

機械翻訳用ソフトウェアGRADEにおける Utilispの改良と問題点

中村順一、辻井潤一、長尾真（京大・工）

1. はじめに

Utilisp[1]は、大学を初め、各種の研究機関等で多く使用されているが、筆者らも、Utilispを使用して、科学技術論文の抄録の日→英／英→日機械翻訳システム[2]の研究・開発を行っている。実用的で高品質な機械翻訳システムを実現するためには、翻訳のための文法や辞書等に大量のデータが必要となるので、小規模なシステムでは考えられないソフトウェア上の問題点が開発の過程で発生し、Utilispを以下の点で強化する必要が生じた。

- (1) 漢字コードを扱えるようにすること。
- (2) 直接アクセス型のファイルを扱えるようにすること。
- (3) セル空間を拡げるためにガーベッジ・コレクションの方法を変更すること。
- (4) 複雑なプログラムを改良するためのデバッグ・ツールを作成すること。

本稿では、開発している機械翻訳システム（GRADEシステム[3,4]）の概要と、その開発の過程で必要になったUtilispの強化点について述べる。また、この機械翻訳システムを他のLisp（Zetalisp）へ移植することも行ったので、その際に使用した手法についても述べる。

2. GRADEの概要

2.1. 機械翻訳システム

機械翻訳システムは、ある言語（原言語）で書かれた文章を他の言語（目標言語）の文章に変換するためのソフトウェア・システムである。機械翻訳システムにおける処理の流れは、一般に、次の3つの段階に分れている（図1）[5,6]。

(1) 原言語の構文・意味解析

入力文中の格単語相互の構文的関係（その単語がどの単語を修飾するのか）および意味的關係（たとえば、格助詞「で」が表現している意味が、場所であるか、道具であるか、等）を分析する。

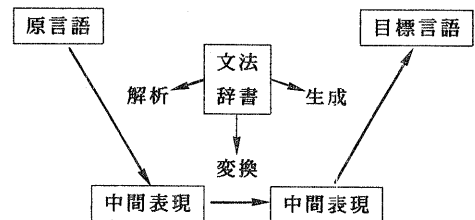


図1 機械翻訳処理の流れ

(2) 原言語から目標言語への構文・意味変換

各単語や句に対して訳語を与える。この場合、適切な訳語の選択だけでなく、たとえば、日本語らしい表現を英語らしい表現にする等の文体に関する処理も必要である。

(3) 目標言語への構文・意味生成

(2)で得られた構造は、翻訳結果の構文的な構造を完全には表現していないので、態変換（能動態で訳出するか、受動態で訳出するか）や、文体の決定（英語であれば、関係詞節を用いるか、to-不定詞句を用いるか等）を行い、訳文を得る。

これらの翻訳処理は、単語の構文・意味的な情報を蓄えてある「辞書（dictionary）」と、処理の方法を記述した「文法（grammar）」により行なわれる。自然言語が持つ多様性を扱うためには、辞書・文法とも大きくて複雑なものになるので、大規模な辞書や文法を作成・保守するための各種のソフトウェアが必要となる[5]。本論文で述べるGRADEシステムは、そのような目的のために設計・開発されている。

2.2. GRADEシステムの開発目的

GRADEシステムは、科学技術庁の機械翻訳プロジェクト（Muプロジェクト[2]）のために開発されたものである。Muプロジェクトは、図2、図3に示すような、日本科学技術情報センタ（JICST）が発行している論文抄録（日→英）と、INSPECの論文抄録（英→日）を対象とした、実用的な機械翻訳シ

テムを開発することが目標になっており、1982年から4年間のプロジェクトとして、電子技術総合研究所（ETL）、工業技術院計算機センタ（RIPS）、日本科学技術情報センタおよび京都大学が協力して、研究・開発を行っている。

このプロジェクトの目標は、科学技術論文の速報用という観点から、前編集（Pre-Edit、機械翻訳が簡単になるように、入力文を修正する）や後編集（Post-Edit、機械翻訳の出力を、完全に正しい訳文に修正する）なしでも、論文抄録の速報用という点で十分に高品質な翻訳結果を得ることができる機械翻訳システムを開発することである。このため、翻訳のための文法には、基本的な規則だけでなく、文の表層や意味的な情報を多用した規則（ヒューリスティック規則を含む）が数多く必要になる。

