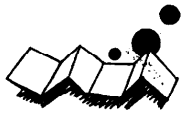
解　説

Prologの言語機能詳説†



後　藤　滋　樹††

1. まえがき

Prologを用いて，実際にプログラムを書く際には，幾つかの事柄を心得ていなければならない．ここではそれらの事柄を大きく3つに分けて解説する．最初の話題はプログラムの制御 (control) に関する部分である．また2番目は，Prologの処理系に最初から組込まれている(built-in) 述語や関数のレパートリの話である．3番目はプログラムで取扱うデータのタイプに関する話題である．

最初の「制御」というのは，例えばFORTRANで言えば，GO TO, IF, DOの類に相当するものである．PrologとFORTRANとでは制御の仕組みが大幅に異なっており，Prologで制御と言えばカット (cut) 演算子が話題の中心である（本稿の**2**章）．

2番目の「組込み述語・関数」は，FORTRANで言えばINT, REALなどの組込み関数や，SIN, COSなどの基本外部関数に相当するもので，Prologにおいても，ほぼ同様の働きを持つ（本稿の**3**章）．

3番目のデータのタイプのところでは，リスト構造と文字列とを取扱う．リストはLispによって有用性が広く認識されている．また文字列は特に我が国で活発に研究されている（**4, 5**章）．

本稿の最後の部分では，以上の各論とは趣を異にし，処理系による表記法や機能の差異について簡単に眺めてみる．

2. バックトラックとカット

Prologの言語仕様の中で，恐らく一番評判が悪いのは，カット演算子であろう．「論理的な解釈になじまない」，「動作が判りにくく，初心者がつまずき易い」等々の批判をよく耳にする．

しかし，カット演算子は現在のPrologの動作方法（＝バックトラックを利用する）と表裏一体を成しているので，そう簡単に除去できるものではない．本節ではバックトラックを利用したPrologの動作を解説した後に，カットの使い方を述べる．

2.1 バックトラック

例題を用いて，Prologのプログラムの動作を説明しよう．次の問題を考える．

問題：次の合金の中で，銅とスズとを両方とも含むものはどれか？

銅(copper)を含む合金	スズ (tin) を含む合金
真鍮 (brass)	はんだ (solder)
ジュラルミン (duralumin)	砲金 (gunmetal)
砲金 (gunmetal)	ホワイトメタル (white metal)
洋銀 (German silver)	易融合金 (fusible alloy)

Prologの立場を離れて，一般的に考察すると，この問題を解くのには2つの方法があり得る．

[並列に解く方法]

ここに並列とは，解の候補を集合として複数個保持して処理を進める意味である．具体的に例示すると，まず，第1の条件（銅を含む）を満たす集合が解の候補となる．

{真鍮，ジュラルミン，砲金，洋銀}

次に第2の条件（スズを含む）が課せられるから，2つの集合の共通部分 (intersection, ∩で表わす) をとれば求める答になる．

{真鍮，ジュラルミン，砲金，洋銀} ∩
{はんだ，砲金，ホワイトメタル，易融合金}
＝{砲金}

今の場合には，たまたま最終的な答が1個であったが，複数個残る場合も同様である．また条件を満たす答が存在しない時には，答が空集合 (empty set) になる．

[直列に解く方法]

ここに直列とは，解の候補を常に1つだけ持ち，その候補についての処理を出来るだけ先に進める方法で

† Prolog Programming by Shigeki GOTO (Musashino Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社武蔵野電気通信研究所

ある.

Prolog の動作はまさにこの方法に基づいているので，以下に Prolog の内部の動作を追跡する形で解を求めてみよう.

まず与えられた条件を Prolog の表現*に書き直す.

```
copper (brass).
copper (duralumin).
copper (gunmetal).
copper (german_silver).
tin (solder).
tin (gunmetal).
tin (white_metal).
tin (fusible_alloy).
```

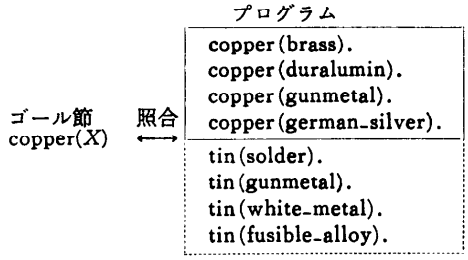
これは Prolog のプログラムの形を成す．また，求めるべき解の満たすべき条件は Prolog のゴール節の形をとる.

