論　　文

論理関係処理言語 LOREL-1*

片山　卓也**　日比野　靖***　榎本　進**　榎本　肇**

Abstract

Logical systems such as graphs, automata and formal languages can be described as n-ary relations and LOREL-1 is a high level language for processing such n-ary relations.

LOREL-1 has four data types; integer, character, tuple (fixed length, components may be of mixed data types) and set (variable length, elements must be of the same data type). As set and tuple can be nested recursively, very complex structural data can be expressed in LOREL-1, and flexible statements for manipulating such complex data make LOREL-1 suitable for logical relation processing.

Definition of LOREL-1 data and explanations of the typical statements are given together with some comments.

1. はじめに

計算機利用の多様化，高度化にともない，複雑な構造をもったデータの扱いが重要になってきたが，これらの問題の多くはデータ間の関係の処理に関するものであり，データ構造の処理と呼ばれているものである．これは，通常，複雑な処理であることが多く，また現在の計算機の構成がこの処理に必ずしも適しているとはいえないので，問題向言語の開発が必要である．

論理関係処理言語 LOREL-1 (LOgical RELation processing language) は，このような観点から設計されたものであり，その記述すべき問題には，グラフ，オートマトン，形式言語をはじめとして，一般的には，データ間の論理的な関係を問題とするような論理システムの処理が含まれており，したがって，非常に広い範囲の問題を記述できなければならない．このような問題は，データ構造の処理としてとらえることができるが，この言語の潜在的利用者が多いことを考えると，ポインターの管理を直接に利用者自身が行なわなければならないような primitive な言語では不十分であり，問題向言語にふさわしい形でデータ構造の取り扱いができることが必要である．

一方，データ集合 $S_1, \cdots, S_n$ の中のデータ間の関係 R は，一般には，

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

という形でとらえられ，かつ，このような記法が多くの論理システムで統一的に使用されていることを考えて，LOREL-1 ではこのような形でのデータの処理をその目的とすることに決定した．

すなわち，n-組（以後 tuple という）と集合（以後 set という）を基本的なデータ構造として，それによって n 項関係を表現し，その処理が統一的に記述できるようにその仕様を定めた．

以上が LOREL-1 の（特にデータ構成の）主な特徴であるが，処理ステートメントについては，基本的なもので，かつ，なるべく互に独立なもののみを取入れるという方針に従った．その他，ステートメントの形がコンパクトで見易くなるように心がけたが，これは論理関係の処理のようにアルゴリズムが複雑になることが多い時には重要なことであろう．

2. LOREL-1 のデータの構成

2.1 データの種類

* Logical Relation Processing Language LOREL-1, by Takuya KATAYAMA, Susumu ENOMOTO, Hajime ENOMOTO (Faculty of Engineering, Tokyo Institute of Technology) and Yasushi HIBINO (Musashino Electrical Communication Laboratory, N.T.T.)

** 東京工業大学工学部電子物理工学科

*** 日本電信電話公社武蔵野通信研究所基礎研究部

LOREL-1 で許されるデータは次の 4 つである.

(1) number, (2) character, (3) tuple, (4) set.

以下, これらについて, その概略を述べる.

number は通常の整数であり, 例えば, 123, −2 のように表わされる. number に対しては, 四則演算, 大小の比較が許される.

character は文字であり, 英数字およびいくつかの特殊記号よりなり, number と区別するために, 記号 # を前につけ, #A, #B, #1, のようにかく. character は単一の文字であり, 文字列ではない. LOREL-1 では文字列は character を要素とする set によって表現される.

tuple とは, いわゆる n-組であり, その構成要素を成分という. 成分が $e_1, e_2, \cdots, e_n$ である tuple を次のように表わす.

$(e_1, e_2, \cdots, e_n)$.

1) tuple は固定長の n-組であり, 実行中にその成分の数 n を変化させることはできない.

2) 成分 e_i としては, number, character, tuple, set が許される.

3) tuple T の i 番目の成分を $T(i)$ で表わす.

T: (#A, 1, (1, 2)).
　　$T(1)$　$T(3)$

4) 成分の参照, 書き換えが可能である.

set はデータ型 (後述) の等しいデータから構成される線形リストであり, このデータを set の要素という. 要素 $e_1, e_2, \cdots, e_n$ がこの順に並んだ set を次のように表わす.

$\{e_1, e_2, \cdots, e_n\}$.

1) set 中の要素の数 n は実行中に変えることができる. また, $n=0$ の set { } も空集合として意味を持つ.

