Construction of a modular and portable translation system

Fujio NISHIDA, Yoneharu FUJITA and Shinobu TAKAMATSU
Department of Electrical Engineering,
Faculty of Engineering,
University of Osaka Prefecture,
Sakai, Osaka, Japan 591

1. Introduction

In recent years the study of machine translation has made great advances and the translation system has been larger and complicated with augmenting facilities. Furthermore, most recently, many powerful workstations have been developed and various MT systems for special purposes are ready to be mounted on these workstations.

In such a state of affairs it will be needed that many MT systems are reorganized or re-constructed on a program module basis for easy modification maintainance and transplantation.

This paper has two purposes. One of them is to show a method of constructing an MT system[2)3)] on a library module basis by the aids of a programming construction system called L-MAPS.[4)] The MT system can be written in any programming language designated by a user if an appropriate data base and the appropriate processing functions are implemented in advance. For example, it can be written in a compiler language like C language, which is preferable for a workstation with a relative slow running machine speed.

The other purpose[4)] is to give a brief introduction of a program generating system called Library-Module Aided Program Synthesizing system (abbreviated to L-MAPS) running on a library module basis. L-MAPS permits us to write program specifications in a restricted natural language like Japanese and converts them to formal specifications. It refines the formal specifications using the library modules and generates a readable comment of the refined specification written in the above natural language every refinement in option. The conversion between formal expressions and natural language expressions is performed efficiently on a case grammar basis.

2. Overview of the MT system organization[2)3)]

Our machine translation system is constructed on the intermediate expressions based on universal subframes of predicates and predicative nouns. It aims at a multilingual transfer system. Up to now, however, no universal precise semantic category system over various languages has been constructed yet, and our MT system is compelled to work rather on a bilingual basis in the selection of equivalents.

The first version of the parsing was written in an extended version[2)] of LINGOL[1)5)]. It has an advice part and a semantic part in each rewriting rule. Both parts of them permit users to describe any Lisp program for designating details of the reduction procedures. These techniques used in LINGOL and ATN seem apparently convenient. However, they often make the data part inseparable from the program part and bring an MT system to much complexity, and accordingly, prevents applicability of the programs of the MT system to another translation between other languages.

Recently, a revised version of our MT system has been constructed. The main program or procedural part consists of unification and substitution, while the data part consists of frame knowledge rewriting rules and word dictionaries.

Rewriting rules with arguments describe the details of the syntactic and semantic structure of the language explicitly. For example, the predicate part of the Hornby's verb pattern VP13A of English is written as follows:

$$\mathrm{PREDP}(\ \mathrm{PRED}\text{-}c_0\text{:}t_0,\mathrm{MOD}\text{:}m,\ k_1\text{-}c_1\text{:}t_1,k_2\text{-}c_2\text{:}t_2)$$
$$\text{--> VP}(\ \mathrm{PRED}\text{-}c_0\text{:}t_0,\mathrm{MOD}\text{:}m)\ \mathrm{NP}(\ k_1\text{-}c_1\text{:}t_1)$$
$$\mathrm{PP}(\ k_2\text{-}c_2\text{:TO-}t_2)$$

where PREDP, VP, NP and PP denote a PREDicate Phrase, a Verb Phrase, a Noun Phrase and a Prepositional Phrase respecively. k-c:t denotes a triple of a case label, a semantic category and a term and m denotes various modal values such as tense and aspect. These rewriting rules are tabulated in several tables for an efficient processing.

The parsing system first applies the syntactic part of a rewriting rule to that of a handle in a reduction sequence of a given source sentence. If the system finds a unifiable rewriting rule, it checks whether the semantic part is unifiable. The category check of a term in a handle for the case-frame condition is processed by a special kind of unification under an interpretation that the term category in a rule is a restricted variable.

The intermediate expression of the handle part is constructed by substituting the unified terms for the arguments in the left-hand side of the rewriting rule.

3. The L-MAPS system and language conversion

The L-MAPS system is costructed on a fundamental library module basis. When a user gives a specification by referring to the libraly module, L-MAPS searches applicable library modules and refines the specification by linking several modules or replacing it by the detailed procedure called the Operation Part of an applicable library module.

The formal specifications of programs as well as the library modules are generally difficult for users to read and write correctly though they are efficient and rigorous for machine operation. Hence, it is desirable to rewrite the formal specification in a natural language. L-MAPS performs a conversion between a restricted natural language expression and a formal language expression through the intermediate (or the internal) expression of the natural language expression with the aids of case labels.

The conversion between a restricted natural language expression and the intermediate expression can be done in a similar manner to the conversion carried out in machine translation.

Formal specifications generally have different forms from those of the intermediate expressions.

