REWMOLのVAX/VMSへの移植と自然語処理のための機能拡張

石原　好宏*・刈谷　丈治**

REWMOL : Transfer of the Interpreter to VAX/VMS, and Some Augmentation of Its Facilities for Natural Language Processing

Yoshihiro ISHIHARA and Joji KARIYA

Abstract

REWMOL is a REal World MOdeling Language with two operation modes. The first is the basic mode, where the conventional LISP equivalent facility is available, and the second the process mode, where a pseudo parallel processing mechanism is available. The interpreter of REWMOL was first developed on ACOS-6 (an OS of a mainframe, ACOS-800/850), which was later shifted to UNIPLUS (an OS of a workstation, U-station).

Recently, we transferred the interpreter to VAX/VMS, and augmented some of its facilities by adding a number of basic REWMOL functions for natural language processing. Then, in order to check the validity of the transfer of the series, we not only transferred some application programs, originally written in UCI-LISP and dealing with conceptual data, but also wrote some object-oriented programs on the VAX system.

This paper reports on the outline of the transfer of the interpreter and the application programs, and also on the necessity, the functional taxonomy, and the specifications concerning the related augmentation of the functions.

1．まえがき

REWMOL[1]は，実世界の諸現象，例えば人間の思考過程，のモデルを記述するための言語で，二種類の実行モードを持つ．その一つはベーシックモードで，従来の記号処理言語LISPとほぼ同等の機能と記述形式を提供するが，時間の進行は管理しない．もう一つはプロセスモードで，並列処理すべきプロセスを極めて低次レベルの要素で構成し，時間進行の管理を極力抑えた実行制御を行う機能を提供する．併せて，効率よいプログラム開発に必要な幾つかの機能，例えば便利なデバッグ／トレースの機能，を提供する．

REWMOL処理系は，C言語で書いたインタープリタであり，最初メインフレームACOS－800（後にはACOS－850）のオペレーティングシステム（以下，OS）ACOS－6の上で開発したが，最近はワークステーションU－STATIONのOS，UNIPLUS，の上に移植し，保守と更新の作業を続けている[2]．

REWMOLは，こうした特徴と背景をもつので，人工知能向きのプログラミング言語として必要な条件をかなりの程度備えていると言える．従って，その処理系を人工知能の分野で国際的に普及したスーパーミニコンピュータVAX－11のOS，VAX/VMS，の上でも

*山口大学工業短期大学部情報処理工学科
**山口大学情報処理センター

使えるようにすれば，その有用性は増し，REWMOLで書いた応用プログラムの流通性も向上するであろう．

こうした見地から我々は，REWMOLの処理系をVAX－11のOS，VAX/VMS，の上に移植した．続いて，移植の結果を確認するため，UCI－LISPで書かれた概念依存構造を扱う応用プログラム[3]の幾つかをREWMOLで書き直し，動作させてみた．その過程では，主として英語文の文字／文字列処理に焦点を絞り，内部処理および入出力処理に便利な関数の大幅な増強も試みた[4]．

本稿では，最初に，REWMOLの処理系と応用プログラムの移植の概要を述べる．続いて，移植の各段階で増強した関数について，その必要性を述べ，増強した関数を機能別に分類し，関数機能の概略を述べる．最後に，VAX/VMS版REWMOLの特徴を整理する．

2．移植の概要

この章では，これまで試みた移植について，その基本方針と作業の概要を，REWMOLの処理系と応用プログラムに分けて述べる．

2.1　REWMOL処理系の場合

REWMOLの処理系は，原版を更新するたびにこれまで三度，VAX/VMS（以下，VAXと略称）の上へ移植した．最初の作業（昭和60年3月）は，結果的に，Cで書いたプログラムの移植性（Portability）を高めるのに必要な更新・修正の要点を知るために，開発用マシーン及び移植先マシーンのハードウェアとOSを詳細に比較検討する機会となった．この経験は，その後の，処理系原版の更新でも十分に役立っている．

さて，最初の移植のとき，開発用マシーンのハードウェアはACOS－850，OSはACOS－6であった．又，REWMOL処理系の原版（ACOS版REWMOL）は，バージョンがV3.5で，ソースプログラムは4,000行を越えていた．