2) 要素 $e_1, e_2, \cdots, e_n$ としては, number, character, tuple, set, の全てが許されるが, それらは全て同一のデータ型でなければならない.

3) set X の i 番目の要素を $X[i]$ $(i \geq 1)$ で表わす. また, 空集合を含めて, 全ての set X について 2 つの特別な要素位置 0, ∞ を仮想的に考えるが, $X[0]$, $X[\infty]$ については, 形式的に,

$X[0]=$ {
$X[\infty]=$ }

が常に成立しているものとし, それらの値を書き換えることは許されない. これは, set の始めと終りの判定に利用される.

X: { #A , #P , #1 , #Z }.
　　↑ $X[0]$　↑ $X[1]$　　$X[3]$　　↑ $X[\infty]$ または $X[5]$

4) 要素の書き換え, 取り出し, 追加, 削除, $x \in X$ の形の判定などが許される.

2.2 データとデータ型

LOREL-1 では, set の要素や tuple の成分には再び set や tuple が任意の深さまで入れ子になることが許されており, 複雑な構成のデータが可能であるので, LOREL-1 データの形を明確に規定するには, データ型の概念が必要である.

データ型は, $n, c, (,), \{, \}$ および , から構成される文字列であって, その集合 T はつぎのように帰納的に定義される.

(i) $n, c \in T$,
(ii) $\tau_1, \cdots, \tau_n \in T$ ならば $(\tau_1, \cdots, \tau_n) \in T$,
(iii) $\tau \in T$ ならば $\{\tau\} \in T$.

このとき, **LOREL-1 データの集合** $\mathcal{D}$ は,

$$\mathcal{D} = \bigcup_{\tau \in T} D_\tau.$$

ここで, D_τ は**データ型が τ のデータ集合**であり, つぎ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) $D_{(\tau_1, \cdots, \tau_n)} = \{(d_1, \cdots, d_n) \mid d_1 \in D_{\tau_1}, \cdots, d_n \in D_{\tau_n}\}$,
(iv) $D_{\{\tau\}} = \left\{\{\ \}\right\} \cup \left\{\{d_1, \cdots, d_m\} \mid d_1, \cdots, d_m \in D_\tau, m \geq 1\right\}$.

$d \in D_\tau$ のとき, d のデータ型は τ であるという. また, d のデータ型が一意に定まるとき, それを $\tau[d]$ と表わす. (d が空集合 { } を含んでいなければ, データ型は一意に定まる.)

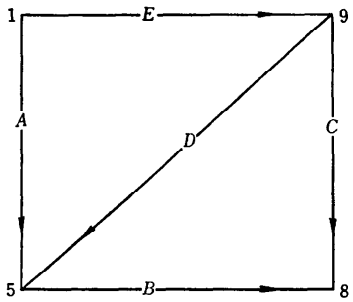
以上の定義からも明らかなように, tuple は, いくつかのデータを単に (,) を用いてまとめたものであり, set は, 等しいデータ型をもつデータの並びを {,} で囲んだものである.

〔例〕
(i) $\{1, 2, 3, 4\} \in \mathcal{D}$, データ型: $\{n\}$,
(ii) $(1, \#A, 2) \in \mathcal{D}$, データ型: (n, c, n),
(iii) $\{(1, \#A, 2), (2, \#B, 4)\} \in \mathcal{D}$,
　　データ型: $\{(n, c, n)\}$,
(iv) $\{1, \#A, 2\} \notin \mathcal{D}$, なぜなら $\tau[1] \neq \tau[\#A]$,
(v) $\{\{1, 2, 3\}, \{10, 20\}, \{\ \}\} \in \mathcal{D}$,

データ型：{{n}}.

〔例〕 グラフの LOREL-1 データによる表現法

Fig. 1 のような有向グラフを LOREL-1 データで表わす方法には種々あるが，代表的な2つの方法を示す．



X: {(1,#A,5),(1,#E,9),(5,#B,8),(9,#D,5),(9,#C,8)}
Y: {(1, {(#E,2),(#A,4)}),(9, {(#C,3),(#D,4)}),(8, { }),
(5, {(#B,3)})}

図 1 有向グラフの LOREL-1 データによる表現
Fig. 1 Representation of a directed graph by LOREL-1 data

ここで，データ X は，枝を，(始点のラベル，枝のラベル，終点のラベル)，の形で表わし，これらからなる set としてグラフを表現している．一方，Y は，各点を，(点のラベル，{(その点から出る枝のラベル，行先の点の Y 中での位置)，…})，によって表わし，このような点の記述の set として，グラフを表現しており，点の接続関係は，set 中での点の位置を利用したポインタで表現している．

