NAIST Natural Language Tools

Yuji Matsumoto
Nara Institute of Science and Technology
matsu@is.aist-nara.ac.jp

Yasuharu Den
ATR Interpreting Telecommunications
Research Laboratories
den@itl.atr.co.jp

Takehito Utsuro
Nara Institute of Science and Technology
utsuro@is.aist-nara.ac.jp

Kiyoshi Yanagi
The Kansai Electric Power Company, Incorporated

Abstract

NAIST-NLT (Nara Intitute of Science and Technology Natural Language Tools) aims at providing a flexible natural language processing environment. The system consists of morphological analyzers for Japanese and English, a compiler of a DCG into a bottom-up Chart parser, a visual interface for showing the partial results of the parsing process, and supporting programs for implementing natural language grammars. Modularity and extensibility are important features of the tools, and various customization is possible by the users.

1 Introduction

Practical natural language processing requires a number of resources, such as dictionaries, grammars, parsers for implementation of runnable systems, and other tools for coping with various linguistic problems. A usable system is indispensable for the development of dictionaries and/or grammars, for designing lexical representation and grammar formalisms, and for building application systems.

NAIST Natural Language Tools (NAIST-NLT) intends to provide an environment for researchers of natural language processing systems to develop and test grammars and dictionaries for langugage processing. Systems for such a purpose

should be modular and neutral, that is, they have to be flexible to implement a variety of systems within them.

The NAIST-NLT consists of morphological and syntactic parsers for general purpose natural language analysis, and a visual interface to encapsulate the implemetation details of the parsing programs from the users. The Japanese morphological analyzer JUMAN is flexible in that most of the components such as definitions of the grammar, inflection types, costs for estimating plausibility, are user-definable. The SAX parser assumes a grammar in a Definite Clause Grammar form [Pereira 80] and translates it into a bottom-up Chart parser. Because of the expressiveness of DCGs, it can handle almost all phrase structure type grammar formalisms. The system also provides a monitor process to observe partial parse results for modifying and comparing them and to work as an interface to other systems. The VisIPS system is a visual interface to phrase structure based parsing systems and shows partial parse results of the parser on a CKY-like triangular table.

The subsequent sections give overviews of the components of the NAIST Natural Language Tools. Although they are meant to be used in an integrated way, each of the compoments can be used as a stand-alone system.

2 Morphological Analyzers

The Japanese morphological analyzer is called JUMAN. The system is implemented in GCC and produces a lattice like structure of Japanese morphemes when given a Japanese sentence. It works as a UNIX filter. Besides, an interface to SICStus Prolog is provided, so that the system is invoked from Prolog and returns a lattice of morphemes back to the Prolog program. The attached dictionary contains about 120,000 entries.

The most important feature of the system is that the basic definition of Japanese morphological grammar system, such as the set of part of speech, inflection rules, and connection rules of morphemes are easily redefined by users. Since a number of Japanese grammars have been proposed and there is no "standard" Japanese grammar, this feature is indispensable for the system to be widely used. The detail of the system is found in [Kurohashi 94] and [Matsumoto 94].

The English morphological analyzer deals with inflection of English nouns, verbs and adjectives. A general purpose procedure is provided for separating a suffix from the root. The list of inflection forms is written in a Prolog code and is easy to modify or rewrite for other languages. After separation of a suffix, the root and the suffix are dealt with as distinct morphemes. Therefore, suffixes should be defined as distinct morpheme in the grammar. For example, if the user defined dictionary contains verbs "try" and "play", then "tries" and "plays" are separated automatically into "try + s" and "play + s". The following sample grammar rule handles those cases.

```
verb([3rd_singular|V]) --> verb(V), [s], {root_form(V)}.
```

Figure 1: Process communication between SAX processes

3 SAX Parsing System

The SAX parsing transforms a Definite Clause Grammar (DCG) [Pereira 80] into a Prolog program that realizes a bottom-up Chart parser. The system is implemented by a collection of Prolog clauses directly derived from DCG rules. They are called SAX clauses. A set of the SAX clauses with the same predicate name corresponds to a grammatical phrase and defines a concurrent process. Parsing is performed through data communication between those processes.

