PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

A Software Environment for Research into Discourse Understanding Systems

Ryôichi Sugimura Kôiti Hasida Kouji Akasaka Kôzi Hatano Yukihiro Kubo
Toshiyuki Okunishi
Institute for New Generation Computer Technology (ICOT)
Takashi Takizuka
KDD Kamifukuoka R&D Laboratories

ABSTRACT

This paper describes a software environment for research on discourse understanding systems.

Discourse understanding systems based on logic programming have been investigated since 1982 at ICOT. The focus of the research has been mainly on the basic mechanisms of *discourse understanding*, and the *parallel algorithms* for these mechanisms.

Experimental systems called **DUALS-I** and **DUALS-II** were developed to verify the research results.

These results have been put together to form a general purpose software environment for natural language processing named the Language Tool Box **LTB** .

This paper presents an overview of the LTB and discusses its future research and development direction.

1 INTRODUCTION

A natural language understanding system (NLS) must meet many kinds of demands.

As the interface system between computer systems and their users, an NLS should be able to understand not only what users say but also what they intend. An NLS is expected to make the interface more comfortable.

An NLS for researchers of linguistics or computer linguistics should be very flexible. It should enable researchers to input their ideas freely, such as grammars and lexical entries, so that it can be used to verify their research ideas. There should also be some debugging tools to check how a grammar or a lexicon works.

An NLS for software system development should have modularity, high performance, and compatibility with other application software such as an expert system. Maintenance tools should be prepared in the NLS.

The Language Tool Box (LTB) has been developed to meet these demands.

Since the foundation of ICOT in 1982, we have been researching many fundamental issues of natural language. Our basic starting point was logic. We have been focusing our research on complicated phenomena of natural language, using logic and logic programming.

Versions I and II of **DUALS** (Discourse Understanding Aimed at Logic Based Systems) have been already developed. Many ideas were poured into *DUALS*. Its main goal has been to verify fundamental ideas that arise from our basic research on natural language understanding. In the course of the development of *DUALS*, many kinds of software were developed and evaluated.

The LTB is the NLS developed from a collection of these software systems. The latest version of DUALS, **DUALS-III**, was developed on the LTB as shown in Figure 1.

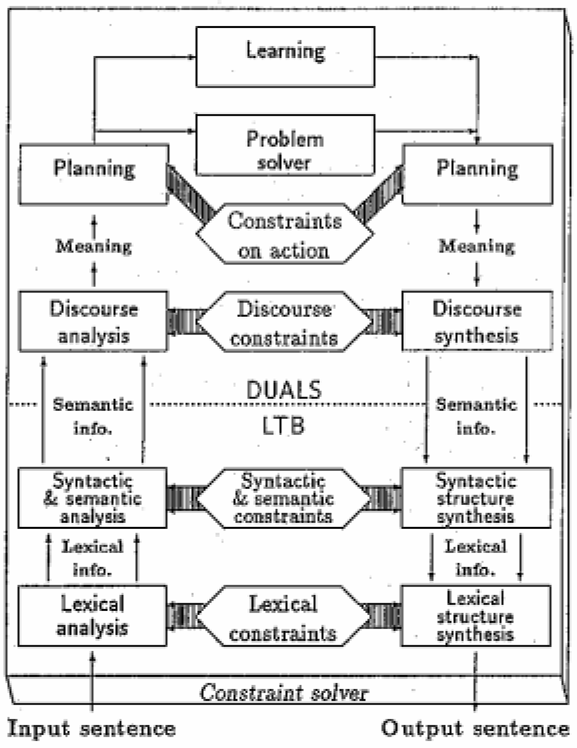

Figure 1: DUALS configuration

Features of the LTB include the following.

- It provides up-to-date logic-based tools for natural language understanding research.
- It accommodates many types of software environments such as debuggers and editors.

- All the tools are written in ESP [6] so that they have high modularity and compatibility with each other.

We believe that a variety of NLS could be developed based on the LTB.

2 STRUCTURE OF LTB

As shown in Figure 2, the LTB is constructed from the following component tools.

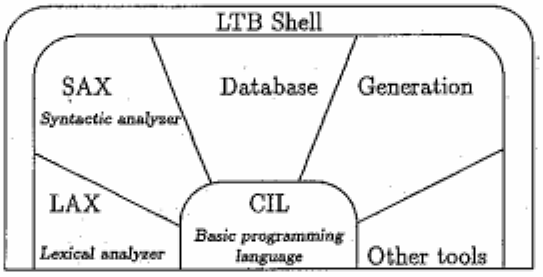

Figure 2: Configuration of the LTB

Shell: The LTB shell [40] facilitates the interaction with tools in the LTB. The shell itself can also be executed in a parallel logic programming language.

Database: The collection of Japanese lexicon, Japanese morphological grammar and syntactic grammars [41].

CIL: The basic programming language of the LTB. Every tool in the LTB has access to CIL and its software environments.

LAX: The morphological and semantic analyzer. The morphological grammars which the LAX looks up are written in a kind of extended regular expression. The LAX can execute analysis both in sequential logic programming languages such as Prolog or ESP, and in parallel logic programming languages such as GHC [44].

SAX: The syntactic and semantic analyzer. The grammar is written in DCG [32]. The SAX [21] can also execute analysis both in sequential and parallel logic programming languages.