The intermediate expression of a sentence takes the following form :

```
(PRED:tp, K1:t1, ....., Kn:tn)          (1)
```

where PRED K1 and Kn are case labels and tp t1 and tn are terms corresponding to their cases. On the other hand, a procedure expression appearing in formal specifications as well as in a heading of each library module has the following form:

```
proc-label(K1':t1', K2':t2',...., Kn':tn')  (2)
```

where the procedure name plays a role of the key word and it is constructed from the predicate term the object term and others of the intermediate expression. It is used for retrieving library modules applicable to a given specification.

L-MAPS performs the conversion between the intermediate expression(1) and the procedural expression(2) by a method similar to the case structure conversion between different languages.

The conversion is applied not only to the construction of a formal specification from an informal specification written in restricted Japanese or English but also to the generation of Japanese or English comments on the refined specifications generated by L-MAPS itself.

4. Modularization of programs

The revised MT system is reconstructed based on library modules by the aids of L-MAPS. Each library module has a structure as shown in Table 1.

Table 1 A part of library modules

```
PROC: HANDLE_REDUCE(SO:reduced_sequence, OBJ:handle,
                    INSTR:reduction_rule,
                    GOAL:new_reduced_sequence)
IN: GIVEN(OBJ:reduced_sequence,handle,
                    reduction_rule)
OUT: REDUCED_FORM(OBJ:new_reduced_sequence)

ENTITYTYPE:.............................
OP: RULE_APPLY(OBJ:reduction_rule, PARTIC:handle,
                GOAL:reduced_symbol)
    BRANCH1(COND:EQUAL(reduced_symbol,NULL),
            OP:RETURN(FAIL))
    FOR(COUNT:n,FROM:1,TO:-(stack_pointer,
                        symbol_number_of_handle),
        OP:COPY(OBJ:reduced_sequence(n),
                GOAL:new_reduced_sequence(n)))
    COPY(OBJ:reduced_symbol,
         GOAL:new_reduced_sequence(+(n,1)))
    RETURN(TRUE)
```

The heading of each module has both the procedural expression and the input-output predicate expression (abbreviated to the IO expression). Program specifications given by a user can call a module by using one of these headings.

The IO expression consists of a pair of an input and an output predicate and asserts that the output predicate holds under the given input predicate.

The IO expressions are used to automatically link some modules for a specification and to check linking of modules specified by their procedural expressions.

The type part describes the types of regions structures and roles of input output or local variables.

The OP part describes the procedures for the function assured in the heading part. The procedures are described in a little more detail by using the headings of more fundamental modules.

Control statements are represented by using a prefixed-form of Pascal called the General Control Expression (abbreviated to GCE) here. The control statements are expanded into a programming language such as Lisp and C designated by users. Some conversion rules are shown in Table 2.

Table 2 Conversion rules to objective languages

```
IF_THEN(COND: p, OP: s.....................(GCE)
    (COND (p s))  .........................(Lisp)
    IF(p) s ;  ............................(C)
IF_THEN_ELSE(COND: p, OP1: s1, OP2: s2....(GCE)
    (COND (p s1) (t s2))  .................(Lisp)
    IF(p) s1 ELSE s2 ;  ...................(C)
FOR(COUNT: i, FROM: m, TO: n, OP: s).......(GCE)
    (SETQ i m)
    (LOOP ()(COND ((EQUAL i n) (EXIT-LOOP))
                  (T s (SETQ i (ADD1 i)))
          )).............................(Lisp)
    FOR(i = m; i <= n; i++) s  ...........(C)
 ....................................
 ....................................
```

Corresponding to the general control language, general data structures are also introduced. If refined specifications are designated to be expanded to a programming language which do not have the corresponding data structures inherently, the equivalent data structures and the access function must be implemented in advance by combining the fundamental data structures in the language. For example, if Lisp is designated as the programming language, a struct which appears in a general data structure of a specification is expanded to the correspondig associative list structure and the lists can be accessed by associative functions.

5. Refinement and Expansion by the L-MAPS system

Figure 1 shows an outline of the processing by L-MAPS.

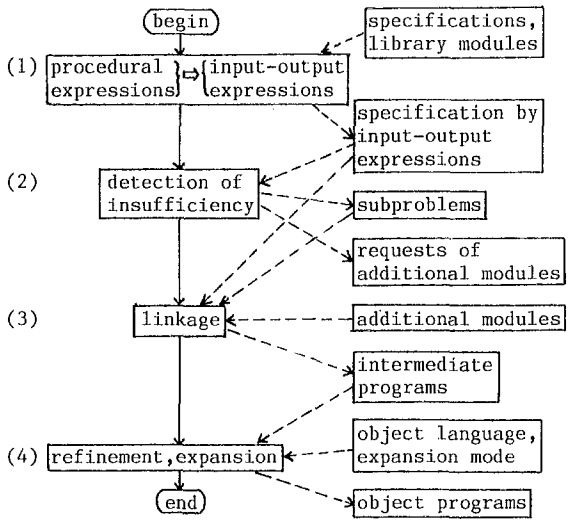

Fig.1 The processing by L-MAPS

In refinement, L-MAPS tries to unify the heading of an expression in a given specification and the corresponding heading of a library module. If L-MAPS succeeds in the unification, it constructs a more detailed description of the specification by using the unified Operation Part of the module .