2.3. 文法記述用言語GRADEの特徴

GRADEシステムは、翻訳用の文法を記述するための言語GRADE[3,4]と、GRADEで表現された文法をテスト・実行するための各種のソフトウェアから

構成されている。ここでは、多様な言語現象を扱うためにGRADE言語が備えている特徴について述べ、次項で、GRADEシステムの構成を述べる。

(1) 木構造変換型の書き替え規則

GRADEでは、文法は、文脈自由文法（CFG）等と同様、書き替え規則（プロダクション・ルール）の集合として表現されるが、CFGとは異なり、翻訳処理の中間の状態を木構造（節点に属性名と属性値の組のリストを持ったもの）で表し、文法中の各書き替え規則（文法規則）は、木構造を木構造に変換する形式で記述される。書き替え規則の条件部分の検査は、木構造のパターン・マッチングにより行われるが、このパターン・マッチングは、任意要素や順序任意性を許すなど、柔軟で強力なものになっている。このため、CFGのように、隣接した構文要素に関する情報だけでなく、文全体を見渡したような条件を記述することも容易であり、非常に詳細な文法規則を記述することが可能になっている。また、解析・変換・生成のすべての文法を同一の枠組で記述することもできる。

NO. 201: E82060054_4_1 (06/20/85, 06/20/85, 06/20/85)

この定義は、抽象的なダイナミカルシステム表現を持つnポート回路網へも適用される。

This definition is also applied to n-port networks with abstract dynamical system expression.

NO. 202: E82060054_5_1 (06/20/85, 06/20/85, 06/20/85)

集中定数、非線形nポートは有限次数のダイナミカルシステムで表現でき、受動性と無損失性の条件を状態方程式を用いて求める。

Lumped constants and nonlinear n-port can be expressed by dynamical systems of finite degrees, and they obtain conditions of passivity and losslessness by using state equations.

NO. 203: E82060054_7_1 (06/20/85, 06/20/85, 06/20/85)

この条件は、よく知られている線形、時不変、集中定数nポート回路網の時間域、周波数域の受動性、無損失性条件の非線形な場合への拡張となっている。

This condition becomes the expansion of passivity and losslessness conditions for nonlinear cases of hour region and frequency region of linear and time-invariable well known concentrated constant n-port networks.

図2 日→英システムの入力文と訳文の例

B82000019_4_1 (形解 08/07, 解析 08/17, 依存 08/17, 変換 08/17, 生成 08/19, 形合 08/19)

Abstracts of individual papers can be found under the relevant classification codes in this or future issues.

この号または将来の号において個々の論文の抄録を関連がある分類コードの下に得ることができる。

B82000020_3_1 (形解 08/07, 解析 08/17, 依存 08/17, 変換 08/17, 生成 08/19, 形合 08/19)

Seventy-five sessions were held covering a wide range of telecommunication topics.

電気通信項目の広い範囲を含んで、75のセッションが開催せられた。

B82000022_2_1 (形解 08/07, 解析 08/17, 依存 08/17, 変換 08/17, 生成 08/19, 形合 08/19)

The development and the interdependence of physics and electrical engineering in Moldavia during the 19th and 20th centuries, are presented.

19世紀および20世紀の間のモルドヴィアにおける物理学と電気工学の成長と相互依存が示されている。

図3 英→日システムの入力文と訳文の例

(2) 部分文法方式

CFGとは異なり、翻訳の各処理段階を更に細分し、文法規則の優先順序や処理の制御を文法記述上に明示することができる。細分された文法（文法規則の集合）を部分文法と呼んでいる。このことにより、翻訳のための処理を詳細に制御することが可能になっている。

(3) 辞書規則の使用

自然言語の言語現象には、単語固有のものが多く、文法が詳細になればなるほど、特定の単語専用の文法規則が増加してくる。そこで、GRADEでは、単語の辞書中にも、文法規則が記述できるようになっている。これにより、翻訳システムの改良が辞書を中心として行えるようになり、継続的なシステムの保守・改良が容易になった。

