What is a word, What is a sentence? Problems of Tokenization

Gregory Grefenstette, Pasi Tapanainen
Rank Xerox Research Centre
Grenoble Laboratory
38240 Meylan, France
grefen@xerox.fr, tapanai@xerox.fr

Abstract

Any linguistic treatment of freely occurring text must provide an answer to what is considered as a token. In artificial languages, the definition of what is considered as a token can be precisely and unambiguously defined. Natural languages, on the other hand, display such a rich variety that there are many ways to decide upon what will be considered as a unit for a computational approach to text. Here we will discuss tokenization as a problem for computational lexicography. Our discussion will cover the aspects of what is usually considered preprocessing of text in order to prepare it for some automated treatment. We present the roles of tokenization, methods of tokenizing, grammars for recognizing acronyms, abbreviations, and regular expressions such as numbers and dates. We present the problems encountered and discuss the effects of seemingly innocent choices.

1 Introduction

The linguistic exploitation of naturally occurring text can be seen as a progression of transformations of the original text. The original text is a sequence of characters. Before any syntactic analysis of the corpus is performed, two transformations usually take place. Sentences must be isolated since most grammars describe sentences. And, in order for sentences to be isolated, words must be isolated from the original stream of characters. The isolation of word-like units from a text is called tokenization. The results of this tokenization are two types of tokens: one type corresponding to units whose character structure is recognizable, such as punctuation, numbers, dates, etc.; the other type being units which will undergo a morphological analysis.

In linguistic textbooks tokenization is quickly dispatched as a relatively uninteresting preprocessing step performed before linguistic analysis is undertaken. In reality, tokenization is a non-trivial problem. Confronted with large corpora of raw text, the computational lexicographer must come to grips with the transformations presented schematically in Figure 1 and make the difficult choices, choices whose repercussions are sometimes only felt long after.

In this paper we will discuss the choices that must be made, how they can be made, when they should be made and their possible effects on subsequent linguistic treatment.

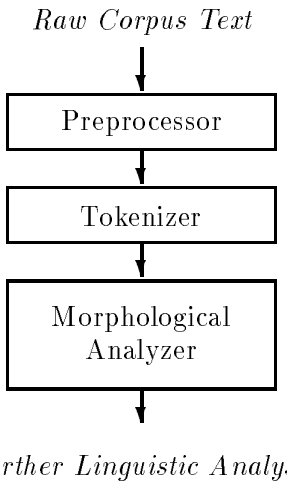

Figure 1: Text Transformations Before Linguistic Analysis.

2 Preprocessing

We will consider throughout that we are dealing with a text in electronic form as a sequence of characters, rather than a scanned image of text. Electronic text is readily available these days, in increasing numbers, usually produced as a by-product of typesetting. Such text often contains extra whitespace and a number of mark-ups that indicate font-changes, text subdivisions, special characters, and a hundred other things. Although such indications carry meaning — they are there to help the reader understand the text — they are usually filtered out from the text in a preprocessing stage before any linguistic processing, or even before tokenization begins.

Since little normalization exists in typesetting codes, we will not discuss the matter further, except to provide a method of eliminating SGML-type code from a running text[1]. Unix-based workstations furnish a general-purpose character stream scanner called *lex* or *flex*. This scanner permits the definition of actions to be taken when certain regular expressions are matched in the input text. Figure 2 provides a simple *lex* program [2] which deletes SGML markings from an

[1] A public domain SGML parser called *SGMLS* is available from the anonymous *ftp* site ifi.uio.no. (129.240.64.2) in the directory /pub/SGML/SGMLS. This parser allows much finer handling of SGML codes.

