ADAPTIVE PROBABILISTIC GENERALIZED LR PARSING

Jerry Wright[*], **Ave Wrigley**[*] and **Richard Sharman**[+]
[*] *Centre for Communications Research*
Queen's Building, University Walk, Bristol BS8 1TR, U.K.
[+] *I.B.M. United Kingdom Scientific Centre*
Athelstan House, St Clement Street, Winchester SO23 9DR, U.K.

ABSTRACT

Various issues in the implementation of generalized LR parsing with probability are discussed. A method for preventing the generation of infinite numbers of states is described and the space requirements of the parsing tables are assessed for a substantial natural-language grammar. Because of a high degree of ambiguity in the grammar, there are many multiple entries and the tables are rather large. A new method for grammar adaptation is introduced which may help to reduce this problem. A probabilistic version of the Tomita parse forest is also described.

1. INTRODUCTION

The generalized LR parsing algorithm of Tomita (1986) allows most context-free grammars to be parsed with high efficiency. For applications in speech recognition (and perhaps elsewhere) there is a need for a systematic treatment of uncertainty in language modelling, pattern recognition and parsing. Probabilistic grammars are increasing in importance as language models (Sharman, 1989, Lari and Young, 1990), pattern recognition is guided by predictions of forthcoming words (one application of the recent algorithm of Jelinek (1990)), and the extension of the Tomita algorithm to probabilistic grammars (Wright *et al*, 1989, 1990) is one approach to the parsing problem. The successful application of the Viterbi beam-search algorithm to connected speech recognition (Lee, 1989), together with the possibility of building grammar-level modelling into this framework (Lee and Rabiner, 1989) is further evidence of this trend. The purpose of this paper is to consider some issues in the implementation of probabilistic generalized LR parsing.

The objectives of our current work on language modelling and parsing for speech recognition can be summarised as follows:

(1) real-time parsing without excessive space requirements,

(2) minimum restrictions on the grammar (ambiguity, null rules, left-recursion all permitted, no need to use a normal form),

(3) probabilistic predictions to be made available to the pattern matcher, with word or phoneme likelihoods received in return,

(4) interpretations, ranked by overall probability, to be made available to the user,

(5) adaptation of the language model and parser, with minimum delay and interaction with the user.

The choice of parser generator is relevant to objectives (1) and (3). All versions satisfy objective (2) but are initially susceptible to generating an infinite number of states for certain probabilistic grammars, and in the case of the canonical parser generator this can happen in two ways. A solution to this problem is described in section 2. The need for probabilistic predictions of forthcoming words or phonemes (objective (3)) is best met by the canonical parser generator, because for all other versions the prior probability distribution can only be found by following up all possible reduce actions in a state, in advance of the next input (Wright *et al*, 1989, 1990). This consumes both time and space, but the size of the parsing tables produced by the canonical parser generator generally precludes their use. The space requirements for the various parser generators are assessed in section 3. The grammar used for this purpose was developed by I.B.M. from an Associated Press corpus of text (Sharman, 1989).

Objective (4) is met by the parse forest representation which is a probabilistic version of that employed by Tomita (1986), incorporating sub-node sharing and local ambiguity packing. This is described in section 4.

The final issue (objective (5) and section 5) is crucial to the applicability of the whole approach. We regard a grammar as a probabilistic structured hierarchical model of language as used, not a prescriptive basis for correctness of that use. A relatively compact parsing table presumes a relatively compact grammar, which is therefore going to be inadequate to cope with the range of usage to which it is likely to be exposed. It is essential that the software be made adaptive, and our experimental version operates through the LR parser to synthesise new grammar rules, assess their plausibility, and make incremental changes to the LR parsing tables in order to add or delete rules in the grammar.

