Vol. 19 No. 6　　情　報　処　理　　June 1978

論　文

Lorel-2 言語について*

榎本　進** 宮地利雄***
片山卓也*** 榎本　肇***

Abstract

A very high level programming language, Lorel-2, is designed to describe combinatorial systems such as graphs and languages. Lorel-2 has 7 data types: *integer*, *real*, *boolean*, *string*, *procedure*, *tuple* (fixed length list of mixed data type components) and *set* (variable length list of same data type elements). Storage attributes (hash, file and virtual) may be declared for Lorel-2 data.

Compact notations to denote complex data objects are introduced, such as; *Σ-expressions* (generalized form of summation), *set formers* (sets satisfying given conditions) and *associative search expressions*.

Lorel-2 has strong iterative and control statements *exit*, *entrance* and *cycle* to allow the writing of concise and well-formed programs without *goto*.

1. はじめに

Lorel は，グラフ，形式言語等の組合せシステムの簡潔な記述を目指して研究，開発された高級言語である．我々は，先に Lorel-1[1)-4)] を発表し，種々の使用経験を積んだ結果，さらに豊富なデータ構成と強力な機能を備えた Lorel-2[5)-8)] を開発したが，その特長は次の通りである．Lorel-2 のデータには，複合データとして，set および tuple があり，一般に，データ間の関係はすべて n 組から成る集合として表せるため，このようなデータは有効である．また，グラフ等の問題では，木が重要であるため，再帰的な tuple である tree を用いることにより，n 分岐木を直接的に表現することができる．Lorel-2 は，この他にも様々のデータをもち，それらを組合せることにより，複雑なデータを構成することができるが，その構成を明確にするためにデータ型が用意されている．データ型の存在は，それをもたない高級言語[9)] に比べて，実行効率の面からも有利な結果をもたらしている．Lorel-2 データには記憶域属性とよばれるものを指定することにより，そのデータを種々の記憶領域にとること，また，set や tree データをハッシュ化することができる．

代表的な機能としては，ある条件を満たす set をつくる set former，数学で用いられる summation の一般形である Σ 式，および，連想検索式がある．Lorel-2 では，強力なくり返し指定が行え，制御文として，exit 文，entrance 文および cycle 文が用意してあるため，goto 文を使うことなしにプログラムを簡明に書くことができる．

2. Lorel-2 データの構成

2.1 データの種類

Lorel-2 で許されるデータは，次の 7 種類である．

(1) integer (2) real (3) boolean (4) string
(5) procedure (6) tuple (7) set

(1) **integer** は通常の整数のことである．

(2) **real** は通常の浮動小数点実数のことである．

(3) **boolean** は **true** または **false** なる論理値のことである．

* On the Programming Language Lorel-2 by Susumu ENOMOTO (Faculty of Science and Engineering, Science University of Tokyo), Toshio MIYACHI, Takuya KATAYAMA, and Hajime ENOMOTO (Faculty of Engineering, Tokyo Institute of Technology).

** 東京理科大学理工学部情報科学科
*** 東京工業大学工学部情報工学科

(4)**string** は8文字以内の文字列で 'ABC' のように引用符 ' で囲んで表す.

(5)**procedure** は手続き名であり, Lorel-2 の手続きの他に Cobol, Fortran および Hpl のものが許される.

(6)**tuple** は固定長の線型リストであり, その構成要素を**成分**とよぶが, 成分としては, 任意の Lorel-2 データが許され, 各成分が同一のデータ型 (後述) でなくてもよい.

- 成分が $e_1, \cdots, e_n$ の tuple を次のように書く.

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

- tuple T の i 番目の成分の指定には, $T(i)$ または $T\cdot\iota$ という2通りの方法がある. ここで, τ は, T が vector なら integer 式, vector 以外の tuple なら integer 定数であり, また, ι は成分識別子であって, Tのデータ型の宣言において, i 番目の成分にはられた名前である (**2.2 参照**).

- tuple は, 第2成分以下に特殊原子記号 **nil** を含ませることで, 木構造を表すことができ, これを **tree** とよぶ. また, **nil** 自身も空 tree として意味を持つ.

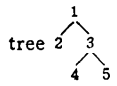
- 各成分が同一のデータ型の tuple を **vector** とよぶ.

- tree に対して, 連想検索を行うことができる.

[例 2.1] ('PAI', 3.14) string と real から成る tuple

(1, 2, 3, 4, 5) 成分が integer の vector

(1, (2, **nil**, **nil**), (3, (4, **nil**, **nil**), (5, **nil**, **nil**)))



(7)**set** は可変長の線型リストであり, その構成要素を**要素**とよび, 要素はすべて同じデータ型をもたなければならない.

- 要素が $e_1, \cdots, e_n$ の set を次のように書く.

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

- 要素の参照, 書き換えの他に要素の追加, 削除によって要素の数 n を実行中に変えることもでき, また, $n=0$ の set も空集合 { } として意味を持つ.

- set X の i 番目の要素を $X[i]$, 後から i 番目の要素を $X[\infty-i]$ で指定することができ, i は1以上の数を表す integer 式である. また, 特別な要素位置 $0, \infty$ によって, $X[0]$ で X の左括弧 {, $X[\infty]$ で X の右括弧を表し, 代入文で, 有効に使われる.

- set 間の等価や包含を確める関係式がある.
- set に対して, 連想検索を行うことができる.
- subset の指定が行える.
- set の要素を取り出す形のくり返し指定子がある.