(4) 保守しやすい文法記述の形式

機械翻訳用の文法は、大規模なものである。文法の作成・保守は、多人数で長期間にわたる必要がある。そこで、Lisp等を知らない人でも、文法記述が容易に行えるよう、図4に示すような、S式ではない、記述形式を設計した。また、複雑な木構造のパターン・マッチング用のパターンを簡単に記述できるよう、木構造自体の形と、付随する制約条件を分離して記述するようにした。なお、図4に示したように、文法記述中に漢字を直接表現することも可能である。

```
move_clause.rr;
mi;
level(0,0);
order(1);
tree;

mc;
%((vadj #1 del #2 vadj1 #3)); ← 木構造の指定
vadj.j_cats = 'dsi'|'adj';
vadj1.j_cats = 'dsi'|'adj';
vadj1.j_sentence_connector = '接続詞'; ← 属性に関する条件
del.j_cats = 'del';

cr;
%(( vadj #1 vadj1 del #2 #3)); ← 結果の指定
end_rr.move_clause;
```

図4 GRADEによる文法記述の例

2.4. GRADEシステムの構成

GRADEシステムは、文法記述用言語GRADEで表現された翻訳用の文法を実行するためのソフトウェアと、翻訳システムの保守・改良を支援するための

ソフトウェアとから構成されている。GRADEシステムの構成を図5に示す。

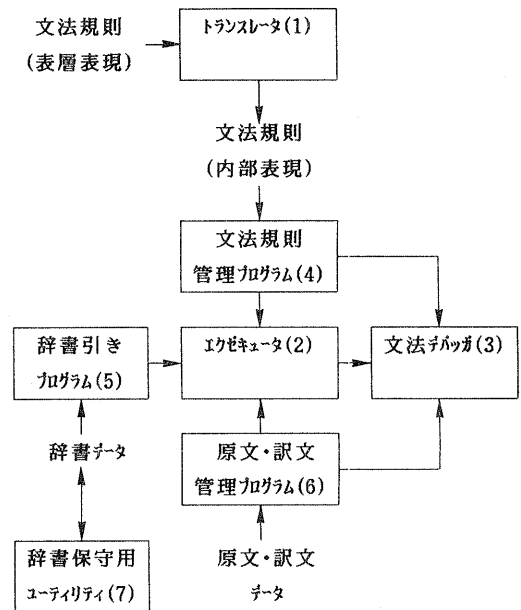


図5 GRADEシステムの構成

```
(rr 'r_move_clause
  ((("owner" = m.kume" "version" = v01l01"
    "last_update" = 8/21/85" "translated" = 08/29/85 15:18:47"
    "source" = 'ayi1111.nakamura.list(rule)')")
  rrapplier
    (nil nil nil ((n000) (vadj) (#1) (del) (#2) (vadj1) (#3)))
    (j_cats (0 . 0) lbd (1 . skip) para nil)
    (matcher
      (!cont n000
        (!nop vadj
          nil
          (!prop-check
            ((por
              (ueq (pvar vadj j_cats)
                (pvalue ((dsi))))
              (ueq (pvar vadj j_cats)
                (pvalue ((adj))))))))
          (!cont vadj
            ((!anythings #1)
              (!nop del
                (!prop-check
                  ((ueq (pvar del j_cats)
                    (pvalue ((del)))))))
                (!anythings #2)
                (!nop vadj1
                  nil
                  (!prop-check
                    ((ueq
                      (pvar vadj1 j_sentence_connector)
                      (pvalue ((接続詞))))
                    (por
                      (ueq (pvar vadj1 j_cats)
                        (pvalue ((dsi))))
                      (ueq (pvar vadj1 j_cats)
                        (pvalue ((adj))))))))
                  (!anythings #3))))
              (creation
                (construct
                  nil
                  (constpatt (vadj #1 vadj1 del #2 #3))))))))))
```

木構造と属性に関する条件
結果の指定

図6 内部表現の例

(図4の文法規則の実行用の表現)

図4に示した文法記述（表層表現）を、文法を実行するためのLispのS式（内部表現、図6）に変換する。

トランスレータにより生成された内部表現を解釈しながら、書き替え規則（文法規則）を木構造に対して適用し、文法の実行を行う。

(4) 文法規則管理プログラム

開発した機械翻訳用の文法は、非常に大きなものであり、たとえば、日英の翻訳用の文法規則の数が3,000以上ある。また、柔軟な木構造のパターン・マッチングを行うために、文法規則の内部表現が図6に示したように、かなり大きなものになった。このため、文法規則の内部表現のすべてを同時にUtilispのセル空間上に展開することが不可能になった。また、単語固有の辞書規則の数は、辞書の

tbx com> show tree in

```
ds1 <述べる> <>
|--advp <?> <>
|  |--np <?> <subject>
|  |  |--adjp <?> <>
|  |  |  |--np <?> <>
|  |  |  |  |--n <1> <支配>
|  |  |  |  |--bkk <の> <>
|  |  |  |--n <part> <支配>
|  |  |--bkk <ga> <>
|--advp <?> <>
|  |--np <磁気> <object>
|  |  |--np <?> <paraelement>
|  |  |  |--n <電気> <支配>
|  |  |  |--psym <to> <>
|  |  |--np <?> <paraelement>
|  |  |  |--n <磁気> <支配>
|--bkk <wo> <>
```

