手続き型処理のための拡張を施された PROLOG システム FLOGS の開発†

河合和久††　願化真志††　豊田順一†††

述語論理型言語 PROLOG は，知識情報処理に向いた各種の優れた性質をもつが，画像や音声等の大量データに対する手続き的な処理には不向きである．しかし，知的 CAD や画像理解システム，連続音声認識システムの開発には，PROLOG のように知識の取扱いが容易で，かつ FORTRAN のように大量の手続き処理を高速に処理できるような言語が必要である．本論文では，こうした言語として，われわれが設計・開発した FORTRAN と PROLOG のリンクが可能な新しい PROLOG システム FLOGS について述べる．FLOGS では，FORTRAN サブルーチンを呼び出す組込み述語を用意し，PROLOG のゴールとして実行することによって，両言語のリンクを可能にした．さらに，一元的大量データの取扱いならびに両言語間の融通をよくするために，PROLOG 側に新しい基本データ型として配列を導入し，リストと配列との変換を行う機能を組込み述語として用意した．また，FLOGS のこのほかの特徴としては，入出力機能の拡張，支援ツールの整備，効率的な単一化アルゴリズムの採用，およびインプリメント言語 FORTRAN による移植性のよさ等が挙げられる．

1. まえがき

述語論理型言語 PROLOG は，単一化によるリスト処理，後戻り制御による非決定性動作等の優れた性質により，知識情報処理向き言語として期待されている．一方，知識情報処理の重要な分野である画像情報処理や音声認識処理には，大量のデータに対する高速処理が要求され，従来から FORTRAN やアセンブラ等の手続き型言語を用いて多くのプログラムが作製され，種々のシステムが開発されている．こうした状況から，PROLOG のように知識の取扱いが容易で，かつ FORTRAN のように大量の数値計算を高速に処理できる知識情報処理向き言語の必要性が指摘できる．そこで，FORTRAN と PROLOG のリンクを可能にした新しい PROLOG システム FLOGS (FORTRAN-based PROLOG system) を設計し，その処理系を開発した．FLOGS では，FORTRAN サブルーチンを呼び出す組込み述語を用意し，PROLOG のゴールとして実行することによって両言語のリンクを可能にした．さらに，一元的大量データの取扱いならびに両言語間の融通をよくするために，PROLOG 側に新しい基本データ型として配列を導入し，リストと配列との変換を行う機能を組込み述語として用意した．

FLOGS のこのほかの特徴としては，入出力機能の拡張，支援ツールの整備，効率的な単一化アルゴリズムの採用等が挙げられる．またシステムは，FORTRAN を用いて実現されている．FORTRAN は，現在使用されている言語としては最も移植性に富み，各種のプログラミング，デバッギングツールが整備され，プログラムの改良，保守が容易であり，最適化処理による高速コンパイラをもつ高速な言語で，実用システムの記述言語として優れている．

以下，2章では FLOGS の言語仕様上の特徴と支援ツールの機能について述べる．3章ではその実現の面から，内部データ形式，インタプリンタと支援ツールの実現方式について述べる．4章では FLOGS の特徴を活かしたプログラム例を示し，システムの性能評価を行う．

2. 言語仕様と支援ツール機能

FLOGS の言語仕様は Edinburgh 版 Prolog-10[1] を基に機能の拡張を行ったもので，その拡張機能としては，1) 基本データ型配列の導入，2) FORTRAN 呼出し機能，3) 入出力機能の強化等が挙げられる．以下では，これらの特徴ならびに支援ツールについて述べる．

2.1 基本データ型配列の導入

従来の PROLOG では，二値画像のような一元的なデータ列はリストを用いて表現せざるをえなかった[2]．しかし，この方法では個々のデータへのアクセスに時間がかかり，また，その表現も同様のデータが羅列され，見やすさの点で劣っていた．そこで，

† FLOGS: An Extended PROLOG System for Procedural Processing by KAZUHISA KAWAI, MASASHI GANKE (Faculty of Engineering Science, Osaka University) and JUN'ICHI TOYODA (Institute of Scientific and Industrial Research, Osaka University).

†† 大阪大学基礎工学部情報工学科
††† 大阪大学産業科学研究所

```
:-  .                                   :-  .
    .                                       .
    .                                       .
  proc([_,_,_,_,                            .
        [_,_,_,_,X,_],                      .
        _],...),                            .
    .                                       .
    .                                       .
  X >= 0,                                   arr(5,5) >= 0,
    .                                       .
    .                                       .
    .                                       .
a) list version.                        b) array version.
```

図1 リストと配列を用いたプログラム例
Fig. 1 Illustrations of programs written using list and array.