Generator: The Japanese sentence generator, whose input is a frame structure written in CIL, outputs Japanese surface sentences.

3 LTB SHELL

The LTB shell has the following features.

- It manages all the communications in the LTB. It controls all the information flow among the user and application tools in the LTB.
- It accommodates the notify function which informs the shell of data communication among processes, so that the shell can measure the execution interval of a process in the LTB.
- It controls all the processes in the LTB.
- It provides a standard window. The window has a multi-lingual menu so that users can use LTB tools in either English or Japanese.
- It provides help instructions.

On parallel machines like PIM, the shell will enable each process to move simultaneously. Communications among the parallel processes in the LTB are performed as shown in figure 3.

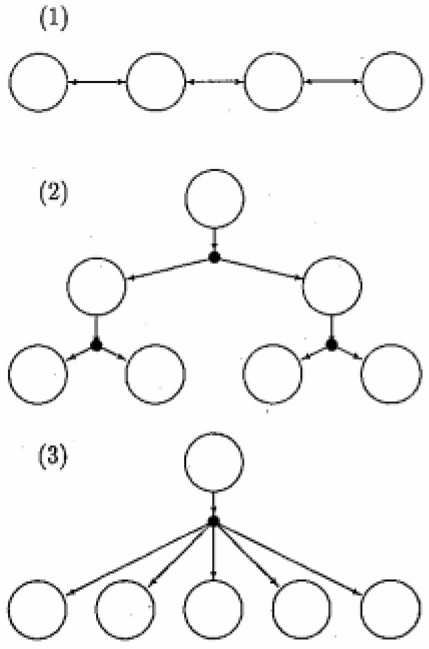

Figure 3: Redirection of stream data

In figure 3, graph (1) shows the data flow through a pipe-line in which there is no need for the data sender to specify the data receiver. Graph (2) enables communication between hierarchical processes in which the data sender should specify the direction, right or left. Graph (3) enables network communication between processes in which the data sender should specify the name of the receiver. We think that these three ways of communication are enough to install natural language processing systems on the LTB.

4 DATABASE

The LTB uses the following three databases for discourse understanding research. See [42] for details.

Japanese Word Dictionary: It provides a master dictionary of 4000 word entries for morphological analysis and synthesis, and a Japanese thesaurus of 800 concepts.

Japanese Grammars: Grammar for morphological analysis and syntactic analysis. Their features are as follows.

Morphological Grammar

1. It is written in a regular grammar.
2. It is based on Morioka grammar [24] along the line of structural linguistics.
3. Its semantics is formulated in situation theory [3] and situation semantics [2].
4. 2,000 lines/1,000 rules for derivations and inflections.
5. 30,000 lines/10,000 rules for word stems.

Syntactic Grammar

1. It is written in DCG.
2. It is based on dependency grammar [45].
3. Its semantics is formulated in situation theory and situation semantics.
4. 1,000 lines/2,000 rules.

Japanese KWIC: A Japanese key word in context (KWIC) has been under development, making it possible to analyse Japanese grammatical features.

5 BASIC PROGRAMMING LANGUAGE

CIL [30], the basic programming language of the LTB, is being developed to make it easy to describe natural language processing systems. An overview of CIL is given. See [25] [26] [27] [28] for further information.

CIL has two augmentations to Prolog (figure 4). One is a record-like structure called a **partially specified term** (PST). The other is the **freeze mechanism**, originating in PrologII [7]. A passive constraint solver is provided as a built-in predicate based on the freeze mechanism.

Currently, CIL is implemented on PSI machines and prepares the entire programming environment. Its debugging environment, which is a full-screen source image tracer based on the extended procedure box model, is customized for all LTB application tools.

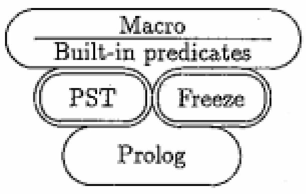

Figure 4: CIL language

5.1 PST

A PST is a record-like structure. [29] demonstrates that a record structure is useful for complicated data representation such as linguistic information, and that many useful data structures can be seen as derived from record structures.

A PST is a set of attribute-value pairs in the form:

$$\{a_1/b_1, ..., a_n/b_n\}$$

where

$$n \geq 0, \quad a_i \neq a_j \quad (i \neq j)$$

The order of pairs whose label is a_i and whose value is b_i can be ignored. For instance, `{a/1,b/{c/2,d/3}}` and `{b/{d/3,c/2},a/1}` are identical. They represent a tagged tree in Figure 5.

