記号処理 18－7
（1982. 3. 25）

ロジック・プログラミングと高機能パーソナル・コンピュータ

横井俊夫，佐藤泰介，元吉文男，新田克己，梅山伸二
大谷木重夫（電総研），後藤淑樹（横須賀通研），中島秀之（東大），
松田利夫（理科大），白石富勝（シャープ），黒川利明（埼芝），梅村
護（日電），國藤 進（富士通・国際研），林 弘（富士通研），上田謙一（松下）

1．はじめに

Prologを代表格に，ロジック・プログラミング（論理型）言語も方々で処理系が試作され，多くのプログラムが書かれ，様々の経験が蓄積されつつある。その有用性が具体的に明らかにされるとともに，インプリメンテーション法，言語仕様の拡張，理論的課題等，多くの新しい研究テーマを派生し，今後の展開に大きな期待が持たれている。

高機能パーソナル・コンピュータ（スーパ・パーソナル・コンピュータ，単一ユーザ用高機能コンピュータ）は，米国を中心に，色々なタイプのものが研究開発され，商品化されるものもでてきた。色々なタイプの中で，より小型のものは高機能なOA機器として，大型のものは高度な（人工知能）システムの開発用，特に，今後の主要な情報処理の研究開発用ツールとして試作されている。

第五世代コンピュータ・プロジェクトにおいてもこの2つの動向に着目し，重要な研究・開発課題として位置づけている。一つは，システムの中心をなすプログラム言語を，とりあえず核言語と名付け，これをロジック・プログラミング言語を母体にし，知識情報処理に向け高度化を計ったものとして研究・開発を行う。もう一つは，研究・開発用ツール特に様々なソフトウエアの試作ツールとして，より一段と高機能化したパーソナル・コンピュータを開発する。このコンピュータを核言語向きとすることにより，全体システムの研究・開発をいくつかの独立したサブ・プロジェクトに分解することが可能になる。

著者らは，以上のような位置付けに基づき，既存の論理型言語や，Lisp等，記号処理的言語等々と，Lispマシンをはじめとする高機能パーソナル・コンピュータを参考にして，核言語向な高機能パーソナルマシン（'FGKL/S'マシン）の先行研究を行った。本稿でその結果の一部を紹介する。

まだ，問題点の列挙とイメージ固めの段階であるため，全体としての整合性がきちんととられている状態ではない。第2章では，システムの全体構成を中心に紹介し，第3章では，基本メカニズム部分を少し詳しく紹介する。

2．'FGKL/S'(注)マシンの構成

システム全体の構成を図1に示す。大きく3つの部分に分かれる。基本メカニズムと3つのサブ・システムとユティリティ・サブ・システムとである。基本メカニズム部分は，さらに図2に示すような構造を持つ。ユティリティを入れた4つのサブ・システム群は，図3に示すような構成となる。ハードウエア化の対象となるのは，基本メカニズム部分と，データベース，計算，インタフェース，各サブ・システムの一部である。

基本メカニズム部は，次章にゆずるとして，サブ・システム群を図3に従って

注）Fifth Generation Kernel Language／Sequential typeの略で，先頭の'（クォート）は，まだ評価（eval）を受けていないことを示す。

説明する。

○データ・ベース・サブ・システム

内部データ・ベース：主記憶上に構成されるデータ・ベースで、計算実行に必要な論理式（定義体）が蓄えられる。論理式の高速な検索を可能にするインデックス機構が中心となり、この機構は、ハードウエア化されたハッシング・メカニズムに支えられている。この機構により、論理型言語の大きな特徴であるデータ・ベースの検索とプログラムの呼びが一体化されたメカニズムが提供される。階層的なデータ・ベースを構成するコンテクスト機構により、多数の世界を表現することができる。また、ユーザの定義する通常の論理式に対する様々な属性も、同じく述語の形式で蓄えられデバッグ等に利用される。