The system is implemented in two levels: The first consists of the transformed grammar rules, and the second works as an interface with other programs as well as the interface to the user.

A number of supporting programs are provided to make users easy to implement their own grammars in the system, e.g., interface programs with the morphological analyzers and an visual interface described in the next section, a unification progam for feature structures.

3.1 Basic algorithm of SAX parser

The underlying algorithm is based on Chart parsing [Kay 80], in which the algorithm is described as an interaction between active edges and inactive edges. An active edge stands for a phrase some parts of which are not fulfilled yet. An inactive edge stands for a complete phrase structure. An active edge is usually expressed by a dot notation, such as $[s \rightarrow np \bullet vp, i, j]$, which indicates the part of the body of the rule up to the dot (in this case, '*np*') has been constructed using $(i+1)$-th through j-th words.

This active edge is expansible by an inactive edge '*vp*' to compose a '*sentence*.' Taking into consideration that a new active edge is in fact produced from an inactive edge, once we get a complete phrase structure (i.e., an inactive edge) there are two tasks to be done related with the inactive edge: 1) To produce all active edges from grammar rules the left-corner of which coincides with the inactive edge. 2) To check active edges that are expansible by the inactive edge.

Given all the grammar rules, we can precompile all the possible tasks of each inactive edge into Prolog clauses. Some simplification is done to gain efficiency. For instance, the dot notation is not used in the original form. Instead, a unique identifier is allocated to each internal position in the body of grammar rules. When an identifier '`id1`' is allocated to the above dot notation, a '*vp*' produces a '*sentence*' when it finds this identifier (`id1`) at the left adjacent position. Suppose the input string is "私 は 花子 に 会った," the initial Prolog goals look like those of Figure 1 where a circle represents a SAX process (a program clause corresponding to a phrase)

Figure 2: Implementation image of meta-processes

and an arrow represents a communication channel. The processes run concurrently using SICStus Prolog's block mechanism. SAX processes are produced in a bottom-up way, corresponding to the construction of larger phrases in the bottom-up Chart parsing algorithm. Once a SAX process is constructed, its own information is thrown into the communication channel right adjacent to the process. Processes looking at the channel can use the information to build up larger phrases. The details of the system are found in [Matsumoto 87].

3.2 Meta-processes

A special procedure called a meta-process is introduced in the system. the process communication then looks like those of Figure 2, where a circled 'p' is a meta-process monitoring a communication channel. Although the difference between Figure 1 and Figure 2 is not conspicuous, note that the words are put into a program "`process`" (abbreviated as `p` in the figure). The system gains the flexibility by modifying the definition of the program "`process`".

Suppose the input sentence is "I love Kyoto," the initial goals of parsing is as follows[1]:

```
process([ie(np([begin(t,[])],[np(true),'I'],
                'I',sngl1st),true)],
        S1),
process([ie(v(S1,[v(true),love],
                love(S,O),S,O,not_sngl3rd),true)],
        S2),
process([ie(np(S2,[np(true),kyoto],
                kyoto,sngl3rd),true)],
        S3),
fin(S3).
```

Meta-process (`process`) has a list of phrases at the first argument (`ie` stands for an 'inactive edge'). `S1`, `S2` and `S3` are shared variables used as communication channels, which are at the second argument of a `process`. `S1` will be instantiated with the (partial) parse results of the phrase that is given at the first argument, then it is read by the phrases adjacent to the right. Note that `S1` is received by the phrase '`v`'. The arguments of the phrases in '`ie`' are defined in the original user-written DCG rules, except for the first argument, which is for the channel variable and is added by the SAX translator automatically. The process `fin` receives the results of the final morpheme, which contains the overall parse results.

