An Overview of NLP in ALICE-chan

**D.A. Evans, L.S. Levin, M. Thurn, S.K. Handerson,
K. Horiguchi, K.C. Scarpinatto, D. Varma**

December 1993

Report No. CMU-LCL-93-3

Laboratory for
Computational
Linguistics

139 Baker Hall
Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA 15213

Contents

List of Figures

List of Tables

1 Character-Independent Processing

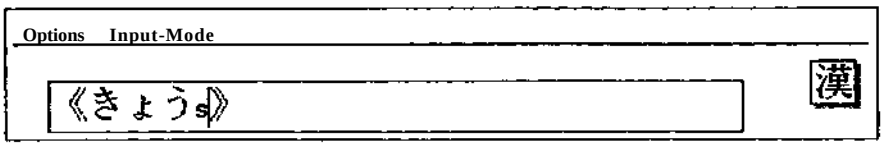
One of I lie oaiTiost design decisions for AUC'R-Hian was that the system will operate on input in *himgana, kanji,* or *romaji.* The NLP system is designed to accept any mixture of *romaji* and Japanese characters as input, and returns a mixture of English and Japanese in its output. Thus, the student can type her answer in *romaji, hiragana, kanji,* or any mixture of these. Feedback to the student is presented as English sentences sprinkled with references to the student's response (possibly in Japanese characters).

Vox- inputting Japanese characters, ALICE-chan utilizes the common Japanese input method whereby the user types *romaji* on a standard English keyboard, the *romaji* is converted to *hirafjana* automatically as the user types, and the user can request that I lie *hirafjana* be converted to *kanji* by pressing a special key. Other functions include scrolling through alternative *kanji;* and stretching or shrinking the length of word segments, which controls how many *kana* syllables will be grouped together for conversion into *kanji.*

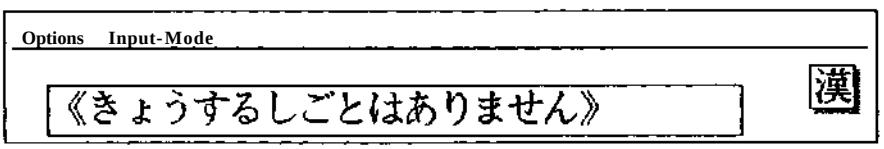
liomaji-to-hiragava conversion is carried out by a finite state transition process written in C. *Hiraf/ana-to-kanji* coversion is carried out by external function calls to jserver, a public domain *kanji*-conversion server.

English	Tokyo	5 characters, 5 bytes
romaji	toukyou	7 characters. 7 bytes
hiragana	とうきょう	5 characters, 10 bytes
kanji	東京	2 characters. 4 bytes

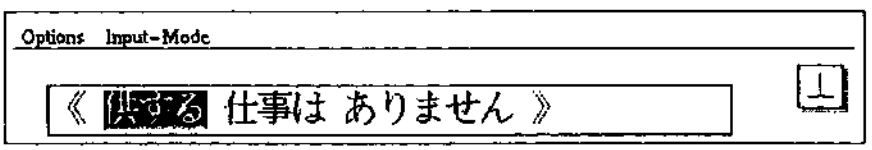
Table I: Tlio Japanese word for "Tokyo" in various character formats.



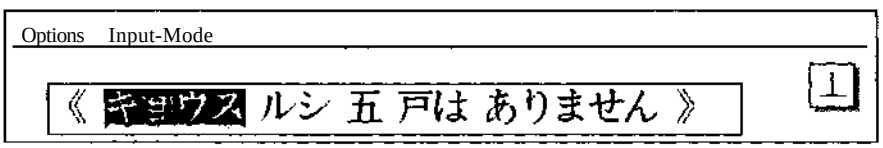
The user has typed "kyous".



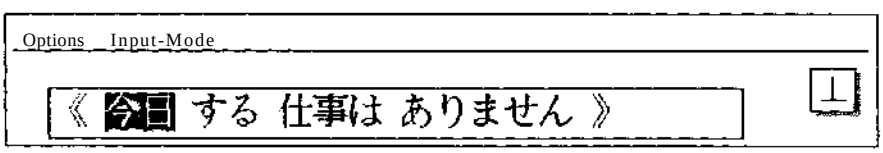
The user has typed "kyousurushigotohaarimasen".