そこで，ソースプログラムは，保守管理の便宜，プログラムの論理構造・機能などを考慮し，約150個のファイルに分散格納した．この方針はVAXへの移植後も順守し，原則としてACOS上とまったく同一のファイル内容および構造を保つこととした．

なお，REWMOLの処理系の原版は，この時点で，ソースプログラムの記述言語がACOS－Cであり，大文字で表記されていた．一方，VAX－11 Cは，小文字表記である．従って最初の移植では，ACOS上で語彙解析プログラムジェネレータLEXによる文字変換だけを行い，直ちに電話回線とミニフロッピディスクを使ってVAXへファイル転送し，以後の作業はすべてVAX上で行った．

その手始めは，両システムのハードウェア・ソフトウェアの違いに依存する部分の修正と関数定義であった．これを必要とした理由は：

1) 1ロングワードが，ACOSでは36ビット，VAXでは32ビット，であること；
2) C言語の若干の関数で，パラメータの書き方が処理系によって異なること；
3) ACOS－Cの標準関数であっても，VAX－11 Cでサポートしないものがあること；

などであった．

次に，VAX－11 Cコンパイラ，リンカ，シンボリックデバッガなどを用いたデバッグを行った．その過程で，ACOS－Cの処理系ならば受理するのにVAX－11 Cの処理系では受理しない箇所が若干出てきた．そこで，ハードウェアとOSに依存する部分は，C言語プリプロセッサの条件コンパイル機能を使い，必要な部分だけ自動的に選択されコンパイルされるよう，修正した．但し，実行状態の持越機能の部分だけは懸案として残し，作業をいったん終えた．

ところで，この時点のREWMOL処理系原版（V3.5）にはバグがなお若干残っており，必ずしも期待どおりには動作しなかった．しかし，この経験は貴重で，その後，移植性の高いソースプログラムを書くのに十分役立っている．これは次の事実からも明らかである．

つまり，二度目（昭和61年3月）と三度目（昭和61年9月）の移植では，開発用マシーン，そのOS，記述言語Cの処理系など，開発環境を大幅に変更した．そのため，新しいCの処理系（UNIPLUS－C）に含まれるがVAX－11 Cには含まれない関数が時折出てきて，書き換えに多少手間どった．それにもかかわらず，作業は極めて順調に終了できた，という事実である．

2.2　応用プログラムの場合

二度目の作業でVAX上にREWMOL処理系（V3.6）を移植し終えると，その動作を確認すべく，他のLISP方言で書かれた応用プログラムの幾つかを移植し，動作実験を試みた．その題材としては，R. C. Schankらが書いた，自然言語／概念依存構造を扱うプログラム（原版の記述言語はUCI－LISP）[3]のマイクロ版合計5本*を選んだ．なぜならば，筆者がREWMOLで書いてみたい自然語処理プログラムにも共通するプログ

*SAM，PAM，TALE－SPIN，POLITICS，ELIの各マイクロ版

ラミング技法を含んでおり，長さも妥当（一本当たり250～750行）であると考えたからである．

移植に当たり，UCI－LISPのソースコードは極力原文を保存し，それとREWMOLとの隔たりは，別に定義する関数で埋める，という基本方針を採った．この方が作業は簡単で，しかも移植後のプログラムが読み易く，保守管理にも便利だ，と判断したからである．

この立場からの移植は，UCI－LISPで書かれたMicro ELIをACOS－LISP処理系の上に移植した[5]ときにも一度経験した．そのときは，概念依存構造を扱うプログラムを書く上で重要だとR, C, Schankらが指摘した，54個のUCI－LISP関数[3]に特に注目し，それらがACOS－LISP処理系の上でも元の仕様どおりに機能するよう，25個に上る関数を新しく定義した．

しかるに，このたびのVAXへの移植では，まず，上に述べた機能範囲で，REWMOLの標準関数に組入れたものを除き，15個の関数の再定義や修正が必要であった．

更に，ACOSへは移植しなかった出力関数MSGの定義を含め，入出力および文字／文字列処理に焦点を絞って，応用プログラムのユーザインタフェースを一層人間の側になじみ易く，且つ使い易くするため，関数の大幅な追加・機能拡張を試みた．この範囲では，40個に近い関数を新しく定義した．

