A Japanese sentence analyzer

by N. Maruyama
M. Morohashi
S. Umeda
E. Sumita

This paper presents the design of a broad-coverage Japanese sentence analyzer which can be part of various Japanese processing systems. The sentence analyzer comprises two components: the lexical analyzer and the syntactic analyzer. Lexical analysis, i.e., segmenting a sentence into words, is a formidable problem for a language like Japanese, because it has no explicit delimiters (blanks) between written words. In practical applications, this task is made more difficult by the occurrence of words not listed in a dictionary. We have developed a five-layered knowledge source and used it successfully in the lexical analyzer, resulting in very accurate segmentation, even in cases where there are unknown words. The syntactic analyzer has two modules: One consists of an augmented context-free grammar and the PLNLP parser; the other is the dependency structure constructor, which converts the phrase structures to dependency structures. The dependency structures represent various key linguistic relations in a more direct way. The dependency structures have semantically important information such as tense, aspect, and modality, as well as preference scores reflecting relative ranking of parse acceptability.

1. Introduction

Sentence analysis (i.e., lexical and syntactic analysis) is a key technology in various natural-language processing systems such as machine translation, question answering, and information retrieval, since it is the first process an input sentence undergoes, and overall system performance depends heavily on its quality.

Although a number of Japanese sentence analyzers have been developed for various systems [1, 2], they were designed specifically for their applications. It is noteworthy that a significant amount of knowledge embedded in such systems depends solely on knowledge about the Japanese language itself, not on a specific application; hence, there has been much duplication of effort. Considering the large number of emerging natural-language applications, it is important to have a wide-coverage Japanese sentence analyzer which is independent of its application.

The main difficulties in analyzing a written Japanese sentence, compared with those of other languages such as English, are the following: First, Japanese has multiple character sets (three basic types: Kanji, Hiragana, and Katakana, as well as the Roman alphabet for borrowed words); thus, a single word can be expressed in different ways. Second, in Japanese no explicit delimiters (blanks) are supplied between words. These two characteristics of Japanese make lexical analysis more difficult and yet even more important than for other languages, especially when an input sentence contains unknown words. The third characteristic which makes analysis of Japanese difficult is the fact that Japanese is a so-called nonconfigurational language, i.e., has relatively few restrictions on word order. Thus purely syntactic analysis is inadequate for Japanese, because it produces excessively ambiguous results. To remedy this, we introduce semantic features which are domain-independent.

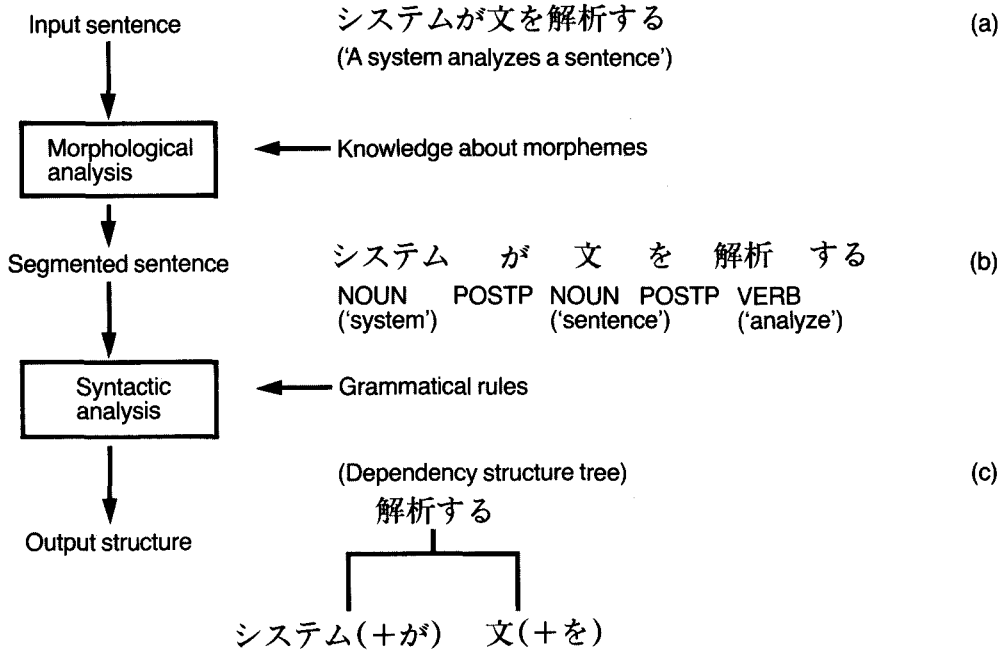

Figure 1

Sentence analyzer and example sentence.

In this paper, we present the design and current status of a Japanese sentence analyzer which we are developing. The objective of this project is to establish a Japanese sentence-analysis technology which is robust enough to cover almost every Japanese sentence, including those containing unknown words, while at the same time retaining application-independence. No natural-language application system can be successful without a good sentence analyzer, since natural-language sentences are diverse, and enormous linguistic knowledge is needed to handle the diversity. One of the keys to the success of the English-to-Japanese machine translation system SHALT [3] was that a very-broad-coverage English syntactic analyzer known as PEG [4, 5] was already available, and developers could fully utilize its output. Our sentence analyzer will serve as a good starting point for further research and development.

The major characteristics of our system are as follows:

- Hypothesis of words not listed in a dictionary.
- Systematic application of five types of lexical knowledge.
- Mizutani's grammar [6] as the base grammar.
- Preference attachment using valency theory.
- Special handling of auxiliary verbs to extract modal information.
- Dependency structures as output.

An overview of the components is given in Section 2, followed by a description of each component: lexical analysis in Section 3, syntactic analysis in Section 4. The current status of this project, which we started in 1985 and have since been enhancing, is briefly summarized in Section 5. We offer our conclusions in Section 6.

2. System overview

As shown in **Figure 1**, the overall system consists of two components: a lexical analyzer and a syntactic analyzer.

A Japanese sentence, comprising Kanji, Hiragana, Katakana, and other special characters without delimiting blanks, is first fed into the lexical-analysis component. An example of a Japanese sentence is shown in Figure 1(a).

The lexical analyzer segments the input sentence into words by consulting the system dictionary and by applying a set of segmenting rules. It also attaches part-of-speech information to each word identified during the process. The output of the lexical-analysis component is a sequence of segmented words, as shown in Figure 1(b).

The syntactic analyzer produces the dependency structures [Figure 1(c)], which represent various key linguistic relations. They contain semantically important information such as tense, aspect, and modality, as well as preference scores reflecting relative ranking of parse acceptability.