?—copper(X), tin(X).　　(1)

ここに，大文字のXは変数を表わす．ゴール節は2つの条件，①copper(X):「X は copper を含む」，及び②tin(X):「X は tin を含む」，が同時に成り立つようなXを求めることを要求している.

さて，Prolog は与えられたゴール節(1)をプログラムと照合する．この時，2つ以上の条件があれば左から順番に処理する．今の場合で言えば，①copper(X)，②tin(X) の2つの条件のうち，まず①から処理する.

そこで，話は次のように限定される．つまり，ゴール節中の copper(X) と，上述のプログラムとを照合すれば良い.



copper(X) と照合するプログラムは，copper(…) の形をしたものだけに限って良い．tin(…) の形のものは，もし照合しても copper(X) の答にはなり得ないからである.

照合の結果，変数Xの値として，次の4つの候補が考えられる．①brass，②duralumin，③gunmetal，④german_silver．直列的な処理では候補を1つに絞らざるを得ないから，とりあえず①を候補として採用して先に進む．(★：この部分を後で参照する.)

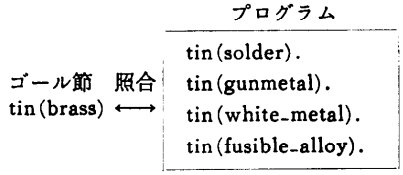
すると，ゴール節のXとして X=brass と代入したことに相当するから，ゴール節(1)は下の(1)′となる.

?—copper(brass), tin(brass).　　(1)′

しかも，copper(brass) はプログラムの中に存在するから，これは解決済であり，次は2番目の条件を満たすか否かをチェックする番である.

?—tin(brass).　　(2)

今度は，プログラム中の tin(…) の形の部分とゴール節(2)とが照合される.



今回は，ゴール節の中に変数が含まれないから，照合はゴール節とプログラムとの一致を検出する．ところが，上の照合では tin(brass) と一致するものはプログラムの中に無い．言い換えると，X=brass ではゴール節の②番目の条件「スズ (tin) を含む」が満たされない．このように，答の候補が条件を満たすのに失敗した時には，処理を逆戻りして候補を選び直せば良い．この処理（逆戻り・選び直し）をバックトラック (backtracking) と言う.

バックトラックの処理を具体的に見ると次の通り．上述の処理の途中（上の★印の個所）でXには4つの候補があり得た．とりあえず① X=brass を試したところ失敗した.

(失敗)　①X=brass
(次候補)　②X=duralumin
　　③X=gunmetal
　　④X=german_silver

そこで，以前には「次候補」であった②duralumin が繰り上がり当選して，Xの新しい値となる.

以下，詳しい動作の追跡は割愛して，主要な結果のみを述べる．②X=duralumin であるからゴール節は ?—tin(duralumin). となるが，これも照合の結果失敗する．したがって，再びバックトラックする．今度は，③X=gunmetal が選ばれて，?—tin(gunmetal). は無事成功する.

結局，求める答は X=gunmetal(砲金) と分った.

* ここでは DEC-10 Prolog[1] の表記法に従う．単語の先頭が大文字になると変数として扱われてしまうため，German を german と書いた.

答を得るまでの道筋を図示すると図-1のようになる．この図にも記入してあるように，バックトラックが2度起っている．ただし，これはあくまでも内部の処理を図解したものであって，実際にプログラムを動作させた時には，瞬時に X=gunmetal という答が出力される（図-2）．

2.2 バックトラックの積極的活用

もしもPrologにバックトラックの機能が無いとすれば，最初に選んだ候補（上の例では X=brass）が途中で行詰まると，そこで処理が停止してしまい，解を得ることはできない．したがって，バックトラック（逆戻り・選び直し）は直列的な解法においては必須のものであると言える．

さらに，直列的な解法で複数個の解を求める場合にもバックトラックが活躍する．図-3を眺めてみよう．この問題は先の問題を少し変えてあり，複数個（2個）の解を持つようにしてある．具体的な解は並列的に解いてみれば分る．ところで，一般に直列的な処理では答が1つでも求まれば，それで処理が終る．同様に，通常のPrologの処理でも，答を1つ出力すれば yes（=成功）であるとして，次の入力待ちとなる（図-3の中央部）．

