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

April 22, 1994

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.

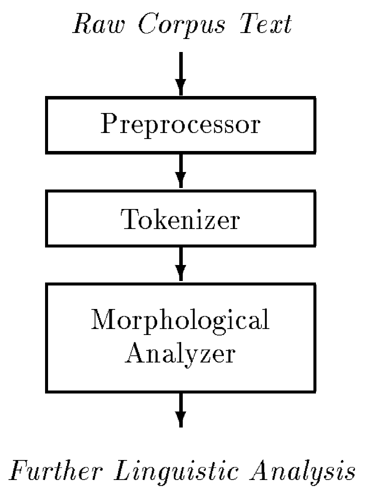

Figure 1: Text Transformations Before Linguistic Analysis.

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?

Sentences end with punctuation. The exclamation point and the question mark are almost always unambiguous examples of such punctuation. But the period is an extremely ambiguous punctuation mark. It is not trivial to decide when it is a full-stop, a part of an abbreviation, or both. In the Brown corpus, there are 48885 sentences and 3490 (1 in 14) contain at least one non-terminal period.

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*), punctuations, 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.

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 *lex* 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 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.

[0-9]+(\/[0-9]+)+	date
([+\-])?[0-9]+()?[0-9]*%	percent
(\[0-9]+,?)?+(\.[0-9]+|[0-9]+)*	numbers, dollars

The above table gives some regular expressions for English numbers, dollar value and date-like constructions that can 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.

4.2 Abbreviations

Another important class of tokens incorporating the period as an element are abbreviations. Lists of abbreviations can be long and, like lists of proper names, incomplete since creation of abbreviations is a productive process. Yet their recognition is imperative for proper sentence division. Consider that in the Brown corpus there are 4819 non-unique (323 unique) abbreviations containing only letters and ending in a period. There are 48805 sentences, so taking a simplistic route and considering every period followed by a space as a final period, we would be right only for only 90% of the word-ending periods.

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 automatically consider each of these sequences as abbreviations we will be right 3876 out of 3939 times in the Brown corpus. The detail is given in the table below.

regular expression	*Correct*	*Errors*	*FullStop*
[A-Za-z]\.	1323	30	14
[A-Za-z]\.([A-Za-z0-9]\.)+	626	0	63
[A-Z][bcdfghj-np-tvxz]+\.	1927	33	26
totals	3876	63	103

This means that, without consulting a lexicon, but only by using the structure of the words we will correctly recognize 3876 of the token-ending periods as part of an abbreviation (out of 4819 true abbreviations). We will introduce 63 errors by recognizing true full stops as false abbreviations, and introduce 103 ambiguities in abbreviations which should also be full stops. The number of correctly recognized sentence boundaries is then be 47696 out of 48805 (97.7%). The abbreviations in Brown that do not match the above regular expressions are the following:

> *etc. Fig. No. Co.* 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, we 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, a lower-case letter, or a number, or any string beginning with a capital letter terminated by a period.

Likely Abbreviation	*Correct (unique)*	*Errors*
[A-Za-z][∧⊔]*\.([,?]|⊔[a-z0-9]	155	81
either, not appearing without period	122	1

Using this definition of likely abbreviations recovers 138 of the 323 unique abbreviations in Brown, but introduces 54 false positives such as *chili, continuous, every, everyone, fed, feelers, finally, for, he, historians,* ... which are words that happen to end sentences that are followed by initial numbers.

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 eliminates all but the word *light-hearted* from the list of false positives, and reduces the number of likely abbreviations to 122, that are validated by this filtering technique. Using the corpus as a filter for validating likely abbreviations and accepting all structures of the form [A-Z]\. or [A-Za-z]\.([A-Za-z0-9])\.)+ as non-terminal abbreviations means that 909 non-unique abbreviations remain unrecognized as abbreviations (*Mrs.* accounts for 534 of these) and are thus taken as sentence dividers, to which are added 14+63+1 = 78 cases of falsely joined sentences, thus 986 sentences are incorrectly divided giving us still a 97.9% percent recognition rate. After using the corpus as a filter, without any lexical access.

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

Mrs. No. Sept. Sen. Rev. Jan. fig. Rep. Mass. Corp. Pa. Lt. 24-hr. La. Col. Tex. Mt. Capt. in. Wash. Prof. Miss. Atty. 6-hr. eqn. chap. Ore. Mar. oz. hp. ex-Mrs. Wm. Tech. Supt. Reps. Mmes. Minn. Eq. Ed. Colo. 8-oz. seq. prop. nos. no. mos. min. mil. ex-Gov. eqns. ed. dept. al. Yok. Vs. Tenn. Sha. Sens. SS. Ry. Presbyterian-St. Pfc. OK. Mfg. McN. Maj. Kas. Jas. Ind. Eng. Del. Cmdr. Cal. Brig. 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.