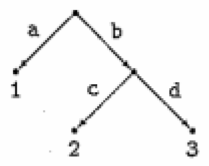

Figure 5: PST as a tagged tree

A value may also be a PST, which may be the parent PST, giving rise to circular data. A PST is designed to represent partial information. Unification of CIL has been extended to deal with this aspect. Pairs of different labels are merged with each other.

```
CIL> {a/{b/2}}={a/{b/3}}.
no
CIL> X={a/1},X={b/2}.
X = {a/1,b/2}
```

A PST can simplify the representation of and operation on data structure. CIL provides various operation predicates for a PST. For example, `role(L,P,V)` accesses value `V` with label `L` in PST `P`. In `t_subpat(P1,P2)`, `P1` is a transitive subpattern of `P2`.

```
CIL> role(a,{a/1,b/2},V).
V = 1
CIL> t_subpat({a/{b/Y}},{b/1,a/{c/2,b/3}}).
yes
```

5.2 Constraint Solver

`Freeze` can specify constraints that variables must satisfy. These constraints are solved on a lazy evaluation basis. That is, they are executed when the connected variables are instantiated. For instance,

```
CIL> freeze(X,X=2),print(ok),X=1.
ok
no
```

The freeze mechanism is one of the approaches to (passive) constraint solving. CIL has the `constr` predicate composed from `freeze`. Arithmetic and propositional constraints can be written in `constr`. For example,

```
CIL> constr((X=1;Y=:=X+2)),constr((X=3)).
X = 3,
Y = 5
```

While only constraints (not values) are accumulated, however, `constr` can only wait for a value and yields no solution:

```
CIL> constr((X=1;X=2)),constr((X=2;X=3)).
yes      % X is frozen.
```

For this reason, the constraint solver in CIL is passive. Introduction of an active constraint solver such as a Boolean constraint is being considered.

5.3 Macro and Built-in Predicates

For effective programming with a PST and freeze, CIL supplies a macro function and many built-in predicates, such as `role`, `t_subpat` and `constr`. Users can define their own macro with the following system macros.

$$\begin{aligned} \texttt{p(X:C)} &\Rightarrow \texttt{solve(C),p(X)} \\ \texttt{X!Y} &\Rightarrow \texttt{Z:role(Y,X,Z)} \\ \texttt{X\#Y} &\Rightarrow \texttt{X:X=Y} \\ \texttt{p(X?)} &\Rightarrow \texttt{freeze(X,p(X))} \\ \texttt{X@p} &\Rightarrow \texttt{X:p(X?)} \end{aligned}$$

CIL expands the form of the left-hand side of $\Rightarrow$ to that of the right-hand side in the program.

5.4 Programming Environment

Figure 6 shows the configuration of the CIL system.

CIL has extended unification, various extended notations and a freeze mechanism. Therefore, a standard debugger for Prolog cannot give very useful information. *Debug aid* extends the ordinary procedure box model and provides a full-screen source image tracer [1]. The box model is modified for constraint and head unification. The full-screen source image tracer enables the user to debug visually programs executed on a lazy evaluation basis. The *inspector* can inspect a nested PST. A *CIL editor* with a syntax checker is also available. Users can invoke them interactively through the *command interpreter* during program debugging. The *Compiler* translates the CIL program, after debugging, to an ESP program, which is executed efficiently.

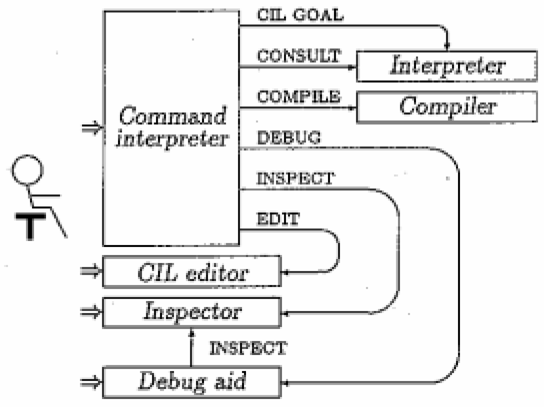

Figure 6: Configuration of the CIL system and user interface

6 APPLICATION TOOLS

This section describes the fundamental application tools in the LTB. The application tools consist of analysis tools and a synthesis tool. Currently, these tools process only Japanese, but can be extended to process other languages by changing the grammars and dictionaries. All application tools have programming environments with multi-window interface facilities.

7 MORPHOLOGICAL AND SEMANTIC ANALYSIS

The *LAX* is a software environment which enables us to develop morphological analysis programs effectively. These programs have an analysis engine and a dictionary which is compiled from a morphological dictionary (the *LAX dictionary*).

Morphological analysis consists of two phases. The first phase is to recognize morphemes from an input sentence that consists of kanji (Chinese characters) and the two kinds of Japanese phonograms, and put them into a sequence of words. The second phase is to construct the meaning of each word from semantic information written in the *LAX dictionary*, and output sequences of nonterminal symbols with semantic structures.This output is used as an input for syntactic analyzer *SAX*.

The *LAX system* supports a sequential LAX analysis algorithm and has two types of development tools called the *LAX inspector* and *LAX dictionary*.

7.1 Morphological Analysis Algorithm

This section briefly explains the features of the LAX analysis algorithm. For details, see [38]. In this method, a morphological dictionary is transformed into sequential logic programming languages such as ESP or Prolog, or into parallel logic programming languages such as GHC. These programs can analyze Japanese sentences without backtracking and output all ambiguities at one time. This algorithm is based on the layered stream technique and can be regarded as a parallel to the method of Earley's algorithm or the bottom-up Chart parsing algorithm applied to regular grammars.