2.3 変数と部分構造

すべてのデータは変数を通してアクセスされるが，変数には，名前によって表わされる**基本変数**と，基本変数のあとに**部分構造指定子**の列をつけた**部分変数**とがある．

2.3.1 基本変数

基本変数は，それが表わすデータのデータ型とともに宣言されなければならず，そのデータ型以外のデータを受けることはできない．

基本変数 $v_1, \cdots, v_k$ が，データ型 τ のデータを受けることを

$v_1, \cdots, v_k : \tau$

で表わすが，**データ型の宣言**はこのような表示の並びである．

〔例〕 前節のグラフに対する set X, Y の宣言はつぎのようになる．

X: {(n, c, n)}, Y: {(n, {(c, n)})}.

2.3.2 部分変数

部分変数は，大きな構造の一部分を指すためのものである．LOREL-1 では set, tuple の入れ子が許されるので，複雑なデータを構成できる．このようなデータ X が与えられたとき，set の i 番目の要素であるという要素指定 [i]，tuple の j 番目の成分であるという成分指定 (j) をつぎつぎに繰り返すことによって，基本変数 X から得られるデータを X の部分構造といい，その部分構造を指すのに，それを得るまでに行なった指定を X のあとにその順番につけた部分変数によって行なう．これは配列に対する配列要素に対応する．部分変数は，一般に，つぎの形をとる．

$Xd_1d_2\cdots d_n$.

ここで，X：基本変数，d_i：**要素指定子** [i] または**成分指定子** (j)，である．(ただし，i：算術式または ∞, j：number 定数.)

部分変数 $Xd_1d_2\cdots d_n$ のさすデータのデータ型を $Xd_1\cdots d_n$ のデータ型という．

$$A: \{(\overbrace{1}^{A[1](1)}, \overbrace{\{1,2\}}^{A[1](2)}), (0, \overbrace{\{10,20,30\}}^{A[2](2)[2]})\}.$$

図 2 部分構造と部分変数
Fig. 2 Partial structures and partial variables

また，データ型が $n, c, (\cdots), \{\cdots\}$ である変数をそれぞれ **number 変数**，**character 変数**，**tuple 変数**，**set 変数**という．

2.3.3 リテラル

LOREL-1 における**リテラル**には，具体的な number, character, (,), { , }, , から **2.2** で述べた方法によって構成され，その意味がそれ自身で明らかな定数リテラルと，式を含んでいて，その意味がその式の値に依存する複合リテラルとがある．

定数リテラルは，**2.2** で定義された LOREL-1 データである．なお，character からなる set は，文字列としてよく利用されるので，リテラルの記法上つぎのような便法を用意している．

{#A, #B, #C} ⇒ "ABC".

複合リテラルは，その構成要素として，式を書くことを許したもので，その場所にその式の値を書いたものと同等の意味をもつ．

〔例〕

{X+1, 2, 3}, {$X \cup Y$, {10, 20}, { }}.

3. 式

式はデータと演算子から構成され，算術式，集合式および論理式がある．

（1） **算術式**は，算術素データ（データ型が n のリテラル，変数，関数呼出し，および，set の要素の総数を与える演算子 card を set 式に作用させたもの），四則演算子 +, −, ＊, / および括弧（, ）から通常の規則によって構成され，ALGOL 60 における単純算術式に相当する．算術式のデータ型は n である．

（2） **集合式**は，set 素データ（データ型が $\{\tau\}$, $\tau\in T$ の形のリテラル，変数，関数呼出し），集合演算子 ∪, ∩, −, tail および括弧（, ）から構成される．

各集合演算子の意味はつぎの通りである．ただし，$A=\{a_1, a_2, \cdots, a_n\}$, $B=\{b_1, \cdots, b_m\}$ とする．

(i) $A\cup B=\{a_1, \cdots, a_n, b_1, \cdots, b_m\}$.

(ii) $A\cap B=\{a_{i_1}, \cdots, a_{i_k}\}$, ここで，(イ) $1\leq i_1<\cdots<i_k\leq n$ かつ，(ロ) $x\in A$ and $x\in B\Leftrightarrow x=a_{i_j}$ for some $j(1\leq j\leq k)$.

(iii) $A-B=\{a_{i_1}, \cdots, a_{i_k}\}$, ここで，(イ) $1\leq i_1<\cdots<i_k\leq n$ かつ，(ロ) $x\in A$ and $x\notin B\Leftrightarrow x=a_{i_j}$ for some $j(1\leq j\leq k)$.

