TM-0078

情報処理学会第29回全国大会発表論文集

(37件)

ICOT ならびに再委託先メーカ

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191－5
Telex ICOT J32964

Institute for New Generation Computer Technology

逐次型推論マシンψのファームウェア

—— マイクロインタプリタの概要と特徴 ——

3-1

山本 明　横田 実　西川 宏　渡 和男　内田 俊一
（新世代コンピュータ技術開発機構）
中島 克人（三菱電機）　三井 正樹（沖電気）

1．はじめに

パーソナル逐次型推論マシンψ（PSI）は第5世代コンピュータシステム研究開発の一環として開発されているソフトウェア開発用のパーソナルマシンである。

ψではKL0と呼ばれるPrologライクな論理型言語を機械語としており、このKL0をマイクロプログラムで記述されたインタプリタ（マイクロインタプリタ）と専用のハードウェアにより直接解釈実行する。本論文ではマイクロインタプリタの特徴と概要を報告する。

2．KL0

ψ上で実行されるプログラムはOSを含めて全て論理型言語で記述されることになっており、OSを効率良く実行するためKL0にはPrologライクな機能の他に低レベルのハードウェアを操作出来る機能や拡張された制御機能が含まれている。 KL0で記述されたクローズ表現は、コンパイラによりクローズヘッド部、ヘッドの引数部及び複数のゴールからなるボディゴール部から構成されるワード列のテーブル（内部形式）に展開される。マイクロインタプリタはこのテーブルを順次解釈しながら処理を行なう。

3．マイクロインタプリタの特徴

ψは論理型言語KL0及びその上に構築されたOSを効率良く実行するための各種の方式を採用している。

a．スタック方式とストラクチァ・シェアリング

KL0プログラム実行の基本的な動作はクローズの呼び出し、呼び出し元へのリターン及びユニフィケーションに失敗した際のバックトラック等の実行順序に関するものと、ユニフィケーションのようにデータ操作に関するものである。 これらの動作を実現するため、マイクロインタプリタは4本のスタックを使用して実行環境の管理を行なっている。 4本のスタックはそれぞれローカルスタック、グローバルスタック、トレイルスタック、コントロールスタックと呼ばれる。ローカルスタック、グローバルスタックは変数セルの格納用として、トレイルスタックは変数セルの無効化用情報の格納エリアとして、又コントロールスタックはクローズの実行順序を制御する情報の格納用として使用される。

ユニフィケーションで操作される構造体のデータ表現の形式として構造を共有するストラクチァ・シェアリング方式と構造を新しくコピーするストラクチァ・コピー方式とがありそれぞれ長所と短所もつが、ψでは仮想記憶をサポートしていないこと、ストラクチァの共有によるアクセスのオーバヘッドの方がコピーによるオーバヘッドよりも少ないとの予想からストラクチァ・シェアリングの方式を採用している。

b．テイル・リカージョン・オプティミゼイション

ユニフィケーションは呼び出し側ゴールの引数と呼び出された側のクローズヘッドの引数との間で行われる。しかし、これは呼び出し側での処理と呼び出された側での処理の2つのフェーズに分解して考えることが出来る。マイクロインタプリタはユニフィケーションに先立って呼び出し側ボディゴールの引数を評価し、呼び出される側の変数セル領域にその値をコピーした後、このコピーされたセルの値と呼び出された側のクローズヘッドの引数との間でユニフィケーションを行なう。 このようにユニフィケーションを呼び出し側ゴールの処理と呼び出されたクローズヘッドの処理とに分離することにより、クローズヘッドのユニフィケーションが失敗し他の選択肢のユニフィケーションを行なう際にも以前に生成した呼び出し側の引数セルの値をそのまま使用でき、処理の高速化を計ることができる。

又、クローズの最後のボディゴールの呼び出しでは、そのクローズにもはや他の選択肢が存在せず決定的に処理が終了する場合そのクローズの変数セル領域は保存する必要がない。したがって、マイクロインタプリタはこのようなゴールの処理では呼び出し側の引数のコピーを完了した時点で呼び出し側クローズの変数セル領域を解放する。この方式はテイル・リカージョン・オプティミゼイションと呼ばれ、不要な変数セル領域によるスタックの消費を極力押えることが出来る。

c．拡張制御機能

論理型言語Prologをプログラミング言語として実用的に使用出来るものにしている機能としてカットがある。カットはプログラムの実行時に自クローズの他の選択肢を刈り取ることにより、不要な枝の探索を行なわないようにする機能であるが、KL0にはさらに自クローズの先祖までもカットできる強力な遠隔カットの機能が備えられている。

その他、KL0にはユニフィケーションに失敗してバックトラックした際に指定された処理（on-backtrack）を行なう機能、ユニフィケーションによって特定の変数に値がバ

インドされた際、その時点であらかじめ指定しておいた処理を開始する機能（bind-hook ）が備えられている。

マイクロインタプリタはこれらの機能を実現するための制御情報をダイナミックに生成する。

d．OSサポート機能

OSの重要な機能のうちハードウェアにディペンドしたものとして割込み処理、プロセススイッチ、実メモリの割付け等がある。 割込み処理では割込みの検出、割込み原因の固定エリアへのセーブ及び割込みハンドラへのプロセススイッチが必要であり、プロセススイッチ処理では実行中のプロセスの実行環境のセーブ及び新しいプロセスの実行環境のロードを行なう必要がある。これらの効率はシステム全体の性能に大きく影響する。したがって、ψにおける割込み処理、プロセススイッチはすべてマイクロインタプリタによって処理される。また、メモリ割付けについても実行中の各プログラムに対してマイクロインタプリタが動的に実メモリの割付け及び不要となったメモリの回収を行なう。 マイクロインタプリタによりこれらの処理をサポートすることにより処理速度の向上だけでなく、OSのプログラム記述が容易になる。

4．マイクロインタプリタのモジュール構成

マイクロインタプリタはその機能により次の４モジュールから構成されている。（図１）

a．核部

KL0 の内部形式と 4本のスタックを使用して実行順序の制御及び管理を行なうと共にユーザ定義述語のユニフィケーション処理を行なう。内部形式中に組込述語のコードが検出されると組込述語部へ制御が移される。拡張制御機能実現のための実行順序の制御もこのモジュールで行なわれる。

b．組込述語部

ψには約 160種の既成語として組込まれた述語（組込述語）が用意されており、核部により組込述語を実行する必要があることが検出されるとこのモジュール中の対応した組込述語処理ルーチンへ制御が移される。このモジュールでの各組込述語の処理が失敗または終了すると制御は核部へ戻される。

c．OSインタフェース部

核部又は組込述語部が割込みを検出するとこのモジュールに制御が渡される。このモジュールでは、割込み原因の解析、割込み原因の固定エリアへのセーブ及び割込みハンドラへのプロセススイッチを行なう。又、実メモリの自動割付けもこのモジュールで行なわれ、もし割付ける実メモリが無くなった場合にはガベージコレクタを呼び出す。

d．初期化部

ψを立ち上げるには実メモリやCPU 内の各種のハードウェアテーブルの初期化を行なう必要があり、これらの処理はこのモジュールによって行われる。

初期化の後OSのプログラムの基本部が実メモリにローディングされ、システムの立ち上げが完了する。

5．開発サポート

ψのマイクロインタプリタを早期に開発するためのツールとして、ほぼC言語風に記述出来るマイクロプログラムアセンブラとマイクロプログラムシミュレータが開発され、マイクロプログラムは入出力命令等のハードウェアにディペンドした部分を除いて全てマイクロプログラムシミュレータ上で開発された。

6．おわりに

マイクロインタプリタの開発はほぼ完了し、現在OSの開発が行なわれている。又これと平行してマイクロインタプリタの評価を行なっているが、性能はほぼ当初の目標を達成できる見込みである。 今後さらに、細々の評価を行なうと共にシステム全体としての評価も行なう予定である。

最後に、日頃有益な助言をいただくＩＣＯＴメンバ及び協力いただいたメーカの方々に深く感謝する。

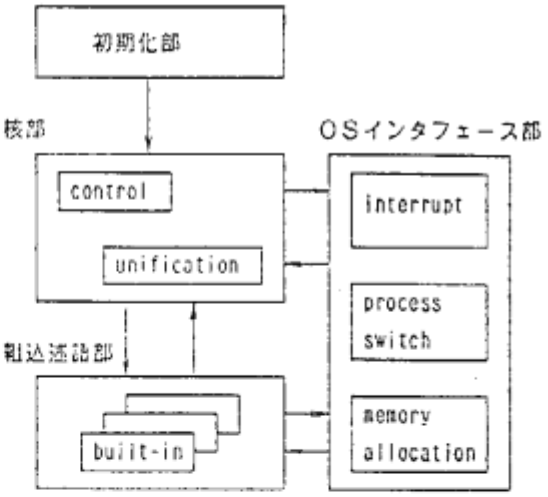


図１ マイクロインタプリタモジュール構成

参考文献

1.Warren.D , H.D 「An improved Prolog Implementation which optimises Tail Recursion」Proc. of the Logic programming workshop , Hunguary (July,1980)

2.Chikayama,T, et al 「Fifth Generation Kernel language Version-0」Proc. of the Logic programming conference,Japan (March,1983)

3.Yokota,M , et al 「The Design and Implementation of a personal Sequential Inference Machine:PSI」New Generation Computing Vol.1 NO.2 '83 ohmsha,ltd.

逐次型推論マシンのファームウェア
―― 基本制御部 ――

中島克人　　山本　明　　横田　実
（三菱電機）　（新世代コンピュータ技術開発機構）

1．はじめに

逐次型推論マシン（PSI）ではKL0と呼ばれるPrologライクな論理型言語を直接解釈・実行するマイクロインタプリタ方式を採用している。本論文では述語呼出しやバックトラックなどの実行順序を制御するマイクロインタプリタ基本制御部の概要と、システム性能の向上のための処理方式について報告する。

2．基本制御部の概要

基本制御部では4本のスタックを使用して、実行環境の管理を行なっている。

a．コントロール・スタック

述語呼出しの際の環境を、コントロール・フレームと呼ぶ10語を単位として格納する。コントロール・フレームは述語の呼出し元へのリターン時や、バックトラック時の環境の復帰に使用される。

b．ローカル・スタック

ヘッド引数および、ボディ引数にのみ現れる変数のためのセル（ローカル・フレーム）を述語呼出し単位に格納する。ローカル・フレームはスクラッチパッド・メモリ上に設けた2面のバッファに交互に生成され、保存が必要なフレームのみ、スタックに積まれる。

処理対象のクローズが決定的に終了する場合には、コントロールおよびローカル・フレームはたたみ込まれる（ティル・リカージョン・オプティミゼーション：TRO）。つまり、すでにそのクローズのフレームが存在すればそれを放棄し、まだ存在しなければ格納しない。

c．グローバル・スタック

構造体内変数のためのセルや、拡張制御機能用の情報など、クローズの実行が決定的に終了しても解放できない情報を格納する。

d．トレール・スタック

バックトラック時に値の束縛を解放（UNDO）すべき変数セルのアドレスを積む。また、拡張制御機能用の情報の内、UNDOすべきものに関しても、そのアドレスとデータの2語を対として格納する。トレール・スタックの先頭部分は、やはりスクラッチパッド・メモリ上に設けたバッファに保持される。

基本制御部の概略フローを図1に示す。図には示していないが、基本制御部では、述語呼出しの切れ目における割込チェックや、拡張制御機能用に述語呼出しのレベルの深さの管理なども行なっている。

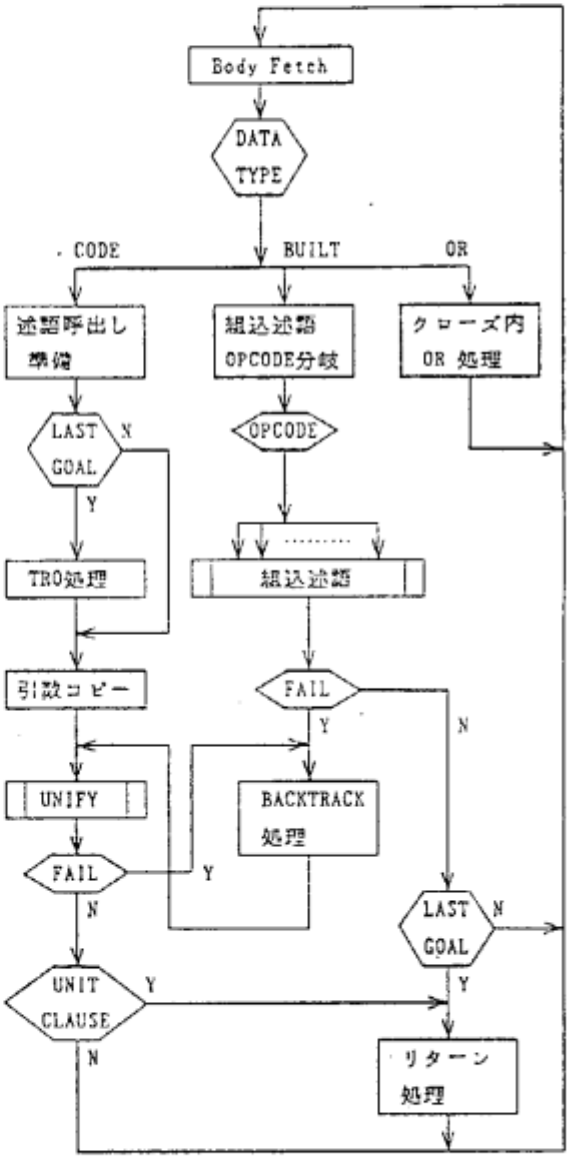


図1　マイクロインタプリタ基本制御部概略フロー

3．引数コピー方式

XLO の内部コードにおいては、クローズ内の変数は、そのクローズの呼出し元のフレーム内オフセットという意味の変数番号で表現されている（図２）。インタプリタでは、そのクローズのボディゴール部からの述語呼出しにあたって、まずボディ引数の評価を行ない、ローカル・フレームを生成する。つまり、ボディ引数が変数の場合は、その変数番号をオフセットとしてそのクローズの呼出し元フレームをアクセスし、新しいフレームにコピーする。ボディ引数が定数の場合は、その値自身をコピーする。

ユニフィケーションは、ボディ引数の情報に基づきコピーされたローカル・フレームと、呼出された側のクローズヘッド引数との比較という形で行われる（図２）。

この方式は、ローカル・フレームへの格納・読出しが若干のオーバーヘッドになるが、

(1) ユニフィケーションが成功する場合には、このローカル・フレームはいずれ呼出し側クローズのボディゴール引数の評価時に、呼出し元フレームとして使用されるため、引数評価の手間は同じである。

(2) ユニフィケーションが失敗し、他のクローズが呼び出される場合には、コピー済のフレームが再利用される。

(3) 呼出しがクローズの最後のゴールであり、かつ、処理が決定的な場合には、必要な引数の値はすでに新しいフレームにコピーされているため、呼出し元のフレームの保存が不要となり、対応するスタック領域の解放が可能

となる。いずれもコピー方式の実現を目的とした専用のハードウェアにより、バッファ・アクセスのオーバーヘッドは小さくなっている。

4．クローズ・インデックシング

XLO では呼出し側の任意の引数によるクローズ・インデックシングを可能としており、第１引数でのインデックシングの後に、更に第２引数でインデックシングするなど、多重のインデックシングも可能である。XLO コンパイラは、インデックシングに用いる引数の位置、ハッシュによる分岐数および分岐テーブルをコードとして生成することにより、インタプリタへの指示を与える。インタプリタでは、クローズ・コードの位置に挿入されたこのコードを解釈し、指示された呼出し側引数を評価し、ハッシュ値を計算し、分岐テーブルによりクローズを選択する。

現在、ハッシュ関数は１種類であるが、コードの指示により、複数のハッシュ関数を選択する方式も容易に実現できる。

多重のクローズ・インデックシングは、大量のデータ・ベース・クローズの選択などには非常に有効な手段となるであろう。

5．おわりに

マイクロインタプリタの基本制御部は約８００語／64bit のマイクロ命令で構成されている。今後、処理速度向上のために、各種の評価および改良を行なっていく予定である。

最後に、本インタプリタの作成にあたり、適切な御指導・御助言をいただいた、ICOTメンバおよび、関連メーカの方々に深謝する。

参考文献

1. Warren. D . H. D "An improved Prolog Implementation which optimises Tail Recursion" Proc. of the Logic programming workshop (July. 1980)

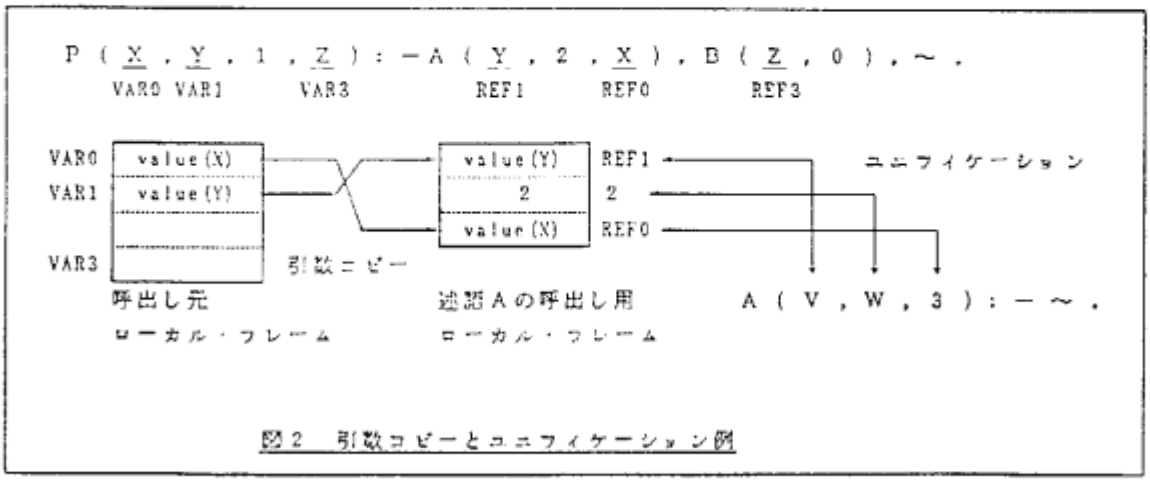


図２　引数コピーとユニフィケーション例

逐次型推論マシンφのファームウェア
―ユニフィケーション部―

/E-3

三井 正明（沖電気）　横田 実（新世代コンピュータ技術開発機構）　西川 豊（新世代コンピュータ技術開発機構）　藤村 茂（シャープ）

1．はじめに

パーソナル逐次型推論マシンφ（ＰＳＩ）はＫＬ０と呼ぶPrologライクな論理型言語を機械語としている。従って、そのファームウェアはＫＬ０を直接解釈実行するインタプリタとして機能する。

本稿では、マイクロインタプリタ核部のうち、ユニフィケーション部について報告する。

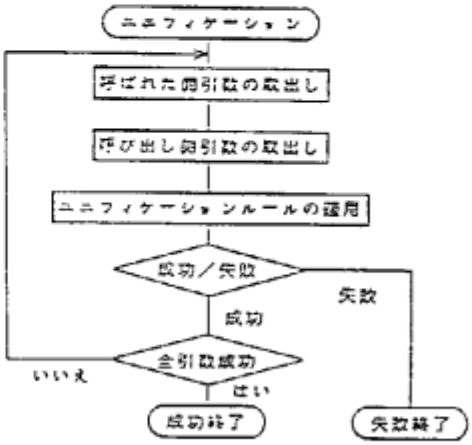
2．概要

ユニフィケーション部はＫＬ０のユーザ定義述語に関して、呼出し側ボディゴール引数と呼ばれた側クローズヘッド引数との間の比較あるいは代入といったユニフィケーション機能を担っている。

呼出し側の引数は引数コピー方式の採用により既に基本制御部に於て評価され、呼ばれた側の変数セル領域の先頭にコピーされているので、変数セル領域のポインタ経由で逐次的に取出せることが約束されている。呼ばれた側の引数はコード中に存在し、命令ポインタ経由で取出せる。引数間のユニフィケーションの順序は本質的には任意であるが、φでは前述の２つのポインタをインクリメントしながら第１引数より逐次的に行なっている。

ユニフィケーションの手順を図－１にフローチャートで示す。

全引数のユニフィケーションが成功すると基本制御部の成功リターン処理に制御を戻す。又、失敗した場合はその時点でユニフィケーションを打切り基本制御部のバックトラック処理に制御を戻す。



図－１　ユニフィケーションの手順

3．ユニフィケーションルール

ユニフィケーションの成功・失敗は取出した引数のデータタイプに応じたユニフィケーションルールの適用によって決定する。ユニフィケーションルールの説明と一覧表（表－１）を以下に示す。

表－１　ユニフィケーションのルール

データタイプ ＼ データタイプ	アンバウンド	アトミック	ヒープベクタ	ストリング	スタックベクタ	コード	プログラミングタイプ
アンバウンド	成　功	成　功	成　功	成　功	成　功	成功	成　功
アトミック	成　功	タグ：比較 データ：	失　敗	失　敗	失　敗	失敗	失　敗
ヒープベクタ	成　功	失　敗	要素数：比較 各要素：再帰的にユニフィケーション	失　敗	失　敗	失敗	失　敗
ストリング	成　功	失　敗	失　敗	文字コード：比較 ポインタ：比較 長さ：比較 アドレス：比較	失　敗	失敗	失　敗
スタックベクタ	成　功	失　敗	失　敗	失　敗	要素数：比較 各要素：再帰的にユニフィケーション	失敗	失　敗
コード	成　功	失　敗	失　敗	失　敗	失　敗	データ：比較	失　敗
プログラミングタイプ	成　功	失　敗	失　敗	失　敗	失　敗	失敗	データ：比較

＊　比較の結果　一致すれば成功

＊　スタック中のベクタでは各要素について、ユニフィケーションのルールを適用する。

(1)アトミックデータ

タグ部のデータタイプとデータ部の値の比較を行ない一致すればユニフィケーションは成功、一致しなければ失敗とする。

(2)構造体データ

構造体データには本体がヒープ中に格納されている場合とグローバルスタック中にその本体セルが作り出されている場合の２種類がある。サイドイフェクトを伴うヒープ中の構造体は物理的に同一であるとき成功とする。undo処理の対象となるスタック中の構造体は要素同士の比較を行なって成功・失敗を決定する。各構造体についてのユニフィケーションルールを示す。

(a)ヒープベクタ

両方のヒープベクタの要素数と本体アドレスの一致を調べる。

(b)ストリング

両方のストリングの要素数とタイプ、オフセット及び本体アドレスの一致を調べる。

(c)コード

両方のコードの本体アドレスの一致を調べる。

(d)プロテクティドタイプ

両方のタイプ記述子のアドレスの一致を調べる。

(e)スタックベクタ

両方のスタックベクタの要素数の一致を調べ、個々の要素同士の比較又は代入を行なう。

(3)アンバウンド変数セル

(a)共にアンバウンド

アドレスの大きい方から小さい方にポインタを張って両方のデータが論理的に同一であることを示す。ポインタが代入される変数セルにフックがかけられていたらその内容を他の変数セルに移す。両方共フックがかけられているときは更にその内容にマージを施し、アドレスの小さい方にその結果を格納する。

(b)片方のみアンバウンド

アンバウンドであるセル中に他方のセル内容をコピーする。フックがかけられていたらその内容はφ内の特定レジスタとの間でマージを行なう。

4．構成と実現法

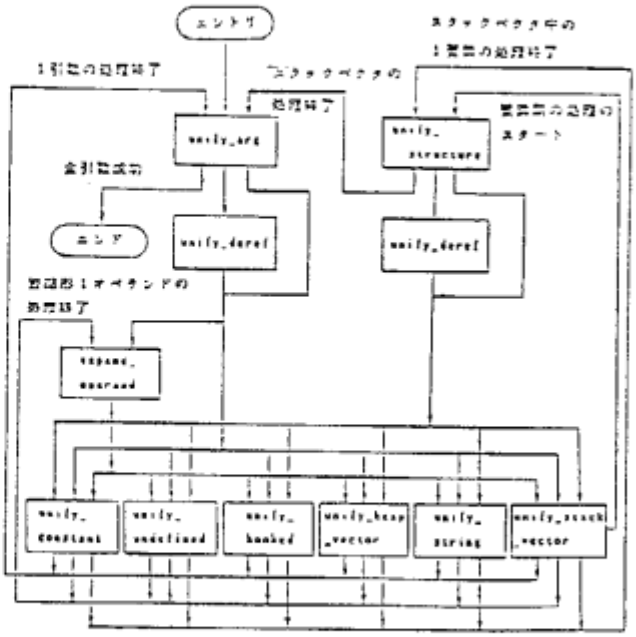
ユニフィケーション部のルーチンは図－1の手順を図－2に示すモジュール構成で実現したマイクロプログラムである。ルーチンの大半はユニフィケーションルールの適用部で表－1の組合を分類するグループと、組合を分類し規定された処理を行なうグループの2つに大きく分けることができる。前者のグループはヘッド側引数オペランドに対して

・トップレベルのオペランドの分類
・短縮形オペランドの分類
・構造体要素オペランドの分類

の3種のモジュールとそれに付随するデレファレンスのモジュールから成っている。後者のグループは分類されたヘッド側引数オペランドのデータタイプに対応して

・アトミック、コード、プロテクティドタイプ
・未定義変数
・バインドフック変数
・ヒープベクタ
・ストリング
・スタックベクタ

といったデータタイプ別モジュールに分けられる。



図－2　モジュール構成

後者のグループのモジュールは前者のグループで共用され、ボディゴール側引数オペランドを分類して表－1に対応する処理を行う。共用の方法は、ハードウェア資源のjr(jump register)に成功・失敗時の戻り先情報を格納しておき処理終了後jrの間接ジャンプオーダを発行するという形で実現している。

データタイプの分類はタグディスパッチ機能を有効に利用して高速化を図っている。前者のグループにヘッド側引数オペランドのデータタイプをタグディスパッチ機能で分類し後者のグループへ分岐する。後者のグループはボディゴール側引数オペランドとのタグの比較を陽に行なわず、タグディスパッチを利用した自動分類によってヘッド側引数オペランドとつきあわせ、成功・失敗の処理を行なっている。

タグディスパッチは2種類の分岐条件指定を使用している。それぞれ5面、8面の受け口があり、あわせて38回のタグディスパッチオーダを発行しユニフィケーション部の高速化に役立たせている。

5．おわりに

マイクロインタプリタの開発はユニフィケーション部も含めてほぼ完了し評価の段階に入っている。今後はシステム全体の評価による見直しを進めていく予定である。

参考文献

Yokota.M．et al　「The Design and Implementation of a Personal Sequential Inference Machine:PSI」 New Generation Computing Vol.1 No.2 '83 ohmsha.ltd.

