Efficient Disjunctive Unification for Bottom-Up Parsing

David Carter
SRI International Cambridge Research Centre
23 Millers Yard, Mill Lane, Cambridge, CB2 1RQ, U.K.
dmc@ai.sri.com, dmc@sri.co.uk

Abstract

This paper describes two novel techniques which, when applied together, in practice significantly reduce the time required for unifying disjunctive feature structures. The first is a safe but fast method for discarding irrelevant disjunctions from newly-created structures. The second reduces the time required to check the consistency of a structure from exponential to polynomial in the number of disjunctions, except in cases that, it will be argued, should be very unusual in practical systems. The techniques are implemented in an experimental Japanese analyser that uses a large, existing disjunctive Japanese grammar and lexicon. Observations of the time behaviour of this analyser suggest that a significant speed gain is achieved.

1 Introduction

This paper describes the approach taken to the unification of disjunctive feature structures in an experimental bottom-up shift-reduce Japanese analyser called Propane, for **Pro**log **P**arser using the **N**adine Grammar. Nadine (Kogure, 1989; Kogure and Nagata, 1990), which is implemented in Lisp, is the analysis and translation component of SL-TRANS, the spoken language translation system under development at ATR Interpreting Telephony Research Laboratories, and its large (12,000 line) grammar and lexicon make extensive use of disjunction.

The general problem of unifying two disjunctive feature structures is non-polynomial in the number of disjunctions (Kasper, 1987). That is, barring revolutionary developments in the theory of algorithms, the problem is NP-complete, and the time taken to perform such a unification can, in general, at best be an exponentially increasing function of the number of disjunctions. However, in writing large grammars of natural languages, it is often convenient to be able to specify constraints in terms of disjunctions. This seems especially to be the case for Japanese, because of its relatively free word order and widespread ellipsis. It is therefore important to develop unification algorithms that can in practice unify disjunctive feature structures in a reasonable time, despite the inherent NP-completeness of the task.

Propane's unification method embodies two novel techniques. Firstly, when a new mother constituent is created by the application of a grammar rule to daughter constituents during bottom-up parsing, disjunctions not relevant to the mother can safely be removed. However, deciding on relevance in less than exponential time is a non-trivial problem. Propane's technique is rapid, and results in the removal of enough irrelevant disjunctions that constituents higher in a parse tree are not burdened with inordinately many of them. Secondly, Propane adopts a modification to Kasper's (1987) disjunctive unification algorithm that "almost all the time" (in a sense of that phrase to be discussed), runs in binomial time.

Practical results, which will be presented throughout this paper, suggest that these techniques have the desired effect of allowing Propane to parse even quite long sentences in a reasonable time. These results need, however, to be evaluated in the context of ATR's Japanese language processing research programme in general and of Propane's approach to parsing in particular, which will therefore be presented in the next section as a preliminary to the main body of the paper.

2 Bottom-up Parsing of Japanese

The Nadine system is geared towards the processing of Japanese sentences of the type encountered in telephone conversations. At ATR, a substantial corpus of dialogues has been collected by simulating, both by speech and by keyboard, telephone calls to the organizing office of an international conference. At the time the research described here was carried out, Nadine's grammar and lexicon were being developed and tested mainly on a subcorpus of 100 sentences comprising five of these dialogues. The results presented in this paper therefore all derive from applying Propane to this same sentence set. Although the size of the set is comparatively small, the sentences in it were not in any sense "made up" to suit either the Nadine or Propane parsers. Rather, to the degree that a simulation can approach reality, they can be taken as representatives of the kinds of sentences to be handled in a realistic language processing application.

Japanese has several characteristics which suggest that bottom-up parsing approaches might be particularly fruitful. The language is a head-final, strongly left-branching one. This means that modifiers always attach to a head on their right, and that there is a preference for attachment to the nearest such head that obeys the constraints that syntax, semantics and pragmatics place on possible combinations. This preference is so strong as to suggest a parsing algorithm that first constructs analyses that obey it, backtracking and producing analyses with different bracketings only if the initial analysis or analyses are judged unacceptable by some outside process.

Attempts have been made, for example in Nadine and by Shimazu and Naito (1989), to use the left-branching preference to select among alternative actions in a chart parser. However, the approach adopted in Propane is to implement the preference directly into the mechanism of a shift-reduce parser.