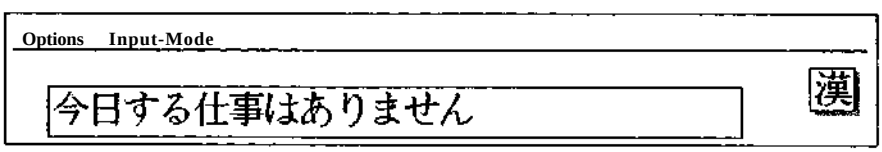
The user has pressed the spacebar
and jserver has given its first guess at segmentation and kanji selection.



The user has typed "Ctrl-i" to shrink the length of the first segment by one kana,
and jserver has given its first guess at kanji selection for the new segmentation.



The user has typed "Ctrl-i" again to shrink the length of the first segment by one more kana,
and jserver has given its first guess at kanji selection for the new segmentation.



The user has typed "Return" to solidify the entire sentence of kanji.

Figure 1: An example of Japanese text entry.

2 The NLP Lexicon

The NLP lexicon contains information Mint allows I lie system (1) to recognize words in all of I heir morphological variants and (2) to identify syntactic and semantic features oi the word.

Each lexical entry consists of two main parts, one is a list of keys in *romaji, kana,* and *kanji,* and the other is a list of syntactic and semantic features. The keys indicate possible orthographic realizations of a word. For example, the word in Figure 2 can appear as *hon* (romaji). *ll/x,* (hiragana). or * (kanji). If a word can be written in more than one way, all the possible keys are listed as an embedded list. For example, the word that means "violin" (Figure .'{) can be written in two ways in *kaiakana.* therefore there are two *kalakana* strings embedded in the key list. Lexical entries for proper names often include many *kanji* variants, as shown in Figure 4.

```
(("hon" "J5//[1] "*")
 (S "book" M (hon) COUNT volume CAT noun INANI-SUBJ +
  BL + R (FINAL PART) L (BEGIN G-HON)))
```

Figure 2: The NLP lexicon entry for the Japanese word "hon" ("book").

```
(("baiorin" ("(£KU-5 *) A," "'v-f.+ '.J>" "*fr i t 0 > "))
 (S "violin" M (baiorin) COUNT machine CAT noun INANI-SUBJ +
  BL + R (FINAL PART) L (BEGIN)))
```

Figure *4:* The NLP lexicon entry for the Japanese word for "violin." Notice the multiple strings n«pr<\scMirni£ f IM» liir;ig;iiia ;UMI katakami versions of the word.

```
(("yosiko" ".til" (".tl-f" "T?1'" "tfH[1]" "R^r" "#-?" "H^f" "ftf"))
 (S "Yoshiko—girl's name" M (yosiko) CAT noun INANI-SUBJ +
  ANIMATE + PROPER + HUMAN + BL + L (BEGIN) R (SAN NAME FINAL PART)))
```

Figure I: The NLP lexicon ^ntry for the Japanese proper noun "Yoshiko." Notice I he multiple strings lepresc Miliuu; 1 he various kanji versions oI the word.

The second part of a lexical entry contains a list of feat lire-value pairs. The feature S stands for *sense.* Its value is a short English gloss indicating the meaning of the word. The feature M (morpheme) is a *romaji* spelling of the citation form of the morpheme.

Morphological analysis is guided by the special features L and R. whose values are names of si ales in a non-deterministic finite stale transition network. The value of L is a sel of possible* starting slates for a transition and the value of R is a set of possible ending states. The keys of the lexical entry are the possible input strings which license the transition. Figures 5 through 7 show lexical entries involved in the analysis of several forms of the verb *kaku* (write). For example, processing of *kakimasu* starts at the initial state (BEGIN), and uses the input *ka* to get to state K. From state K. the input *ki* allows a transition to stale MAS. From slate MAS. the input masu allows a transition to state FINAL.