[2] The notation for the regular grammars shown here are the following:
. matches any character except newline.
∧ matches the beginning of a line.
$ matches the end of a line.
\n matches the newline character.
[abc...] character class, matches any of the characters abc...
[∧abc...] negated character class, matches any character except abc... and newline.
r1|r2 alternation: matches either r1 or r2.
r1r2 concatenation: matches r1, and then r2.
r+ matches one or more r's.
r* matches zero or more r's.
r? matches zero or one r's.
(r) grouping: matches r.

```
/*  Call this file StripSGML.lx, and then run:
flex -8 -CF StripSGML.lx; gcc -o StripSGML lex.yy.c -lfl -s
To pass this simple filter over a text file called toto,  run:
StripSGML < toto                                                    */
%%
"<"[^\n<>]+">"  ;
.               ECHO;
[\n]            ECHO;
%%
```

Figure 2: Flex program for filtering out SGML markings.

```
/*  Call this file dehyphen.lx, and then run:
flex -8 -CF dehyphen.lx; gcc -o dehyphen lex.yy.c -lfl -s
    To pass this simple filter over a text file called toto,  run:
dehyphen < toto                                                     */
%%
[a-z]-[ \t]*\n[ \t]*    { printf("%c",yytext[0]); }
%%
```

Figure 3: Flex program for dehyphenating a text.

input file.

Not only do some things have to be filtered out of marked-up text, some things have to rejoined. The most common case that appears in raw text is hyphenation at right margins. Since this hyphenation is usually only circumstantial, related to the width of the page and not to the meaning of the text, one might easily consider eliminating it from text files that employ it. The short *lex* program of Figure 3 eliminates a trailing hyphen from a text and rejoins the hyphenated word to its second half on the next line. The regular expression that the filter recognizes is a lower-case letter, followed by a hyphen, then any number of tabs or spaces, followed by a newline character and more spaces. Only the alphabetic character is retained and printed out by the filter. All other characters in the file pass through unchanged.

Of course, introducing hyphenation into a text during typesetting can produce lines ending in a hyphen not because the word was split there, but because a naturally occurring hyphen happened by chance to appear where the word would be split. Suppose that the word *small-town* was split at the end of line by the typesetting, then this filter would return the string *smalltown* as one token. In order to test just how often this might happen in reality, we took the Brown corpus (Francis and Kucera, 1982), a corpus whose tokenization was hand corrected, and ran it through a typesetting program (*nroff*) which introduced end-line hyphenations. The Brown corpus contains about 1 million words. Typesetting the untokenized Brown corpus produces 101860 lines of formatted text, of which 12473 (12%) ended in a letter plus hyphen. Joining these lines using the filter given in Figure 3, produced 11858 correct dehyphenations and 615

errors (4.9%), i.e. words which did not appear in the original text. Examples of erroneously joined words are *ring-aroundthe-rosie, rockcarved, rocketbombs, rockribbed, roleexperimentation, rookie-of-theyear, satincovered, sciencefiction.* This experiment gives a taste of the type of choices that must be made during tokenization. Here, if one had access to a dictionary and morphological package at this stage, one could test each of the 12473 cases by analyzing the constituent parts and making more informed decisions, but such a mechanism is already rather sophisticated, and its construction is rarely considered for such a preliminary stage of linguistic treatment. One may consider the 615 errors (out of 1 million words) as so many unknown words to be treated at some later stage, or just accept them as noise in the system.

3 Roles of Tokenization

Once the input text of the corpus is preprocessed, we have a string of characters corresponding to what the linguistic processors will consider as the text. At one stage in this linguistic processing the elements of the text will be considered as belonging to a certain syntactic class. For example, the string *dog* will be considered as a SINGULAR-NOUN. In order for classes to be assigned to strings, the original text, which can be considered as one long string, has to be divided into units which will be recognized as members of a class. One traditional role of tokenization is the recognition of these units.

The other traditional role of tokenization is the recognition of sentence boundaries, since most linguistic analyzers consider the sentence as their unit of treatment. We will consider this traditional view here, demonstrate how it can be implemented, and show its limitations in handling certain ambiguous cases of word and sentence boundaries.

4 What is a word, What is a sentence?

Isolating word and sentence boundaries involves resolving the use of ambiguous punctuation. The second role of tokenization is, then, the one which must be attacked first. Some structurally recognizable tokens contain ambiguous punctuation, such as numbers, alphanumeric references (e.g. *T-1-AB.1.2*), dates (e.g. *02/02/94*), acronyms (e.g. *AT&T*), punctuation, and abbreviations (e.g. *m.p.h.*). Some of these classes can be recognized via regular expression grammars which predict the structure of the tokens as will be illustrated below. Once these units are recognized the only uses of separators are non-ambiguous, and they can thus be used surely to delimit words and sentences.