3．VAX/VMS版REWMOLの機能拡張

本章では，自然語文（主に英語文）の頻繁な入出力を伴う処理を扱う応用プログラムを書き易くするため，VAXへ移植した後で試みたREWMOLの機能拡張について，その必要性と，拡張の概要を述べる．

3.1　必要性

多くの応用プログラムでは，マンマシーンインタフェースの媒体として自然語を用いるのが，人間にとってもっともなじみ易い．その場合，入（出）力は，従来のOA機器，タイプライターなど，へ（から）の入（出）力で見られる伝統的な書式に倣うのが，もっとも自然であろう．しかし，そうした書式による入出力を支援する関数が，LISPの標準数関数として準備されることは少ない．REWMOL処理系の場合もその例に漏れない．従って，そのための支援関数はユーザ側で準備せざるを得ない．

文字／文字列処理に関しても，基本的なものを除けば，便利な機能がそれほど豊富に標準関数として提供されることはない．しかし，それは文字／文字列処理の機能で一般性の高い関数となりうるものが少ないからではない．それらを処理系の標準関数に含めるとシステムが重くなり過ぎるので避けたにすぎない．

こうした事情から，個々の応用プログラムを構築するたびに定義するには余りに煩雑で，且つ一般性の高い文字／文字列処理の機能を，ユーザ側でよく整理し，ユーティリティ関数として定義しておくことは，むしろ望ましく，必要でもある．このたびの機能拡張は，こうした見地からの試みの一つである．

3.2　増強した関数の作成形態

増強した関数群は，作成の形態で分類すると，次の5つのタイプがある．

タイプ1）　REWMOLの基本関数*の新しいメンバーとして，REWMOL処理系の本体に直接組込んだもの．

タイプ2）　UCI－LISPの標準関数のうち，ACOS－LISP上で動かすためその関数定義機能を使って定義したもの[5]を，REWMOL上でも同じ仕様で動くように，REWMOLの関数定義機能で定義し直したもの．

タイプ3）　ACOS－LISPへの移植のときは取上げなかったUCI－LISP標準関数のうち有用なものを，REWMOLの関数定義機能を使って定義し直したもの．

タイプ4）　REWMOLの機能を増強するために，その関数定義機能を使って新しく定義したもの．

タイプ5）　既存のREWMOLの基本関数あるいはブートファイル関数**の機能を増強したもの．

こうした関数は，総数で既に50個を越えた．次節以降で，このたび増強した関数をタイプ別に整理する．特にタイプ4と5については，関数機能の仕様の概略も示す．なお，各関数を定義する関数（個別に括弧内に記す），及びそれらが定義する関数の関数タイプを下に列記しておく***．

DE	：評価型開関数，
DE *	：環境がNILの評価型閉関数，
DF	：非評価型開関数，
DF *	：環境がNILの非評価型閉関数，

*Cで定義して処理系の本体に組込んだ，REWMOLの標準関数で，システム標準関数[2]ともいう．

** REWMOL自体で定義した，REWMOLの標準関数．

***詳細については［刈谷］[2]を参照のこと．

DM　　：マクロ関数，
IS__SAME__TO：別名定義関数，

3.3 関数の機能別分類

この節では，VAX/VMS 版 REWMOL で新しく増強した関数を機能別に分類する．

(1) タイプ1の関数

当面，二つの関数 DATE と DAYTIME を処理系に組込んだ．DATE は，これを呼出したときの日付（年月日）を，DAYTIME は，そのときの日付と時刻（時分秒1/100秒）を，VAX システムから読み取って，関数値とする．

近々，このほかに，日本語文字（漢字，かな；2バイト系）と ANK 文字（1バイト系）が混在するデータも扱えるよう，幾つかの関数を追加する予定である．

(2) タイプ2の関数

REWMOL 処理系を VAX 上へ移動するに当たり，タイプ2の範疇に属する関数で，新しく定義する必要があったのは15個で，内訳は次のとおりである．

2－a）実行制御関数
for* (DM)，loop (DM)，let (DM)，progl (DF)，progn (DF)，selectq (DF)