7.2 Morphological Dictionary

Figure 7 shows the format of the *LAX dictionary*. Definitions of morphemes that belong to the same category are written between `begin(Category Name)` and `end(Category Name)`. Three kinds of information are declared for each morpheme. The first is a *left-hand feature* which is the identifier of the morpheme. The second is a *right-hand feature* which indicates what kinds of morphemes can follow that morpheme. The third is a *semantic rule* which is evaluated in the second phase of analysis. These semantic rules are written in CIL, and semantic structures are represented in terms of PSTs. This dictionary describes a regular grammar so that the analyzer can be regarded as a non-deterministic finite state automaton.

```
begin(Category Name)
morpheme ::  Lefthand Feature
         &&  Righthand Feature
         $$  Semantic Rule
              ⋮
end(Category Name)
```

Figure 7: LAX dictionary format

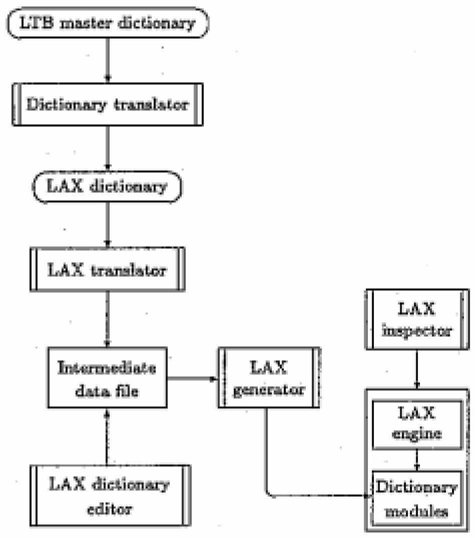

Figure 8: Configuration of the LAX system

7.3 Configuration of LAX Software Environment

Figure 8 shows the configuration of the *LAX system*. The *dictionary translator* transforms the *LTB master dictionary* into the *LAX dictionary*. Using the *LAX translator*, the *LAX dictionary* is transformed into an intermediate data file. The analysis program is generated by the *LAX generator* from this intermediate data file. From the user's point of view, this intermediate data file is identical to the *LAX dictionary*.

To debug the *LAX dictionary*, the *LAX inspector* provides many pieces of useful information. It has two modes: the analysis mode and the inspection mode. In the analysis mode, you can check the results of analysis, looking at the time and the number of ambiguities for each input sentence. If the analysis fails, you can also see which morpheme cannot be connected with its adjoining morpheme. In the inspection mode, you can examine the contents of the dictionary of the morphemes that belong to each adjoining point of input sentences.

We can easily modify the definitions of morphemes and add new entries by using the *LAX dictionary editor*. To modify a definition of a morpheme, we only have to rewrite the lef-thand or right-hand features or semantic rule in the edit window after locating the definition. To add new definitions, we may pick up another definition, copy it, and rewrite it.

7.4 Morphological Grammar

Using the *LAX system*, we have been developing a Japanese morphological grammar [35]. We have defined about 2000 morphemes and succeeded in analyzing about 200 sentences. A Japanese morphological grammar can be developed naturally in the *LAX system*.

8 SYNTACTIC AND SEMANTIC ANALYSIS

This section describes the syntactic and semantic analysis part of the LTB, called the SAX system. First, in section 8.1, the grammar syntax and SAX semantics are represented. Then, the SAX is outlined, mainly referring to its relationship with other LTB components. Section 8.3 briefly explains the SAX parsing method. A restricted Definite Clause Grammar is assumed as the grammar description. See [21] for a detailed description. Finally, section 8.4 introduces the debugging tools [46] provided by our system.

8.1 SAX Grammar Rules

Figure 9 shows the SAX grammar rules. Basically, they are rules extended from DCG [32].

$$\begin{aligned} head \;\rightarrow\;\; & body_1, \{extra_1\} && (1)\\ & :: \{pref\_rule_1\}, && (2)\\ & \ldots && (3)\\ & body_n, \{extra_n\} && (4)\\ & :: \{pref\_rule_n\}, && (5)\\ & \&\{delayed\_extra\}, && (6)\\ & \&\&\{pref\_rule\} && (7)\end{aligned}$$

Figure 9: SAX grammar rules

In figure 9, *head* at line (1) and $body_i$ in line (1) or (4) represent grammatical categories. They are presented in the form of Prolog terms which can have optional arguments. $extra_i$ in line (1) or (4) is the extra condition in which an optional Prolog program can be written. *delayed_extra* in line (6) is also an extra condition called the *delayed extra condition*. The optional Prolog program in the *delayed extra condition* is evaluated after completion of the parsing.

SAX executes parsing bottom-up and breadth-first so that there is some limitation on the rules. Firstly, on the left-hand side of $body_1$ in line (1), we cannot write any extra conditions. Secondly, the Prolog variables in $extra_i$ should be instantiated when $extra_i$ is evaluated.

Rules in lines (2), (5), and (7) never fail. They are used to calculate *lexical preference*. Generally, a sentence has more than one interpretation. Disambiguation of the sentence interpretations has been one of the hardest problems in natural language processing. Therefore, many kinds of approaches have been studied, such as disambiguation with local lexical constraints [43], or disambiguation with the discourse [37]. The rules in lines (2), (5), and (7) enable us to calculate one of the local lexical constraints.

The SAX system supplies several items of information related to the calculation of *lexical preference* as follows.

