B 4　LOREL言語の使用経験とその検討

片山卓也，榎本　進，榎本　肇　（東京工業大学工学部）

1. はじめに

筆者等は，数年前より，論理関係処理言語ＬＯＲＥＬの研究を行ってきたが，これは，オートマトン，グラフ，形式言語等の組合せシステムを記述するための高級言語である。これらの組合せシステムの処理は，データ間の関係，処理と考えられるが，このような問題を記述する高級言語を設計する際の最大の問題は，高級言語にふさわしい形でデータ構造をいかに導入するかということである。

ＬＯＲＥＬでは，データ集合S_1，…，S_n中の関係Ｒが一般に

$$R \subseteq \{ (x_1, \cdots, x_n) \mid x_1 \in S_1, \cdots, x_n \in S_n \}$$

という形でとらえられ，かつ，このような記法が多くの組合せシステムの記述に広く利用されていることを考えて，ｎ組（以後ｔｕｐｌｅという）と集合（以後ｓｅｔ　という）をその基本的データ構成とすることに決定した。

これにもとづいて言語仕様が検討され，その１ｓｔ　ｖｅｒｓｉｏｎであるＬＯＲＥＬ－１がｉｍｐｌｅｍｅｎｔされたが，これには，言語仕様上及びｉｍｐｌｅｍｅｎｔａｔｉｏｎ上いくつかの問題が含まれており，今後，＂実用になる＂LORELシステム構成する上で，これについて検討することは非常に重要であると考えられる。本報造では，使用経験（現在までのものでは必ずしも十分ではないが）も含めて，ＬＯＲＥＬ－１の仕様及びｉｍｐｌｅｍｅｎｔａｔｉｏｎについて，検討を加える。

以下，簡単にＬＯＲＥＬ－１の仕様，ｉｍｐｌｅｍｅｎｔａｔｉｏｎについて述べ，次に，そのデータ構成，仕様およびｉｍｐｌｅｍｅｎｔａｔｉｏｎ上の問題点について述べる。なお，使用経験については，原稿提出時点でデータの収集が必ずしも十分ではないので，詳細は当日発表する。

2. LOREL－1言語の概略 [1),2)]

2.1　データの種類

ＬＯＲＥＬ－１で許されるデータは次の４種類である。

ｎｕｍｂｅｒ …… 通常の整数

ｃｈａｒａｃｔｅｒ…… 単一の文字

　　Ａ～Ｚ，０～９，特殊文字

ｔｕｐｌｅ …… $(e_1, \cdots, e_n)$，順序組

ｓｅｔ …… $\{e_1, \cdots, e_m\}$，整列集合

(1)　ここで，ｔｕｐｌｅは固定長であり，その成分e_1，…，e_nは異ったデータ型のものでもよく，又ｓｅｔについては，同一のデータ型の要素e_1，…，e_mからなる可変長データであり，要素の数が０である{ }も空集合として意味をもつ。

(2)　ｔｕｐｌｅ，ｓｅｔがともにデータの並びでありながら，上のように区別されている理由は次のとおりである。関係を集合の直積によって記述する場合には，通常，集合は同じような性質（データ型）を持った不定個のデータの集りであり，それに対する操作は，全ての要素に対して一様なものになるのが普通である。これに反し，関係の要素であるｔｕｐｌｅは，性質の異るデータを単にｇｒｏｕｐｉｎｇ　したものと考えられる。

(3) tupleの成分，set の要素としては，再びtuple，set であることが許されるので，非常に複雑なデータを構成することが出来る。

(4) tupleZのi番目の成分をZ(i)で，又，set Sのj番目の要素をS〔j〕で表し，このような修飾を何重にも行うことを許す。 例：X〔2〕(3)〔N+5〕

2.1 データ型とデータ構成

LOREL-1データの正確に定めるには，データ型を導入しなければならない。

データ型は，n（number に対応），c（characterに対応），(，)，{，}，' から構成される文字列であり，その集合Tは次のように帰納的に定義されている。

