プログラミングシステム LINKS-C

出口 弘 河合利幸 中西 隆 西村仁志
河田 亨 白川 功 大村皓一（大阪大学 工学部）

[1] まえがき

多次元画像生成システムを開発するにあたり、そのシステムの主要言語として何を選ぶかは、その後のプログラム開発に重大な影響を与える。主要言語として持つべき特長は、汎用であること、実行効率が良いこと、記述性が良いこと、言語自身がコンパクトで処理系の開発期間が短いこと、移殖性が良いことなどである。この様な条件を最もよく満たす言語として、C言語を採用した。

マルチコンピュータシステム用に拡張したり、専用演算ハードウェアが効率よく利用できる様に、さらに実行時における効率的デバッグを行なうためのオブジェクト生成を可能とするために、C言語処理系を設計し、作成した。

画像生成システムは、コンピュータに関する知識を持たない様なユーザ（アーチスト）にも、簡単に利用できるものでなければならない。そこで日本語でプログラミングできる様なシステムについても述べる。

[2] C言語処理系LINKS-Cの設計と作成

2-1. C言語処理系への要求

言語処理系が満たすべき条件を挙げる。

(1) 処理速度が速いこと。
(2) オブジェクトの実行効率が良いこと。
(3) デバッグが容易であること。
(4) 移殖性が良いこと。
(5) 他の言語プログラムとのリンクが容易であること。
(6) 既に開発されているプログラム資産が有効に利用できること。

(1)の処理速度に対する要求は、(3)のデバッグの容易さに大きく関係する。これらの条件を満たせば、プログラム開発期間の短縮、及び実行効率の良いプログラム作成が可能となる。

2-2. 処理系の概要

図2-1に本処理系の構成を示す。本処理系におけるコンパイラは、マクロアセンブラのソースプログラムをオブジェクトとして出力する。こうすることによって、番地の解決を全てアセンブラに任せることができたので、1パスのコンパイラにすることが可能となった。従ってコンパイラ自身のプログラムサイズも小さくなり、開発期間も短縮できた。

エディタによって作成されたCプログラムは、まずCプリプロセッサによってコンパイラ制御行の処理が行なわれる。この中間ソースプログラムをコンパイラで処理することにより、マクロアセンブラのソースプログラムが出力される。アセンブラの出力は、リロケータブルオブジェクトである。完全なプログラムを構成するためには、必要な全てのリロケータブルオブジェクトと、ライブラリをリンカに指定することにより、アブソリュートオブジェクトを作る。実行はこれをローダによってメモリ上にロードすることによって開始する。

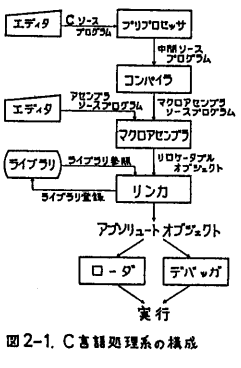


図2-1. C言語処理系の構成

本システムでは、

実行時の効率的デバッグを行なうため、シンボリックデバッガを用意した。このデバッガは、デバッグモードの指定を受けたコンパイルオブジェクトを入力することによって実行時のデバッグを可能にする。

本言語処理系はザイログ社Z8001MPUのセグメントモードの機械語オブジェクトを生成するものである。

2-3. Cプログラムの実行形式

実行時のメモリ配置を図2-2に示す。実行はスタックマシーンとして行なう。自動変数領域は、関数に入る時スタック内に確保され、関数から帰る時に消滅する。ヒープ領域は記憶割り当て関数allocで使用する領域である。なお、コードセグメント、外部変数・静的変数領域、ヒープ領域、エバリュエーションスタック・自動変数領域をZ8001のメモリ空間の別々のセグメントに配置することが可能である。

演算はスタックを用いて行なうが、高速化のために、トップオブスタックレジスタ（以下TOSR）を2個用意し、可能な限り演算をこの2個のTOSR間で行なうようにした。（図2-3）

関数呼び出しは、引数をスタックに積み、関数をcallする。callされた関数ではその上に自動変数領域を確保し帰り番地と直前のマークスタックポインタ（以下MP）を保存する。そして新しいMPを引数と自動変数の参照用にセットする。（図2-4）関数からのリターンは、関数の値をTOSRにセットし、スタックとMPを元にもどしてリターンする。