1. In $\{pref\_rule_i\}$ after ::

   pref_cat In $pref\_rule_i$, preference for $body_i$ which is calculated in the rule whose head is $body_i$.

   pref In $pref\_rule_i$, preference for $body_i$ in the rule.

2. In $\{pref\_rule\}$ after &&

   pref_CAT Preference for the *head*.

   prefs(i) *pref* in $pref\_rule_i$.

3. In $\{pref\_rule\}$ after ::, or &&

   super_cat_set The set of grammatical categories which includes the *head* as its partial tree.

   next_pos The grammatical category ahead of the right most leaf node of the tree whose root node is *head*.

Another feature of the SAX is that Gapping Grammar (GG) [8][9] can be written. GG is the grammar formalism in which a discontinuous sequence of grammar categories can be written. It is performed with the expansion of the SAX parsing algorithm[20][22].

Generally, the grammar rules in GG are as follows.

$$\begin{aligned} \beta \;\rightarrow\;\; & \alpha_1, skip(G_1),\\ & \ldots\\ & \alpha_n, skip(G_n), \alpha.\end{aligned}$$

In the rules, α and α_i are a sequence of grammatical categories whose length is greater than 0. $skip(G_i)$ is a special symbol in GG. β is a sequence of symbols including some grammatical categories and $skip(G_i)$. In β, every $skip(G_i)$ in the rule body should be included. In the course of the parsing, $skip(G_i)$ can be matched to any grammatical category, and a $skip(G_i)$ in β should represent the same grammatical category of a $skip(G_i)$ in the rule body.

In SAX, rules for GG are analyzed bottom-up so that there are limitations and expansion for the GG rules in the SAX as follows.

- α_1 should not be empty.
- GG rules should not be applied recursively.
- $skip_i$ can be matched with grammatical categories.
- $skip_i$ is represented as follows.
  $skip(i)$ or $skip(i, S)$
  i is a natural number, and S is the set of grammatical categories which *skip* matches or does not match. Specifically, S is either of the following: $+\{a_1, \ldots, a_n\}$ representing *skip* can be matched only with a_1, or, ..., or a_n, or $-\{body_1, \ldots, body_n\}$ representing *skip* cannot be matched with $body_1$, and, ..., and $body_n$.

8.2 SAX System

Figure 10 shows the configuration of the SAX. Given grammar rules are translated into a parsing program written in ESP. In these grammar rules, CIL notations are allowed in DCG's extra condition, so that many CIL functions are available. We have developed Japanese grammar rules, using CIL's built-in predicate for describing syntactic and semantic constraints, and for constructing semantic structure.

The translated parsing program receives the results of morphological and semantic analysis and parses at high speed. Semantic representation is constructed simultaneously from the semantic information supplied by lexical semantic processing.

Some other tools, such as the grammar debugger, and graphic display utility, are provided for effective development of the grammar.

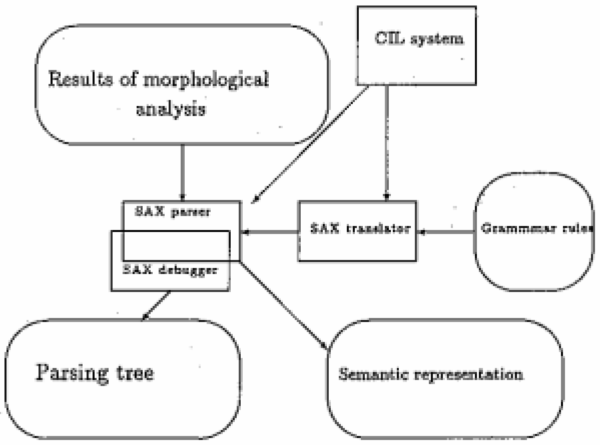

Figure 10: Configuration of the SAX system

8.3 Parsing Method

The SAX parsing algorithm was devised for parallel parsing [21]. However, it turned out to work efficiently in sequential implementation as well [31].

The system employs bottom-up parsing with top-down prediction. The major advantages of our system are that the system works bottom-up: therefore, the left-recursive rules do not cause problems, and the parsing process does not involve backtracking, which means that there is no redundant construction of syntactic structures.

8.4 Debugging Environment

Grammar debugging tools are indispensable for the development of practical grammar rules. The SAX provides two types of debugging tools.

The first is an interactive tracer, which shows the parsing process, displaying the rules currently being applied. As with the Prolog tracer, the user can control the trace by issuing commands such as "skip" and "leap". However, this kind of debugging tool is not very suitable for the SAX since it is hard to follow the process of parsing breadth-first, and in order to use the tracer, the user needs to know the SAX parsing mechanism.

The second is an algorithmic debugger on DCG. Using this tool, the user interacts with the system and tries to construct the expected parsing tree from input words according to the grammar rules. The user specifies the start point and end point of the sequence of words, and the system parses the sequence. If the parsing succeeds, the system displays the obtained root categories of the resultant trees.

If the user does not find the expected category in them, an error is in the grammar rules that are to be applied when parsing that sequence of words. In this way, the user can narrow the candidates for the wrong rule.

9 JAPANESE GENERATION

The Japanese generation module provides a function for generating sentences, and tools for customizing the module itself. This section describes the basic concept of sentence generation and tools in the module.