図7 木構造の表示例

図7 木構造の表示例

```

1 ((j_node_weight (6)) (j_vp (v1)) ($ 自動詞/他動詞の区分 (T))
  (j_verb_aspect (kelzok)) ($ 意味コード (記述)) (j_verb_int (ari))
  ($ 動詞活用型 (下一)) (applied (t)) (e_lex (deal with))
  (e_cat (v) (e_deep_cat (pred)) (j_cat (j-dsi)) (j_lex (述べる))
  (j_deep_tense (pres)) (j_deep_aspect (tyouji)) (ej_transferred (t))
  (j_cats (dsi)) (j_passive_no (ptn2)) (j_causat_no (ptn2))
  (j_possible no (ptn1)) )

```

規模が増大するにつれ、増加し、同様に、直接セル空間上に展開することはできない。そこで、後述するように、Utilispに直接アクセス型のファイルに対するインターフェース（VSAMインターフェース）を付加したものを使用し、文法規則を必要に応じて動的にセル空間上に展開（read）し、翻訳処理を行っている。これにより、大規模な実験が可能になったが、ガーベッジが、文法の実行により直接生じるだけでなく、文法規則の動的なローディングにより生じ、ガーベッジ・コレクションの回数が増加したことと、入力（read）を頻繁に行う必要が生じたことのため、翻訳の実行効率が低下した。

辞書内容もS式で表現されており、現在、約2万語の日本語、日英変換、英日変換、英語辞書を使用している[8]。辞書内容も文法規則と同様、VSAMインターフェースを使用して、必要な辞書引きを行っている。

機械翻訳用の文法を改良、保守していく場合、常に原文と訳文を対象とする必要がある。また、処理の一部分（たとえば、変換）だけを実行してみたい場合も頻繁に発生する。GRADEシステムでは、原文や訳文、中間結果をVSAMデータセットに保

```
tbx com> show rule reference

+ sgn_j_vp_generation in processing +

sgn_j_vp_generation
- sg_j_vp_generation
  - rr_j_vp_advlke_gen
  - rr_j_simple_vp_gen
    - sgn_j_vp_creator
      - sg_j_vp_creator
    - sgn_j_np_vp_generation
      - sg_j_np_vp_generation
    - sgn_j_vp_vp_generation
      - sg_j_vp_vp_property
      - sg_j_vp_vp_generation
    - sgn_j_vp_vp_para_generation
      - sg_j_vp_vp_para_generation
    - sg_j_mod_vp_creator
      - rr_j_mod_vp_creation1
      - rr_j_mod_vp_creation2
      - rr_j_mod_vp_aux
      - rr_j_mod_inf

unit no kosu ----- sgn      = 5
                        sg       = 7
                        rr       = 6
                        call-dic = 0
```

図8 文法規則の参照関係の表示例

存することにより、文法記述者が自由に必要な部分の翻訳実験を行えるようになっている[9]。たとえば、日英翻訳システムでは、このプログラムの支援のもとに、現在、約3000文のサンプル文を用いて文法の改良を詳細に行っている。

(7) 辞書保守用ユーティリティ

辞書の保守は、登録されている語数が多いこと、内容の誤りが発見しにくいことなどから、容易でない。そこで、辞書内容の形式検査、検索(図9)、各種の形式変換のためのユーティリティを開発した[9]。

```

;;; edic-print-summary for 'axc0650.eana.n.mtf.vsam'
;;; area: 1, lex: (e_lex)
;;; info: ((e_sem (of))).

; 136 : America
; e_sem = ((of sa))
; 139 : Argentina
; e_sem = ((of sa))
; 258 : Canada
; e_sem = ((of sa))
; 376 : EEC
; e_sem = ((of))
; 550 : IEEE
; e_sem = ((of))
; 882 : Paris
; e_sem = ((sa of))
; 956 : Rumania
; e_sem = ((sa of))
; 1140 : USSR
; e_sem = ((of sa))
; 1147 : United States
; e_sem = ((of sa))
; 1462 : agency
; e_sem = ((of))
; 1921 : association
; e_sem = ((of))
; 2320 : board
; e_sem = ((of))
; 2659 : center
; e_sem = ((of))
; 3148 : company
; e_sem = ((of))
; 5510 : facility
; e_sem = ((of sa))
; 6444 : group
; e_sem = ((of))
; 7270 : institute
; e_sem = ((of))
; 14462 : university
; e_sem = ((of))
; 15029 : world
; e_sem = ((sa of))