逐次型推論マシンφのファームウェア
-データ・タイプとその操作-

小松 英二　三井 正樹　(沖電気)
横田 実　山本 明　吉川 宏　宮崎 敏彦
(新世代コンピュータ技術開発機構)

1. はじめに

パーソナル逐次型推論マシンφは、第五世代コンピュータ・プロジェクトにおけるソフトウェア開発用パイロット・モデルである。φにはＫＬ０と呼ばれるＰｒｏｌｏｇライクな言語をベースとした機械語があり、約１６０種類の組込述語をサポートしている。データ操作系組込述語はこのうちの約５０種類を占めている。φにおけるデータ操作は、ヒープ上のデータ操作に特徴を有している。本論では、φファームウェアにおけるデータ操作部として、ヒープ上のデータ操作に触れながら、データ操作系組込述語について報告する。以下、特に断らない限りスタックとはローカル・スタックまたはグローバル・スタックを指すとする。

2. データ・タイプ

φには将来の拡張用も含めて約６０種類のデータ・タイプがある。以下に特徴のあるデータ・タイプを幾つかあげる。

a. ヒープ・ベクター、スタック・ベクタ

一次元の配列であり、その本体の置いてある場所がヒープかスタックかにより、ヒープ・ベクタまたはスタック・ベクタと呼ぶ。

b. コード

ＫＬ０のプログラムのクローズを表すために用いられる。構造はヒープ・ベクタと同じであるが、誤動作を防ぐ為、別のデータ・タイプにしてある。

c. モレキュール

コード中のヒープ・ベクタは、変数の値がスタック中にある。これと、オブジェクト生成述語により、動的に作られたヒープ・ベクタを区別するために、読み出し時にモレキュールを作り、スタック・ベクタと同じ扱いにする。

d. ロケーション

このデータ・タイプは「visibleなポインタ」と考えられ、構造は一要素のベクタと同じである。ロケーションの指す要素はdangling referenceを防ぐ為ヒープ上のデータ・タイプに限っている。　図1.、2.にモレキュール及びロケーションの構造を示す。

e. プロテクテッド・タイプ

プロテクション・キーと呼ばれる保護キーと保護されるデータを持った構造体である。図３に構造を示す。

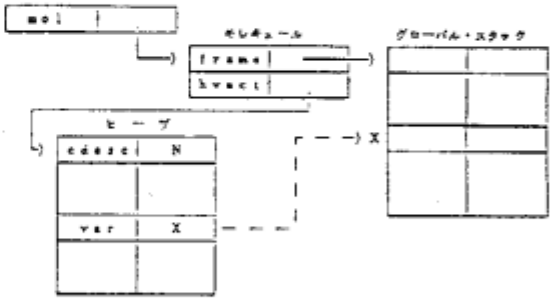


図1　モレキュールの構造

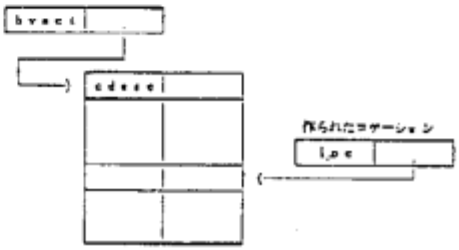


図2　ロケーションの構造

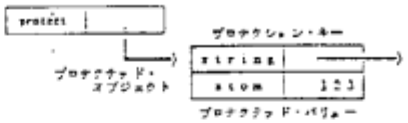


図3.　プロテクテッド・タイプの構造

3. non stack objectとstack object

データは、ヒープ又はスタックに置くが、スタックは動的に伸縮するためにヒープからスタックへのポインタは、ロケーションのときのようにdangling referenceになる可能性がある。このためφでは、ヒープ上に置くことのできるデータ・タイプを制限することにより、ヒープからスタックへのポインタを防止している。

a. non stack object

ヒープ上に置くことのできるデータ・タイプをnon stack objectとよぶ。このグループには、未定義語のように将来ポインタを張る可能性のあるデータ・タイプも除いている。上記のヒープ・ベクタ、コード、ロケーション及びロケーションの指すデータはnon stack objectである。

b．stack object

スタック上に現れるデータ・タイプをstack objectと呼ぶ。スタック・ベクク、モンキュール、はstack objectである。

c．その他

インタプリタ制御用データ・タイプの中には、トレイル・スタックにしか現われないものがある。このようなデータ・タイプは、non stack objectでもstack objectでもなく、このデータ・タイプがヒープ又はトレイル以外のスタックに現れたときはエラーとなる。

φのデータ・タイプは上記のa)．～c)．のどれかに属する。

4．データ操作

ヒープ上のデータとスタック上のデータではデータの書換えに関して、データ操作における扱いが全く異なる。ヒープ上のデータは常にサイドエフェクトにより上書きが行われ、バックトラックによってundoされない。一方、スタック上のデータは、ユニフィケーションによってのみ書き変えられ、バックトラック時にはundoする。ヒープへの上書きはnon stack objectに限る。

5．データ操作系組込述語

データ操作系組込述語は、次のような３つのグループからなる。

a．属性チェック組込述語

データのタイプ・チェック及びその他の属性の取りだしに用いられる。タイプ以外の属性としては、データのバリュー部、構造体データの長さ、ストリング要素のタイプ（ビット、バイト、２バイト、ワード）、プロテクテッド・タイプのプロテクション・キー及び値などがある。

述語の例：atom(X)、value(X、Value)

code(X、Length、Narg)、string(X、Length、Type)

b．オブジェクト生成組込述語

アトム、ヒープ・ベクク、スタック・ベクク、ストリング、プロテクテッド・タイプ、コードを生成する述語を用意している。構造体データの本体は作るデータ・タイプによって、ヒープまたはスタックのトップに生成し、データは第一引数にセットする。

述語の例：new_heap_vector(X、Length)

new_protected_object(X、Key、Value)

c．構造体データ操作組込述語

構造体データ操作述語は、構造体アクセス、サブ構造体アクセス、ロケーション・アクセスに分けられる。さらに、データのバインディングの方法が、ユニフィケーションによるものとサイドエフェクトによるものを用意している。

○構造体アクセス

構造体の要素を取り出したり、上書きを行なったりする述語である。サイドエフェクトにより書き込みが行われる構造体はヒープ上に本体を持つ構造体に限り、また書き込む要素はnon stack objectである。

述語の例：vector_element(Vector、Position、Element)

set_vector_element(Vector、Position、Element)

○サブ構造体アクセス

構造体のサブ構造体を取り出したり、他の構造体を上書きしたりする述語である。サイドエフェクトによりそのデータを書き込むベクタは要素がnon stack objectでなければいけない。サブ構造体は記述子だけを新たに生成し、要素は元の構造体と共用する。図３．にサブ構造体の構造を示す。

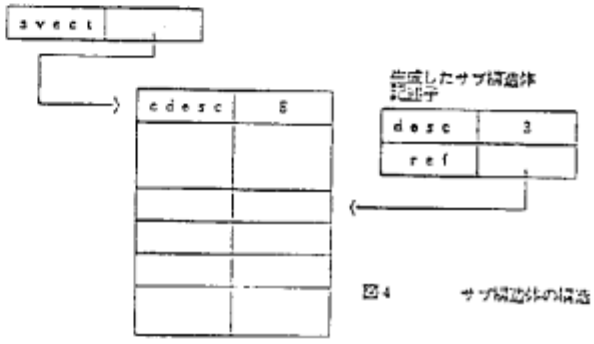


図4　サブ構造体の構造

述語の例：sub_vector(Vector、Position、Length、Subvector)

set_sub_vector(Vector、Position、Length、Subvector)

○ロケーション・アクセス

ロケーションの生成や、ロケーションの指しているデータの取り出し、及びデータ・セットを行なう述語を用意している。ロケーションにセットするデータはnon stack objectに限る。

述語の例：vector_element_location(Vector、Position、Element)

location_element(Location、Element)

set_location_element(Location、Element)

6．おわりに

φでは、データ操作に関して、内部的にはヒープ等に関連したデータ・タイプのチェックを徹底的に行い信頼性を増す一方、ＫＬ０言語レベルでは、ロケーションなどの導入により多様な機能を提供している。また、本論でのべた以外に、データ操作系全体での整合性や特権述語の設定により、ユーザ・プログラムであるコードの破壊防止なども考慮している。

逐次型推論マシンφのファームウェア

――割込み処理とプロセス管理――

1B-5

池田守宏　　横田 実　山本 明　渡 和男　西川 宏
（三菱電機）　（　新世代コンピュータ技術開発機構　）

1．はじめに

パーソナル型逐次推論マシン φ （ＰＳＩ）では、プログラムの実行の単位をプロセスと呼び、ＯＳのプロセス管理ルーチンの管理下で複数プロセスの実行環境を作っている。このようなマルチプロセスの管理には、複雑で細かなハードウェア資源の操作が必要であるが、それによる処理速度の低下及び誤操作を避けるため多くの部分をファームウェア化し、ＯＳとのインタフェースを簡潔にすることに努めている。このプロセス管理のファームウェアは、割込み，プロセス切替処理を中心としたＯＳインタフェース部と、ソフトウェアインタフェースのための組込み述語部の２つの部分から構成されている。

本論文ではこれらファームウェアに関し、論理型機械言語ＫＬ０実行マシンとして特徴的な処理を中心にその概要を報告する。

2．プロセス

a．プロセスの実行環境

φ では最大６３個のプロセスを設けることができる。各プロセスはそれぞれ独立に機械語ＫＬ０のプログラムを実行する。このためにＰＣＢ，ＥＰＣＢと呼ばれる実行制御用の情報をそれぞれのプロセスが個別に持っている。

ＰＣＢ (process control block)はソフトウェアが参照，操作するステイタス情報でプロセスの割込みレベル等より成り４語で構成される。ＥＰＣＢ(extended process control block)はファームウェア，マイクロインタプリタが参照，操作するための制御情報である。これはマイクロアドレススタックの内容，汎用マイクロレジスタ，命令レジスタ，スタック操作用ポインタ類などより成り最大６０語で構成される。以上の制御情報の他各プロセスは、それぞれＫＬ０実行時に必要な４本のスタックエリア（グローバル，ローカル，コントロール，トレイル）用に主記憶が割当てられており、エリアごとのＡＴＰ(area top pointer)がこれらスタックのトップを管理している。

b．プロセス切替

プロセス切替は上記プロセス実行環境を全てファームウェアにより切替る処理である。

プロセス実行時ＰＣＢ，ＥＰＣＢ，ＡＴＰは、ハードウェアレジスタ上にロードされ、ＫＬ０実行に従ってファームウェアにより参照，更新されている。ここでプロセス切替の要求が起るとプロセス切替の処理ルーチンがサブルーチンコールされる。これによりＰＣＢ，ＥＰＣＢ，ＡＴＰはプロセスごとの退避領域に格納される。その後新たに起動されるプロセスのそれら情報が、退避領域からロードされ新しい実行環境が作られる。このプロセス切替の状態を図１に示す。

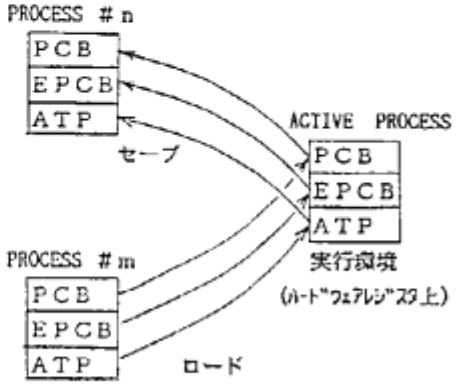


図　１　　プロセス切替の状態

3．割込み処理

プロセス切替の要因として割込み及びトラップがある。割込みはプログラムと非同期に発生しＰＣＢ中のマスクレベルの設定により抑止出来るもの、トラップはプログラムとは同期して発生し抑止出来ないものである。それぞれの要因を表１に示す。ファームウェア上割込み，トラップは特に区別無く扱い、以後単に割込みと称する。

表　１　割込み，トラップ要因

名称	要因
ｶｰﾈﾙ　ｴﾗｰ　１　ﾄﾗｯﾌﾟ	ﾊｰﾄﾞｳｪｱ 検出ﾒﾓﾘｱｸｾｽ ｴﾗｰ
ｶｰﾈﾙ　ｴﾗｰ　２　ﾄﾗｯﾌﾟ	ﾌｧｰﾑｳｪｱ 検出ｴﾗｰ
ＧＣ　ｺｰﾙ ﾄﾗｯﾌﾟ	ｶﾞｰﾍﾞｰｼﾞｺﾚｸｼｮﾝ 要求
ｴﾘｱ ﾘﾐｯﾄﾁｪｯｸ　ﾄﾗｯﾌﾟ	指定ｻｲｽﾞを越えたﾒﾓﾘｴﾘｱｱｸｾｽ
ﾄﾚｰｽ ﾄﾗｯﾌﾟ	KLO ｽﾃｯﾌﾟ 実行用
ｵﾍﾟﾚｰﾀ割込み	ｺﾝｿｰﾙ 割込み
温度，電源異常割込み	CPU 内部温度，電源異常
入出力高位割込み	入出力装置（高位ﾚﾍﾞﾙ）
ﾀｲﾏｰ割込み	ｲﾝﾀｰﾊﾞﾙﾀｲﾏｰ
入出力低位割込み	入出力装置（低位ﾚﾍﾞﾙ）
ﾒﾓﾘｴﾗｰ訂正割込み	主記憶１ﾋﾞｯﾄｴﾗｰ 訂正

a．割込みの検出

割込みの検出は原則としてＫＬ０実行処理の切目、即ち述語呼び出しの切目でハードウエアの割込みフラグを判定することにより行う．この割込み検出のタイミングを基本タイミングと呼び、ユーザ定義述語の場合は、ユニフィケーションの成功又は失敗の直後，組込み述語の場合は実行終了時点である．ここでのプロセス切替では、　ＥＰＣＢの退避情報が最も少く従って高速におこなえる．ＫＬ０プログラムと割込み基本タイミングの関係を図２に示す．

しかし許容待時間の短い割込み又は緊急度の高い割込みに対しては、上記方式だけでは対処できずそれぞれ次のような処理を行う．許容待時間の短い割込み（タイマ，電源異常割込み等）の場合，基本タイミングの他に実行時間の長い組込み述語中、及びユニフィケーション中に割込み検出点を設け、これにより一定間隔以内の割込みチェックを保証している．また要因発生時点で処理しないと以降の実行が継続できない緊急な割込み（物理メモリ不足によるＧＣｺｰﾙ等）については，要因発生時点で即刻処理ルーチンを呼び出す．これらの場合はＥＰＣＢの退避情報は基本タイミングに比べ多くなる．

b．割込み処理

ファームウェア割込み処理ルーチンでは、割込みレジスタ上にセットされている割込み原因を読取り、必要な情報を付加して割込み原因（割込みインデックス番号）に対応する`割込み原因コードテーブル`に設定する．さらに同じく割込み原因に対応する`割込みインデックステーブル`を読みだし、処理プログラムのプロセス番号を知りそのプロセスにプロセス切替を行う．割込みインデックステーブルはあらかじめソフトウェアにより設定されている．割込み応答プロセスのプログラムでは通常、割込み原因コードテーブルから必要な、原因情報を得る．

4．プロセス管理用組込み述語

プロセス管理用組込み述語は、ソフトウェアとのインタフェースのために設けられた一群の組込み述語である．

a．プロセス生成用組込み述語

新たにプロセスの実行環境を作るために　initialize_process　及び　set_process_control_block　組込み述語がある．intialize_process はＥＰＣＢの初期設定を行う組込み述語で３引数で起動され、第１引数でプロセス番号を指定し、第２引数でプログラムのスタートコード（ヒープ上当該プレディケートへのポインタ）、第３引数で使用する４本のスタックエリアのうちの先頭のエリア番号を指定する．あらかじめメモリ管理系の組込み述語を使用してスタックエリアを生成した後、この組込み述語を実行することによりＥＰＣＢの初期値が作られ格納される．

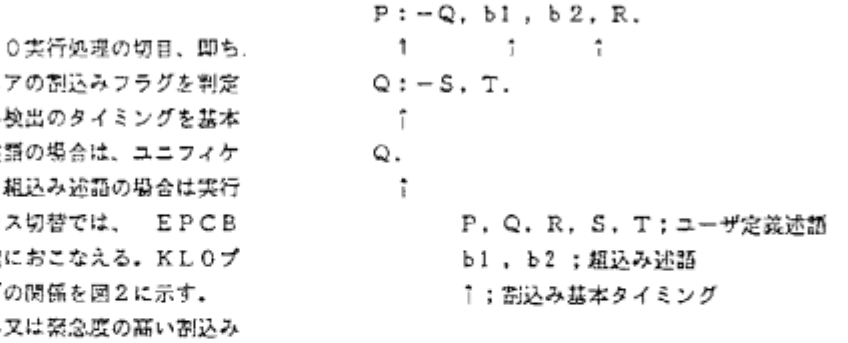


図　２　ＫＬ０プログラムと割込み基本タイミング

set_process_control_block 組込み述語は４語構成のベクタ形式で表すＰＣＢデータとプロセス番号の２引数で起動され、ＰＣＢデータを対応ＰＣＢに格納する．これによりＰＣＢの初期設定又は値の更新がおこなえる．

b．プロセス切替，トラップ組込み述語

生成したプロセスへの実行の切替又は割込みプロセスからのリターン等、任意のプロセスへの切替には　change_process 組込み述語が使われる．change_processでは引数として与られたプロセス番号へ、無条件でプロセス切替を行う．

trap組込み述語は４語構成のベクター形式で表す．割込み原因コードデータと、割込みインデックス番号の２引数で起動され、これにより任意のインデックス番号の割込みをソフトウェアで発生できる．

c．プロセス管理用テーブル，レジスタ類アクセス組込み述語

プロセス管理のためソフトウェアとのインタフェースとなるテーブル，レジスタ類をアクセスするための組込み述語で、割込みインデックステーブル，割込み原因コードテーブル，システムレジスタをそれぞれ読み，書くための組込み述語が用意されている．

5．おわりに

割込みトラップ処理，プロセス管理の他 μ では、ＯＳ記述用にメモリ管理等多くのファームウェアモジュールが用意されている．現在これらを使用してＯＳの開発が進められており、プログラムの記述が簡単になる等の効果がみられている．今後ＯＳ評価フェイズを待って性能面等の評価をおこなって行く予定である．

最後に本ファームウェアの開発に当り適切な御指導を賜いた、ＩＣＯＴメンバーの方々及びコーディング，テストに御協力戴いた　三菱総合研究所、小沢殿，岡沢殿に深く感謝する．

逐次型推論マシンφのファームウェア

―マイクロプログラムシミュレータ―

1B-6

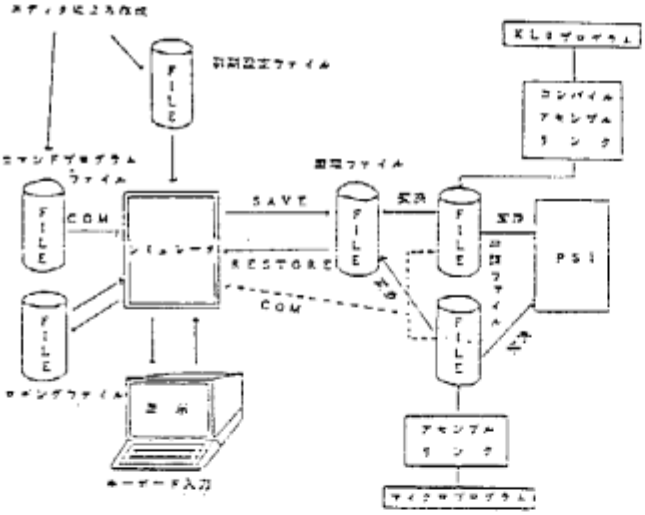
守山　貴（沖電気）　　三井　正明（沖電気）

横田　実（新世代コンピュータ技術開発機構）

1．はじめに

パーソナル逐次型推論マシンφ（ＰＳＩ）は第５世代コンピュータシステム研究開発の一環として開発されているソフトウェア開発用のパーソナルマシンである。

φマイクロプログラムシミュレータはファームウェアの開発をサポートするソフトウェアの一つとして、マイクロ命令の実行動作を論理的にソフトウェアでシミュレートすることにより、φハードウェアの開発と並行してファームウェアの開発及び検証を行うことを目的として作成したものである。本論文ではマイクロプログラムシミュレータの機能及び構成等について報告する。



図－１実行環境

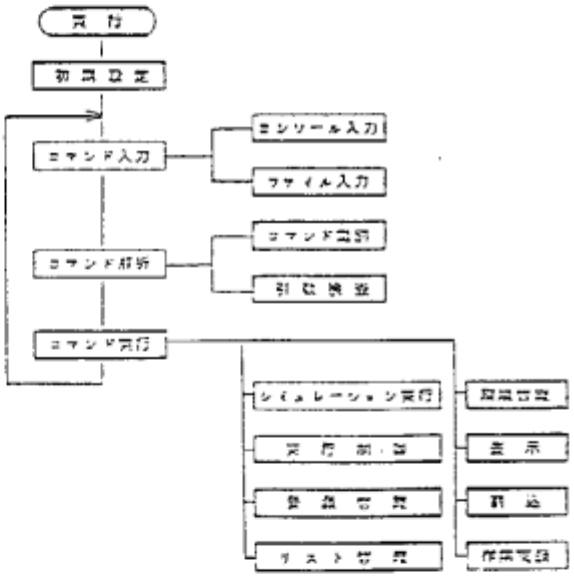
2．特　徴

本マイクロプログラムシミュレータは、移植性、互換性を考慮に入れ、ＰＡＳＣＡＬを用いて記述したものであり、約３４０個のモジュールに分かれ、約６０００ステップより成る。

また、機能的には、割込み、アドレス変換等もサポートしており、ほとんど実機と同等であり、ＤＥＣ２０６０上で、１秒間（ＣＰＵ　ＴＩＭＥ）に約４００マイクロステップ実行可能である。