[例 2.2]

{1, 2, 3} {sin, cos} {(1, 'ONE'), (2, 'TWO')}

2.2 データ型

Lorel-2 では, 前述したデータを組合せることによって, 複雑なデータを構成することができるが, その構成を明確に表すために**データ型**が用意されている.

データ型の集合 T はアルファベット

$\Sigma = \{i, r, b, s, (,), \{, \}, [,], ', \rightarrow, *, \times,$
$:$, **element**, **subtree**, **Fortran**, **Cobol**,
Hpl$\} \cup N \cup V \cup I \cup P$

上の言語である. ただし, N: 自然数の集合, V: 部分変数の集合, I: 成分識別子の集合, P: 引数識別子の集合

T は, 次の帰納的な定義を満たす最小の集合である. ただし, $T' \equiv T - \{$**element** d, **subtree** $\alpha \mid d \in V\}$ とする.

(1) $i, r, b, s \in T$

(2) $t_1, \cdots, t_k, t \in T$, $L \in \{$**Fortran**, **Cobol**, **Hpl**$\}$ ならば,

$$[t_1, \cdots, t_k] \in T \qquad [t_1, \cdots, t_k] \rightarrow t \in T$$
$$L[t_1, \cdots, t_k] \in T \qquad L[t_1, \cdots, t_k] \rightarrow t \in T$$

引数中に, 他の引数上の element/subtree 変数を含む場合は, 引数識別子 $p \in P$ を用いて,

$$\left[\!\left[\cdots, p: t_i, \cdots, \begin{Bmatrix} \textbf{element } P \\ \textbf{subtree } p \end{Bmatrix}, \cdots \right]\!\right] \text{ のように書く.}$$

(3) $t_1, \cdots, t_k, t \in T'$, $N \in N$ ならば,

$$(t_1, \cdots, t_k) \in T \qquad (t'_1\ *, \cdots, *) \in T$$

tuple では, 成分のセレクタである成分識別子をデータ型で宣言するため, 成分識別子 $\iota_1, \cdots, \iota_k \in I$ を用いた

$$(\iota_1 : t_1, \cdots, \iota_k : t_k) \in T$$
$$(\iota_1 : t, \iota_2 : *, \cdots, \iota_k : *) \in T$$

なるデータ型も用意されており, 成分識別子を除いたものと同等と見なされる.

また, 同一のデータ型の成分が連続する

$(\cdots, \underbrace{t, \cdots, t}_{N\text{個}}, \cdots)$ の場合, ×を用いて

$(\cdots, t \times N, \cdots)$ と表現してもよい.

従って, 成分のデータ型 t, 成分数 N の vector の

データ型は $(t\times N)$ と書くこともできる.

(4) $t\in T'$ ならば,

$$\{t\}\in T$$

(5) $d\in V$ かつ d は set または tree を表すデータならば,

element $d\in T$ または **subtree** $d\in T$

このとき, Lorel-2 データの集合 D は, データ型を用いて, 次のように定義される.

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

D_t は, データ型が t であるデータの集合で, 次の帰納的定義を満たす最小の集合である.

(1) D_i=integer データの集合

(2) D_r=real データの集合

(3) D_b=boolean データの集合

(4) D_s=string データの集合

(5) $D_{[t_1,\cdots,t_k]}$=引数のデータ型が $t_1,\cdots,t_k$ である手続き名の集合

$D_{[t_1,\cdots,t_k]\to t}$=引数のデータ型が $t_1,\cdots,t_k$, 値のデータ型が t である関数名の集合

$D_{L[t_1,\cdots,t_k]}$, $D_{L[t_1,\cdots,t_k]\to t}$ も同様だが, 各々, 他言語 L の手続き名, 関数名の集合である.

(6) i) $t_1,\cdots,t_k\neq *$ のとき

$D_{(t_1,\cdots,t_k)}=\{(d_1,\cdots,d_k)\mid d_1\in D_1,\cdots,d_k\in D_k\}$

すなわち, 成分のデータ型が, 各々, $t_1,\cdots,t_k$ である tuple の集合

ii) $D_{(t,*,\cdots,*)}=\{(d_1,\cdots,d_k)\mid d_1\in D_t, d_2,\cdots,d_k\in D_{(t,*,\cdots,*)}\}\cup\{\textbf{nil}\}$

すなわち, node のデータ型が t の $(k-1)$ary tree の集合

(7) i) $D_{\textbf{element }d}$=set データ d の要素の集合

ii) $D_{\textbf{subtree }d}$=tree データ d の部分木の集合

[例 2.3] $(\cos, 0.5, \text{'PAI'})\in D_{([r]\to r,r,s)}$

$\{1,2,3,4,5\}\in D_{[i]}$

$(1, (2, (3, \textbf{nil}, \textbf{nil}), (4, \textbf{nil}, \textbf{nil})), (5, \textbf{nil}, \textbf{nil}))\in D_{(i,*,*)}$

2.3 変数と部分構造

変数には, 名前によって表される**基本変数**と, その後に**部分構造指定子**の列を付けた部分変数がある.

(1) 基本変数は, それが受けるデータ型とともに宣言されなければならない (**4.2** 参照).

基本変数の中には, 他の set 変数の要素や tree 変数の部分木を表す **element/subtree 変数**がある.