2-4. データ部の処理

図2-5にコンパイラの構成を示す。また図2-6にはシンボルテーブルのデータ構造を示す。

ブロック構造による名前の適用範囲の問題は、次のようにして解決した。図2-7に示すように、シンボルテーブルの属性レコードは、同じ名前の変数が宣言されているブロックに出入する時に動的に変化するようにした。

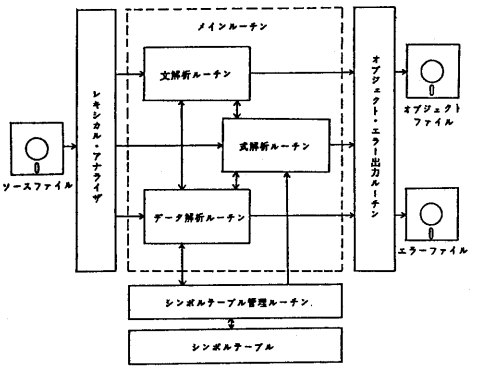


図2-5 コンパイラの構成

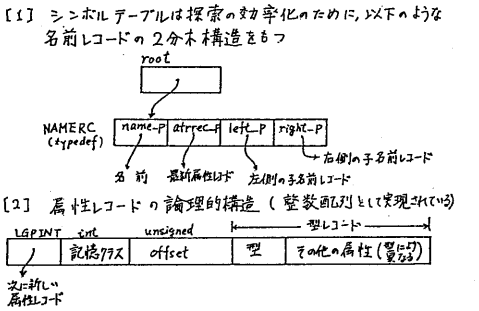


図2-6 シンボルテーブルのデータ構造

```
int a(x,y)
  int x,y;
{
  int i,j;
  return(x+y);
}
```

```
{ int i;                    /* ブロック1 */
  { int i,j;                /* ブロック2 */
    { int i;                /* ブロック3 */
    }
  }
}
```

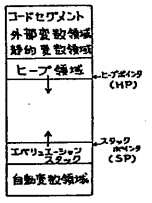
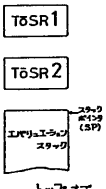


図2-2 実行時のメモリー配置



トップ・オブ・スタック・レジスタ

図2-3

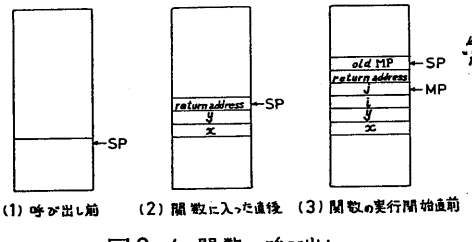


図2-4. 関数の呼び出し

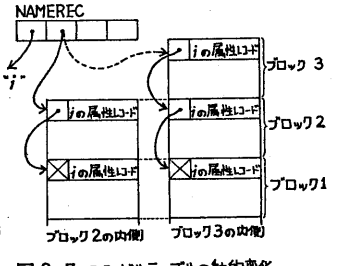


図2-7. シンボルテーブルの動的変化

自動変数と引数はスタック内に領域が確保されるため、参照はMPからのオフセットによって行なわれる。

外部変数、静的変数はアセンブラの擬似命令によって領域確保と初期化がなされる。外部変数はグローバル宣言、外部静的変数はローカル宣言をすることによってアセンブラに処理を任せる。

2-5. 制御命令の処理

制御命令の分岐先番地も、ラベルをアセンブルソースとして出力することによりアセンブラによって解決される。図2-8に実際のコンパイル結果の一部を示す。

2-6. 式の処理

式の解析は演算木を作っていくことによって行ない、仮想スタックマシーンに対する命令を中間オブジェクトとして生成する。定数計算や結合則を用いた最適化等の処理がこの演算木を用いてなされる。図2-9, 10にこの演算木と中間オブジェクトの例を示す。

```
        CSEG    main
          LD        R1w,a
        TEST    R1w
        JP      Z,_ELSE0
        CSEG    main
          INC       i+(0),#1
        JP      _IFEXT0
_ELSE0:
        CSEG    main
          DEC       j+(0),#1
_IFEXT0:
_WHILE0:
          LD        R1w,i
        TEST    R1w
        JP      Z,_WHILEXT0
        CSEG    main
          INC       a+(0),#1
        JP      _WHILE0
_WHILEXT0:
```