The addition of the lexical entries shown in Figures 8 through 11 allow the system to

```
((Mka^M "*»" "<")
 (S "write" M (ka) WO-PATIENT + NI-GOAL + COMP + CAT verb BL +
  ROOTKEYS dummy UNUM dummy SUFKEYS ("ku^M "<" ^M<n) L (BEGIN HON) R (K)))
```

Figure 5: The NLP lexicon entry for the stem of the Japanese verb "kakn" ("write").

```
(("KI" "*") (M (ki) L (K) R (P-NI MAS ASPV VHON DESIDER)))
```

Figure (3: The NLP lexicon entry for the Japanese verb morpheme [b]'kr.

```
(("MASU" "if") (M (masu) STYLE formal TENSE present L (MAS) R (FINAL)))
```

Figure* 7: The NLP lexicon entry for the Japanese present formal verb morpheme [w]masif\

process *kaiicimasu* ("write/* gerund + auxiliary "to be") and *kaila* ("write/* past informal), which illustrate the changing of the final stem consonant *-k-* to *-/-* before the gerund *(-tr-)* and past tense *(la)* morphemes. Processing of these words also starts in the initial stair, using the input *ka* for a transition to state k. Then, the input / allows a transition to stair T-GER or T-PAST, etc. Notice the treatment of the stem allomorphs *kak-* and *kai-*. The lexical (Mitrirs for *-ki-* and *-/-* both start in state k. but they end in different stairs, rrllrrl in<; thr different possible continuations of each stem.

```
(("I" 'HV«) (M (i) L (K) R (T-GER T-PAST)))
```

I'ignir S: The NLP lexicon entry for < ho stom of the Japaneso ninrpltomo *1."

```
(("TE" »X") (M (te) GERUND + ADVERBIAL - L (T-GER) R (FINAL REQUEST WA MO GER-ASP)))
```

Hgnro f): Thp NLP lexicon ontry for HIP .Japanese vorb gennul morphomo "tr".

```
(("TA[11] "it") (M (ta) STYLE informal TENSE past L (T-PAST) R (FINAL TENT)))
```

Figuro 10: The NLP lexicon entry for the Japanese past informal verb morpheme "ta"\

```
(("I" "v>")
 (M (i) A (doing) GERUND - L (GER-ASP)
  R (V-NEG MAS DESIDER V-CAUS PL-PRES IMP T-PAST T-GER V-PASS POTEN V-VOLIT V-PROV)))
```

I'ignro | 1: Tlie NLJ^ lexicon entry for the stem of the Japanese auxiliary verb '"inf.

Notice also, that lexical entries do not necessarily correspond to whole morphemes. *Ka-. -ki-.* and *-/-.* lor example, are parts of the *kak-jkai-* morpheme, and *-/,•/-* might even span morpheme boundaries if the analysis of *kakimasu* is *kak+imasu.* Lexical entries actually

correspond more closely to Japanese orthography than to a linguistic morphological analysis. Each entry corresponds to a whole number of syllables which are the pronunciations of *kanji* or *hana* characters. Since a syllable can be part of a morpheme, or can span morpheme boundaries, character boundaries do not correspond to morpheme boundaries. More details about morphological processing are found in the chapter on morphology.

Tlirrr ate currently about 1700 entries in the ALlCE-chan NLP Lexicon.

3 Morphological Analysis

The morphological processor is conceptually an interpreter for a nondeterministic finite state automaton, specified by the lexicon. This allows the process to be simple, fast, and flexible (lexicon-driven). This simplicity and speed motivated combining morphological processing with segmentation as a single initial pass over the input string. For sentences with successful parses, the parser is limited to determining the grammaticality of a segmentation, instead of also having to determine a sentence's segmentability. In this section, we will refer to "morphemes" and "words" without the double-quotes, a morpheme simply being a string entered in the lexicon and a word the result of a successful morphological analysis.

Morphemes entered in the lexicon describe a nondeterministic finite-state automaton (NDFA) where the transitions are the morphemes. Each morpheme also specifies the allowable prefix states and the potential result states. Morphemes may be entered multiply in the lexicon to express different sets of applicable contexts and result contexts. There are special states signifying the initial state (morphemes that can begin a word) and final state (morphemes that can end a word). A word, according to our morphological process, is any sequence of morphemes that cause the NDFA to reach a final state.