As we can see in the above lists, we have a number of titles that pass through these filters. We can recuperate some titles directly from the corpus by extracting sequences of [A-Z][a-z]+\. which are followed at least twice in the corpus by a two capitalized string. When we filter these results by eliminating words that appear elsewhere in the corpus without a terminal period, we produce a list of abbreviations with no false positives, although some of the abbreviations are not properly titles: *Atty. Ave. Capt. Cmdr. Col. Dist. Dr. Drs. Gen. Gov. Jas. Lt. Mmes. Mr. Mrs. Mt. Pfc. Prof. Rep. Reps. Sen. Sens. Sr. St. Supt. Vs.* . Adding in these discovered titles reduces the number of unrecognized abbreviations to 290. and reduces the number of incorrectly recognized sentences to 368, or 1 in every 133 sentences.

4.2.3 Experiment: lexicon without abbreviations

These observations 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 51240 letter-initial tokens ending in a period. Suppose that we have a complete lexicon of the language in which any form of a word may be found except abbreviations and proper names. Can we discover abbreviations using this method?

We tried the following 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 the word is a lower case string and the same word exists 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, and appears elsewhere in the corpus as a known abbreviation, it is an abbreviation;
4. if it begins with an upper case, and appears elsewhere in the corpus without a trailing blank, it is not an abbreviation (probably a proper name).
5. if it begins with an upper case, appears only once or twice, take it as sentence-ending word;
6. otherwise, it is an abbreviation.

The list of known abbreviations defined above contains 194 unique abbreviations such as *U.S. Jr. Mr. i.e. U.N. Co. p.m. e.g. S. a.m. Inc.* The list derived from (2) classifies most of the cases in the corpus since 42197 out of 52204 period-terminated strings fall into this class. It misclassifies *chap. fig. no. nos. prop.* and *u.* as words ending a sentence 29 times. (3) decides 78 other cases correctly, (4) decides 1827 cases but mistakes *App. Cal. Del. E. Ed. G. Jan. L. Mar. P. Rev. SS. Sept. Tech. V. W. Y.* since they appear elsewhere without a period. By the time the filter reaches the step (5), there are 563 words to consider, corresponding to 1327 instances of the 52204 period-terminated strings. This fifth step eliminates most of these instances, but incorrectly identifies the following abbreviations appearing one or two times as words: *Wm. Vol. Rte. Reps. Mmm. MMes. Eq. Aubr. Brig. Cmdr. Eng. Ind. Jas. Kas. Maj. McN. Mfg. Ore. Pfc. Pt. Rul. Ry. Sens. Sha. Vs.* and *Yok.* as words. Step (6) identifies the following as abbreviations: *Mrs. Fig. Sen. Oct. Nov. Dec. Feb. Rep. Aug. Figs. Op. Lt. Co. Pp. Mt. Capt. Wash. Ref. Prof. Atty. Stat. Schaack. Martinez. H.M.S. Christendom. Ch.* . The result of this filtering period-ending tokens through the lexicon gives the following technique. When the above filter is passed over the Brown corpus, 144 words are incorrectly tagged out of the 51240 candidates. Counting in terms of erroneously divided sentences this gives us a 99.7% success rate.

4.2.4 Experiment: lexicon with some abbreviations

Now let us suppose that our lexicon 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.*). Now the procedure will be the following, given a sequence of letters terminated by a period, it will be considered as an abbreviation: 1) if it is followed by a lower-case letter, comma or semi-colon, then it is an abbreviation; 2) if it is a known abbreviation, 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 53 incorrectly recognized words over the 51240 abbreviation candidates in Brown.

> Single-Letters State-Names *Assn. Av. Ave. Bldg. Blvd. Chmn. Co. Corp. Ct. Dept. Dist. Dr. Drs. Eng. Gen. Gov. Inc. Jr. Ltd. Messrs. Mr. Mrs. Msec. Mts. No. Rd. SS. Sr. St. Tech. Ter. U. USN. al. cc. cm. cu. dia. ed. etc. ft. gm. hp. hr. kc. lb. lbs. mEq. mc. mg. mil. min. ml. mm. mos. nw. pl. prop. sec. sq. 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 (Müller 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-à-dire, presqu'île, tape-à-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

[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*.

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

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

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.