(iv) tail $A=\{a_2, \cdots, a_n\}$.

演算順位は，括弧 ＞tail＞∩＞∪, − の順であり，同じ強さのものについては，左→右である．また，2項演算子 ∪, ∩, − については，そのオペランドのデータ型は同一でなければならず，また，set 式のデータ型は，それを構成している素データのデータ型に等しい．

LOREL-1 における set は，その要素が一列に並んだ線形リストであり，したがって，演算もそれに合わせて定義されている．（6節 “仕様の検討” 参照.）

〔例〕

$A=\{1, 2, 3\}$, $B=\{2, 1\}$ のとき，

$A\cup B=\{1, 2, 3, 2, 1\}$, $B\cup A=\{2, 1, 1, 2, 3\}$,

$A\cap B=\{1, 2\}$, $B\cap A=\{2, 1\}$,

$B\cup B=\{2, 1, 2, 1\}$ であり，

一般には，$A\cup B\neq B\cup A$, $A\cap B\neq B\cap A$,

$A\neq A\cup A$ である．

（3） **論理式**は，各種の関係式，∨(or)，∧(and)，¬(not)，括弧（, ）から通常の規則によって構成され，ALGOL 60 の単純論理式に相当する．関係式には，算術関係式，構造比較式，要素関係式および包含関係式がある．

算術関係式：$e_1\rho e_2$，ここで，e_1, e_2：算術式，ρ：$<, \leq, =, \geq, >, \neq$ であり，算術式の値の大小関係を表わす．

構造比較式：$e_1=$(または $\neq$)e_2，ここで，e_1, e_2：set 式または tuple, character 素データ（データ型が，(τ) または c のリテラル，変数，関数呼出し）．これはデータ e_1 がデータ e_2 に等しい（等しくない）時のみ true となる．これによって，任意の character, tuple, set データの同一性の判定を行なうことができる．e_1, e_2 のデータ型が等しくなければならない．

〔例〕

$X\cup\{\#A, \#B\}=Y$, $T=(1, \{\ \}, 2)$.

要素関係式：$e_1\in$(または $\notin$)e_2，ここで e_1：式または tuple, character 素データ，e_2：set 式であり，e_1 が e_2 に属する（属さない）時のみ true である．

e_1, e_2 のデータ型について，$\{\tau[e_1]\}=\tau[e_2]$ が成立していなければならない．

〔例〕

$A\in B$, $X+1\in N\cap\{1, 2, 3, 4\}$.

包含関係式：$e_1\subset$(または $\not\subset$)e_2，ここで e_1, e_2：set 式であり，set e_1 が set e_2 に含まれる（含まれない）時のみ true である．

e_1, e_2 のデータ型は同一でなければならない．

〔例〕

$A\subset B$, $A\cap B\subset X\cup\{1, 2, 3\}$.

〔例〕

つぎのものは論理式である．

$(X=1\vee 10\in S)\wedge Y\in W$,

$\neg T[1]=(1, 2, \{\ \})$.

（4） 現在の LOREL-1 の仕様では，簡単化のために，算術式と論理式は概念的に厳密には区別されておらず，論理式は，その値が true, false に応じて，1, 0 なる整数値をとる算術式と見なされる．また，条件文の条件部に算術式を書くことが許されるが，これは，その値が，not 0, 0 に応じて，true, false と解釈されている．このような意味で，以後，算術式と論理式を合わせて，**算術論理式**と呼ぶ．

4. LOREL-1 プログラムの構成

4.1 プログラム構成の概略

(i) プログラムは，ひとつの**主プログラム**と，いくつかの**副プログラム**から構成されるが，いずれも手続きであり，これは，文を区切り記号；で区切って並べたものである．一連の手続きのうち，最初の手続き

が主プログラムとなる.

(ii) **手続きは**

D_0; D_1; ⋯; D_n; S_1; ⋯; S_m; end

の形をしている. ここで D_0 は**手続きの宣言**であり,

define (X: $p_1, \cdots, p_k$),

X: 手続き名, p_i: 仮引数

の形をしている. $D_1, \cdots, D_n$ は基本変数の**データ型の宣言**で, すべての基本変数はここでデータ型の宣言が行なわれていなければならない. その形式については, **2.3** を参照されたい. $S_1, \cdots, S_m$ は宣言以外の文である.

手続きの再帰呼出しが許される.

(iii) 文 S_i としては, 無条件文, 条件文, 繰返し文がある. いずれの文にも名札をつけることができ, goto 文の行先とすることができる. 名札 l のついた文 S は, $l: S$ と表わされる.