During analysis, a simple feature-value pair *feature structure* is built from information associated with the morpheme in the lexicon. This information reflects syntactic information and is converted into a form used by the parser. This information can also be used later, by application-specific processes (for instance, to facilitate user feedback). Since the processor (or NDFA) lacks the ability to do even simple checks based upon the built-up features and values, this information is not used directly to enforce morphological constraints, but could be so used by the parser.

Currently, the runtime lexicon is stored in a hash table indexed by all morpheme strings and their prefixes, but other implementations are possible (for instance, a trie). All that is necessary is to know whether a given prefix string can be continued to form a morpheme. A hash table implementation was chosen because the lexicon contains *kanji* and *hiragana* as well as *romaji*, and so a space-efficient trie representation would be difficult, but we have not studied this to any great extent.

4 Segmentation

A problem wil-li NLP for Japanese is Hiat the written language does nol have spares separating words as in most Western languages. One of the **first,** problems a parser (human or machine) for Japanese faces is to determine a sentence's word boundaries. We have found that the left-to-right nature of our morphological analysis, together with a few assumptions (or observations) concerning the input sentence, yield a "knowledge-poor*'ˋ (i.e. simple*) algorithm which is nonetheless very efficient. The advantage of segmenting prior to parsing is that many words can be ruled out from further consideration, since they cannot be part of a complete[1] sentence parse.

Obviously, in the absence of misspellings, any correct parse of the input sentence must divide the input into non-overlapping words. If we first find all possible words that begin at the first position, the second word in the intended interpretation (parse) must end after one of these possible first words, etcetera. Also, the intended interpretation must have its last word end at the end of the sentence: there are no "extra characters" at the end. Note that there is an implicit assumption that the segmenter knows whether a given string can be continued to form a new word: otherwise the "find all possible words" step might have to look at the entire remaining sentence, for every character position. These are the main assumptions we make about- the input: all other information is in the morphology and the lexicon.

These constraints translate into an algorithm with a single left-to-right word generation phase and a right-to-left elimination or filtering phase. Note that words need not he generated for every character position: only those "reachable" from the beginning of the sentence. 'The liltcring phase is simply the right-to-left analogy of the generation phase, keeping in mind that we do not need to add words that are not ¨reachable¨ from the beginning of the sentence.

All interesting but not crucial last step constructs the iuput to the parser, not as words or characters, but- ¨chunks¨. A ¨chunk*ˆ is a sequence of characters that- is never divided between words in any segmentation. Note that passing words to the parser would be difficult or impossible (except in a chart parser), since there may be several segmentations with different woid boundaries. Parsing a sequence of ¨chunks¨, however, is concept uallv similar to parsing a sequence of characters with an LR parser. Also, the information needed to construct the "chunks"' is already present in the segmentation data structures.

Hnfort unatelv, this approach does not work well in the ca.se where there are spelling errors. Iİ the spelling error is such that there are correct segmentations, this would make such spelling errors harder to detect. A good approach might be to first assume no spelling errors, and if either there are no segmentations, no parses for the resulting segmentations, or many or serious grammatical or exercise-specific errors, then a more complex and time-consuming process ensue, that starts by trying to identify expected words in the input.

5 Analysis of Syntactic Structure

The goal ofsynta.ct.ic analysis is to identify (I) the predicate of each clause. (2) the predicate's syntactic and morphological features, and (3) a grammatical function and/or semantic role for every other eleinement of the clause. An example of a sentence and the raw form of its analysis are shown in Figures 12 and 13.

Syntactic analysis is used during exercise authoring and during student exercises. In authoring, it is used for analyzing correct answers so that the syntactic analysis of the answers can be annotated by the author, and stored for later comparison to student answers. In student exercises, syntactic analysis is used for analyzing student answers so that they can be compared to previously stored authored answers.