Sentences end with punctuation. The exclamation point and the question mark are almost always unambiguous examples of such punctuation. The semicolon is sometimes a separator of list elements, and sometimes a sentence separator. But the most prevalent of ambiguous separators is the period which is extremely ambiguous. It is not at all trivial to decide when it is a full-stop, a part of an abbreviation, or both. In the Brown corpus, there are 52511 sentences ended by a full stop (period or question mark) and 3569 (about 1 in 15) contain at least one non-terminal period. If one were to consider every period as a full stop, then 93,20% of the original 52511 sentences would be correctly recognized. In some cases, one might consider this most simple of heuristics as sufficient. In the following sections, we will see how this sentence recognition count can be improved by adding increasing levels of linguistic sophistication.

4.1 Ambiguous Separators in Numbers

Numbers are the least ambiguous of the structural types. Still, the structure of numbers are language specific constructions, for example the English number "123,456.78" will be written as "123 456,78" in French newspaper text.

A regular expression which recognizes the English version of numbers is

```
([0-9]+[,])*[0-9]([.][0-9]+)?
```

while a regular expression accepting the French version is

```
([0-9]+[ ])*[0-9]([,][0-9]+)
```

These expressions would overgenerate strings, outside the class of numbers, but used as recognizers they are sufficient. One rarely sees strings such as "12,45.678" in ordinary text, and even if one did one would probably want it considered as a number.

The table below gives some regular expressions for English numbers, dollar values and date-like constructions that can be incorporated into a tokenizer. Recognizing these strings eliminates some of the ambiguity of the comma and the period, since these characters are comprised in the token and are thus no longer considered as separators.

```
[0-9]+(\/[0-9]+)+                  Fractions, Dates
([+\-])?[0-9]+(\.)?[0-9]*%         Percent
([0-9]+,?)+(\.[0-9]+|[0-9]+)*      Decimal Numbers (e.g.  1,234.56)
```

Once we recognize numbers using the above expressions, not considering their included periods as full stops, only 3340 sentences are now incorrectly recognized, adding 229 sentences to the count of correct Brown sentences, still using the simple heuristic "remaining period equals full stop." This improves sentence recognition from 93.20% to 93.64%.

4.2 Abbreviations

Besides numbers, the other, most important, class of tokens incorporating the period as an element is the class of abbreviations. Lists of abbreviations can be long and, like lists of proper names, incomplete, since creation of abbreviations is a productive process. Let's consider now that we have no such lists, and see how far we can get using regular expressions to recognize abbreviations, so that their periods will not be considered as full stops. Let's consider first that any period not followed by a blank is not a full stop. Using the number recognizers given above, and this added heuristic adds 73 more correctly recognized sentences, so that now 49244 of the original 52511 sentences are now correctly recognized, raising the percentage to 93.78%.

4.2.1 Experiment: No lexicon

We want to do better than this, of course. We can find a better approach by analyzing the structure of abbreviations. Let us consider three classes of abbreviations: A single capital followed by a period, such as *A.*, *B.*, *C.*; A sequence of letter–period–letter–period's, such as *U.S.*, *i.e.*, *m.p.h*; and a capital letter followed by a sequence of consonants followed by a period, such as *Mr.*, *St.*, *Assn.*

If we insert blanks around parentheses, commas, colons, and questions marks, then 4037 such sequences are found in the Brown corpus. If we automatically consider each of these sequences as non-sentence-ending abbreviations and not as an unabbreviated word followed by a final stop, we will be right 3835 out of 4037 times. The details are given in the table below. For example, the third class of regular-expression defined abbreviations (a word beginning in uppercase without any following vowels, such as "Mr.") matches actual abbreviations 1938 times, commits 44 errors recognizing strings as abbreviations that should not be (for example, "Ash."), and recognizes a real abbreviation 26 times that is also a sentence terminator. In this case, the sentence ending period is absorbed in the abbreviation-ending period (Nunberg, 1990).

regular expression	Correct	Errors	Full Stop
[A-Za-z]\.	1327	52	14
[A-Za-z]\.([A-Za-z0-9]\.)+	570	0	66
[A-Z][bcdfghj-np-tvxz]+\.	1938	44	26
Totals	3835	96	106