(i) n，c∈T，(ii) t_1，…，t_n T ⇨ (t_1，…，t_n)∈T，(iii) t∈T⇨{t}∈T

このとき，LOREL-1データの集合Dは

$$D = \bigcup_{t \in T} D_t$$

と定められる。ここでD_tは，データ型がtのデータ集合であり，次のように帰納的に定義される。

(i) D_n = number 全体からなる集合

(ii) D_c = character 全体からなる集合

(iii) $D(t_1, \cdots, t_n) = \{(d_1, \cdots, d_n) \mid d_i \in D_{t_i}\}$

(iv) $D\{t\} = \{\{d_1, \cdots, d_m\} \mid d_i \in D_t, m \geq 1\} \cup \{\{\}\}$

〔例〕 i) {1，2，3，4}∈D，データ型t={n}

ii) (1，#A，{10，20})∈D，t=(n，c，{n})

iii) {(1，#A，2)，(2，#B，3)}∈D，t={(n，c，n)}

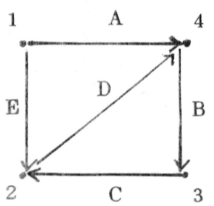
iv) {{1，2，3}，{4，5}，{}}∈D，t={{n}}

v) {1，#A，2}∉D

2.3 構造データのDによる表現の例

Dによって構造データを表す方法のいくつかを示す。

(i) グラフ



GRAPH1
={(1，#A，4)，(4，#B，3)，(3，#C，2)，
(2，#D，4)，(1，#E，2)}

GRAPH2
={{(#A，4)，(#E，2)}，{(#D，4)}，
{(#C，2)}，{(#B，3)}}

GRAPH1は，グラフを枝の集合に表現したものであり，枝は（始点，ラベル，終点）の形に表わされている。

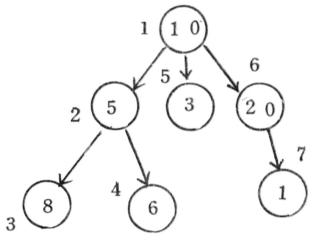
GRAPH2は，上とは逆に頂点の集合でグラフを表現したものであり，頂点は，（その頂点に接続している頂点，枝のラベル）の集合によって表わしている。

グラフのパスを辿る等の作業は，GRAPH2の形式の方が行いやすい。

(ii) 木

グラフと同様に表わされるが，施される操作を考えると，GRAPH2と同じ形式で，かつ，木をpreorderに辿った順に頂点を並べたものが便利である。以下では，各頂点から出る枝の本数が不定で，かつ，頂点にのみラベルのついている場合を示す。

TREE＝{（10，{2，5，6}），（5，{3，4}），
（8，{}），（6，{}），（3，{}），
（20，{7}），（1，{}）}



このような表現に対しては，部分木，木の合成等の操作は簡単に行える。

(iii) 形式文法

V_N，V_Tの元が整数に符号化してあれば，書換え規則の集合Pは

$$P=\{（ルール番号，\underbrace{\{n_1，\cdots，n_k\}}_{左\ 辺}，\underbrace{\{m_1，\cdots，m_t\}}_{右\ 辺}），\cdots\}$$

と表すのが適当であろう。

(iv) スパース行列

$$\begin{bmatrix} 10 & 0 & 40 & 0 \\ 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 \\ 20 & 30 & 0 & 60 \end{bmatrix}$$

MATRIX＝{（1，1，10），（1，3，40），（2，4，50），（4，1，20），
（4，2，30），（4，4，60）}

ROW＝{{1，2}，{3}，{}，{4，5，6}}

COLUMN＝{{1，4}，{5}，{2}，{3，6}}

ここで，MATRIXは，非零要素について，（i，j，a_{ij}）を集めたものであり，又，ROWのi番目の要素はi－th row中の非零要素のMATRIX中での位置を示している。COLUMNも同様。

2.4 文

LOREL－1の文の設計を行う際に特に考慮したことは，

(i) 表現がコンパクトで，一見して書かれている内容がすぐにわかること。

(ii) データが特定の形の文でのみ変化し，又，implicit な変化がないこと。

(iii) 入力データ以外のデータ型チェックが，コンパイル時に可能であること。

等であり，又，文のレベルは基本的なもののみとし，複雑なものは手続きによるという方針をとった。以下，各文について簡単に説明を行う。

1) 代入文

LOREL－1では，データの操作は全て代入文によって行われるが，これらには，データのコピー，set への要素の追加，set からの要素の削除があり，それぞれ次のような形の代入文によって行われる。