2－b）属性値管理関数
defprop (DF)

2－c）リスト処理関数
cons－end (DE)，ncons (DE)，some (DE)，multiappend (DE)

2－d）列関数
mapc (IS__SAME__TO)，mapcan (DE)，mapcar (IS__SAME__TO)

2－e）関数定義関数
undefine (DF)

(3) タイプ3の関数

応用プログラムを REWMOL 版に書き改めるに当たり，タイプ3の汎用関数で，新しく移植する必要があったのは5個である．内訳は次のとおりである．

3－a）値管理関数
GET* (IS__SAME__TO)

3－b）リスト処理関数
MATCH (DE)，SUBST (DE)

3－c）入出力関数
MSG (DF)，sprint (DE)

(4) タイプ4の関数

このタイプの関数は，今回の機能拡張の中心であり，現在24個ある．内訳は次のとおりである．

4－a）実行制御関数
lambda (DM)

4－b）値管理関数
set (DE)

4－c）リスト処理関数
nconc (DE)，remove (DE)

4－d）文字処理関数
LWRCASE__P (DE)，UPRCASE__P (DE)，LASTCHAR (DE)，PART-OF-SPEECH__P (DE)，G__UPRTOLWR (DE)，S__UPRTOLWR (DE)，T__UPRTOLWR (DE)，LWRTOUPR (DE)

4－e）入出力関数
READ-SENT (DE)，READ－TEXT (DE)，BELL (DE)，CURCOL (DE)，MSG__CCG (DF)，PPFN (DF)，PRIN－SENT (DE)，prin－sent (DE)，PPF (DF)，PRIN－TEXT (DE)

4－f）列関数
member (DE)

4－g）シンボル管理関数
FUNCDEP (DF)

(5) タイプ5の関数

このタイプの関数は，REWMOL 処理系のブートファイルの中で定義済みである．しかし，このたびの移植に伴う機能拡張のため，一部の関数（当面，7個）について，定義内容の一部を修正または変更した．内訳は次のとおりである．

5－a）リスト処理関数
equal (DE)

5－b）出力関数
fputc (DE)，fputx (DE)，PP (DE)，pp (DE)，PUTNL (DE)，PUTXL (DE)

以上のほかにも，概念依存構造のデータを扱うための関数[3]を10個ばかり移植した．但し，それらについては言及しない．

3.4 関数機能の設計

この節では，今回の増強の主眼であるタイプ4と5の関数のうち，特に文字／文字列処理と入出力の関数

---

* REWMOL では，大文字と小文字を完全に区別する．原則として，小文字名の関数は，既存の関数の何らかの拡張になっている．一方，大文字名の関数は，UCI－LISP など，別の LISP 方言にある同名の関数と等価な機能か，あるいは，まったく新しい機能，を導入する．

について，導入した動機，設計の基本方針，及び関数機能の仕様の概略をまとめる．

(1) 動機，設計の基本方針

今回の増強では特に文字／文字列処理と入出力の機能に焦点を絞ったが，これは脈絡のない複数の動機によったのでは決してない．UCI－LISPで書かれた応用プログラムをREWMOLで書き直すとき，システムと人間とのインタフェースを，従来のタイプライターにおける自然語文入出力の様式に合わせたい，という一つの動機からであった．このたび移植した応用プログラム原版そのままでは，出力がすべて大文字で表記され，読みづらかったのである．

そこで，応用プログラムにおける自然語文の入出力の様式を，次のように改めることにした．

(イ) 自然語文(単語列)の入出力では，LISP特有のS式は使わず，単語間を空白で区切るだけのベタ書き方式を採用する．

(ロ) 入力の場合，入力操作およびその後の処理の便宜を考え，通常の方式（大小文字混在）でなく，原則として大文字だけを使う．

(ハ) 出力の場合，その結果を単語列として見たとき，文頭語の先頭文字だけは大文字，そのほかはすべて小文字による表記，を原則とする．

(ニ)但し，その場合，たとえ単語が文中にあっても，それが固有名詞または代名詞“I”であれば，その先頭文字も大文字で表記する．

(ホ) (ハ)，(ニ)の原則にかかわらず，出力の場合で，特に指定があれば，原表記を保存する．

(2) 機能設計の基本事項

まず二つの術語を定義する．本稿で，“文”という場合，文頭語から主文の句点までの単語列を指す．又，“文章”とは，そうした文の連鎖であって，“テキスト(text)”や“ディスコース(discourse)”が指す概念と同じである．