図2-8 コンパイル結果の例

```
int s[100];
sort(l,r)
    int l,r;
{
    int   x;
    x=s[(l+r)/2];
}
```

```
VAR,ADR,GBL,INT,INT,s
VAR,VAL,AUTO,INT,INT,2
VAR,VAL,AUTO,INT,INT,4
PLS,INT,0,0
CNST,INT,2
DIV,INT,0,0
PLS,PTR,0,2
PTR,INT,INT,0
VAR,ADR,AUTO,INT,INT,12
LET,INT,INT,0
CMA,0,0,0
```

図2-10 中間オブジェクト

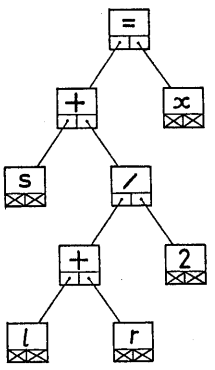


図2-9. 演算木

2-7. オブジェクト出力の処理

中間オブジェクトをZ8001アセンブルソースに変換する。プロセッサに依存する最適化処理はこの段階でなされる。この部分のみを他のプロセッサ用のものと交換するだけで本コンパイラは任意のプロセッサに対するアセンブルソースを出力することができる。

[3] マクロアセンブラ・リンカ・ローダ

C言語処理用に拡張したアセンブラシステムの構成を図3-1に示す。

Cの関数はその関数名をモジュール名とする実行コードの置かれたCSEGと、内部静的変数、定数、文字列等の置かれたDSEGになる。外部変数はグローバルラベルとなり、DSEGだけからなる1つのモジュールを構成し、外部静的変数はファイル内からだけ参照可能なローカルラベルになる。モジュール外のラベルを参照する時はそのラベルをエクスターン宣言する。

Cプログラムは、多数の小さな関数から構成されるのが普通である。1つ1つの関数のCSEGのサイズが64kbyte(Z8001の1セグメントの最大サイズ)を超えることはないと考えられる。DSEGのサイズについても64kbyteを超える大きな配列や構造体を定義しない限りリンカが適切に配置処理をするので全体として大きなプログラムでもコンパイルできる。

Cプログラムの相対番地オブジェクトと、ライブラリは完全に同一視できるので入出力や高速処理を要求される部分をアセンブラで記述したものはライブラリとしてCプログラムの関数とすることができる。

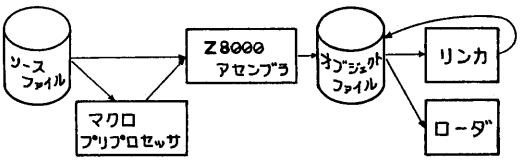


図3-1. アセンブラシステムの構成

[4] 浮動小数点演算に関して

画像処理をはじめ、多くの科学技術計算には浮動小数点演算が必要であり、高速であることが望まれる。アセンブラで演算パッケージを作るとどうしても時間がかかってしまう。そこで演算ハードウェアを使用することが考えられる。本システムでは演算用に i8086, i8087 のプロセッサを使用した。演算プロセッサ i8087 に加えて演算制御用に専用 LPU i8086 を用いることにより、単純な四則演算だけでなく、ベクトル演算、行列演算をはじめ複雑な演算を独立に行なうことができるようになった。この場合、Cプログラム(z8001)はデータを用意し、演算ユニット(i8086, i8087)を起動する関数を呼び、結果が返ってくるまでは別の処理を続けるということができる。

[5] シンボリックデバッガの設計

Cのプログラムのバグは、コンパイル時にコンパイラが文法誤りを検出する他、強力な型チェックを必要とする場合には lint と呼ばれるプログラムによって型の不一致、引数使用上の矛盾等のバグが検出できる。しかし実行時のデバッグは一般に煩雑な作業である。そこでシンボリックデバッガを用意した。