外部データ・ベース：論理型言語と関係データ・ベースとが共通の土台の上に成り立っていることは良く知られている。第五世代コンピュータ・プロジェクトでは、ここで述べる'FGKL/Sマシンとは別に、本格的な関係データ・ベース・マシンをほぼ同じ時期に試作し使用実験することを計画している。この関係データ・ベース・マシンは、単体として機能することも考えるが、'FGKL/Sマシンの外部データ・ベース機器としての役割に重点を置いて開発する。これにより、内部データ・ベースの機能では処理しきれない多量データに対する検索が可能になる。関係論理型と関係代数型の両方向からインタフェースを考慮している。基本となる機能は集合概念と集合の演算操作である。

ファイル（入出力）システム：一般のファイルに対する入出力をつかさどる機能とインタフェース・サブシステムも含めてすべての入出力に対する共通のインタフェース機能を提供する。ユーザは概念チャネルというものを対象にして、すべての入出力を考えれば良いようになる。概念チャネルは、チャネル変数として扱われる。多数の概念チャネルが設定でき、各チャネルには属性と機能が割り当てられる。属性とは、このチャネルにはどのようなデータがどのような形態で流されてゆくかを表わす。機能とは、このチャネルが、入力としてか出力としてか、あるいはトレース用、エディタ用等の使われ方を表わす。概念チャネルの利点は、属性と機能を独立させ、同一のデバイス（物理的ファイル）上に複数のチャネルを設定できることによって、入出力のトラブルの原因となる副作用の混乱を整理する点にあ

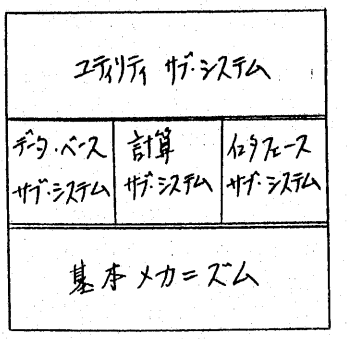

図.1 システム全体構成

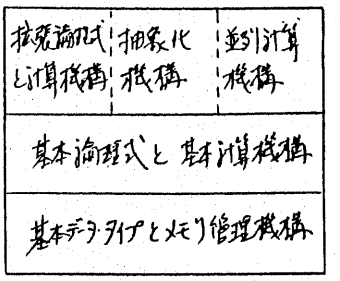

図.2 基本メカニズム

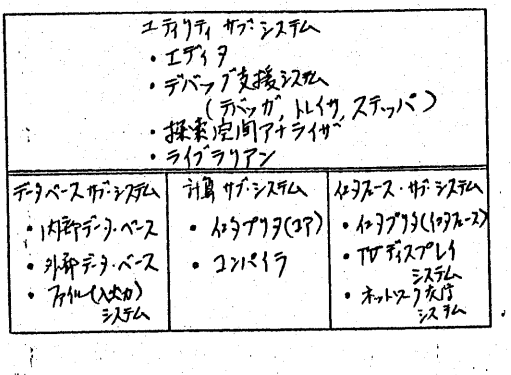

図3 サブ・システム群

る。

◦計算サブ・システム

インタプリタ（コア部）：インタプリタの核となる部分で、ほとんどがハードウエア化される。ただし、内部状態を参照したり、さらに変更したりの組込み述語が用意される。デバッグ支援システムは、この述語を用いインタプリタとのやりとりを行う。またインタプリタの拡張、変更を行うこともできる。

コンパイラ：FGKL/Sマシンの機械語は、できるだけ高いレベルに設定するのが望ましい。詳しくは次章で紹介する。したがって従来のプログラム言語に比べ、コンパイラのはたす役割は相対的に小さくなる。