このように, 複雑な構造を持つデータの一部分を表すものをそのデータの**部分構造**とよぶ.

(2) 部分変数は, 部分構造を表すためのものであり, 基本変数 v が与えられたとき, その後に次の部分構造指定子をくり返し付けることによって, その部分構造を指定できる.

(i) 各々, set の i 番目の要素, 後から i 番目の要素, set 括弧 {, または set 括弧} を指定する

要素指定子 $[i]$, $[\infty-i]$, $[0]$, または $[\infty]$

(ii) tuple の j 番目の成分に成分識別子 ι_j が付けられているとき, その成分を指定する

インデクスによる成分指定子 (j)

成分識別子による成分指定子 $\cdot\iota_j$

(iii) set 要素の n 個右隣りの要素を指定する

要素移動指定子 $\mathfrak{z}n$

(iv) set の k 番目から l 番目までの要素から成る subset を指定する

subset 指定子 $[k,l]$

[例 2.4] element/subtree 変数は, 次のように図示する.

$$\underset{B:}{A: \{1, \underline{2}, 3\}}$$

B は set A 上の element 変数であり, 2を表し, 部分変数 $A[2]$ と全く同じようにふるまう.

$$\underset{S:}{T: (1, \underline{(2, (3, \textbf{nil}, \textbf{nil}), (4, \textbf{nil}, \textbf{nil}))}, (5, \textbf{nil}, \textbf{nil}))}$$

S は tree T 上の subtree 変数であり, $\overset{2}{\underset{3}{\wedge}}\!\!\underset{4}{}$ を表し, 部分変数 $T(2)$ と全く同じようにふるまう.

[例 2.5]

変数 A のデータ型が, $\{(\text{first}: s, \text{second}: i)\}$ のとき,

$A[1]$ または $B\mathfrak{z}(-1)$ $\quad$ $A[2,3]$

$$\underset{B:}{A: \{(\overbrace{\text{'ONE'}, \underbrace{1}}), (\overbrace{\text{'TWO'}, 2}), (\overbrace{\text{'THREE'}, 3})\}}$$

$A[1](2)$ または $A[1].\text{second}$

(3) **リテラル**は, 具体的な integer, real, boolean, string, procedure, (,), {, }, [,], ×, **nil** および・から構成され, 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, その意味が式の値に依存する**複合リテラル**がある.

[例 2.6] 3 1.0 'Lorel' $\{1,3,5\}$……定数リテラル $\{1, N+5, 3\}$

$(1, (2, \text{TREE}, \textbf{nil}), \textbf{nil}))$……複合リテラル

2.4 変数の記憶域属性

変数には, 宣言文において, normal, virtual, file, record, global および hash という**記憶域属性**を指定することができる (**4.2** 参照).

(1) **normal 属性**は変数に対し, virtual, file,

record または global 属性を指定しないとき，自動的に与えられる．normal 属性をもつ変数は，**常駐型記憶域**とよばれる常駐度の高いセグメント上にとられ，高速にアクセスすることができるが，この領域は余り大きく設計されていない（約 4000 セル）．

（2） **virtual 属性**は，variable 宣言文において，変数に対して，**virtual** を指定することにより与えられる．virtual 属性をもつ変数は，**非常駐型記憶域**とよばれる常駐度が通常であるセグメント上にとられる．この領域は比較的大きく設計されている（約 60000 セル）．

（3） **file 属性**は，変数を file 宣言文で宣言することにより与えられる．file 属性をもつ変数（file 変数）は，**外部記憶域**（ディスク）上に，ACOS-4 の提供する VSAS file としてとられるため，その編成も，順編成，相対編成または乱編成のものを指定することができる．file 変数は，他の属性をもつ変数と異なって，通常の代入文によるアクセスを禁止されており，record 変数と file 制御文によってアクセスされる（**4.5** 参照）．file 変数のアクセスは，その編成の仕方により異なる．順編成のものは，file の先頭からのシーケンシャルなアクセスに適し，相対編成のものは，file の先頭からの record 数を指定することによりアクセスできる．乱編成のものは，file 宣言文で指定された key 部のデータにより，そのデータをもつ record がアクセスされる．

（4） **record 属性**は，変数を file 宣言文で file 変数とともに宣言することにより与えられる．record 属性をもつ変数（record 変数）は，一緒に宣言された file 変数中のある record を表すとともに，global 属性をもつデータでもある．すなわち，file 制御文中に record 変数を書くことによって，その record 変数の表す file 変数中のある record をアクセスでき，また，“大域的”なデータでもあるため，**共有型記憶域**とよばれるセグメント上にとられ，Lorel-2 の外部手続きも含めて，他の言語（Fortran, Hpl）の外部手続きからもアクセス可能な値をもつ．

（5） **global 属性**は，変数を global 宣言文で宣言することにより与えられ，この変数は record 変数と同様に，“大域的”なデータである．

（1）と（2）の変数は，それが宣言されている手続きが呼ばれる度に動的にその世代の値がとられるが，（3），（4）および（5）の変数は静的に管理されているため，常に同一の値しかアクセスできない．

（6） **hash 属性**は，set や tree 変数に対し，variable 宣言文において，**hash** または **key hash** を指定することにより与えられる．set 変数に対し，**hash** を指定することにより，要素全体がハッシュ化され，その set に対するデータの所属を速やかにきくことができる．要素が n 成分 tuple である set 変数に対し，**key hash** を指定することにより，tuple の第 1〜第 $n-1$ 成分がハッシュ化され，それらを key とした連想検索を速やかに行うことができる．tree 変数に対し，**hash** を指定することにより，その node がハッシュ化され，部分木の所属や temperate matching による部分木の取り出しを速やかに行うことができる（**3.4** 参照）．