デバッガに要求されること

(1) 実行を停止して変数値の参照、変更ができ、続きを再実行できること。
(2) 変数は名前で指定できること。
(3) 制御構文の各分岐点でどちらに行くかを確認、変更できること。
(4) 未だバグのとれていない関数(又は未定義関数)に対するダミー関数の返してきた値を参照、変更できること。
(5) デバッグ条件(引数の値等)を設定し、プログラムの各部をどのように通ったかわかること。
(6) (5)の history が蓄積できること。

以上のことが満足されれば、実行時のデバッグ及びプログラムの検査が可能となるであろう。

本デバッガでは、デバッグ対象となる関数を数個指定するものとする。多くのCプログラムは関数を用いて構造的に作られるため、下位の関数から順にデバッグを進めれば一度に多くの関数をデバッグする必要はないと思われる。実行を停止できる場所を、その関数に入った時、各制御文の条件式の評価の後、関数呼び出しの直後とする。プログラムの制御構造を、分岐点、合流点等を節点とし、文を枝とするようなグラフで表現することにより、各部をどのように通ったかを記録する。(図5-1参照)。デバッグモードのコンパイラは、変数テーブルとプログラム構造グラフを出力する他、停止点にデバッグ関数の call 命令を挿入する。

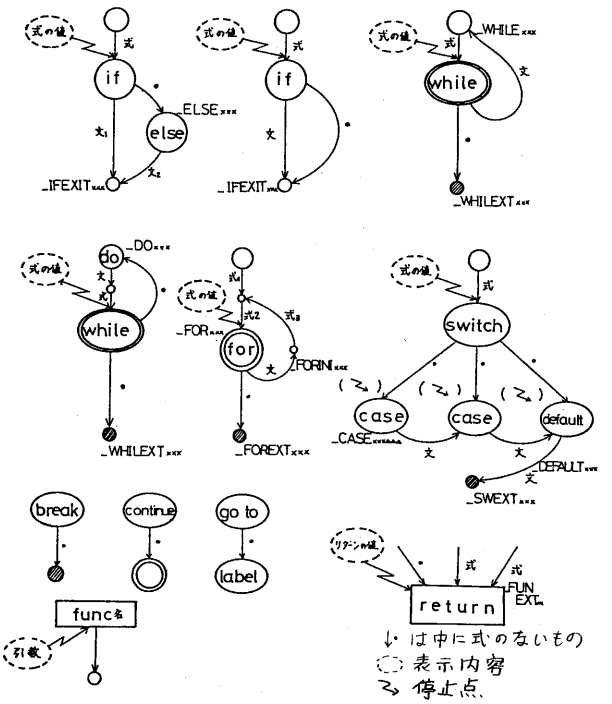


図5-1 デバッガ出力グラフ

[6] 日本語プログラミングシステム LINKS-J

マイクロコンピュータの急速な普及により「1人に1台のコンピュータ」という時代が目前のように思われる。しかしそこには大きな問題がある。それはいかにそれ使いこなすかということである。プログラミングという作業は現存の言語の複雑な構文規則のためか、英語的なものへの拒否反応か、日本では広く一般に受け入れられているとはいえないようである。そこでもし日本語でプログラムが書けるとすれば、どれほど多くの日本人が、しかも簡単にプログラミングできるようになることであろうか。そこで一つの日本語プログラミングシステムを提案する。

6-1 LINKS-Jの概要

日本語プログラミングシステムといっても、日常言語をそのままプログラムとして処理するようなことは不可能であろうから、おのずから制限がある。しかしまたFORTRAN等のプログラミング言語のステートメントを逐語訳しただけのものでは、日本語にはならずかえって使いづらいものになってしまうことは明らかである。

そこで、従来のプログラミング言語のようなシーケンシャル言語ではなく、関数的プログラミング(FP)システムの考えを取り入れた。プログラムは単に変数を持たない関数である。このシステムは対象、対象を対象に写像する関数、関数定義、構文規則、作用から構成される。原始関数を元にして個々の関数を定義し、それらを階層的に組合せてさらに機能の高い関数を定義していく。