(i) X＝Y　コピー

(ii) E↑＝Y又は，↑E＝Y　追加

(iii) E＝ε　削除

ここで，Xはset変数，Yは式（算術・論理式，set 式）又はtuple，character 素データ（リテラル，変数，関数呼出し）であり，Eはset の要素を指す変数でS〔 i 〕の形をしているか，set の要素を取出す形の繰返文での制御変数である。(i)は，Yをコピーしてに与えることを，(ii)は，Eの指す要素の右，左隣りにYを追加することを，又，(iii)はEの指す要素を取除くことを表す。

2） 繰返し文

繰返し文には，（i＝α，β）σ （ただしσ：文）の形の通常のものの他に，（ξ∈X）σの形の繰返し文がある。これはset Xの要素を次々と取出し，これをξに与えて文σを実行しようとするものである。

3） 宣言文

全ての基本変数は，それが受けるデータのデータ型とともに宣言されていなければならない。

例 TREE：{（n，{n}）}，ROW：{{n}}

4） その他の文

以上の他に，〔B_1 ⇨S_1 ，B_2 ⇨S_2 ，…，B_n ⇨S_n 〕の形の条件文，〔S_1 ；S_2 ；…；S_m 〕の形の複合文，入出力文，手続き定義・呼出し文，その他の制御文等がある。

2.5 プログラム例

(i) グラフの頂点集合間の距離（それらを結ぶパスの最短長）を求める手続き。

以下で示す手続きDISTANCE1は，グラフが2.3のGRAPH1 の形に表現された場合であり，又，DISTANCE 2は，GRAPH2 のように表現された場合のものである。

```
define（DISTANCE1：X，Y，GRAPH）；
GRHPH：{（n，n）}，X，Y，Z：{n}，T：（n，n），
DISTANCE 1，N：n；
Z＝X；DISTANCE 1＝0；
TEST：（N∈Z）〔N∈Y⇨return〕；
（T∈GRAPH）〔T(1)∈X⇨Z〔0〕↑＝T(2)〕；
DISTANCE 1＝DISTANCE 1＋1；
goto（TEST）；end；

define（DISTANCE 2：X，Y，GRAPH）；
GRAPH：{（n，{n}）}，X，Y，Z：{n}，
L，M，DISTANCE 2：n；
Z＝X；DISTANCE 2＝0；
TEST：（L∈Z）〔L∈Y⇨return〕；
（L∈X）（M∈GRAPH〔L〕(2)）〔Z〔0〕↑＝M〕；
DISTANCE 2＝DISTANCE 2＋1；
goto（TEST）；end；
```

(ii) 部分木，葉の集合を求める手続き

2.3の形式で与えられた木の最初の部分木（根に接続している最左の部分木）をコピーする手続きCAR (TREE) を

を示す。

```
define (CAR:TREE) ;
CAR, TREE : { (n , { n }) } , Z={ n } , I , N , T : n ;
CAR={ } , N=number (TREE) ;
〔 N≦1⇨return 〕;
〔 number (TREE〔1〕(2) ) ≠1⇨N=TREE〔1〕(2)〔2〕−1 〕;
(I=2 , N) 〔 Z={ } ;
            (T ∈ TREE〔I〕(2) ) 〔 Z=Z ∪ { T−1 } 〕;
          CAR=CAR ∪ { (TREE (1) , Z) } 〕;
return ; end ;
```

又，木の葉を与える手続きLEAVES (TREE) は，

```
define (LEAVES : TREE) ;
TREE : { (n , { n }) } , LEAVES : { n } , NODE : (n , { n })
LEAVES={ } ;
(NODE ∈ TREE) 〔 NODE (2) ={ } ⇨ LEAVES=LEAVES ∪ { NODE (1) } 〕;
return ; end
```