FLOGSでは，基本データ型の一つとして配列を用意した．配列は一元的データ列表現のためのみならず，次節で述べるFORTRANとのリンクに際して，互いのデータ構造の整合をとるために導入したものである．

配列は複合項と区別するために，その使用に先立って，配列名と寸法（次元を含む）が，組込み述語decarrayによって宣言される．宣言された各配列の配列要素は，変数として機能し，値として任意の項をもち，他の項と同様に単一化が行われる．たとえば，

decarray (arr (10,10))

という述語を実行した場合，arrという配列名で，2次元の配列が宣言される．arrの寸法は，10×10である．この述語の実行後，たとえば，arr(3,4)という項が現れると，この項は，関数名arr，第1引数3，第2引数4の複合項ではなく，arrという配列名の配列の第(3,4)番目の要素がその項となる．

decarrayで宣言されたとき，すべての配列要素は，値が未束縛の変数である．また，各配列要素の値はsingle assignment[3]である．各配列要素のスコープは，他の変数のように一つの節内だけにはとどまらず，プログラム全体にわたり，いわゆるグローバルとなる．これは，配列に使用されるメモリ量の抑制と速度の向上のためである．一般に，配列で表されるようなデータはたいへん大きく，配列をローカルにすると，節ごとに配列用の領域がとられ，メモリ使用量が激増してしまう．また，配列の受け渡しのための単一化にも時間がかかってしまう．

配列の導入に伴い，配列とリストとの変換を行う組込み述語listarrayを用意した．すなわち，listarrayは二つの引数をとり，第1引数のリストと第2引数の配列との変換を行う．第1引数がたんなるリストの時は，第2引数は1次元の配列，リストのリストの場合は，2次元の配列となる．

図1に，リストと配列を用いた場合の簡単なプログラム例を挙げる．プログラムは，6×6の二次元配列の第(5,5)番目の要素の正負を判定するものである．図1(a)はリストを用いた場合で，正負の判定を行う述語よりも以前に，リストのリストを記述し，変数Xに第(5,5)番目の要素を取り出しておかなければならない．これに対し，図1(b)の配列を用いる方法では，正負を判定する述語を書くだけで済む．

2.2 FORTRAN呼出し機能

画像情報処理等の目的のために，FORTRANとPROLOGのリンクが可能な言語の設計を考える場合，両言語のリンクの方法として，次の二つが考えられる．①FORTRANからPROLOGを呼び出すようにする．たとえば，FORTRANプログラム中にPROLOGのサブゴールを書けるようにし，FOTRANからPROLOGを呼び出す．②PROLOGからFORTRANを呼び出すようにする．この2方法を比較すると，本来PROLOGとFORTRANのリンクの目的である，PROLOGによる知識の取扱いとFORTRANによる数値計算的処理の融合という面から，PROLOGが主導権をもってプログラムが動くべきであり，言語設計もそうすべきであるから，②の方法が優れていると考えられる．

次に②の方法において，いかにPROLOGからFORTRANを呼び出すかについても次の二つの方法が考えられる．①FORTRANのサブルーチンを呼び出す組込み述語を用意し，PROLOGプログラム中のあるルールのゴール（または，コマンド）に書けるようにする．②PROLOGプログラム中のあるルールのゴール（または，コマンド）の任意の位置にFORTRANの任意の実行文を書けるようにする．この2方法を比較すると，プログラムの記述性，記述されたプログラムの読みやすさ等の点で①の方法が優れている．FLOGSでは，①の方法を採用している．このFORTRANサブルーチンの呼出しを行う組込み述語fcallは，その引数が次の条件を満たすときに成功する．すなわち，FORTRANサブルーチンの呼出しの実引数が，数であるか，数に束縛された変数（配列要素を含む），または，未束縛の変数の場合である．この条件が満たされない場合は，FORTRANサブルーチンの呼出しは行われず，述語fcallも失敗に終わる．この条件が満たされている場合には，述語fcallの実行によって，FORTRANサブルーチンが呼び出

され実行される．FORTRAN のサブルーチン内で変数に値が代入されることがあるが，サブルーチンの呼出し時にすでに束縛されていた変数が，FORTRAN 側で再代入されても，それは PROLOG に戻ってきたときに呼出し時の値に戻される．これは，PROLOG のもつ変数の single assignment という原則を保持するためである．呼出し時に未束縛であった変数に対する FORTRAN での代入は，PROLOG での代入と同様に扱われる．後戻りが起こったとき，述語 fcall は再充足されることはない．また，FORTRAN のサブルーチン内で束縛された変数は，後戻り時にすべて未束縛に戻される．

2.3 入出力機能の強化

入出力機能は，その言語の実用性を考えた場合，最も重要な要素の一つである．FLOGS の入出力機能は，LISP 1.9[4] の入出力機能を参考にして強化されている．そのおもな特徴は，1)論理チャネルの導入，2)カレントチャネルの導入，3)入出力チャネルの分離，4)チャネル属性の充実等である．