This means that, without consulting a lexicon, but only by using the structure of the words we will correctly recognize 3935 of the non-numeric token-ending periods as part of an abbreviation (out of 4951 (330 unique) true Brown abbreviations). We will introduce 96 errors by recognizing true full stops as false abbreviations, and and another 106 by correctly recognizing abbreviations but not realizing that they should also be full stops. The number of original Brown sentences that will be correctly recognized using the number recognizers above and this abbreviation recognition scheme is now 51282, or 97.66%. 825 sentences still contain some type of abbreviation not recognized by the above expressions, and 404 sentences will have been incorrectly joined since the final stop is not recognized as such in 202 cases.

The abbreviations in Brown that do not match the above regular expressions are the following, listed in order of decreasing frequency:

> Month-Names Sen. Gen. Rev. Gov. U.S.-State-Abbreviations fig. Rep. Ave. Corp. figs. Figs. 24-hr. lbs. Capt. yrs. dia. Stat. Ref. Prof. Atty. 6-hr. sec. eqn. chap. Messrs. Dist. Dept. ex-Mrs. Vol. Tech. Supt. Rte. Reps. Prop. Mmes. 8-oz. viz. var. seq. prop. pro-U.N.F.P. nos. mos. min. mil. mEq. ex-Gov. eqns. dept. Yok. USN. Ter. Shak. Sha. Sens. SS. Ry. Rul. Presbyterian-St. P.-T.A. Msec. McN. Maj. Lond. Jas. Grev. Gre. Cir. Cal. Brig. Aubr. 42-degrees-F. 400-lb. 400-kc. 36-in. 3-hp. 3-by-6-ft. 29-Oct. 27-in. 25-ft. 24-in. 160-ml. 15,500-lb. 12-oz. 100-million-lb. 10-yr. 1.0-mg. 0.5-mv./m. 0.1-mv./m. 0.080-in. 0.025-in.

4.2.2 Experiment: No lexicon, Corpus filter

In order to reduce this list of non-recognized abbreviations without referencing a lexicon, you can use the corpus itself as a filter for identifying abbreviations. Let us define as a *likely abbreviation* any string of letters terminated by a period and followed by either a comma or semi-colon, a question mark, a lower-case letter, or a number, or followed by a word beginning with a capital letter and ending in a period.

Using this definition of likely abbreviations matches 239 of the 330 unique abbreviations in the Brown corpus, but introduces a large number of false positives such as

based become behavior better board box break bull's-eye ...

which are words that happen to end sentences that are followed by another sentence beginning with a number.

We can apply the corpus itself as a filter by eliminating from the list of likely abbreviations those strings that appear without terminal periods in the corpus. This drastically reduces the collection of likely abbreviations to 231, but still misrecognizes strings such as "furlongs," "light-hearted," and "rev'rend" as abbreviations.

Likely Abbreviation	Total (unique)	Correct (unique)	Incorrect (unique)
[A-Za-z][^]*\.([,?;] [a-z0-9])	947	239	718
not appearing without period	231	197	34

We can apply the corpus itself as a filter by eliminating from the list of likely abbreviations those strings that appear without terminal periods in the corpus. This drastically reduces the collection of likely abbreviations to 231, but still misrecognizes strings such as "furlongs," "light-hearted," and "rev'rend" as abbreviations.

Likely Abbreviation	Total (unique)	Correct (unique)	Incorrect (unique)
[A-Za-z][^]*\.([,?;] [a-z0-9])	947	239	718
not appearing without period	231	197	34

When we use the corpus as a filter for accepting the 231 candidates as likely abbreviations and accepting all structures with internal periods (i. e., not ending in a period) or ending in a period and of the form "[A-Z]." or "[A-Za-z](.[A-Za-z0-9])+" as non-terminal abbreviations, then 337 sentences are incorrectly divided. Another 266 sentence ends are incorrectly identified as sentence internal abbreviations, mistakenly joining 532 sentences, meaning that 51642 of the 52511 original Brown sentences are now correctly recognized. This gives us a 98.35% recognition rate, after using the corpus as a filter but without any lexical access.