◦インタフェース・サブ・システム

インタプリタ（インタフェース部）：端末等から入力された文字列データとしての論理式を内部のデータ表現に変換し、そのデータを処理する適切なサブ・システムを呼び出す。文法解析を行い、前置型の標準型に変換したり、変数や定数（シンボル）を一意的な内部アイデンティファイアに変換したり、数値データや構造体データの内部データ構造への変換を行う。なお文法解析を行うための文法規則は、ユーザが自由に変更できるようになっている。しかも文法規則も論理式と同じ形式で扱えるようになっている。漢字・カナ文字が標準文字セットとして扱えることは当然であるが、自然言語としての文に対する文法解析機能も組込むことを想定している。

TVディスプレイ・システム：このサブ・システムの機能の充実の度合が、このマシンの使い勝手を大きく左右することになる。ハードウェアとしてもソフトウェアとしても最も多くの作業を必要とする部分である。システムの内部構成は、Lispマシン等を参考として設定した。マルチ・ウィンドの概念が中心となり、高度なグラフィック機能が備わっている。実際のTV画面に比べ、ウィンドは論理的（概念的）な画面である。多数のウィンドを設けることができ、ウィンド間に階層的な関係を定義することができる。これらは実行時に動的に変化させることができる。ウィンドは概念チャネルに結びつけられることによりユーザの利用が可能になる。さらに、ウィンドの中にいくつかのサブ・ウィンドを作ることができる。

この機能は、会話型のシステムの作成に大いに効力を発揮する。あるウィンド上に表示されているプログラムの処理の結果、エラーが生じたとする。その時、サブ・ウィンドをいくつか設け、メニューの選択や、エラー処理用の会話をもとの画面を保持したまって行うことができる。

ネットワーク交信システム：パーソナル・コンピュータは、計算機ネットワークの中にあってその価値は倍化される。まず、Ethernetのようなローカル・ネットワークに接続できる標準的なインタフェースを持ち、基本的なプロトコールが用意される。

◦ユティリティ・サブ・システム

エディタ：プログラムやテキスト・ファイル作成のために、高機能な編集機能を持つ。TVディスプレイシステムの機能を活用した画面エディタである。編集モードとしてプログラミング・モードとテキスト・モードが設けられる。プログラミング・モードでは、言語の文法を考慮した編集機能や画面上への表示機能が利用できる。テキスト・モードは、和文、英文等の自然言語テキストを扱うモード

で、語、文、パラグラフ、ページ等の単位を意識して編集操作が可能である。さらに図形情報に対する基本的な編集機能も用意される。テキスト・ファイルを印刷物の形に整えるラン・オフ機能や、図面や多種類のフォントを利用できる出版システムも用意される。

デバッグ支援システム：ユーザの指示に従い、ワン・ステップずつ実行し、内部状態を表示してみせるステッパ、指定された変数や論理式の履歴を集めるトレイサ、割込みによって実行を中断し、内部状態を観察したり、変更して実行を再開したりするデバッガ等が用意される。バック・トラックによる選択はデバックを難しくするので、強力な支援システムが必要である。いずれにしろ、ユーザにバグを発見しやすくするように十分な情報を見易いレイアウトで表示してやらねばならない。例えば、その情報には、呼び出し側のパターン、どの選択肢が選ばれたか、変数の束縛状況、呼び出しの深さ、バック・トラックの状況、実行の履歴等がある。

探索空間アナライザ：ある呼び出しが失敗となったら、その失敗の原因が取り去られるところまでバック・トラックしなければ成功とはならない。原因となっているところまでのバック・トラックによる試行錯誤は無駄な選択を繰返すことになる。そして、あるリテラルの失敗の原因を生む可能性のあるのは、同じ変数を含む既出のリテラルである。節内の変数間の依存関係が失敗の因果関係を表わすことになる。この性質を利用して、不必要なバック・トラックを省略しようという手法が知的（選択的）バック・トラックと呼ばれているものである。この手法を用い効率の良いインタプリタを作成しようという提案もあるが、ここでは探索空間のアナライザとして活用し、プログラムの最適化を進めるためのツールとして活用する。知的バック・トラックモードで実行すると、失敗が生ずるたびにその失敗を回復できる所までの戻りの距離や探索木の状態が表示される。ユーザは、探索空間が小さくなるようプログラムの改良を行う。