3. 式の構成

3.1 式の全体的構成

式には，その表すデータ型により，**integer 式**，**real 式**，**boolean 式**，**string 式**，**procedure 式**，**tuple 式** および **set 式**があり，各々，値が確定的な**単純式**と，条件によって値の選択が行える**条件式**から構成される．

演算子は，integer 式および real 式に対して，＋，－，＊，／，＊＊，boolean 式に対して，∨，∧，～，set 式に対して，∘，∪，∩，－ が用意されている．

単純式は，これらの演算子とリテラル，変数，関数呼出しおよびその後に部分構造指定子の列を付けたもの，Σ 式，set former，および，括弧（.）から構成されたものであるが，＊＊を除いて，2 項演算子の両辺のデータ型は一致しなければならない．

条件式は，次の形をとる．

if $B_1 \rightarrow E_1, \cdots, B_n \rightarrow E_n$ **fi**

ただし，$B_1, \cdots, B_n$: boolean 式，$E_1, \cdots, E_n$: データ型の等しい式

条件式は，$B_1, B_2, \cdots$ と評価していき，初めて **true** となった B_i に対する式 E_i の値を，この式の値とする．

$B_1, \cdots, B_{n-1}$ がすべて **false** で，B_n が省略されている場合は，式 E_n の値をこの式の値とする．

[例 3.1] $I+J*2$ $R+1.5$ $A \vee B \wedge C$ $X \cup Y \cap Z$
func[I, J][K] [$N=1 \rightarrow 10, N=2 \rightarrow 20, \rightarrow 0$]

3.2 set 式の構成要素

Lorel-2 の set は要素として同じデータを含むことを許し，要素の並び方も考慮された ordered set であり，数学で用いられる unordered set と異なる．しか

し，これから説明する pack, ∪, ∩, − と ≦, ≧を用いることによって充分に unorder なものと等しく処理することができる．

これから，Lorel-2 で用いられる代表的な set の機能を説明する．ただし，x と y は，その値のデータ型（すなわち，element/subtree 変数の場合は，それが表すデータのデータ型）が等しい set とする．

（i） pack x: x の要素の中にいくつも等しいものがある場合，その先頭の要素のみを取り出した set を表す．

（ii） $x \circ y$: 2つの set を単につなげた set を表す．

（iii） $x \cup y$: x の後に x に含まれない y の要素をつなげた set を表す．

（iv） $x \cap y$: x の要素の中で，y に含まれるものを集めた set を表す．

（v） $x-y$: x の要素の中で，y に含まれないものを集めた set を表す．

[例 3.2] pack $\{1,2,3,4,5,1,3,3\} = \{1,2,3,4,5\}$

$\{1,2,3,4\} \cup \{3,1,5\} = \{1,2,3,4,5\}$

$\{1,2,3,4\} \cap \{3,1,5\} = \{1,3\}$

$\{1,2,3,4\} - \{3,1,5\} = \{2,4\}$

（vi） Σ 式

$$(\rho|\gamma_1, \cdots, \gamma_n \textbf{ while } B_0 : B_1)E$$

- γ_i: **くり返し条件**とよばれ，次の形がある．

（a） $v=\alpha,\beta,\delta$ （b） $v \in X$ （c） $v \ni X$

（d） $v \in T$

ここで，v: 基本変数 α, δ: integer 式 β: integer 式または ∞ X: set 式 T: tree 式

（a）は，v に初期値 α，終端値 β，刻み δ で integer を与える指定で，β が ∞ のときは，終端値無限を表し，また，δ の省略値は 1 である．

（b）／（c）は，（イ） X の値のデータ型が $\{t\}$ で，v のデータ型が t なら，X の表す set の要素を左／右側から順次取り出して，その値を複写して v に与えていく．（ロ）X が set を表す部分変数で，v のデータ型が **element** X なら，element 変数 v が，set X の要素を，順次左／右側からさしていく．

（d）は，（イ）T の値のデータ型が，v のデータ型とともに $(t, *, \cdots, *)$ なら，T の表す tree の部分木を preorder で順次取り出して，その値を複写して v に与えていく．（ロ）T が tree を表す部分変数で，v のデータ型が **subtree** T なら，subtree 変数 v が tree T の部分木を preorder でさしていく．

くり返し条件の列 $\gamma_1, \cdots, \gamma_n$ の省略値は，$v=1, \infty$ である．

- **while** B_0: **脱出条件**とよばれる boolean 式で，省略値は **while true** である．
- B_1: **選出条件**とよばれる boolean 式で，省略値は **true** である．

E: 式

- ρ: 2項演算子または引数2つの関数名

$\gamma_1, \cdots, \gamma_n$ の指定によって，γ_n の方から先に変化するように値を v に取り出してゆき，B_1 が **true** のときの E の値に，次々と ρ を作用させていき，B_0 が **false** になったとき，くり返しを終了し，その時の値を Σ 式の値とする．

[例 3.3]

$(+|I=1,7,2\ J \in \{3,5,4\}$ **while** $I \leqq 5 : J \geqq 4)(I+J)$
$=(1+5)+(1+4)+(3+5)+(3+4)+(5+5)+(5+4)=45$