3. Lexical analysis

It is well known that Japanese text can be described, from the lexical point of view, as a sequence of phrases, each of which consists of a content word and zero or more succeeding function words. A content word is a word that

Table 1 Character types and action names.

Character types

A:	Alphabetic
N:	Numeral (Kanji or Arabic numeral)
K:	Katakana character (phonogram mainly used to represent foreign words)
H:	Hiragana character (phonogram mainly used to represent function words)
J:	Kanji character
I:	Independent character, such as "(", which makes a word by itself
S:	Special character, such as "-", a sequence of which makes a word
D:	Dependent character (the segmentation depends on the context)
B:	Blank character
L:	End-of-line virtual character
blank:	Initial-nil/end-of-data virtual character

Action names

U:	Throw away the current character
V:	Put word boundaries before and after the current character
W:	Insert a word boundary and throw away the current character
X:	Put a word boundary before the current character
Z:	No action
1:	Invoke the action "Z", if the current character = the previous character; otherwise invoke "X"
2:	Invoke the action "X", if the type sequence = "TTU" (where T, U mean any types and $T \neq U$); otherwise invoke "Z"
3:	Invoke the action "Z", if the type sequence = "TUT" (where T, U mean any types and $T \neq U$); otherwise invoke "U"

has some meaning by itself, such as a noun, verb, or adjective. A function word, on the other hand, is a word that adds some features such as tense or aspect to the preceding content word: an auxiliary verb, for example, or a word that relates two content words, such as a case particle. Although the order of phrases in a sentence is very flexible, the order of words within a phrase is highly restricted and hence recognizable by means of simple "phrase-internal" connection rules, which characterize consecutive word combinations as well as word-stem and inflectional/derivational morpheme combinations [7].

Japanese lexical-analysis algorithms have two goals: to divide a sentence into words and to assign a part of speech to every word. Typically, the analysis is carried out in two steps: First, the input string is segmented into phrases, using information derived from shifts in the character sets; then the resulting segments are analyzed into words by means of phrase-internal connection rules [7–9]. This approach has been taken because phrase boundaries are typically accompanied by a shift from a Hiragana character to a Kanji character. This regularity comes from Japanese writing conventions, in which content words are customarily written in Kanji characters and function words in Hiragana characters.

These algorithms can recover missing proper phrase boundaries by repeatedly applying some sort of phrase analysis routine. However, they have no effective way to prevent "overcutting," i.e., cutting the string at a point where there is, in actuality, no phrase boundary. To overcome this problem, we

1. Introduce a more sophisticated combination of techniques for checking character types (names of character sets such as Kanji, Hiragana, etc.) than the shift from a Kanji character to a Hiragana character.
2. Use a "loose" definition of phrase in the phrase-analysis algorithm in order to accept spurious segments (possible phrases) resulting from overcutting.

Another problem for other lexical analyzers is that because they rely completely on dictionary entries, they fail when a word is encountered that is not listed in a dictionary. To overcome this important limitation, we use rules that characterize the combinations of character types that can make words.

The lexical analyzer consists of five stages, each using a distinct kind of knowledge that is stored in the form of a rule or a table. The first and second stages of analysis segment the input string into phrases according to allowable character-type combinations and permissible character sequences, respectively. The key point here is that these initial segmentations never segment the text in the middle of a word. The third and fourth stages are phrase-analysis algorithms, both of which use a dictionary and morphological rules based on the loose definition of phrase, but segment the text in different ways. The fourth stage has additional rules which hypothesize a word not listed in the dictionary; it is invoked only when the third stage fails. The last stage is different from the preceding four in that it bridges the segmentations based on our phrase definition with the phrase as normally defined. For example, the first four stages may wrongly cut a compound into phrases. This stage combines those components into a compound. Through the five stages, the analyzer selects the correct part of speech from the alternatives provided by the dictionary.

To summarize, the stages, described in detail below, are as follows:

1. Segmentation by character type.
2. Segmentation by character sequence.
3. Segmentation by a longest matching algorithm based on a "loose" definition of phrase.
4. If Stage 3 fails, segmentation by a bottom-up parallel algorithm based on the same definition with "word presumption" rules.
5. Compound-word composition.

The analyzer provides no ambiguous segmentation, but may

assign ambiguous parts of speech if the local context does not give enough information. This kind of ambiguity may be resolved by syntactic analysis.

• *Segmentation by character type*

A sequence of character types sometimes gives definite information on segmentation. To take a simple example from English, the sequence NDN (N = numeral, D = ".") implies that "." is used as a decimal point (not a period), so that no word boundaries should be inserted before and after the ".".

Knowledge for segmenting by character types can be effectively represented by a set of character-type definitions (character "a" → alphabetic), and rules having type patterns and their corresponding action names (type "AJ" → action "X"). The types and action names used to describe this knowledge are shown in **Table 1**.

The pattern of a rule with length two causes the analyzer to check only the character types between two consecutive characters, and decides whether a word boundary is to be inserted or not. But the register in the analyzer stores the preceding character types to allow actions based on longer character sequences. Experimentally, three is sufficient for the maximum pattern length to be stored in the register.

Figure 2 depicts how the knowledge in this stage is used to segment text.

• *Segmentation by character sequence*

The knowledge used in this stage is a list of character sequences, each of which provides a definite segmentation boundary immediately after the last character of the sequence. **Figure 3** shows an example of the segmentation resulting from this stage.

In Japanese, most sequences useful for this stage are those of one or more function words. The most powerful character to give a definite boundary is "を", which is the case particle for objects, and which consists of a unique character not appearing in any other words (in fact, "を" is totally reliable).

• *Segmentation by a longest matching algorithm*

Stage 3 performs segmentation within a phrase determined by Stages 1 and 2. A longest matching algorithm from left to right, with a morphological check between each consecutive two-word candidate, is cheap yet very effective (and commonly used in existing analyzers); it is applied to a phrase, according to rules based on a loose definition of phrase (**Figure 4**). The input phrase may contain more than one phrase based on the loose definition, so the algorithm applies the rules iteratively until the input-phrase string is exhausted. As stated previously, we define <phrase> to accept phrases resulting from overcutting by the previous stages. Rule 2 is different from the "precise" definition (the ordinary definition in Japanese grammars) in that it permits

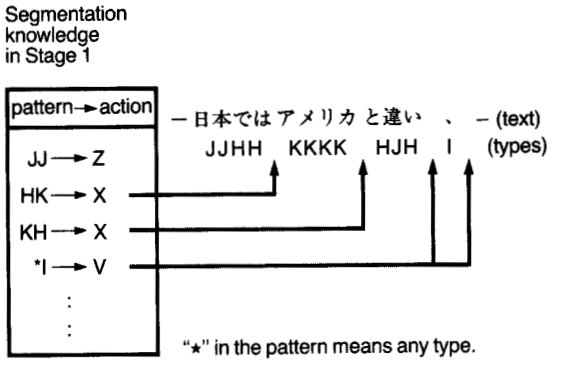

Figure 2

Segmentation in Stage 1.