(iv) **無条件文**には, 代入文, 手続き呼出し文, 入出力文, 制御文, 注釈文, 空文などの基本文と, 文の列を括弧 [,] で囲んだ複合文がある. 複合文には, 変数名の scope の概念はない.

4.2 代入文

LOREL-1 におけるデータの処理は, ほとんどが代入文の形を通して行なわれる. 代入文によって行なわれる処理は, (1)通常のデータの書換え, (2) set への要素の追加, (3) set からの要素の削除である. これらのうち, (1)は**基本代入文**, (2)は**挿入代入文**, (3)は**要素削除代入文**によって行なわれる.

4.2.1 基本代入文

$L=R$,

ただし, L: 変数, R: 式または tuple, character 素データ, かつ $\tau[L]=\tau[R]$.

(i) 右辺 R を評価し, それを左辺の変数 L に与える. 左辺と右辺のデータ型さえ一致していれば, どのような複雑なデータの書きかえも可能である. もちろん右辺の変数の表わしているデータは影響を受けない. また, 代入前に左辺が表わしていたデータは代入後無効となり, これに再びアクセスすることはできない.

(ii) 右辺が, 論理式の時には, その値が true, false に応じて, 1, 0 をとる算術式として扱われる.

〔例〕

N: 5, T: (1, #P, 3),

X: {(1, #A, 2), (2, #B, 3)},

U: {{10, 20}, { }, {0, 5}} とするとき,

(a) $T(1)=T(3)+N \Rightarrow T$: (8, #$P$, 3),

(b) $X[1]=T \Rightarrow X$: {(1, #P, 3), (2, #B, 3)},

(c) $U[2]=\{N\} \Rightarrow U$: {{10, 20}, {5}, {0, 5}},

(d) $X=U$ は $\tau[X] \neq \tau[U]$ だから許されない.

4.2.2 挿入代入文

$L\downarrow=R$ または $\downarrow L=R$.

ここで, L は set の要素を表わす変数であり, $X[i]$ (X: set 変数, i: 算術式) の形をしているか, または, 繰返し条件 (後述) で指定された, set の要素を指す基本変数である. R は式または, tuple, character 素データである.

(i) この代入文は, set 中の指定された要素 L の右または左隣りに新しいデータ R を挿入するためのものであり, $L\downarrow$ は L の右隣りに, $\downarrow L$ は L の左隣りに挿入することを表わす.

いずれも, L の右(左)隣りの仮想的な場所 $L\downarrow(\downarrow L)$ に R を代入するという形をとっている.

(ii) 特に, $X[0]\downarrow=R$ によって, set X の先頭に, また, $\downarrow X[\infty]=R$ によって, set X の末尾にデータを挿入することができる.

(iii) データ型に関して, $\tau[L]=\tau[R]$ でなければならない.

〔例〕

X: {1, 2, 3, 4, 5} のとき,

$X[3]\downarrow=0 \Rightarrow X$: {1, 2, 3, 0, 4, 5},

$X[0]\downarrow=0 \Rightarrow X$: {0, 1, 2, 3, 4, 5},

$\downarrow X[\infty]=0 \Rightarrow X$: {1, 2, 3, 4, 5, 0}.

4.2.3 要素削除代入文

$L=\varepsilon$.

ここで, ε は**削除記号**, L は set の要素を表わす変数である.

set からその要素 L を取去ることを意味するが, これは削除したい要素に特殊記号 ε を代入するという形式をとっている.

〔例〕

X: {1, 2, 3, 4, 5} のとき,

$X[1]=\varepsilon \Rightarrow X$= {2, 3, 4, 5}.

4.3 その他の基本文

以上の他の基本文としては, 入出力文 (定数リテラルの形の入出力), 手続き呼出し文 (形式は, 関数呼出しとともに, ALGOL 60 と同じ), goto 文, return 文, stop 文, 注釈文, 空文があるが, これらについては省略する.

4.4 複合文

実行文の列を括弧 [,] で囲んだものであり，全体が1つの文として扱われ，条件文，繰返し文の中で有効に用いられる．

$[S_1; S_2; \cdots; S_n]$, ただし S_i: 文.

複合文には，変数名の scope 概念はない．

〔例〕

$[X=1; Y[2]=\{1,2,5\}; \text{goto}(L)]$.

4.5 条件文

$[B_1 \Rightarrow S_1, B_2 \Rightarrow S_2, \cdots, B_n \Rightarrow S_n]$.