さらに、本シミュレータはデバッグを容易に行えるように、マイクロアドレスのトレース機能、トレース中トレース数が２５６に達したら実行を停止するリミット機能、マイクロ命令のステップ実行機能、全てのハードウェア資源に対して設定可能なイベント条件を用いることにより、その条件が成り立てばトレースを開始するディレードトレース機能または実行を停止するブレイク機能、そしてロギング機能を有する。そのほか、シミュレータは画面のスクロール範囲を自動的にコントロールし、自動表示、トレーススタック表示を行う機能、マクロコマンドの実行機能も有している。従って、これらの機能を利用することにより効率のよいデバッグが行える。

（図－１）にシミュレータの実行環境を示す。



図－２基本構成

3．プログラム構成

本シミュレータは、（図－２）に示す様に、大別して(a)初期設定部、(b)コマンド入力部、(c)コマンド解析部、(d)コマンド実行部から成り、シミュレーションを実行する。

(a)初期設定部

シミュレータは起動されると、ハードウェア資源に相当する領域とシミュレータ内部資源をクリアし、ユーザーにより作成されたファイルを読み込んでコマンド名、タグのニーモニックの設定等を行いシミュレーションに備える。

(b)コマンド入力部

シミュレータへの入力は、通常のキーボードからの入力の他ファイルからの入力があり、マクロコマ

ンドを実現している。

(c)コマンド解析部

コマンドが与えられるとシミュレータは初期設定部で設定したどのコマンドと一致するかを判別し、一致すればそのコマンドに適した引数が揃っているかを検査する。

(d)コマンド実行部

コマンド実行部は以下の8つの機能よりなる。

(i)シミュレーション実行機能

マイクロプログラムの実行をシミュレートするシミュレータ本来の機能で、連続実行、継続実行等が可能である。

(ii)実行制御機能

シミュレーション実行を制御する機能で、モードを設定することにより行われる。

(iii)資源管理機能

φのハードウェア資源及びシミュレータの内部資源に対して内容表示、内容変更をする機能である。

(iv)リスト管理機能

実行制御モードの条件リスト、自動表示の対象を示すハードウェア資源リストを管理する機能である。

(v)表示機能

実行制御モードの状態を常時表示すると共に、自動表示として指定されたハードウェア資源の内容を実行停止毎に画面上部から表示する機能である。

(vi)マクロコマンド実行機能

コマンドプログラムファイルに定義されているマクロコマンドを実行する機能である。

(vii)環境管理機能

シミュレーションの一時停止、再現等に必要な実行環境の退避、再設定を行う機能でバイナリファイルを用いて行う。

(viii)操作補助機能

10進数と16進数の変換をサポートする機能である。

表－1に以上の機能を実現しているコマンドの一覧表を示す。

4．シミュレーション方式

本シミュレータは、ハードウェア資源（レジスタ、レジスタファイル等）をすべてソフト的な変数や配列として持ち、ソース1、ソース2等のバスも変数として持っている。それらの変数をｍａｒ（マイクロアドレスレジスタ）の指すマイクロ命令に従って操作しシミュレーションを実現している。ｍａｒの指すマイクロ命令は64ビットで構成されており、シミュレータはマイクロ命令の上位32ビット、下位32ビットを2つの整数として受け取り、各フィールド毎のビットパターンを値として切り出し、そのフィールドに応じた処理を行う。

また、特殊な処理として、割込み処理がある。シミュレータは、割込み名をコマンドとして受け取りその引数のＯＮ／ＯＦＦの指定により、割込み・トラップ用の論理的に連動する3つのレジスタを自動的に操作する。マイクロ命令のシミュレーション実行中に割当てられていない実メモリへのアクセス等の割込みが起こると、シミュレータはコマンド指定による割込みと同じことを自動的に行う。そして、外部デバイス等からのシミュレータ内部では起こりえない割込みをシミュレートする場合に前述のブレイク機能やシミュレータ自身の割込み（＾Ｙ）を使ってマイクロ命令の実行を一時停止させ、割込み名を指定して割込みが起こった状態とした後、マイクロ命令を継続実行させる等の方法がある。

種　別	コマンド名	機　　能
シミュレーション実行	GOTO	指定のアドレスからのマイクロプログラムの実行
	STEP	一ステップ毎のマイクロ命令の実行
実行制御	STEP	ステップ　モードの設定、解除
	BREAK	ブレイク　モードの設定、解除
	TRACE	トレース　モードの設定、解除
	DTRACE	ディレードトレース　モードの設定、解除
	LIMIT	リミット　モードの設定、解除
	LOG	ログ　モードの設定、解除
資源管理	DISPLAY	資源の内容表示
	CHANGE	資源の内容変更
リスト管理	SET	イベント表示リスト、及び自動表示対象ハードウェア資源リストの登録
	RESET	イベント表示リスト、及び自動表示対象ハードウェア資源リストの抹消
マクロコマンド実行	COM, COMND COMLOG	コマンドプログラムファイルからコマンド入力、及び実行
	PAUSE	マクロコマンド実行の一時停止
	CONTINUE	マクロコマンド実行への復帰
環境管理	SAVE	シミュレーション実行環境の保存
	RESTORE	シミュレーション実行環境の設定、再設定
画　面	REWRITE	画面の更新
	PREVIOUS	一つ前の自動表示の内容を表示
操作補助	DTOH	10進 → 16進
	HTOD	16進 → 10進

表－1コマンド一覧

5．おわりに

φのファームウェアはすべて実機完成前にシミュレータを用いてデバッグが行われたため実機デバッグを短期間で完了できた。更に基本的な性能評価も本シミュレータを用いて行われている。

今後もこうしたシミュレータはファームウェア開発には不可欠のものであろうが、今回その有効性を確認した。

最後に日頃有益な助言をいただくＩＣＯＴの方々及びメーカの方々に深く感謝する。

逐次型推論マシンPSIのハードウェア
—— 入出力システムと割込制御方式 ——

1E-7

三石彰統　京　敬人　真取実明　　西川　宏
（三菱電機）　（新世代コンピュータ技術開発機構）

1. はじめに

PSIマシンはコンピュータシステム研究用第5世代プロジェクトにおいてソフトウェア開発用ツールとして開発された逐次型推論マシンPSI（PSI-I）は、マウス，ビットマップディスプレイ，LANインタフェース，ディスクなど豊富な入出力装置を備えている。これらの入出力装置はIEEE-796に準拠した入出力バスに接続されており、中の核言語であるKL0（Kernel Language 第0版）の入出力組込述語を用いて制御される。またPSIでは優先順位の付いたベクター割込方式の割込制御を採用しており、割込によってプロセスの切替えを行なっている。本報告では入出力システムと割込の制御方式について述べる。

2. 入出力システム

PSIの入出力装置は図1.に示すように全て入出力バスに接続され、入出力制御部を介してCPUから制御される。

入出力バスは16メガバイトのアドレス空間を有しており、この空間は主記憶のメモリ空間とは完全に独立している。入出力バス空間には512キロバイトの入出力バスメモリと、1.5メガバイトのビットマップメモリ（ビットマップディスプレイの制御部に内蔵）が実装され、64キロバイト分の空間が制御装置への起動用あるいは割込のリセット用の制御装置識別アドレスとして使用されている。残りの空間は現在未使用。

CPUが入出力バスに接続されている制御装置との間でデータ転送を行なう場合、CPUは入出力組込み述語、例えばputbyte, put_vectorなどを用いて全ての入出力装置に共通なHMCB（Hardware Module Control Block）及びTMSG（Transmission Message）と呼ぶ2種の情報を入出力バスメモリ上に生成し、入出力バス空間上の制御装置アドレスを用いて起動をかける。入出力制御装置はHMCB及びTMSGに従って動作し、動作完了後CPUに割込を送る。CPUは入出力完了の割込を受信後HMCB, TMSGの状態情報を確認することで一連の動作を完了する。なおデータ転送は入出力装置と入出力バス

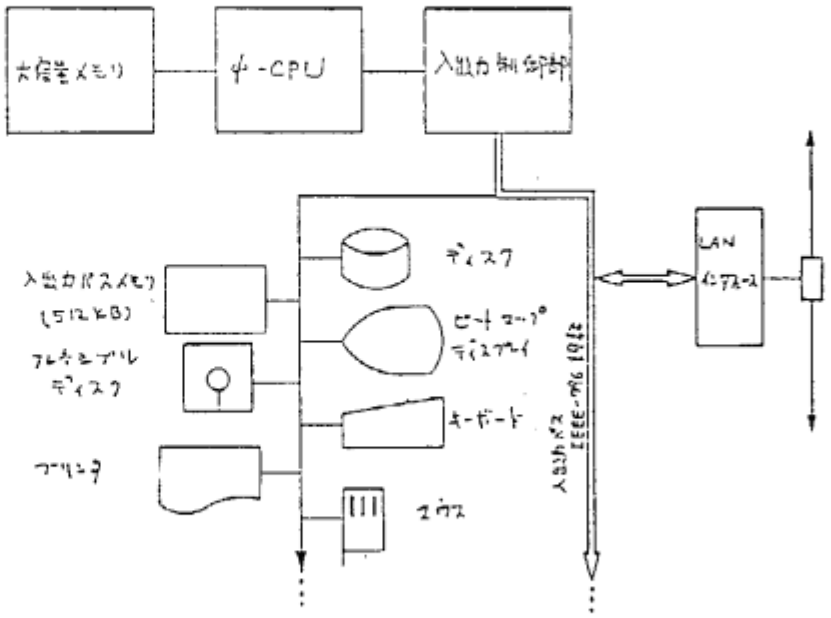


図1. PSIの入出力システム構成

メモリの間で行なわれ、入出力バスメモリとCPUとの間の転送は別途入出力処理装置を用いて行なう。

中の入出力転送処理は以下の17種が用意されている。

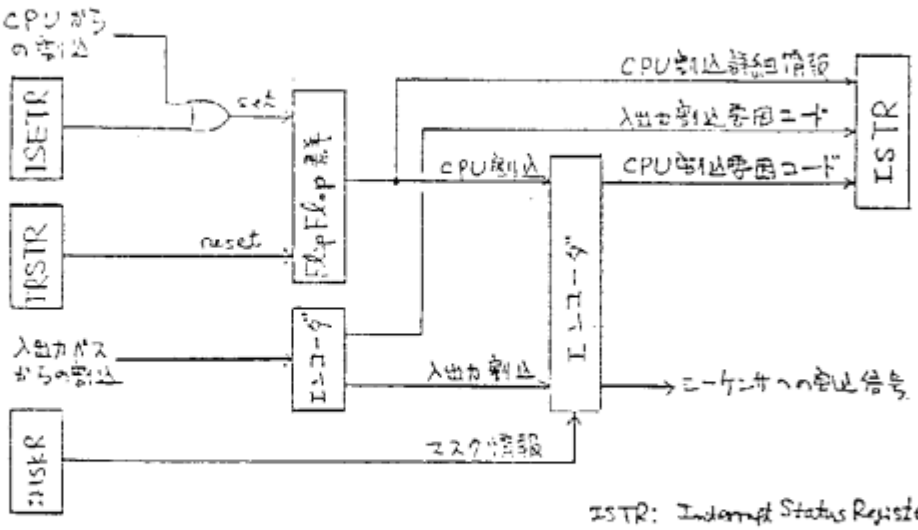
(入力)	(出力)
get_byte	put_byte
get_double_byte	put_double_byte
get_double_byte_swapped	put_double_byte_swapped
get_vector	put_vector
get_vector_swapped	put_vector_swapped
get_string	put_string
get_string_swapped	put_string_swapped
get_with_tag	put_with_tag
(バスの初期化)	bus_reset

3. 割込制御

中は割込をプロセススイッチで実現しており、イベントによりプロセスを切替える機構として割込とトラップを持っている。割込は入出力動作のような外部的要因によってプログラムの実行とは非同期に発生し、トラップは不当なページへのアクセスのような内部的要因によりプログラムの実行に同期して発生するものである。中の割込方式は優先順位の付いたベクター割込方式であり、優先順位を3レベル持っている。トラップに優先順位はなく、マスクもできない。割込制御回路は図2.に示すようにフリップフロップ群、2組のエンコーダ、4本のレジスタから成り、本回路でトラップも扱っている。割込またはトラップの要求が発生すると、フリップフロップ群内のフリップフロップがセットされる。割込受付の優先順位がIMSKR (Interrupt Mask Register) にセットされたマスク順位より高い場合、受付けられる。トラップ要求の場合はIMSKRの内容に関係なく受付けられる。割込/トラップが受付けられると、CPUはエンコードされた割込要因コードを用いて割込インデックステーブルをひき、対応する割込ハンドラのプロセスを起動するとともに、割込原因コードテーブルをセットする。割込インデックステーブルは主記憶中に存在する64ワードの表であり、各ワードには割込ハンドラのプロセス番号が入っている。割込原因コードテーブルも主記憶内の表であって、1エントリ4ワードで構成されている。各エントリには割込原因の詳細コードと付随するパラメータおよび被割込プロセスの番号が格納される。

4. おわりに

本報告では中の入出力システムと割込制御についてその制御方式を述べた。現在は入出力システムの機能強化を検討している。



ISTR: Interrupt Status Register
ISETR: Interrupt Set Register
IRSTR: Interrupt Reset Register
IMSKR: Interrupt Mask Register

図2. 割込制御回路の構成

逐次型推論マシンφのハードウエア

—— システム制御とRASについて ——

12-2

大上貴英　吉市昌一　　森　和男
（三菱電機）　（新世代コンピュータ技術開発機構）

1．はじめに

逐次型推論マシンφ(PSI)は第5世代コンピュータ・プロジェクトで開発が進められているソフトウェア開発用のパーソナル・マシン〔1〕であり，PROLOGに似た論理型言語KL0を直接実行する。ここでは，このφについてそのシステム制御とRASについて報告する。

2．システム制御

2．1．システム制御概要

φのシステム制御においては，ファームウェアで実現されるマイクロインタプリタやKL0で記述されるプログラムの開発を支援する機能の実現を目標に設計されており，従来のミニコンの上位クラスのものに相当する機能になっている。これは，φがパーソナル・マシンとはいえ，将来これを使って各種の大規模なソフトウェアの開発が行われることになっており，これを支援できるように考慮されているためである。

φのシステム制御はコンソール・プロセッサ(CSP)を中心に行われる。CSPは，システムの立ち上げ，実行のスタート/ストップ，各種実行モードの設定，エラー処理，ファームウェアやソフトウェアのデバッグ支援，CPUやメモリの各種レジスタのリード/ライトなどを行うマイクロプロセッサをベースにしたプロセッサである。CSPの構成と機能については他の報告〔2〕で述べられているので本報告では主にCPUとのインタフェースについて述べる。

CSPとCPUとは8ビットのインタフェース・データ・バス，5ビットのアドレス，及びその他の制御信号によって接続され，各種のデータがインタフェース・データ・バスに接続されたレジスタに書き込まれ，また読み出される。図1にインタフェース・データ・バスとこれに接続されたレジスタを，また，表1にレジスタの一覧表を示す。レジスタの読み出し/書き込みは特別に1ビットずつ行うもの以外は8ビットを単位に行われ，この単位に5ビットのアドレスが付けられている。

2．2．制御レジスタ

CSCNTRは8ビットのレジスタでCPUの制御/状態の各種情報を保持する。図2にCSCNTRの8ビットの意味を示す。CMAは，コンソールからCSMIRにセットされたマイクロ命令をアクティブにする制御ビット，BREはDBGFフィールドでブレークを指定したマイクロ命令を実行した時，停止するかどうかを制御するビット，EFMはエラーの発生に無関係に命令を実行することを指定するビット，FERRは強制的にエラーを発生することを指定するビット，KBTはKL0のステップ実行のためのトラップを発生することを指定するビットである。また，STOPS. STOPR. STOPEはCPUが停止したときの要因を示すビットで，それぞれ，STOPRへの書き込みを指定したマイクロ命令の実行により停止したことを示すビット，DBGFフィールドでブレークを指定したマイクロ命令の実行により停止したことを示すビット，そして，エラーの発生により停止したことを示すビットである。尚，この3ビットは読み出すことはできるが書き込むことはできない。

CSCNTRの8ビットのうち，KBTは，KL0のプログラムのデバッグに使用されるもので，これを'1'にしておくと，述語呼出しの度に割込みが入る。これによって述語呼出しの流れをトレースしたり，実行を停止してチェックを行うことができ，非常に強力なデバッグ機能となっている。

また，CSCNTRの他に，CPUに対して動作を指定する1ビットの仮想的なレジスタが3つ(RESET. RUN. STEP)あり，RUN以外は書き込みだけが可能である。RESETはCPUとI/Oをリセットし，RUNはマイクロプログラムを連続的に

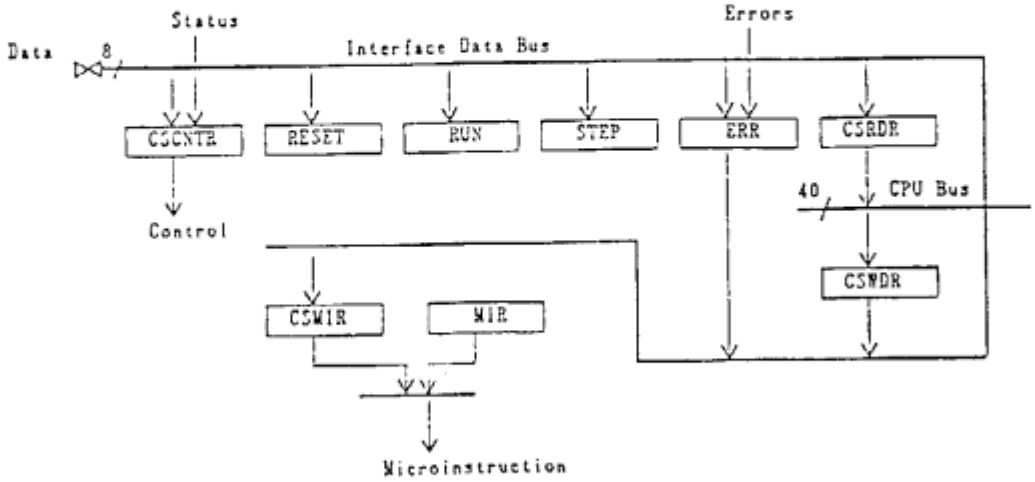


図1．CSPインタフェース

Register	# Bits	Read	Write
CSCNTR	8	o	o *
RESET	1	x	o
RUN	1	o	o
STEP	1	x	o
CSRDR	40	x	o
CSWDR	40	o	x
ERR	32	o	o
CSMIR	64	x	o

* Except SRSTP, BSTP, and ESTP Bits

図１．インタフェース・レジスタ

CMA	: Console Mode Active
BHE	: Break Halt Enable
EFM	: Error Free Mode
KBT	: KL0 Break Trap
FERR	: Force Error
STOPS	: Stop due to STOPR Write
STOPB	: Stop due to Break
STOPE	: Stop due to Error

図２．CSCNTRの制御／状態ビット

実行させることを指定し，STEPはマイクロ命令を１命令だけ実行することを指定する。

マイクロインタプリタの開発においては，STEPを使用して，１ステップずつマイクロ命令を実行させてデバッグを行うことができる。また，チェック・ポイントとなるマイクロ命令でブレークを指定しておき，CSCNTRのBBE ビットを'１'にしておくと，このチェック・ポイントを通る度に停止させることもできる。この時，レジスタの内容などをチェックした後にRUN に'１'を書き込めば，再び実行が開始される。

２．３．　その他のインタフェース・レジスタ

ERR はCPU およびメモリで発生するエラーの原因を保持する３２ビットのエラー・レジスタである。CSRDR とCSWDR はCPU の内部バス（３２ビット・データ＋８ビット・タグ）に接続された共に４０ビットのレジスタで，CSRDRはCPU あるいはメモリ内のレジスタにCPU の内部バスを介してセットするデータを保持し，CSWDR はこのCPU の内部バスに出力されるデータを格納取り込んでいる。従って，CPU やメモリ内のレジスタの読み出しや書き込みにはこのCSRDR およびCSWDR が用いられる。

CSMIR はコンソールからマイクロ命令をセットして実行させるための６４ビットのレジスタで，これを使用すればマイクロ命令でアクセス可能なハードウェア資源を全てコンソールからアクセスすることができる。マイクロインタプリタやKL0 プログラムの実行状態を調べたり，変更したりするのに有効である。

３．　ＲＡＳ

３．１．ＲＡＳ機能

φでは信頼性向上のために，ほとんどすべてのRAM と主なレジスタにはデータ８ビットに対して１ビットのパリティ・ビットが付加されており，データが転送されてレジスタにセットされたときにはパリティ・チェックが行われる。また，パリティ・ビットの付加単位である８ビットが他の８ビットとマージされたり，分解されるような論理においてはパリティのプレディクションが行われている。さらに，主記憶については４０ビットのデータタグについて２７ビットのSECDED （Single Error Correction and Double Error Detection） のECC(Error Correction Code)が付加されており，読み出し時に１ビットの誤りならば自動的に訂正し，２ビット以上の誤りならばその旨報告するようになっている。各所で検出されたエラーは３２ビットのエラー・レジスタERR に集められ，１ビットでもエラーがあるとCSCNTRのESTPビットが'１'になってCPU の動作が停止する。ERR はエラーによる停止後のエラー解析で使用される。

エラーが発生してCPU が停止した場合には，OSP がエラー発生時の状態（ERR の値など）を読んで，フレキシブル・ディスクにログ・アウトするため，後にこのディスクからエラー発生状態を知り，原因を究明することが可能になっている。

３．２．　メモリ系のエラー

アドレス変換においては，３２ビットの論理アドレスが２４ビットの物理アドレスに２段のテーブル変換を経て変換されるが，このとき，PMM (Page Map Memory) に保持されたページ・アドレスにはinvalid ビットが付加されており，このページを使ってアドレス変換を行った場合には割込みが発生するようになっている。これは不正ページ参照と呼ばれる割込みで，プログラムが誤って割り付けされていないページをアクセスしようとした時に発生する。これはシステムのエラーで回復不能の重大な障害と見なされる。

アドレス変換におけるこのような不正ページ参照が発生した場合や，論理アドレスのパリティ・エラーなどその他のメモリ関係の論理にエラーが発生した場合には論理アドレスと物理アドレスを保持するアドレス・レジスタを凍結し，メモリ関係のエラー解析を容易にしている。

４．　まとめ

以上述べたようなシステム制御とRAS が逐次型推論マシンφで実現されており，十分とはいえないまでも，ハードウェア，ファームウェア，ソフトウェアの開発においてかなり有効であった。これにより，実用的な推論マシンへの手掛かりが得られたと思われる。しかし，推論マシンは新しいタイプのマシンであり，従来の機能だけでは不十分で，論理型プログラムの高いレベルでの開発支援やデバッグ支援を十分に行うためにφでの経験を基により高度な機能の研究が必要であろう。

参考文献

〔１〕高　他，「パーソナル逐次型推論マシンPSI のハードウェア設計」，Proc. Logic Programming Conf.. １９８４，Session 8.1.