この時，解が求まった（=成功した）にもかかわらず，あたかも失敗したかのように強制的にバックトラックを起こせば，候補が取替えられて次候補が繰り上がる．したがって，2番めの解を得ることができる．3個以上の解を持つ時も同様である．

これが，バックトラックの積極的な利用法である．実際のPrologでは，図-3の下半部のように最初の答の後に；（セミコロン）を入力すると次の答が出力される．

2.3 カットによるバックトラックの抑制

以上の説明では，Prologにとってバックトラックがいかに必須の機能であるかを強調した．しかし，時にはバックトラックの働きを抑制したい場合もある．この役割を果すのがカット演算子である．

まず，カットの作用を説明しよう．カットは！（感嘆符）と書かれる*．プログラムの中にカットが含まれていると，バックトラックが起きても候補の選び直しができない．つまり，複数候補があったとしても1番目の候補以外は選ばれることがない．換言すれば，カットとは，2番目以降の候補に仮想的に「失敗」のラベルを貼ることと等価である．

* Micro-Prolog[2]では，カットは /（斜線）で表わされる．

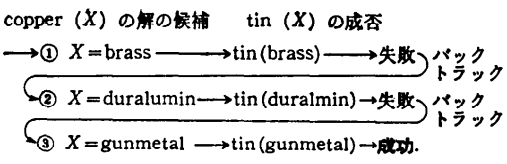


図-1 直列的な解法

```
| copper(brass).
| copper(duralumin).
| copper(gunmetal).
| copper(german_silver).

| tin(solder).
| tin(gunmetal).
| tin(white_metal).
| tin(fusible_alloy).

| ?- copper(X),tin(X).

X = gunmetal

yes
```

左端に｜（縦棒）のある行はユーザが入力した行である．｜の無い行はシステム側の出力である．

図-2 Prologの動作

```
| copper(brass).
| copper(duralumin).
| copper(gunmetal).
| copper(red_brass).

| tin(solder).
| tin(gunmetal).
| tin(white_metal).
| tin(red_brass).

| ?- copper(X),tin(X).

X = gunmetal   ←――（注）

yes

| ?- copper(X),tin(X).

X = gunmetal ;    （次の答を要求した）

X = red_brass ;   （さらに次の答を要求）

no                （no：もうこれ以上は答が無い）
```

（注）答を1つ出力した後，システムは入力を待っている．

図-3 複数個の答を持つ場合

この辺の説明は，文章だけではうまく表現できないので，具体例を用いて説明を補いたい．すでに何度も登場しているcopperとtinのプログラムを使おう．ゴール節（1）を次のように3通りに変化させてみよう．

?－!, copper(*X*), tin(*X*).　(1a)
?－copper(*X*), !, tin(*X*).　(1b)
?－copper(*X*), tin(*X*), !.　(1c)

それぞれの場合にプログラムの実行結果は**図-4**のようになる．図中でカットの作用する範囲とは，カットの左側の部分を指す．

(1a)の場合には，カットの作用する範囲が無く，実行結果も図-3と変りがない．

(1b)の場合には，最初の候補 *X*=brass だけが選択の対象となり，tin(brass) が照合に失敗してもカットの作用する範囲の中では他の候補に切替えることができない．したがって(1b)のゴール節は失敗 (no) してしまう．

(1c)の場合は，カットの作用する範囲内での局所的なバックトラック (brass→duralumin→gunmetal という選び直し) は許されるから，最初の解が出力される．しかし；(セミコロン) による強制的なバックトラックは，カットの作用する範囲の外部からのバックトラックとなり，この場合の解の選び直しは不可能である．したがって，最初の解 (gunmetal) のみが求まる．

以上の例をまとめると，ゴール節中に含まれるカットの働きは**図-5**のように表すことができる．

今度は，プログラムの側にカットが含まれる場合を観察してみよう．**図-6**はプログラムの上半分にカットを付けた場合，**図-7**は下半分に付けた場合である．結論を言うと，図-6はゴール節(1b)と同じ動作になり，図-7は一見奇妙に映るがゴール節(1a)と同じ結果を

```
| ?- !,copper(X),tin(X).
X = gunmetal ;  ……2つの解が求まる
X = red_brass ;
no
| ?- copper(X),!,tin(X).
no
| ?- copper(X),tin(X),!.
X = gunmetal ;  ……1つの解だけが求まる
no
```