(iii) Greibach 標準形をしたＣＦ文法に対する構文解析手続きが，文献１）に与えられている。

3. LOREL−1 プロセッサ構成の概略

ＬＯＲＥＬ−１では，データ構成が複雑で，かつ，その構造が実行時に動的に変化するので，プロセッサの構造も簡簡ではないが，データ構成の言語理論的形式化，記憶構造のセル構造としての形式化およびプロセッサの形式的記述によって，その implementation を見通しよく行うことが出来た。以下，これらについて簡単に述べる。

ＬＯＲＥＬ−１プロセッサは，コンパイラとインタプリタから構成されているが，コンパイラは，ソース文を解析してインタプリタに対する命令（以後，ｓ−命令とよぶ）に展開し，インタプリタは，このｓ−命令の列を受けて実際のデータ構造の処理を行っている。

(1) コンパイラの動作は通常のものと大差ないが，入力文以外の文のデータ型に関する正しさのチェックが，全てコンパイル時に行われており，それによって，特に式の評価の部分が少し複雑になっている。このチェックには，LOREL−１データの集合Ｄの言語理論的形式化が利用されているが，次にこれについて簡単に触れる。

(2) ＬＯＲＥＬ−１で許されるデータの集合Ｄは 2.1 で，データ型を介して帰能的に定義されているが，これに対する受理機械①が構成され，Ｄがｎｏｎ−CFL なＣＳＬであることが明らかにされた。機械①はコンパイラに組込れて，ソース文のデータ型チェックに用いられているが，①は，ｓｅｔ の要素が同一のデータ型であることを調べるために，擬データ型のｉｎｆ を計算するのが主な働きである。

データ型がｔのデータ集合D_t は，決定性ＣＦＬとなるが，これに対する受理機械は，インタプリタが入力データのデータ型チェックに用いている。[2)]

(3) インタプリタは実行時にデータ構造の処理を行う機械であるが，記憶構造のセル構造による形式化にもとづいて，

その構造が整理されているので，implementation は比較的楽であった。

セル構造は基本的な８種類のセルから構成され，リテラル以外のＬＯＲＥＬ－１データは，実行時には，全てこれによって表現されている。（文献３参照）

インタプリタは，次の各部分から構成されている。

スーパバイザ

セル・プロセッサ，セル・プール

リテラル・プロセッサ，リテラル・プール

データ型・プロセッサ，データ型・プール

変数テーブル・プロセッサ，変数テーブル

レジスタ群

インタプリタは，読込んだｓ－命令を解読して（by スーパバイザ）によって，これらの各機械を動作させるが，ｓ－命令としては，現在のｖｅｒｓｉｏｎ のものでは，３８個用意されている。又，ｓ－命令は，スーパバイザによって，各部分プロセッサに伝達されるが，これを実行するために部分プロセッサの持っている命令の総数は，２６個である。更に，インタプリタ内の情報の交換は，１０個のシステム・レジスタと１２８個のワーク・レジスタによって全て行われている。

(4) implementationはFORTRANによって，NEAC ３２００モデル５０（３２KW，１６bit／W）上でなされた。常駐部であるインタプリタの大きさは約８KWである。

4. LOREL－１データ構成の検討

ＬＯＲＥＬ－１のデータによって直接表現しうるのは，線形リスト，宣言によって形が定った木および，宣言によって定められた形のそれらの入れ子構造である。

このような構造にデータの形を制限した理由は，既に述べたように，多くの組合せシステムがこのような形で統一的に記述可能であり，この限定されたデータのもとでコンパイル時データ型チェックを可能な限り行うことによって，デバッグを容易にすることが出来ると考えたからである。

このような設計思想は，いくつかのプログラムを書いてみることによって，実際に妥当であることが確認されたが，複雑な形の構造データをＬＯＲＥＬ－１データで表現する場合には，いくらかの問題が残り，これを解決するために２，３の工夫がなされた。