ここで，前項(1)で述べた動機と設計方針を実現するための基本事項を整理しておく．

(イ) 自然語文の内部表現

いま，自然語文jの，第i番目の単語をW_{ij}，その文種をT_jとしたとき，自然語文の内部表現のデータ構造として，次のようなリスト構造を採用する．なお，S式中の要素の下線付き添字$_{nj}$は，文jの構成単語数を表す．

① n_j個の単語からなる文の場合；

$$(W_{1j}W_{2j}\cdots\cdots W_{\underline{nj}j})$$

② m個の文からなる文章の場合；

$$(((W_{11}W_{21}\cdots\cdots W_{\underline{n1}1})\ T_1)$$
$$((W_{12}W_{22}\cdots\cdots W_{\underline{n2}2})\ T_2)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$((W_{1m}W_{2m}\cdots\cdots W_{\underline{nm}m})\ T_m))$$

(ロ) 文中における単語位置の管理

自然語文を生成する際に，各単語の表記文字の大小まで管理したければ，出力する各単語の位置情報（文頭語か，否か）は少なくとも管理しておく必要がある．

ここではトップレベルのローカル変数をフラッグとして用い，出力関数MSG__CCGの中でその値を更新することで管理する．

(ハ) 出力単語の品詞の識別

英語文の表記では，通常，単語が固有名詞か代名詞‘I’ならば，たとえそれが文中にある場合でも，その先頭文字は大文字で表す．従って，出力文の表記文字の大小をうまく管理するには，(ロ)に述べた位置管理のほかに，出力する各単語の品詞も認識できる必要がある．

ここでは，次の方法で実現する．つまり，一つのトップレベルのローカル変数が保持する2段の属性リストで，枠構造として単語辞書を定義し，品詞情報を蓄える．これを出力関数MSG__CCGの中で，文字種変換関数G__UPRTOLWRが文字判定関数PART—OF—SPEECH__Pを使って参照し，単語品詞を認識する．

(ニ) 文種情報の取扱い

文の種類（平叙文，疑問文，感嘆文，命命文など）は，通常，句点(英語文の場合，'.', '?', '!'など）で表現する．しかし，これをデータ入力時に、端末から直接入力しようとしても，必ずしもうまくいかない．なぜならば，REWMOL処理系で入力時マクロとして使われることの多いこうした特殊記号は，その扱い方を変更できるものの，切換えのタイミングがやや難しいからである．

そこで，文種情報は，HEIJO，GIMON，KANTANといった文字列で入力し，それをシステム内部で必要に応じて参照する．

(ホ) ディスプレイ端末のカラム制御

ディスプレイ端末の画面を読み易く作るには，出力時のカラム制御をユーザ側にある程度任せる必要がある．しかし，REWMOL処理系の原版には，そのための関数がない．

そこで，VAX/VMS版では，一つのトップレベルのローカル変数をその制御用に当て，次に出力すべきディスプレイ画面上の文字位置を常に保持させる．そして，出力関数が端末画面へ文字を出

りするときは，必ずそれを参照し，先頭の文字位置を決めることにする．

(3) 関数機能の仕様

ここでは，本節の(1)で述べた基本方針，及び(2)で述べた基本事項に基づいて，定義済みの標準関数*の一部を変更し，あるいは新しく定義して，実現した関数群の仕様を概説する．記述は，3.3で試みた機能別分類の関数タイプごとに行う．

(イ) タイプ5の出力関数

REWMOLのシステム標準関数（基本関数）のうち基本的な出力関数は，FPUTCとFPUTXだけである．しかし，どちらもカラム制御機能を持たない．そこで，新しいブートファイル関数fputcとfputxをそれぞれに対して定義し，出力文字(列)の字数と字種に基づくカラム制御を行う．