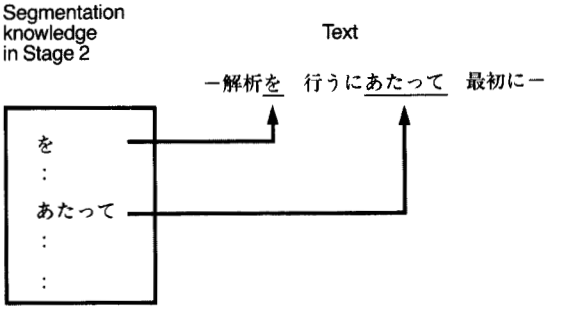

Figure 3

Segmentation in Stage 2.

1. <text> : : = <phrase>...<phrase>
2. <phrase> : : = <content word> <function words> |
   <content word> |
   <function words>
3. <content word> : : = <a word in the dictionary> |
   <a word by char type rules>
4. <function words> : : = <extended func>...<extended func>
5. <extended func> : : = <function word> |
   <inflection>

Figure 4

"Loose" definition of phrase in Stage 3.

<function words> to be a <phrase>. This is necessary because some kinds of boundaries between a content word and a function word are very similar to the boundaries between phrases, and sometimes an inserted clause with parentheses appears at such a boundary.

In Rule 3, a content word is supplied either by the dictionary or by the rules of character types ("NN ··· N" as a numeral word). This rule is also looser than the precise definition, thereby permitting a compound to be segmented into several phrases. But this looseness is necessary because Stage 1 provides good word-boundary information (as a phrase boundary) within a compound. The discrepancies between the phrases determined by our algorithm at this stage and the desired phrases are resolved in the last stage.

A function word, <extended func>, is supplied by the function-word table, which is stored separately from the dictionary—although the number of function words is small, they occur frequently.

Morphological checks occur only at the first term of Rules 2 and 4. A 96-bit part of speech based on Okochi's categories [10] is used. The morphological check is performed by Boolean operations among the parts of speech and connectivity vectors for word combinations, eliminating illegal segmentations from the list of alternatives, as well as illegal part-of-speech assignments from the alternatives for a word.

• *Segmentation by a bottom-up parallel algorithm*

The previous stage can fail because it encounters a word not listed in the dictionary or because it cannot find an analysis in which all word connections are legal. When this occurs, the phrase resulting from Stages 1 and 2 is segmented again by a bottom-up parallel algorithm (at high cost) based on the same phrase definition used in the previous stage (see Figure 4) and so-called word-presumption rules. Word-presumption rules are used to guess that some unidentified substring is a word on the basis of character-type sequence. More specifically, they are used to expand Rule 3 of Figure 4, by adding new alternatives for <content word>. The bottom-up parallel segmentation generates many possible alternative segmentations, so the stage selects the highest-scoring alternative by using an evaluation function.

Informally, the word-presumption rules look for

- Two consecutive Kanji characters (a well-known heuristic rule).
- One Kanji character and preceding or succeeding Hiragana characters of two *moras*[1] (a variation of the preceding rule).
- Three consecutive Kanji characters, which is a special pattern for some kinds of proper nouns.
- A string of Hiragana characters of any length.
- Voicing of the first consonant of a word in a compound word except the first one (known as *rendaku* in Japanese). (This should be found in the dictionary.)

The above rules also guess the part of speech of the word from one of three candidates, *taigen* (noun), *sahen* (stem of a certain class of verbs), and *keiyoudoushi* (stem of a certain class of adjectives).

Applying these rules to every substring of the segment (result from Stages 1 and 2) yields content-word alternatives and finds segmentations according to our phrase definition. A segmentation evaluation function is then applied to each alternative segmentation, evaluating it according to the score determined by the following expression:

$$\text{Score} = (\text{MaxPhraseCnt} - \text{PhraseCnt}) \times (\text{CharCnt} - \text{PresumedCharCnt}),$$

where

MaxPhraseCnt	= maximum number of phrases in possible alternative segmentation. (Actually, the number of characters in text is used for convenience in the calculation.)
PhraseCnt	= number of phrases in alternative segmentation.
CharCnt	= number of characters in text.
PresumedCharCnt	= sum of characters of every presumed word.

Note that, even though the first and second stages segment the input string into "phrases," some "phrases" may include more than one actual phrase because of missed phrase boundaries.

The expression implies a strategy in which the fewer the phrases and presumed words, the more reliable the segmentation. Selection from among the alternatives with the highest scores is done by the same strategy as the longest matching algorithm: The alternative that has the longest word in leftmost position should be selected, which is the same strategy used in the third stage.

• *Compound-word composition*

This is the stage used to resolve discrepancies that come from the loose definition of phrase. The typical discrepancy is that "情報検索" ('information-retrieval') may be recognized as a compound if it is stored in the dictionary as a compound. But if it is not, the analyzer recognizes this as a sequence of two independent words, "情報" ('information') and "検索" ('retrieval'). The result depends completely on the content of the dictionary, which implies the continuous refinement of the dictionary over many years. To circumvent this, the rule "NN → N" ("noun noun" combination generates a "noun" compound) is employed.

[1] A mora is a linguistic unit used to characterize languages (such as Japanese) in which vowel or syllable lengths are linguistically significant. In Japanese, a short vowel is one mora in length; a long vowel, two.

Knowledge concerning part-of-speech patterns is used in this stage. This knowledge describes not only noun compounds like the above example, but also compounds of other part-of-speech combinations and word–morpheme combinations (morphological checks treat morphemes as words).

4. Syntactic analysis

The syntactic-analysis component receives a segmented sentence (a sequence of words) from the previous step and produces its structural description. This section describes some of the major decisions made in developing the syntactic component.

• *Developing a large grammar*

At the outset of a project such as ours—one with both research and development goals—there is a formidable obstacle: how best to develop a large grammar that is suitable for computer applications. Given the difficulty of developing large grammars, the idea of starting from scratch was unattractive. A review of existing grammars, however, revealed a variety of problems.