2. PARSING TABLE SPACE REQUIREMENTS

2.1 INFINITE SERIES OF STATES: FROM THE ITEM PROBABILITIES

When applied to a probabilistic grammar, the various versions of LR parser generator first produce a series of item sets in which a probability derived from the grammar is attached to each item, and then generate the *action* and *goto* tables in which each entry again has an attached probability, representing a frequency for that action conditional upon the state (Wright *et al*, 1989, 1990). Sometimes these probabilities can cause a problem in state generation. For example, consider the following probabilistic grammar:

$$S \rightarrow A,\ p_1 \mid B,\ p_2$$
$$A \rightarrow c\ A,\ q_1 \mid a,\ q_2$$
$$B \rightarrow c\ B,\ r_1 \mid b,\ r_2$$

where p_1, p_2 and so on (with $p_1 + p_2 = 1$) represent the probabilities of the respective rules. After receiving the terminal symbol *c*, the state with the (closed) item set shown in Table 1 is entered, with the first column of probabilities for each item. After receiving the terminal symbol *c* again, a state with the same item set is entered but with the second column of probabilities for each item, and these are different from the first unless $q_1 = r_1$. For the probabilistic parser these states must therefore be distinguished, and in fact this process continues to generate an (in principle) infinite sequence of states. Although it may sometimes be sufficient merely to truncate this series at some point, the number of additional states generated when all the "goto" steps have been exhausted can be very large.

$A \Rightarrow \cdot a$
$B \Rightarrow \cdot cB$
$B \Rightarrow \cdot c$

Table 1: Item set with probabilities.

Item	First probability	Second probability
$A \to c \cdot A$	$p_1 q_1/(p_1 q_1+p_2 r_1)$	$p_1 q_1^2/(p_1 q_1^2+p_2 r_1^2)$
$B \to c \cdot B$	$p_2 r_1/(p_1 q_1+p_2 r_1)$	$p_2 r_1^2/(p_1 q_1^2+p_2 r_1^2)$
$A \to \cdot\ c\ A$	$p_1 q_1^2/(p_1 q_1+p_2 r_1)$	$p_1 q_1^3/(p_1 q_1^2+p_2 r_1^2)$
$A \to \cdot\ a$	$p_1 q_1 q_2/(p_1 q_1+p_2 r_1)$	$p_1 q_1^2 q_2/(p_1 q_1^2+p_2 r_1^2)$
$B \to \cdot\ c\ B$	$p_2 r_1^2/(p_1 q_1+p_2 r_1)$	$p_2 r_1^3/(p_1 q_1^2+p_2 r_1^2)$
$B \to \cdot\ b$	$p_2 r_1 r_2/(p_1 q_1+p_2 r_1)$	$p_2 r_1^2 r_2/(p_1 q_1^2+p_2 r_1^2)$

Table 2: Separated item sets.

$A \to c \cdot A$	1
$A \to \cdot\ c\ A$	q_1
$A \to \cdot\ a$	q_2

$B \to c \cdot B$	1
$B \to \cdot\ c\ B$	r_1
$B \to \cdot\ b$	r_2

We can avoid this problem by introducing a multiple shift entry for the terminal symbol c, in the state from which the one just discussed is entered. Multiple entries in the *action* table are normally confined to cases of shift-reduce and reduce-reduce conflicts, but the purpose here is to force the stack to divide, with a probability $p_1 q_1/(p_1 q_1+p_2 r_1)$ attached to one branch and $p_2 r_1/(p_1 q_1+p_2 r_1)$ to the other. These then lead to separate states with item sets as shown in Table 2.

The prior probabilities of a, b and c are obtained by combining the two branches and take the same values as before. Further occurrences of the terminal symbol c simply cause the same state to be re-entered in each branch, and eventually an a or b eliminates one branch. If c is replaced by a nonterminal symbol C, the same procedure applies except that a multiple *goto* entry is required, and this in turn means that a probability has to be attached to each *goto* entry (this was not required in the original version of the probabilistic LR parser).

Suppose in general that a grammar has nonterminal and terminal vocabularies N and T respectively. Conditions for the occurrence of an infinite series of states can be summarised as follows: there occurs a state in the closure of which there arise either

(a) two distinct self-recursive nonterminal symbols (A, B say) for which a nonempty string $\alpha \in (N \cup T)^+$ exists such that

$$A \overset{*}{\Rightarrow} \alpha\ A\ \beta \quad \text{and} \quad B \overset{*}{\Rightarrow} \alpha\ B\ \gamma$$

where $\beta, \gamma \in (N \cup T)^*$,

or (b) two (or more) mutually recursive nonterminal symbols for which a nonempty string $\alpha \in (N \cup T)^+$ exists such that

$$A \overset{*}{\Rightarrow} \alpha\ B\ \beta \quad \text{and} \quad B \overset{*}{\Rightarrow} \alpha\ A\ \gamma$$

where $\beta, \gamma \in (N \cup T)^*$, and in addition either

(i) a left-most derivation of one symbol from the other is possible:

$$A \overset{*}{\Rightarrow} B\ \delta \quad \text{where } \delta \in (N \cup T)^*$$

or (ii) a self-recursive nonterminal (C, say, which may coincide with A or B) also arises such that

$$C \overset{*}{\Rightarrow} \alpha\ C\ \theta \quad \text{where } \theta \in (N \cup T)^*$$

for the same α.

