Invited Paper

On Japanese-based Programming

TETSUO TAMAI*

Since many of the existing programming languages have been designed in English speaking countries, their phrases and structures are influenced by the English convention. However, as the technology of Japanese processing has made rapid progress, time is ripe for reconsidering new Japanese-based computer language design. This paper surveys the current status of "Japanese programming" and also reports on an experiment of investigating its usefulness.

1. Introduction

Many of the existing programming languages have been designed in English-speaking countries and their phrases and structures are more or less influenced by the english grammar and vocabulary. There used to be some efforts to "Japanize" programming languages but their scope of activities was limited, typically taking common languages like Fortran and Cobol and translating their keywords into corresponding Japanese words. Naturally, those languages failed in attracting attention of many Japanese programmers.

As the technology of Japanese input and output devices has been making rapid progress, time is ripe for reconsidering construction of new Japanese-based programming languages and programming style.

The principal motive for developing Japanese-based programming is to let broad classes of the Japanese population have easy access to programming. Personal computers have enlarged the opportunities of ordinary people to come in contact with computers, thus it is desirable to provide them with means of programming in a familiar style. At the same time, the shortage of professional software engineers is currently a serious problem. The Japanese programming may help alleviate this shortage problem in two ways. One is to open a way for end-users to produce software by themselves, thus reducing software demands. The second is to enhance software productivity and maintainability in Japan. This is expected, because Japanese is a mother tongue for most of software engineers in Japan and specifications for such software are commonly written in Japanese, thus the gap between specifications and programs is expected to be narrowed if Japanese-like notations and phrases are admitted in the programming language.

Some may argue that programming is a task of a universal nature and a programming language should be considered as a means to express formal and universal concepts just like mathematical notations. In that sense, a programming language based on a specific culture is not useful, even does harm. To this, we may contend that this approach can be expected to bring new aspects to programming by incorporating some features of the Japanese language or even Japanese way of thinking. Also, it can be one step towards the multilingualization of computer usage thus help spreading computer technology to non-English cultures.

From the above point of view, we conducted a survey on programming languages and specification description languages based on Japanese and also made an experiment using some of those languages to examine the effectiveness and potentialities of "Japanese programming". In the followings, we report the results of this survey and the experiment and make discussions.

2. Overview of Japanese Programming

In this paper, we define Japanese programming as "software development activities in the environment where specifications/programming languages, their compilers and other related tools are used which incorporate notations, constructs and/or grammars based on the Japanese language." There is a case in which Japanese text processing is included in the category of Japanese programming, i.e. programming using languages with Japanese character-string handling features such as Japanese SNOBOL or more sophisticated tools, but here we exclude them from our consideration.

Some of Japanese programming languages are designed by transforming existing (English-based) languages

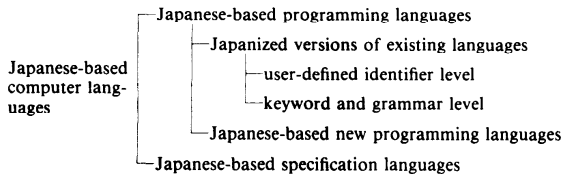

Fig. 1 Classification of Japanese-based computer languages.

*Graduate School of Systems Management, University of Tsukuba, Tokyo, Japan.

like Cobol, Fortran and C. We can divide this type of languages into two classes by considering to what extent Japanese features are introduced. Thus we can classify the Japanese-based computer languages as shown in Figure 1.

When we say Japanese programming, the word Japanese can imply two different aspects: 1) not English but Japanese based programming; 2) not artificial but natural language based programming. Most of Japanese programming languages are designed from the former standpoint. The latter is not unique to Japanese and actually there have been works upon this idea taking English as a natural language (e.g. [5], [13]). Many Japanese-like specification languages are taking the latter standpoint as described in the following section.

3. Specification Description Languages

For writing specifications, be it at requirement level or at systems design level, it is desirable to have a description language that allows natural and readable expressions. The natural language is the most convenient for that purpose, except that it is often ambiguous and is not easy to treat by computers.