There are, of course, large traditional grammars of Japanese which contain a wealth of information relevant to our task [11–14], but these contain highly informal descriptions and so were judged a poor starting point. On the other hand, there are also a number of formal grammars of Japanese. All of these except one were originally developed primarily to explore various issues in theoretical linguistics [15, 16]. There are several reasons why we judged these to be inappropriate for our purposes. One major problem was that the published grammars are too restricted in coverage, at least given our requirements. We might have chosen to extend one or another of these formalized grammars, but formalization does not guarantee practical extensibility nor suitability for computer applications. For example, the use of a Generalized Phrase Structure Grammar (GPSG) [17] entails the adoption of a parsing algorithm suitable for this class of grammars. However, there are a number of controversial issues [18] in the parsing of GPSG grammars that we preferred to avoid. Similar comments hold for transformational grammars. In fact, our general view of current linguistic theories is that, however interesting, they are only partially formalized and highly speculative and are constantly undergoing changes in response to current theoretical concerns. Since our project has long-range practical goals, we wish to avoid this instability and keep our purely theoretical commitments to an absolute minimum.

This left us with the one exception, Mizutani's grammar [6], which was relatively robust, had been developed for computer applications, and, since it is an unaugmented context-free grammar, afforded us a choice of efficient and well-understood parsing algorithms.

```
(5700)  NP POSTP
          --> PP(¢NP,CASE=CASE(POSTP),
                TOPIC=TOPIC(POSTP),KOOU=KOOU(POSTP),
                PSMODS=PSMODS...POSTP)
(5840)  VERB --> VP(¢VERB,HINSISEI='YOU')
(5851)  PP(¬DAI,¬NO,SF,CASE)
        VP(¬NO,CASES0,CASE(PP).ISIN.CASES0,
          CASE(PP).NOTIN.CASES,
          <@CASE(PP).EQ.'DIV'|
          @CASE(PP).ISIN.SF(PP)>)
          --> VP(-NAI,CASES=LISTIFY<CASE(PP)>...CASES,
                PRMODS=PP...PRMODS,N=N(PP)+N+4,
                <TOPIC(PP),+DAI>)
```

Figure 5

Example grammar rule.

Table 2 Feature types.

	Feature types	*Number of features*
Features for syntactic checking	**CLASSIFY**	27
Features for meaning extraction	**FILLSIN, CF, SF, MODAL**	17

The major difficulties with the grammar were syntactic ambiguity, overgeneration, and insufficient coverage.

• *Feature system*

Mizutani's grammar was designed to cover as many phenomena as possible, although at the expense of accepting ungrammatical sentences and producing too many alternative (and mostly nonsense) parses for grammatical sentences. However, it is well known that the size of a context-free grammar (CFG) increases sharply when a grammar writer tries to incorporate fine control (number agreement in English, for example) as a means for rejecting ungrammatical or meaningless parses. A common solution to this problem is to introduce a feature system [19]. We decided to introduce a feature system into our grammar rules for another reason as well: to improve the coverage and maintainability. In addition, an excellent parsing system was available: PLNLP (Programming Language for Natural Language Processing [4, 20, 21]) supports interactive grammar development and provides a convenient rule-writing language that includes feature handling. An example grammar rule is shown in **Figure 5** (a discussion is given in the later subsection entitled *Dependency structure*).

We defined five types of features, which are summarized in **Table 2**. The **CLASSIFY** features are for classifying

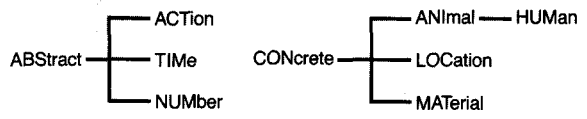

Figure 6

Semantic feature system.

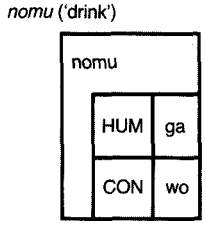

Figure 7

Example of case frame.

nonterminal symbols. For example, we have four kinds of *ku*. They are represented by four nonterminals in the original grammar: *Mudaijuttai-ku*, *Arujuttai-ku*, *Shunukijuttai-ku*, and *Noshujutsu-ku*. In our grammar these are replaced by a single nonterminal *ku* augmented with a set of features of a type called **CLASSIFY**, thus reducing complexity and redundancy. There are four other types of features: **FILLSIN**, **CF**, **SF**, and **MODAL**. They are not used to reject illegal parses, but rather to carry information useful for various applications. The **FILLSIN** feature indicates that the node with this feature may fill in a *trace* (or *gap*) within its premodifier. For example, the head noun *otoko* ('the man') in the phrase

watashi ga aisuru otoko (the man whom I love)
('I') ('love') ('man')

receives the **FILLSIN**='**WO**' (case particle for direct object) feature.

The **CF** (case frame), **SF** (semantic feature), and **MODAL** features represent information which is sometimes categorized as "semantic." We treat them as both domain-independent and useful for any application. Our intention is to keep our analyzer as general as possible. For example, **SF** as indicated in **Figure 6** is quite simple. Use of the features is explained in the succeeding sections.

Our feature system is implemented by adding attributes to each nonterminal symbol in the original base-grammar, and by coding tests on these attributes for the rules. In parsing, these attributes are passed only from bottom to top.

• *Attaching preference scores*

Ambiguity is always a serious problem for natural-language processing systems. In particular, domain-independent parsers like ours tend to produce numerous alternative parses because they cannot reject meaningless interpretations in a certain domain. Various purely syntactic principles of preference such as right attachment [22], minimal attachment [23], and lexical preference [24] have been suggested for English, but no such principles have been posited for Japanese.

Like the prepositional-phrase-attachment problem in English, argument–predicate attachment is one of the main sources of ambiguity in Japanese. If we know what kinds of modifiers usually modify what kinds of predicates, we can attach preference scores to alternative parses. Since this information depends on individual predicates, we decided to encode it in the entries in the lexicon. However, adoption of preference semantics [25] or case grammar [26] is inadequate for our purpose, since these methods require deep semantic coding which inevitably has to be hand-crafted and is subjective. The number of predicates in our lexicon is estimated to reach over 20000 at the final stage. Keeping such information consistent requires an enormous effort.

Our solution to this problem is to use valency grammar [27] instead of case grammar. Valency grammar is based on modifier–modifiee patterns with surface case particles (case grammar uses deep semantic roles such as **AGENT** and **RECIPIENT**). An example of a valency pattern is given in **Figure 7**. The verb *nomu* ('drink') takes two modifiers, i.e., a **HUM**(an)-type noun phrase with the case particle *ga* and a **CON**(crete)-type noun phrase with the case particle *wo*. [There are several varieties of valency theories. Some of them do not designate "distribution" (i.e., the semantic class of modifier noun phrases) in valency patterns. In Japanese linguistics, valencies are usually discussed along with their distribution.] **HUM** and **CON** are **SF**-type features. Although the patterns (or *case frames*)[2] still have to be coded by hand, this is much easier and the result more reliable than in the case of using deep semantic roles, because these patterns can be found in existing text and thus are not subjective. In addition, there is a possibility of future automation of this process.