恵子さんが寿司を食べました。

Figure 12: A Japanese sentence meaning "Keiko ale sushi."

```
(((CAT VERB) (SUFKEYS (*LIST* "ru" "S" "&")) (BL +) (WO-PATIENT +) (S #:EAT)
  (UNUM 3) (ROOTKEYS (*LIST* "tabe" "fz'<" "&'<")) (TENSE PAST)
  (M (*LIST* TABE MASITA)) (STRKEYS (*LIST* "j£'<" "&ltl"))
  (PARTSTRKEYS (*LIST* "&K" "&ltz")) (ALLKEYS (*LIST* #:G13417 #:G1598))
  (MFSS (*LIST* #:G13418 #:G1599)) (END-CHUNK 3) (BEGIN-CHUNK 3)
  (STR "&'<&lfz") (END-PHRASE NIL) (DICT-FORM "•&'<.£>") (BEGIN-PHRASE 0)
  (SUBCAT
   ((GA ((ACTOR ANIMATE))) (WO ((ACTEE ANY))) (TO ((ACCOMPANIMENT ANIMATE)))
    (DE ((LOCATION PLACE))) (NI ((TIME-AT TIME-NI)))
    (KARA ((ACTOR ANIMATE) (TIME-FROM TIME))) (MADE ((TIME-TO TIME)))
    (WA ((ACTEE ANY) (ACTOR ANIMATE))) (MO ((ACTEE ANY) (ACTOR ANIMATE)))))
  (STYLE FORMAL)
  (ACTOR
   ((BL +) (HUMAN +) (PROPER +) (ANIMATE +) (INANI-SUBJ +)
    (S #: IKEIKO—GIRL'S NAMED (TITLE (*LIST* MR./MRS./MS./MISS)) (RESTRICT +)
    (M (*LIST* KEIKO SAN GA)) (STRKEYS (*LIST* "M?" "S/v" "#"))
    (PARTSTRKEYS (*LIST* " S T " "J/u" "-£»"))
    (ALLKEYS (*LIST* #:G5744 #:G11652 #:G11400))
    (MFSS (*LIST* #:G5745 #:G11653 #:G11401)) (END-CHUNK 1) (BEGIN-CHUNK 0)
    (STR "gH1*/,#") (BEGIN-PHRASE 0) (END-PHRASE NIL) (CAT NOUN) (PART GA)
    (ROLES ACTOR)))
  (ACTEE
   ((BL +) (INANI-SUBJ +) (COUNT OBJECT) (S #:SUSHI) (RESTRICT +)
    (M (*LIST* SUSI WO)) (STRKEYS (*LIST* "3fW]M "&"))
    (PARTSTRKEYS (*LIST* M^?W]" "-£")) (ALLKEYS (*LIST* #:G8724 #:G11414))
    (MFSS (*LIST* #:G8725 #:G11415)) (END-CHUNK 2) (BEGIN-CHUNK 2)
    (STR fl2FWJ£") (BEGIN-PHRASE 2) (END-PHRASE NIL) (CAT NOUN) (PART WO)
    (ROLES ACTEE)))
  (PERIOD +)))
```

Figure 13: The raw feature-structure resulting from parsing the sentence above.

6 The Parser

SvnfaHir analysis is performed via a neutralized J/R parser wil-li pseudouuifVaf ion. (The current, version of tire parser is written in Lisp.) The parser implements a lexical-functional grajiunar (LFG). However, in designing the syntactic analyzer, we have had to confront several hard problems in NLP—illformed input, ambiguity, and robustness of coverage—which have led us to augment and modify a basic LFG approach.

7 The Grammar

The grammar is a set of about 150 rules written in a pseudo-unification formalism, with some calls to Lisp functions. The grammar covers basic mono-clausal sentences and some sentences involving embedded complement clauses, gerund clauses, and sentential modifiers of nouns. A range of noun phrase structures are also allowed including noun phrases with a-djoctives, modifiers marked with *no.* and conjuuets marked with *to* and *j/n.* (-overage is siidicienf for a first-year Japanese course, with some additional coverage of more complex structures. Figure 11 shows a single grammar rule.

```
;;; SURU-VERB
;;; The following two rules are designed to give the same f-structure to verb
;;; phrases with SURU-VERB both with or without "wo".
;;; Nouns that takes the light verb ("suru") is marked by "suru" feature in
;;; the lexicon with the meaning of the verbal,
;;; (eg. "kekkon" s "marriage" suru "marry")
;;; (1) NP SURU
(<VP> <==> (<NP> <VP>) ; "benkyou suru"
       (
((x2 suru) =c +)                ; verb is "suru"
((xl suru) = *DEFINED*)          ; noun takes "suru"
((xl part) = UNDEFINED*)
(xO=x2)
((xO previousword) <= xl)
((xO previousword separator) <= * I I)
(xO <= (combined-word-from-word-and-previousword (xO)))
((xO subcat) <= (xl subcat))    joverwrite with the noun's
;subcategorization frame
((xO s) <= (xl suru))  ;overwrite the sense
((xO begin-phrase) <= (xl begin-phrase))
))
```

Figure 1-1: A rule from the parser's grammar.