一方，既存のブートファイル関数に対しては次のように変更を行う．まず，PUTNLでカラム制御変数を1に初期化する機能，PUTXLで出力文字（列）に応じたカラム制御を行う機能，を追加する．又，プリティプリント関数ppで文の書き出し位置が指定できるよう拡張し，更に，PPの機能から冒頭の復帰改行を省いた関数として，ppを定義する．

(ロ) タイプ4の文字処理関数

いずれも文字（列）出力の前処理に使う新しいブートファイル関数である．LWRCASE__PとUPRCASE__Pは文字の大小判定用，PART-OF-SPEECH__Pは語の品詞判定用，G__UPRTOLWR，S__UPRTOLWR，T__UPRTOLWR，及びLWRTOUPR文字の大小制御用，LASTCHARは文字列の末尾文字をアスキー符号で返す関数，としてそれぞれ定義する．

(ハ) タイプ4の入出力関数

まず，3.4の(1)と(2)に示した方針に従い，文を入力する関数READ-SENTと，文章を入力する関数READ-TEXTを新しく定義する．これは，入力促進の案内メッセージを随所で出す．更に，エラー入力を回避するためのチェック機能も持つ．

次に，上記の二つに対応する出力関数として，PRIN-SENT，prin-sent，及びPRIN-TEXTを定義する．但しprin-sentは，PRIN-SENTの機能から文種情報に基づく句詞の表記機能だけを省いている．

このほか，出力関数MSG__CCGも定義する．

この基本機能は，UCI-LISPの標数関数MSG[3),6)]に等しい．しかし，更に，次のような新しい機能も含む．つまり，自然語文の生成システムの任意の場所から単語を出力したいとき，この関数であれば，単語の位置と品詞を手がかりとして，表記文字の字種（大小）をうまく制御する．

更に，ブートファイル関数の定義体をプリティプリント形式で出力する関数としてPPFとPPFNを定義する．関数定義体に加えて，前者ではその関数タイプを，後者ではその関数名を，出力する．

4．VAX/VMS版REWMOLの動作環境と実装

本章では，VAX/VMS版REWMOLの動作環境と，その上にプログラミング言語としてREWMOLを実装する方法を述べる．

このたびの移植先マシーンVAX－11/730は，主記憶がわずか1 MBにすぎない．従って，マルチユーザ方式で運用する場合，1ユーザ当たりのワーキングセットサイズの最大許容値は，500ブロック(250KB)程度とするのが限度である．この大きさは，REWMOLで書いた応用プログラムが走る環境としては余りに狭い．しかし，メモリ管理が仮想記憶方式であるため，論理的な動作を確認することを主目的とするのであれば，ほとんど支障は起こらない．

ついでながら，近く導入する予定のスーパーマイクロコンピューターMicroVAX IIの場合，主記憶が9 MB，OSがMicroVMSであり，そのOSはVAX/VMSと本質的に同じである．つまり，MicroVMSにはPDP－11との互換モードがなく，VAX/VMSの特権プログラムをMicroVMS上で実行するとき修正を要するという二点を除けば，両者には完全な互換性がある．このため，REWMOLをVAX/VMSの上からMicroVMSの上へ移植することは極めて容易であり，その実行環境としても当分，十分である．

VAX/VMS版REWMOLの処理系は，ソースプログラムのコンパイルとリンクを終えた時点で見ると，大きく分けて次の4つの部分からなる．括弧内に各部分を納めたファイル名を示す．なお，….rwは，REWMOL自身で書かれたファイルである．

(1) REWMOL処理系の本体（rewmol.exe）
(2) システム標準のブートファイル群（….rw）
(3) エラーメッセージの出力手続き（jdlerr.exe）
(4) ユーザ作成のブートファイル群（….rw）

REWMOLの立上げは，他のプログラム言語と同様に，VAX/VMSのDCLコマンドの一つとして実行で

*REWMOLの基本関数とブートファイル関数の総称．なお，P.95の脚注も参照のこと．

きると便利である．そこで，ユーザファイルディレクトリにあるコマンドファイル login. com の中で，文字列 rewmol を．

rew * mol：＝＝$ dqa0：[XXX]rewmol. exe

と定義する．これで，コマンド名 rewmol による立上げが可能となる（下線部のみ必須）．なお，XXX は，ファイル rewmol. exe を格納するサブファイルディレクトリの名前である．