（vii） **set former**

$$\{E|\gamma_1, \cdots, \gamma_n \textbf{ while } B_0 : B_1\}$$

各記号は（vi）と同様である．

$\gamma_1, \cdots, \gamma_n$ の指定によって，γ_n の方から先に変化するように値を v に取り出していき，B_1 が **true** のときの E の値を集め，B_0 が **false** になったとき，くり返しを終了し，集めた値から成る set を set former の値とする．

[例 3.4]

$\{I+J|I=1,7,2\ J \in \{3,5,4\}$ **while** $I \leqq 5 : J \geqq 4\}$
$= \{1+5, 1+4, 3+5, 3+4, 5+5, 5+4\}$
$= \{6, 5, 8, 7, 10, 9\}$

3.3 関 係 式

関係式には，**integer 関係式**，**real 関係式**，**構造比較式**，**包含関係式**，**要素関係式**および**部分木関係式**があり，その意味する判定の真偽に従って，**true** または **false** をとる．

（1） integer 関係式: $e_1 \rho e_2$

e_1, e_2: integer 式 ρ : $>$ $\geqq$ $=$ $\leqq$ $<$

e_1, e_2 の値が ρ で表す関係を満たせば **true**，さもなければ，**false** をとる．

（2） real 関係式: $e_1 \rho e_2$

e_1, e_2: real 式 ρ: $>$ $\geqq$ $\leqq$ $<$

（1）と同様の評価をうける．

（3） 構造比較式

ρ: （i）＝または≠ （ii）≦または≧ （iii）≡または≢

（i）は，値のデータ型が等しい式 e_1, e_2 の値が等しいときに，各々，**true** または **false** をとる．（ii）は，

値のデータ型が等しい set 式 e_1, e_2 において，お互いに，一方の set の任意の要素が必ず他方に含まれるときに，各々，**true** または **false** をとる．(ⅲ)は，e_1, e_2 が同一の要素や部分木をとるときに，各々，**true** または **false** をとる．

[例 3.5] $\{1,2,3\} \fallingdotseq \{3,2,1\}$ $\{1,2,3\} \triangleq \{3,2,1\}$

A: $\{1, \underline{2, 3}\}$ のとき，$B \equiv A[2]$ $B \fallingdotseq C$

$\overline{B}$: $\overline{C}$:

(4) 包含関係式: $e_1 \rho e_2$

e_1, e_2: set 式 ρ: $\subset$ または $\not\subset$

値のデータ型が等しい set 式 e_1, e_2 において，e_1 が e_2 に含まれるときに，各々，**true** または **false** をとる．

[例 3.6] $\{1,2,3\} \subset \{1,3,5,4,2\}$

(5) 要素関係式: (ⅰ)$?v$: $e_1 \in e_2$

(ⅱ)$e_1 \notin e_2 v$: e_2 上の element 変数で省略可能

e_1: 式，または，成分として␣を含む tuple リテラルである**擬 tuple** e_2: set 式

(ⅰ)の場合，e_1 が式なら，それが set e_2 中に含まれるときに，e_1 が擬 tuple なら；␣である成分のところを無視した意味で set e_2 中に含まれるときに，この関係式は **true** をとり，満足する要素を v が表す．含まれなければ **false** をとる．

(ⅱ)の場合，上述のように比較して，e_1 が e_2 に含まれないとき **true**，さもなければ **false** をとる．

(ⅰ)を特に，**要素関係式型の連想検索式**とよぶ．

[例 3.7] R: {('TOKYO', 1), ('NAGOYA', 2), ('OSAKA', 3), ('NAGOYA', 4)} のとき，

$?E$: ('NAGOYA', ␣)$\in R$ は **true** をとり，element 変数 E は R の第2要素を表す．

(6) 部分木関係式: (ⅰ)$?v$: $e_1 \in e_2$

(ⅱ)$e_1 \notin e_2 v$: e_2 上の subtree 変数で省略可能

e_1: 式，または，成分として␣を含む tree リテラルである**擬 tree** e_2: tree 式

(ⅰ)の場合，e_1 が式なら，それが tree e_2 の部分木であるときに，e_1 が擬 tree なら，␣である部分木のところを無視した意味で tree e_2 の部分木のときに，この関係式は **true** をとり，満足する部分木を v が表す．部分木でないならば **false** をとる．

(ⅱ)の場合，上述のように比較して，e_1 が e_2 の部分木でないときに **true**，さもなければ **fales** をとる．

(ⅰ)を特に，**部分木関係式型の連想検索式**とよぶ．

[例 3.8] T: (1, (2, **nil**, **nil**), (3, (4, **nil**, **nil**), **nil**)) のとき，$?S$: (3, ␣, ␣)$\in T$ は **true** をとり，subtree 変数 S は下線の部分木を表す．

3.4 連想検索式

連想検索式には，前述した要素／部分木関係式型のものと，これから述べる関数呼出し型のものがある．何れも，set または tree 変数に hash 属性を指定することで，高速の検索を行うことができる．

関数呼出し型の連想検索式 $S[e_1, \cdots, e_{n-1}]$

S: 値のデータ型が $\{(t_1, \cdots, t_n)\}$ の単純式

e_i: 値のデータ型が t_i の式 $(i=1, \cdots, n-1)$

S の要素で，その第1成分が $e_1, \cdots$，第 $n-1$ 成分が e_{n-1} に等しいものの第 n 成分からなる set をこの式の値とする．