ライブラリアン：プログラム・ライブラリやドキュメントの管理を行い、ユーザの問い合せに対し適切な答えを返す機能を持つ。将来はアルゴリズム・バンクやコンサルタントシステムとして、知識情報処理の典型的なシステムとして発展していくであろうが、当面は単純な機能で実現する。

以上がサブ・システムの機能の概略説明である。なお、ハードウェアに対する基本仕様は大よそ次のようなものと考えている。

- 数MIPSの処理速度を持つCPU
- 実数MW以上、仮2^{32}以上の主記憶
- 数100 MBのウィンチェスタ・タイプのディスク
- 2000×2000（8W）あるいは1000×1000（kGB）のTVディスプレイ　等々

3. 基本メカニズム

3.1 基本データ・タイプとメモリ管理

この部分には、あまりロジック・プログラミング特有というものは少ない。むしろLispマシンも含め、すべての記号処理機能を有する汎用コンピュータが共通に持たねばならないメカニズムの部分である。

- 基本データ・タイプ（図4）

変数を基本データ・タイプとするところが特徴といえる。ベクタは、一次元にある種のデータがならんだ任意長のデータ構造で、動的に生成しうるものである。

したがって構造内へはランダムな参照が可能である。構造内のデータの種類(ポインタ，文字，ビット)によって3つのサブ・タイプに分かれる。ポインタとは変数、定数、整(実)数、多倍長数やベクタ本体へのポインタ等のことで、いわゆる1語で表わされる。さらにポインタ・ベクタは、一般のもの以外に埋込みベクタとハッシュ・ベクタのサブ・タイプが用意される。埋込みベクタは図5に一例を示すように、木構造状になる通常タイプ(a)のもののポインタ部にベクタ本体を埋込み線形化したもの(b)である。したがって埋込みベクタは、ランダムな参照は意味が無くなり、先頭からの逐次参照のみが意味を持つことになる。ハッシュ・ベクタは、キーとして与えられてポインタをハッシュすることにより、位置決めを行うベクタである。リストはLispと同様のものである。

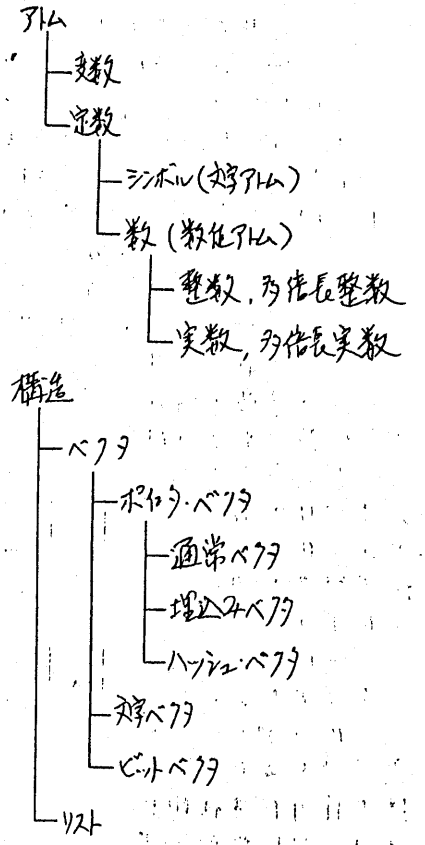

図4 基本データ・タイプ

これらの基本データ・タイプに対し、判定、変換、生成、演算等を行う組込み述語が用意される。

。領域管理機構

各基本データ・タイプ毎の割当領域を管理し、要求に応じたメモリ量を配分する。自由領域が無くなるとゴミ集めを行う。ゴミ集めの起動や統計データの問い合せ、領域の使用状態の問い合せ、領域間の再分割、データ・セルのALLOC、FREE等の組込み述語が用意される。