Japanese-based specification languages aim at some middle point between natural languages and artificial formal languages. Their objectives are various but typical ones include: 1) standardization of specification writing; 2) automatic generation of programs from specifications; 3) unified description of specifications and programs; and 4) software development by end-users.

According to their objectives, the features of these languages vary. They can be roughly divided into two groups; one emphasizes the description capabilities and the other emphasizes automatic program synthesis. The latter can also be divided into two; one synthesizes programs every time from scratch and the other reuses stored program components. Figure 2 shows the above classification and system examples. Table 1 lists typical systems. At the moment, all of these systems are research prototypes.

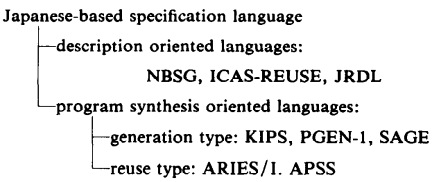

Fig. 2 Classification of Japanese-based specification languages.

4. Programming Languages

Early attemps to change keywords of Cobol and Fortran into Japanese words go back to 1960's. Those languages were not widely accepted, mainly because computers did not have the capability of handling Japanese characters (kana and kanji) in those days, thus Japanese keywords had to be spelled by alphabets.

Since Japanese character processing became common around 1978–1980, many programming languages have come to allow kana and kanji use in character-strings data, comments, and user-defined identifiers. No particular necessity seems to be felt so far by most programmers to use Japanese words in place of keywords of conventional languages.

Table 1 Japanese-based specification languages/systems.

Language/system	Developer	Characteristics	Major target domain	Object language	Reference
KIPS	Sugiyama et al (Fujitsu)	Automatic program derivation	File processing	Hyper-Cobol	[25] [26]
NBSG	Shiino et al (Oki)	Standardizing specification writing	System software	C (indirect)	[14] [21]
PGEN-1	Iwamoto et al (NEC)	Automatic program derivation	Banking system	Cobol/S	[6]
ARIES/I	Harada (ORIEPI)	Synthesis from reusable components	File processing	Hyper-Cobol	[4]
ICAS-REUSE	Chigira et al (Hitachi)	Reuse of specifications	Business applications	—	[1]
SAGE	Gyotoku et al (Mitsubishi)	Automatic program derivation	File processing	A high-level language for business	[3] [27]
APSS	Uehara et al (Osaka Univ.)	Synthesis from reusable components	Data-base retreival	A data base language	[29] [30]
JRDL	Ohnishi et al (Kyoto Univ.)	Support validating specifications	General but mainly business applications	—	[20]

Table 2 Japanese-based programming languages/systems.

Language/system	Developer	Characteristics	Major target domain	Target users	Reference
Nihongo AFL	Sugano et al (Matsushita)	All data from numeric to programs are treated as char-strings. A pioneer language on PC	Text processing	novices	[11] [22] [23]
Syo-syusin	Mizutani (TWCU)	Snobol like language for teaching literature	Text processing	students	[15] [16] [17]
Mind	Katagiri (TWCU)	Stack based language influenced by FORTH	Business appl. & system software	from novice to SE	[7] [8] [9]
YPS	Fujitsu	Program generation from chart language YAC II	Various (generate Cobol, Fortran & C programs)	SE	[2] [19]
Nihongo program	Kimura (Nagano Kenshinren)	Generate Cobol programs from Japanese-like expressions.	Banking business applications	average programmers	[10]
NBSG/PD	Shiino et al (Oki)	Generate C programs from Japanese-like expressions	Control software	SE	[24]
SPPM	Morimoto et al (Toshiba)	Object-oriented language	Business applications	End-users	[18]

On the other hand, some new attemps have been made to design new programming languages introducing features of Japanese. Table 2 shows typical examples. We take some languages in the table to give detailed explanations below. Among these, Nihongo AFL, Syo-syusin, and Mind are the languages that we chose for a programming experiment to be described later.