ここで，B_i: 算術論理式，S_i: 文である．B_n はなくともよい．$B_1, B_2, \cdots$ の順に調べてゆき，初めて true (または not 0) となった B_i に対応する S_i が実行される．もし，そのような B_i が存在しなければ，何も実行されない．また，B_n が存在しない時には，$B_1, \cdots, B_{n-1}$ がすべて false (または 0) なら，S_n が実行される．

〔例〕

$N=0$;
LOOP: $[X[N+1] \neq X[\infty]$
$\Rightarrow [N=N+1;$ goto (LOOP)$]]$.

上のステートメントによって，set X の要素の数 N が数えられる．

4.6 繰返し文

LOREL-1 における繰返し文には，通常の繰返し文の他に，set の要素を取り出す形の繰返し文があるが，これは set の処理上重要である．繰返し文は一般に次の形をとる．

$(\gamma)S$.

ここで，γ: 繰返し条件，S: 名札のつかない文，である．**繰返し条件** γ には，つぎの2つの形が許される．

(i) $\xi=\alpha, \beta$, (ii) $\xi \in \Omega$.

(i) は通常の do 文での繰返し条件であり，ξ: 型が n の基本変数，α: 算術式，β: 算術式または ∞, である．

$(\xi=\alpha, \beta)S$ の実行は，$\beta \neq \infty$ ならば，まず始めに α, β の値が評価され，$\alpha \leqq \beta$ ならば，$\xi=\alpha, \alpha+1, \cdots, \beta$ として，文 S が繰返し実行される．また，$\alpha > \beta$ なら何も実行されない．$\beta=\infty$ の時には，文 S が $\xi=\alpha, \alpha+1, \cdots$ と繰返され，繰返しの停止は，繰返し条件では与えられないので，文 S によって与えなければならない．

〔例〕

$(I=1, 5*N)$ $P=P+I$; .

(ii) は，set の要素をひとつずつとり出す形の繰返し条件であり，ξ: 基本変数，Ω: set 式，であり，$\tau[\Omega] = \{\tau[\xi]\}$ でなければならない．繰返し文 $(\xi \in \Omega)S$ は，$\Omega = \{\ \}$ なら空文として扱われる．$\Omega \neq \{\ \}$ の時には，Ω の要素を始めのものからひとつずつとり出し，それを変数 ξ に与えて S を実行し，これを Ω の最後の要素についてまで実行することを意味する．文 S の中で，set Ω を変化させることが許される．

〔例〕

$(L \in Z)[L=1 \Rightarrow L\downarrow=0]$.

この文は，number よりなる set Z に，1 があれば，その右隣りに 0 を挿入する．

〔例〕

文 S の中で，set Ω を変化させることを上手に利用すると，巧妙なプログラムが作れることがある．

$Y=\{1\}$;
$(I \in Y)[(T \in R)[T(1)=I \wedge T(2) \notin Y$
$\Rightarrow \downarrow Y[\infty]=T(2)]]$; .

これによって，整数上の関係 ρ が，データ型が $\{(n, n)\}$ のデータ R によって，$x \rho y \Leftrightarrow (x, y) \in R$, なる形で与えられているとき，整数1から ρ を何回か辿ることによって得られる整数のすべてからなる集合 Y が求められる．

さて，繰返し文 $(\gamma)S$ において，S は再びラベルのつかない繰返し文でもよいから，一般に繰返し文はつぎの形になる．

$(\gamma_1) \cdots (\gamma_n)S$.

ここで，$\gamma_1, \cdots, \gamma_n$ は繰返し条件である．この時は，もちろん，最も内側の繰返し条件から先に変化する．

〔例〕

$N=0$; $(S \in X)(T \in S)[T=\sharp A \Rightarrow N=N+1]$.

これによって，文字列からなる set X について，文字 A の総数 N を求めることができる．

5. LOREL-1 プログラムの例

ここでは，LOREL-1 プログラムの例として，文脈自由文法の構文解析のための手続き ANALYZE (X, S, P) を示す．この手続きの対象とする文法 $G=(V_N, V_T, P, \sigma)$ は Greibach 標準形をしており，終端記号，非終端記号は整数にコード化されているとする．また，i 番目のルール $i: A \to aB_1 \cdots B_k$ は，tuple $(i, A, a, \{B_1, \cdots, B_k\})$ という形で表現され，したがってルール集合 P のデータ型は $\{(n, n, n, \{n\})\}$ である．X は入力記号列，S は初期非終端記号列であり，データ型は $\{n\}$ である．構文解析の結果は，$S \overset{\pi}{\Rightarrow} X$ (最左