〔２〕中島　他，「逐次型推論マシンφのハードウェア－コンソール・プロセサの構成と機能－」，本全国大会講演論文集。

逐次型推論マシンψのハードウェア
——コンソール・プロセサの構成と機能——

1B−9

中島 浩　田辺隆司　　瀧 和男
（三菱電機）　（新世代コンピュータ技術開発機構）

1．　はじめに

逐次型推論マシンψ（PSI）には，コンソール・プロセサ（CSP）というマイクロプロセサが内蔵されている。　CSPは，システムの立上げ，異常処理，ファームウェア／ソフトウェアのデバッグ支援などの，ψの開発や運用のための強力な支援機能を有している。

2．　ハードウェア構成

Fig.1 にCSPのハードウェア構成を示す。　ψのCPUには，Table 1 に示すコンソール・インタフェース・レジスタ（CSIFR）があり，CSPはこれらのレジスタを介して，CPUの起動，停止，エラー情報の収集などを行うことができる。　また，CSMIR にCPU内部のレジスタ等をアクセスするためのマイクロ命令を格納し，シングル・ステップの実行を行わせることにより，ψの任意のハードウェア資源をアクセスすることができる。

なお，CSPには標準のI／O機器（マウス，キーボード，etc.）が接続されており，これらの機器のコントローラとしての機能も有している。

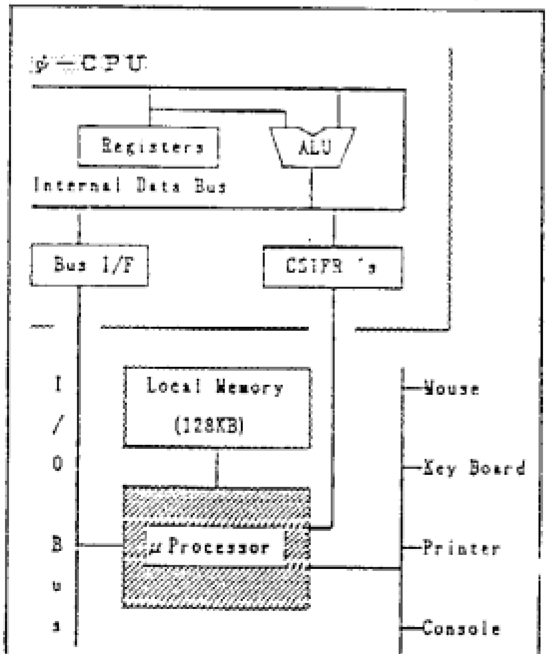


Fig.1 Hardware Organization

Table 1　Console Interface Registers

name	function
CSCNTR	Run/Halt. Execution Mode. etc.
ERR	Hardware Error Status
CSRDR	Put Data to Data Bus
CSWDR	Get Data from Data Bus
CSMIR	μInstruction for Resource Access

3．　コンソール・システム

CSP上にインプリメントされたコンソール・システムでは，システムの立上げ，異常処理，ファイル管理，ファームウェア／ソフトウェアのデバッグ支援を行う。　これらの機能は，73種に及ぶコンソール・コマンドにより会話的に実行される。　以下，各機能について説明する。

3．1　システムの立上げ

Fig.2 に示すように，BOOTコマンドによって，一連の立上げ手順が連続的に行われる。　これらの手順は，コンソール・パネルのスイッチ設定により，電源投入と同時に自動的に行うこともできる。　また，開発の段階に応じて，手順の途中までを行ったり，個々の手順のみを行ったりすることもできる。

```
$ BOOT
$$ CLEAR        ....Register/Memory Clear
$$ INITIALIZE   ....Table Set Up
$$ IMPL         ....Initial μProgram Load
$$ IPL          ....Initial Program Load
$$ START        ....System Start
```

Fig.2　System Start Up Sequence

3．2　異常処理

ハードウェアの重大な障害や，マイクロインタプリタやOSのバグによる続行不能の障害が発生した場合，ψのCPUは直ちに停止する。　コンソール・システムはCPUの停止状態を識別し，状況に応じてコンソール・ディスプレイに障害の内容を表示

する。 また，同じ内容をファイルにログし，障害の解析の便宜を図る。

3．3 ファイル管理

コンソール・システムでは，ファームウェア／ソフトウェアのロード・モジュールを始めとする様々のファイルを管理している。 これらのファイルはフレキシブル・ディスク又は固定ディスクに格納される。 コンソール・コマンドにより，開発中のプログラムのロード／セーブや，ディレクトリの表示，ファイルのコピーや削除などの，ファイルの保守も行うことができる。

3．4 デバッグ支援

ファームウェアのデバッグ支援のために，レジスタの表示／変更，連続／ステップ実行，ブレーク・ポイントの設定や解除などのコマンドが用意されている。 これらの基本的なコマンドの他に，より効率的なデバッグを行うためのコマンドが用意されている。 その一つがFig.3 に示す，停止時のレジスタ表示である。 これは，あらかじめ定義されたレジスタ（複数の定義も可）の内容を，ＣＰＵが停止するたびに自動的に表示する機能であり，コマンド入力の手間を大きく軽減することができる。 また，Work File(WF) の特定のアドレスを，そのアドレスに割付けられた制御用レジスタのニモニック(Logical Name) で参照する機能や，データ・タイプを識別するタグの値をニモニック・コードで指定する機能などもある。

```
> SET BBREAK  78B          ;Define Break Point
> SET BREAK  13AC          ;
> SET DISPLAY 0 PMAR       ;Define Register to
> GO  1F3                  ;Display
%I-BRKPNT, BREAK POINT
PMAR = 78B                 ;Auto-Display
> SET DISPLAY 0 CLBR       ;Current Local Base
> GO                       ;Register (WF 16)
%I-BRKPNT, BREAK POINT
PMAR = 13AC                ;Auto-Display
CLBR = REF(21)  0D00E53C
                           ;REF=reference
```

Fig.3 Firmware Debug Support

ソフトウェアのデバッグ支援のためには，機械語(KL0) レベルでのステップ実行やブレーク・ポイントの他に，主記憶や制御用レジスタの内容があらかじめ定義された値になったときに停止する，データ・ブレーク機能や，実行した述語のアドレスなどの情報をトレースする機能がある (Fig.4)。 これらの機能は，ＫＬ０のマイクロインタプリタと協力して，以下に示す手順で行われる。

(1)コマンドに応じたデバッグ支援機能の内容を，WFの特定アドレスに割付けられた，ファームウェア制御用レジスタ(FCR) にセットする（例えば，データ・ブレーク機能を示すコード，アドレス及びデータをセット）。

(2)ＫＬ０ブレークのフラグをオンにし，述語呼出しのたびに，割込が発生するようにする。

(3)マイクロインタプリタは，述語呼出しを行う際にＫＬ０ブレークの割込を検知し，デバッグ支援用のマイクロルーチンへ分岐する。

(4)デバッグ支援用ルーチンでは，ＦＣＲの内容を解析し，その内容に応じて，デバッグ情報の表示や実行停止などの処理を行う（例えば，データ・ブレークが発生したアドレスとデータを表示して停止する）。

```
> STEP /KL0                ;KL0 Step Exec.
*** ........ IBR = 00012780 ***
> SET ABREAK  0:13A6B      ;Define KL0 Break
> GO                       ;Point
*** ........ IBR = 00013A6B ***
> SET DBREAK 0:20 INT 7    ;Define Data Break
> GO
*** ........ MW(00000020) = 04 00000007
```

Fig.4 Software Debug Support

4．おわりに

コンソール・システムによる強力なデバッグ支援は，ＰＰＯのファームウェア／ソフトウェアの開発の効率化に大きく貢献した。 現在は，システムの評価を行うためのデータ収集機能の追加を行っており，性能評価や改良などに利用される予定である。

データベース・マシンの機能設計について

5F-1

宮崎収兄, 矢吹健男, 栄山武樹, 横田治夫, 村上国男
(財)新世代コンピュータ技術開発機構

1 はじめに

データベース・マシン(DBM)の開発にはその基本処理方式やハードウエア構成だけでなくDBMが全体として実現すべき機能の検討も必要である。

DBM機能の検討はホストとDBMとの間の機能および負荷の最適配分の問題として把えられる。

2 DBMの使用形態

DBMはホストとの関係によって2つに大別できる。

(1)バックエンド型DBM

(2)サーバ型DBM

バックエンド型DBMは図1に示すように1または少数のコンピュータのバックエンドとして用いられる。その導入の目的はホストの性能向上と処理負荷軽減にある。他方サーバ型DBMは一般に図2のようなネットワーク環境で使用され多数のコンピュータにデータベース機能を提供する。この場合の目的はデータベースの共有化・大容量化や性能向上にある。

DBMの設計にあたってはできるだけの機能をマシン側に移しホストの負荷を軽減すべきだとの意見もある。しかしサーバ型DBMでは機能をマシンに集中するとボトルネックとなる恐れもあり負荷バランスを考えた設計が必要であろう。どちらで実現しても処理量があまり変らずまた並列処理による性能向上が期待できないものはホスト側機能とした方が良い場合もある。

DBMのトラヒック・パターンはそのアプリケーションによって変るのでそれを考慮した機能の設計が必要である。例えば定型業務の場合predefined request機能が必須であるし, Rolog

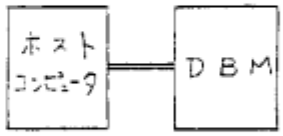


図1 バックエンド型DBM

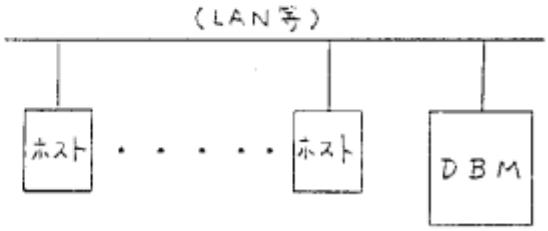


図2 サーバ型DBM

のfactを格納するにはリレーショナル・モデルが最適である。

3 DBMの機能

DBMの機能を論じるにはそれを3種に分類すると良い。

(1)データベース・アクセス機能

(2)データベース制御機能

(3)システム管理機能

3.1 データベース・アクセス機能

ホストより指定された検索・更新などを実行するDBMの基本機能である。従って本機能の性能向上はさかんに検討されている。検討すべき機能として以下のものがある。

(1)DBアクセスのレベル(フィジカル, 内部スキーマ, スキーマ, サブスキーマ)

(2)データ・モデル

(3)DBアクセス言語(コマンド)

(4)インタラクションの単位(タプル/集合)

(5)データ独立性の保持(フォーマットおよびコード, 型, 単位等の変換)

(6)DBアクセス機能の実現度
・関係完備性, 宣言式表現.

・算術演算，aggregate関数
・出力のソート・ユニーク処理
・集合比較
・least fixed point operation
(7) predefined request

3.2 データベース制御機能

データベース制御機能はデータベースアクセスを実行する際のデータ保護などの機能である。DBMとして最低限の機能は必須であるが何をどのように実現すべきかについてアクセス機能ほどには検討されていない。従って実現される場合でも通常のDBMSの技術の応用にとどまっている場合も多い。

(1)複数トランザクションのコンカレンシ制御
(2)データ・リカバリ（ロールバック，バックアップ，ロールフォワード）
(3)セキュリティ管理（アクセス権，暗号など）
(4)インテグリティ管理
(5)ディクショナリ管理（D.D.アクセスの提供，コンカレンシ制御，データとのコンシステンシ維持）
(6)複数DB，共有DB，個人DB
(7)DB administratorのサポート
・バルク・ローディング
・DB再編成，ストレージ管理
(8)データベース設計支援

3.3 システム制御機能

システム制御機能は汎用機では通常DBMSではなくOSの機能である。バックエンド型DBMではある程度ホストに分担させることもできるがサーバ型DBMでは重要な機能となる。

(1)コンソール・インタフェース
(2)システム状態制御
(3)ホストからの状態に応じて，実行中断指示
(4)ユーザ登録，管理
(5)課金処理
(6)システム稼動パラメータの変更
(7)ログテープ等の保守
( )障害検出と回復，系の再構成
( )試験設計
( )機器増設，機能追加
( )評価用データ収集

などが検討課題となる。

4. DBM機能の設計

近年"機能的に完全なDBM"の研究がさかんになってきた。これは基礎研究だけでなく実用化にも重点がおかれてきたことを示唆している。しかし"機能的に完全なDBM"がどのようなものかについてはあまり議論されていない。1つの作業仮説としてここで述べた全項目を高度に実現したマシンを"機能的に完全なDBM"とみなし，すべての処理を実行する各種基本方式のもとでその実現方法を検討することは基本方式の比較やDBM機能の決定を行うための有力な手段であろう。

データベース処理におけるデータベース制御処理のしめる比率はかなり高い。DBMにおけるデータベース制御機能の高性能化が検討されるようになってきたがアクセス処理の基本方式やハードウェア構成との関係はまだ充分解明されていない。DBMの機能の設計においてはこの点の検討が重要である。また以下の点も考慮せねばならない。

(1)DBMの用途・アプリケーションによる機能の必要性
(2)DBMの使用形態，ホストの種別
(3)各機能のDBMおよびホストでの実現方法による性能とコスト比較
(4)将来の機能追加・拡張

DBMのように新しいシステムでは未解決の問題もあり(4)は重要である。

5. まとめ

DBM機能の設計における検討課題と留意点を考察した。機能検討にあたってはホストを含めた全体系の中での機能分担の視点が重要である。

3F-2

Unification in A Knowledge Base Machine

Haruo Yokota, Shigeki Shibayama, Nobuyoshi Miyazaki,
Takeo Kakuta, Kunio Murakami
Institute for New Generation Computer Technology (ICOT)

1. Introduction

It is important for the knowledge information processing system to manage large amounts knowledge efficiently. When we implement such a system using a logic programming language based on Prolog, collections of Horn-clauses can be viewed as a kind of knowledge. If the volume of knowledge exceeds the capacity of main memory, the current version of Prolog processor cannot handle the load. We must consider how to deal with knowledge by using the secondary memory effectively.

Horn-clauses are classified into three types: unit clauses without variables, unit clauses with variables, and non-unit clauses [1]. It is known that a unit clause without variables bears a one-to-one correspondence to a tuple in a relational database. We plan to construct a knowledge base machine as an enhanced relational database machine. We are now developing relational database machine Delta which manipulates streams between the hierarchical memory system and the relational database engine. The knowledge base machine then handles Horn-clauses by manipulating the streams.

We have presented a draft for putting three types of Horn-clauses into the knowledge base in a tuple style [1]. In order to retrieve all types of clauses, we proposed using the unification command as a knowledge base operation; it is treated as an enhanced relational algebra command. In this paper, we further investigate that approach.

2. Scheme for Storing Horn-clause

We group clauses by name and arity (number of arguments) of their positive literal. Each group forms a table. The name of the positive literal in the grouped clauses becomes the name of the table. A tuple in the table corresponds to a clause and has a number of attributes. Clause arguments are converted to attributes on a one-to-one basis. Two special attributes are provided for the bodies of clauses and for the most general unifiers generated by the unification commands. If a clause has no body, the corresponding attribute is empty.

Example 1: We use the DEC-10 Prolog syntax here. If there are four clauses, such as:

```
p(abc,12).
p(efg,22).
p(hij,X) :- q([klm,X],Y),r(Y).
p([X,Y],11) :- q([X],stt(klm,Z,Z)), Y \= Z.
```

the following table, named p, is formed.

arg1	arg2	body of clause	most general unifier
abc	12	-	-
efg	22	-	-
hij	X	q([klm,X],Y),r(Y)	-
[X,Y]	11	q([X],stt(klm,Z,Z)),Y\=Z	-

When a query *?-p(A,12)* is made, the knowledge base machine searches for unifiable tuples and generates the most general unifiers for the clauses found, as follows:

arg1	arg2	body of clause	most general unifier
abc	12	-	{A/abc}
hij	X	q([klm,X],Y),r(Y)	{A/hij,12/X}

From the latter table, one answer, *A=abc*, is obtained and a new query, *?-q([klm,12],Y),r(Y)*, is created. []

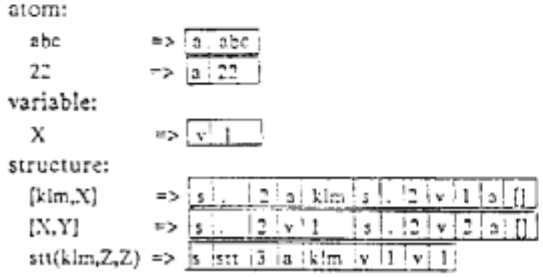
3. Argument Forms

An argument of a Prolog clause is formed by either an atom, a variable, or a structure. We distinguish each type using tags. An atom is merely a character string with a tag. A variable can be represented using a sequential number with a tag. Since the current version of Prolog does not support global variables, a variable is used to relate the argument to other arguments in the clause.

The Prolog processor does not place structures in the same location at which arguments are stored. There are pointers to indicate the location at which the structures are actually stored. It is for this reason that structures are treated dynamically in Prolog. For instance, a variable can be unified with any kind of structure.

However, when unifiable clauses are searched for, instantiation is not necessary. It is only needed to verify whether the variables are unifiable with the structure. Moreover, if we use streams to retrieve clauses, pointers are not suitable. Thus, in the knowledge base machine, structures may be placed at the same location at which the arguments are stored. Structures are represented by functor name, argument count, and a list of the arguments. Notice that variable-length records must be supported for storing structures.

Example 2: Arguments appearing in Example 1 are represented as follows:



Here, [a,b] stands for .(a,.(b,[])). []

4. Operational Flow

When a knowledge base query, which is a Prolog goal sequence, is made, one goal is selected from the sequence and unifiable clauses are searched for in the table having the same name as the goal. A new goal sequence can be created from each unifiable clause and the most general unifier for that clause.

If a unifiable clause has no body, the rest of the sequence to which the most general unifier has been applied becomes a new goal sequence. Otherwise, i.e., if the unifiable clause has a body, the most general unifier is applied to both the body and the rest of the sequence. A concatenation of these two instances becomes one of the new goal sequences. Emptiness of the new goal sequence indicates that one of the answers to the query has been obtained.

Concurrently obtaining an entire new goal sequences for the query implies the use of OR-parallelism. However, further new goal sequences must be generated concurrently for each obtained goal sequence. The number of goals that must be handled concurrently increases rapidly. This phenomenon is called OR-parallelism explosion.

There are three types of optimization for sequentially generating new goal sequences. First, we allow processes to be invoked in the inference mechanism, especially recursive call processes, which access only main memory. Consequently, we partition knowledge into two types, according to its volume and contents. Small quantities of iterative knowledge are put into the inference mechanism. Large quantities of knowledge are stored in the knowledge base mechanism. To take this approach, some interface between the inference mechanism and knowledge base mechanism is needed. In [2], we propose a deferred evaluation method for interfacing Prolog with relational databases. We can improve this method to interface these two mechanisms without massive modifications.

Second, we can group processes that access the same table. A head literal in the body of the unifiable clause indicates a table that will be accessed in the next stage. We can collect clauses that have the same head literal in each body by going through the clause-body attribute of the unifiable clauses table being searched.

Last, we can control the order of clause selection according to the literal count in the clause body. The length of the body often indicates the complexity of the resolution process when the process does not contain a recursive call. The count of the negative literals can be used as a kind of evaluation function for priority control. A unifiable clause that has a shorter body is invoked prior to those having longer bodies.

5. Conclusion

We have presented a method that stores any type of Horn-clauses in the knowledge base. We have shown that atoms, variables and structures can be handled by the knowledge base. It is difficult to implement OR-parallelism in the knowledge base mechanism. For sequential execution, we have proposed three types of optimization.

Acknowledgement

The authors thank the members of ICOT Working Group 1 (KBM subgroup) for their valuable discussion.

References:

1. H. Yokota, et al., An Investigation for Building A Knowledge Base Machine, *Proceeding of the 27th IPSJ Conference*, 2K-7 (Nagoya, October, 1983). [in Japanese, English version to appear as ICOT Technical Memorandum]
2. H. Yokota, et al., An Enhanced Inference Mechanism for Generating Relational Algebra Queries, *Proceeding of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp.229-238 (Waterloo, April, 1984).

Relational Database Processing on an Attribute-Based Schema

Shigeki Shibayama, Takeo Kakuta, Nobuyoshi Miyazaki
Haruo Yokota, Kunio Murakami

Research Center
Institute for New Generation Computer Technology (ICOT)

1. Introduction

ICOT's Relational database machine (Delta) has adopted an attribute-based database schema for storing relations in secondary storage devices (moving-head-disks). The reasons we selected the attribute-based schema are: a) predefined access paths are not available, b) unnecessary accesses to disk should be minimized, and c) tuples are expected to be accessed evenly. In this paper, we will investigate the properties associated with this schema.

2. Attribute-based Schema and Tuple-based Schema

The differences between attribute-based schemas and tuple-based schemas are as follows:

(1) Tuple reconstruction time: attribute-based schemas must reconstruct corresponding attributes for output or tuple-oriented operations such as union with duplicate elimination.

(2) External storage access cost (both seek + latency time and data transfer time): tuple-based schemas must access extra (unnecessary) attributes in tuples not used in the query.

(3) Clustering of tuples/attributes: there are a number of methods of storing tuples/attributes in clusters and provide access to them. This is closely related to how symmetrically the access methods can handle the attributes of a relation.

(4) Space efficiency: an attribute-based schema usually requires tuple-identifiers (tids) to indicate tuple relationships among attributes. This requires additional storage.

(5) Temporary relation representation: in a tuple-based schema, usually, temporary relations must actually be made. With an attribute-based schema, temporary relations exist in concept only; however, efficient internal representation of temporary relations is possible.

3. An Attribute-Based Schema with Two-Level Clustering

In Delta, we have chosen an attribute-based schema with two-level clustering [1]. An example of this clustering is shown in Fig. 1. This schema is devised to match the requirements for a database machine used within a logic programming research environment. In particular, the symmetrical attribute access property is thought to be essential. In Prolog, a representative logic programming language, facts (unit clauses with no variables) directly correspond to tuples in a relational database. Prolog facts do not explicitly incorporate the concept of keys. We believe that unequal processing times for manipulating various attributes is an undesirable feature for database machine users. Therefore, we decided to handle all the attributes symmetrically. The appropriateness of this specific schema remains to be evaluated.

4. Hardware and Software Requirements for Attribute-Based Schema

There are several hardware and software requirements for adopting an attribute-based schema in a database machine.

1) Fast join capability: generally, the process of performing relational database operations with an attribute-based schema consists of, a) producing a tid list denoting result relations, and b) generating result relations from the tid list. As the attributes used in the query are stored separately with their corresponding tids, tid-restrictions to the tid list are necessary for all the attributes. This is the same, from a processing point of view, as equi-joins between attributes. If equi-joins are performed slowly, overall performance will not be satisfactory.

2) Fast tuple/attribute transposition capability: tid-restriction is the first stage of generating result relations. After the tid-restrictions are complete, attributes (tid-restricted and sorted with the tids) must be transposed with tuples.

3) Cluster-formation capability: this is specific to the two-level clustering schema of Delta. After the transposition of attributes (in the form of attribute buffers), each attribute must go through the clustering process. This is done by, a) sorting each attribute with the attribute value, b) dividing the value-sorted buffers into a certain number of smaller sections (zones) according to value ranges, c) sorting each zone with the tid-values, and d) dividing each zone into a number of smaller sections (pages) according to the tid value ranges and creating a directory for the clusters.

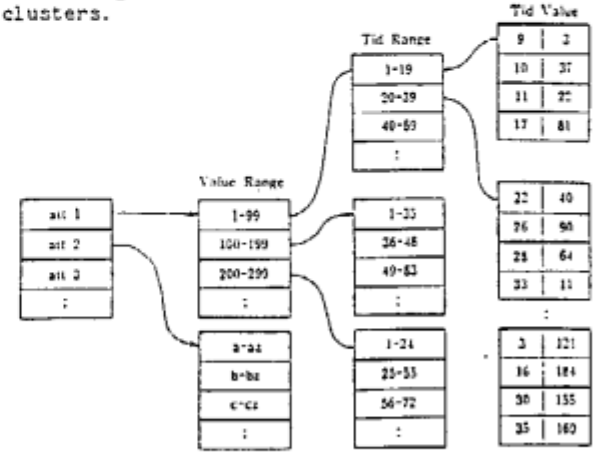


Fig. 1 Delta's Attribute-Based Schema

5. An Example of Relational Database Command Processing using the Attribute-Based Schema

Consider the following SQL-like query:

```
[Sample Relations]
   icot(name,age,company,laboratory,group)
   company(company_name,location)

[Sample query]
   select name, laboratory, group
   from icot
   where company = next
   select company_name
   from company
   where location = [yokohama]
```

The following lists the processes performed in response to this query. Shared item numbers indicates source-result relationships. (Increasing primes indicate successive source-result relationships.)

```
(1)  prepare comp-location attribute
(1)' tid(c):comp-location (=[yokohama])
(1)''tid(c):comp-location (sort by tid(c))
(2)  prepare comp-name (= tid(c) of (1)'')
(2)' tid(c):comp-name (= tid(c) of (1)'')
(3)  prepare icot-company (= comp-name of (2)')
(2)',(3)->(3)' new-tid:tid(c):tid(i)
     (natural-join by comp-name and
     icot-company)
(3)''new-tid:tid(c):tid(i) (sort by tid(i))
(4)  icot-name (= tid(i) of (3)'')
(4)' new-tid:icot-name (tid-restriction by
     tid(i))
(5)  icot-lab (= tid(i) of (3)'')
(5)' new-tid:icot-lab (tid-restriction)
(6)  icot-group (= tid(i) of (3)'')
(6)' new-tid:icot-group (tid-restriction)
(4)''new-tid:icot-name (sort by new-tid)
(5)''new-tid:icot-lab (sort by new-tid)
(6)''new-tid:icot-group (sort by new-tid)
(4)'',(5)'',(6)''->(7) icot-name:lab:group
     (transposing to tuples)
```

here, tid(c), tid(i) and new-tid denote tid of company relation, icot relation and generated new tid, respectively. For example, new-tid:icot-name denotes a buffer with a data structure of new-tid and icot-name attribute.