4.1 Nihongo AFL

The design of Nihongo AFL was started as early as 1977 by K. Ueda, J. Sugano and others as a Japanese version of AFL, a language specialized in processing character data, which had been developed earlier by the same group. Nihongo AFL is a simple language that runs on personal computers under CP/M or MS/DOS. In 1983, its processor was released as a commercial software package by the name of Wakan and sold a few hundred copies, but now no sales efforts are being taken.

One of the unique features of this language is that it uniformly treats characters, numbers, and even programs as the same character-string data. Another feature is its programming style that lets programmers write software by piling up definitions of words.

Nihongo AFL is too simple for large scale software; its control structures such as loops and conditional statements have too little variations; it has no local variables or other means for encapsulation; it has no logical expressions and very few numerical expressions. Even with these drawbacks, Nihongo AFL should be given high evaluation as a pioneer in the area of original Japanese-based computer language exploration and for its unique language design.

4.2 Syo-syusin

Syo-syusin was created by S. Mizutani for the educational purpose to teach literature classes at Tokyo Women's College. its major objective is to manipulate Japanese character-strings (in that sense, similar to Nihongo AFL) and is carefully designed to allow expressions that can be read as natural Japanese sentences.

The first version of its language processor is called REDLIPS (REDuction-to-LIst Processing System, the language name Syo-syusin is a literal translation of "(small) red lips") and implemented on TOSBAC ACOS 400, a small business computer which is rather old model with no kanji handling function. Thus, the current version of Syo-syusin uses katakana (Japanese alphabet) for pre-defined keywords and Roman letters for user-defined identifier names. This awkward situation is going to be altered in a new version now being implemented on a workstation HITAC 2020.

Syo-syusin has only a small number of simple data types: strings and numbers (real, integer and binary). On the other hand, its control structures are as rich as found in conventional programming languages: functions and subroutines, sequences, conditionals, and loops. String manipulating operations are abundant, especially rich in pattern matching functions, because text-processing is the main target of this language.

Probably, the nicest feature of Syo-syusin is its ability to express concisely without introducing unnatural Japanese phrasing. In designing this kind of languages, compatibility between conciseness and readability is a critical issue. Syo-syusin is fairly successful in this point. Although its use has been limited within a university, Syo-syusin is worth studying as a language model.

4.3 Mind

Mind was designed by A. Katagiri and its processing software is sold by his company, RGY Corporation. The software package, which runs on personal computers such as NEC PC9800 Series, Fujitsu FM Series, and hitachi 2020, is actually the only Japanese programming language processor currently marketed. The language is based on the design of FORTH, a stack based language, mainly for PC use. FORTH adopts the reverse Polish notation for writing mathematical expressions. The fact that the word ordering of reverse Polish expressions fits well to Japanese convention was the original motivation for Katagiri to attempt designing a Japanese-based language on FORTH.

One of the key features of Mind is its distinctive use of kanji, katakana, hiragana, and Roman letters. Kanji, katakana and Roman letters are used for spelling identifiers and hiragana can be inserted freely except in places where specific keywords spelled in hiragana that define cases (subject, object, direction-to, direction-from, etc.) are expected. The processor simply ignores hiragana except those keywords, which allows natural Japanese expressions without any load to the processor.

The processing package includes functions for debugging, graphics, mouse manipulation and communication, which make it possible to write practical-scale software in Mind. The package has sold several thousand copies and has shown that a Japanese-base programming language has a potential of acquiring wide use.

4.4 YPS and Chart Languages

Many structured chart notations for software design and programming have been proposed and are being used in Japan such as PAD, HCP, PSD, and YAC II. Many of them have drawing and description support tools, in which Japanese expressions are allowed.

YPS is one of such systems developed by Fujitsu. It is based on YAC II chart, on which a Japanese-based formal language (or language generator) is defined. The description can be transformed into programs in an appropriate language. Currently, three versions are available: systems for Cobol, Fortran and C. It has been successfully used for developing large-scale system software.