導出）となるようなルール番号の列 π の集合として ANALYZE に渡される．

```
define (ANALYZE: X,S,P);
P: {(n,n,n,{n})}, R: (n,n,n,{n}),
ANALYZE, Z: {{n}}, X,S,Q: {n};
[X={ }⇨[[S≒{ }⇨ANALYZE={ }, ⇨ANALYZE
  ={{ }}]; return],⇨[S={ }⇨[ANALYZE={ };
   return]]]; Z={ };
(R∈P)[R(2)=S[1]∧R(3)=X[1]
  ⇨(Q∈ANALYZE(tail X,R(4)∪tail S,P))
   [Z={Q}∪Z; Z[1]={R(1)}∪Z[1]]];
ANALYZE=Z; return;
end.
```

図 3 LOREL プログラムの例
Fig. 3 An example of LOREL program (a syntax analysis procedure for CFG of Greibach normal form)

6. 仕様の検討

ここでは，LOREL-1 言語，特に，データ構成について，その特徴，問題点および今後の方向について，簡単に述べる．

（1） set, tuple の区別

set, tuple によるデータの記述が，論理システムの記述上極めて一般的であることは既に述べたが，set, tuple は，ともにデータの並びであり，これをプログラム仕様上区別すべきか否かは，必ずしも自明ではないが，つぎのような理由から，この区別は正当なものと考えられる．

i）関係を集合の直積の形で記述する場合，集合は，通常，同じような性質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，すべての要素に対して uniform なものになるのが普通であろう．一方，関係の要素である n-組は，性質の異なるデータを単に grouping したものであり，それに対しては，成分を指定する以外の操作を考える必要はない．

このような仮定のもとに，set と tuple が導入され，それに対する操作が決定されたが，これは，概念上極めて自然なものであることが，LOREL-1 の使用経験からも確かめられている．

ii）このように，必要な範囲内でデータの形を制限することは，プログラムのエラー・チェックの点で有益である．すなわち，LOREL-1 では，すべての基本変数には，そのデータ型が宣言されていなければならず，また，各文では，それが満たされなければならないデータ型に関する条件が規定されているので，ソース文のデータ型チェックを行なうことによって，データ操作に関するプログラム・ミスの検出に役立てることができる．

iii）さらに，implementation の立場からは，set, tuple の区別は，記憶容量の点で有利である．すなわち，set は可変長であるので，linked memory を実現するのが最も適当であるが，tuple は固定長であるので，連続した場所に allocate することが可能である．

（2） set の整列性

LOREL-1 における set は，実質的には線形リストであり，その要素は一列に並べられており，また，同じデータが重複して含まれることが許されており，通常の“集合”とは異なっている．set をこのような形にした理由は，多くの問題で，この要素の整列性が有効に利用でき，かつ，計算機内部でも，この方式の方が自然に，かつ，効率的に処理することができるからである．例えば，文字を要素とする set によって，文字列を表現したり，要素の set 中での位置を利用して，plex 構造を容易に実現することができる．もちろん，通常の集合としての処理が要求される場合も考えられるが，その場合には，整列性を無視するような手続きを書くことによって，そのような処理を行なうことが可能である．なお，集合を扱うことのできる言語として，STDT[7]，SETL[6] などがあるが，そこでは，通常の集合が扱われている．

（3） reference データの欠除

LOREL-1 データで直接記述することができるのは，宣言によって定められた形での，set, tuple の入れ子構造であり，一般のグラフのような構造は，例えば，2.2 の Fig. 1 のような方法によって表現しなければならない．このとき，Xのような表現は，簡単ではあるが，構造を辿る操作を，set X 中の探索によって実現しなければならないので，Yのように，set 中での位置を利用した plex 構造で表現する方が複雑な処理には向いている．

さらに，アドレスを直接利用する reference データを導入すれば，この辿る操作は，より高速化されるが，これには，いくつかの問題があり，LOREL-1 からは除かれている．まず，reference データを含んだ場合に，コンパイル時データ型チェックを完全に行なうことは非常に困難であり，また，これを実行時に行なうと，実行速度の点で望ましくない．事実，LOREL-1 でも set 中の場所に対して reference データが導入され，データ型チェックを実行時に行なうようなインタプリタが作製されたが，その実行速度は現在のものの

半分以下であった．さらに，構造のコピー，デバッグのための出力などを考えると，Yのように数値を使ったポインタの方が便利である．

また，初期の仕様[1~4]では，組成集合，インデックスという概念を導入して，transitive な探索を高速化し，Xの形の表現での，辿る操作の高速化が計画されたが，implementation が複雑で，かつ，組成集合に implicit な変化が起こるので，現在の仕様からは除かれた．