```

↑
通し番号
(辞書から意味マークと(2 OF(組数)
を持つ語を取り出した例)

↑
見出し語

↑
内容

図9 辞書の検索例

3. Utilispの拡張点

機械翻訳用のGRADEシステムを開発する過程で、漢字の扱い方、多量のデータの扱い方等の点について、オリジナルのUtilispの機能を強化する必要が生じた。以下では、この点について紹介する。

3.1. 漢字化

日本語を含む機械翻訳システムを考えた場合、漢字コードの扱い方の問題は、避けることができない。オリジナルのUtilisp(富士通版)では、ストリング中では漢字コード(JEFコード)を扱うことができるが、シンボルでは、関数INTERNが英小文字を英大文字に変換するため、扱うことができなかった。また、ストリングの場合も、シフト・コードを意識してプログラムを作成する必要がある。そこで、GRADEシステムでは、以下の拡張を行った「漢字版Utilisp」を使用している[10]。

- (1) Lispのリーダが漢字コードを扱えるようにする。
- (2) 漢字ストリングを扱うための関数を作成する。
- (3) Lispのプリンタが漢字コードを扱えるようにする。
- (4) 関数ATOMLENGTH, STRINGPが漢字ストリングを扱えるようにする。

(2)の漢字ストリング処理用としては、以下の関数等が準備されている。

KSUBSTRING

漢字ストリングから指定の位置の部分ストリングを取り出す。

KSTRING-APPEND

漢字ストリングを結合する。この場合、不要なシフト・コードを取り除く。

KSTRING-SEARCH

漢字ストリング中で、指定した漢字ストリングがある位置を探す。

KSTRINGP

データが漢字ストリングかどうかを調べる。

この拡張により、シンボルに関しては漢字であることを意識せずに、また、ストリングに関しては、シフト・コードや2バイト・コードを意識せずにプログラムの作成が行えるようになった。

3.2. VSAMインターフェイス

2節で述べたように、機械翻訳システムでは、ルールや辞書等の大量のデータを扱う必要がある。Lisp言語の基本的な考え方は、「必要なデータ

はすべてセル空間上に展開しておく」というものであるが、GRADEシステムが扱う必要のあるデータの場合は、實際上、Utilispのセル空間上に展開しておくことは、不可能になった。多量のデータを扱うために、GRADEシステムでは、VSAMデータセット(直接アクセス型のファイル)に対する入出力を行うためのインターフェースを付加した「VSAM版Utilisp」を使用している[11,12]。

このインターフェースでは、VSAMデータセットのアクセス用のキーとして、エリア番号と呼ばれる0-255の数字とシンボル、ストリング、数などのキーを組にしたものを用いており、たとえば、GRADEシステムの辞書では、品詞をエリア番号に、見出し語をキーにして扱っている。以下に、VSAMアクセス用の関数の一部を示す。

STREAMDB

VSAM用ストリームを作成する。

INOPENDB, INOUTOPENDB, OUTOPENDB

VSAMストリームを入力／入出力／出力モードでオープンする。

CLOSEDB

VSAM用ストリームをクローズする。

PUTDB

キーによるレコードの追加・更新を行う。

GETDB

キーによるレコードの入力を行う。

KEYDB

キーによるレコードの存在の確認を行う。

READDB

順アクセスによるレコードの入力を行う。

PRINTDB

順アクセスによるレコードの追加を行う。

このように、直接アクセスだけではなく、順アクセスも可能になっており、図9に示したような、辞書内容の検索等に使用している。

3.3. ガーベジ・コレクションの方法の変更

システムが大きなものになるにつれ、必要なセル空間(UtilispではHEAPと呼ばれている)も大きなものになる。GRADEシステムを運用している京都大学大型計算機センタのTSSの利用者の主記憶領域の最大は4Mバイトであるので、オリジナルの