5. Experiment

We conducted an experiment of writing programs using Japanese programming languages. The purpose of this experiment was to evaluate the significance of Japanese programming in general and not to compare individual languages or programming systems.

We chose three languages, Nihongo AFL, Syo-syusin, and Mind by the following criteria: 1) manuals that describe language specifications are available; and 2) language processors (compilers or interpreters) have been developed. Although the three languages satisfy these conditions, only the processor of Mind was available for this experiment. Thus, the evaluation of programs written in the experiment was done mainly by manual inspection.

5.1 Example Problems

We chose three example problems, considering the following points:

(1) The size of programs should not be so large, but say between 50 to 100 lines of statements;

(2) The problem should not be too simple, including appropriate logical complexity;

(3) Three problems should be chosen from different areas;

(4) Problems can be simply specified so that there is little room for misunderstanding or arbitrary interpretation.

Selected problems are as follows.

[Problem 1] Kana Frequency Counting
Given a text of kana, output frequency table of kana appearing in the text in descending order.

[Problem 2] Date Transformation
First, write a subroutine that determines a given year to be a leap year or not. Then, given a year and a sequence number of a day of the year, output the month and the day, using the previous subroutine.

[Problem 3] Prime Number Enumeration
Given a natural number n, output all prime numbers between 1 to n.

5.2 Subjects

Three persons were chosen as subjects. We could have selected them from those with no programming experience, but in that case, we were afraid, it would be difficult to determine the criteria for selecting subjects, experiment design, and the way of managing the experiment. Therefore, we decided to choose subjects from software engineers with enough programming experience except Japanese programming. As a matter of fact, the three subjects are none other than the members of this projects.

Table 3 shows the profile of the subjects.

Table 3 Profile of subjects.

Subject	Sex	Experience	Fluent languages
A	male	18 years	Lisp, Fortran, PL/I
B	female	6 years	Fortran, C, Basic
C	male	5 years	Prolog, Fortran

Table 4 Combinations of subjects, problems and languages (A, B, C are subjects.).

	Mind	Syo-syusin	Nihongo AFL
Problem 1	A	B	C
Problem 2	C	A	B
Problem 3	B	C	A

Table 5 Results of the experiment—working hours.

legend:
(unit: hours)

Subject	Total hours	Manual studying hours / Programming/debugging hours

(a) working hours by language and by problem

	Mind			Syo-syusin			Nihongo AFL			Total	
Prob. 1	A	3.5	1 2.5	B	13.5	8 5.5	C	7	2 5	24	11 13
Prob. 2	C	3	1 2	A	3	1.5 1.5	B	5.5	3 2.5	11.5	5.5 6
Prob. 3	B	11.5	8 3.5	C	4.5	1.5 3	A	3	1 2	19	10.5 8.5
Total		18	10 8		21	11 10		15.5	6 9.5	54.5	27 27.5

(b) total working hours by subject

Subject	A		B		C	
Total	9.5	3.5 6	30.5	19 11.5	14.5	4.5 10

5.3 Experiment Design

Our experiment is composed of three problems, three languages and three subjects. Exhaustive combination will make 27 cases, but it will not bring significant results because the order of assignment will strongly affect the result owing to learning effects.

Thus, we designed the experiment to let each subject write three programs, each in a different language. See Table 4 for actual assignments.

5.4 Results

Measured data of this experiment are working hours for each task, size of programs (in number of lines of code), and numbers of errors for each program. Since the sample size is too small to give significant statistical analysis, following results should be interpreted with care.

(1) Working hours

We collected data of hours spent for studying each language through reading manuals and for developing programs (problem analysis, programming, and debegging) by subject and by program. Table 5 shows the results.

The results indicate the following trends.

—Time for learning is generally very short.

—Time considerably varies by subject.

—There is no significant differences of time between the three languages.

(2) Program size

The number of program lines is certainly not a perfect measure for evaluating size but still it is a most convenient measure especially in comparing program size over different languages. The results are shown in Table 6.