(4) 問題点と今後の方向

i) LOREL-1 で記述する内容は，主に，set の処理に関するものであるが，現在これに対して用意されている機能は，かなり primitive なものである．これは，機能の重複が余りないように文のレベルを設定したためであるが，見易さという点では必ずしも十分ではなく，今後，高度な数学的な記述を導入したいと考えている．

ii) 木構造は LOREL-1 データでは直接記述することはできず，Fig. 1 のYと同様に，set 中での位置を利用した plex 構造によって表現しなければならない．これは，木構造は直接表現できるが，線形リストが表現しにくい LISP[5] と対照的である．さて，この場合でも，適当な手続きを用意すれば，その処理に問題がないことは事実であるが，木構造が直接記述できれば，その方が望ましい．set, tuple と調和した形での導入方法が明らかでなかったので，LOREL-1 には木構造の直接的表現が含まれていないが，現時点では，set, tuple の拡張として十分に満足のゆく解決法が得られている[10]．また，LOREL-1 では，データ型に関する規定が厳し過ぎて，データ型と独立な処理の記述が困難であるが，これについても，木構造の導入法と関連した解決法が考えられている[10]．

iii) その他に，LOREL-1 には，細かい点で改良すべきことがいくつかあるが，以上述べた問題点とともに，現在行なっている LOREL-2 の設計に十分反映させたいと考えている．

7. インプリメンテーション

LOREL-1 プロセッサは，複雑な構造の動的な変化を処理しなければならず，その構成も簡単ではないが，データ構成の言語理論的考察[8]，記憶構造の抽象化に基づくプロセッサ動作の machine independent な記述によって[9]，その implementation は，割合簡単に行なうことができた．

プロセッサは，ソース文の構文解析，データ型チェック，コード発生を行なうコンパイラと，コンパイラの発生したコードにしたがって，実行時にデータ構造の処理を行なうインタプリタから構成されている．インタプリタは，各種の部分プロセッサに分解され，それぞれ仮想的 machine (オートマトン) の形に形式化されている．

実際の implementation は，NEAC 3200 (モデル 50) 上で FORTRAN によって行なわれた．現在のところ実行速度は遅く，例えば，文

$X=\{(1, 2), (30, 40), (5, 6)\}$

の実行 (右辺をリテラル・プールから読出し，それに対応するセル構造(線形リストの入れ子構造) を作り，これをXにわたす) に，約 170 msec を要している．この原因としては，(i)記述言語が FORTRAN であること，(ii)プロセッサをその形式化にしたがって，多レベルに分解したために，サブルーチンのリンケージが非常に多いことが考えられるが，現在マイクロ・プログラムによるプロセッサの構成を計画中であり，これによってこれらの原因は除けるものと考えている．

プロセッサの形式化，その implementation の詳細については，予定している続報 "LOREL プロセッサの形式的構成" にゆずりたい．

謝辞

LOREL-1 の初期の仕様の検討に参加された香取和之氏 (現三菱電機)，小林哲郎氏 (本学大学院)，およびプロセッサ構成について検討された松沢収氏 (現日立製作所) をはじめとして研究室の皆様に感謝します．また，有益な助言をいただいた京都大学，堂下修司教授に感謝します．

参考文献

1) 片山，日比野，榎本：論理関係処理言語 LOREL のコンパイラの構成，第 13 回プログラミング・シンポジウム報告集，pp. 207～223 (1972)．
2) 榎本，片山，日比野：論理関係処理言語 LOREL のコンパイラの構成，情報処理学会第 12 回大会(1971)．
3) 榎本，片山，日比野：論理関係処理言語 LOREL のデータ構成の形式化，情報処理学会第 12 回大会 (1971)．
4) 榎本，堂下，片山，香取，日比野：論理関係処理言語についての一提案，電子通信学会オートマトン研資，A 70-70 (1970)．

5) J. McCarthy et al.: LISP 1.5 Programmers Manual, MIT Press (1962).
6) M. C. Harrison: Data Structure and Programming, p. 243, Scott Foresman and Company (1973).
7) D. L. Childs: Description of a set theoretic data structure, FJCC, pp. 557～564 (1968).
8) 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, No. 10, pp. 754～761 (1973).
9) 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資，AL 73-40 (1973-09).
10) 片山，榎本，榎本：LOREL 言語の使用経験とその検討，第 15 回プログラミング・シンポジウム報告集 (1974).

(昭和 48 年 9 月 22 日受付)
(昭和 48 年 11 月 19 日再受付)