2.4 支援ツールの機能

PROLOG は後戻り制御による非決定性動作を行い，その実行過程はかなり複雑なものとなる．そのため，逐次型（手続き型）言語等に比べて，いつ，どこで変数への代入や，値の比較等が行われたかを見きわめるのはむずかしい．この欠点を補うために，プログラムの実行状態を追跡し，逐次ユーザに知らせる支援ツールが必要となる．

FLOGS では，1)述語の引数値が単一化の前後でいかに変わるかを知る，2)プログラムの実行をユーザが自在に制御できる，3)デバッグを迅速に行うために，ユーザが指定した述語だけに注目するという要求を満たすような支援ツール機能を設計し，システムに組み込んである．トレース情報としては，Edinburgh 版 Prolog-10 で用いられている四つのポート[1),2)]におけるゴールの値と，単一化が行われたルールとが表示される．

3. システムの実現

FLOGS は，大型計算機 ACOS-1000 ならびにスーパミニコン MV-8000 の二つの計算機上に実現された．インプリメント言語はそれぞれ ACOS-6 FORTRAN[5], FORTRAN 77[6] である．

以下では，内部データ形式，インタプリタと支援ツールの実現，ならびに移植について述べる．

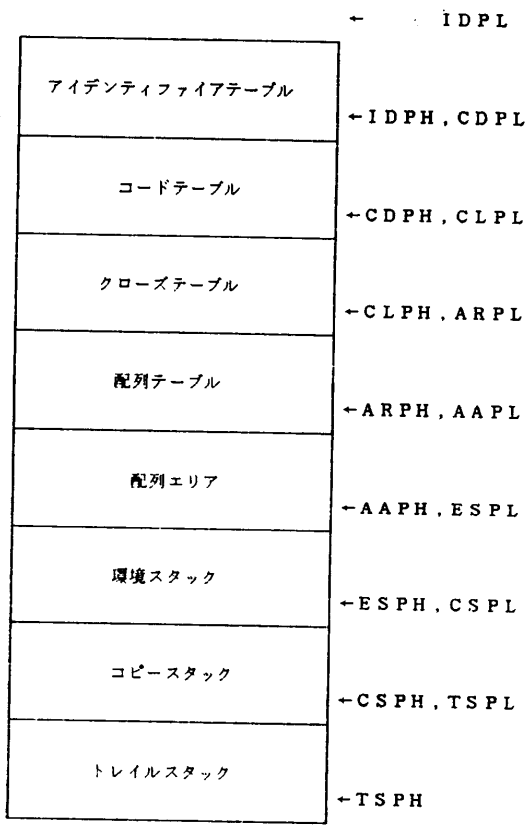

図 2 内部データのメモリ割付け
Fig. 2 Diagram showing memory layout of internal data.

3.1 内部データ形式とメモリ割付け

FLOGS に入力されたルール等は入力解析部で内部データ形式に変換され，内部データ領域に登録される．内部データは，その生成・削除時期によって大きく二つに分けられる．すなわち，個々のプログラムの実行中に書きかえが行われる動的データ，個々のプログラムの実行中には書きかえられず，ルールの読込みによってデータが蓄積されていく静的データの二つである．さらに，内部データは細かく 8 個のブロックに分類される．この 8 個のブロックは，**図 2** のようにメモリ上に連続して割り付けられ，必要に応じて各ブロックの大きさを変更できるようにしている．さらに，各ブロックのメモリ使用状況を表示する機能も用意しているので，ユーザはこれらを用いて，各プログラム向きにメモリを効率よく利用することができる．実際のインタプリタプログラムにおいては，メモリを一次元配列としているので，配列の添字がアドレスとなる．

次に，各ブロックのデータ形式について述べる．

I) アイデンティファイアテーブル 各アイデンティファイアについての情報を登録するテーブルである．テーブル上での登録される位置は，アイデンティファイアの印字名から得られるハッシュ値によって決まる．登録される情報は，

1) 印字名（テーブルへのポインタ）
2) クローズテーブルへのポインタ
3) 配列テーブルへのポインタ
4) 組込み述語番号

等である．

II) コードテーブル コードテーブルは，ルールやアサーションを含めたすべての項そのものを表す内部データである．コードテーブルは，項を構成する個個のアトムに対応するアトムコード部と，アトム間の親子関係（関数-引数関係）を与える親コード部，子コード部から成る．親子関係を双方向に表すのは，Paterson の単一化アルゴリズム[9]のためである．アトムコード部には，

1) アトムの種別
2) アイデンティファイアテーブルへのポインタ（定数のとき），番号（変数のとき），値（整数のとき），実数テーブルへのポインタ（実数のとき）
3) 親コード部へのポインタ
4) 子コード部へのポインタ

等の情報が登録される．

III) クローズテーブル クローズテーブルには，各節（ルール，アサーション）に対応する情報がおかれる．記録される情報は，

1) 節の先頭アトムである“:-”に対するアトムコードへのポインタ
2) 節の左辺の述語アトムのアトムコードへのポインタ
3) 同じ手続きを構成する次の節に対応するクローズテーブルへのポインタ
4) 節の右辺のゴール列の述語アトムのアトムコードへのポインタ列
5) その節に含まれる変数の数