Utilispで使用可能なセル空間は、約1.5Mバイトと、Symbolics の Lisp Machine (200Mバイト程度は現実に可能)等と比較してあまり大きなものではない。これは、主記憶領域の問題だけではなく、図10のように、コピー方式のガーベジ・コレクション(GC)を行っているためでもある。GRADEシステムを開発する場合、セル空間を少しでも多くする必要があったので、GCのためのコピーを主記憶上で行うのではなく、図11に示すように、作業用のディスクを利用してコピーを行うように、変更を行った。

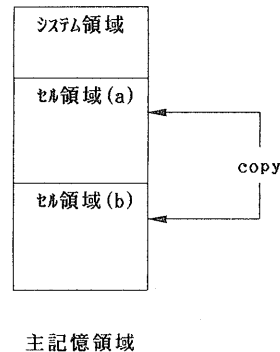


図10 Utilispのガーベジ・コレクション

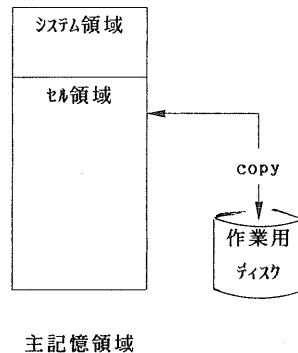


図11 ディスクを用いたガーベジ・コレクション

この変更により、オリジナルのUtilispの約2倍のセル空間(約3Mbyte)を使用することが可能になった。しかし、ディスク・アクセス回数が非常に増大するため、実行時間(特にレスポンス・タイム)が悪化するという欠点が生じた。

3.4. ユーティリティ (デバッガ)

GRADEシステムは、一般のAIシステムと同様、多人数 (延べ10人、ソフトウェア部分のみ) が、プログラムの追加・変更・改良を繰り返して作成されたものである。このため、プログラム相互の参照関係を明確に理解することが容易でなくなり、システムの開発・改良を行うための便利なデバッガが必要となった。ところが、Utilispには、デバッグ用のツールとしては、トレサ程度があるだけであり、そのトレサも、コンパイル・コードに対しては適用できないという問題があった。そこで、以下の機能を持ったデバッグ用のツール (XREF-TRACEと呼ぶ) を作成した。

(1) ソース・プログラムの検索・自動ローディング

プログラムを効率良く開発・改良していく上で、どのような関数がすでに定義されているか、また、その定義のソース・ファイルの名前が何であるか、を知ることは、重要なことである。ところが、大規模なシステムでは、定義される関数の個数やソース・ファイルの個数が多くなり、それぞれの関数の名前や機能、ソース・ファイル名を完全に把握しておくことは容易ではない。たとえば、GRADEシステムでは、3000個以上の関数が定義されており、また、ソース・ファイルは、100個以上もあるため、内容を把握することは困難になっている。

この問題に対処するため、XREF-TRACEには、図12(a)に示すように、関数名を文字列のパターン・マッチングにより検索したり、図12(b)に示すように、関数を含むソース・ファイルをロード

(EXFILE) する機能がある。これにより、必要な関数を見つけ出すことや、その具体的な定義を知ることが容易になり、システムの開発・改良の効率が良くなった。

(2) 関数の呼び出し系列スタック (関数BACKTRACEの値) を容易にたどる機能

エラーが発生した場合、関数の呼び出し系列を調べることは、デバッグの第一段階であろう。Utilispには、BACKTRACEという関数の呼び出し系列 (関数名等) を値とする関数があるが、この情報を直接見るだけでデバッグを行うことは容易ではなく、実際のデバッグでは、関数名だけでなく、関数の定義そのものを参照したい場合が多い。

XREF-TRACEには、図12(c)に示すように、関数の呼び出し系列を自由にたどる機能 (UPとDOWNコマンドによる) があり、その際、関数の定義が自動的にPretty Printされる (Utilispの関数PPを使用)。これにより、エラーが発生した状況を詳細に検討することができ、更に、値を調べたい変数の名前等も容易に知ることができるようになった。なお、呼び出し系列とは独立な関数をたどることも可能であり、たとえば、エラーが発生した状態の以降のプログラムの流れを調べることも容易である。また、(1)のソース・プログラムの自動ローディングの機能との組み合わせにより、通常はコンパイル・コードで実行している関数についても、簡単にその定義を知ることができ、デバッグを効率よく行うことができる。