These conditions ensure that the α-successor of this state also contains items with A and B after the dot but with probabilities different from those for the earlier state, and moreover that this continues to generate an infinite series (different item probabilities do not always imply this). One way to prevent this series is to associate with each item in the lists (which form the states) an array of states for each nonterminal symbol, recording the state(s) in which that symbol occurred as the left-hand side of an item from which the current item is descended. These arrays can be created during the course of the LR "closure" function. Pairs of nonterminals satisfying the conditions above are then easily detected within the "goto" function, so that an appropriate multiple shift or goto entry can be automatically created for the last symbol of α. Only the items leading to the looping behaviour need to be separated by this means, and the number of additional states generated is small. Cases of three-way (or higher) mutual recursion with a common α are very rare.

2.2 INFINITE SERIES OF STATES: FROM THE LOOKAHEAD DISTRIBUTION

For the probabilistic LALR parser generator the lookaheads consist of a set of terminal symbols, as in the case of non-probabilistic grammars. However, for the canonical parser generator there is a full probability distribution of lookaheads and this creates a second potential source of looping behaviour. It is possible for item sets to have the same item probabilities but different lookahead distributions. Suppose that a state contains an item with a right-recursive nonterminal symbol (C, say) after the dot, with nothing following. If the state also contains another item with C after the dot followed by a non-null string, thus

$$C \overset{*}{\Rightarrow} \alpha\ C \quad \text{and} \quad C \overset{*}{\Rightarrow} \alpha\ C\ \beta$$

where $\alpha, \beta \in (N \cup T)^+$, then the new lookahead probability distribution computed for C will be a mixture of the old one and a distribution derived from β in the second item. The state automaton possesses a loop because of the right-recursion, and the lookahead distribution is different each time around so that again an (in principle) infinite series is generated. The second item can arise within the same state if C is also left-recursive (the simplest example of this is the grammar $S \rightarrow S\ S,\ p_1 \mid a,\ p_2$), and this problem also arises for an item of the second kind on its own, if β is nonempty but nullable.

It is possible to break the loop by introducing multiple shift (or goto) actions as before, but the procedure is complicated by the presence of null rules and/or left-recursion. These can allow the distribution to change even when there is just a single item in the kernel. In the absence of this behaviour the state from which the one just discussed is entered can be treated with a multiple entry in order to prevent the lookaheads from mixing, the conditions which give rise to this problem being checked within the "goto" function. The prior probability calculations at run-time are correct.

This procedure has not been fully implemented at the present time, but it seems that this kind of looping behaviour is more common than that discussed in the previous section. The additional states created by the multiple shifts exacerbates the already major disadvantage of the canonical parser with regard to space requirements.

2.3 MERGING OF CANONICAL STATES

Consider the following grammar:

$$S \rightarrow A \mid b\ A\ c$$

$$A \rightarrow e\ f\ S \mid g$$

(the rule-probabilities do not matter). The full canonical parser generator produces eighteen states, of which eight are eliminated by shift-reduce optimisation (Aho *et al*, 1985). Of the remaining states, a further eight consist of two sets of four, the sets distinguished only by the lookaheads. These states propagate the dot through the longer rules, but in fact the lookaheads are not used because in each case the series terminates in a shift-reduce entry. When this action occurs the parser moves to a state wherein the possible next symbols are revealed. These states can therefore be merged without compromising the predictive advantage of the canonical parser. This reduces the number of states to six, the same as for the LALR parser generator.

All this applies to the probabilistic version where the lookaheads are propagated as a distribution. A fairly simple procedure allows each state to be endowed with a flag to indicate whether or not any of the lookahead data are important. If not, merger can be based purely on the rule and dot positions for the items, and their probabilities. The numbers of states saved varies very much with the grammar: the above example represents an extreme case, and equally there are grammars for which no saving occurs.