である．

IV) 配列テーブル

V) 配列エリア

配列は，その使用に先立って，その配列名，次元，および寸法が宣言される．システムは，この宣言に基づいて，配列としての登録を行い，配列要素のためのデータ領域を確保する．

配列テーブルには，宣言された配列名（印字名テーブルのポインタ），次元，寸法，配列エリアへのポインタが登録され，配列エリアへのポインタが指示する位置以降の配列エリア領域に，配列の寸法分の配列要素のための領域がとられる．配列エリア上の配列要素は，コピースタックの変数と同じデータ構造で，宣言時には，すべて未束縛の変数として登録される．配列要素へのアクセスにおいては，配列テーブルの情報を用いて，配列エリア上でのアドレスを計算し，そのアドレスにある値を得る．

VI) 環境スタック FLOGSでは，単一化のためのデータ構造として，コピー方式[7]を用いている．この環境スタックには，変数の値と探索に関する情報の双方がおかれる．

VII) コピースタック コピー方式において生成される構造のコピーがおかれるスタックである．

VIII) トレイルスタック 後戻り時に未束縛の状態に戻さなければならない変数に対応する環境スタック，およびコピースタックのポインタがおかれる．

図3は，手続き append に対応する内部データ形式を表す．

3.2 インタプリタの実現

図4は，FLOGS インタプリタシステムの構成図である．端末やファイルから入力されたルールやアサーションは，各論理チャネルの入力バッファに蓄えられる．入力解析部は，入力バッファ中のデータに対して，オペレータ表現，リスト構造の複合項化を行い，さらに，内部データ形式に変換し，内部データ領域に登録する．

入力がコマンドやクエスチョンの場合には，ルールやアサーションの場合と同様に，内部データ領域への登録を行った後，その実行に移る．制御モニタは，最初に実行すべきサブゴールを決定し，それが，組込み述語の場合には，その組込み述語の処理を行うプログラムを呼び出す．そうでないときには，単一化部を呼び出し，単一化を行う．すべてのサブゴールが処理されると，制御モニタは出力生成部を呼び出し，実行結果を出力する．出力生成部は，入力解析部とは逆に，内部形式のデータを複合項形式に直し，さらに，オペレータやリストの表現形式にして，出力バッファを通して，端末やファイルに出力する．

組込み述語処理部から FLOGS システム外のユーザ作成の FORTRAN サブルーチンを呼び出しているのは，FORTRAN サブルーチン呼出し機能を示

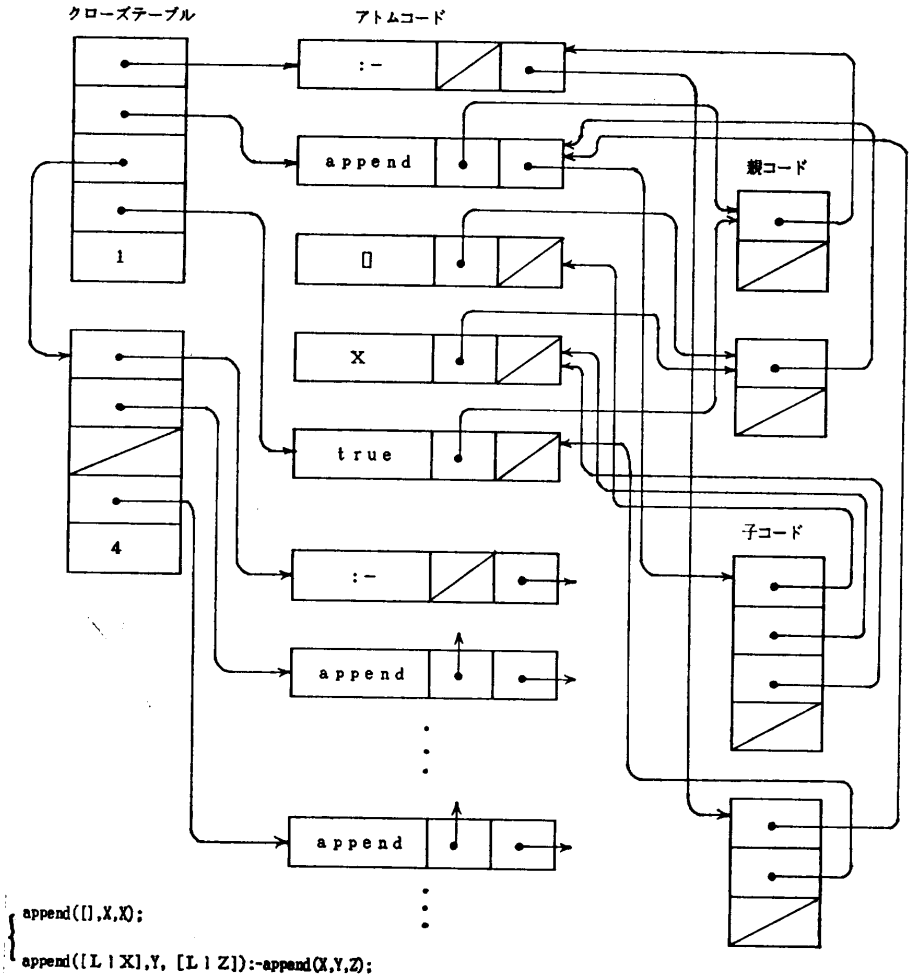

図 3 内部データ形式の例
Fig. 3 Illustration of internal data for "append".