[例 3.9] 例3.7で，R['NAGOYA']$=\{2, 4\}$

4. プログラムの構成

4.1 プログラムと手続きの宣言

(1) **Lorel-2 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**である．主プログラムは，Lorel-2 言語で書かれなければならないが，副プログラムは，その他に Fortran, Hpl または Cobol 言語で書くことも許される．

(2) 手続きには，**外部手続き**と**内部手続き**があり，いずれも次の手で宣言される．

H; D_1; …; D_n; P_1; …; P_m; S

(ⅰ)H は**手続き頭部**とよばれ，一般に，次の形をとる．

define $X(p_{11}, \cdots, p_{1l_1}: d_1, \cdots,$
$p_{k1}, \cdots, p_{kl_k}: d_k): d$

X: 手続き名，$p_{11}, \cdots, p_{kl_k}$: 仮引数名

$d_1, \cdots, d_k$: 仮引数の type (後述)

d: 関数値の type

(ⅱ)$D_1, \cdots, D_n$ は，外部手続きならば，**type 宣言文**，**procedure 宣言文**，**variable 宣言文**，**global 宣言文**および **file 宣言文**であり，内部手続きならば，**variable 宣言文**である．

(ⅲ)$P_1, \cdots, P_m$ は，内部手続きの宣言である．

(ⅳ)S は**実行文**である．

(3) 実行文には，**無条件文**，**条件文**およびくり返し文がある．無条件文には，**代入文**，**手続き呼出し文**，**file 制御文**，**入出力文**，**制御文**および空文の**基本文**と文の列を **begin** と **end** で囲んだ**複合文**がある．

(4) 主釈は，PL/I のように /* と */ で囲んで表す．

4.2 宣 言 文

(1) type 宣言文: **type** $t_{11}, \cdots, t_{1l_1}$: $d_1, \cdots, t_{k1}, \cdots, t_{kl_k}$: d_k

$t_{11}, \cdots, t_{kl_k}$ は **type 識別子**とよばれる名前である.

$d_1, \cdots, d_k$ は **type** とよばれ, type 識別子を用いて記述したデータ型である. type 識別子 t_{ij} は右側の type d_i の意味するデータ型を表す.

(2) procedure 宣言文: **procedure** $p_{11}, \cdots, p_{1l_1}$: $d_1, \cdots, p_{k1}, \cdots, p_{kl_k}$: d_k

$p_{11}, \cdots, p_{kl_k}$: 手続き名, $d_1, \cdots, d_k$: type

この外部手続き内で用いられる手続き名の type の宣言を行う.

(3) variable 宣言文: **variable** $v_{11}, \cdots, v_{1l_1}$: $d_1 s_1, \cdots, v_{k1}, \cdots, v_{kl_k}$: $d_k s_k$

$v_{11}, \cdots, v_{kl_k}$: 変数名 $d_1, \cdots, d_k$: type

$s_1, \cdots, s_k$: **virtual, hash** または **key hash** の並び

その手続きで有効な変数と, その type および記憶域属性 (virtual および hash) の宣言を行う.

(4) global 宣言文: **global** $v_{11}, \cdots, v_{1l_1}$, : $d_1, \cdots, v_{k1}, \cdots, v_{kl_k}$: d_k

$v_{11}, \cdots, v_{kl_k}$: 変数名 $d_1, \cdots, d_k$: type

global 変数は, 他の外部手続き (Fortran, Hpl も許す) の同じ名前の global 変数と値を共有することができる.

(5) file 宣言文: **file** $f_1, \cdots, f_k$

f_i は file 変数 F とその上の record 変数 $R_1, \cdots, R_n$ の宣言で, 次のいずれかである.

(a)順編成 file F: d **sequential** $R_1, \cdots, R_n$

(b)相対編成 file F: d **relative** $R_1, \cdots, R_n$

(c)乱編成 file F: d **random** $(c_1, \cdots, c_m) R_1, \cdots, R_n$

ここで, d は type, $c_1, \cdots, c_m$ は乱編成 file の record の keyを表す成分識別子である.

[例 4.1] **type** set: {triple}, triple: (s, i, r);
procedure subr: [p: set, **element** p],
factorial: [i]→i;
variable x: set **virtual keyhash**, y: {i};
file F_1: {(s, i)} **sequential** R_1,
F_2: {(one: i, two: r, three: s)} **randam** (one, three) R_2, R_3

例えば, type 識別子 set はデータ型 {(s, i, r)} を表す.

4.3 代 入 文

Lorel-2 におけるデータの処理は, file 変数以外は, すべて代入文の形を通して行われる.

代入文によって行われる処理は, (1)データの書き換え, (2)set にデータを要素として追加, (3)set からの要素の削除および(4)element/subtree 変数の設定とあるが, 各々, 次の4種類の代入文によって行われる.

(1) **基本代入文**: $L := R$

L: 変数, R: 式で, L と R の値のデータ型は一致しなければならない.

この文によって, R の値が L に代入される.

(2) **挿入代入文**: $L\downarrow := R$ または $\downarrow L := R$

L: set 要素を表す変数, R: 式で, L と R の値のデータ型は一致しなければならない.

この文によって, L の表す set 要素の右または左隣りに, R の値を要素として挿入する.