[2] We use the term *case frame* to represent the fact that there are a fixed number of named slots. This is to be distinguished, however, from the usage in [26], since our case labels are superficial.

The preference calculation is performed as follows: A nonnegative integer, called a *preference score*, is given to every node in a parse tree. Initially, every node has the value 0. The score propagates from bottom to top by adding all children's scores to that of their parent. When a modifier–predicate modification is encountered in the process, a test to determine whether the modifier fits the expected slot of the predicate is made. If it does fit, a "reward," in accordance with the values shown in **Table 3**, is added to the predicate node. The table reads that when both semantic features for modifiers and predicates and their case particles are equal, the score is +4, and so on. The feature SF was designed for this purpose (see Figure 6). If the modifier does not fit any slot of the predicate, no score is added. The score appearing in the root node of the tree indicates the plausibility of the tree itself.

An example of scoring is shown in **Figure 8**. The reader may have noticed that the first tree (the correct interpretation in a neutral situation) gets a higher score. Of course, low-rated trees which violate the case frames cannot be discarded. Metaphors and negative sentences such as

My car drinks gas.

and

Personal computers never drink beer.

are some examples which violate the case-frame patterns.

There is also a possibility of using valency theory for rejecting incorrect parses by means of encoding not only valency patterns which are likely to occur but also ones which *never* occur. For example, the case particle *wo* always indicates the object case and therefore is never used in combination with an intransitive verb or an adjective. Most other case particles, however, may be used to indicate optional modifiers such as time and location, so the modifiability depends on the semantic class of the modifier noun. Since we cannot completely rely on the semantic features of nouns [e.g., *ji* ('o'clock') is coded as TIME but may be used as a direction in certain very specific domains], it is too dangerous to discard parses based on this information. Hence, we decided not to employ this approach generally. Impossible modifications such as the *wo* case just described are handled individually by the feature checks in the grammar rules.

Although the figures in the scoring table are intended only to indicate relative strength of modification and although further improvement is being planned, the current setting is producing notable results. We tested our analyzer using 225 sentences of which 97 were ambiguous with respect to our grammar. The effectiveness of the preference scoring is shown in **Table 4**. In our body of test data, admittedly small, the correct parses always received the highest score. Although more than one parse usually received the highest score, no incorrect parse was ranked higher than any correct parse. That is, our algorithm made no errors.

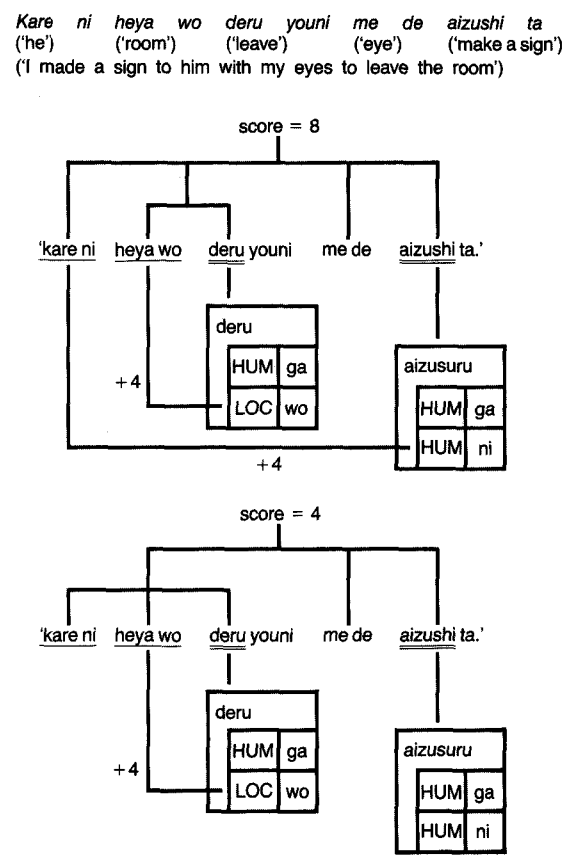

Figure 8

Example of scoring.

Table 3 Scoring calculation.

	Sm = Sp	*Sm —, = Sp*	*Sm = Unknown or Sp = Unknown**
Cm = Cp	+4	+1	+2
Cm = Unknown†	+2	+0	+1
Otherwise	+0	+0	+0

Sm: Semantic feature of modifier
Sp: Semantic feature of predicate
Cm: Case particle of modifier
Cp: Case particle of predicate
* Information not found in the lexicon
† Case article missing

Table 4 Effectiveness of preference attachment.

No. of predicates per sentence	*A. No. of ambiguous sentences*	*B. No. of differentiated sentences*	*C. Hit ratio (100 B/A)*	*D. No. of all parses per sentence*	*E. No. of differentiated parses per sentence*	*F. Reduction ratio (100 E/D)*
1	17	3	17.6	2.00	1.00	50.0
2	60	23	38.3	3.43	1.61	46.8
3	17	11	64.7	4.18	1.91	45.7
4	3	3	100.0	8.00	4.67	58.3
Total	97	40	41.2	3.88	1.88	48.4

Sentences are from "Japanese–English Sentence Equivalents," Asahi Publishing Co.

In Table 4, columns B, D, E, and F correspond to the cases where our method could differentiate parses and reduce ambiguity. The hit ratio indicates that our method was effective in about 50% of cases where a sentence has two or three predicates, which is quite typical and thus important. In the remainder of the data, all parses get the same score. The reduction ratio shows the percentage of the average number of parses with the same highest score. It can be seen that this ranking algorithm is effective in that it reduces ambiguous trees by about half. In other words, our algorithm produces only one or two parses for these typical sentences.

Our method deals with argument–predicate attachment. There are various other kinds of causes of ambiguity, such as NP–NP modification and words with multiple parts of speech. The sentences which were not differentiated had only ambiguities of types other than argument–predicate attachment. The differentiated sentences had other ambiguities; however, our scoring method was able to handle them. On the basis of these experimental results, we believe that we can deal with the remaining ambiguities in accord with this method.

• *Extracting modal information*

Although the main purpose for introducing valency theory and the semantic features is to provide preferences for ambiguous parses, the instantiated case-frame information is also useful for application systems. Modal information—aspect and mood, for instance—is another example which is both useful for the application and domain-independent. Therefore it is the responsibility of our parser to extract this kind of information from a sentence structure.