9.1 Aim and Features of the Japanese Generation Module

The Japanese synthesis module generates sentences from "intermediate representation". The intermediate representation is basically a syntactic structure and contains all the information needed to generate sentences. The module does not reshape its output using contextual information.

One of the important goals of designing the module is to enable it to be used as an NLS interface in various systems such as expert systems, so the rule definition such as grammar in it must be easy to customize. We made customization easy by introducing the following features.

– Most of the generating process is done by macro expansion.

- Tools for tracing the process and making data are provided.

9.2 Sentence Generation as Macro Expansion

Although the module does not process contextual information, sometimes users may want to input abstract information such as deep structure (or a thematic role). The module allows users to include such information in the intermediate representation, and regards it as a macro expression in the syntactic structure.

Figure 11 shows the sentence generation flow. First, the macro expander expands macro expression in the intermediate representation and makes a pure syntactic structure. In this phase, the macro expander uses several macro definitions and a lexicon. Each macro definition is a rule for translating a macro expression to part of the syntactic structure. The lexicon gives lexical information such as the correspondence between thematic roles and case markers. The next phase is to translate the syntactic structure to the output sentence (character string). In this phase, the string synthesizer picks up lexical information in the syntactic structure, applies a morphological rule to it, and inflects each word.

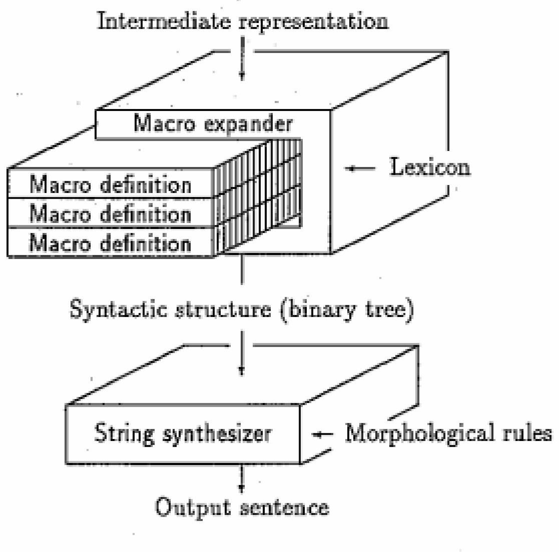

Figure 11: Sentence generation

The advantages of this method are:

- Users can design the intermediate representation that they want, by changing the macro definition.
- By defining macros on another definition, intermediate representation can be constructed hierarchically.
- Users can specify information roughly or precisely in intermediate representation. For example, if they want to specify a strict word order, they may write it in pure syntactic structure, but if they do not need to specify it as exactly, they may write it in terms of a more abstract expression.

It has been said that the syntactic structure of a Japanese sentence is clearly represented by a binary tree [12]. In the Japanese generation module, the syntactic structure is represented in a binary tree. Figure 12 shows a simplified example of the Japanese phrase "Kodomo wo gakkô e ika seru (to make a child go to school)" written in the form of a binary tree.

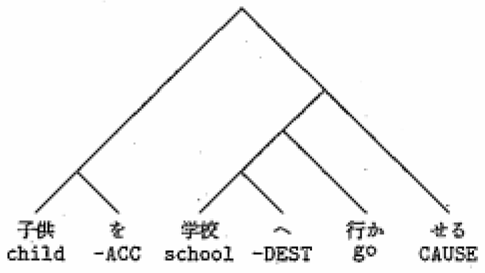

Figure 12: Example of syntactic structure

Although users can define the syntax of intermediate representation, the *standard macros* are already defined in the module [11]. The definition includes several rules, including the following.

- A meaning role may be expanded into a binary tree with a case marker.
- Voice information may change case markers.
- Auxiliaries are added to the tree when aspect, tense, or modality information is given.

Figure 13 shows the intermediate representation of the sentence shown in Figure 12 using standard macros.

9.3 Tools

The configuration of the whole generation module, including the tools, is shown in Figure 14. Thus, the generation module includes the following two parts besides the generation engine.

Debugger: Traces the execution of the generation engine (macro expander and string synthesizer), switches the rules (lexicon, macro definition, and morphological rule), and reads the input data (intermediate representation).

```
{関係 / {語彙 / 行く},
 rel   lex   go
 ロール / {行為者 / {補語 / {語彙 / 子供}},
 role    agt     comp  lex   child
         終点 / {補語 / {語彙 / 学校}}},
         dest  comp  lex   school
 ヴォイス / 使役}
 voice    causative
```

Figure 13: Example of intermediate representation

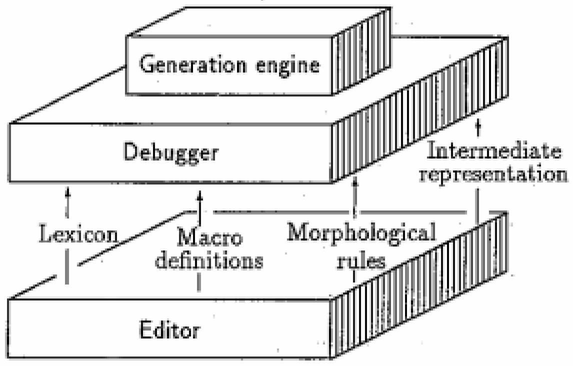

Figure 14: Configuration of the Japanese generation module

Editor: Edits the rules and input data.