す．FORTRAN 呼出し時には，サブルーチンの実引数として特定の FORTRAN の配列を用意し，この配列を介してサブルーチンを呼び出す．このとき，本来の実引数が変数であるか定数であるかを記録しておく．サブルーチンの実行後，配列を介して返される実引数の値のうち，呼出し時に変数であったものは，その値を元の変数に代入し，呼出し時に定数であったものは，返された値がいかなるものであっても元の変数に再代入しない．これによって，2.2 節に述べた変数束縛に関する機能を実現している．

FLOGS の探索メカニズムは，縦型，深さ優先方式である．探索のための情報は，環境スタックに記録される．環境スタックには，この探索のための情報と変数の値が，フレームと呼ばれる一つの固まりとして記録されている．1 フレームはちょうど一つの節に対応しており，その節が呼び出されたとき，すなわち，節の左辺がサブゴールと単一化が行われたときに生成される．フレーム中の変数の値がおかれる部分の大きさは，対応する節中に現れる変数の数となる．探索のための情報には 2 種類あり，一つはそのフレーム生成の際の単一化に関するもので，単一化の行われた親ゴールに対応するコードテーブルへのポインタと親ゴールを含む節に対応する環境スタックのポインタである．もう一つは後戻り時に用いる情報で，単一化の終了した時点でのコピースタックの最上部へのポインタ，トレイルスタックの最上部へのポインタ，および単一化の行われた節の（手続きの）次の節に対応するコードテーブルへのポインタ，そして，このフレームより以前に作られたフレームで後戻り時にそこへ戻っていくべき節に対応するフレームへのポインタで