3. COMPARISON OF PARSER GENERATORS

To compare the parser generators a test grammar developed by I.B.M. from an Associated Press corpus was used (Sharman, 1989). This grammar consists of 677 rules, ranked with a rule-count which was easily converted into a probability. It was convenient to use reduced versions of the grammar based on a rule-count threshold. Simply truncating the grammar is not sufficient, however, for two reasons. First, the resulting grammar can be disconnected in that there exist rules whose left-hand sides cannot occur in any string derived from S. Second, the grammar can be incomplete in that nonterminal symbols can arise within strings derived from S but for which there are no corresponding rules because all have counts below the threshold. The solution to these two problems is basically the same: recursively to add to the truncated grammar a small number of additional rules, with counts below the threshold, until the resulting grammar is connected and complete.

Applying this procedure for various rule-count thresholds creates a hierarchy of grammars and allows the relationship between the size of the grammar and the parsing tables to be explored. Table 3 contains a summary of the results. The number of states and total

Table 3: Parsing table space requirements.

		Non-prob LALR			Probabilistic LALR			Canonical		
Rules	Size	States	Entries	%>1	States	Entries	%>1	States	Entries	%>1
15	37	15	58	0	15	58	0	17	68	0
27	63	23	128	2	23	128	2	27	151	0
42	104	38	239	2	38	239	2	110	780	1
77	191	71	845	6	71	845	6		(lookahead	
115	291	120	1931	13	146	2297	7		looping	
194	510	214	7522	22	359	12051	19		behaviour)	
677	2075	1011	126322	46	3600	>250000				

number of entries in the parsing tables are compared for non-probabilistic and probabilistic LALR parser generators, the latter incorporating the multiple-shift procedure discussed in section 2.1. Shift-reduce optimisation was applied in all cases. The "size" of each grammar is the total length of right-hand sides of all rules plus the number of nonterminal symbols. The number of table entries is the total of all non-error *action* and *goto* entries including multiple entries. Also displayed is the percentage (%>1) of non-error cells in the tables (*action* and *goto*) which contain multiple entries.

Only limited results are available for the canonical parser generator because the lookahead loop suppression procedure (section 2.2) has not yet been implemented. Despite the use of the canonical merging procedure (section 2.3) the size of the parsing tables is clearly growing rapidly and this version of parser generator is only a practical proposition for rather small grammars.

What stands out most from the LALR results is not that the total number of entries grows with the size of the grammar but that it does so exponentially. The space requirements of LR parsers for unambiguous computer languages tend to grow in a linear way with size (Purdom, 1974). It is also notable (and no coincidence) that the proportion of multiple entries also grows with the size of the grammar. Although further stages of optimisation may enable space to be saved, attention must be focussed on the grammar itself.

The parser generation algorithm of Pager (1977) is similar to the LALR algorithm except that states are merged only when doing so results in no additional multiple entries. All such entries are therefore the result of non-determinism in the grammar (with the exception of loop-breaking multiple shifts as discussed in section 2). This algorithm has been implemented for probabilistic grammars, but for the test series the results are identical to those for the LALR generator. It follows that the growing proportion of multiple entries is the product not of state merger but of rich non-determinism in the grammar.

The last two rows in the table

correspond to the addition to the grammar of two large groups of rules, used twice and once respectively in the corpus. These infrequent rules appear to introduce a high degree of ambiguity, which also shows up during the state-generation procedure. Each state is first generated as a "kernel" of items, and the presence of more than one item within a kernel implies that there is a local ambiguity which is being carried forward in order that the state automaton is deterministic. For the non-probabilistic LALR parser generator with the full grammar of 677 rules, the average kernel contained 6.1 items and the largest contained no fewer than 68!

According to Gazdar and Pullum (1985), it has never been argued that English is inherently ambiguous, rather that a descriptively adequate grammar should be ambiguous in order to account for semantic intuitions. However, the I.B.M. grammar may suffer from excessive ambiguity and the parser would benefit considerably if some way could be found to reduce it.

Finally, the physical storage requirements are easily stated: each table entry requires four bytes, two to specify the action and two for the probability in logarithmic form, converted to a short integer.