As can be seen from this example, attribute-based schemas carry out essential database query operations using concise representations. Queries that frequently handle large temporary relations are performed efficiently with this schema. The most time-consuming factor of this schema is the actual formation of result relations. This requires, a) readout of output attributes from a disk device (with disk cache), b) tid-restriction of the output attributes and, c) transposition of attributes with tuples. In Delta, a large semiconductor disk (about 120MB) is incorporated for disk cache use to improve the hit ratio and for buffer use to ensure that almost all the intermediate results in a query remain in high-speed storage. Frequent operations with this schema, such as tid-restriction and join, are performed using special-purpose hardware called the Relational Database Engine (RDBE) [2].

The relative performance is compared to that of result tuple selectivity in Fig. 2. Estimation model is a selection put in SQL as:

```
select A1,A2,..,An
from A
where Ai (cmp) C(criterion)
```

The following conditions are assumed:

(1) a specific page can be identified by either its tid or value
(2) tid-restricted attributes are read from disk using a secondary index
(3) average seek and latency time are required for each page access
(4) tid-restriction requires buffer memory read and write
(5) transposition requires 10 buffer read/write operations
(6) there is constant overhead time (accumulation of overhead associated with each operation)

The estimate shows that, a) a full scan of tid-restricted attributes is required for a selectivity range larger than 1 % , b) tuple reconstruction time affects performance for selectivity ranges larger than 10 % , c) tid-restriction time is short compared with disk access time, and d) overall processing time increases gradually. Evaluation of the join operation will show similar results, because most of the processing time consists of tid-restricted attributes access and transposition time.

6. Conclusion

Attribute-based schemas can exploit efficient intermediate representations of temporary relations. The processing cost lies in read-out of tid-restricted attributes. Tuple reconstruction time is not very large in low selectivity ranges.

[References]

[1] Onibayama, S., et al., "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor," New Generation Computing, Ohmsha-Springer, Vol.2, No.2, pp.131 - 135.

[2] Oka, T., et al., "Development of a Relational Database Engine (1) to (6)," Proc. 29th IPSJ Conference, 4F-5 to 4F-10.

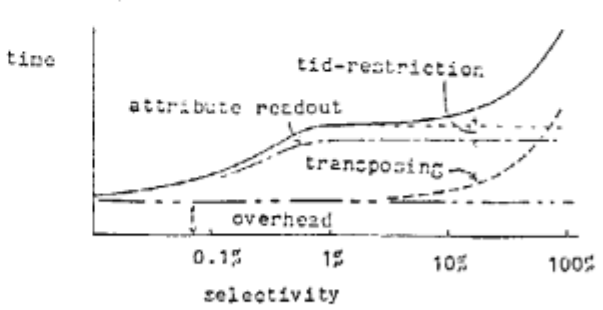


Fig. 2 Performance Estimation

RDBM Delta のリカバリ方式

3F-4

角田健男, 宮崎收兄, 栗山茂樹, 横田治夫, 村上国男
(財)新世代コンピュータ技術開発機構

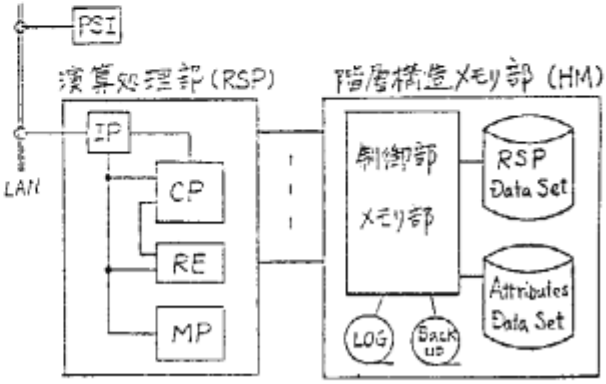
1. はじめに

ICOTで開発中の関係データベースマシン(RDBM) Deltaは、ホストマシンとなる複数台のパーソナル逐次型推論マシン(PSI)とLANを介して接続され、本プロジェクトの中期研究開発課題の1つである知識ベースマシン開発のため基礎的実験を行うこと、およびPSI上のユーザプログラムからの外部データベースアクセス要求を効率良く処理することを主目標としている。外部データベースには英和・和英辞書などの大容量共用データや各ユーザ専用のデータベースなどが格納され、検索・更新処理の対象となるため各種障害発生時のデータベースのリカバリ処理を考慮し実現する必要がある。
本稿では、Deltaのリカバリ方式の概要について報告する。

2. Deltaのデータ格納

Deltaでは、リレーションを構成するタプルをアトリビュートに分割し、アトリビュート毎に格納する縦型格納法[1]を採用した。一方、リレーション管理情報には、ホストが参照するディクショナリとDeltaのControl Processor (CP)がDeltaコマンド解析・実行やリカバリ処理に使用するディレクトリがある。階層構造メモリ部(HM)のメモリ管理上、アトリビュート情報とディクショナリ情報はアトリビュート・データセットとして、またディレクトリ情報はRSPデータセットとして取扱う。
Deltaシステム構成を図1に示す。

Deltaのリカバリ処理は、Deltaコマンドの実行を制御するCPとDeltaシステムの状態管理、オペレータインタフェースを処理するMaintenance Processor (MP)との制御下でHM内データセットを、障害発生時に実行中のトランザクション開始前の状態に復旧させることである。



IP: Interface Processor, RE: RDBM Engine
RSP: RDBM Supervisory and Processing subsystem
HM: Hierarchical Memory subsystem

図1. Deltaのシステム構成

3. Deltaのリカバリ処理基本方針

(1) 更新処理を含むトランザクションの開始後、トランザクション凍結前までにホストからのアボート指示またはDelta内で検出したDead Lockやリソース不足等のときトランザクション開始時点の状態まで戻し、実行中の更新処理を全て取消す。

(2) Deltaシステムダウン時：システム再立上げ後、オペレータによるシステム状態チェックを行ない、コミット/ロールバック指示に基づいてリカバリ処理を行う。

(3) 媒体系の障害時：人間による原因調査、装置構成変更等の後、バックアップ情報とログ情報によりロールフォワードし、次にロールバックを行う。

(4) 電源障害時：無停電電源装置を使用し電源ダウン時のデータベース破壊を防止すると共に処理再開できる環境を確立後システムダウンさせる。その後の処理は(2)と同様である。

4. コンカレンシィ制御

Deltaでは、2フェーズ・ロック方式[2]に基づくコンカレンシィ制御をCPで行い、最大64個までのコマンド・トランザクションを同時動作させる。

(i) ロックの単位：リレーション

(ii) ロックの期間：トランザクションの開始から終結または終了まで。

(iii) ロックの種類

- READロック（シェアラブル）
- WRITEロック（エクスクルーシブ）

(iv) ロックの相互関係

新ロック要求 ＼ 既ロック	READ	WRITE
READ	○	×
WRITE	×	×

○：許可
×：禁止

5. Deltaのリカバリ方式

Deltaのデータベース・リカバリ処理方法は、以下の2ケースにまる。

① オンライン・ロールバック
② オフライン・ロールバック

(i) リカバリ用サブコマンド

CPで管理するディレクトリには、リレーションおよびアトリビュート・スキーマ情報があり、実行中のトランザクションID、ロックするリレーションIDとリカバリ用に上記時スキーマ情報のシャドウ情報をトランザクション毎に持つ。一方、HMでは各アトリビュートID毎にアトリビュート情報を管理し、リカバリ用にアトリビュート情報のシャドウを持つ。

CPは、コマンド・トランザクションの同時実行のためトランザクションID毎にDeltaコマンドの解析の後各サブコマンドに分解生成し、HMやREを起動し、その実行を管理する。

CPからHMへ送出される検索、更新系サブコマンドには、トランザクションIDやアトリビュートIDなどのパラメータが含まれている。

リカバリ系サブコマンドには次の2つがあり、これもCPからHMに送出される。

(a) Commit-atributesサブコマンド

本サブコマンドで指定のアトリビュートID群とトランザクションIDが更新処理実行中のトランザクションIDとアトリビュートID群と一致するとき、リカバリ対象と判定しそのアトリビュート情報をコミットする。

(b) Rollback-atributesサブコマンド

パラメータ判定によりリカバリ対象であるときは、指定アトリビュートをロールバックする。

(ii) オンライン・ロールバック

更新処理中に上記リカバリ系サブコマンドを発行したときのディレクトリとアトリビュート情報の状態遷移を下表に示す。

HM ＼ サブコマンド	ディレクトリ		アトリビュート情報		備考
	新エントリ	シャドウ	旧エントリ	シャドウ	
更新系 → Commit-att.	(更新) → 新エントリ	旧エントリ → 廃棄	(更新) → 新エントリ	旧エントリ → 廃棄	
更新系 → Rollback-att.	(更新) → 廃棄	旧エントリ → 旧エントリ	(更新) → 廃棄	旧エントリ → 旧エントリ	トランザクション アボート (オンライン)

(iii) オフライン・ロールバック

3の(i)～(ii)項に記す通りリカバリ処理を行う。

6. おわりに

Deltaのリカバリ処理を実現するための障害の分類と処理手順について更に詳細検討を進め、Deltaのリカバリ機能を充実させてゆく予定である。

日頃Deltaの開発に尽力いただいている(株)東芝および(株)日立製作所の各位に感謝致します。

参考文献

[1] 宮崎他，データベースマシンにおけるデータ格納法に関する一考察，27情処全大（1983.10）

[2] Eswaran et.al., The Notions of Consistency and Predicate Locks in a Database System, Com. of the ACM, No.11, Vol.19, 1976

ＳＩＭＰＯＳのオペレーティング・システム概要

服部　隆　　辻順一郎　　内田俊一　　横井俊夫
（（財）ＩＣＯＴ）

１．はじめに

ＳＩＭＰＯＳ(Sequential Inference Machine Programming and Operating System)は逐次型推論マシンＰＳＩのためのプログラミング／オペレーティング・システムである。ＰＳＩはＩＣＯＴで開発されたソフトウェア開発用パイロット・モデルであり、ＳＩＭＰＯＳはそのために新規に設計、開発されている。本稿では、ＳＩＭＰＯＳのオペレーティング・システムの構築概念とシステム構成を述べる。

ＰＳＩのプロセサとメイン・メモリは、核言語第０版（ＫＬ０）と呼ばれるＰｒｏｌｏｇベースの機械語を効率良く実行するように設計されている。また、入出力装置として、ハード・ディスク、フレキシブル・ディスク、ビット・マップ・ディスプレイ、オプティカル・マウス、およびローカル・エリア・ネットワーク（ＬＡＮ）インタフェースなどを装備している。

ＳＩＭＰＯＳの主目標は、ＰＳＩの特質や装置を効果的に使用して、ＰＳＩを良好な論理型プログラミング・ワークステーションとすることである。また、その設計思想は機能の豊富さよりも簡潔性と拡張性を重視している。これは、パイロット・モデルとして今後の種々の試みを容易に実現できるようにするためである。

オペレーティング・システムはプログラミング・システムの下にあって、マルチ・プロセシング、マルチ・ウィンドウ、ディスク・ファイル、およびネットワーク通信などをサポートしている。プログラミング・システムは、テキスト／プログラム・エディタ、デバッガ／インタプリタ、ライブラリ／コンパイラなどを含み、それらとユーザとの会話はコーディネータにより管理される。

２．構築概念

ＳＩＭＰＯＳではＰｒｏｌｏｇにモジュール性と副作用を導入する基本的手段としてオブジェクト指向的アプローチを選んだ。オブジェクトは、外部的にはそのオブジェクトに許される（つまり、オブジェクトが受けつける）操作の集りにより定義され、内部的には操作を記述するクローズおよび前や他のオブジェクトを保持するスロットで定義される。同じ定義をもつオブジェクトはクラスとして分類され、それらのテンプレートはクラス定義で与えられ、各オブジェクトはこのテンプレートから生成される。ＳＩＭＰＯＳのシステム記述言語であるＥＳＰでは、クラス定義は以下のシンタックスで与えられる。

```
class クラス名
    〔マクロ・パンク宣言〕
has
    〔継承定義：〕
    〔クラス・スロット定義：〕
    〔クラス・クローズ定義：〕
〔instance
    〔インスタンス・スロット定義：〕
    〔インスタンス・クローズ定義：〕〕
〔local
    〔ローカル・クローズ定義：〕〕
end.
```

クラスはいくつかのクラスを多重継承して定義できる。そのクラスは外部的に、自ら定義した操作に加えて、継承したクラスの操作をすべてもつことになる。内部的には、これらのクラスにある同じ操作のクローズがＯＲされ、スロットが集められる。また、デーモン述語をクラスおよびインスタンス操作に定義できる。これはプライマリ述語の前か後にＡＮＤされ、暗黙的に呼出される。これらの機構はオペレーティング・システムの記述に有効である。

３．システム構成

オペレーティング・システムの機能は、最小限の機能をもつ、限られた数の基本クラスと、これらを継承した付加機能をもつクラスとして提供され、アプリケーション・プログラムはこれらクラスのオブジェクトを介してその機能を利用する。オペレーティング・システムでは、これらのシステム・クラスを入出力メディア・システム、実行管理（スーパバイザ）、および資源管理（カーネル）の３層に分割する。

３．１　資源管理

カーネルはprocessor やareaで表現されるハードウェア資源を管理し、diskやkeyboardなどで表現される物理入出力装置を制御する。なお、プロセサやメイン・メモリはファームウェアで基本的に管理されている。

また、カーネルはＩＰＬ(Initial Program Loader)やスタンド・アロン・デバッガであるシステム・トレーサをも含む。

３．２　実行管理

スーパバイザはＳＩＭＰＯＳにおける実行モデルをサポートする。そのためのクラスとして、process(実行主体)、program(処理手続)、stream (プロセス間通信)、pool (オブジェクトの収納)、world(実行環境) などを含む。このモデルでは、プロセスがワールドの下でプログラムを実行することでオブジェクトを操作し、また必要ならば、ストリームを介して他のプロセスと通信する。

なお、プログラムはクラスas-programを継承したユーザ定義のクラスとして与えられる。

３．３　入出力メディア・システム

入出力メディア・システムは、window (ユーザと会話的に通信する)、file (ディスクにデータを格納、検索する)、cable (ネットワークを介して他のマシンと通信する) などのクラスで論理入出力機能をサポートする。

(１) ウィンドウ・サブシステム

ウィンドウ・サブシステムは、ビット・マップ・ディスプレイ (スクリーン)、キーボード、およびマウスを制御して、複数の会話セッションをユーザに提供し、並行していくつもの仕事ができるようにする。この機能は、マルチ・ウィンドウと呼ばれ、ワークステーションでは必須の機能となりつつある。ＳＩＭＰＯＳでは、多重継承の機構を利用して、様々なタイプのウィンドウをユーザが容易に定義できるようにしている。このため、ウィンドウ・サブシステムはウィンドウ構成用クラスを提供している。

また、ウィンドウ・マニピュレータにより、ユーザはマウスを使ってスクリーン上のウィンドウを操作することができる。

(２) ファイル・サブシステム

ファイル・サブシステムは、ハード・ディスクおよびフレキシブル・ディスクを管理し、データの格納や検索をサポートする。ＳＩＭＰＯＳで扱うファイルは、一般のデータを対象とするデータ・ファイルと、オブジェクトを対象とするインスタンス・ファイルとに分類される。

データ・ファイルには、バイナリ・ファイル、固定長レコードを格納するテーブル・ファイル、可変長レコードを格納するヒープ・ファイルがある。アクセス法として、ダイレクト・アクセス (ファイルに対し) およびシークエンシャル・アクセス (ファイル・タップに対し) がサポートされる。

インスタンス・ファイルには、保存したいオブジェクトを格納し、後でそのオブジェクトを取出して再生できる。例えば、ファイルやユーザといったオブジェクトがインスタンス・ファイルに格納される。さらに、ディレクトリ・ファイルを使って、これらのオブジェクトに名前を付けて管理できる。

また、ユーザはファイル・マニピュレータにより、ウィンドウを利用したファイルの操作ができる予定である。

(３) ネットワーク・サブシステム

ネットワーク・サブシステムは、ＬＡＮで接続されたマシン (ＰＳＩやＤＥＬＴＡ) 間でのデータ通信を実現する。ＳＩＭＰＯＳがサポートするネットワーク通信は３つのレベルに分類される。まず、マシン間通信のレベルは、異種マシン間の通信に使われ、データの入出力として扱われる。次に、プロセス間通信のレベルでは、異なるＰＳＩ上のプロセスの間でチャネルを介したメッセージ通信を可能にする。最後に、リモート操作のレベルでは、他のＰＳＩ上のオブジェクト (リモート・オブジェクト) を操作する機能を実現するためのサポートを提供する。

４．おわりに

ＳＩＭＰＯＳは、新しいマシン上の新しいシステムであり、様々な試みがなされているが、限られた環境下で開発できたことについては、正しい選択であったと考えられる。また、利用上の評価はこれからユーザがアプリケーション・プログラムを開発し始めたときになされるであろうが、そこからのフィードバックによりＳＩＭＰＯＳをよりよいものにしていく予定である。

なお、今回のＳＩＭＰＯＳオペレーティング・システムの紹介では、以下の順でその各部分を説明する。

- オペレーティング・システム概要 (本稿)
- 資源管理
- 実行管理 － プロセスとストリーム
- 実行管理 － プール
- 実行管理 － ワールド
- ウィンドウ・サブシステム
- ネットワーク・サブシステム
- ファイル・サブシステム
- ＩＰＬ方式
- システム・トレーサ

参考文献

T.Hattori, et al., "An Operating System for Sequential Inference Machine PSI", ICOT TM-0061(日本語版 TM-0065).

ＳＩＭＰＯＳの資源管理

川上　幸仁　（三菱電機㈱）
上三　尚純　（三菱電機㈱）
堀江　健志　（三菱電機㈱）
服部　隆　（㈱ＩＣＯＴ）

１．はじめに

第５世代コンピュータシステム研究開発プロジェクトの一環としてＩＣＯＴを中心に、知識情報処理ソフトウェアの開発用ツールとして、逐次型推論マシンφ（ＰＳＩ）とそのプログラミング／オペレーティング・システム（ＳＩＭＰＯＳ）を開発中である。本論文では、ＳＩＭＰＯＳのうち、その最下層に位置し、φのハードウェア資源（プロセッサ，メモリ，入出力デバイス）を管理する資源管理（カーネルとも呼ばれる）について報告する。

２．資源管理の概要

資源管理の目的は、つぎの３つである。

○ハードウェア資源をオブジェクトとして記述し、ＳＩＭＰＯＳの基本構成概念であるクラスの枠組にのせること。

○実行管理プログラム（スーパバイザ）およびファームウェアと協力して、６３個までのプロセスが並行動作できるマルチ・プロセス環境を提供すること。

○デバイス仕様から独立した高い論理レベル・インタフェースの入出力実行機能を提供すること。

φのハードウェア資源としては、

（ｉ）プロセッサ，プロセス・コントロール・ブロック

（ii）エリア，ページ・マップ・メモリ，物理ページ

（iii）各種入出力デバイス

があり、これらを管理するソフトウェアをそれぞれ、プロセッサ管理、メモリ管理、デバイス管理と呼ぶ（図１）。

３．プロセッサ管理

プロセッサ管理は、２つのクラスprocessor，context　からなる。クラスprocessorは、φのＣＰＵに対応したクラスであり、インスタンスは１個である。割込インデックス・テーブル、割込原因コード・テーブル、ＣＰＵ内システム・レジスタへのアクセス機能および、ＣＰＵを制御する各種特権組込述語の実行機能を実現する。

プロセス実行に必要なハードウェア資源として、つぎのものがある。

（ｉ）１組のＰＣＢ（Pocess Control Block）／ＥＰＣＢ（Extended PCB）

（ii）４個のＡＣＢ（Area Control Block）

（iii）ページ・マップ・メモリ領域

これらを総称してハードウェア・コンテクストと呼ぶ（図２）。クラスcontextは、このハードウェア・コ

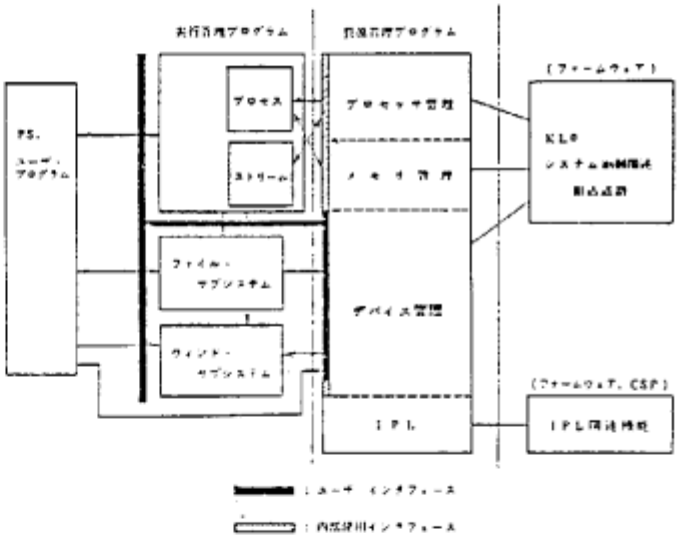


図１．資源管理プログラムの機能分担

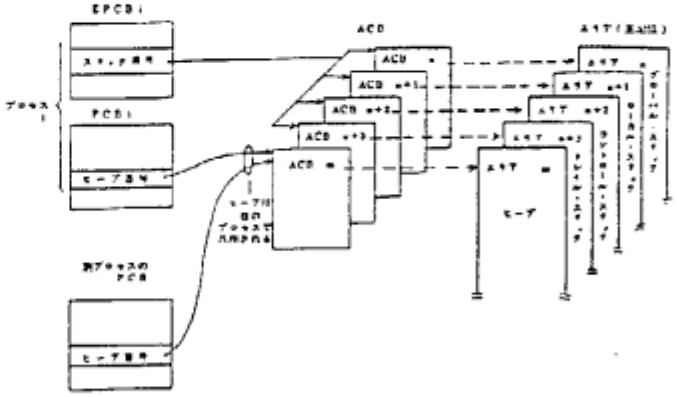


図２．ハードウェア・コンテクスト

ンテクストに、ソフトウェアでプロセスを管理するためのいくつかの情報を付加したものである。インスタンスは、ハードウェアのＰＣＢの数と同じく６３個、存在する。

４．メモリ管理

前述のように、プロセスを実行するとき、４個のスタックに対応した４個のエリア（ＡＣＢ）と、それぞれのエリア毎に、そのページ数分の大きさのページ・マップ・メモリ領域を使用する。ページ・マップ・メモリとは、すべてのエリアの論理ページ・アドレスを物理ページ・アドレスに変換する変換テーブル（ハードウェア・レジスタ）である。プロセスの実行に伴って、ヒープ・エリアの伸長と、スタック・エリアの伸縮があるが、エリアを拡張するときの、ページ・マップ・メモリの確保と、物理ページの確保、および、物理ページ・アドレスのページ・マップ・メモリへの書き込みは、ファームウェアが行う。すなわち、ページ・マップ・メモリ管理の一部と、物理ページ管理は、ファームウェアが分担している。

ソフトウェアとしてのメモリ管理は２つのクラス area, page-map からなる。ハードウェアのＡＣＢに対応するクラスareaには、インスタンスが２５６個あり、各インスタンスは、ヒープ、４種類のスタックのいずれかとして用いられる。クラスpage-mapはページ・マップ・メモリに対応し、ページ・マップ・メモリの領域管理を行う。

上層の実行管理プログラムがプロセスを生成するときには、プロセッサ管理、メモリ管理を用いて、つぎのように行う。

（１）contextのひとつを確保する。
（２）４個のareaを確保する。
（３）それぞれのareaに対し、適当な大きさのpage-mapを割付ける。
（４）page-mapの割付けられた４個のareaをcontextにセットする。

５．デバイス管理

デバイス管理は、入出力デバイスに対応した下記のクラス、

・disk-with-lrucache
・flexible-disk
・bitmap-display
・keyboard
・mouse
・printer
・console-CRT