8 The Mapper

The mapper assigns grammatical functions and/or semantic roles to phrases by determining the best match of phrases with the predicate's subcategorization frame. This is complicated by the fact that student Japanese contains many errors in the use of case marking particles. There is also some ambiguity caused by the use of the topic marker *ira* or the conjunctive particle* *wo* in place of other particles.

The mosl. recent version of the mapper is called from the grammar rules that affaeh noun phrases to sentences. Each phrase can match zero or more slots in a predie:ate\s subcategorization frame on the basis of its particle and its semantic class. During analysis of sttielcnt sentences, because of particle errors, there are often phrases that do not match any subcategorization slots. In this case, the mapper searches for the meaning of the phrase in the authored parse, the function/role of the phrase with the same meaning in the authored parse* is hypothesized as the intended function/role in the student sentence, and an error in particle* use* is flagged. Phrases that still do not have a function/role after this proevduiv are* given the* function EXTRA. Phrases without particles are given the function **NOMINAL.** At this point, each phrase is fagged with zero or more slots if can fill. There may be fwe> phrases thai *aw)* (ill f he same slot, and any given phra.se may be able to fill more than one sle>f. The* mapper then identifies one or more ways of filling the predicate's subcategorizafie>n frame*. pe>ssibly ivlegating more phrases to the extra function.

The* mapper is written in LISP and TransformationKit. a pseudo-unification language' for speciïying translormatieMis of feature structures.

9 The Disambiguator

Boca use ALK'E-rhan currently has only minimal semantic analysis, there can be many unresolved ambiguities in the out-put of syntactic analysis. The disambiguator resolves synatctic ambiguities interatively with the user. First, it detects certain types of ambiguities in the output of the segmentation program, parser, and mapper. Then it formulates questions which are presented to the user one at a time. Sometimes more than one question is formulated to resolve the same ambiguity, so that the user can answer the one that is most clear. Currently, the disambiguator can detect ambiguities in segmentation, homographs, grammatical function/semantic role assignment, nesting of noun modifiers marked with *no* and conjuncts marked with *lo* and *ya.* and attaclinieiit ambiguities involving sentential modifiers of nouns and gerund clauses.

'The disambiguator is written in LISP, while the questions are presented and user input is obtained with cT.

Examples of some disambiguation dialogs are shown in Figure 15.