Table 6 Result of the experiment—program size (unit: lines of code).

(a) program size by language and by problem

	Mind		Syo-syusin		Nihongo AFL		Total
Prob. 1	A	84	B	51	C	65	200
Prob. 2	C	37	A	38	B	65	140
Prob. 3	B	54	C	44	A	32	130
Total		175		133		162	470

(b) total program size by subject

Subject	A	B	C
Total	154	170	146

There are few particularly interesting findings but the followings can be observed.

—Problem 1 is significantly larger than the others.

—Comparing productivity defined by a number of lines divided by developing hours, Problem 1 and Problem 3 show similar values (15.4 and 15.3), but the value for Problem 2 is much larger (23.3). It implies that Problem 2 has a simple logical structure.

—There is no significant difference by subjects.

—Syo-syusin seems to produce more concise programs than the other two.

(3) Errors

We counted program errors discovered by inspection. There probably remain more errors undetected. Actually, in the case of Mind programs, new bugs were found in the programs when they were executed on a com-

Table 7 Results of the experiment—number of errors.

legend:

Total errors	Major errors
	Minor errors

(a) number of errors by language and by problem

	Mind			Syo-syusin			Nihongo AFL			Total	
Prob. 1	A	2 (4)	1 (1) 1 (3)	B	9	4 5	C	0	0 0	11	5 6
Prob. 2	C	1 (3)	0 (0) 1 (3)	A	3	0 3	B	2	0 2	6	0 6
Prob. 3	B	4 (5)	1 (0) 3 (5)	C	0	0 0	A	1	0 1	5	1 4
Total		7 (12)	2 (1) 5 (11)		12	4 8		3	0 3	22	6 16

note) all errors were detected by manual inspection except Mind, for which numbers in ( ) show additional errors found by execution.

(b) total number of errors by subject

Subject	A		B		C	
Total	6	1 5	15	5 10	1	0 1

puter.

Numbers of errors are shown in Table 7. We distinguished major errors and minor errors but the distinction was somewhat arbitrary.

The results can be summarized as follows.

—Difference by subjects turned out to be the greatest.

—Errors found by running the Mind programs suggest that there are considerable errors not found by inspection. They include the cases where errors are caused by ambiguous or imprecise descriptions in the manual.

—Comparing errors by language, Nihongo AFL programs have the fewest errors. This can be explained at least partly by: 1) the subject *B*, who produced the largest number of errors, was assigned problem 2 for Nihongo AFL but that problem is the easiest among the three; 2) the description of the language specification given in its manual is rather too simple and sometimes it is difficult to judge if certain expressions are correct or not.

6. Discussions

6.1 Current Status of Japanese Programming

The quantitative results of the experiment described in the previous section should be considered with care because the sample size is small. Observations obtained from the analysis of error causes and comments given by the subjects during the process of programming as well as reviewing the programs may be more important in clarifying characteristics of each language and advantages/disadvantages of Japanese programming in general.

General conclusions are as follows.

(1) Definite advantages of Japanese programming were recognized, i.e.:

a) natural naming of variables and modules are possible;

b) it is easy to read and understand programs written by others;

c) it is relatively easy to learn.

These were not proved statistically but the fact that all the subjects, who had been rather suspisious about the significance of Japanese programming, actually enjoyed their programming experience seems to indicate its usefulness well.

(2) Some problems were observed in the current Japanese programming language design:

a) More concise expressions are desirable for writing logically complicated algorithms and mathematical formula.

b) Some of the current languages attempt to sup-

port the concept of software components reuse through increasing vocabulary of "words" (modules), but there remains a lot to be improved especially from the aspects of module independence and the interface for retrieval.

c) One of the key factors for enhancing Japanese-likeness is natural usage of particles ("joshi"). If designed properly, particles will allow relatively flexible word sequence. A related problem is word punctuation. Japanese are normally written without any space between words. It may not be appropriate to eliminate spaces between words entirely from programs but what should be the units for chunking is yet to be explored. Certainly, a particle should be glued to a noun or noun phrases, but there remain many other factors to be considered.

d) More efforts should be made to decrease input load, e.g. abbreviation of keywords and user-defined identifiers (some kind of abbreviation is adopted in Mind).