⌣ でカットの作用する範囲を示す．

図-4　カットを含むゴール節

(例) ?- copper (*X*), !, tin (*X*).
(一般形) ?- a, b, c, !, d, e, f.
カットの作用する範囲　範囲外

この中で局所的にバックトラックすることは許される．
ただし，範囲全体としては候補は1つに限られる．

範囲外の中だけに閉じたバックトラックは許される．
範囲外から範囲内へバックトラックしても次候補は得られない．

図-5　ゴール節に含まれるカットの働き

```
| copper(brass)      :- !.
| copper(duralumin)  :- !.
| copper(gunmetal)   :- !.
| copper(red_brass)  :- !.

| tin(solder).
| tin(gunmetal).
| tin(white_metal).
| tin(red_brass).

| ?- copper(X),tin(X).

no (失敗する)
```

枠で囲んだ部分がカットの作用する範囲

図-6　上半分にカットを付けた場合

```
| copper(brass).
| copper(duralumin).
| copper(gunmetal).
| copper(red_brass).

| tin(solder)      :- !.
| tin(gunmetal)    :- !.
| tin(white_metal) :- !.
| tin(red_brass)   :- !.

| ?- copper(X),tin(X).

X = gunmetal ;  ……2つの解が求まる
X = red_brass ;

no  (これ以上はない)
```

図-7　下半分にカットを付けた場合

もたらす．これは少し説明を要する．

まず図-6の場合は，カットの作用する範囲は図の枠で示した部分となる．この部分での解の候補が1つに絞られることから，*X*=brass となり，ゴール節は失敗する．

次に図-7の場合を説明しよう．カットの作用する範囲は上と同様に図の枠の部分である．この部分での解の候補が1つに制限される．ところで，注意すべきは候補の数である．と言うのも，tin(*X*) というゴー

ル節中の条件は，実際の動作においては，tin(brass), tin(duralumin), tin(gunmetal), … のように，変数に具体的な値が代入された形で使われる．こういう場合には，照合の結果は yes または no であり，変数の値の候補は特に存在しない．したがって，カットによる候補の制限は実効的な意味を持たない．これが図-7の動作がゴール(1a)と同じ結果をもたらす理由である．

カットは，以上のように候補の数を制限するために使われる．

また，カットの実用的な側面からは，もう1つ大切な効用がある．それは，バックトラックを抑制することにより，バックトラックのためにメモリ中に蓄えるべき情報が不要になることである．この点に関する詳細は，本稿の範囲外であり割愛するが，今日の大規模なプログラムには必ずと言って良いほどカットが含まれており，プログラムの効率化に寄与していることを指摘しておきたい．

3. 組込み述語・組込み関数

多くのプログラミング言語では，システムの中に最初から多種類のプログラムが組込まれている．これらは既製品のプログラムとしてユーザが自由に利用することができる．

例えば，FORTRANにおける組込み関数（FORTRAN では intrinsic function と呼ばれる）には，INT, MOD, FLOAT, SIGN, REAL などがあり，さらに基本外部関数として EXP, SIN, COS, SQRT 他が備えられている．普通は，これらの関数を使ってプログラムを組立てれば良いのであり，例えば SIN（三角関数）のプログラムをわざわざ書くことは滅多にない．

Lisp の場合には，組込み関数という言葉はあまり使われないが，muLISP[3] には primitively defined function として 77 個の関数が用意されている．大規模な Lisp の例では，Lisp Machine Manual[4] の索引には約1300個の関数が掲げられている．

プログラミング言語の機能の比較をするのに組込み関数の個数だけをうんぬんするのは，いささか乱暴であるが，利用者サイドから見た便利さの指標の1つには違いない．

Prolog では，述語（predicate）が基本的な構成要素になるから，組込み関数の相当品は「組込み述語」ということになる．なお，適当な述語と組合せて「関数」が使われることもある．

具体的な Prolog の処理系についてみると，Micro-Prolog[2]の組込み述語(built-in program と呼ばれる)は38個であり，有名なDEC-10 Prolog[1] の組込み述語は127個である．

個々の述語の詳細については省略するが，特徴的な点を以下に述べておく．

3.1 算術に関する組込み述語・関数

Prolog のプログラムの動作の説明をするのに，加算のプログラムが良く使われる（図-8）．このプログラムは動作の説明には便利であるが，実際に 100+200 の計算を実行すると，節②が100回，節①が1回呼ばれることになり，いかにも効率が悪い．