一般のグラフや形が動的に変化する木をＬＯＲＥＬ－１で表現する方法のうち代表的なものは，2.3で述べたグラフに対する表現ＧＲＡＰＨ１とＧＲＡＰＨ２の方法である。

前者の方法は，通常，数学等で関係を表現するのに用いられ，グラフ，オートマトン等の組合せシステムは，このような形で定義されるのが普通であり，ＬＯＲＥＬ－１も初期の導入過程ではこのようなデータの処理をその目的と考えていた。この表現法は確かに一般的ではあるが，その処理では大きな問題がある。それは，構造データの処理では，それを辿るという操作が多いが，この表現では，辿る操作はｓｅｔ 中の探索という時間のかかる操作によって実現されるからである。したがって，ｓｅｔ の探索の高速化が必要であり，特に，ｔｒａｎｓｉｔｉｖｅ な探索を能率良く行うことが重要である。

ＬＯＲＥＬの初期の仕様[3] では，この目的のために，組成集合，インデックスという概念が導入され，手の込んだ探索をシステムが自動的に実行する方法が考えられたが，これは，そのimplementation のための記憶構造が非

常に複雑で，かつ，組成集合に implicit な変化が生じるので，現在のＬＯＲＥＬ－１には取入れられなかった。

一方，グラフを表現するためのＧＲＡＰＨ２の方法は，グラフ中の頂点の接続関係をポインタ（要素のｓｅｔ中での位置を利用したもの ― 以後ｓｅｔ ポインタという）によって表現したものである。この方法は，ｓｅｔとｔｕｐｌｅを上手に利用したもので，ポインタ操作をユーザが行わなければならないが，適当な標準的な手続きを用意しておけば，十分に使い易い方法である。この方式を更に進めると，データとして，いわゆるｒｅｆｅｒｅｃｅ データを導入した方がｔｒａｖｅｒｓｅ 等の速度が上り有利である，という立場も考えられる。

我々も，ｒｅｆｅｒｅｎｃｅデータを＂ｓｅｔ 中の場所の名前＂という形で導入することを考え，これを含めた形でプロセッサ（インタプリタ）を構成したが，これには次のような問題がある。

ｉ） ｒｅｆｅｒｅｎｃｅデータがデータ構成に含まれるときには，コンパイル時におけるデータ型チェックに，いろいろの問題が生じ，又，データ型チェックを実行時に行うことは，実行速度，デバッグの点で面白くない。（最初に試作したインタプリタは，実行時にチェックを行っていたので，現在のインタプリタに比較して２倍以上遅い）

ⅱ） ｒｅｆｅｒｅｎｃｅデータを用いると，構造のｔｒａｖｅｒｓｅ は確かに速いが，構造のコピーやデバッグのための出力等は，ｎｕｍｂｅｒ を使ったｓｅｔ ポインタ方式の方が格段に簡単である。

以上の理由によって，ｒｅｆｅｒｅｎｃｅデータもＬＯＲＥＬ－１には加えられず，その結果として，ＬＯＲＥＬ－１のデータ構成は，単純かつ平易なものになったが，この選択は誤りではなかったと考えている。

ｓｅｔ ポインタ方式に関連して，ＬＯＲＥＬ－１でのｓｅｔ では，要素が整列されていることに触れておく。

ｓｅｔ ポインタ方式を実現するには，整列集合の方が便利であり，又，ＬＯＲＥＬ－１では文字列を，文字を要素とするｓｅｔ で表現しているのでこの方式の方が都合よい。応用上，整列集合と非整列集合のどちらの比率が多いかは定かでないが，現在の方式の方がｆｌｅｘｉｂｌｅ であると思われる。

5. ＬＯＲＥＬ－１ データ構成の拡張

5.1 木データの導入可能性

ＬＯＲＥＬ－１データによって＂直接＂に記述出来るのは，基本的には線形リストであるが，これを，仕様上調和のとれた形で拡張出来るならば，言語の記述能力の点で望ましい。特に，動的に変化する木構造の導入は非常に有用である。

ＬＯＲＥＬ－１の設計段階でもこの問題が考えられたが，ｓｅｔ，ｔｕｐｌｅ で実現出来る形の定った木構造との調和を保ちながら導入するための方法が，その時点では見つけられなかった。