。アトム管理機構

変数、シンボル定数の外部表現を一意的な内部アイデンティファイアに変換し、関連する情報の管理を行う。外部から内部、内部から外部への変換の指示、新たな内部アイデンティファイアの生成や削除、状態の問い合せ等の組込み述語が用意される。

。メモリ階層機構

仮想メモリを実現する。ただし一般的なワン・レベル記憶の上に領域管理機構を単純に上乗せするだけでは、ゴミ集め等の効率が悪くなる。領域管理の仕組みを前提とした仮想メモリを考案する必要がある。

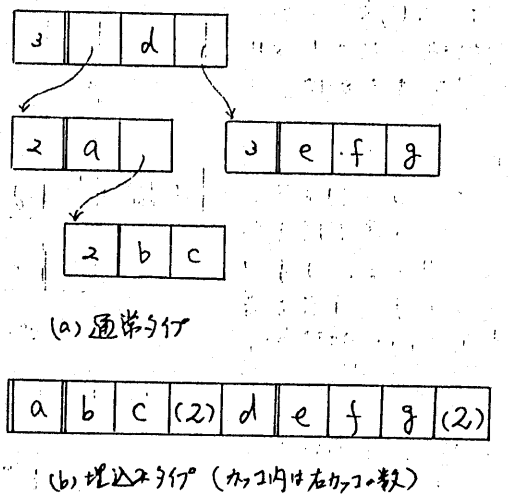

図5. ベクタの例

3.2 基本論理式と基本計算メカニズム

基本となる節は

ガード無し $P \leftarrow g_1, g_2, \cdots, g_n$

ガード付き $P \leftarrow g_1, g_2, \cdots, g_i \mid g_{i+1}, \cdots, g_n$

の2種類を考える。ガード付きの場合、"|"

の左部分をガード部、右部分を実行部と呼ぶ。各g_iは、リテラルか、OR節か、NOT節である。P部の述語名を同一とする節の集合を定義体と呼ぶが、定義体に含まれる節は、全てがガード無しか全てがガード付きでなければならない。定義体の節は、当面は書かれた順番に選択されていくものとする。

ガード無しの場合は、定義体の1つの節が選択されると、その右辺の$g_1, g_2 \cdots$が順番に実行され、すべてが成功すると節が成功したことになる。ガード付きの場合は、ガード部が成功し、続いて実行部が成功すると節は成功する。ただし、ガード部が成功し、更に実行部の実行を開始した場合、この節は選択されたと呼ばれ、同じ定義体に含まれているこの節以外の節の実行を禁止する。これはカット・オペレーションのガード付命令的解釈である。

定義体をいくつか集めて、モジュールとすることができる。モジュールには名前が付けられ、モジュール内の定義体にはそれぞれglobal(exported)かlocalの宣言がなされる。外部からの参照(呼び出し)が許されるのはglobalと宣言した定義体だけである。モジュール名で修飾した述語名を用いることによって、他のモジュールを参照することができる。モジュールのプレートの定義法とそれから実体を作りだす仕組により、対象指向的考え方も導入される。

'FGKL/Sマシンの機械語をどのレベルに設定するかは、まだ色々と議論のあるところである。ここでは一つの例を用い、考え方の基本的方向を説明する。まずコンパイラの介在も考慮して高い処理効率となる機械語としなければならないが、できるだけ高いレベルに設定するのが望ましい。その方が選択の巾も縮まるし、