DCL コマンド rewmol で処理系本体を立上げた後，引き続いて，(2)を立上げる．これは全部で16個のファイルから成り，立上げに順序制限がある．従って，コマンドファイルを一つ作り，そこから一括して立上げるのがよい．一方，(3)は，エラー時に(1)から適宜呼ばれるので，エラーメッセージファイル errmsg. dat を準備しておくだけでよい．

以上で REWMOL 処理系の標準部分の立上げは完了する．なお，REWMOL 処理系V4.0が占有する記憶領域は，この段階のシステム状態を関数 SYSSAVE でディスクファイルに退避させたときの大きさで，約330KBである．又，V3.6の移植で懸案として残した実行状態の持越機能は，V4.0の段階で実現している（詳細は省略）．

本稿で導入した関数群のほとんどは(4)に格納してある．しかし，それらがもたらす機能も REWMOL の基本機能の一部である，と我々は考える．よって，(4)のファイル群（当面，2個）も，システムのブートファイル群に続けて，同じコマンドファイルの中から立上げる．

こうした準備の下に，我々は，このたび移植した概念構造データを扱う応用プログラムの動作実験を試みた．その結果，処理系の使い勝手は極めてよく，拡張した機能もうまく作動していることを確認した．

5．REWMOL の特徴

REWMOL は，データタイプとして，多くの LISP 処理系に共通の，リスト，アトム（シンボル，生成アトム，整数値アトム，実数値アトム，文字列，関数アトム）のほか，アトム（プロセス，オブジェクト）も準備している．これらの管理については，それぞれのデータタイプの特性に応じた，適切な方法を採っている．

例えば，プロレス（時間と共に変化していく個々の対象）について言えば，その属性リストの管理をユーザに任せ，自由に利用できるようにしたので，並列処理に関して，種々の実験を極めてたやすく行うことができる．

このほか REWMOL は，言語構造がよく階層化され透明性も高い．その上にこのたび自然語処理に便利な機能を幾つか組み込んだことで，実世界記述用言語として一層使い易くなったと言える．

6．むすび

REWMOL の VAX/VMS への移植，他の LISP 方言で書かれた応用プログラムの移植，それに伴う REWMOL の，特に文字／文字列処理と入出力に焦点を絞った機能拡張，などについて概要を述べた．

この段階の VAX/VMS 版 REWMOL で，ANK 文字（1バイト系）だけからなる自然語データならば，既にほぼ満足に取扱える．又，漢字（2バイト系）についても，それだけから成るデータならば，一応は取扱える．しかし，通常の自然語処理の場面では，漢字データだけより，ANK 文字と漢字が混在する自然語データを扱う方がはるかに一般的である．しかるに，現在の処理系ではそうした混在データまでは考えていない．従って，そうしたデータを自在に取扱うためには，混在する文字の識別，計数，連結，切り放しなどの機能を導入することが必要で，今後の課題である．

［謝辞］本研究の経費の一部として昭和59年度文部省特定研究経費の交付を受けた．記して謝意を表する．

参考文献

［1］刈谷丈治，石原好宏："実世界記述言語 REWMOL の開発"，信学論(D)，J68－D，2，114－121（Feb.1985）．．

［2］刈谷丈治："REWMOL（V4.0）取扱説明書"，山口大学情報処理センター，p.39(Nov.1986)．

［3］Schank, Roger C. and Riesbeck, Christopher K.(Eds.)："Inside Computer Understanding：Five Programs Plus Miniatures", LEA, Hillsdale, N. J. (1981).

［4］石原好宏："VAX/VMS 版 REWMOL（V4.0）拡張関数取扱説明書"，山口大学工業短期大学部情報処理工学科，p.26（Nov.1986)．

［5］石原好宏："ACOS－LISP の機能拡張と Micro－ELI の移植"，山口大学工学部研究報告，34，1，113－123（Oct.1983)．

［6］Meehan, J. R. ："The New UCI LISP Manual", LEA, Hillsdale, N. J. (1979).

（昭和61年10月7日受理）