現在，この問題に対して，ALGOL 68 のmode definition と類似の方法で，以下に述べるような解決方法が考えられている。

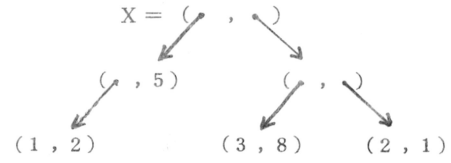
この方法は，動的に変化する木を，それが変化する過程に現われる全ての木（これはＬＯＲＥＬ－１データによって直接表現出来る）の集合と考え，このような集合に属するどの木をも受けることの出来るデータ型を新しく定義するものである。このデータ型は，内容的には，その集合に属する木のデータ型よりなる集合，という形でとらえることが出来，そのデータ型集合（一般には，無限集合である）を生成する文脈自由文法の始記号によって表わされる。

以下，例について説明する。

〔例１〕

```
$TREE1＝（$TREE1，$TREE1）｜n
X：$TREE1
```

これによって，Xは，葉にのみnumberのラベルのついたbinary tree として宣言される。ここで，$TREE1は，"非終端データ型記号"であり，生成規則$TREE1 =（$TREE1 ｜$TREE1 ）｜nによって生成されるデータ型n，（n，n），（（n，n），n），（n，（n，n）），……の全てを，変数Xは受けることが出来る。



〔例2〕

$TREE2＝（$TREE2，$TREE2，{c}）｜{c}

これは，全ての頂点が文字列によってラベルづけされているbinary tree を定義している。

〔例3〕 set を含んでいる場合には，

$TREE3＝{$X}；$X＝$TREE3｜n

によって，各頂点から出る枝の本数が不定である木を定義するように拡張するのが良いであろう。このときには，$TREE3で表わされるどのデータも，現在のLOREL－1のデータでは表現することが出来ない。

さて，以上述べたようなデータ型の拡張は，次に述べる，非終端データ型記号を終端させずに残しておく方法，とともに，これを実現するためにLOREL－1プロセッサを増強すべき機能は，データ型チェックに関する部分だけである。又，言語仕様としては，基本的には，ある変数がさすデータがset，tuple，character，number のいづれかであるかを判定するための述語を用意するだけでよく，このような機能の導入は，簡単に行うことが出来る。

5.2 不定データ型の導入

既に述べたように，LOREL－1では全ての変数，関数は，それが受けることの出来るデータ型が一意に決められていなければならず，これによって，厳格なコンパイル時データ型チェックが可能になっているのではあるが，これは，一面では，手続きのflexible な記述を妨げている。

例えば，set の要素の数を求める手続きCARD（X）について考えてみると，このような手続きの本体は，set Xのデータ型とは無関係であり，例えば，次のように書かれる。

CARD＝0；（T∈X）〔CARD＝CARD＋1〕；

このとき，これは，{Tのデータ型}＝Xのデータ型，が成立していれば，どのようなXについても正しく働くが，宣言部をつけてLOREL－1の手続きとするためには，T，Xのデータ型を1つに定めねばならず，したがって，データ型ごとに1つの手続きを必要とする。

この問題は，データ型チェックを少し弛めれば解決されるが，論理的に可能な範囲内ではデータ型チェックは行うべきであるから，その方法には工夫が要る。

現在では，次に述べるような方法が適当であろうと考えている。

それは，前節5.1で導入した非終端データ型記号を終端させずに残しておくという方法である。このとき，データ型チェック時においては，この非終端データ型記号は自分自身とのみマッチすると考える。

例えば，2つのset のunionを求める手続き（これは，LOREL－1ではset オペレータとして用意されているが）は，次のように書くことが出来る。

```
define (UNION:X,Y);
UNION,X,Y:{$ELEMENT},T:$ELEMENT;
UNION=X;
(T∈Y)〔↑UNION〔∞〕=T〕;
return;end;
```

この方法を用いて関数手続きを書く際に重要なことは，パラメータのデータ型から，関数のデータ型を計算する方法が用意されていることである。例えば，次の手続きMSET（X，Y）のでそれは，そのような簡単な例であるが，もっと表現力の強いものが望まれる。

```
define (MSET:X,Y);
MSET:{$T},X,Y:$T
MSET={X,Y};
return;end;
```