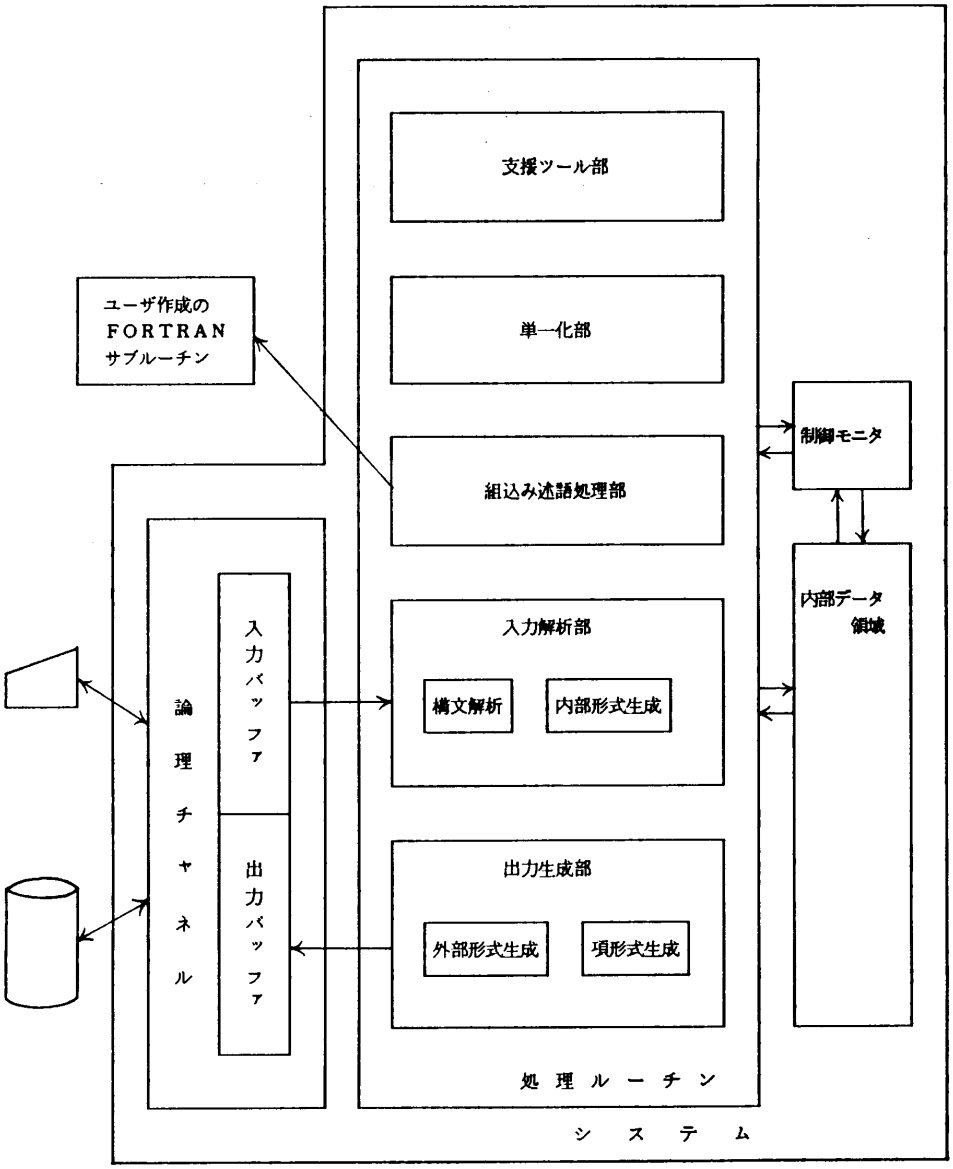

図 4 FLOGS インタプリタシステム構成
Fig. 4 Schematic representation of FLOGS interpreter system.

ある.

後戻りによって変数の束縛が破棄され未束縛の状態に戻される処理は，環境スタック，コピースタックのポップアップによって自動的に行われる．ポップアップされない位置にある変数に対する束縛の破棄は，トレイルスタックを用いて行われる.

従来の PROLOG システムの多くは，Robinson の単一化アルゴリズム[8]を用いている．このアルゴリズムは，単一化の行われる項が複合項同士の場合，各複合項の引数を左から右へ順にマッチしていくもので，引数に変数が現れるごとに，その変数の束縛を調べなければならず，この束縛のチェックのために，アルゴリズムは単一化の行われる項の大きさに比例した時間内に終了しない.

これに対して，Paterson ら[9]は，変数の「ある半順序」に従い，単一化の順序を決定することによって，変数の束縛のチェックをなくし，項の大きさに比例した時間内に処理が終わるアルゴリズムを開発した.

FLOGS では，この Paterson のアルゴリズムを PROLOG 向きに改良し，データ構造にコピー方式[7]を用いて実現している．データ構造に Structure Shar-

ing 技法[10]を用いずに，コピー方式を用いたのは，Paterson の単一化アルゴリズムとの整合性がよいと判断したからである．

FLOGS の入出力機能は，2.3 節で述べたように強化されている．これを実現するために，各チャネルの属性等を登録するチャネルテーブルと，実際に入出力を行う入口，出口となる入出力バッファを用いている．さらに，移植性を考慮し，入出力機能はすべて FORTRAN 言語上において実現し，OS 等に依存しないようにしている．

3.3 支援ツールの実現

トレーサ，ステッパ等のユーザ支援ツールの実現には，ユーティリティプログラムとして PROLOG で作製する方法と，インタプリタプログラムの一部としてシステムに組み込む方法がある．FLOGS では，速度およびステッパでの実行の制御の自在度の両面から，インタプリタプログラムの一部として FORTRAN で記述しシステムに組み込んだ．

3.4 移　　植

すでに述べたように，FLOGS は移植性を考慮して FORTRAN でインプリメントした．FLOGS は，まず ACOS-6 FORTRAN[5] を用いて，大阪大学大型計算機センターの大型計算機 ACOS-1000 上で開発された．次に，スーパミニコン MV-8000 上の FORTRAN 77[6] への移植を行ったが，移植に要した期間は約 2 週間で，インプリメント言語 FORTRAN の移植性のよさが確認された．