In general, a shift-reduce parser uses a table of parse states and possible actions that determine, at each stage, whether a shift or a reduction is appropriate, and in the latter case, what grammar rule should be used. However, when Japanese is formalized using a grammar in which every rule has exactly two right-hand-side elements – as is the case in Nadine grammar – the left-branching preference corresponds to a strategy of reducing the top two categories on the stack whenever there is a grammar rule that allows them to be reduced, and shifting only when this cannot be done. No table is therefore required. Nadine's grammar rules include syntactic, semantic and pragmatic information, so that Propane's decision to reduce or not depends on the acceptability of the result at all three of these linguistic levels. Such a test takes advantage of the maximum amount of available information, and applies it in a fairly straightforward and efficient way.

Alternative lexical entries for words, and alternative grammar rules that can apply to the same pair of daughter categories, mean that each position on the parser's stack is in fact occupied not by a single category but by a list of categories (each of which, of course, contains a disjunctive structure that may have many realizations). The lengths of these lists do not grow significantly as parsing progresses, because just as the lexicon and the grammar can introduce alternatives, so the application of grammar rules can remove them. The attempt to reduce each of m possible head daughters with each of n possible non-head daughters typically results in far fewer than $m*n$ mother structures, because not every rule application succeeds.

One complication that arises in parsing written Japanese is that word boundaries are not indicated explicitly. This means that the lexicon imposes a lattice structure, not a simple sequence of tokens, on the input, so that when a shift operation is needed, the point to shift from is not necessarily well-defined.

Propane deals with this situation in the following way. When shifting, edges of all lengths are placed onto the stack, and are allowed to participate in any following sequence of reductions. Before the next shift, however, Propane "prunes" the edges that constitute the top of the stack, removing all but the longest. This corresponds to the assumption that there is a preference for longer strings of characters to correspond to lexical items where possible, but that this preference should be overturned when a shorter string, but not a longer one, allows a reduction with what precedes it.

A large proportion of the 100-sentence subcorpus targeted by Nadine can be parsed correctly by this simple approach of always preferring reductions to shifts and longer edges to shorter ones. Nevertheless, on many occasions the correct parse will involve at least one violation of these preferences. In general, some kind of intelligent backtracking and/or lookahead is required. In Propane, only a limited form of lookahead exists. Sometimes, an examination of the parts of speech (i.e. category names only and not feature values) in the grammar and those of the constituents in the stack and of the item that would be consumed in a shift shows the following situation: a reduction is possible, but if it is performed, the next shift cannot itself be followed by a reduction, whereas if a shift is performed next, two reductions may be possible. That is, there are two alternatives: reduce now and then be forced to shift twice, or shift now and, unless unification failure prevents it, reduce twice. In such situations, Propane chooses the second option. This often allows sentences to be parsed which would not otherwise be, and does not prevent the parsing of any sentences in the subcorpus. Because only category names, and not features, are examined, the lookahead procedure is very quick.

With this small amount of lookahead included, Propane was able to parse 75 of the 100 sentences in the subcorpus. No attempt was made to check thoroughly the validity of these because of the present author's limited familiarity with Japanese and the Nadine grammar; however, they were inspected informally, and none seemed to be obviously wrong. Of the 25 sentences for which no parse was found, ten involved an incorrect reduction. Eight of these might have been prevented had information corresponding to Gunji's (1988) treatment of "sentence levels" for modification been present in the grammar. Twelve sentences failed through incorrectly favouring longer edges over shorter; all of these failures involved a lexical entry for the same particle sequence, and could have been prevented either by altering the treatment of that sequence or by implementing the same kind of limited lookahead for the long-over-short preference as was done for the reduce-over-shift preference. Of the other three failures, two were sentences on which the Nadine parser also failed, suggesting that they were outside grammatical and/or lexical coverage, and one remained unexplained. Thus in summary, up to 98% of the

subcorpus could have been assigned plausible analyses by Propane given the improvements just listed.

3 Pruning Irrelevant Disjuncts

If bottom-up parsing is to be efficient, it is important that disjunctions that are irrelevant to a newly-created mother constituent – that is, disjunctions whose values never affect the realizations of the constituent, i.e. the set of terms in its disjunctive normal form – are discarded whenever possible. Otherwise, the number of disjunctions in a constituent will be roughly proportional to the number of lexical entries and grammar rules used to construct it, and the time taken to unify two constituents will increase at least as fast as that number and probably rather faster.