4. PROBABILISTIC PARSE FOREST

In keeping with the first and fourth objectives set out in the Introduction a probabilistic version of the parse forest representation of Tomita (1986) has been developed. In the presence of ambiguity, and even more so with uncertainty in the data, the number of interpretations may increase exponentially with the length of the input string. The impact of this is minimised by sub-node sharing and local ambiguity packing. Where two or more parses contain parts of their interpretation of a sentence which are identical they can share the relevant nodes. And, two or more parses may differ because of ambiguity which is localised: if part of the sentence is derivable from a nonterminal symbol in more than one way then the relevant nodes may be packed together. By thus compacting the parse forest the space requirement becomes $O(n^2)$ for most grammars (Kipps, 1989).

Employing this representation for probabilistic grammars requires that a value be attached to each node which enables the eventual calculation of the parse probability for the whole sentence given the data. In addition it is necessary that the m (say) most probable interpretations be obtained without an exhaustive search of the compacted parse forest.

A value $P(\Delta, \{D\}_{i...j} \mid A)$ is attached to each node in a parse tree, where Δ denotes a particular derivation of the string $w_i \dots w_j$ from the symbol A, and $\{D\}_{i...j}$ represents the corresponding acoustical data. This probability is the product of the probabilities of all rules used in the particular derivation of $w_i \dots w_j$ from A and the likelihoods of those words given the data, and is easily found for a particular node from the probabilities attached to each subnode and the rule probability when the reduce action occurs. This calculation is not affected by the context of $w_i \dots w_j$, and therefore shared nodes need have only one value. For locally ambiguous packed nodes the probabilities of each alternative are recorded, in order that the correct ordering of alternatives can be created at further packed nodes higher in the parse forest.

The probability attached to the *S*-node at the apex of any parse tree is $P(\Delta, \{D\}_{1...M} \mid S)$ where M is the length of input. If all alternatives are retained in the parse forest then the parse probabilities given all the data can be obtained by normalisation. If all that is required is to identify the single most probable parse then all local ambiguity can be resolved by maximising at packed nodes (in the manner of the Viterbi algorithm) and retaining only the most probable derivation, because this ambiguity is invisible to higher-level structures in the forest.

The general problem of identifying the m most probable parses is more complex. The current m most probable are stored at each shared node together with a compact way of indicating which combination of subnodes corresponds to each derivation. Upon reduction by a rule whose right-hand side is of length k, the new m most probable derivations must be found and sorted from the (in the worst case) m^k possibilities. If two reductions are possible, to the same nonterminal symbol and spanning the same data, and if the right-hand sides are of length k_1 and k_2, then the worst-case number of possibilities is $m^{k_1} + m^{k_2}$, and so on. With appropriate book-keeping there are efficient ways to find and sort the m most probable of these, and record the subnodes. In practice this requires an array of size $4m$ stored for each node in the parse forest.

This approach has several advantages as compared with the original Bayesian algorithm for uncertain input data (Wright *et al*, 1989, 1990). The results are essentially equivalent, and most of the exponentially-growing number of possible interpretations are truncated away on grounds of probability, so the algorithm requires polynomial time and space. Furthermore, in this version the probabilities in the parsing tables are used only for prediction (to guide the pattern-matcher) and not for calculating the parse probabilities. These predictions do not have to be very precise so space can be saved by storing each probability in logarithmic form as a short integer. All this applies to isolated-word recognition; for connected speech the situation could be different.

5. ADAPTABILITY

The speed and effectiveness of the probabilistic LR parser would be seriously compromised if a large grammar (of, say, thousands of rules with a high degree of ambiguity) were adopted, and yet it would seem that if a grammar-based language model is to be employed for large-vocabulary speech recognition then the need for a large grammar will be unavoidable. The full version of the I.B.M. grammar referred to in section 3 extends to many thousands of rules but the greater part of these consist of oddball-rules that are used only once or twice in the corpus. Collectively the oddballs are important because they allow the corpus to be modelled, but individually each one is rather insignificant. It may be the case that generalisations (perhaps going beyond a context-free grammar) would eliminate a lot of these rules, but there is also a case for the parser to be made adaptive.

One approach to adaptation would be to assume a probabilistic grammar in Chomsky normal form and then use the inside-outside algorithm (Lari and Young, 1990). This approach has a lot to recommend it, but here we consider an alternative approach based on a rule-adaptive enhancement of the LR parser. The principle is that at any time the parsing tables are based on a relatively small core grammar of important rules, but with an error-recovery procedure and a backup grammar. Error-recovery allows new rules to be created as required, and rules can be transferred between backup and core grammars in response to usage.