[1]In this example, we assumed that the system first looks up the dictionary and places the nonterminal phrases into the `process` as the current system does

Figure 3: Initial goals for "くるまでまつ"

Suppose the input sentence has a complicated morpheme graph, for instance, in the case of a Japanese sentence "くるまでまつ". The morpheme graph of the sentence is shown in Figure 3. The morphological analyser gives the two possible segmentations of morphemes, (くる + まで, and くるま + で). The remaining morpheme (まつ) has two possible parts of speech. The following initial goals are created by the system:

```
process([ie(verb([begin(t,[])],[verb(true),くる]),true)],
        S1),
process([ie(noun([begin(t,[])],[noun(true),くるま]),true)],
        S2),
process([ie(conjunction(S1,[conjunction(true),まで]),true),
         ie(particle(S2,[particle(true),で]),true)],
        S3),
process([ie(verb(S3,[verb(true),まつ]),true),
         ie(noun(S3,[noun(true),まつ]),true)],
        S4),
fin(S4))
```

Note that the shared varibles connect the morphemes exactly the same structure as the morpheme graph. For instance, the output variable of the meta-process taking care of "くる" is inputted to the conjunction "まで," and so forth. As will be seen form the above goals, morphemes finishing at the same position in the input, "まで" and "で" for example, and two morphemes of "まつ", are contained in a single meta-process, so that they can share the output variable. A graph structured output of the JUMAN system is directly transformed into an initial goal sequence of meta-processes.

4 VisIPS: visual interface for parsing systems

The VisIPS system is a visual interface to parsing systems that shows partial parse results of the parser in an intuitive way.

The SAX parsing system works as a black box for the users in that the user-defined DCG is transformed into a bottom-up concurrent Chart parser. When running the system, directly looking into the Prolog code is quite cumbersome since the original grammar rules are transformed in a nontrivial way. However, in the

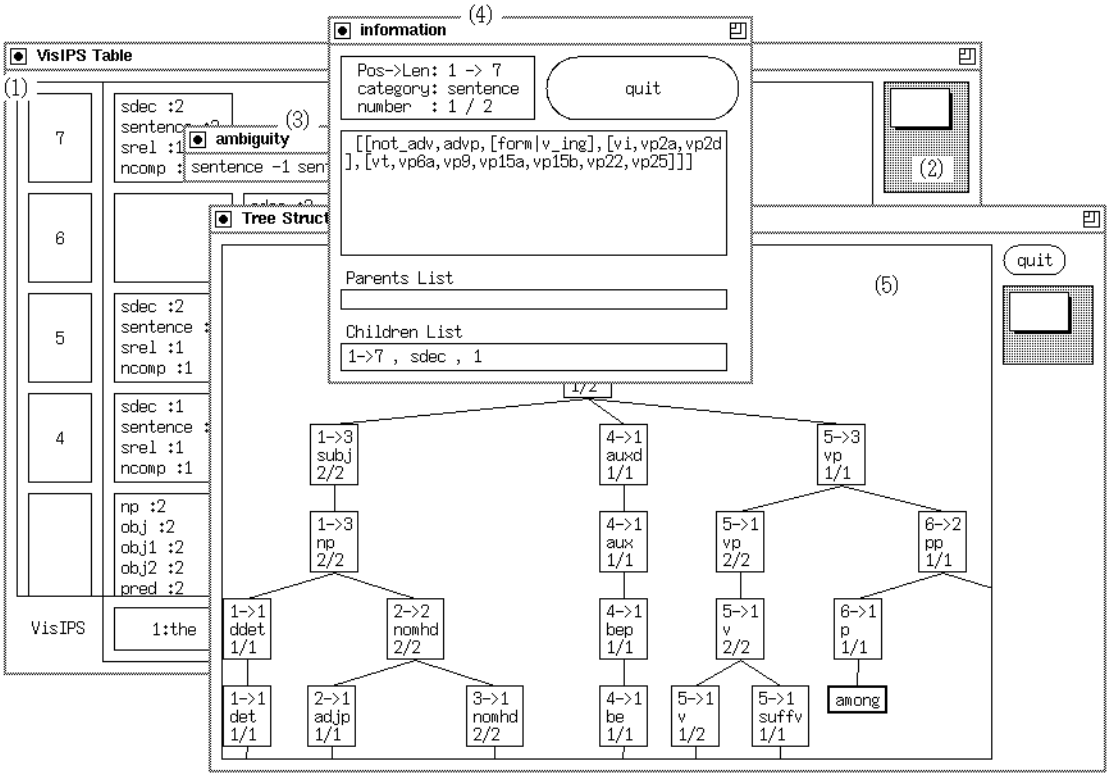

Figure 4: VisIPS in operation

developing phase of a grammar and a dictionary, it is indispensable to have some ways to figure out the system's behaviour. It should be noted that the users are usually not interested in the execution details of the system. The system, therefore, should inform the behaviour that is related only with the user defined grammar and dictionary.