The abbreviations which are still uncaptured by this technique are the following

> No. Sept. Rev. Jan. fig. Mass. Corp. no. Pa. La. 24-hr. cf. Tex. Mt. Miss. in. Wash. Hon. 6-hr. eqn. chap. a. Ore. Mar. sp. oz. hp. ex-Mrs. Tech. Supt. Mmes. Minn. Eq. Ed. Colo. 8-oz. u. seq. prop. nos. mos. min. mil. fed. ex-Gov. eqns. ed. al. Yok. Vs. Tenn. Sha. Sens. SS. Presbyterian-St. Pfc. OK. McN. Maj. Kas. Eng. Del. Cmdr. Cal. App. 42-degrees-F. 400-lb. 400-kc. 36-in. 3-hp. 3-by-6-ft. 29-Oct. 27-in. 25-ft. 24-in. 160-ml. 15,500-lb. 12-oz. 100-million-lb. 10-yr. 1.0-mg. 0.5-mv./m. 0.1-mv./m. 0.080-in. 0.025-in.

4.2.3 Experiment: lexicon without abbreviations

The observations above suppose that the abbreviation recognition process has no access to a lexicon. Let us examine what can be gained by using a lexicon to look up the litigious cases. Suppose now that, instead of trying to solve all the ambiguities during this tokenization phase, tokenization is reduced to number recognition and splitting words on spaces and unambiguous

separators. Then every word ending a sentence as well as real abbreviations ending with the period will be sent to the morphological analyzer with a trailing period. It will then be the role of the morphological analyzer to decide if the trailing period should be isolated as a separate, sentence-ending, character. Under this supposition, the Brown corpus produces 52430 letter-initial tokens ending in a period that must be sorted. Suppose that we have a complete lexicon, containing at least all the words in the Brown corpus, except abbreviations and proper names. Can we discover abbreviations using this method?

Consider this ordered filter on all strings terminated by a period:

1. if it is followed by a lower-case letter, comma or semi-colon, it becomes a *known abbreviation*;

2. if it is a lower case string, not a known abbreviation, and exists as a word in the lexicon without a final period, it is not an abbreviation, otherwise it is an abbreviation;

3. if it begins with an upper case letter, is not a known abbreviation, and appears elsewhere in the corpus without a trailing period, or only appears once or twice in the corpus, it is not an abbreviation (probably a proper name);

4. otherwise, it is an abbreviation.

The list of known abbreviations defined under (1) contains 183 unique upper and lower-case abbreviations (occurring a total of 1003 times in Brown). A sample of such such known abbreviations, given in order of decreasing frequency, follows:

> U.S. Jr. Mr. U.N. i.e. Co. p.m. e.g. S. a.m. etc. Inc. St. D.C. B.C. A.L.A.M. vs. Calif. lb. cm. ...

The list derived from (2) captures most of the cases in the corpus. There are 42865 lowercase initial strings appearing with a final period. 458 of these instances correspond to known abbreviations, and 42344 others correspond to words without final periods appearing in the lexicon. In some instances these words are really abbreviations. This happens when some string appears as both as an entire word and in an abbreviation, such as "fig" also appearing as "fig." for "figure." If we consider all these 42344 cases as sentence-ending non abbreviated words, then we misrecognize 29 sentences which contain

> chap. fed. fig. no. nos. u.

since these words are not considered as sentence-internal abbreviations (which the really in these sentences) but as full stops.

63 other instances (19 words) are recognized by step (2) as abbreviations:

> ca. cf. ed. eqn. eqns. ex-Gov. ex-Mrs. figs. hp. mil. min. mos. oz. pp. r.p.m. seq. sp. v. yrs.