これに対して，組込みの加算の機能を使うと，Micro-Prolog で図-9，DEC-10 Prolog で図-10 のようになる．加算の他の算術演算もほぼ同様の形をとる．

図-9からも分るように，Micro-Prolog の加算の述語 SUM は減算にも使える．($x+200=300$ の x や，$100+y=300$ の y を求めることができる．)

DEC-10 Prolog（図-10）の加算は関数＋で表わされ，評価の述語 is と組合せて使われる．ここに is は2変数の述語であるが，infix 形式（2変数の間に述語が入る）で使われている．

```
①…… | plus(0,Y,Y).
②…… | plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

     | ?- plus(s(s(0)),s(s(s(0))),Z).

     Z = s(s(s(s(s(0)))))

     yes
```

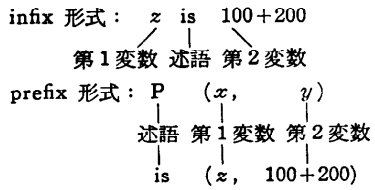
図-8 加算のプログラム

```
Which(z SUM(2 3 z))
Answer is 5
No (more) answers

&.Which(z SUM(100 200 z))
Answer is 300
No (more) answers

&.Which(x SUM(x 200 300))
Answer is 100
No (more) answers

&.Which(y SUM(100 y 300))
Answer is 200
No (more) answers
```

図-9 Micro-Prolog における加算の述語 SUM

```
| ?- Z is 2+3.

Z = 5

yes

| ?- Z is 100+200.

Z = 300

yes
```

図-10 DEC-10 Prolog における加算の関数



DEC-10 Prolog の標準的な表記法は prefix 形式であるが，中には is のように infix 形式をとるものもある．

3.2 入出力の述語

プログラムの中で入力 (input) や出力 (output) を取扱うには，どうしても組込み述語の力を借りなければならない．具体的な述語のレパートリは，処理系が前提としているハードウェアの構成やオペレーティング・システムの機能に大幅に依存する．したがって，一般的には論じにくい．**表-1** には Micro-Prolog の入出力用の組込み述語を，**表-2** には DEC-10 Prolog の入出力用の組込み述語のうち代表的なものをまとめておいた．

3.3 データベースの操作

今までに説明した組込み述語は，Prolog 以外のプログラミング言語においても対応する機能が見られた．組込み述語の中には，Prolog に特有なものも存在するので，ここで簡単に紹介しておこう．

代表的な述語として，データベースに対する操作と呼ばれる一群のものがある．ここで，データベースというのは実は Prolog のプログラムそのものを指す*．例えば，前出の合金の例題として2つのプログラムが登場した．第1のプログラムから第2のプログラムへ変形 (削除＋追加) するためには，**図-11** のようなゴール節を実行すれば良い．ここに，retract(…), assertz(…) はともに組込み述語であって，プログラムに対する削除と追加の機能を持つ．

* Prolog のプログラムは，データベースとして解釈することもできる．

表-1 Micro-Prolog の入出力用の述語

	述語名	機能
コンソール入出力	R	read：キーボードから入力する
	P	print：コンソールに出力する
ディスク入出力	OPEN	ファイルのオープン
	CREATE	出力ファイルの新規作成
	CLOSE	ファイルのクローズ
	READ	ファイルからの読込み
	WRITE	ファイルへの書出し

表-2 DEC-10 Prolog の入出力用の述語 (代表的なもの)

	述語名	機能
ファイル操作	see	入力ストリームのオープン
	seen	入力ストリームのクローズ
	tell	出力ストリームのオープン
	told	出力ストリームのクローズ
入出力	read	読込み
	write	書出し
	get	1文字単位の入力
	put	1文字単位の出力

なお，図-11 の中ではプログラムの全体を見るのに listing という指示を行っているが，これも一種の組込み述語である．

4. リストと差分リスト

リストが柔軟性に富む優れたデータ構造であることは広く知られている．またリスト構造に基礎を置く Lisp (語源は LISt Processor) というプログラミング言語が成功をおさめていることは誰しも認めるところである．

4.1 リストの記法

Prolog においても，リストを取扱うために特別の記法を用意している．それは，要素を [,] でくくることである．

(例) [north, east, west, south]