(3) その他の機能

XREF-TRACEには、注目している関数 (図12でプロンプトで示されているもの) に対して、Pretty Printを行うだけでなく、マクロ展開結果を表示したり (図12(c))、内装エディタのUSEや、関数TRACE、UNTRACEを呼び出したりする機能もある。これにより、同じ関数名を何度もタイプすることが不要になり、説明的な長い関数名を使用することが心理的に抵抗なく行えるようになった。説明的な関数名は大規模なシステムを作成する場合に(1)の検索の機能と組み合わせることにより、非常に有効である。

4. Zetalispへの移植

Muプロジェクトでは、翻訳システムをUtilisp以外のLisp (Franz-lisp, Zetalisp) に移植を行っている。また、現在、Symbolics 3600 Lisp Machine上に、機械翻訳システムの文法および辞書の改良・保守のためのワーク・ステーションを研究・開発中である[13]。そこで、Utilispで記述されたGRADEシステムをLisp MachineのZetalispへ移植する必要が生じた。そこで、ZetalispのPackage機能[14] (シンボルの同定用のテーブルを複数個仕様できるようにした機能) を利用して、Lisp Machine上にUtilispで記述されたプログラムを少量の修正だけで実行できる環境を開発した (これを、Utilisp Compatibility Package、UCPと呼ぶ) [15]。

UtilispとZetalispは共にMACLISP系の考え方のLisp処理系であるので、基本的な部分に大きな差はなく、UtilispはZetalispのサブ・セットと考えてもよい。しかし、細部については、幾つかの差異があり、実際には、Utilispで記述されたプログラムは、そのままでは、ほとんどZetalisp上では実行できない。代表的な差としては、LAMBDA変数の記述の差と、MAP系の関数の引数の順序の差がある。たとえば、省略可能な引数を持った関数の定義は、Utilispでは、

```
(DEFUN FOO (X (Y T)) ... )
```

であるが、Zetalispでは、

```
(DEFUN FOO (X &optional (Y T)) ... )
```

であり、変更が必要である。また、MAPCARの引数は、Utilispでは、

```
(MAPCAR list function)
```

であるのに対して、Zetalispでは、

```
(MAPCAR function list)
```

であり、引数の順序が逆になっている。また、ある意味で正常でない引数が与えられた場合の動作も異なる場合がある。たとえば、関数ASSQに対して、

```
((A B) C (D E))
```

のように連想リストの形をしていないものを与えた場合、Utilispでは、シンボル「C」を無視するが、Zetalispでは、シンボルのCARを取るというエラーが起こる。

GRADEシステムは、プログラム量が多いこと（全システムで約3万行）、Utilispでのシステムの改良も継続的に行っており、何度も移植を行わなければならないこと、移植そのものが目的ではないこと、から、これらのUtilispからZetalispへの変更をエディタで直接行うことは現実的でない。そこで、ZetalispのPackage機能を用いて、等価でない関数をZetalisp自体に影響を与えることなしに、再定義し、移植を容易に行えるようにした。再定義の例を図13に示す。引数の順序が異なるもの、関数名が異なるだけのものは、マクロ定義を用いることにより、極力オーバーヘッドがでないようにした。

なお、入出力の扱い方が、UtilispとZetalispでは、かなり異なるため、ファイルのオープン部分は、プログラムを直接変更する必要があること、UCPでは、VSAMインターフェイスは作成していないことなどの問題がある。また、Zetalispでは、幾つかの特殊記号（「#」、「:」、「|」等）が特別な意味を持つこと、Utilispでは、ストリング中の「/」は、特

```
(defmacro UTI:MAPCAR (list func)
  (global:mapcar ,func ,list))
```

(a) 引数の順序の変更

```
(defmacro UTI:CONSP (x)
  (global:listp x))
```

(b) 関数名の変更

```
(defun UTI:ASSQ (item alist)
  (global:loop for x in alist
    when (and (listp x)
              (eq (car x) item))
    do (return x)))
```

(c) 例外処理の変更

```
(defun UTI:CR (x) x)
```

(d) 単純な定義の追加

図13 関数の再定義の例

(global:で始まるシンボルはZetalisp固有のもので、uti:で始まるシンボルはUtilisp用のものである。)

に意味を持たないが、Zetalispでは、エスケープ文字になっていること等、Lispのリーダに差がある部分もエディタ等で機械的に修正しておく必要がある。

5. おわりに — Utilispの対する希望

本報告では、Utilisp上で開発している機械翻訳システムの概要と、その開発の過程で必要になったUtilispの機能の強化について述べた。更に、Utilispで記述されたプログラムをLisp Machine上のZetalispに移植した方法についても述べた。