The following rule is assumed by many to represent the highest level of Japanese sentence structure [13, 26]:

$S \rightarrow P + M,$

where S = sentence, P = proposition, and M = modality. P consists of main verb, case elements (usually NP + case particle), and adverbial expressions; valency case frames account for quasi-semantic information for P in our parser. As in English, M is expressed by auxiliary verbs.

Although many Japanese auxiliary verbs have more than one meaning, it is known that certain meanings are not used with certain types of modified verbs. We studied such combinations in detail and developed a set of rules to exclude impossible readings by analyzing the features attached to modifying auxiliary verbs and modified verbs. At the same time, the rules try to give all information which can be inferred concerning tense, aspect, and mood.

Each auxiliary verb is associated with a rule that represents its meaning. Using the MODAL features described above, it is coded as a mapping from the meaning of the sequence of words already processed, to a new meaning. For example, the meaning of the auxiliary verb *teiru* is written as shown in **Figure 9**. This rule says that the word *teiru* follows the grammatical category V, produces category V, and that when the input V has the feature +CH(ange), the aspect information of the resultant V is either STATE, RETRO(spective), or ITERA(tive); STATE is preferred to the other two; and so on.

For example, the Japanese expression *aruiteiru* ['walking' = *aruku* ('walk') + *teiru*] is analyzed as follows:

```
'aruku': V(+ACT,+CONT,+PROG) + 'teiru'
        --> 'aruiteiru': V(aspect=PROGR>RETRO,ITERA)
```

The framework is also useful for some types of the modality-scoping problem. If there are two Vs, say V_1 and V_2, preceding an auxiliary verb, the scope of the auxiliary verb becomes a problem. Then, if the rule corresponding to the auxiliary verb does not allow its modification of V_1, the scope is uniquely determined as V_2.

Muraki [28] tried to formalize auxiliary-verb meanings as rules in a way similar to ours. However, this was not intended to produce any useful information for applications. Moreover, his method assumed numerous interactions between P and M in the sentence scheme described above, and consequently was complex and redundant. The clear separation of P and M in our parser made it more powerful as well as more concise.

```
'teiru': V --> V
      +CH                    --> aspect=STATE>RETRO, ITERA
      +ACT, +CONT, +PROG --> aspect=PROGR>RETRO,ITERA
      +ACT, -CONT            --> aspect=RETRO>ITERA

+CH, +ACT..: semantic information of verbs
STATE: state after V,
PROGR: progressive, RETRO: retrospective, ITERA: iterative

">" and "," represent strength. A>B means A is stronger than B.
```

Figure 9

Example of auxiliary-verb rule.

(a) *mario ga orenji wo taberu*
('Mario') ('orange') ('eat')
('Mario eats an orange')
(b) *orenji wo mario ga taberu*

Figure 10

Example of free word order.

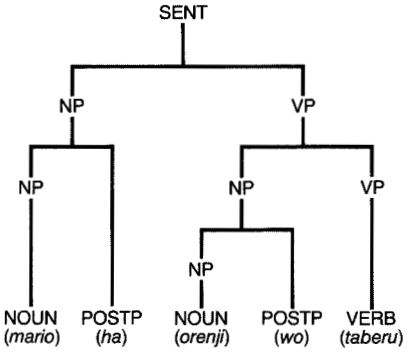

Figure 11

Phrase structure tree for Figure 10(a).

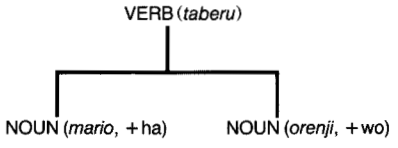

Figure 12

Dependency structure for Figure 10(a).

• *Dependency structure*

We have adopted dependency structures (DS) for the structural description of Japanese sentences, rather than phrase structures (PS), which are more commonly used.

Dependency theory was proposed by Hays [29, 30] in order to represent the structure of sentences. It corresponds to the operator–operand formalism of Harris [31]. A DS is a tree with only words as nodes and their dependencies as links. In contrast, a PS is a tree with nonterminal symbols for phrases which bind its constituents. **Figure 10** gives two possible word orders for a Japanese sentence meaning 'Mario eats an orange.' **Figures 11** and **12** give PS and DS representations, respectively, of the sentence shown in Figure 10(a). In DS, *mario* is directly linked to *taberu*; in PS, *mario* and *taberu* are bound through many nonterminals (NPs and VPs). We have, however, introduced VPs for important practical reasons. (It should be noted that there is disagreement over whether Japanese in fact has a VP node [15, 32, 33].)

As shown in Figure 12, in our implementation of DS, content words (noun: *mario*, *orenji*; verb: *taberu*; adjective. . .) are represented as node labels, while function words (particle: *ha*, *wo*; auxiliary verb. . .) are represented as features which are prefixed with a + sign.

We have two major reasons for adopting DS:

1. An ordinary PS is by nature the trace of rule application, so it is not adequate in some cases for structural description. There should be computed structure—i.e., structural description—different from rule structure. DS gives a useful computed structure because it is well suited for nonconfigurational (i.e., permitting relatively free word order) languages like Japanese. There are, in Japanese, many case elements before a verb, and they can be interchanged freely. For example, the two sentences in Figure 10 have the same meaning except for noun-phrase focus. There are two ways to describe nonconfigurationality in context-free-grammar terms: One is to set up rules for all possible combinations of word order; the other is to use a binary rule whose shape is AB → C and to apply it recursively. The former is not

(a) *kyou no atusa ha kakubetu da*
('today') ('heat') ('extreme')
noun postp noun postp adj aux
(b) 'Heat of today is extreme'
(c) *It is extremely hot today.*
pron be adv adj adv

Figure 13

Example of translation.

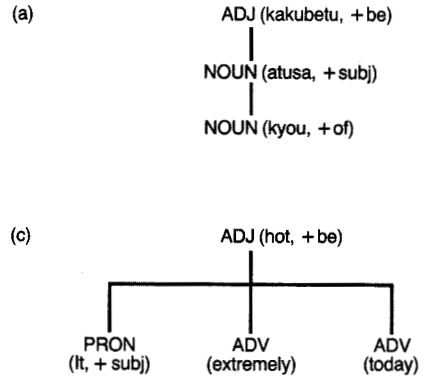

Figure 14

Dependency structures for (a) and (c) of Figure 13.

practical because the number of combinations is large. The latter is practical, but the structure generated is unnatural because the extra VP nodes resulting from binary rules are ultimately unnecessary linguistically and for some purposes actually cause problems. Therefore, we convert PS to DS, which treats every case-element impartially (see Figure 12) and for that reason is suitable for the representation of nonconfigurational languages.