6.2 Language Design Issues

The discussions above suggest that the following issues should be tackled in the future Japanese-based programming language design.

(1) Fundamental language specifications

Solutions must be explored to determine what are good design principles for language constructs such as:

(a) notation, e.g. way of punctuation and proper use of alphanumeric, katakana, hiragana, kanji, and special symbols.

(b) basic language elements, e.g. particles, relation between predicate type elements like procedures and functions and noun type elements like variables and data, and word order.

(c) syntax and functions for complex programming constructs, e.g. formula, control structures (repetitive and conditional), and modules.

(2) New programming concepts such as object-oriented programming

The idea of constructing software component library can be interpreted as accumulating vocabulary in the context of Japanese-based programming. Since this idea fits well to the object-oriented concept, it should be worth trying to design Japanese-based programming language based on object-oriented principle (SPPM being developed by Morimoto et al [18] is one example of exploring this approach).

(3) Natural language processing

Natural language processing techniques are indispensable for making expressions closer to natural Japanese. There are already some research works that applied natural language processing technology of AI to specification writing system. It must be a long way to realize communication with computers in a really natural human language but technologies developed in the natural language processing research community should be introduced into specification and programming language design step by step.

(4) Automatic programming

Many of the existing Japanese-based specification languages set as their final goal the generation of programs from specifications written in those languages. It is none other than the objective of automatic programming and thus it must be worthwhile to review the research of automatic programming from the standpoint of Japanese programming.

Acknowledgements

This paper is based on the research work sponsored by Mechanical and Social Systems Foundation [12], [28].

The author firstly thanks Iwao Kokubo and Kyoko Makino of Mitsubishi Research Institute, Inc., who collaborated in this project. He would also like to thank to those pioneers of Japanese-based language or system development, who accepted our interviews for this project, including Nobumasa Takahashi, Izumi Kimura, Shizuo Mizutani, Kenichi Ueda, Jun Sugano, Akira Katagiri, Tsutomu Shiino, Yoshio Mouri, Kenji Sugiyama, Masaaki Shimazaki, Masahiro Udo, and many others.

References

1. Chigira, E., Nagamatsu, Y. and Kobayashi, M. Software Development by Reusing System Development Specifications, *Hitachi-Hyouron*, **69**, 3 (1987) 249–254, in Japanese.
2. Fujitsu Ltd. *YPS (YAC II Programming System)*, a pamphlet by Fujitsu, in Japanese.
3. Gyohtoku, I., Doi, H., Suzuki, Y. and Uehara, K. A Method for Analyzing Specifications Used in the Programming Generation System SAGE, *Proc. 35th National Conf. IPS Japan* (1987), 1003–1004, in Japanese.
4. Harada, M. and Sinohara, Y. A Program Generator ARIES/I—By Automatic Fabrication of Reusable Program Components—, *J. IPS Japan* **27**, 4 (1986) 417–424, in Japanese.
5. Heidorn, G. E. Automatic Programming Through Natural Language Dialogue: A Survey, *IBM J. Res. Develop.*, **20**, 4 (1976) 302–313.
6. Iwamoto, K., Nishitani, Y. and Wada, T. Automatic Program Generation System, *NEC Gihou*, **40**, 1 (1987) 35–38, in Japanese.
7. Katagiri, A. *Japanese Programming Language: Mind, Operation Manual*, Micro Software Associates (1987), in Japanese.
8. Katagiri, A. and Maezono, Y. *Japanese Programming Language: Mind, Programming Manual*, Micro Software Associates (1987), in Japanese.
9. Katagiri, A. and Maezono, Y. *Japanese Programming Language: Mind, Library Programming Manual*, Micro Software Associates (1987), in Japanese.
10. Kimura, S. Japanese Program, *Proc. 1st National Conf. JSAI*, (1987), 465–468, in Japanese.
11. Matsushita Giken Co., Ltd. *Nihongo AFL Users' Guide* (1983), in Japanese.
12. Mechanical and Social Systems Foundation *Survey Report on Japanese Programming* (1988), in Japanese.
13. Miller, L. A. Natural Language Programming: Styles, Strategies and Contrasts, *IBM Syst. J.*, **20**, 2 (1981) 184–215.
14. Minami, T., Sugio, T., Takeuchi, A. and Shiino, T. A Japanese-based Specification Description Language NBSG, *JPSJ WG Softw. Eng. (WGSW)*, **30**, 2 (1983), in Japanese.
15. Mizutani, S. Design Principles of a Programming Language "Syo-syusin", *IPSJ WG Prog. Lang. (WGPL)*, **5**, 3 (1986), in Japanese.
16. Mizutani, S. *Syo-syusin—Language Specifications*, Tokyo Women's College (1986), in Japanese.
17. Mizutani, S. *A Programming Language Syo-syusin Users' Guide*, Tokyo Women's College (1986), in Japanese.