By the time we reach step (3), we have decided in 46474 of the 52430 period-terminated string cases. Step (3) has to decided the case of the remaining 6056 uppercase initial possible abbreviations. Step (3) finds that 4628 of the remaining 6056 cases correspond to uppercase initial words somewhere else in the corpus without a final period; and of the remaining 1428, 583 appear only once or twice, so they are not considered abbreviations, either. This heuristic incorrectly identifies all occurrences of the following strings as sentence-ending non abbreviations since they appear elsewhere without a period:

App. Cal. Del. E. Ed. G. Jan. L. Mar. No. P. Rev. SS. Sept. Tech. V. W.

or only one or two times in the corpus:

Aubr. Brig. Cf. Cmdr. D.J. D.W. E.O. E.T. Eng. Eq. F.S.C. H.L. H.M. H.P.R. H.W. I.L. J.D.H. J.H. Jas. K.G. K.J.P. Kas. Maj. McN. Mfg. Mmes. N.A. N.D. N.L. P.L. P.S. P.m. Pfc. Presbyterian-St. Pt. R.H. R.L. Reps. Rte. Rul. Ry. S.S. Sens. Sha. Spec. Supt. T.W. U.S.C. U.s. Vol. Vs. W.G. W.H. W.M. W.R. Wm. Yok.

Step (4) identifies all the remaining candidates as abbreviations:

Atty. Aug. Capt. Ch. Christendom. Col. Dec. Feb. Fig. Figs. H.M.S. Hon. Lt. Martinez. Mrs. Mt. Nov. Oct. Op. Pp. Prof. Ref. Rep. Schaack. Sec. Sen. Stat.

Combining the abbreviations recognized by all four steps, only 205 sentences are erroneously split because they contain as yet unrecognized abbreviations, but 351 sentences end in strings thought to be sentence internal abbreviations. So we will incorrectly join 702 sentences. In other words, the above method of using a lexicon without abbreviations and the corpus as a filter to tokenize recognizes 51604 sentences out of 52511 original Brown sentences, or 98.27%. This slight degradation comes from the fact that "in." is recognized as a known abbreviation by Step (1), and so the 79 sentences ending in "in." are incorrectly joined to the sentence following them.

4.2.4 Experiment: lexicon with some abbreviations

Consider now a lexicon that has not only all the lower-case words in the corpus, but also contains frequent abbreviations, here meaning titles ("Mr.", "Mrs.", "Dr.", "Sen."), month name abbreviations ("Jan.", "Feb.", "Mar."), U. S. state abbreviations ("Ala.", "Calif.", "Penna.") and some common abbreviations ("etc.", "fig.", "no.", "Co.", "Ltd.", "Corp.") but not abbreviations like ("in.").

Now we can implement the following procedure, given a sequence of letters terminated by a period: 1) if it is followed by a lower-case letter, comma or semi-colon, then it is an abbreviation; 2) if it exists as an abbreviation in the lexicon, consider it as such; 3) otherwise, consider the word as a sentence terminator. Using the following list as a list of abbreviations in the lexicon provides us with only 74 sentences contain unrecognized non-terminal abbreviations candidates in the Brown Corpus. And we still have the original 207 sentences which end in an abbreviation that cannot be recognized correctly by any of the above techniques, giving a success rate of 52023 correctly recognized sentence boundaries out of 52511, or 99.07%.

The abbreviations recognized here are:

Strings containing internal periods, Single-Letters, State-Names, Titles, and the following: Assn. Av. Ave. Bldg. Blvd. Cf. Co. Corp. Ct. Dept. Dist. Eq. Fig. Figs. Inc. Jas. Jr. Ltd. Mfg. Msec. Mt. Mts. No. Op. Rd. Rte. Sr. St. Stat. Tech. USN. Vol. Vs. Yo. a. al. ca. cc. cf. cm. cu. dia. ed. eqn. eqns. etc. fig. figs. ft. gm. hp. hr. kc. l. lb. lbs. mEq. mc. mg. mil. min. ml. mm. mos. nw. oz. p. pl. pp. prop. sec. sq. v. var. viz. vs. yd. yrs.

4.2.5 Related work on sentence boundary recognition

Palmer and Hearst (1994) have recently produced a technical report[3] describing an approach to sentence boundary that uses a neural net applied to morphologically tagged text to decide the case of terminal periods. They achieved a 98.5% success rate following only one minute of neural net training. Since they do not use capitalization clues, this technique might be applied to languages such as German, or to all-upper case text. In this technical report, they mention other work applied to solving this problem using regression analysis based on the individual probabilities of words appearing before punctuation(Riley, 1989), and rules based on the lexical endings of words surrounding punctuation (Mller et al., 1980).