(3) **要素削除代入文**: $L := \varepsilon$

L: set 要素を表す変数, ε: 要素削除記号

この文によって, L の表す要素が, その set から削除される.

(4) **element/subtree 代入文**: $L :\equiv R$

L: データ型が **element** S または **subtree** T なる変数, R: set S の要素または tree T の部分木を表す変数

この文によって, L が R の意味する set 要素または tree 部分木を表す element/subtree 変数となる.

[例 4.2] A: {$\underset{B:}{1}, \underset{C:}{2}, 3$}, X: {10, 20} のとき

$X := A \Longrightarrow X$: {1, 2, 3}

$B := 0 \Longrightarrow A$: {$\underset{B:}{0}, \underset{C:}{2}, 3$}

$\downarrow C := 0 \Longrightarrow A$: {$\underset{B:}{1}, 0, \underset{C:}{2}, 3$}

$A[0]\downarrow := 0 \Longrightarrow A$: {$0, \underset{B:}{1}, \underset{C:}{2}, 3$}

$B := \varepsilon \Longrightarrow A$: {$\underset{C:}{2}, 3$}

$A[3] := \varepsilon \Longrightarrow A$: {$\underset{B:}{1}, \underset{C:}{2}$}

$B :\equiv A[3] \Longrightarrow A$: {$1, \underset{C:}{2}, \underset{B:}{3}$}

$C :\equiv B \ni 2 \Longrightarrow A$: {$\underset{B:}{1}, 2, \underset{C:}{3}$}

4.4 手続き呼出し文

手続き呼出し文は次の形をとる.

$p[a_1, \cdots, a_n]$

p: 単純 procedure 式, a_i: 実引数

手続きは, 再帰呼出しが許される.

実引数は，参照取りで引渡される．

[例 4.3] subr のデータ型が $[r,i]$，A: 3.14，B: {1, 2, 3} のとき，subr $[A,C]$ によって，第1実引数として変数 A，第2実引数として値2が subr に引渡される．

4.5 file 制御文

file 制御文は，file 変数のアクセスのために用いられ，その代表的なものは次の形をとる．

動詞 (record 変数, record 指定子)

各々の動詞に対する file 制御文の意味は次の通りである．

(1) **get**: record 指定子で指定された record の値を record 変数に与え，record 変数はその record をさす．

(2) **put**: record 指定子で指定された record に record 変数の値を書き込み，record 変数はその record をさす．

(3) **add**: 指定の record に値が書き込まれていたなら書き込みを行わないという点を除けば，**put** と同じである．

(4) **delete**: record 指定子で指定された record を削除する．

record 指定子は次の意味をもつ．

begin/end: 順編成 file に対して，file の先頭／末尾の record を表す．

current/next: record 変数がさす record/ その次の record を表す．

record=n: 相対編成 file に対して，integer 式 n によって，file の先頭から n 番目の record を表す．

key=データの列: 乱編成 file に対して，key のデータを与えて，それをもつ record を表す．

動詞による file のアクセスに伴って発生する異常状態は，特別のシステム変数によって知ることができる．

4.6 制 御 文

(1) **stop** 文は，実行の停止を表し，次のように書く．

stop

(2) **return** 文は，手続きからの復帰を表し，

return または **return**(d)

と書き，戻すべき関数の値を式 d で指定する．

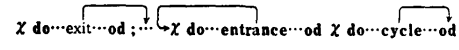
(3) **exit** 文は，**exit** または **exit**(l)

entrance 文は，**entrance** または **entrance**(l)

cycle 文は，**cycle** または **cycle**(l)

と書き，l はくり返し文または複合文にはられた名札である．

くり返し指定子 χ による文の列 $S_1,\cdots,S_n$ のくり返し文は，χ **do** S_1; …; S_n **od** と書かれるが (**4.8** 参照)，くり返し文中の **exit** が実行されることで，χ によるくり返しをここで打ち切り，次の文が実行され，**entrance** が実行されることで，χ によるくり返しが再び最初から実行され，**cycle** を実行することで，χ によるくり返しが引続き実行される．



名札のはられたくり返し文 l: χ **do** S_1; …; S_n **od** に対しても，**exit**(l)，**entrance**(l) および **cycle**(l) によって同様のことが行える．また，複合文に対しても，exit 文は有効であり，これによって，その複合文の次の文が実行される．

4.7 複 合 文

複合文は，実行文 $S_1,\cdots,S_n$，名札 l に対し，

begin S_1; …; S_n **end** または l: **begin** S_1; …; S_n **end** [l] と書かれ，名札 l は exit 文の指定中に用いられる．

[例 4.4] **begin** I:=1; J:=2; K:=3 **end**

4.8 条 件 文

条件文は，次のように書かれる．

if $B_1\rightarrow\mathfrak{C}_1,\cdots,B_n\rightarrow\mathfrak{C}_n$ **fi**

B_i: boolean 式

$\mathfrak{C}_i$: 実行文の列，すなわち，実行文; …; 実行文

$B_1, B_2, \cdots$ の順に調べてゆき，初めて，**true** となった B_i に対する $\mathfrak{C}_i$ が実行されるが，そのような B_i がなければ，何も実行されない．$B_1,\cdots,B_{n-1}$ がすべて **false** で，B_n が省略されている場合は，$\mathfrak{C}_n$ が実行される．

[例 4.5] **if** $K\neq 0\rightarrow I:=X[K]; K:=K+1,$
$\rightarrow I:=0$ **fi**

4.9 くり返し文

くり返し文は，**くり返し指定子** χ および文の列 $S_1,\cdots,S_n$，名札 l に対し，

χ **do** S_1; …; S_n **od** または l: χ **do** S_1; …; S_n **od** [l] と書かれ，名札 l は exit, entrance および cycle 文の指定中に用いられる．

くり返し指定子 χ は，一般に，次の形をとる．

$$\left(\gamma_1,\cdots,\gamma_n\left\{\begin{matrix}\textbf{while}\\ \textbf{until}\end{matrix}\right\}B_0\colon B_s\right)$$

各記号は 3.2(iv) を参照.