6. LOREL—1文の検討

複雑な内容を記述する言語としては，その表現がコンパクトで見易い，ということが非常に重要であろう。LOREL－1でも，一応この方針にしたがって文の設計を行ったが，文のレベルを低いものに設定したので，見易さという点で必ずしも十分なものとはいえず，もっと高レベルの文を，たとえ機能の重複があったにしても，導入すべきであると考えている。

どのような形の文が良いか，ということは，心理的な問題もあり一概には決めがたいが，筆者らの経験では，データの生成，変換過程を示す手続きを見せられるより，その計算結果を示される方が多くの場合見易いようである。例えば，

```
S={F(X,Y)|X∈P,Y∈Q:X+Y≥0}
```

なる表現は，内容的には，

```
S={};
(X∈P)(Y∈Q)〔X+Y≥0⇨↑S〔∞〕=F(X,Y)〕
```

と同じであるが，前者の方が優れている。

同じような理由から

```
(Σ,X∈P)G(X),(∀,X∈P)B(X)
```

等の記述も許したいと考えているが，LORELがset を中心にしているという特徴が，このような記述の面でも生かせると思われる。

また，記号の使い方については，長い歴史の検討をうけてきた数学での表現方法は，内容をコンパクトに表現するという観点から，プログラム言語の設計にも参考になることが多いと思われる。

なお，現在のLOREL－1の文について欠点と思われるものは，次の通りである。

i） tupleの成分の指定が数字によるものだけで，名前による指定が出来ない。

ii） 文字が，記号1字だけであるので，単なる標識として用いるだけの記号列も，set として構成しなければならない。

iii） 繰返し文の実行を途中でやめるためのexit 機能がない。

iv） カッコが入れ子になることが多く，見にくくなりやすい。

v) associative search を行うための，コンパクトな文が用意されていない。

vi) 論理データが独立してない。（ＰＬ／１方式）

7. implementation上の問題点

ＬＯＲＥＬ－１プロセッサのimplementationは，3.で述べたように，形式化に忠実に従って行われ，これはimplementationを円滑にする上では有用であった。

しかしながら，形式化での階層構造がそのままプログラムの形に導入されているので，サブルーチンのリンケージの回数が非常に多く，これは，実行速度を遅くする大きな原因となっていると考えられる。

又，インタプリタの命令であるｓ－命令の設計にもいくらかの欠点がある。それは，各ｓ－命令が，機能の重複がないように，比較的小さい単位で設計されていたことである。このようにすると，１ソース文当りのｓ－命令の数が増加し，実行時でのｓ－命令格納エリヤを増大させ，又，インタプリタの高レベルの階層で実行される計算の量を増大させ，実行速度を遅くさせることになる。これらを防ぐためには，ある程度の機能をまとめたｓ－命令を設計することが必要である。

また，現在のversion のプロセッサは，最適化ということを考慮せずに作られているが，実用になるシステムを作るには，これについて研究する必要があろう。

プロセッサの実行速度は遅いが（X＝{（１，２），（３，４），（５，６）}の実行に約１７０msec，又，手続の再帰呼出しを利用したfactorialの計算で５！を計算するのに約４０msec を要している。），これは，以上の原因の他に，記述言語がＦＯＲＴＲＡＮであることにも大きく起因するものであろう。

我々は，現在，内容の増強されたＬＯＲＥＬ言語に対して，マイクロプログラムによるプロセッサを構成することを計画中であるが，このために，ＬＯＲＥＬ－１での経験を十分に生かしたいと考えている。

最後に，本研究は文部省科学研究費の援助を受けて行われたものであり，ここに深謝する。

参考文献

1) 片山，日比野，榎本，榎本；
論理関係処理言語ＬＯＲＥＬ，情報処理（採録決定）

2) 片山；
ＬＯＲＥＬデータの言語理論的クラスについて，情報処理Vol １４，№１０，Ｐ１～８，１９７３.

3) 片山，日比野，榎本；
論理関係処理言語ＬＯＲＥＬコンパイラの構成，第１３回プログラミング・シンポジウム
Ｐ２０７～２２３（１９７２）

4) 片山，榎本，日比野，榎本；
ＬＯＲＥＬプロセッサの形式化，電子通信学会オートマトンと言語研究会 ＡＬ７３－４０（１９７３－０９）

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html