The refined part with a more detailed description can be substituted directly for the original part in the specification or can be called in a form of a subprogram as a procedure or a closed subroutine from the specification part. One of them is selected by the user.

The principal part of the above refinement is unification of a specification and the heading of a library module. When the arguments of a module are confined to the individual variables and the number of arguments of a function is confined to a constant the unification can be carried out by an ordinary unification of the first order logic. Otherwise, the unification procedure for the second order logic is needed.

L-MAPS has a unification procedure for the second order logic. However, the unification procedure is confined to a unilateral unification from a module to a specification in which each symbol is interpreted as a constant under the condition that any substitution for the symbol in the specifications is forbidden. Accordingly, the unification procedure can be much simplified for practical purpose.

Fig.2 shows parts of a given specification written in the restricted English for a parsing program of English sentences and Fig.3 shows a part of the generated formal specification.

```
  ..........
for i from 1 to m
    js:=j
    search handles from reduced_sequences(i)
           by using reduction_rules, and
    store it in handle(1..k) and rule(1..k)
    if k is greater than 0
       for n from 1 to k
           reduce handle(n) in reduced_sequences(i)
                  by using rule(n), and
           store the result in
                        new_reduced_sequences(j)
  ..........
```

Fig.2 The informal specification for a parsing program

```
  ..........
FOR(COUNT:I,FROM:1,TO:M,
    OP: =(JS,J)
       HANDLE_SEARCH(SO:REDUCED_SEQUENCES(I),
                     INSTR:REDUCTION_RULES,
                     GOAL:(HANDLE(1..K),RULE(1..K)))
       BRANCH1(COND:>(K,0)
               OP:FOR(COUNT:N,FROM:1,TO:K,
                   OP:HANDLE_REDUCE
                      (SO:REDUCED_SEQUENCES(I),
                       OBJ:HANDLE(N),
                       INSTR:RULE(N),
                       GOAL:NEW_REDUCED
                                  _SEQUENCES(J))
                      )
            )
  ..........
```

Fig.3 A part of formal specifications

L-MAPS refines the formal specification by referring to library modules such as shown in Table 1 and generates a refined specification and the comment shown in Fig.4.

```
RULE_APPLY(OBJ:REDUCTION_RULE,PARTIC:HANDLE,
           GOAL:REDUCED_SYMBOL)
BRANCH1(COND:EQUAL(REDUCED_SYMBOL,NULL),
        OP:RETURN(FAIL))
FOR(COUNT:N,FROM:1,TO:-(STACK_POINTER,
                        SYMBOL_NUMBER_OF_HANDLE),
    OP:COPY(OBJ:REDUCED_SEQUENCE(N),
            GOAL:NEW_REDUCED_SEQUENCE(N))
 ..........
```

Fig.4(a) A part of the refined specification

```
    apply the rule to the handle, and
    store the result in a reduced_symbol.
    if the reduced_symbol is null return(fail).
    for n from 1 to stack_pointer-symbol_number
                        _of_handle
        copy reduced_sequence(n)
             into new_reduced_sequence(n).
    ..........
```

Fig.4(b) The comments of the refined specification in Fig.4(a)

In the refinement process global optimizations are tried to be done at the user's option. Some of them are rearrangement of conditional control statements and fusion of several iteration loops into one loop.

6. Conclusion

The translation system is constructed on a modular basis consisting of 24 application modules and 30 basic modules by refining and expanding specifications by the aids of the L-MAPS system consisting of about 1000 lines of Lisp statements. The generated translation-system programs is about 1000 lines in both C language and Franz Lisp. Besides various advantages due to the modularization, the translation speed is almost the same as that of the old version in LISP. Furthermore, the translation speed in C language is about three times faster than that of Franz Lisp at a compiler mode in English-Japanese translation.

References

1)Pratt,V.R.: LINGOL-A Progress Report, IJCAI4, 422-428 (1975).

2)Nishida,F., Takamatsu,S. and Kuroki,H.: English-Japanese Translation through Case-Structure Conversion, COLING-80, pp.447-454 (1980).

3)Nishida,F. and Takamatsu,S.: Japanese-English Translation through Internal Expressions, COLING-82, pp.271-276 (1982).

4)Nishida,F. and Fujita,Y.: Semi-Automatic Program Refinement from Specification Using Library Modules, Trans. of IPS of Japan, Vol.25, No.5, pp.785-793,(1984), (Written in Japanese).

5)Tanaka,H., Sato,T and Motoyoshi,F.: Predictive Control Parser: Extended LINGOL, 6th IJCAI, Vol.2, pp.868-870, (1979).