The probabilistic LR parser has been enhanced with such a procedure. The minimum adaptation which can allow an ungrammatical sentence to be accepted is a local change to a single existing rule: in this sense it is assumed that the sentence is "close" to the language. The conditions for rule-adaptation can be summarised as follows:

Input string:

$$w_1 \ldots . w_i \ldots . w_j \ldots . w_n$$

Existing rule: $A \rightarrow \alpha_1 \; \beta \; \alpha_2$

Adapted rule: $A \rightarrow \bar{\alpha}_1 \; \delta \; \bar{\alpha}_2$

such that

$$S \overset{*}{\Rightarrow} \gamma_1 \; A \; \gamma_2$$

$$\gamma_1 \; \bar{\alpha}_1 \overset{*}{\Rightarrow} w_1 \ldots . w_i$$

$$\delta \overset{*}{\Rightarrow} w_{i+1} \ldots . w_j$$

$$\bar{\alpha}_2 \; \gamma_2 \overset{*}{\Rightarrow} w_{j+1} \ldots . w_n$$

and where $\bar{\alpha}_1$, $\bar{\alpha}_2$ consist of α_1, α_2 with any unused nullable symbols suppressed.

The adaptation therefore consists in the deletion, insertion or replacement of a substring within the right-hand side of a rule, and the suppression of unused nullables simply ensures that all remaining symbols actually contribute to the parse of the sentence. This procedure usually generates a number of rule-candidates. Assuming that one of these is chosen as correct (although it may not be possible to automate this entirely), it is then added to the backup grammar as a potential core rule. With sufficient evidence of usage a rule may be promoted from backup to core grammar, and likewise a rule may be demoted.

A version is being developed in which the LR parsing tables are updated incrementally as rules are transferred between backup and core grammars. This should occur on-line, with the intention that the core grammar be kept reasonably compact (and the parser correspondingly fast) while adapting to the user. This is still very far from a complete solution to the problem of context-free grammar adaptation, but a system operating along these lines would satisfy (at least to some degree) all the objectives as set out in the Introduction.

ACKNOWLEDGMENT

This work was funded by the I.B.M. United Kingdom Scientific Centre.

REFERENCES

A V Aho, R Sethi and J D Ullman (1985), *Compilers: Principles, Techniques and Tools*, Addison-Wesley.

Gerald Gazdar and Geoffrey K. Pullum (1985), "Computationally relevant properties of natural languages and their grammars", *New Generation Computing*, 3, 273-306.

Fred Jelinek (1990), "Computation of the probability of initial substring generation by stochastic context free grammars", I.B.M. Research, Yorktown Heights, New York.

J R Kipps (1989), "Analysis of Tomita's algorithm for general context-free parsing", *Proceedings* *of the International Workshop on Parsing Technologies,* Carnegie-Mellon University, 193-202.

K Lari and S J Young (1990), "The estimation of stochastic context-free grammars using the inside-outside algorithm", *Computer Speech and Language*, **4**, 35-56.

Chin-Hui Lee and Lawrence R. Rabiner (1989), "A frame-synchronous network search algorithm for connected word recognition", *IEEE Trans. on Acoustics, Speech and Signal Processing*, **37**, 1649-1658.

Kai-Fu Lee (1989), *Automatic Speech Recognition*, Kluwer Academic Publishers.

D Pager (1977), "A practical general method for constructing LR(k) parsers", *Acta Informatica*, **7**, 249-268.

P Purdom (1974), "The size of LALR(1) parsers", *BIT*, **14**, 326-337.

Richard Sharman (1989), "Observational evidence for a statistical model of language", I.B.M. United Kingdom Scientific Centre Report 205.

Masaru Tomita (1986), *Efficient Parsing for Natural Language*, Kluwer Academic Publishers.

Jerry Wright and Ave Wrigley (1989), "Probabilistic LR parsing for speech recognition", *Proceedings of the International Workshop on Parsing Technologies,* Carnegie-Mellon University, 105-114.

Jerry Wright (1990), "LR parsing of probabilistic grammars with input uncertainty for speech recognition", *Computer Speech and Language*, **4**, 297-323.