このリストを木 (tree) で図示すると **図-12** のようになる．図中の | (縦棒) は Lisp で言えば CONS に相当する関数である．右下の [ ] は空リスト，すなわち Lisp の NIL である．上のリストを Lisp 風に書き直せば，(CONS 'north (CONS 'east (CONS 'west (CONS 'south NIL)))) となる．[north, east, west, south] という表記法は，このような木構造を表現していると考えれば良い．

```
| ?- listing.

copper(brass).
copper(duralumin).
copper(gunmetal).
copper(german_silver).

tin(solder).
tin(gunmetal).
tin(white_metal).
tin(fusible_alloy).

yes
```

操作前のプログラム
図-2 のプログラムと同じもの.

```
| ?- retract(copper(german_silver)),assertz(copper(red_brass)).
yes
| ?- listing.

copper(brass).
copper(duralumin).
copper(gunmetal).
copper(red_brass).

tin(solder).
tin(gunmetal).
tin(white_metal).
tin(fusible_alloy).

yes
```

copper (german_silver) を削除 (retract) し, copper (red_brass) を新たに追加 (assert) した結果.

```
| ?- retract(tin(fusible_alloy)),assertz(tin(red_brass)).

yes
| ?- listing.

copper(brass).
copper(duralumin).
copper(gunmetal).
copper(red_brass).

tin(solder).
tin(gunmetal).
tin(white_metal).
tin(red_brass).

yes
```

tin (fusible_alloy) を削除し tin (red_brass) を追加した結果. これで図-3のプログラムと同じになる.

図-11 データベースに対する削除と追加 (および listing)

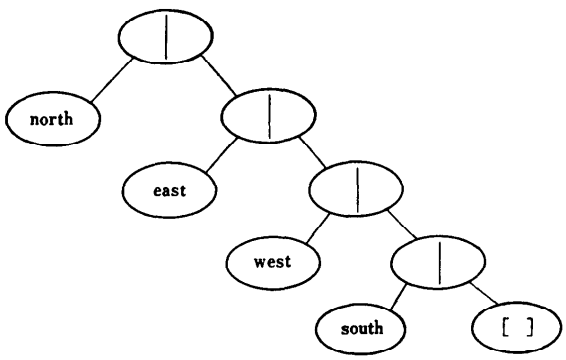


図-12 木 (tree) による表現

なお, [,] の中の区切り記号として , (コンマ) の他に | (縦棒) を使用することも認められている. これは一般に変数を含むリストに使われて,

[alpha, beta|L], L は変数

と書けば, alpha, beta で始まるリストを意味する (**図-13** 参照).

このように, Prolog のリストの表記法は Lisp の場合とあまり変らない. ただし, 実際のプログラムに使う段になると, それぞれの言語の特色が出てくる.

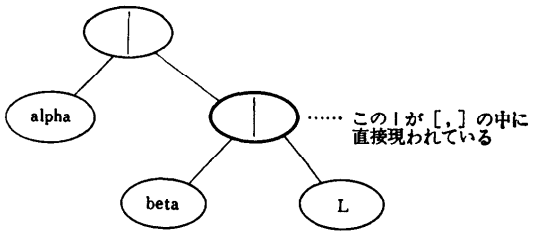


図-13 〔alpha, beta|L〕

例えば，次のプログラムは，要素XがリストLの中に含まれているか否かを判定する Prolog のプログラムである．

① member $(X, [X|L])$.

② member $(X, [Y|L])$: —member(X, L).

これと同じ働きをする Lisp のプログラムを書くと下のようになる．

```
(DEFUN MEMBER (X L)
  (COND ((ATOM L) NIL)
        ((EQUAL X (CAR L)) T)
        (T (MEMBER X (CDR L))) ))
```

この2つのプログラムは動作の仕方も似ている．ただし Prolog では $[X|L]$, $[Y|L]$ というリストのパターンを使って X や L についての条件を表わしているのに対し，Lisp では同じ仕事を (CAR L), (CDR L) というリスト分解の関数を使って書いている．

この違いは，両言語の特徴を素直に反映している．

4.2 差分リスト

リストの話の余録として，差分リスト (difference list) について触れておこう．差分リストという言葉は，リストの使い方を指す言葉であり，道具立としては通常のリストと同じものを用いる．