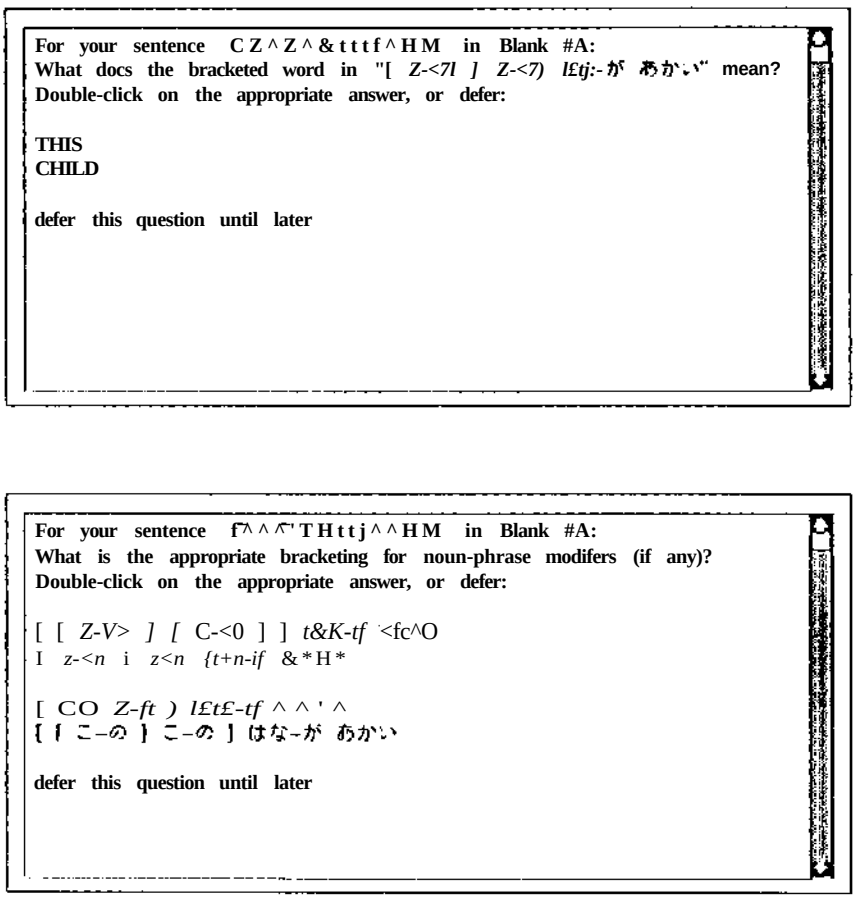


Figure 15: Some disambiguation dialogs.

10 The Matcher

Si ndent responses t hat are well formed may nevertheless he incorrect in a particular context. *Vov* example, they may convey the wrong level of formality or the wrong relationship between speakers, or they may simply express the wrong proposition. For this reason, even when the syntactic analyzer assigns a well-formed structure to a student response, it must be checked lor appropriateness in context. This is the job of the matcher.

'The matcher compares the analysis of a student's response with the analysis of the teacher's response, which was stored during authoring. Errors are reported if required features or phrases are missing, if extra features or phrases are used, or if the values of any features differ.

Some degree of flexibility is inherent in the matcher because it matches parsed analyses of sentences instead of comparing the sentences themselves. In many cases, different sentences with the same meaning have identical analyses. Notable examples of this in Japanese arc sentences that dilfer in word order, but have the same meaning. If the parser assigns the same analysis to two sentences, the matcher will treat them as identical.[1]

The current matcher is written in cT as a recursive string-comparison based process.

'II should ⊱ noted, flmiiɒh. Mint not all synonymous sentences have identical analyses. Our nirrenf rrsrarHi involves ina-easing fk« number of type* of synonymy I hat are rerngnized by the NLP programs.

11 Overview of Error Detection and Feedback

Errors in student responses are detected at three stages. First, during parsing, special grammar rules are designed to parse errorful structures and raise error flags. Second, during mapping, the mapper may detect phrases whose grammatical function or semantic role cannot he determined. Third, during matching, contextually inappropriate features of the studenTs response are detected by comparing (he analysis of the student's response to Mir prr-slored analysis of the authored response.

```
causer  私                                OPT
    HUMAN                                 REQ
    ANIMATE                               REQ
    particle  は                          REQ
    meaning  I
actor=causee  いくお                      REQ
    PROPER                                REQ
    HUMAN                                 REQ
    ANIMATE                               REQ
    particle  に                          REQ
    meaning  IKUO--BOY'S FIRST NAME
thing_acted_on  野菜                      REQ
    particle  を                          REQ
    meaning  VEGETABLES
predicate  食べさせた                     REQ
    CAUSATIVE                             REQ
    meaning  EAT
    PAST                                  REQ
    PRESENT                               ILL
    dictionary-form  食べる
    INFORMAL                              OPT
    FORMAL                                OPT
    EXTENDED-PREDICATE                    OPT
sentence-particle                         OPT
```

Figure 16: Feature-slot for the author's sentence

```
actor=causee  いくお
    HUMAN
    PROPER
    ANIMATE
    particle  を
    meaning  IKUO--BOY'S FIRST NAME
thing_acted_on  野菜
    particle  を
    meaning  VEGETABLES
predicate  食べさせました
    meaning  EAT
    PAST
    dictionary-form  食べる
    CAUSATIVE
    ERRORPART      PART-ERROR ACTOR=CAUSEE WO いくお
    FORMAL
```

Figure 17: Feature-slot for the student's sentence

```
  In blank # A:  You seem to have used the wrong particle on いくお.
[The particle of the actor=causee (いくお) should be に.]
You made the predicate (食べさせました) FORMAL, which is optional.
```

Figure 18: The feedback produced after matching the above two feature-slots