In order to convert PS to DS, the context-free-grammar rules have been augmented with three features: PRMODS (premodifier list), HEAD, and PSMODS (postmodifier list). These three features, in effect, encode the dependency relations. The HEAD feature values correspond to governer and the others to dependents; thus, conversion is straightforward.

2. DS is suitable for a variety of applications such as machine translation, question answering, and information retrieval [1, 34, 35]. DS is less hierarchical, so it is easy to find a specified substructure and to modify it. DS also requires a smaller amount of memory and time for such processes as pattern matching.

In **Figure 13**, an example of machine translation from Japanese to English is provided. State-of-the-art machine-translation systems have usually adopted the so-called transfer approach, which has three complicated phases: analysis, transfer, and generation [1, 36, 37]. Because Japanese is quite different in various aspects from English, in order to make the transfer phase tractable it is important to get a more language-neutral representation of the information in a Japanese sentence. We have therefore adopted DS as the input to transfer (i.e., the output of analysis).

Figure 13 exemplifies the fact that if we translate a Japanese sentence (a) preserving lexical and syntactic properties as much as possible, we will not produce a normal English sentence (b). In order to get a correct, idiomatic translation (c), we must change both lexical and syntactic structures substantially. In other words, we have to change the part of speech of the three words *kyou*, 'today' (NOUN → ADV); *atusa*, 'hot' (NOUN → ADJ); and *kakubetu*, 'extremely' (ADJ → ADV). These lexical changes naturally induce various sorts of syntactic changes as well.

But, since DSs have no nonterminals (**Figure 14**), these differences, relative to PS, are minimized.

5. Status

The parsing portion of the syntactic-analysis component is written entirely in Kanjified PLNLP. The remainder of the syntactic-analysis component is written in Kanjified Lisp/VM. We started writing the grammar rules in May 1985, and are continuously enhancing the grammar. Currently, there are 291 grammar rules with restrictions. The lexicon, which comprises about 100000 words, was initially transferred from a Kana-to-Kanji conversion dictionary [38], and necessary information, such as case frames and semantic features, has been added.

The sentence analyzer is now being used for prototype development in both the Japanese-to-English machine-translation project and the natural-language front-end project [34] in our laboratory.

6. Concluding remarks

We have presented the design and the current status of a robust Japanese sentence analyzer, which comprises a lexical analyzer and a syntactic analyzer. We have concentrated on achieving both robustness and domain-independence.

There are a number of applications where our sentence analyzer can play a key role. The most obvious ones are machine translation and natural-language interfaces. Less

frequently considered, but still a significant application, is proofreading of Japanese text (cf. [39]), since even spell-checking is impossible for Japanese unless exact word boundaries are identified.

Much work remains to be done. We are continuously enhancing the grammar rules on a trial-and-error basis using a large amount of training text. It will take some time, but we believe this to be the only way to create a practical grammar.

Acknowledgments

The authors would like to thank George Heidorn, Norman Haas, and Martin Mikelsons for Kanjification; and Karen Jensen, Chie Yamanouchi, Hiroshi Kitamura, and Hiroshi Maruyama for their valuable discussions and suggestions. The authors also would like to thank David Johnson and the three anonymous referees. They have kindly taken the time to make detailed comments.