差分 (difference) という言葉のニュアンスをつかむには図解して示すと分りやすい．上田和紀氏の差分リストの解説[5]には，リストの終端をアースするという図法が載っている (**図-14**)．筆者はアースの連想から乾電池の接続図を考えた．すると，良く知られているように電圧とは「電位の差」であるから，一連の電池からはさまざまな電圧が取出せる (**図-15**)．

このたとえ話をリスト構造に置戻すと，2つのリストの差で部分リストを表わすことができる (**図-16**)．これが差分リストである．差分リストを活用することによって，ある種のリストの演算を効率良く実行することができる．例えば，2つのリストをつなぐ (append) 操作は，差分リストを使うと高速に実行できる．

差分リストに関する文献としては，上記上田氏の解説[5]，Clark と Tärnlund によるクイックソートのアルゴリズムの論文[6]がある．

5. 文字列の取扱い

文字列 (string) を取扱うことのできるプログラミング言語としては SNOBOL が代表的存在であるが，他の言語にも文字列操作機能は良く備えられている．

例えば，多くの Lisp には string に関する関数が備わっている．また，最近のパーソナルコンピュータ用の BASIC の中には文字列処理を強化したものが多い．

Prolog における文字列の取扱いは，従来はとかく片手間であり，DEC-10 Prolog では文字列を2重引用符 " で囲んで

"abc"

とすると，

[97, 98, 99] (ASCII コードのリスト)

というリストを返す．という程度の略記法しか持って

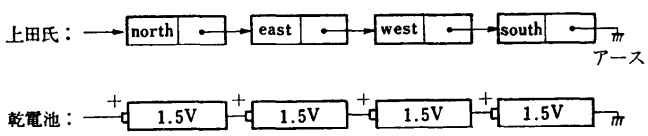


図-14 上田氏の図[5]と乾電池の接続図

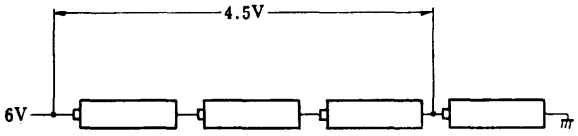


図-15 電位差（リストとして見れば部分リスト）

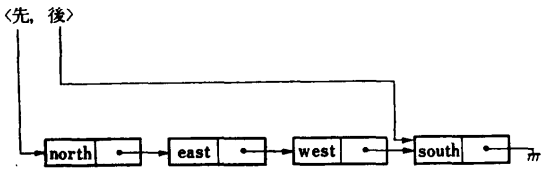


図-16 差分リストによる〔north, east, west〕の表現
（先と後との差が意味を持つ）

いなかった.

これでは,本格的な文字列処理には不十分であるとして小長谷・梅村[7]両氏はSNOBOL並みの文字列パターンマッチ機能をPrologに導入している.ただし,同論文[7]も指摘しているように,文字列の取扱いには難しい問題がある.

例えば,「文字列の難しさ」という文字列を考えてみよう.この文字列の分解の仕方は,図-17に示すように幾通りもある.図中の・(黒丸)は文字列の連接を表わす.つまり,より一般的に言うと連接(・)という関数は次の式(associative law)を満たす.

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

ところが,Prologの他の構成部分は,これと異なり,各構成要素は一意に分解される.換言すれば,文字列の性質はPrologにとって異質のものである.

この問題は本質的であり,Prologに対する文字列の追加は単純にはいかない.それだけに活発な研究が行われており[8],今後の発展が期待される.

6. 処理系による違い

目下のところ,Prologは発達中の言語であり,各種の処理系が提案されている.したがって,上に述べたような話題も,処理系によっては必ずしも当てはまらない部分もあり得る.

本稿では,筆者の使用経験からDEC-10 PrologとMicro-Prologを中心に述べて来たことをお断りしたい.

最後に,他の処理系の話題も多少交えて,処理系による差異について付言しておく.

6.1 シンタックスの違い

プログラムの表記法は,DEC-10 Prologにならうものが増えている.この表記の特徴は,述語が( )の外側(左)に出ていることで,

```
述語(変数, 変数)
member(X, [Y|L])
```

という形になる.一方,Micro-Prologでは述語が( )の内側に入るので,

```
(述語 変数 変数)
(member X (Y|Z))
```

となる.節全体の形を示せば,DEC-10 Prologが,

```
menber(X,[Y|L]) :—member(X,L).
```

一方Micro-Prologでは,

```
((member X (Y|Z)) (member X Z))
```

となり,Micro-Prologの書き方はLispと見かけ上は同じである.