The VisIPS system is originally developed to monitor the behaviour of the SAX processes. It is, however, applicable to any phrase structure based parsing systems. It shows occurrences of phrases in a triangular table. Two versions of the system have been developed: One is a batch system where the information of phrase structures are written out into a file in a predetermined format and VisIPS shows the results after the parsing process terminates. The other is an interactive mode where a newly constructed phrase structure is immediately presented. Both versions are implemented in C and the X11R5 system. Current system uses socket I/O facility of SICStus Prolog for data communication.

Figure 4 shows the system in operation. On the background is the VisIPS triangular table, that shows the partial parse results in a triagular table like the one used in the CKY parsing algorithm. When the user points the phrase name in the box by the mouse and click the button, a small window ((3) in the Figure) appears, in which as many number of the phrase are written if there is syntactic ambiguity. Pointing and clicking one of the phrase name in the box produces another window (4), in which detailed information is included. In (4), the first box shows the position and length of the phrase, the name of the phrase, and the identifying number if there is

ambiguity. The second box shows the inside information of the phrase. The third and fourth boxes show the list of parent phrases and the list of child phrases in the parse tree. By clicking the words "Children List" in the box, a parse tree appears in another window (5). The nodes in the parse tree interrelate with the phrase names in the triangular table, so that clicking a node in the tree makes the corresponding phrase in the table to blink.

The VisIPS system has been effectively used in developing and debugging grammar rules.

Common description of the tools

Name of the system: NAIST Natural Languate Tools
(JUMAN Morphological Analyzer, SAX parser, VisIPS, BUP)
Manuals: Japanese manuals are obtainable as NAIST technical reports. English manuals will be provided shortly.
Price: free
Limitation: no limitation
FTP site: cactus.aist-nara.ac.jp: ftp: /pub/(juman,sax,visips,bup)
Media: available in an 8mm tape
Format: UNIX file format (EUC for Japanese)
Style of demonstration: On-line
Contact person: Yuji Matsumoto (matsu@is.aist-nara.ac.jp)
Platform: Sun OS 4.1.3
Implementation Languages: SICStus Prolog, gcc, X11R5
Size: 15M bytes (approximate size of the whole tools)
Mailing list: nlt@cactus.aist-nara.ac.jp (in Japanese)

References

[Kay 80] Kay, M., "Algorithm Schemata and Data Structures in Syntactic Processing," XEROX PARC, CSL-80-12, Oct. 1980.

[Pereira 80] Pereira F.C.N. and Warren, D.H.D., "Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks," Artificial Intelligence, Vol.13, pp.231-278, 1980.

[Matsumoto 87] Matsumoto, Y., "A Parallel Parsing System for Natural Language Analysis," New Generation Computing, Vol.5, No.1, pp.63-78, 1987.

[Matsumoto 93] Matsumoto, Y., Den, Y. and Yanagi, K., "A Flexible Natural Language System with Concurrency and Meta-level Processing," Fourth International Workshop on Natural Language Understanding and Logic Programming, pp.146-157, 1993.

[Kurohashi 94] Kurohashi, S., Nakamura, T., Matsumoto, Y. and Nagao, M., "Improvements of Japanese Morphological Analyzer JUMAN," in this volume, 1994.

[Matsumoto 94] Matsumoto, Y., Kurohashi, S., Utsuro, T., Myoki, Y., and Nagao, M., "Japanese Morphological Analysis System JUMAN Manual, version 2.0," Nara Institute of Science and Technology, 1994.