4. システムの性能評価

本章では，FLOGS の特徴である FORTRAN 呼出し機能の有用性について，画像処理に用いられる基本的なアルゴリズムを例に，その性能評価を行う．

4.1 実験内容

画像処理の基礎的なアルゴリズムである周辺分布処理[11]，細線化処理[11]を，Ⓐ PROLOG のみ，Ⓑ PROLOG から FORTRAN 呼出しの 2 種の方式で記述し，それぞれの実行時間を比較する．プログラムの条件としては，リストのリストで表された二値画像に対

```
test1(List) :-
   cputime(X),
   projection(List,XA,YA),                    Ⓐ PROLOG のみ
   cputime(Y),
   nl,
   writec('projection test 1'),nl,
   write(XA),nl,
   write(YA),nl,
   Z is Y - X,
   writec('CPU time '),write(Z),writec(' Msec'),nl;
projection(List,Xans,Yans) :-
   zerolistX(List,XA),
   zerolistY(List,YA),
   proj1(XA,YA,List,Xans,Yans);
proj1([],YA,[],[],YA);
proj1([XAH|XAT],YA,[H|T],[XA1H|XA1T],YA2) :-
   proj2(XAH,YA,H,XA1H,YA1),
   proj1(XAT,YA1,T,XA1T,YA2);
proj2(XA,[],[],XA,[]);
proj2(XA,[YAH|YAT],[LH|LT],XA2,[YA1H|YA1T]) :-
   YA1H is YAH + LH,
   XA1 is XA + LH,
   proj2(XA1,YAT,LT,XA2,YA1T);
zerolistX([],[]);
zerolistX([H|T],[0|X]) :-
   zerolistX(T,X);
zerolistY([H|T],Y) :-
   zerolistX(H,Y);
/*   */
test2(List) :-
   cputime(X),                                Ⓑ PROLOG から FORTRAN 呼出し
   list_size(List,Xsize,Ysize),
   decarray(xa(Ysize)),decarray(ya(Xsize)),decarray(larr(Xsize,Ysize)),
   listarray(List,larr),
   fcall(proj(larr,xa,ya,Xsize,Ysize)),
   listarray(XA,xa),listarray(YA,ya),
   cputime(Y),
   nl,
   writec('projection test 2'),nl,
   write(XA),nl,
   write(YA),nl,
   Z is Y - X,
   writec('CPU time '),write(Z),writec(' Msec'),nl;
list_size([H|T],X,Y) :-
   length(H,X),
   length([H|T],Y);
```

```
      SUBROUTINE PROJ(IN,OUT1,OUT2,XSIZE,YSIZE)
      IMPLICIT INTEGER (A-Z)
      DIMENSION IN(XSIZE,YSIZE),OUT1(YSIZE),OUT2(XSIZE)
C
C                                          FORTRAN サブルーチン
C
      DO 100 I=1,XSIZE
  100 OUT1(I)=0
      DO 200 I=1,YSIZE
  200 OUT2(I)=0
      DO 300 I=1,YSIZE
        DO 300 J=1,XSIZE
          OUT1(I)=OUT1(I)+IN(J,I)
          OUT2(J)=OUT2(J)+IN(J,I)
  300 CONTINUE
      RETURN
      END
```

```
? ?- test1([[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]]);
projection test 1
[4,4,4,4]
[4,4,4,4]                                  Ⓐ 実行例
CPU time 287 Msec

yes
? ?- test2([[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]]);
projection test 2
[4,4,4,4]
[4,4,4,4]                                  Ⓑ 実行例
CPU time 19 Msec

yes
```

図 5 周辺分布処理プログラムと実行例
Fig. 5 Program of projection and its execution example.

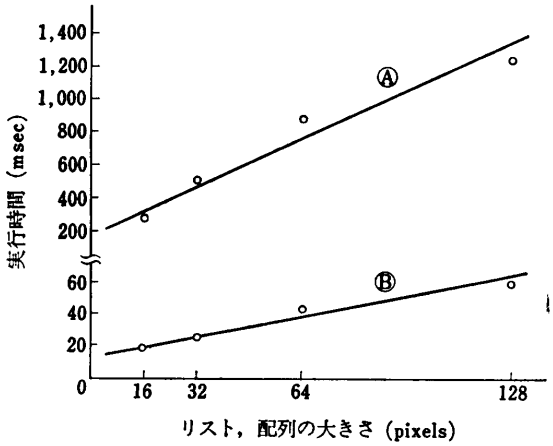

図 6 ⒶPROLOG のみ，ⒷPROLOG から FORTRAN 呼出し両方式の周辺分布処理の速度比較
Fig. 6 Comparison of CPU times to execute projection programs written in PROLOG (Ⓐ) and written in PROLOG with FORTRAN-call (Ⓑ).

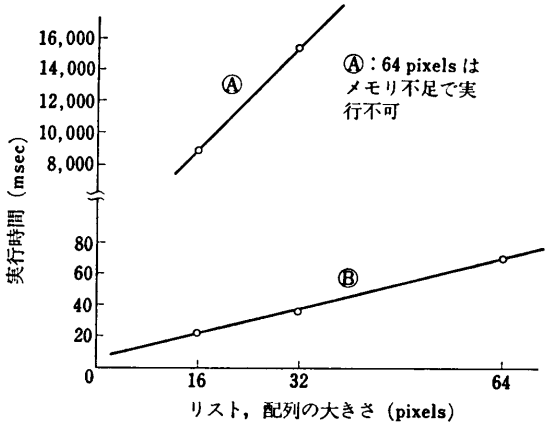

図 7 ⒶPROLOG のみ，ⒷPROLOG から FORTRAN 呼出し両方式の細線化処理比較
Fig. 7 Comparison of CPU times to execute thinning programs written in PROLOG (Ⓐ) and written in PROLOG with FORTRAN-call (Ⓑ).