4.3 Morphologically Analyzed Words

A major question that must be answered by the designer of the tokenizer is whether there exists a one-to-one correspondence between a token and a set of classes, or can a token correspond to a sequence of classes. For example, in the Brown corpus the word *governor's* is considered as one token and is tagged as a possessive noun. In the Susanne corpus[4] the same string is divided into two tokens *governor* and *'s* each possessing its own tag. In this case, the choice between one or two tokens seems of little importance since one would suspect that subsequent linguistic treatment would rebuild a possessive structure corresponding to that produced by one token anyway. Of greater significance is the division of *'s* in the case of strings such as *it's, he's, that's, there's, who's, she's* and with the other English contractions. If the strings are retained as one token, then the linguistic analyzer must handle the case where a single token corresponds to a sequence of tags.

The same questions must be answered for other languages. In French it must be decided whether *l'addition, m'appelle, donne-le, va-t-il, c'est--dire, presqu'le, tape--l'oeil, d'abord, ...* are to be retained as one token, or divided into many. One problem with this choice is that there are arguments to make it either way: in order to make generalizations about grammar, it would be good to break out *l'* as a separate article but this introduces some ambiguity during tagging since it could also be a preverbal pronoun. A word like *rendez-vous* has possible readings as one or two tokens if the hyphen can separate words. In one case it is the noun *rendez-vous* and in the other it can be the imperative form of the reflexive verb *rendre* or the interrogative form of this verb with an inverted subject. Once the choice is made the linguistic component can take it into account, but different systems will make different choices which in turn makes comparing results or sharing tokenized text between researchers difficult. For example, available statistical tagging programs which choose parts of speech for words using their immediate context (Brill, 1992) cannot treat the case where a surface form might correspond to one or two tokens.

5 Conclusion

As we have seen, the problem of preparing raw text for a linguistic treatment raises many problems. In order to maintain as much flexibility as possible, the tokenization process should be considered as a series of modular filters through which text can be selectively passed. We

[3]This technical report can be retrieved by anonymous *ftp* at tr-ftp.CS.Berkeley.EDU. It is in the subdirectory /pub/cs/tech-report/cds-94-797, in postscript format.

[4]Available via anonymous *ftp* at 129.67.1.165 in the directory *ota/susanne*.

have seen here that the original text file undergoes preprocessing that eliminates some markings and rejoins hyphenated words. The tokenization proper begins. One of the main purposes of tokenization is to recognize sentence and word boundaries so that lexical look-up can proceed. Certain character ambiguities can be resolved be analyzing the structure of the the input strings, in order to produce a first pass at tokenization.

Once this pass is produced, one can consider other treatments of the tokenized text before lexical lookup is performed. For example, one might consider at this point rejoining parts of a proper name separated by blanks. This can be justified as a role of the tokenizer if the space is considered as an ambiguous separator which can be disambiguated by contextual clues. In English these contextual clues are uppercase letters appearing after the first word in the sentence.

Though rarely discussed, and quickly dismissed, tokenization in an automated text processing system poses a number of thorny questions, few of which have any perfect answers.

References

Brill, E. (1992). A simple Rule-Based part of speech tagger. In *Proceedings of the Third conference on Applied Natural Language Processing*, Trento, Italy. ACL.

Francis, W. N. and Kucera, H. (1982). *Frequency Analysis of English.* Houghton Mifflin Company, Boston.

Mller, H., Amerl, V., and Natalis, G. (1980). Worterkennungsverfahren als Grundlage einer Universalmethode zur automatischen Segmentierung von Texten in Stze. Ein Verfahren zur maschinellen Satzgrenzendestimmung im Englischen. *Sprache und Datenverarbeitung*, 1.

Nunberg, G. (1990). *The Linguistics of Punctuation.* C.S.L.I. Lecture Notes, Number 19. Center for the Study of Language and Information, Stanford, CA.

Palmer, D. D. and Hearst, M. A. (1994). Adaptive sentence boundary disambigation. Technical Report UCB/CSD 94/797, University of California, Berkeley, Computer Science Division.

Riley, M. D. (1989). Some applications of tree-based modelling to speech and language indexing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 339–352. Morgan Kaufmann.