However, it is not possible simply to discard disjunctive constraints that refer only to the daughter nodes, because feature structures are graphs, not trees; the same substructure frequently appears in more than one place. When a grammar rule has identified part of the mother structure with part of a daughter one, then any disjunctions involving the latter must be preserved. Some means must therefore be found of keeping track of what pieces of structure are shared, or in other words, what pairs of feature paths lead to the same values. If this is done, a disjunction that explicitly involves only daughter constituents can safely be discarded if no feature path through the mother leads to it or to any of its components.

Of course, the set of feature paths that share a value will differ for the different realizations (complete choices of disjuncts) of a disjunctive structure. It is not even simply the case that each disjunct contributes its own set of common paths; members of two different disjunctions can cause two paths to have the same value, in a realization in which they are both selected, if they place the same variable in two different positions. Thus to decide infallibly whether a given disjunct should or should not be discarded, one would need to cycle through every possible realization of the whole structure, a process that is exponential in the number of disjuncts and therefore unacceptable. This rules out, for our purposes, a representation similar to that of Eisele and Dörre (1988), in which equivalences between different parts of a structure are denoted by explicit pointers from one part to the other. Eisele and Dörre's pointers can occur inside a disjunction, and the values at the positions to which they refer can also be affected by disjunction.

The alternative adopted in Propane is one that errs on the side of caution, in the sense that it never throws away a disjunct that should be kept, but does sometimes keep a disjunct that should be thrown away. The result of the latter kind of error is not to give incorrect results but merely to encumber the representation with some irrelevant information.

Each disjunctive structure returned by a lexicon or grammar predicate, therefore, is assigned a set of "path groups", which each correspond either to a variable that appears more than once in the original Nadine definition, or to an explicit identity equation between two or more positions in the feature structure. To some extent, a path group is analogous to a set of Eisele and Dörre pointers that all point to the same position. However, the crucial point is that in Propane, no record is kept of which position in the and/or tree each path comes from. This means two things. Firstly when deciding whether to throw away a disjunction referring to a particular position in a daughter structure, Propane can check the (unique, disjunction-independent) set of path groups, and if no possible equivalence with part of the mother structure is found, the disjunction can safely be pruned. The price we pay for this disjunction-independence is that the pathgroups can specify spurious equivalences. It is possible for two paths to be associated when they arise from two different, incompatible disjuncts, or to remain associated after the disjunct(s) from which they arose have been eliminated through later unification. However, since path groups are used only for deciding what disjunctions to discard, and not as part of the feature structure representation itself, a spurious path group can only result in some inefficiency, and not in an incorrect result.

This technique is thus a compromise between, on the one hand, carrying out possibly exhaustive computation to achieve a perfect result, and on the other hand not discarding anything at all. It avoids any exponential expansion of disjunctions at the cost of some slight unnecessary processing at a later stage. In practice, the cost involved seems quite acceptable, in that the number of disjuncts in a constituent does not increase greatly with its height in the parse tree.

Another consequence of keeping irrelevant disjuncts is that if, at the end of the parse, the set of all full realizations of a disjunctive feature structure is exhaustively enumerated, then the same realization may be encountered repeatedly. However, experience suggests that for the current Nadine grammar, this is not a problem. The average number of realizations (identical or different) per parse, of the 75 sentences successfully parsed, was exactly two, and only one sentence received more than six realizations.

The pruning operation in fact resulted in, on average, a 20% decrease in the number of disjunctions in a newly created mother constituent, over all "reduce" operations performed in processing the corpus. Probably for this reason, the number of disjunctions in a new mother constituent only barely shows a positive correlation to the size, in constituents, of the subtree that it dominates and from which it has been built.[1] On the other hand, if pruning were not

[1] The correlation between subtree size and number of disjunctions, for the 406 tree nodes created, was only just significant at the 5% level, given the null hypothesis that the

performed, each constituent could be expected to add its quota of irrelevant disjuncts to every other constituent that dominated it. Despite the relatively modest figure of a 20% decrease over one reduction, the cumulative effect of such decreases over a whole parse is therefore quite significant.