して，それぞれの処理を施し，その結果をリストとして返すものを考える．したがって，Ⓑの方式の場合，処理の前後でリスト・配列変換が含まれる．

4.2 周辺分布処理

リストのリストで表された二値画像に対して，縦方向，横方向の黒画素の総和を求める処理を行う両方式のプログラムとその実行例が**図5**である．また**図6**のグラフは，与えられた二値画像の大きさ（縦×横）と，両方式の実行時間の関係を表す．図6よりわかるように，両方式の速度比は二値画像の大きさに関係なくほぼ一定で，約20倍ほどⒷの方式が速い．これはいずれのプログラムも二値画像の大きさに比例したオーダの計算量であることによる．

4.3 細線化処理

細線化処理は，データ量の削減や追跡の容易さ等のために，太い線を細い線に変換する操作のことで，ここでは Hilditch の細線化法[12]を用いる．実験は，リストのリストで表された二値画像に対して細線化を行い，その結果をリストのリストとして返すプログラムをⒶ,Ⓑの両方式で作成し，その実行時間を比較するものである．**図7**は二値画像の大きさと実行時間の関係を表すグラフである．両方式の速度比は，約400倍で，Ⓑの方式が速い．

4.4 結　　果

周辺分布処理，細線化処理の実験結果より FORTRAN 呼出し機能によって，実行効率が著しく改善されることが示された．FORTRAN 呼出し機能を用いたプログラムは，用いないプログラムの数10倍以上の速度をもち，速度向上は複雑なプログラムほど著しい．

プログラムの可読性としては，プログラムの手続き的な部分が FORTRAN サブルーチンの中に隠され，PROLOG で書かれた部分は PROLOG 本来の可読性のよさを保持している．

5. むすび

手続き的処理と知識の融合に関して，PROLOG と FORTRAN のリンクに対する一つの考えを提示し，これに基づいた FORTRAN とのリンク機能，および新しいデータ型配列の導入等の拡張機能をもつ新しい PROLOG システム FLOGS について述べた．性能評価の結果に見られるように，FORTRAN サブルーチン呼出し機能によって得られる実行効率の改善は著しい．また，ACOS-1000 から MV-8000 への移植の容易さは，FLOGS のインプリメント言語 FORTRAN の移植性のよさを示すものである．

現在，FLOGS を用いて，新聞紙面の自動割付けシステム NEES[13] の作製を行っている．この研究を通じて，より有用な組込み述語の充実を図ること，および，PROLOG 構造向けエディタ，高速コンパイラの開発等のプログラミング環境の充実が今後の課題として挙げられる．

謝辞 日頃，有益なご教示をいただく大阪大学産業

科学研究所角所収教授，熱心なご討論，有益なご助言をいただく同上原邦昭助手，溝口理一郎助手，ならびに山口高平助手に深く感謝します．また，スーパミニコン MV-8000 の使用に際し，種々のご協力をいただいた同角所研究室の諸氏に深謝します．

参考文献

1) Pereira, L. M., Pereira, F. C. N. and Warren, D. H. D.: USER'S GUIDE to DECsystem-10 PROLOG, Lisboa, Outubro de 1978 (1978).
2) Clocksin, W. F. and Mellish, C. S.: *Programming in Prolog*, Springer, New York (1981).
3) Nilsson, N. J.: *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York (1971).
4) 黒川利明：LISP 1.9 プログラミングシステム，情報処理，Vol. 17, No. 11, pp. 1056-1063 (1976).
5) 日本電気(編)：ACOS-6 プログラム管理 FORTRAN 文法説明書 (1974).
6) 日本データゼネラル (編)：AOS/VS FORTRAN 77 解説書 (1980).
7) Bruynoogh, M.: The Memory Management of PROLOG Implementations, Proc. of Logic Programming Workshop, pp. 12-20 (1980).
8) Robinson, J. A.: A Machine-oriented Logic Based on the Resolution Principle, *JACM*, Vol. 12, No. 1, pp. 23-41 (1965).
9) Paterson, M. S. and Wegman, M. N.: Linear Unification, *JCSS*, Vol. 16, pp. 158-167 (1978).
10) Boyer, R. S. and Moore, J. S.: The Sharing of Structure in Theorem Proving Programs, in Meltzer, B. and Michie, D. (eds.), *Mach. Intell.*, Vol. 7, pp. 101-116, Edinburgh Univ. Press, Edinburgh (1972).
11) 安居院猛，中嶋正之：コンピュータ画像処理，産報出版，東京 (1979).
12) Hilditch, C. J.: Linear Skeltons from Square Cupboards, in Meltzer, B. and Michie, D. (eds.), *Mach. Intell.*, Vol. 4, pp. 403-420, Edinburgh Univ. Press, Edinburgh (1969).
13) 成田健一，河合和久，豊田順一：英字新聞の割り付けを行なうエキスパートシステム NEES について，第 26 回情処全大 (1983).
14) 河合和久，豊田順一：FORTRAN 呼出し機能を持つ PROLOG：FLOGS について，信学技報 AL 82-90 (1983).

(昭和 59 年 4 月 5 日受付)
(昭和 59 年 7 月 19 日採録)