These tools have the following features.

- The debugger is built on the CIL debugger. Its window, mouse and key manipulation are the same as those of CIL.
- The editor is built on pmacs [17]. Its key assignments are the same as those of pmacs. It checks and guides the syntax of the data at the users request.

10 CONCLUSION

Current research subjects in the *LTB* are as follows.

- Filling up the more comfortable programming environments such as debugger, inspector, and pretty printer. To improve the programming environments, they should be evaluated in terms of performance and competence by users and developers.
- Building up more convenient linguistic databases and interface facilities for them.
- Installation of the LTB on the parallel logic programming language GHC which runs on the multi-PSI [23] or PIM. Some LTB tools have been installed on the multi-PSI for experiments.
- Supplements of built-in predicates such as constraint unification [13][14][15][16], Boolean Gröbner base [34], type inference [4] [33], and finite domains [10], which execute fundamental inferences for discourse processing.

Current research and development in the LTB are supported by a working group (WG-Sig2) at ICOT which is chaired by Professor Hozumi Tanaka of the Tokyo Institute of Technology.

There is much room for improvements in the LTB. Improvements and evaluations of the LTB will be carried out mainly by ICOT and WG-Sig2 researchers.

To supplement built-in predicates or functions, we need to study implicit rules of discourse [5] [18] [19] [36] [39] in more detail. In discourse analysis, we have not yet found or created explicit rules, which are exemplified by sufficient numbers of experimental results. We should be able to use some general tools such as constraint unification and the Boolean Gröbner base solver, and experimental tools such as the KWIC. These tools will be installed in the LTB.

After finding some rules, we will research basic core constraints and solution mechanisms for them. These mechanisms will then be new tools for the LTB.

ACKNOWLEDGEMENTS

We would like to thank Professor Hozumi Tanaka at Tokyo Institute of Technology and the members of working group Sig-2 for their intensive discussions and suggestions. We would also like to thank Dr. Fuchi, the director of ICOT Research Laboratory, and Dr. Uchida, the chief of the Second Research Laboratory at ICOT, for their encouragement and support for this work.

References

[1] T. Amanuma, T. Suzuki, T. Okunishi, and K. Mukai. CIL Programming kankyô (CIL Programming Environment (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 211–214, 1988.

[2] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial Intelligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[3] J. Barwise and J. Perry. *Situations and Attitudes.* MIT Press, Cambridge, 1983.

[4] L. Cardelli. Typechecking Dependent Types and Subtypes. Technical report, DEC Systems Research Center, 1987.

[5] D. Carter. *Interpreting anaphors in natural language texts.* Ellis Horwood Series in Artificial Intelligence. Ellis Horwood, 1987.

[6] T. Chikayama. ESP Reference Manual. Technical Report 044, ICOT, 1984.

[7] A. Colmerauer. Prolog-II: Reference Manual and Theoretical Model. Internal report, Group Intelligence Artificielle, Université d'Aix-Marseille II, 1982.

[8] V. Dahl. More on Gapping Grammar. In *Proceedings of the International Conference on FGCS '84*, pages 669–677, Tokyo, 1984.

[9] V. Dahl and H. Abramson. On Gapping Grammar. In *Proceedings of 2nd ICLP*, pages 77–88, Sweden, 1984.

[10] M. Dincbas, H. Simonis, and P.V. Hentenryck. Extending equation solving and constraint handling in logic programming. Internal Report IR-LP-2203, ECRC, 1987.

[11] T. Ikeda et al. Sentence generation in LTB (*in Japanese*). In *5th Conference Proceedings of Japan Software Science and Technology*, 1988.

[12] T. Gunji. *Japanese Phrase Structure Grammar.* Dordrecht D. Reidel, 1987.

[13] K. Hasida. Conditioned Unification for Natural Language Processing. In *Proceedings of the 11th COLING*, pages 85–87, 1986.

[14] K. Hasida. Dependency Propagation: A Unified Theory of Sentence Comprehension and Generation. In *Proceedings of the 10th IJCAI*, pages 664–670, 1987.

[15] K. Hasida. Izondenpa (Dependency Propagation (*in Japanese*)). In *Proceedings of the 29th Programming Symposium*, pages 147–158, 1988.

[16] K. Hasida and H. Sirai. Zyôkentsuki tan-itsuka (Conditioned Unification (*in Japanese*)). *Computer Software*, 3:28–38, 1986.

[17] ICOT. PSI/SIMPOS Editor Manual. 1988.

[18] M. Kameyama. *Zero Anaphora : The case of Japanese.* PhD thesis, Stanford University, 1984.

[19] K. Kimura, R. Sugimura, T. Takizuka, and K. Mukai. Danwa Rikai Jikken System DUALS Dai 2-han no Sekkei to Jissô (Design and Implementation of Discourse Understanding System DUALS-v2 (*in Japanese*)). In *3rd Conference Proceedings of Japan Society for Software Science and Technology*, pages 33–36, Tokyo, 1986.

[20] Y. Matsumoto. Ronri Bunpô no Heiretsu Kôbun Kaiseki (Parallel Analysis of Logic Grammars (*in Japanese*)). *IPSJ*, 29(4):335–341, 1988.