In particular, it is worth noting that if, through pruning, the number of disjunctions in a node does *not* increase with the number of nodes it dominates, then disjunctive unification will have no effect on the time complexity of parsing as a function of sentence length. There is reason to hope that this will often be the case; while disjunction may be widespread in grammar rules and lexical entries, Kasper (1987) observes that in his implementation, "in the analysis of a particular sentence most features have a unique value, and some features are not present at all. When disjunction remains in the description of a sentence after parsing, it usually represents ambiguity or an underspecified part of the grammar." It is tempting to interpolate between the extremes of single words and whole sentences and to speculate that, with thorough pruning, the number of disjunctions in a node should *decrease* with its height in the tree.

4 Pairwise Consistency Checking

When a new mother constituent has been created by rule application, it is essential to verify that it does in fact have at least one consistent realization. Although redundancy is not a major problem for our purposes, a representation that did not distinguish between realizable and unrealizable structures (that is, between success and failure in unification) would seriously flawed. However, consistency checking is, in the general case, an NP-complete problem.

Kasper (1987) describes a technique which, for every set of n conjoined disjunctions, checks the consistency first of single disjuncts against the definite part of the description, then that of pairs, and so on up to n-tuples for full consistency. At each stage k, any disjunct that does not take part in any consistent k-tuple is eliminated.[2] If all the disjuncts in a disjunction are eliminated, the conjunction of which that disjunction is a conjunct is eliminated too; and if the outermost conjunction of the whole feature structure is eliminated, unification fails. This technique has the advantage that the pruning of nodes at stage k will make stage $k+1$ more efficient. Nevertheless, since n can sometimes be quite large, this exhaustive process be time-consuming, and indeed in the limit will take exponential time.

Propane's attempted solution to this problem is based on the hypothesis that the vast majority of large unrealizable disjunctive feature structures that will be created in the use of a practical natural language grammar will be not only unrealizable, but also "pairwise unrealizable", in the sense that they will fail at or before the second stage of Kasper's consistency check, for $k = 2$.

The reason we can expect most unrealizable structures also to be pairwise unrealizable is that most commonly, unrealizability will result from the contents of two nodes in the tree being incompatible, through assigning non-unifiable values to the same position in a feature structure. Although there can clearly be exceptions, the hypothesis is that it is fairly unlikely, in a large disjunctive structure (which is the case where exponentiality would be harmful) that there would be a non-pairwise inconsistency but no pairwise inconsistency.

Following this hypothesis, when the Propane unifier has created a structure, it checks and prunes it first for pairwise consistency, and if this succeeds, risks trying for a single full realization (one choice at each disjunct) straight away. Thus it differs from Kasper's algorithm in two ways: no exhaustive k-wise checks are made for $k > 2$, and when a full check is made, only one success is required, avoiding an exhaustive search through all combinations of disjuncts.[3] Of course, if the structure is pairwise realizable but not fully realizable, the search for a single success will take exponential time; but, according to the hypothesis, such occurrences, for structures with enough disjuncts for exponential time to be unacceptably long, should be extremely rare.

The effectiveness of this strategy can only be judged by observing its behaviour in practice. In fact, *no* instances were observed of the search for a full realizabilion taking an inordinately long time after pairwise consistency checking and pruning have succeeded. Thus it can be tentatively concluded that, with the current version of the Nadine grammar and with bottom-up parsing, the risk is worth taking: that is, a full realization is virtually always possible, in reasonable time, for a pairwise consistent structure. Maxwell and Kaplan's (1989) belief that "... [simple inconsistencies] become less predominant as grammars are extended to cover more and more linguistic phenomena" does not therefore appear to be true of the Nadine grammar, in spite of its coverage of a wide range of phenomena at many linguistic levels; or if it is true, it does not affect the success of Propane's strategy. That is, even if simple inconsistencies are less predominant, they are still common enough that a large structure that is unrealizable because of complex inconsistencies will also

number of disjunctions is independent of subtree size.

[2] Somewhat confusingly, Kasper uses the term "n-wise consistency" for the checking of $n+1$-tuples of disjuncts. We avoid this usage.

[3] According to Maxwell and Kaplan (1989), "in practice, Kasper noted that...once bad singleton disjuncts have been eliminated, it is more efficient to switch to DNF [disjunctive normal form] than to compute all of the higher degrees of consistency." This variation of the algorithm given in Kasper (1987) is closer to Propane's strategy, but the expansion to full DNF is itself in general an exponential process and will, when many disjunctions remain, be far more expensive than looking for a single realization.

be unrealizable because of simple ones.