制御プログラムのクラス、

・device-handler

上層とデバイス管理との間で受渡しされる制御テーブル（入出力メッセージと呼ぶ）のクラス、

・io-message

および、内部で使用するクラスからなる。

入出力デバイス毎に割込番号が決められており、割込が発生するとその番号に対応したプロセス（割込プロセスと呼ぶ）がハードウェアにより起動される。その割込プロセスで実行されるプログラムのクラスがdevice-handlerである。

disk-with-lrucacheは、ディスク・キャッシュ付きの固定ディスクのクラスである。マルチパス・メモリをセット・アソシアティブ方式のキャッシュ・メモリとして使用することにより、ディスク入出力の高速化をはかった。

ビットマップ・ディスプレイは、ビットマップ・メモリの任意の大きさの矩形領域を単位として表示／転送処理を行う装置である。ビットマップ・ディスプレイに対応したクラスbitmap-displayでは、

○矩形のビットマップ・メモリの割り付け／解放
○ビットマップ・メモリ間の演算転送
○マーカの画面への表示

などの機能を実現している。

上層のサブシステムで入出力を行うには、つぎのようにする。

（１）io-messageをひとつ確保する。
（２）io-messageに必要な入出力パラメータ、データ・バッファなどをセットする。
（３）実行管理プログラムのチャネル機能を用いてio-messageをデバイス管理に送る。
（４）チャネルを介してデバイス管理からio-messageを受取り、入出力動作の終了を知る。

６．おわりに

φのハードウェア資源であるプロセッサ、メモリ、入出力デバイスの管理が、ＳＩＭＰＯＳの中でどのように実現されているかについて述べた。性能の評価と改良が今後の課題である。

ＳＩＭＰＯＳの実行管理 －プロセスとストリーム

ムラセ

島津秀雄、　吉田紀彦、　斎藤博一、
（日本電気（株））（（株）三菱総合研究所）（ビーコンシステム（株））
渡辺久晃、　服部隆
（沖電気工業（株））　（（財）ＩＣＯＴ）

１．はじめに

対象指向の言語でＯＳを構築する場合、プロセスの概念をうまく導入させる事は、重要な事柄である。本稿では、ICOTで開発した逐次型推論マシンPSI 上のOSであるSIMPOSのOS核部に含まれるマルチプロセスの実現機構とプロセス間同期・通信機構について述べる。SIMPOSは対象指向の機能を含んだ論理型言語ESP で記述されており、本稿で述べるサブシステムも対象指向的に構築されている。

２．マルチプロセス機構

２．１　プロセス

SIMPOSは、対象指向システムとして構築されているため、システムのすべての基本構成要素はオブジェクトである。

オブジェクトで構築されたシステムが実際に計算機上で動作するためには、各オブジェクトにcontext(実行環境）を与える必要がある。

それを実現するのに、

（１）　最も並列性を重視したものとしては、システム中に存在するオブジェクトの１つ１つに別のcontext を与えるactor 理論のような方法がある。

（２）　最も逐次的なものとしては、すべてのオブジェクトの実行を同一のcontext 上で行なうという方法がある。

SIMPOSは逐次マシン上に構築されているので、オブジェクトの１つ１つにcontext を与えても、実際に並列に動作させることは不可能であるし、また、context の切りかえに伴うオーバーヘッドが多大なものになってしまう危険性がある。一方、既存のOSにもあるプロセスやタスクといった疑似的な並列性を表現するわく組は、ある程度以上の大きさのシステムを構築する上で非常に有効である。そこでSIMPOSでは、process という特別なクラスを導入し、クラスprocess のオブジェクトのみが独立のcontext を持ち得ることにした。従って、利用者があるプログラムをプロセスとして動作させたいときには、クラスprocess のオブジェクトを陽に生成し、生成されたプロセス・オブジェクトにプログラムを渡すという方式をとれば良い。こうすることで、普通のオブジェクトと並行して動作するオブジェクト（プロセス・オブジェクト）とを利用者が陽に使いわけられるようにした。

２．２　プロセス管理

クラスprocess は、context を物理的に切りかえるという下位のレベルの機能を使って、マルチプロセスの環境を利用者に提供することを行なう。クラスprocess は、プロセスの生成・起動・中断・再開・終結・開放を実現する述語を提供する。また、スケジューラの部分はすべてのプロセス・オブジェクトで共有するので、クラスprocess のクラススロットで保持している。

プロセスは、新たに生成されるときに、プロセスとして実行すべきプログラムを渡されるが、このプログラムは、クラスas_programを継承したクラスのインスタンス・オブジェクトである。

図１は、クラスprocess のクラス構成を示している。ハードウェアで認識するプロセス制御ブロックを保持するcontext と、そのプロセスの保持する資源をチェインするresourceは、プロセスのインスタンススロットで保持される。

３．プロセス間同期・通信機構

３．１　ストリーム

オブジェクト間のインタラクションは、一方が他方のオブジェクトに定義された述語を呼び出すことで実現される。プロセス・オブジェクトを除くすべてのオブジェクト間のインタラクションは、逐次的に実行されるので問題ない。しかし、プロセス・オブジェクト同士のインタラクションでは、プロセス間の同期・通信を矛盾なく実現するために、何らかの同期メカニズムを提供する必要がある。

SIMPOSでは、プロセス間の唯一の同期・通信機構として、クラスstreamを提供している。ストリームはオブジェクトを流すパイプである。ストリームの一方の端にオブジェクトを入れ、他端からオブジェクトを取り出すことが出来るという一種のFIFOキューである。もしあるプロセスが、オブジェクトを取り出そうとしたときに、ストリームが空であると、他のプロセスがそのストリームにオブジェクトを入れるまで待たされる。この機能に

より、プロセス間の同期が実現される。このようにプロセス・オブジェクト間の同期・通信には必ずストリームを介することが必要である。 SIMPOSでは、クラスstreamを基本クラスとして提供し、その上に種々の上位機能をもつプロセス間同期・通信機構を構築している。

3．2　その他の同期・通信機構

ストリーム・サブシステムは、機能的に次の3つのレベルに分かれる。

ストリーム：同一プロセッサ上のプロセス間の単方向通信をつかさどる同期機構を有したパイプ
チャネル　：特にメッセージを送受信するためのストリーム
ポート　　：チャネルを、双方向通信可能になるよう拡張したもの

これらのクラスは、継承によって階層化されている。ストリーム・サブシステムでは、これらのクラスに加えて、さらにこれらに種々の機能（優先度制御付き・End_of_stream のあるもの・有限個バッファで構築されるもの・オブジェクトを取りださずに見ることができるもの・複数のストリームを同時に待てるもの）を加えたストリーム類も提供している。それらの付加機能は付加機能単位に1つのクラスとして利用者に提供される。従って例えば、利用者が優先度制御付きで、かつ有限長のバッファで構築されているポートを欲するときは、クラスport、クラスas_priority_stream、クラスwith_bounded_buffer を多重継承するだけで、必要十分な新たなクラスを作成することが出来る。

4．　おわりに

SIMPOSにおけるマルチプロセスの実現機構とプロセス間同期・通信機構について述べた。現在これらのサブシステムはPSI 上で稼働中である。

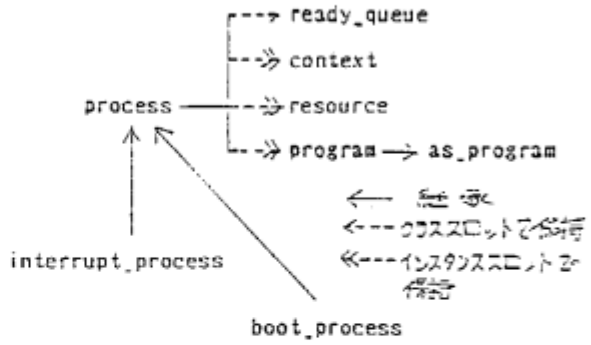


図1　プロセスのクラス構成図

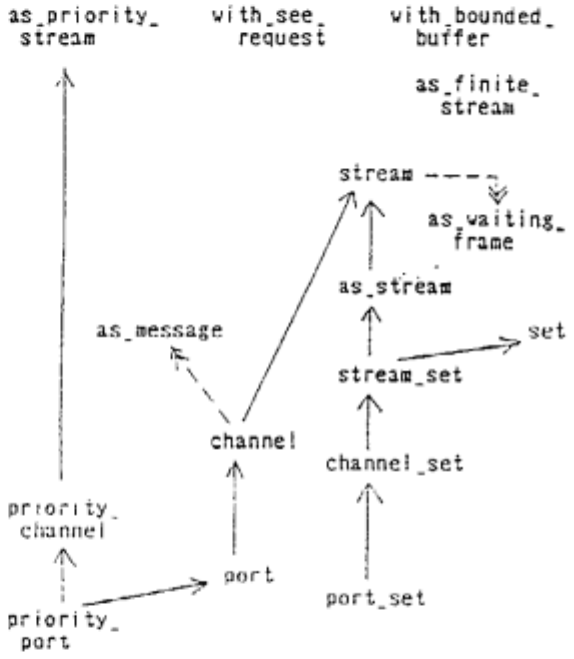


図2　ストリームのクラス構成図

```
class priority_port_with_bounded_buffer
 nature
    as_priority_stream,
    with_bounded_buffer,
    port;
end.
```

図3　付加機能を持ったポートのクラス定義例

SIMPOSの実行管理
ープール

斎藤信一　　渡辺久美　　島津秀雄　　吉田紀彦　　服部隆
(ビーエンシステム㈱)(沖電気工業㈱)(日本電気㈱)(㈱三菱総合研究所)((財)ICOT)

1. はじめに

SIMPOSはオブジェクト指向的に構築されているため、実行主体となるプロセスが複数のオブジェクトを集合として保持する場合が多い。この時、必要なデータ構造を個別に用意したのでは大変だし、拡張性に乏しい。そこで、プール管理はSIMPOSの中で汎用的に使用されるオブジェクトの格納機能を抽象化し、これを継承関係を用いて様々なバリエーションを実現した。これにより、拡張が容易で統一されたアクセス法を持つデータ構造が実現された。

以下では、プールの設計思想と実現法について説明する。

2. プール

プールはオブジェクトを格納する容器である。プールに対しては以下のような操作を行なうことができる。

- プールにオブジェクトを格納する。
- プールからオブジェクトを取り出す。
- プール中のオブジェクトを削除する。
- プール中のオブジェクトの格納情報に関する問い合わせを行なう。

プール管理は、プールの格納機能をクラスとして抽象化し、これを継承させることにより様々な格納機能を持つプールを提供している。このようなプールの格納機能の抽象化は以下の視点で考えた。

- 格納されるオブジェクト間の関係として順序があるかどうか。
- オブジェクトの格納位置をどのような形式で指定するか。
- 同じオブジェクトの重複を許すかどうか。
- どのようなアクセス法が可能か。

(1) 順序

プールは、格納されるオブジェクト間の関係から、順序のあるプールと順序の無いプールに分類される。

順序のあるプールはオブジェクトを順序づけて格納するプールであり、格納されたオブジェクトはその格納位置によって識別される。順序のあるプールの基本となるものはリストとアレイである(図2)。

i. リスト
　要素数が可変のプールであり、いくつでもオブジェクトを格納することができる。

ii. アレイ
　要素数が固定のプールであり、格納できるオブジェクトの数は有限である。

両者は一次元の順序を持つプールであるが、これらを多層に構成することによって多次元の順序を持つプール(バケット)を作ることもできる。例えば、アレイを多層に組み合わせることによりアレイ・バケットができる(図3)。

(2) 格納位置

順序のあるプールにおける格納位置は、0から始まる昇順の整数で表わされる。この格納位置は固定的に決まっている場合(アレイ)と、相対的に決まる場合(リスト)とがある。

また、バケットにおける格納位置はパス・ポジションによって識別される。パス・ポジションは検索される順路に沿って複数の位置の並びを表現するものであり、次のような形式で表わす。

[position1,position2, … ,positionN]

(3) 重複

プールは、格納されるオブジェクトの重複を許すプールと重複を許さないプールとがある。順序の無いプールについては、重複を許すプールとしてバグ、重複を許さないプールとしてセットがある。セットは、オブジェクトを格納する時に重複するオブジェクトを排除することによって実現される。

3. アクセス方式

プールに対する操作を行なうためのアクセス方式としては次の3つがある。

- 直接アクセス
- 順次アクセス
- キーによるアクセス

それぞれのアクセス法においては、その操作のインタフェースが同じになるように設計した。これによって、プールの内部構造を意識せずにアクセスすることが可能となっている。

（1）直接アクセス

直接アクセスは格納位置を指定してアクセスを行なう方式であり、順序のあるプールの機能として実現される．アレイの場合は文字どおりそれぞれの位置へ直接アクセスできるが、リストの場合は内部的に順次検索が必要である．

（2）順次アクセス

順次アクセスはタップという考え方を導入することによって実現している．タップとはプールの蛇口という概念を持つものであり、すべてのプールに対して設定可能である（図4）．

タップを用いることにより、プールの内部構造やオブジェクトの格納位置を意識せずに順次アクセスを行なうことが可能である．すなわち、リストでもアレイでも同様の操作によってタップが設定され、同様の操作でオブジェクトを取り出すことができる．

（3）キーによるアクセス

キーによるアクセスはインデックスを用いることで可能となる．インデックスはオブジェクトにキーを対応させて格納するプールであり、そのデータ構造や操作の方法は順序のあるプールと類似したものとして実現されている．

キーによるアクセスを行なう場合、内部での検索方法として、

・順次検索

・キーをハッシングする検索

があるが、両者の操作方法は同じものとして設計されている．

4．　おわりに

以上述べたように、プールは様々な格納機能を抽象化することによって、インタフェースの共通化や系統的なバリエーションの展開が達成された．

しかし、抽象化と処理効率の間には背反的な事情がある．

・抽象化を重視すると処理効率が低下する．特にプールのようなOSの基本部分では無視できない点である．

・処理効率を重視すると拡張性が低下する．

現在、プールはこのような問題を含んでおり、今後改善の余地があると思われる．

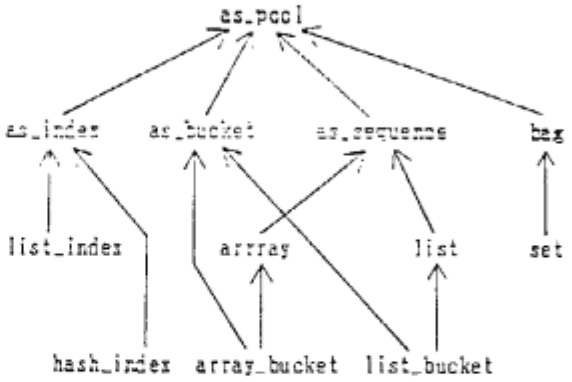


図1　プールのクラス継承図

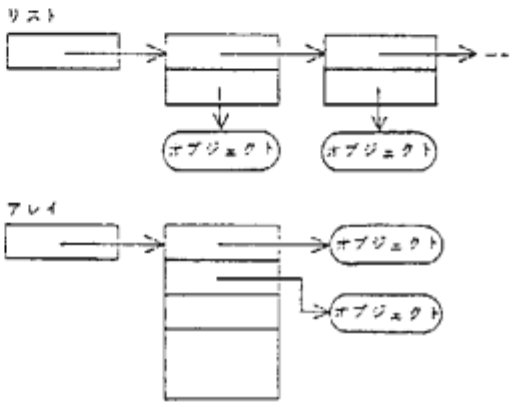


図2　リストとアレイ

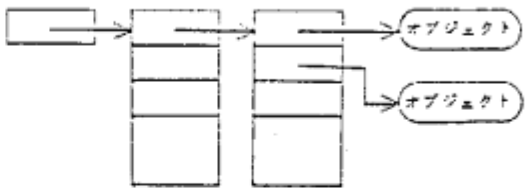


図3　アレイ・バケット

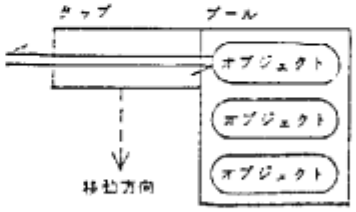


図4　タップ

ＳＩＭＰＯＳの実行管理 — ワールド

4E-5

後迫　久晃
（沖電気工業（株））
島津　秀雄
（日本電気（株））
吉田　紀彦
（（株）三菱総合研究所）
斉藤　信一
（ビーコンシステム（株））
関部　隆
（（財）ＩＣＯＴ）

1．はじめに

ＳＩＭＰＯＳは、超次型推論マシン（ＰＳＩ）のために開発されているプログラミング／オペレーティング・システムである。

ワールド管理は，ＳＩＭＰＯＳの実行管理プログラムの１つであり、プロセスに実行環境を与えることを目的としている。

本稿では、その概念及び基本構成・機能を述べる。

2．ワールド管理の概念

ワールド管理では、主記憶上のオブジェクト（資源）を名前で管理して、ユーザがプロセス実行中の任意の時点でオブジェクトにアクセスできる環境（実行環境）を提供する。オブジェクトを管理するにあたっては、ディレクトリ方式を採用している。ディレクトリは、名前でオブジェクトを登録するテーブルであり、階層構造をとることができる。

システムには、システムに唯一のルートのディレクトリ（システム・ルート・ディレクトリ）を起点とするディレクトリの階層木から成るシステム・ディレクトリ・トリーがある。これは、ディスク上のオブジェクトへアクセスする際にインタフェースとなるディレクトリ（パーマネント・ディレクトリ）を含んでいる。システム・ディレクトリ・トリーは、各プロセスから共通にアクセスできる。

各プロセスは、システム・ディレクトリ・トリーの一部や、プロセス上でテンポラリに作られたディレクトリをリストで連結したものを実行環境として持つ。これは、ワールドと呼ばれる。

次にディレクトリ、パーマネント・ディレクトリ、ワールドのそれぞれについて基本構成・機能を述べる。

3．ディレクトリ

名前でオブジェクトを登録する管理テーブルである。ディレクトリは、次のものから構成されている。

(1)　一連のディレクトリ・エントリ

(2)　ディレクトリに対する付加情報

(1)の各々は、ディレクトリに登録されるオブジェクトの名前とオブジェクトから成る。

(2)は、ディレクトリの新規作成・参照・変更の日時、ディレクトリへアクセスするためのパス名，ディレクトリへのアクセス許可情報から成る。

ディレクトリは、システム・ルート・ディレクトリを起点に階層構造をとることができ、ディレクトリの階層構造上の位置付けは、(4)のパス名によって知ることができる。

ディレクトリの提供する基本機能は、主記憶上のオブジェクトの登録、参照、削除等である。

パス名は、階層構造下にあるディレクトリやパーマネント・ディレクトリを辿ってオブジェクトにアクセスする場合、あるいは現在実行中のプロセスのもつワールドの中の部分ディレクトリを辿って、オブジェクトにアクセスする場合に利用され、名前と"）"記号を組合せたストリングによって表現される。例えば、図１のような階層構造下にあるオブジェクトｄにアクセスするには、パス名"ａ1）ｂ1）ｃ1"が必要になる。

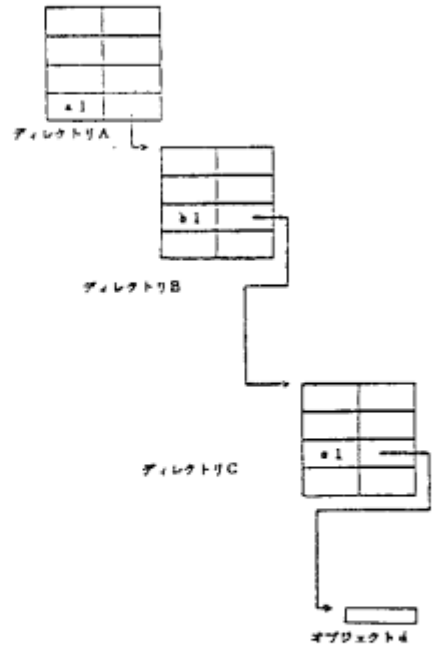


図1　ディレクトリの階層

4．パーマネント・ディレクトリ

パーマネント・ディレクトリは、ディレクトリ・ファイルを保持しているディレクトリである。ディレクトリ・ファイルは、名前でディスク上のオブジェクトを管理している管理テーブルである。パーマネント・ディレクトリは、主記憶上でディレクトリの下に作られ、主記憶上のオブジェクトをディスクへ格納したり、逆にディスクから主記憶上へ取り出す場合のインターフェースとして使われる（図2）。このとき、パーマネント・ディレクトリが保持するディレクトリ・ファイルが、キャッシュの役割を果す。

一度パーマネント・ディレクトリを作れば、内部的には、それに対応するディレクトリ・ファイルがディスク上に1つ作られ、そのディレクトリ・ファイルがパーマネント・ディレクトリにセットされるので、ユーザは、直接ディレクトリ・ファイルへアクセスする必要がなくなる。ユーザが主記憶上のオブジェクトをディスクへ格納したり、ディスクから取り出したい時は、パーマネント・ディレクトリに対して命令を出せばよい。

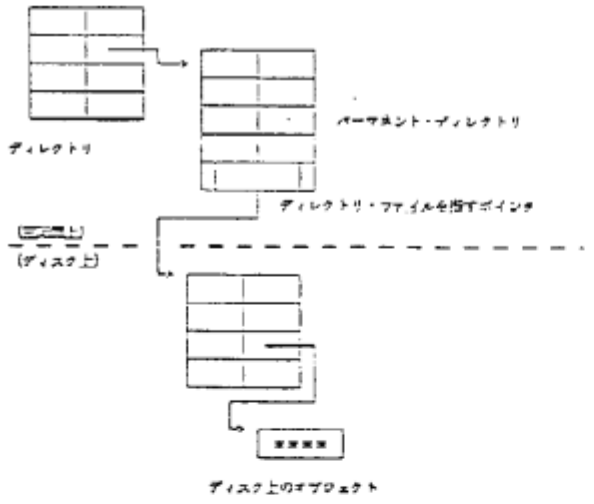


図2　パーマネント・ディレクトリ

5．ワールド

ワールドは、一連のディレクトリがリストで連結された構成になっている。ワールドは、それぞれのプロセスに対して設定できるようになっている。各々のディレクトリは、システムに唯一のルート・ディレクトリを起点に階層構造をなしているディレクトリの部分でも、また、プロセス実行中にユーザがテンポラリに定義したディレクトリのいずれであってもよい（図3）。

基本機能は、次に示す通りである。

(1)　ワールドにディレクトリを付加する。

(2)　ワールドからディレクトリを取り除く。

(3)　ディレクトリの機能のサポートにより、ワールドにおける主記憶上のオブジェクトの検索・付加を行なう。

ユーザは、プロセス実行中に、作ったオブジェクトの格納・取り出しなどができる環境が欲しい時には、ワールドを作ればよい。システム・ルート・ディレクトリを起点とした階層構造下にあるディレクトリを実行プロセス中のワールドにセットして、そこにオブジェクトを登録すれば、プロセス実行中の必要な時点でそのオブジェクトへアクセスできる。再度、ルート・ディレクトリから辿ってオブジェクトへアクセスする必要はなくなる。

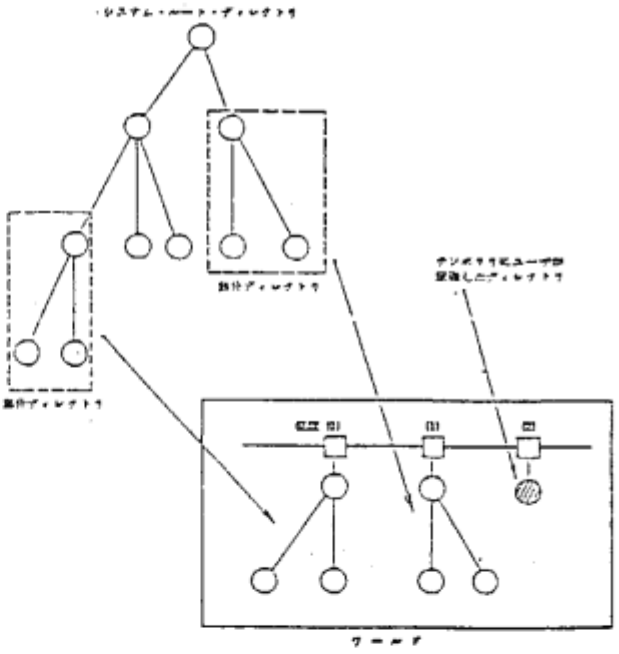


図3　ワールド

6．おわりに

以上のような基本構成と機能をもつ本システムのうち、現在、ディレクトリとワールドは、ＰＳＩを使って、実機デバッグ中である。今後は、ファイル・サブシステムを取り込んでのパーマネント・ディレクトリの実機デバッグ、ディレクトリに対するアクセス許可メカニズムの設計の仕様検討などが、重要な課題である。