[21] Y. Matsumoto and R. Sugimura. A Parsing System Based on Logic Programming. In *Proceedings of IJCAI 87*, 1987.

[22] Y. Matsumoto and R. Sugimura. Kôbun Kaiseki System SAX no tameno Bunpô Kijutsu Gengo (Grammar Description Language for the SAX Parsing System (*in Japanese*)). In *5th Conference Proceedings of Japan Society for Software Science and Technology*, pages 77–80, Tokyo, 1988.

[23] T. Miyazaki and K. Taki. Multi-PSI ni okeru Flat GHC no Jitsugen Hôshiki (Installation of Flat GHC on Multi-PSI (*in Japanese*)). Technical Report 190, ICOT, 1986.

[24] K. Morioka. Goi no Keisei (Formation of a Vocabulary (*in Japanese*)), volume 1 of *Gendaigo Kenkyuu.* Meiji Shoin, 1987.

[25] K. Mukai. Horn Clause Logic with Parameterized Types for Situation Semantics Programming. Technical Report 101, ICOT, 1985.

[26] K. Mukai. Unification over Complex Indterminates in Prolog. Technical Report 113, ICOT, 1985.

[27] K. Mukai. Anadic Tuples in Prolog. Technical Report 239, ICOT, 1987.

[28] K. Mukai. A System of Logic Programming for Linguistic Analysis based on Situation Semantics. In *Proceedings of the workshop on semantic issues in human and computer languages.* CSLI, 1987.

[29] K. Mukai. Partially Specified Term in Logic Programming for Linguistic Analysis. In *Proceedings of the International Conference on FGCS '88*, 1988.

[30] K. Mukai and H. Yasukawa. *Complex Indeterminates in Prolog and its Application to Discourse Models*, volume 3, pages 441–466. OHMUSHA, ltd. and Spring Verlag, 1985.

[31] T. Okunishi, R. Sugimura, Y. Matsumoto, N. Tamura, T. Kamiwaki, and H. Tanaka. *Comparison of Logic Programming Based Natural Language Parsing Systems*, volume 2, pages 1–14. North-Holland, v. dahl edition, 1988.

[32] F. Pereira and D.H.D. Warren. Definite clause grammars for language analysis – a survey of

the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[33] J. Reynolds. Three approaches to type structures. *Lecture Note in Computer Science*, volume 185, Springer Verlag, 1985.

[34] K. Sakai and Y. Sato. Boolean Gröbner bases. Technical Memo 488, ICOT, 1988.

[35] H. Sano, K. Akasaka, Y. Kubo, and R. Sugimura. Go-kôsei ni motoduku Keitaiso Kaiseki (Morphological Analysis with Derivation and Inflection (*in Japanese*)). In *Proceedings of the 36th Conference of Information Processing Society of Japan*, 1988.

[36] R. Sugimura. Japanese Honorifics and Stuation Semantics. In *Proceedings of the 11th COLING*, pages 507–510, 1986.

[37] R. Sugimura. Ronri-gata Bunpô ni okeru Seiyaku Kaiseki (Constraint Analysis on Logic Grammars (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese society for Artificial Intelligence*, pages 427–430. Japanese Society for Artificial Intelligence, 1988. a prize-winning paper.

[38] R. Sugimura, K. Akasaka, Y. Kubo, H. Sano, and Y. Matsumoto. Ronri-gata Keitaiso Kaiseki LAX (Logic Based Lexical Analyzer LAX (*in Japanese*)). In *Proceedings of the Logic Programming Conference '88*, pages 213–222. ICOT, 1988. English version will appear in The Lecture Note on Computer Science.

[39] R. Sugimura, H. Miyoshi, and K K. Mukai. *Constraint Analysis on Japanese Modifying Relations*, volume II, pages 93–106. North-Holland, 1988.

[40] T. Takizuka and R. Sugimura. Ltb Shell no Kôsei (Configurationof LTB Shell (*in Japanese*)). In *Proceedings of the 37th Conference of Information Processing Society of Japan*, pages 1074–1075, 1988.

[41] T. Takizuka, Y. Tanaka, and R. Sugimura. LTB Master Jisho no Kôsei (Configuration of LTB Master Dictionary (*in Japanese*)). In *Proceedings of Logic and Natural Language Conference*. Japan Society for Software Science and Technology, 1987.

[42] Y. Tanaka and T. Yoshioka. Overview of the Dictionary and Lexical Knowledge Base Research. In *Proceedings of the International Conference on FGCS '88*. ICOT, 1988.

[43] J. Tsujii. Bun-kaiseki system KGW+P no seigyo hôsiki (Controlling Methodology of Sentence Analysis System KGW+P (*in Japanese*)). In *4th Conference Proceedings of Japan Society for Software Science and Technology*, pages B–4–3, 1987.

[44] K. Ueda. Guarded Horn Clauses. Technical Report 103, ICOT, 1985.

[45] M. Watanabe. Kokugo kôbun-ron (Syntax of Japanese (*in Japanese*)). Hakama Shobô, 1971.

[46] S. Yamasaki, R. Sugimura, K. Akasaka, and Y. Matsumoto. Kôbun Kaiseki System SAX no Debug Kankyou (Debugging Environment of SAX System (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 411–414, 1988.