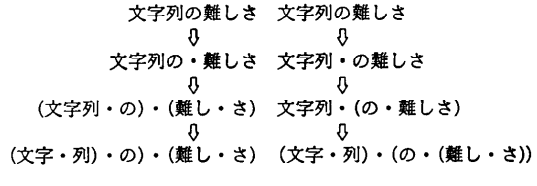


図-17 文字列の分解(文字の意味とは無関係に分解)

DEC-10 Prolog流の書き方を支持する人の言い分は,通常の$f(x,y)$の形の表記に近い方が人間に見易い,という点にある.一方,Micro-Prolog流の利点として,内部表現の占めるメモリ量や実行時間の効率が良いこと,またプログラムによるプログラム自身の操作に便利であること,が主張されている.

これは丁度,Lispにおける M式とS式との比較をしているのに似ている.Lispの歴史では,$f(x)$スタイルの外部形式(M式)はほとんど使われなくなり,元来は内部形式であったはずのS式が言語の表面に析出したが,Prologの場合は今後どのような経過をたどるか興味深い.

なお,シンタックスの細かい話題としては,変数と定数の区別の仕方などがあるが,ここでは省略する.(文献9)の付録A1にまとめてある.)

6.2 モジュール構造

Micro-Prologは,その特徴の1つにモジュール構造の機能をうたっている.実際,Micro-Prologに付属しているプログラム(一種のライブラリ)はモジュールの形式をとっている.

DEC-10 Prologは,インタプリタで使う時にはモジュールに相当する機能は無い.一方コンパイラを使う時には,外部から見える述語(public)を宣言する必要があり,コンパイル単位が実質上のモジュールの役割を果たす.

モジュール機能を持つPrologの例としては,他にMPROLOG[10]がある.またPROLOG/KR[11]におけるworldの考え方は,論理的な基盤を持つものとして注目して良い.

6.3 コンパイラ

プログラミング言語は実際に使われるものであるから,実行速度が速いということは大切な点である.Lispの歴史を見てもコンパイラの発展に伴って大型のアプリケーションが可能になったと言われている.

DEC-10 Prologの開発者の1人,D. H. D. Warrenは「Prologの実行速度はLispに比べて遜色ない」と

いう挑戦的な論文[12]を発表して注目を集めたが，彼の主張の説得力は DEC-10 Prolog のコンパイラの高速性によるところが大きい．確かに，Prolog が今日のように注目を集めるようになったのも，同コンパイラの存在が大きく寄与している．

未だ，多くの処理系にはコンパイラが備わっておらず，Lisp に比べると未発達の感は否めない．しかし事態は急速に進展しつつある．今後の展開に期待したい．

参 考 文 献

1) Bowen, D. L. : DEC System-10 PROLOG USER'S MANUAL, University of Edinburgh, Dept. of Artificial Intelligence (1981).
2) McCabe, F. G. : Micro-PROLOG Programmer's Reference Manual, Logic Programming Associates (1980).
3) The muLISP/muSTAR-80 Artificial Intelligence Development System Reference Manual, The Soft Warehouse (1980).
4) Weinreb, D., Moon, D. and Stallman, R : Lisp Machine Manual, Fifth Edition, System Version 92, Symbolics (1983).
5) 上田和紀：リストと差分リスト，bit, 1983年5月号，pp. 78-82 (1983).
6) Clark, K. L. and Tärnlund, S-Å. : A First Order Theory of Data and Programs, IFIP-77, pp. 939-944, North-Holland (1977).
7) 小長谷明彦，梅村 護：Shape Up の文字列照合アルゴリズムについて，情報処理学会記号処理研究会資料 24-7 (1983).
8) 上田和紀，中島秀之，戸村 哲：Prolog における作用的入出力と文字列処理，Proc. of the Logic Programming Conference '83, 5-1, ICOT (1983).
9) 後藤滋樹：PROLOG入門，サイエンス社(1984).
10) MPROLOG Language Reference Manual, Version 1. 3, SZKI, Hungary (1983).
11) Nakashima, H. : PROLOG/KR User's Manual, METR-82-4, Univ. of Tokyo (1982).
12) Warren, D. H. D. and Pereira, L. M. : Prolog-The Language and Its Implementation Compared with Lisp, ACM, Proc. Artificial Intelligence and Programming Languages, pp. 109-115 (1977).

(昭和59年8月6日受付)