ＳＩＭＰＯＳのウィンドウ・サブシステム

飯間　豊　　中沢　修　　板本　章二　　辻　康一郎
(沖電気工業(株))　(松下電器(株))　((財)ＩＣＯＴ)

1．はじめに

ＳＩＭＰＯＳは，ＰＳＩ(逐次型推論マシン)上に開発中のプログラミング／オペレーティングシステムである。

ウインドウ・サブシステムは，ＳＩＭＰＯＳの中核部であるカーネル／スーパバイザと，上位システムであるプログラミングシステムとの間に位置し，ビットマップディスプレイ，キーボード及びマウスを対象とする入出力管理プログラムである。これらの入出力装置に対する複数プロセスからの同時利用を可能とすることにより，ＰＳＩ上のプログラミング環境を向上させることを目的としている。

本稿ではウインドウ・サブシステムの概要及び特徴について述べる。

2．概要

ウインドウとは，ビットマップディスプレイ，キーボード及びマウス上に作られた論理的端末装置である。ウインドウ・サブシステムはユーザプロセスからの要求に従って，互いに異なった特性を持つ複数のウインドウを作成する。各プロセスは物理的装置を意識することなく入出力を実行出来る。

ユーザ側から見ると，ウインドウはある特定のプロセスに接続されている端末である。ディスプレイスクリーン上に表示される複数のウインドウを通じて，ユーザは同時に複数のプロセスと対話出来る。

ウインドウはビットマップディスプレイ上に長方形領域として表示される。ウインドウのサイズ，スクリーン上の位置は自由に設定可能である。同時に多数のウインドウが存在する場合には，ウインドウ群は，机上の紙の様に互いに重なり合っているかの様に表示される。

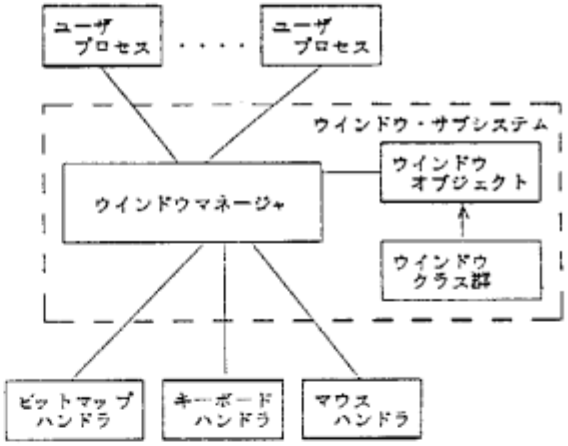
本システムの機能を以下に示す。

- ・オーバーラップマルチウインドウシステム
- ・ウインドウのサブウインドウ分割可能
- ・マウスによるポジショニング
- ・文字及びビットグラフィック出力
- ・ウインドウボーダ／ラベル等の表示
- ・メニュー等の多様なウインドウのサポート
- ・ウインドウのオーダメイドが容易

3．構成

本システムの構成を図．1に示す。ウインドウ・マネージャは，ウインドウ・マネージャプロセス下でバックグラウンド的に実行されるプログラムであり，主にウインドウ・サブシステムとユーザプロセスあるいはデバイスハンドラプロセスとの間のプロセス間通信を司る。ウインドウの実体は，ウインドウクラスのインスタンスであるウインドウオブジェクトである。ウインドウに対する入出力等の要求は，ウインドウ・オブジェクトによって処理される。

ウインドウクラス群は，図形出力や文字入力といった単一機能について記述されている多数の部品クラスと，それらのうちのいくつかを選択的に継承することによりウインドウとしてのまとまった機能を実現している完成品クラスとから構成されている。ウインドウクラスの追加により，容易にユーザ定義ウインドウの作成が可能である。

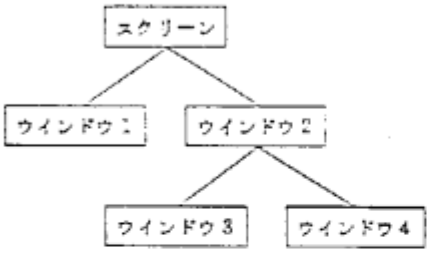


図．1　システム構成

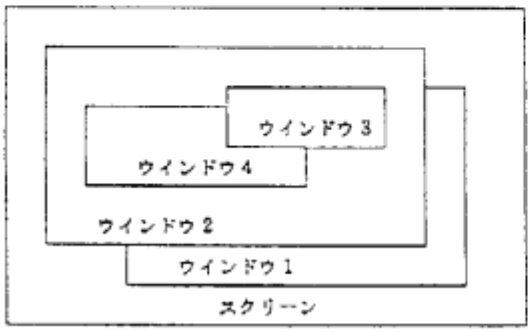
4．ウインドウの階層化

高度なディスプレイ制御を行なうには，単にディスプレイスクリーンのマルチウインドウ化のみでは不十分であり，ウインドウ個々がさらに再帰的にマルチウインドウ化出来ることが必要である。本システムでは，ウインドウに階層性を持たせることによって，再帰的マルチウインドウを容易に実現している。

図．2に示す様に，階層のルートには，スクリーンと呼ばれるディスプレイ画面全体を表すシステムウインドウがある。ユーザ・プロセスからの要求によって作られるユーザウインドウ（ウインドウ１及び２）はすべてスクリーンの子ウインドウとなる。また，ユーザウインドウに子ウインドウ（ウインドウ３及び４）を作ることにより，ユーザウインドウがさらにマルチウインドウ化される。図．2の階層構成に対応する画面表示の一例を図．3に示す。



図．2　ウインドウ階層



図．3　表示画面

本システムには，集中的管理プログラムは存在しておらず，ウインドウの管理はすべてウインドウ階層内で行なわれる。即ち，ウインドウは親ウインドウによって管理される。また，親ウインドウがさらにその親ウインドウによって管理されているため，間接的にすべての祖先ウインドウによって管理される。

5．出力

オーバラップマルチウインドウシステムにおいては，他のウインドウによって隠されているために，ユーザに見えないウインドウが存在する。ウインドウに文字等を出力する際には，それがユーザに見えるものであるか否かを考慮した処理が必要となる。

最も単純な出力方式は，完全に見えているウインドウにのみ出力を許し，隠されているウインドウへの出力を要求したプロセスは，そのウインドウが見える状態となるまで待たされるというものである。しかしこの方式では，バックグラウンド的に実行されているプロセスからのメッセージをユーザに即座に伝えることが出来ないという欠点がある。他にも多くの方式が考えられるが，ユーザプロセスの多様性を考慮すると，いずれにも何らかの欠点があり，ただ1つの方式を導入することは困難である。

そこで本システムでは，以下に示す4種類の出力方式をサポートし，いずれの方式を用いるかはウインドウ個々に設定することとした。

- ・WAIT　　当該ウインドウが見える状態となるまで，ユーザプロセスを待たせる
- ・NOTIFY　警告メッセージを表示する
- ・OUTPUT　隠されたまま，出力を行なう
- ・SHOW　　当該ウインドウを強制的に見える状態とする

6．入力

ウインドウは入力デバイスとして，キーボードとマウスを持つ。これら2つのデバイスは，かなり異なった性質を有しており，それぞれ別個の方式で制御されている。

文字入力デバイスであるキーボードは，複数プロセスからの同時共有が本質的に不可能なデバイスであり，キーボード入力の送り先ウインドウ（セレクテッドウインドウ）は，一意的に決定されている必要がある。このため，セレクテッドウインドウは，主にユーザの指定により決るものとする。

一方ポジショニングデバイスであるマウスは，ディスプレイスクリーン上を対象とするデバイスであり，スクリーンに表示されている複数のウインドウからの共有が可能である。マウス入力の送り先ウインドウは，その時点でマウスがどのウインドウ上に在るか，即ちマウス位置を示すマウスマーカがどのウインドウ上に在るかによって決定される。

7．メニューウインドウ

マウスはメニュー選択のために最も頻繁に使われると考えられる。そこで，本システムではメニュー選択機能を備えた幾々のウインドウ（メニューウインドウ）を提供している。メニューウインドウは，項目の表示及びマウスによってそれらを選択する機能のみを持つウインドウである。項目を選択することにより，ユーザによって定義されたコマンドがユーザプロセスに送り返される。また，項目の選択と同時にウインドウイメージがスクリーン上から自動的に消えるメニューウインドウも作成出来る。

8．おわりに

ウインドウ・サブシステム単体における基本動作については，ＰＳＩ上において検証済である。さらに，エディタ等の上位サブシステムと接続して，総合的検証を進めて行く。

本システムの様に，マンマシンインタフェースに係わるソフトウェアにおいては，実際的環境下での使用経験に基づく継続的な改良が必要である。今後，他サブシステムとのインタフェースをも含めて，処理効率の向上，機能拡充について検討を加える予定である。

ＳＩＭＰＯＳのネットワーク・サブシステム

4E-7

真山　幸秀　　服部　隆
（　沖電気工業（株））　　（（財）ＩＣＯＴ　）

1．はじめに

パーソナル逐次型推論マシン（ＰＳＩ）は第５世代コンピュータシステム開発の一環として作られているソフトウェア開発用のワークステーションある。

ＰＳＩにはネットワーク機能があり，複数のＰＳＩや関係データベースマシンＤＥＬＴＡと繋ぐことにより，ユーザに対してスタンドアロンのマシンでは得られない幅広いプログラミング環境を提供する．すなわち，他の人が作ったプログラムを共用したり，メールシステムを利用したり，あるいは，ＤＥＬＴＡに格納されている膨大なデータベースを利用したりできる。ネットワーク機能は，ウィンドウ，マウスなどの高度なマンマシンインタフェース，ワークステーションならではの一定したレスポンスタイムとともに，ＰＳＩを高性能で使いやすいｐｒｏｌｏｇ処理系にするための機能として重要である。

本論文では，ネットワーク機能をサポートするために開発されたＳＩＭＰＯＳネットワーク・サブシステムの概要を報告する。

2．ハードウェア構成

ＳＩＭＰＯＳネットワーク・サブシステムは，ＬＩＡ（LAN Interface Adapter），ＬＩＢ（LAN Interface Board）という２つのデバイスを使ってネットワーク機能を実現する。

ＬＩＡは，ＬＡＮ用の高性能デバイスであり，コマンドデータを送ることにより仮想回線の形成および解除，更に形成された仮想回線を使ったデータの送受信を行なうことができる。

このように，従来のＯＳに於いて通信機能の下位レイヤとして記述されていたソフトウェアがＬＩＡに吸収されているので，ＯＳで記述しなければならない部分が軽くなり，見通し良く設計できるという利点がある。半面，デバイスの機能が高めに固定されているので，ＯＳをトップダウンに設計してゆく際の自由度が制限されてしまう難点がある。

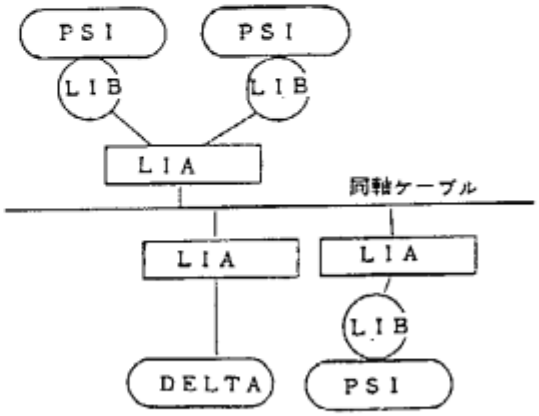
ＬＩＢは，ＰＳＩとＬＩＡとの間のインターフェースボードであり，ＰＳＩ上のデータをＬＩＡに送ったり，逆にＬＩＡのデータをＰＳＩ上に書き込んだりする。

ネットワークの物理的構成を図－１に示す。

3．ネットワーク・サブシステムの機能と特徴

ネットワークＯＳとしてＳＩＭＰＯＳを見たとき，その機能は大きく３つのレベルに分れる。

1）マシン間通信モード



図－１　ネットワークの物理構成

ＬＩＡの機能を直接的に利用して，通信網を形成，消滅させたり，データを送受信したりする機能を提供する。これらの操作は，プラグと呼ばれるオブジェクトを通じて行えるようになっている。

また，通信網は，全二重のセッションによるメッシュ方式を想定し，仮想回線はケーブルと言うオブジェクトで表される。ケーブルは，ソケットと呼ばれる，位置を示すためのオブジェクトの間に張られる。プラグは，いずれかのソケットに"差し込まれて"おり，そのソケットから他のソケットにケーブルが張られると，そのケーブルに対してポインタを張る。ケーブルは，ひとつのソケットから複数本張ることができ，ユーザは一つのプラグからこれらを操作することができる。また，ＰＳＩやＤＥＬＴＡなどのマシンは，ノードと呼ばれるオブジェクトで表現され，マシンアドレスなどの情報が管理される。

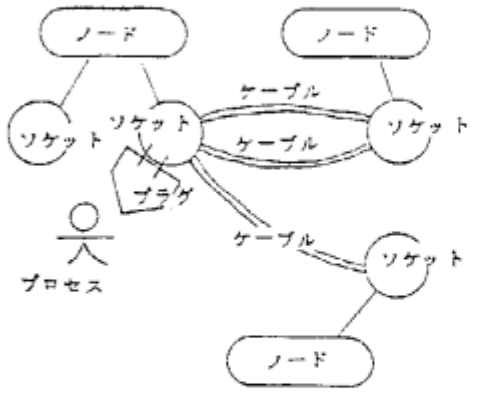
ノードとソケットは，ディレクトリに管理されており，ユーザは必要に応じてパスネームを指定してそれらのオブジェクトポインタを手にいれることができる。ノード，ソケット，ケーブル，プラグの関係を図－２に示す。

2）プロセス間通信モード
［ＰＳＩ－ＰＳＩ通信のみ］

単一マシン内のプロセス間通信をネットワーク規模にまで拡張する機能を提供する。

ＳＩＭＰＯＳにおけるプロセス間通信は，スーパバイザで提供されているポート／チャネル方式である．このモードでは，他マシン内のポート／チャネルを自マシン内のそれと同様にアクセスできる。

3）リモートオブジェクト操作モード
［ＰＳＩ－ＰＳＩ通信のみ］



図－2　ノード，ソケット，ケーブル，プラグの関係

他マシン内のオブジェクトを，自マシンのそれと同様に操作できる機能を提供する。プロセス間通信モードとこのモードによって，ＯＳがユーザに提供できるプログラミング環境が単一のＰＳＩに留まらず，複数ＰＳＩにまで自然に拡張される。

ただし現在想定しているものは，2）のためのポート／チャネル，ディレクトリ，ファイルのリモート操作である。

4．インプリメントの概要

1）マシン間通信モード

このモードでは，ユーザプロセス，ネットワークマネージャプロセス，ＬＩＢハンドラプロセスの３つのプロセスが互いにメッセージ通信をして操作を実現する。プラグは，ポートとしての機能を持っていて，ネットワークマネージャプロセスとのメッセージ通信に使われる。

ユーザプロセスが，プラグに操作をすると，その内容がプラグメッセージとして，ネットワークマネージャに送られる。ネットワークマネージャは，そのメッセージを解釈してＬＩＡ用のコマンドデータのオブジェクトを作り，ＬＩＢメッセージの形でＬＩＢハンドラに送る。ＬＩＢハンドラはそのコマンドデータのオブジェクトをコードに変換してＬＩＢに読み込ませる。

逆に，ＬＩＢからＰＳＩに送られてきたコマンドデータは，ＬＩＢハンドラによってコマンドデータのオブジェクトに変換され，ＬＩＢメッセージの形でネットワークマネージャに送る。ネットワークマネージャは，コマンドデータオブジェクトからプラグメッセージを作り，プラグに返す。

ネットワークマネージャは，プラグメッセージとＬＩＡのコマンドデータとの間の変換，および，プラグメッセージ，ＬＩＢメッセージの通信制御を行なう。

2）リモートオブジェクト操作およびプロセス間通信モード

他マシン内のオブジェクト（実オブジェクト）を操作する場合，そのオブジェクトの"代理オブジェクト"（リモートオブジェクト）を自マシン内に生成する。同時に，他マシン内には"代理プロセス"が生成される。リモートオブジェクトは，実オブジェクトと同じユーザインタフェースをもっており，ユーザはそれが"代理オブジェクト"であることを意識せずに操作できる。

ユーザがリモートオブジェクトを操作すると，その内容がリモートオブジェクトメッセージの形で相手マシンの"代理プロセス"に送られる。"代理プロセス"は，そのメッセージに従って実オブジェクトに操作を実現する。

リモートオブジェクトメッセージは，マシン間通信モードのプラグのデータ送受信機能を使って相手側マシンに送られる。

プロセス間通信モードのインプリメントは，相手マシン内のポートやチャネルの代理オブジェクトを自マシン内に生成することによって行なわれる。

ＳＩＭＰＯＳのリモートオブジェクトは，ＣＭＵで開発されたＡＣＣＥＮＴで提案されたリモートポートの考え方を拡張したものである。

5．おわりに

現在，マシン間通信モードの開発はほぼ終了している。プロセス間通信モード，リモートオブジェクト操作モードは，我々の提案した基本方式をもとにスーパバイザ開発グループなどにおいて検討中，ないし開発中である。

マシン間通信モードは，ＬＩＡの仕様に沿って開発されており，ＬＩＡにソフトの皮を被せたような構成になっている。それに対して，プロセス間通信モード，及びリモートオブジェクト操作モードは，デバイスの仕様からは独立しており，ＳＩＭＰＯＳの論理構成に調和した設計を模索・実験する，といった方針で開発されている。

リモートオブジェクト操作モードなどのインプリメントを，よりすっきりしたかたちで行なえるようにするためには，オブジェクト・ポインティグメカニズムや手続き呼出しの方法など，ＳＩＭＰＯＳ全体の論理構成や，システム記述言語についての検討を更に深めてゆく必要があると思われる。また，ＡＣＣＥＮＴ（前出）や，同じくネットワークをサポートし，オブジェクトオリエンテッド的な考え方で作られているＣＭＵのＨｙｄｒａでは，プロテクションメカニズムがかなり追求されている。ＳＩＭＰＯＳにおいてより強力なプロテクションメカニズムを考えた場合，ネットワークサブシステムをどう設計すればよいかという興味深い問題が起ってくるであろう。

参考文献

Richard F. Rashid. et al.　ACCENT: An communication oriented network operating system kernel. CMU (1981)

W. A. Wulf. et al. Hydra/C.mmp: An experimental computer system. McGraw-Hill. New York (1981)

4E-9

SIMPOSのファイル・サブシステム

小松光雄　　真野忠志　　小長谷明彦　　服部隆
(ビーコンシステム(株))(ビーコンシステム(株))(日本電気(株))((財)ICOT)

1. はじめに

SIMPOSのファイル・サブシステムはPSIにおけるディスクへのデータ格納機能とアクセス機能を提供する。その最大の特徴は、ディスク上のファイルをオブジェクトとしてとらえる点にある。これにより、ファイル・サブシステムをSIMPOSの対象指向の枠組の中に自然に組入れられる。本稿では、この考え方に基づく対象指向ファイル・システムの設計思想と実現法について述べる。

2. ファイル・オブジェクト

一般に、ファイル中に格納されるデータは半永久的な存在であり、プロセスとともに生成、または、消滅するメモリ中のデータとは本質的に性質を異にする。通常、ファイルは外界に存在する特別な記憶領域であり、このデータ格納領域とファイルへのアクセスの為の操作命令を分解してファイル・システムの構築を考えると、SIMPOS全体がオブジェクトを基本単位とした対象指向システムとして構成されることについての一貫性が損なわれる。

SIMPOSのファイル・サブシステムでは、このような問題点を解決する為に、ファイル・オブジェクトという考え方を導入している。ファイル・オブジェクトは、概念的には、格納領域としてのディスク・ファイルにファイル・アクセス機能を付加したものと考えられる。ファイルへの入出力は、このファイル・オブジェクトに備えられたアクセス機能を対象指向呼出しで起動することにより実現される。

また、ファイルをオブジェクトとしてとらえることの利点の1つとして、アクセス機能をクラスとして抽象化させることにより、クラスの継承機構を有効に利用して種々のファイル・オブジェクトを定義できることがあげられる(図1)。

このファイルのクラス化にあたって、格納できるデータに着目し、(1) ビット列、整数、または、文字列等のデータ、(2) オブジェクト、(3) 知識の3段階に分けて実現することにした。さらに、一般のファイルだけでなく、ディレクトリ・ファイルやファイルの物理的な格納情報を格納する場所(VTOC)もオブジェクトとして扱って、対象指向システムとしての統一性を計っている。

3. データ・ファイル

ファイル・サブシステムでは、最も基本的な格納機能として、データを格納するファイルを提供する。このデータ・ファイルに対しては、さらに、データの格納形式及びアクセス法の違いから細分化を行った。すなわち、データ格納形式の違いからは、ビット列の並びから構成されるバイナリ・ファイルと、いくつかの意味のあるデータの固まり(レコード)の並びから構成されるレコード・ファイルに分類される。このレコード・ファイルに対しては、可変長のレコードを扱うヒープ・ファイルと、固定長のレコードを扱うテーブル・ファイルに分類される。

次に、アクセス法の違いからは、前述の3つのファイルが直接アクセス法を持つのに対して、テーブル・ファイルについては、キー値によるインデックス・アクセス法(インデックス・ファイル)が考えられる。さらに、それぞれのファイルに対して順次アクセス法でデータの格納、または、取り出しを行なう機能としてファイル・タップが用意される。

4. インスタンス・ファイル

一般に、対象指向で構成されるシステムにおいて、メモリ中のオブジェクトをディスク・ファイルに格納できる機能を提供することは、対象指向システムにおける単一メモリの実現を可能とする。しかしながら、この機能の実現に当っては、メモリ上で複雑なネットワーク構造を取るオブジェクトを一次元的な領域(インスタンス・レコード)に変換する必要がある。この為の基本的な機能を体系化するには、まだ多くの検討課題を残している。たとえば、インスタンス・レコードを実際に作成する場合、各オブジェクト毎にインスタンス・レコードへの変換手続きを定義しておく必要があり、この変換機能をインスタンス・ファイルが標準的に用意することは、今の所、困難である。

本ファイル・サブシステムでは、レコード・ファイルに対応して、可変長のインスタンス・レコードを扱うインスタンス・ヒープ・ファイルと、固定長かつ同一フィールド様式のインスタンス・レコードを扱うインスタンス・テーブル・ファイル(同じクラスのオブジェクトの格納)の

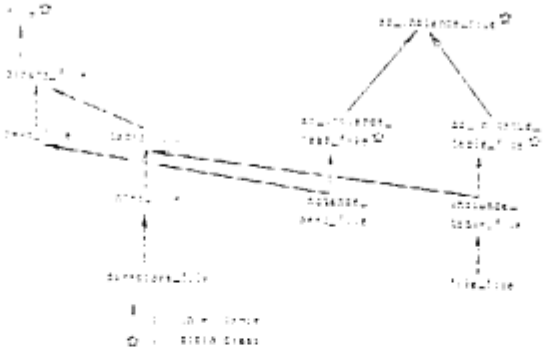
提供を考えた。この中で、インスタンス・テーブル・ファイルの実際のインプリメント例として、ファイル・オブジェクトを格納する為のファイル・ファイルを実現している。ファイル・ファイルは、通常のファイル・システムにおけるボリューム管理テーブル（VTOC）に相当するものであり、ファイル・オブジェクトのボリューム内での格納情報を管理している。また、ファイル・ファイルはマウントされるボリュームに対して唯一つのみ存在し、異なるクラスで定義されるファイル・オブジェクトを１つのインスタンス・テーブル・ファイルに収容する為に、フィールド情報の１つにファイル・クラスを識別する情報を持たせている。

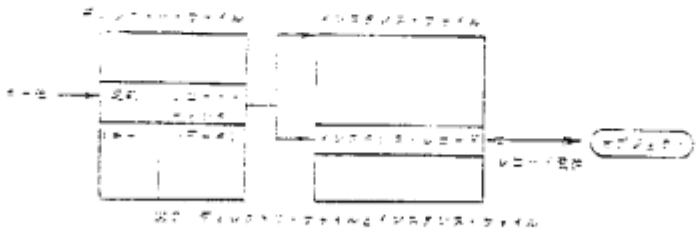
さらに、インスタンス・ファイルのインスタンス・レコードは、格納されるオブジェクトの名前と対応付けられる（図２）。これを実現するものがディレクトリ・ファイルであり、オブジェクトの名前をキー値として、レコード・ポインタをデータとして保持している。レコード・ポインタはメモリ上のオブジェクト・ポインタに相当する。このディレクトリ・ファイルは階層構造を形成していて、オブジェクトは順路名で検索される。また、この順路名は、ワールド・サブシステムの提供するディレクトリ・システムの中に統一されるものである。