References

1. M. Nagao, J. Tsujii, and J. Nakamura, "The Japanese Government Project for Machine Translation," *Amer. J. Computat. Linguist.* **11,** No. 2–3, 91–110 (1985).
2. H. Nomura, S. Naito, Y. Katagiri, and A. Shimazu, "Translation by Understanding: A Machine Translation System LUTE," *Proc. COLING '86* (Proceedings of the 11th International Conference on Computational Linguistics), Bonn, West Germany, August 25–29, 1986, pp. 621–626.
3. T. Tsutsumi, "A Prototype English–Japanese Machine Translation System for Translating IBM Computer Manuals," *Proc. COLING '86*, Bonn, West Germany, 1986, pp. 646–651.
4. K. Jensen, "PEG 1986: A Broad-Coverage Computational Syntax of English," unpublished ms; available from the author, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.
5. K. Jensen, G. E. Heidorn, S. D. Richardson, and N. Haas, "PLNLP, PEG, and CRITIQUE: Three Contributions to Computing in the Humanities," *Proceedings of the Conference on Computing in the Humanities*, University of Toronto, Ontario, April 15–18, 1986, pp. 183–193.
6. S. Mizutani, "Kokubunpou Sobyou (Sketch of Japanese Grammar)" (in Japanese), *Bunpou to Imi I (Syntax and Semantics I)*, S. Mizutani, Ed.; published by Asakura-shoten, Tokyo, 1983, pp. 1–80.
7. S. Yoshida, "Morphology Analysis," *Japanese Information Processing* (in Japanese), M. Nagao, Ed., The Institute of Electronics, Information and Communication Engineers, Tokyo, 1984, pp. 86–113.
8. M. Nagao, J. Tsujii, A. Yamagami, and S. Tatebe, "Data-Structure of a Large Japanese Dictionary and Morphological Analysis by Using It" (in Japanese), *Trans. Info. Proc. Soc. Jpn.* **19,** No. 6, 514–521 (1978).
9. N. Nomura and K. Mori, "A Kanji–Kana Translation System" (in Japanese), *Trans. Inst. Electron. Info. & Commun. Engineers* **J66-D,** No. 7, 789–795 (1983).
10. M. Okochi, "Backtracking-Free Grammatical Analysis for Phrase-Based Kana-to-Kanji Conversion" (in Japanese), *Trans. Info. Proc. Soc. Jpn.* **24,** No. 4, 389–396 (1983).
11. Y. Yamada, *Nihonbunpougaku Gairon (Introduction to Japanese Grammar)* (in Japanese); published by Houbunkan-shuppan, Tokyo, 1936.
12. D. Matsushita, *Kaisen Hyoujun Nihonbunpou (Revised Standard Japanese Grammar)* (in Japanese); published by Bensei-sha, Tokyo, 1974.
13. M. Tokieda, *Nihonbunpou Kougo-hen (Contemporary Japanese Grammar)* (in Japanese); published by Iwanami-shoten, Tokyo, 1950.
14. S. Hashimoto, *Kokubunpou Taikeiron (Systematic Japanese Grammar)* (in Japanese); published by Iwanami-shoten, Tokyo, 1959.
15. K. Inoue, *Henkeibunpou to Nihongo (Transformational Grammar and Japanese)* (in Japanese); published by Taishu-kan, Tokyo, 1976.
16. T. Gunji, *Japanese Phrase Structure Grammar*, D. Reidel, Dordrecht, the Netherlands, 1987.
17. G. Gazdar and G. Pullum, *Generalized Phrase Structure Grammar: A Theoretical Synopsis*, Indiana University Linguistics Club, Bloomington, 1982.
18. G. Barton, R. Berwick, and E. Ristad, *Computational Complexity and Natural Language*, MIT Press, Cambridge, MA, 1987.
19. T. Winograd, *Language as a Cognitive Process*, Addison-Wesley Publishing Co., Reading, MA, 1983.
20. G. E. Heidorn, "Natural Language Inputs to a Simulation Programming System," Ph.D. Dissertation, Yale University, New Haven, CT, 1972.
21. D. T. Langendoen and M. Barnett, "A Linguist's Introduction to PLNLP," unpublished ms; available from D. T. Langendoen, Program in Linguistics, Graduate Center, City University of New York, 33 W. 42nd St., New York, NY 10036.
22. J. Kimball, "Seven Principles of Surface Structure Parsing in Natural Language," *Cognition* **2,** 15–47 (1973).
23. L. Frazier and J. Fodor, "The Sausage Machine: A New Two-Stage Parsing Model," *Cognition* **6,** 291–325 (1979).
24. M. Ford, J. Bresnan, and R. Kaplan, "A Competence-Based Theory of Syntactic Closure," *The Mental Representation of Grammatical Relations*, J. Bresnan, Ed., MIT Press, Cambridge, MA, 1981.
25. Y. Wilks, "An Intelligent Analyzer and Understander of English," *Commun. ACM* **18,** No. 5, 264–274 (1975).
26. C. J. Fillmore, "The Case for Case," *Universals in Linguistic Theory*, E. Bach and R. T. Harms, Eds., Holt, Rinehart & Winston, New York, 1968, pp. 1–88.
27. T. Ishiwata, "Ketsugoukakara Mita Nihonbunpou (Japanese Grammar and Valency Theory)" (in Japanese), *Bunpou to Imi I (Syntax and Semantics I)*, S. Mizutani, Ed.; published by Asakura-shoten, Tokyo, 1983, pp. 81–134.
28. M. Muraki, "Description Using GPSG" (in Japanese), *Proceedings of 1987 University Science Symposium*, Tokyo, 1987, pp. 11–22.
29. D. G. Hays, "Dependency Theory: A Formalism and Some Observations," *Language* **40,** 511–524 (1964).
30. D. G. Hays, *Readings in Automatic Language Processing*, American Elsevier, New York, 1966.
31. Z. Harris, *A Grammar of English on Mathematical Principles*, John Wiley & Sons, Inc., New York, 1982.
32. J. Hinds, *On the Status of the VP Node in Japanese*, unpublished manuscript circulated by Indiana University Linguistics Club, Bloomington, 1974; available from M. Morohashi.
33. M. Saito and H. Hoji, "Weak Crossover and Move α in Japanese," *Nat. Lang. & Linguist. Theor.* **1,** No. 2, 245–259 (1983).
34. H. Maruyama and H. Watanabe, "A Discourse Analysis Technique for a Natural Language Interface System," *Proceedings of COMPSAC* (The Eleventh Annual International Computer Software and Applications Conference, IEEE Computer Society), Tokyo, 1987, pp. 578–585.
35. L. Hirschman, R. Grishman, and N. Sager, "Grammatically-Based Automatic Word Class Formation," *Info. Proc. & Management* **11,** 39–57 (1975).
36. R. Johnson, M. King, and L. des Tombe, "EUROTRA: A Multilingual System Under Development," *Computat. Linguist.* **11,** No. 2–3, 155–169 (1985).
37. W. Bennett and J. Slocum, "The LRC Machine Translation System," *Computat. Linguist.* **11,** No. 2–3, 111–121 (1985).
38. T. Fujisaki, M. Okochi, and M. Morohashi, "Kana to Kanji Conversion Text Input of KOTODAMA Document System" (in Japanese), *Trans. Info. Proc. Soc. Jpn.* **23,** No. 1, 1–8 (1982).

39. Koichi Takeda, Emiko Suzuki, Tetsuro Nishino, and Tetsunosuke Fujisaki, "CRITAC—An Experimental System for Japanese Text Proofreading," *IBM J. Res. Develop.* **32,** No. 2, 201–216 (1988, this issue).

Received March 16, 1987; accepted for publication November 24, 1987

Naoko Maruyama *Tokyo Woman's Christian University, Department of Japanese Literature, 2-6-1 Zenpukuji, Suginami-ku, Tokyo 167, Japan.* Ms. Maruyama received her B.A. and M.A. in Japanese linguistics from Tokyo Woman's Christian University in 1981 and 1983, respectively. She was a member of the Japanese Processing Project at the IBM Tokyo Research Laboratory, where she worked as a researcher from 1983 to 1987. Her research interests include Japanese linguistics (syntax, semantics, and morphology) and natural-language processing.

Masayuki Morohashi *IBM Japan Ltd. Tokyo Research Laboratory, 5-19 Sanbancho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Morohashi received his B.E. in electrical engineering and his M.E. in computer and logical circuits (majoring in electrical engineering) from Waseda University, Tokyo, in 1972 and 1974, respectively. He joined IBM in 1974, and has worked as a researcher in computational linguistics. He is currently the manager of the Japanese Processing Project at the Tokyo Research Laboratory, where he has worked since 1983. Mr. Morohashi's research interests include Japanese linguistics (syntax and morphology), discourse analysis, and natural-language processing in general.

Shigeki Umeda *IBM Japan Ltd. Tokyo Research Laboratory, 5-19 Sanbancho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Umeda received his B.E. and M.E. degrees in industrial engineering (majoring in applied statistics/operations research) from Waseda University, Tokyo, in 1980 and 1982, respectively. He is a researcher in the Japanese Processing Group at the IBM Tokyo Research Laboratory, where he has worked since 1982. Since joining IBM, Mr. Umeda has been involved in the development of methods and tools to improve the productivity of natural-language-processing systems. His research interests include information retrieval, natural-language processing, and artificial intelligence.

Eiichiro Sumita *IBM Japan Ltd. Tokyo Research Laboratory, 5-19 Sanbancho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Sumita received his B.E. and M.E. in computer science from the University of Electro-Communications, Tokyo, in 1980 and 1982. He is a researcher in the Japanese Processing Project at the IBM Tokyo Research Laboratory, where he has worked since 1982. Mr. Sumita's research interests include machine learning and natural-language processing, especially machine translation.