機械翻訳システムのような大規模なソフトウェア・システムを作成する上で最も問題となった点は、セル空間の不足であった。この点について、以下のようなUtilispの改良が期待される。

(1) セル空間の拡張

セル空間が数Mbyte程度では、小さすぎることは明らかである。富士通のMシリーズのAE機能（31ビット・アドレッシング機能）を利用して、数十Mbyteから数百Mbyteのセル空間を使用できるようにはならないか。

(2) メモリ管理機能の強化

文法規則や辞書データのように、ガーベッジにほとんどならないデータも多い。セル空間を幾つかに分割し、ガーベッジ・コレクションの対象とそうでないものとをLispプログラム・レベルで指定し、ガーベッジ・コレクションの効率を向上できないか。

(3) データ (S式) のコンパイル

2節で述べたように、GRADEシステムでは、文法規則の読み込み (read) を頻繁に行なっており、このために、実行時間の約1/3が使用されている。これが、ガーベッジ・コレクションの時間と共に、翻訳の処理速度を低下させる要因のひとつとなっている。そこで、データ (S式) 部分のコンパイルによるLispリーダの高速化が期待される。

また、柔軟なソフトウェア環境を開発する上では、文字単位の入出力や、画面制御機能の付加など、入出力機能の強化が望まれる。

これらの機能は、すでに、Lisp Machine上では実現されており、Zetalisp上でのGRADEシステムでは、文法規則と辞書データのすべてをセル空間上に展開し、翻訳の実行を行なっている。また、マルチウィンドやマウス等の高機能の入出力インターフェースによる柔軟なソフトウェア環境の作成も行なっている。しかし、処理速度や、使用可能なディスク容量や入出力速度の点では、超大型計算機上のUtilispの方が大幅にすぐれており、大規模なシステムを考えた場合、特にUtilispの改良が希望される。

なお、GRADEシステムの開発は国の科学技術振興調整費による「日英科学技術文献の速報システムに関する研究」の一部として行ったものである。

最後に、GRADEシステムを開発するにあたり、多くの協力や有効な助言をいただいたMuプロジェクトの参加者と、開発の前提となったUtilispの開発・拡張を行なわれた関係者のみなさまに感謝します。

参考文献

- [1]Chikayama, T.:Utilisp Manual, Technical Reports METR81-6, Tokyo Institute of Technology, 1981.
- [2]長尾真他:科学技術庁機械翻訳プロジェクトの概要,情報処理, Vol. 26, No. 10, 1985.
- [3]中村順一:文法記述用ソフトウェアGRADE,情報処理学会自然言語研究会資料38-4, 1983.
- [4]中村順一他:文法記述用言語GRADE,日本ソフトウェア科学会第1回大会論文集, pp.243-6, 1984.
- [5]中村順一:機械翻訳のソフトウェア環境,情報処理, Vol. 26, No. 10, 1985.
- [6]中村順一:知識情報処理の応用—機械翻訳への応用,コンピュータール, No. 10, コロナ社, 1985.
- [7]小木正三他:機械翻訳のためのソフトウェアGRADEの文法開発環境,情報処理学会第28回全国大会, p.1271-2, 1984.
- [8]中村順一他:Muプロジェクトにおける辞書の運用方式—日英変換辞書と英語辞書,情報処理学会自然言語シンポジウム予稿集, 1984.
- [9]片桐恭弘他:Muプロジェクトにおける翻訳実験支援環境,情報処理学会自然言語研究会資料47-9, 1985.
- [10]富士通ソーシアル・サイエンス・ラボラトリ:Utilisp漢字化改造使用書, 1983.
- [11]富士通ソーシアル・サイエンス・ラボラトリ:Utilisp VSAMアクセス機能利用手引書, 1984.
- [12]池田尚志他:キー順ファイルをベースとした解析/翻訳の実験環境,情報処理学会自然言語研究会資料50-4, 1985.
- [13]中村順一他:マルチウィンドを利用した機械翻訳のための文法開発ツール,情報処理学会第29回全国大会, pp.1225-6, 1984.
- [14]Symbolics, Inc.:PKG Package, Symbolics Lisp Machine Manual, 1984.
- [15]工業技術院計画課他:日英科学技術文献の速報システムに関する研究—GRADEシステム説明書, pp.209-248, 1984.