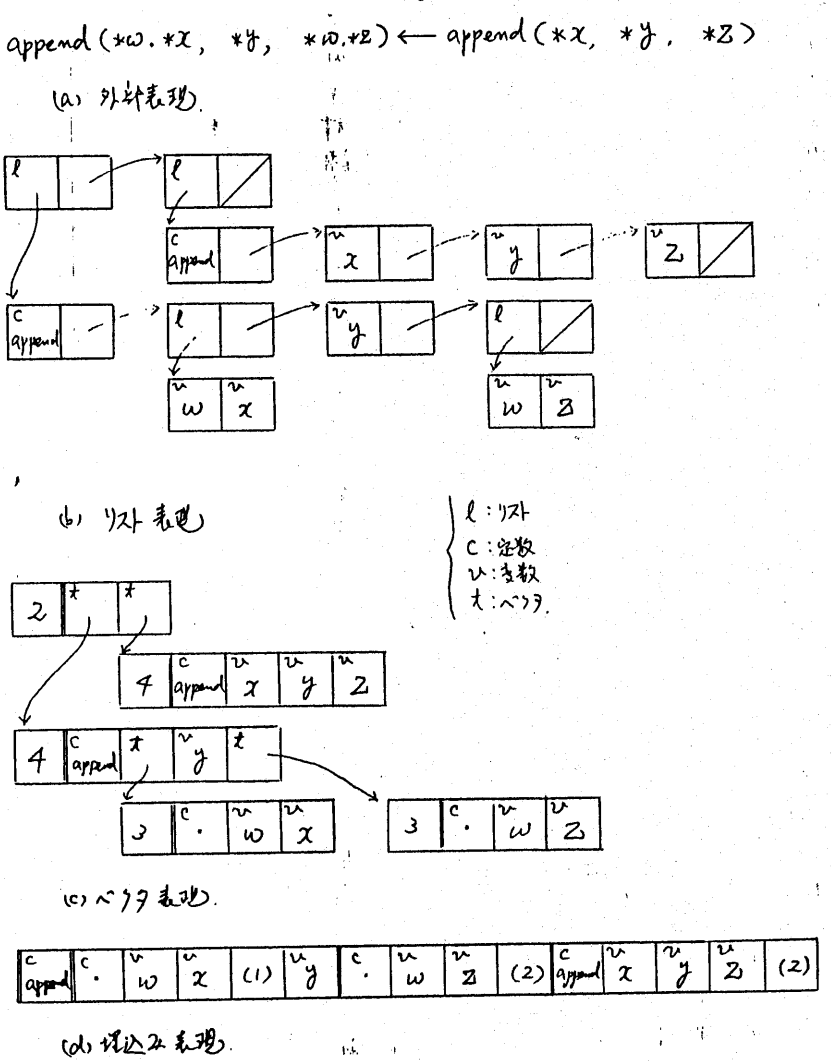

図.6. 論理式の表現

アーキテクチャもすっきりしたものとなるからである。例えば、どのように設定できるかを図6を用いて説明する。(a)が外部表現である。この内部表現に対して、(b)、(c)、(d)が考えられる。リスト表現は実行に際して、構造をたどるには一番手間取るが、変更等に関しては一番柔軟に対応できる。通常ベクタから埋込みベクタに移るにつれ、構造のたどりは楽になるが、変更には対応できなくなる。(b)ないし(c)をインタプリタが実行するプログラムとし、(d)を機械語のレベルと考えることができる。埋込み表現では、先頭から順番に読み込みながらパイプライン的にユニフィケーションを行うことができる。なおappendが入れられる所には、組込み述語であれば述語コードが、localな述語であればその述語ベクタへのポインタが、globalであれば内部idが置かれることになる。変数も自然数に正規化され、レジスタ番号にも対応づけることができる。(d)の表現にさらに複合項が無いという条件をつけると機械語の構造はより簡単になる。しかし、複合項の処理を記述するいくつかの組込み述語を用意しなければならない。

4. おわりに

以上は、筆者らの作業成果の一部の紹介である。整合性のとれた仕様とするためには、後半年位の作業が必要と思われる。富士通・国際研の安達統衛、竹島卓、加藤昭彦、沢村一、松下電器の安川秀樹の諸君に感謝し、結びとする。

なお、参考文献等は近日出版される報告書を参照されたい。