6-2 日本語プログラミング言語への翻訳

日本語の文の構造は

何が　どうする

のように主語と述語がある。関数の様な操作を表現するときには主体即ち主語は省略されて

ＸＸを△△する、ＸＸに△△する

という表現になる。即ち関数は述語である。ＸＸは対象(関数の引数)であり、引数は1個であるから多くの場合省略される。

ここで簡単な関数の数学的表現と日本語による表現を示す。

$h(x)=2x$　　(Ｘを)2倍する　－①

$g(x)=x+1$　　(Ｘに)1を加える－②

hとgの合成関数を考えると

$f(x)=g\circ h(x)=g(h(x))=2x+1$

(Ｘを)2倍して、1を加える　－③

となる。この様に関数は動詞又は形容詞で表現されているので「活用」する。しかし関数を表現するために必要な活用形は、連用形と連体形(終止形)だけで十分であり、「加えよ」等の命令形も許さない。また②の様に「加える」を修飾する目的語や補語等もある。この場合、語と語あるいは句と句との関係を示す助詞、特に格助詞が重要になってくる。本システムでは格助詞として「が、の、を、に、と、より、から、で」の8個を選び、それぞれの用法を限定した。

「が」は主格を表わす。条件にのみ用いられる。

(例) Ｘが原子である

「の」は所属、材料を表わす。名詞が続く。

(例) Ａの転置

「を」は目標、対象を表わす。

(例) 1を加える

「に」は対象を表わす。

(例) Ａに1を加える。

「と」は並列を表わす。

(例) Ｘとｙの積

「から」は起点を表わす。

(例) Ｘを5から引く

「より」は比較を表わす。

(例) 1より大きい

「で」は手段、材料を表わす
（例） xを3で割る

既に出てきたように2項あるいはそれ以上に対して演算（操作）が施されるような場合には、複数の格助詞が用いられてそれぞれの項の関係を示している。（図6-1）

スカラーではなくてベクトルを引数として、ベクトルを値（結果）として返す（持つ）様な関数の場合は、入力を $x=\langle x_1 \cdots x_n \rangle$ 出力を $y=\langle y_1 \cdots y_m \rangle$ とすると $y=F(x)$ と表わされ、出力成分を個々に表現すると $y_i=f_i(x)$ $1 \leq i \leq m$ となるとすれば、出力は $\langle f_1(x), \cdots f_m(x) \rangle$ となる。これを日本語で表現すると「xにf_1を施したもの、…… f_mを施したもの」となる。このようにconstructionの概念は、『関数の連用形"したもの」又は関数名を"、"で区切って連ねることによって表わされる。

一方合成関数の様に関数が逐次施される場合は、「xを2倍して1を引く」のようになるcompsitionの概念「"関数の連用形"してを、"、"で区切ったものを連ねて表わされる。

6-3 構文規則

図6-6にLINKS-Jの構文図を示す。

6-3-1 予約語（keyword）

以下のひらがな、漢字又はその系列は、文中他の用途（関数名その他）に使ってはならない。

関数　と定義する
もし　ならば　さもなければ
間は　次の操作を繰り返す
たもの　て　それ　そのもの
全　名　第　番目の　各々の
全ての　成分　かつ　または　か
が　の　を　に　と　より　から
で

以上のかな等は構文解析において、文の構造に対する情報を与えると共に、単語の区切りの役目も果す。

またこれらを別の目的で使う場合はカタカナを用いるか、又は「---」で識別できるようにする。

6-3-2 関数

関数には真偽を返す関数（例 等しい）と操作をする関数（例 足す）の2種類がある。真偽を返す関数は条件文でのみ用いられる。真偽を返す関数には次の3種類がある。

＜ある型＞"名詞"である（り）
　　例 原子である（り）
＜ない型＞"名詞"でない（く）
　　例 原子でない（く）
＜形容詞型＞"形容詞"
　　例 等しい（く）

操作をする関数には次の4種類がある。

＜する型2＞"名詞"する（し）
　　例 加算する（し）
＜する型1＞"名詞"する（し）
　　例 転置する（し）
＜とる型＞"名詞"をとる（り）
　　例 和をとる（り）
　　転置をとる（り）
＜動詞型＞"動詞"
　　例 足す（し）