Of course, this does not alter the fact that in general, i.e. for an arbitrary input and for an arbitrary grammar written in the Nadine formalism, Propane's unification algorithm, like Kasper's, is exponential in behaviour. In the limit, an exponential term in the formula for the time behaviour of an algorithm will dominate, however small its associated constant factor.

Unlike Nadine's unifier, Propane's strategy has the property that when a structure survives consistency checking, not every member of every disjunct in it can necessarily participate in a full realization; that is, ideally, it should have been pruned. However, this property is only undesirable to the extent that, at the end of the parse, it makes any exhaustive search for full realizations inefficient through excessive backtracking. Again, in practice, this seems not to be a problem; exhaustive full realization is extremely quick compared to parsing.

An analysis of Propane's processing of its corpus reveals quite wide variation in the relationship between the total number of disjunctions in a rule application (in both daughters and the rule) and the time taken to perform the unification. However, although, unsurprisingly, unification time increases with the number of disjunctions, it appears from inspection to be perhaps linear with a small binomial component, and not exponential. This is, in fact, what an analysis of the algorithm predicts. The linear component derives from the check of each disjunct separately against the definite part, while the parabolic component derives from the pairwise check. The relatively small size of the latter may imply that a majority of disjuncts are eliminated during the first phase, so the second has less work to do.

5 Unification and Parsing Times

The absence of any known exponential process (other than the final phase of unification, which appears never to take very long) in Propane's parsing and unification algorithms gives grounds for expecting that in practice, the time taken to parse a sentence of n lexical items should be polynomial in n. Because of the pruning of irrelevant disjunctions, the value of n should be fairly small, leading to a significant speed advantage over systems like the Nadine parser that do not prune disjunctions and that use the full (exponential) version of Kasper's algorithm. The results of a comparison between Nadine's and Propane's parsing times suggest that such an advantage does exist. However, the results are not sufficiently detailed to allow the verification of Propane's exact time behaviour.

As sentence length grows, Propane tends to perform progressively faster in a statistically significant way.[4] In particular, Nadine's attempts to parse two fairly long sentences (12 and 18 lexical items respectively) in the corpus had to be aborted because of the time they took, but both these sentences received a parse from Propane in ten to thirteen minutes. Had Nadine not been aborted in these cases, two more data points would be available that would increase the significance further.

The progressive speed advantage of Propane may be due partly to the fact that, as discussed above, it follows only the single sequence of shifts and reductions specified by the algorithm described in section 2, and does not explore alternative bracketings. However, Nadine is also, through numerical scoring, sensitive to the left branching preference, which guides it to explore, and presumably to find, preferred parses first; and the Nadine times used in the comparison were those taken to find the first parse, not all parses.

Another difference between the two parsers is that Nadine, being chart-based, stores the edges it creates so that later backtracking need not cause work to be repeated. Propane does not backtrack in this way. However, because of a mundane practical limitation in the Prolog implementation used, Propane is also forced to store (assert in the database) every constituent it creates, advancing the parse by successive storing, failing and backtracking rather than by the simple recursion that would otherwise be performed. The time taken to store constituents in fact increases faster than that used by other aspects of processing, and for the longest sentences parsed represents 70 to 80 per cent of the total time. It might be, therefore, that if storage time were ignored for both parsers, Propane's speed advantage would be even more apparent.

Such vague remarks are admittedly unsatisfying and should, given time, be firmed up by the acquisition and analysis of more data, and by separate evaluations of the parsing and unification time behaviours. The latter would involve comparing the two parsers running with the same unifier and then the two unifiers running under the same parsing algorithm. Nevertheless, there are, as already mentioned, *a priori* grounds for expecting Propane's unifier to have an increasingly marked advantage, and the data presented here are fully consistent with that expectation, showing as they do a statistically significant trend.