18. MORIMOTO, Y., OYANAGI, S. and NAKAYAMA, Y. Japanese Language Programming with Accumulating a Vocabulary, *Proc. IEEE COMPSAC'87* (1987), 586–591.
19. MURAKAMI, N. et al., YAC II Editor, in *Structured Editor*, Kyoritsu Shuppan (1987), 135–146, in Japanese.
20. OHNISHI, J., AGUSA, K. and OHNO, Y. Requirements Frame for Requirements Definition, *J. IPS Japan*, **28**, 4 (1987) 367–375, in Japanese.
21. SHIINO, T., TAKEUCHI, A. and SUGIO, T. NBSG: A Specification Description Language Based on Japanese, *Proc. 26th National Conf. IPS Japan* (1983), 547–548, in Japanese.
22. SUGANO, J., HONDA, K., OKAMURA, Y. and UEDA, K. Nihongo AFL that Improves Man-machine Interface Using Natural Language, *Proc. IPSJ Symposium on Computer Human Interface* (1983), 53–60, in Japanese.
23. SUGANO, J. and UEDA, K. Nihongo AFL, in *Wordprocessors and Japanese Language Processing*, Kyoritsu Shuppan (1985), 243–251, in Japanese.
24. Sugio, T., TAKEUCHI, A. and SHIINO, T. Procedural Description in NBSG, a Specification Description Language Based on Japanese, *IPSJ WG Softw. Eng. (WGSE)*, **34**, 13 (1984), in Japanese.
25. SUGIYAMA, K., AKIYAMA, K., KAMEDA, M. and MAKINOUCHI, A. An Experimental Interactive Natural Language Programming System, *Trans. IEICE*, **J67-D** (1984), 297–304.
26. SUGIYAMA, K., KAMEDA, M., AKIYAMA, K. and MAKINOUCHI, A. Understanding of Japanese in an Interactive Programming System, *Proc. 10th Int. Conf. Comp. Ling.* (1984), 385–388.
27. SUZUKI, Y., DOI, H., GYOHTOKU, T. and UEHARA, K. A Method of Understanding Specification in the Program Generation System SAGE, *Proc. 35th National Conf. IPS Japan* (1987), 1005–1006, in Japanese.
28. TAMAI, T., KOKUBO, I. and MAKINO, K. An Experiment and Analysis of Japanese Based Programming, *Proc. 5th Conf. JSSST* (1988), 81–84, in Japanese.
29. TOYODA, J. and UEHARA, K. Automatic Program Synthesis based on Natural Language Understanding, *J. JSAI*, **2**, 3 (1987) 289–298, in Japanese.
30. UEHARA, K., FUJII, K. and TOYODA, J. A Technique for Prolog Program Synthesis from Natural Language Specification, *Computer Software*, **3**, 4 (1986) 55–64, in Japanese.

(Received May 11, 1989)