くり返し文は，次のように実行される.

$\gamma_1, \cdots, \gamma_n$ によって，γ_n の方から先に変化するようにくり返しが行われ，(イ)**while** 指定されている場合，B_0 が **false** のとき，くり返しを打ち切り，次の文が実行される．B_0 が **true** のとき，B_i が **true** なら，文の列 $S_1, \cdots, S_n$ が実行され，B_i が **false** なら，何も実行されないで，引き続きくり返しが行われる．(ロ)**until** 指定されている場合，B_i が **true** なら，文の列 $S_1, \cdots, S_n$ が実行され，B_i が **false** なら，何も実行されない．次に，B_0 が評価され，**true** なら，くり返しを打ち切り，次の文が実行される．**false** なら，引き続きくり返しを行う.

[例 4.6] $\sum_{\substack{k=1 \\ k \neq 5}}^{n} k^2$ を計算するプログラムを2つ示す.

SUM:=0; (K=1, N: $K \neq 5$) **do** SUM:=SUM+K**2 **od**

SUM:=0; K:=1; (**while** $K \leqq N$: $K \neq 5$) **do** SUM:=SUM+K**2; K:=K+1 **od**

もちろん，Σ 式で，(+|K=1, N: $K \neq 5$)K**2 と書くこともできる.

最後に，Lorel-2 のプログラム例を掲げる.

```
define Main;
  type VELEM: element V;
  variable G: {(i, {i})} key hash, EG: element G,
           V: {(vertex: i, visit: b)} key hash,
           T: {(i,i)}, EV: VELEM;
  /* subroutine Search */
  define Search (v: i);
    variable EV: VELEM, w: i;
    begin
      if ?EV: (v,_)∈V→EV.visit:=true fi;
      (w∈G[v][!]) do
                    if ?EV: (w,_)∈V∧~EV.visit
                                  →T:=T∪{(v,w)};
                                    Search [w]
                    fi
                  od;
      return
    end;
  /* Main routine execution part */
  begin
    read (CD: G);
    T:={ };
    V:={(EG(l), false)|EG∈G};
    (EV∈V) do
             if~EV.visit→Search [EV.vertex] fi
           od;
    write (LP: T);
    stop
  end
```

このプログラムは，undirected graph G に対して，その depth first search を行った結果の spanning tree を T に与えるものである[10]. graph の vertex は integer で表している．G は，その要素である tuple の第1成分が vertex を表し，第2成分がそれに隣接する vertex の set を表し，カードからの入力によって与えられる．T は，その要素である tuple が vertex の接続を表し，ラインプリンタに出力される．V は，その要素である tuple の第1成分が G の vertex を表し，第2成分が G を search するとき既に visit したかどうかを表す.

5. おわりに

Lorel-2 の処理系は，コンパイラと実行システムから構成されており，コンパイラが Lorel-2 ソースプログラムを基本命令とよばれる目的コードの列に変換し，これを実行システムが実行する．コンパイラは，構文図に意味づけを施した拡張構文図[8]によって，記述され，データ型の処理，エラー処理および set の目的コードの工夫等を行い，さらに，data flow analysis による変数の busy check も行っている．実行システムは，マイクロプログラム化した基本命令を用いて，データ構造を処理しており，ゴミ集めもマルチタスク方式で実現している.

参考文献

1) 片山，日比野，榎本，榎本：論理関係処理言語 LOREL-1，情報処理，Vol. 15, No. 5 (1974).
2) 榎本，片山，榎本：LOREL-1 による記号処理の一例，記号処理シンポジウム報告集 (1974).
3) 榎本，片山，榎本，吉田：LOREL-1 による構造線の処理と認識，第 16 回プログラミング・シンポジウム報告集 (1975).
4) 榎本，片山，榎本：LOREL-1 の性能評価，情報処理学会第 16 回大会講演論文集 (1975).
5) 榎本，宮地，片山，榎本：LOREL-2 データの記憶域属性とその実現，第18回プログラミング・シンポジウム報告集 (1977).
6) 榎本，宮地，片山，榎本：Lorel-2 言語について，信学会計算機研資，EC 76-83 (1977).
7) 宮地，榎本，片山，榎本：Lorel-2 実行システムの構成，信学会計算機研資 (1977).
8) 榎本，宮地，片山，榎本：Lorel-2 言語システムについて，情報処理学会記号処理研資 2-2(1977).
9) J. T. Schwarz: Automatic Data Structure Choice in a Very High Level, C. ACM 18, No. 12.
10) A. V. Aho, et al.: The Design and Analysis of Computer Algorithms, Addison-Wesley (1974).

(昭和 52 年 4 月 15 日受付)
(昭和 52 年 12 月 16 日再受付)