A formal complexity analysis of a bottom-up parser using the techniques described in this paper would only be of limited interest. Complexity analyses deal with worst cases, and in those terms, the essential hypothesis that pairwise consistency checking will "almost all the time" be sufficient is meaningless. Likewise, to claim that disjunction pruning

[4] For each of the 31 sentences containing more than one lexical item and successfully parsed by both systems, the correlation was measured between the number of lexical items in the sentence and the logarithm of the ratio of parsing times. It was easily statistically significant at the 5% level, and its sign indicated that the correlation is in the direction of Propane performing better for longer sentences.

greatly reduces the number of disjunctions in higher tree nodes in the case of Propane and the Nadine grammar, is to say nothing about its effectiveness in the worst case. One could easily write a grammar in which every disjunction from daughter nodes was needed by mothers, so that nothing would be pruned at all. And thirdly, it is not claimed that the left-branching preference in Japanese is anything more than a preference, albeit quite a strong one.

However, because the grammar, lexicon and sentence set dealt with here are in no sense toy ones written to test unification techniques but are the tools of a major effort to process natural language as it as actually used, it is of interest to analyse Propane's overall time behaviour under the assumption that the relationships inferred above through observation and statistical methods are valid.[5] There seems to be no *a priori* reason to doubt that the same behaviour could be achieved by other systems or for other languages (except, of course, that the left-branching characteristic is language-dependent).

Thus in Propane, the number of unifications attempted during the successful parsing of a sentence of length N is $O(N)$ (this happy situation is, of course, bought at the price of failure when the preference heuristics fail). Let us assume a strongly left-branching structure, which, being maximally unbalanced, is the worst case. Then the number k of nodes dominated by each new mother node the parser (attempts to) create will be uniformly distributed between 0 and N. From observation, it seems that the number of disjunctions d involved in a unification that dominates k nodes will be proportional to k (This is the pessimistic option; as argued earlier, there are grounds for hoping that, with sufficient pruning, d will not increase with k at all, so that disjunctive unification time will made no contribution to parsing time as a function of N). Unification time for d disjunctions, under the pairwise consistency hypothesis, appears to be proportional to d^2. Compositional semantic interpretation will probably mean in the limit that the size of the non-disjunctive part of a constituent will also be proportional to the number of constituents dominated. Unification time here is order $n\ log\ n$ in the sizes n of the input structures (Kasper, 1987). Thus a node dominating k others will take order $k^3 log\ k$ time to create. Summing over k from 0 to N gives an order $N^4 logN$ result. More generally, a parsing algorithm that on atomic categories has order $f(N)$ should, with disjunction, have order $f(N)N^2 logN$ if the distribution of k over nodes created is also uniform.

In conclusion, the assessments of the various aspects of Propane's time behaviour are all consistent with, and in some cases provide good evidence for, the claim that the two novel techniques described here can significantly enhance the speed with which sentences can be parsed using a large grammar containing disjunctions. As long as the essential hypothesis about pairwise consistency holds for the particular grammar and the sentences it will in practice encounter, polynomial time behaviour can be expected, as compared to an exponential time for other approaches involving disjunctive unification.

[5] Statistical correlation tests, of course, cannot tell us *what* relationship, e.g. linear or exponential, holds between two variables; they can only tell us that *some* relationship appears to exist. The time analysis can therefore only be tentative.

Acknowledgements

This research was carried out while I was a visiting researcher at ATR Interpreting Telephony Research Laboratories, Kyoto, Japan. I am grateful to Dr Akira Kurematsu, Mr Kiyoshi Kogure and others at ATR for thought-provoking discussions and for providing a very pleasant research environment.

References

Eisele, A., and Dörre, J. (1988) "Unification of Disjunctive Feature Descriptions", *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics.*

Gunji, T. (1989) "Syntactic Sketch 88: Japanese". In: *Syntax: an International Handbook of Contemporary Research*, de Gruyter.

Kasper, R. (1987) "A Unification Method for Disjunctive Feature Descriptions", *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.*

Kogure, K. (1989) "Parsing Japanese Spoken Sentences based on HPSG", *Proceedings of the International Workshop on Parsing Technologies*, Carnegie Mellon University, 132–141.

Kogure, K., and Nagata, M. (1990) "Parsing Spoken Japanese Sentences Based on HPSG", *Proceedings of Coling-90.*

Maxwell, J.T., and Kaplan, R. (1989) "An Overview of Disjunctive Constraint Satisfaction", *Proceedings of the International Workshop on Parsing Technologies*, Carnegie Mellon University, 18–27.

Shimazu, A., and Naito, S. (1989) "Preference Reading Models of Japanese Sentences", *Gengo Shori to Communication Kenkyūkai*, 89:114 (in Japanese).