（）内は連用形の場合を表わす。関数にはこのほか連体形（終止形も同じ）と名詞形が存在する。

関数名は「～をとる」，「～を求める」とつながることができ，「"連用形"たもの」，「"名詞形"したもの」と同じ概念を表わす。

名詞形は，「～を施す」，「～する」とつながることができ，「～すること」という概念を表わす。関数とこれが同じもの（＜する型1＞）と異なるもの（＜とる形＞）がある。

連用形は「～て」，「～，」とつながることができる。

連体形は言いきりの形である。

関数定義の際、関数名、名詞形、連用形、連体形もそれぞれ定義するわけだが、このとき＜動詞型＞，＜形容詞

型〉は個々に連用形、連体形を定義しなければならないが、とる、求める、する、施す、ある、ない等はキー動詞としてまとめて処理することができる。("ない"は実は形容詞)

6-3-3 apply to all, insert

図6-2のような構文はapply to allとなり、図6-3の場合はinsertとなる。

6-4 あいまいさについて

日常使われている日本語がそうであるのと同様に、図6-1の構文図から明らかなようにこの文法はあいまいである。"dangling else"に対応する「ぶらぶら さもなければ」が存在する。これは直前にある「さもなければ」なしの「もし」に関係づけされる。またその他のあいまいさは、翻訳系が勝手に(正しいと思われるように)処理してしまうので、あいまいな部分、例えば条件文や、各々まとまった操作を「」でくくってあいまいさをなくすようにプログラマがしなければならない。

6-5 処理の概略

日本語プログラムJおよび対象Dから計算値を得る方法について述べる。Jを中間言語にコンパイルし、その中間言語を解釈実行する。ここでは中間言語としてBackusのFPシステムを採用した。関数の活用形、助詞の使用法等で制御構造の明らかになったプログラムJは、次の様にしてプログラムPに翻訳される。

① 関数は全て関数名に置き換える。
② 第S成分等はSelector function s。
③ そのもの,それ,はIdentity id。
④ 引数でない対象Xは Constant X。
⑤ 全成分にxxする等はInsert \f。
⑥ 各成分にxxする等はApply to all df。
⑦ でない等は not。
⑧ fにgする等は Compositor g∘f。
⑨ fとg等は Construction [f,g]
⑩ 条件文は Conditor (p→f;g)
⑪ 繰り返し文 While (while pf)
⑫ 図6-1に示すように助詞の解析の結果はConstruction, Insert, Apply to all, Composition を組合せたものとなる。
例えば、第1成分に第2成分の長さをかけるは
X∘[1, length∘2]
となる。

図6-5に処理系の構成を示す。

6-6 データ構造

対象データは木で容易に表わされるため、計算機内部でも2分木リスト表現の木とする(図6-4)。原子に対するセルは、その型と値を持つ。プログラムJがPにコンパイルされるとき対象Dもこの内部表現に変換される。

〔7〕 あとがき

LINKS-Cに関しては、コンパイラは完成し、実際の画像生成に使われている。シンボリックデバッガは開発中である。

LINKS-Jに関しては、基本設計について述べた。まだ多くの検討を要するが、多くの日本人にとって抵抗なく使えるような、日本語プログラミング言語になる可能性を与えた。現在システムの試作を進めている。

参考文献

(1) B.W Kernighan and D.M.Richie, The C Programming Language, Englewood Cliffs, N.J Prentice-Hall 1978
訳書:「プログラミング言語C-UNIX流プログラム書法と作成」石田晴久訳(共立出版, 1981)

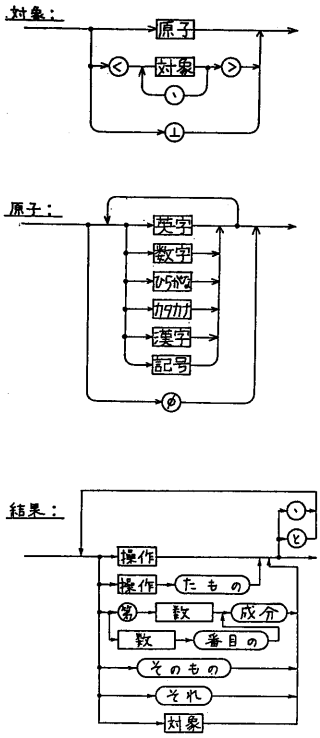
(2) John Backus, Can Programming Be Liberated from the von Newman Style? A Functional Style and Its Algebra of Programs, CACM vol21 No8 1978

