; Boguraev, B. and Carroll, J. 1987 A Formalism and Environment for the Development of a Large Grammar of English. *Proceedings of the 10th International Joint Conference on Artificial Intelligence,.* Milan, Italy: 703-708.

Bryan, M. 1988. *SGML: An Author's Guide to the Standard Generalized Markup Language.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

Futrelle, R. P. 1989. *An Introduction to the Biological Knowledge Laboratory.* Technical Report NU-CCS-89-15. College of Computer Science, Northeastern University.

Futrelle, R.P. 1990a. Strategies for Diagram Understanding Object/Spatial Data Structures, Animate Vision, and Generalized Equivalence. *Proceedings of the 10th International Conference on Pattern Recognition.* Atlantic City, NJ: 403-408.

Futrelle, R. P. 1990b *Current Activities in the Biological Knowledge Laboratory (BKL).* Technical Report NU-CCS-90-20. College of Computer Science, Northeastern University.

Nunberg, Goeffrey 1988, 1990. *The Linguistics of Punctuation.* Technical Report P88-00142. XEROX System Sciences Laboratory, Palo Alto Research Center, Pal Alto, CA (U. Chicago Press, 1990, to appear).

Ritchie, Graeme D.; Pulman, Stephen G.; Black, Alan W. and Russell, Graham J. 1987. A Computational Framework for Lexical Description. *Computational Linguistics,* Vol. 13, No. 3-4: 290-307.

Sperberg-McQueen, C. M. and Burnard, Lou, editors. 1990. *Guidelines For the Encoding and Interchange of Machine-Readable Texts.* Document Number: TEI P1. Draft: Version 1.0. The Association for Computers and the Humanities; The Association for Computational Linguistics; The Association for Literary and Linguistic Computing.

van Herwijnen, Eric 1990. *Practical SGML.* Kluwer Academic Publishers, Dordrecht, The Netherlands.