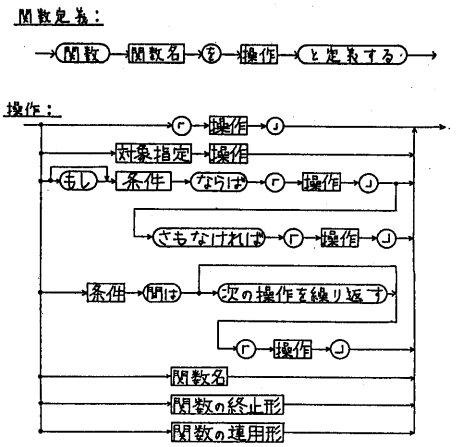
(3) 井上,倉沢,石口,岡本"関数的プログラミング言語FPLの処理系の作成", 信学論(D) J65-D, 5, PP566-573 (昭57-5)

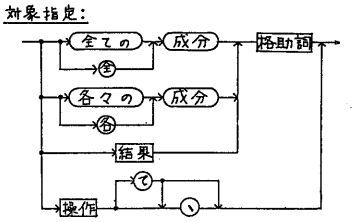
①I ＼ ②II	①I②II op がのをにとよりからで
id より	が × × × + + + × ×
id ×	の × × × × × × × ×
id で	を × × × − − × − +
id を	に + × + × × × × ×
id と	と、+ + + × +(a) × × ×
id が	より − × × × × × × ×
id を	から × × + × × × × ×
id を	で × × − × × × × ×
①I省略時	− × + − − + − +

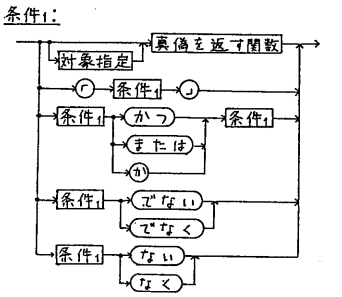
- ①I②II op　①,②は結果を表わす.
  +：op･[①,②]（op[id,②]）
  −：op･[②,①]（op[②,id]）
  (ex) ｵ1成分にｵ2成分を掛ける.
  ①ｵ1成分 1st
  ②ｵ2成分 2nd
  I に
  II を
  op 掛ける. X
  この場合,(に,を)は＋であるから
  X･[1st.2nd]
- (と,と)の(+)は Construction になることを表す.

図 6-1. 格助詞の関係による翻訳規則









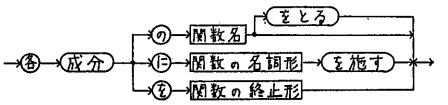


図 6 - 2　apply to all の構文

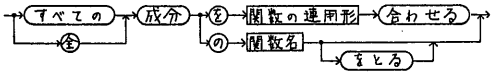


図 6 - 3　insert の構文

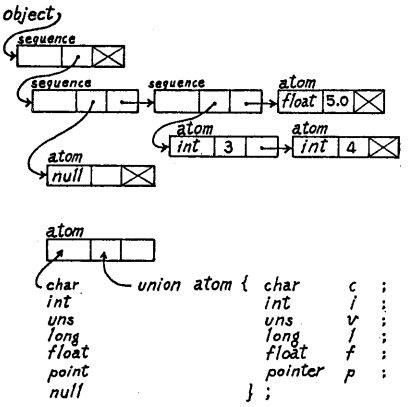


図 6-4. 対象のデータ構造

図 6-6 LINKS-J の構文図

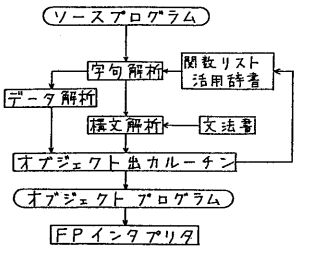


図 6 - 5　LINKS - J の構成