５．　おわりに

SIMPOS設計の基本概念である対象指向に沿ったファイル・サブシステムの構成について述べた。本ファイル・サブシステムをクラス・システムで実際にインプリメントしてみて、新しいファイルの定義やファイル・オブジェクトへの操作としての新しい機能の追加が、クラスの継承機構を利用して比較的容易に行なえることが分った。しかし、反面、クラスの数が多くなることから発生する保守の問題や、継承したクラスのメソッドの中に再定義しなければならないものもあって、必ずしも単純に機能拡張が行なえない場合がある。

また、本ファイル・サブシステムは、格納機能とアクセス機能の中の基本的な機能についてインプリメントがなされているが、将来的には、データ・ベース・システムに統合される方向に向かいつつある。さらに、最終的には、格納データとして知識を格納する知識ベースの構築に対するサポートを目指すものであり、これについては、知識の表現方法等の未解決の問題をかかえて、知識ベース研究の大枠の中からのアプローチ課題が残されている。





ＳＩＭＰＯＳのＩＰＬ方式

上田　尚純　　東条　敏　　黒川　利明
（三菱電機（株））　（（株）三菱総合研究所）　（（財）ＩＣＯＴ）

１．はじめに

個人使用向けのProlog高速実行マシンである逐次型推論マシンψ（ＰＳＩ）上に、論理型プログラム開発支援用プログラミング／オペレーティング・システムＳＩＭＰＯＳを開発中である。ψはPrologインタープリタや一部ＯＳ機能をファームウェアで実現している事、ＳＩＭＰＯＳは対象指向の抽象化機構を持つ言語（ＥＳＰと称される）で記述されている事等の事情で、そのＩＰＬは汎用計算機での通常のＯＳのＩＰＬと一部異なる面がある。ＳＩＭＰＯＳのＩＰＬ時のシステム立上げ手順について、その特徴を中心に述べる。

２．ＳＩＭＰＯＳのＩＰＬの問題点

ＳＩＭＰＯＳのＩＰＬ手順を検討する際、次の点が問題となった。

（１）ψではメモリ管理、一部プロセス管理をファームウェアで行っており、ＩＰＬで最初に実行されるプログラム（ブータ）の実行環境もやはりファームウェアで設定せねばならない。

（２）ψの機械命令（ＫＬ０と称される）はテーブル形式でProlog を表現したものになっており、虫のパッチをワード単位でＩＰＬ時に施すのは困難である。

（３）ＳＩＭＰＯＳを構成するクラスは全てライブラリと称するＳＩＭＰＯＳ内の一部機能で管理されるが、ＩＰＬ中はこの機能がまだ作動していない。

これら問題点に対して、以下の方針でＩＰＬ手順を設計した。

（１）ＩＰＬ時、ブータが動作できるシングル・プロセス環境をファームウェアで設定する。マルチ・プロセス環境はブータが作動して作り上げる。

（２）虫のＩＰＬ時の修正はクラス単位の置換で行う。

（３）ブータ等のＩＰＬで使用するプログラムも対象指向言語で作成し、ＩＰＬ中はＩＰＬ専用の簡易方式でクラス管理を行う。ＩＰＬ終了後、これらクラスを正式にライブラリに登録する。

ＩＰＬで使用するクラスは、出来るだけＳＩＭＰＯＳの標準クラスとして作成したものを一部手直しして流用したが、ＳＩＭＰＯＳのシステム記述言語であるＥＳＰが提供するクラス継承機能等を利用して容易にこの作業を行えた。例えば、割込を使用して動作する入出力ハンドラを一部変更して、割込を使用しないポーリング版の入出力ハンドラを作成した。

３．ＩＰＬ手順

ＳＩＭＰＯＳのＩＰＬ手順を図１に示す。ψには低速デバイスの制御、本体部の診断等を行うマイクロ・プロセッサ内蔵のコンソール・プロセッサ（ＣＳＰ）が装着されており、ＩＰＬもＣＳＰにより開始される。ＣＳＰはハードウェアの初期設定やファームウェアのＷＣＳへのロード等を行う。次いでファームウェアの初期設定ルーチンを実行させてPrologインタープリタの動作環境の整備、シングル・プロセス環境の設定を行う。次にブータをディスクから読み込み主記憶にロードし、ブータを起動する。以後はソフトウェアの担当となり、ブータがＳＩＭＰＯＳのクラスをディスクから順次ロードしていく。

クラスはフロッピー・ディスク（ＦＤ）、または固定ディスク上の論理フロッピー・ディスク領域のファイル（ここでは以後クラス・ファイルと呼ぶ）で保持されている。１ファイルには１ケないし複数ケのクラスが含まれている。ＦＤはＩＢＭ互換形式であり、固定ディスク上の論理フロッピー・ディスク領域も論理的に同じ形式となっている。ＣＳＰおよびブータはＩＢＭ互換形式ＦＤを扱う機能を持っている。例えばブータはＦＤ上に“ＳＹＳＩＰＬ”というファイル名で登録されており、ＣＳＰはＦＤのディレクトリをサーチしてこのファイルを見付けてロードする。ブー

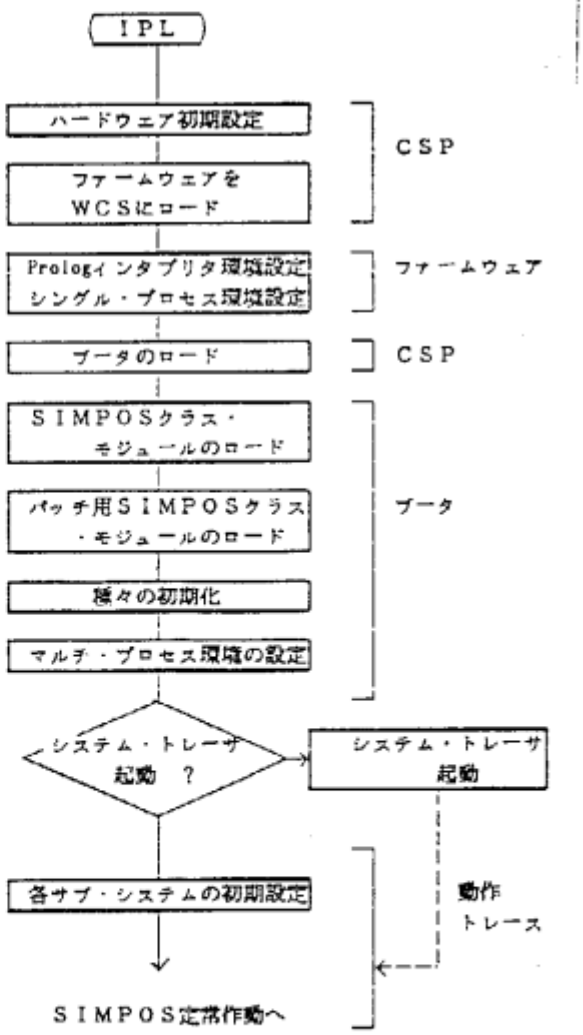


図１　ＳＩＭＰＯＳのＩＰＬ手順

タのクラス・ファイルのロード手順は次のとおりである。

(a) ロードすべきクラス・ファイル名を保持するファイル（"IPLLIST"）の読み込み。

(b) "IPLLIST"で指定されている各ファイルのロード。これによりカーネル、ユーバイザ、ウィンドウ・システム等のSIMTOS中核部を構成するクラスが主記憶にロードされる。

(c) パッチ用クラス・ファイル名を保持するファイル（"PATCHLST"）の読み込み。

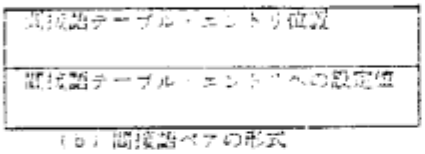
(d) "PATCHLST"で指定されている各パッチ用クラス・ファイルのロード。

パッチ用クラス・ファイルは別のFDからも与えられるようになっており、後述する機構により、通常のクラス・ファイルと同じ方法でロードすれば自動的に虫のある古いクラスと置換えられるようになっている。通常のクラス・ファイルとパッチ用クラス・ファイルの形式は全く同じである。クラスのロードが完了すると、マルチ・プロセス環境の設定等のシステムの初期化を行う。

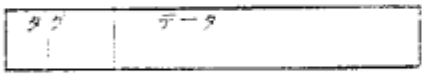
4．FD上のファイル形式と間接語テーブル

ブータによりロードされるクラス・ファイルのFD上のファイル形式を図2に示す。リロケータブル形式であり、ブータによりロード時にリロケートされる。コード本体部はKL0で表現されており、各ワードは8ビットのタグと32ビットのデータ部とより成る。タグの先頭2ビットは本来GC用であるが、IPL時はリロケーション用として使用している。間接語情報部は間接語テーブルにセットする値を指定するもので、対象となる間接語テーブル・エントリを指し示すワードとそのエントリに入れる値を保持すワードのペアが並んだものである。

(a) FD上のファイル形式



(b) 間接語ペアの形式



00XXXXXX : 絶対データ（リロケート不要）
10XXXXXX : 相対データ（リロケート要）
01XXXXXX : 間接語テーブル・エレメント位置

(c) コード部、間接語情報部のワード形式

図2．FD上のクラス・モジュールのファイル形式

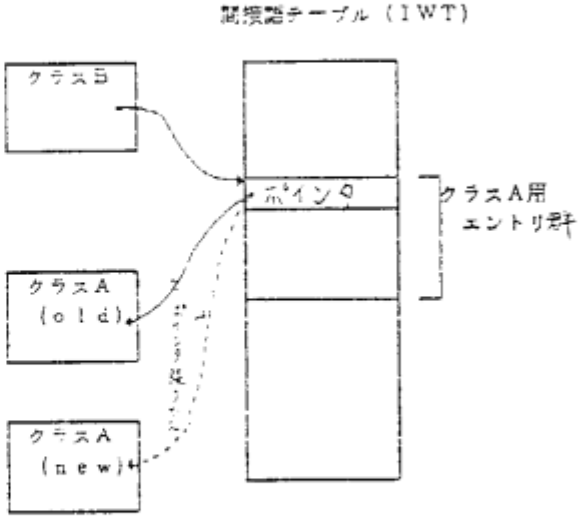


図3．間接語テーブルによるクラス間リンケージとそれによるパッチ方法

間接語テーブル（IWT）は主記憶上に置かれた、IPL中のクラス間の参照関係を管理するリンケージ用テーブルである。（図3） 各クラスが使用するIWTのエントリは、特別なデータベースで管理されており、一旦クラスにエントリ群が与えられると、そのクラスを再コンパイルしても同じエントリ群を使用する。あるクラス（A）が他のクラス（B）から参照される場合、他クラス（B）の参照ポインタは直接クラス（A）内の被参照箇所を指すのでなく、その被参照箇所に対して確保されているIWTエントリを指すようコンパイルされる。IPL時にクラス（A）をロードする際にIWTエントリに被参照箇所へのポインタがセットされるが、これを指示するのが間接語情報部の間接語ペアである。KL0ではこのような間接ポインタは特別なタグを持ち、インタプリタは自動的にポインタをたどってKL0を実行する。

クラス（A）に虫がいた場合、虫を修正して再コンパイルし、パッチ・モジュールとしてロードする。クラス（A）に対応するIWTエントリの内容が全て書き変えられ、新しくロードされたモジュールが古いモジュールに取って換わることになる。なお、古いモジュールが占めている領域はいずれGCで回収される。

5．おわりに

PSIはハードウェア、ソフトウェア共に高機能のため、IPL時の負荷もその分大きくなっているが、以上述べた方法により効率よいIPLを実現した。

（参考文献）

SIMPOSの概要　森末他　Logic Programming Conf. 1984.3

PSIのHW設計　渡他　同上

ＳＩＭＰＯＳのシステム・トレーサ

佐藤裕幸，渡辺久晃，堀敦史，上田尚純，近山隆
(三菱電機(株))　(沖電気工業(株))　((株)三菱総合研究所)　(三菱電機(株))　((財)ＩＣＯＴ)

１．はじめに

我々は現在Prolog高速実行マシンである逐次型推論マシンφ（PSI）のオペレーティングシステム SIMPOSを開発中である。φはPrologを表形式で表現した高いレベルの機械命令体系（KL0と称される）を持つ。SIMPOSは、Prologにオブジェクト操作機能を組込んだESPと称するシステム記述言語で統一して記述されている。このSIMPOSの開発支援システムとしては、 CSP（コンソール・プロセッサ）等があるが、これらはハードウェア・レベルのデバッグを目的として作られたものであり、symbolicなデバッグができず、ソフトウェア・レベル（Prolog）のデバッグには適していない。 そのため、 ソフトウェアで作られているESPプログラムの実行機構で、メソッド呼出し（対象指向のメッセージ・パッシング）単位でのトレース機能が提供されている。しかし、マクロなメソッド・レベルのデバッグしか行えず、それより細かいPrologレベルのデバッグは行えない。 そこで、KL0レベルの細かいトレースもでき、symbolicなデバッグが行えるシステム・トレーサを開発した。以下では、システム・トレーサのトレース機能とその実現法について、また、メモリのダンプ等トレース以外の機能についても報告する。

２．トレース機能

システム・トレーサのKL0トレースは、ヘッド・ユニフィケーションを基本としている。従って、DEC－10Prolog等のゴール呼出しを基本としているボックスモデルのトレースとは異なっている。これはファームウェアのトレース・トラップ機能を利用しているからである。このトレース・トラップ機能には、ユーザ定義述語のヘッド・ユニフィケーション終了時と組込述語終了時にトラップ（プログラムの動きに同期して起こる割込の一種）を発生するKL0 step trap と ある指定されたクローズのヘッド・ユニフィケーション終了時にトラップを発生するKL0 clause trap の二種類がある。これらを利用してシステム・トレーサではトレースを行うタイミングによって以下の５種類のモードを用意している。

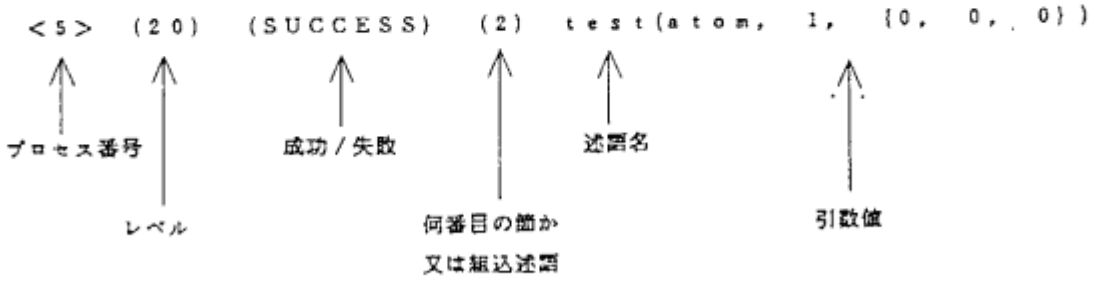
■step
ユーザ定義述語のヘッド・ユニフィケーション終了時と組込述語終了時にトレースする。

■clause
ユーザ定義述語のヘッド・ユニフィケーション終了時のみトレースする。。

■procedure
last clause以外のユニフィケーション成功時とlast clauseのユニフィケーション終了時にトレースする。

■spy
スパイポイントでのみトレースする。 述語単位にスパイポイントを設定できる。

■off
トレースは行わない。



トレース表示例　（図－１）

これらのトレース・モードは、プロセス毎に設定することが出来る。トレース時には、

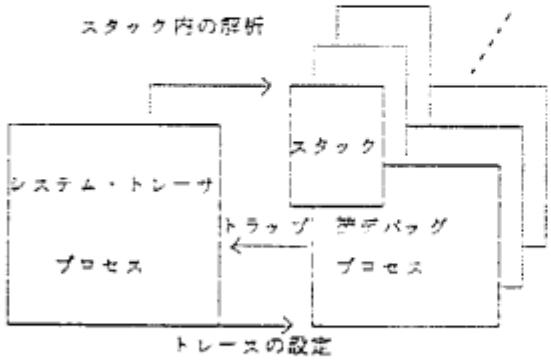
○プロセス番号
○レベル
○何番目の節か、又は　組込述語
○成功／失敗
○述語名、各引数の値

をコンソールに表示する。（図－１）

３．トレース機能の実現法

システム・トレーサは、KLO step trap やKLO clause trap に対するトラップ・ハンドラーという独立したプロセスとして定義される。 IPLによってロードされ初期化を行ったあとは被デバッグ・プロセスからのトラップを待ち、トラップが発生すると起動するようになっている。(図－２)被デバッグ・プロセスからのトレース・トラップがかかると、システム・トレーサは引数の値を表示するが、この引数の値は被デバッグ・プロセスのスタック上に積まれている。 PSIでは各プロセス毎に別々のスタックを持っているため、システム・トレーサは引数の値を得るために他プロセスのスタック内を探索しなければならない。そのためにファームウェアより以下のような情報を提供してもらう。

○トラップ時の実行コード・アドレス（IBR）
○トラップ時のローカル・スタックのベース・アドレス（CLBR）
○トラップ時のグローバル・スタックのベース・アドレス（CGBR）



システム・トレーサと被デバッグ・プロセスの関係
（図－２）

これらの情報より、ファームウェアが行っているようにヒープ上のコードとスタック上の変数の値より引数値を得る。

４．ｓｙｍｂｏｌ管理

システム・トレーサの一つの特徴として、symbolicなデバッグができるということが上げられる。 このsymbol管理を行っているのが　シンボライザである。シンボライザは、アトム名ーアトム番号の対応を主記憶上に保持しており、アトム名ーアトム番号の相互変換や整数ー整数の表示イメージの変換等SIMPOSが扱うデータの外部形式と内部形式の相互変換を行う。

５．その他の機能

システム・トレーサは、トレース機能の他に以下のような機能を持っている。

■ｓｐｙ機能
　指定された述語のヘッド・ユニフィケーション終了時でのみトレースを行う。 この述語（スパイ・ポイント）は、複数個指定できる。
■メモリ・レジスタ類の表示・変更機能
　メモリ及び一部のレジスタ類の値をコンソールに表示し、メモリについては変更もできる。メモリのアドレスの指定には述語名も許される。
■シンボル値の表示機能
　アトム名から対応するアトム番号をコンソールに表示する。又は、アトム番号から対応するアトム名を表示する。
■オブジェクトのスロット値の表示機能
　指定されたオブジェクトの状態（すべてのスロットの値）をコンソールに表示する。スロット名が指定された場合は、その値のみを表示する。
■端末割込による起動機能
　端末割込によってシステム・トレーサが起動する。これにより、ユーザは何時でもシステム・トレーサを呼び出すことができる。

６．おわりに

現在、システム・トレーサはSIMPOSのデバッグに利用されている。今後は、プリンターやファイル等へのログ機能、割込等のイベント・トレース機能、メモリのsymbolic表示等の機能拡張を考えている。

関係データベースエンジンの開発（その1）

ーハードウェア構成ー

岡 孝夫　松田 進　　岩田 和秀　神谷 英雄
（（株）東芝 青梅工場）　（（株）東芝 総合研究所）

1. はじめに

通産省第5世代プロジェクトの一環として、関係データベースマシン Deltaのユニットのひとつである関係データベースエンジン（以下、RDBEと略記する）の開発を行った。

RDBEは、ソート処理を主体とした関係演算を専用ハードウェアで、また算術系の演算と全体の制御をソフトウェアで行うことにより、関係データベースの処理が効率よく行われるよう設計されている。本稿では、Deltaの概要と、RDBEのハードウェア構成の概要を報告する。

2. Deltaの概要

Deltaは、図1に示されているように、HMサブシステムとRSPサブシステムより構成され、ネットワークを介してホスト・マシンから送られてくるコマンドを実行する。RSPサブシステムはIP，CP RDBE，MPの各ユニットから構成され、各ユニットが次のような機能を分担している。

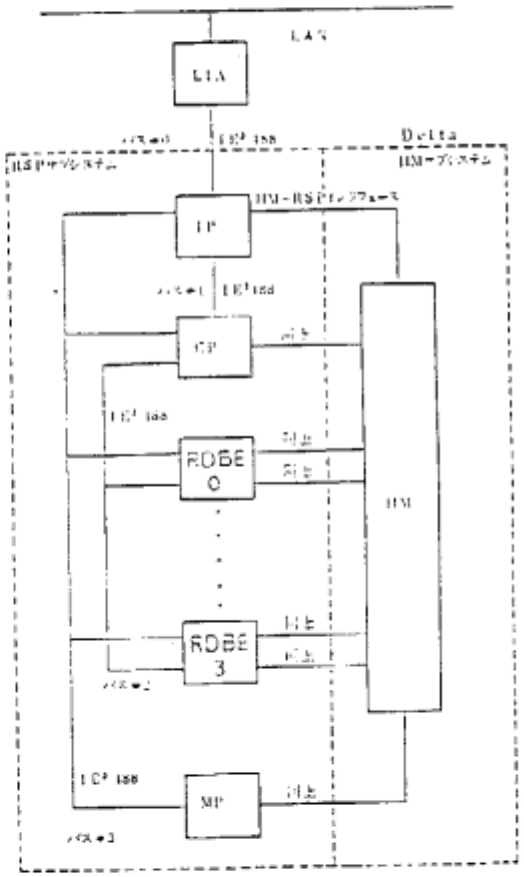
IPは、LIAと、CP及びHMとのインタフェースをとるプロセッサ、 CPは、RDBE、HMなどの主要なハードウェアのリソースを管理するプロセッサ、RDBEは、CPに制御されて、HMよりストリームで供給されるリレーションに対して関係演算を施すプロセッサ、MPは、Deltaの各プロセッサを監視するのを目的とするプロセッサである。

3. RDBEのハードウェア構成

関係データベースの処理を効率よく行うためには、ソート処理や、Restrict，Join等の関係演算を高速に行う必要がある。このため、RDBEでは、ソート処理と、関係演算処理を連動して、入力データに同期したパイプライン処理を行う専用ハードウェア（以下、エンジン・コアと略記する）を設計しこれをCPU及びその制御プログラムで制御する方式を採用した。

図2は、RDBEの機能構成例であり、点線で囲った部分がエンジン・コアである。エンジン・コアは更にINアライナ、ソータ、マージャの3つのモジュールから構成されている。INアライナは、入力データのフォーマット変換を行うハードウェア、ソータはソート処理専用のハードウェア、マージャは関係演算処理専用のハードウェアである。

ソータとマージャは2個のレコード間のKEYフィールドの比較を行い、比較結果が確定した時、演算の種類と比較結果に応じて、出力すべきレコードを確定する。したがって、ソータとマージャの各部では、KEYフィールドの比較が終了するまで、そのレコードをバッファリングする必要がある。しかし、パイプライン処理の単位毎にレコードのバッファリングを行うことは、ハードウェアを大型化し、処理速度を低下させる。そこで、RDBEではエンジン・コアの入口にあるINアライナに1レコード分のバッファメモリを用意して、KEYフィールドをレコードの先頭に移動させるレコード・フォーマット変換を行うようにした。
（図3（a），（b）を参照）



LAN : Local Area Network
LIA : LAN Interface Adaptor
RSP : RDBH Supervisory Processing Subsystem
HH : Hierarchical Memory Subsystem
IP : Interface Processor
CP : Control Processor
REBE : Relational Data Base Engine
MP : Maintenance Processor

図1　　Deltaの構成図

ソータは、１２個のソート・セルと、ソート・チェッカより構成される。１２個のソート・セルは、２ＷＡＹパイプライン・マージソート方式により、４Ｋレコードのソーティングを行う。また、ソート・チェッカは、１レコード分のバッファ・メモリを持ち、ソータの出力の連続した２レコードの大小比較を行うことにより、ソート結果が正しいか否かを監視する。

マージャは主に、２個のリレーション間の関係演算を行う専用ハードウェアであり、各リレーションを格納するために、２個のバッファ・メモリを用意している。マージャは、ソートされたリレーション（Ａ）を一方のバッファ・メモリに格納し、次にリレーション（Ｂ）がソータより転送されて来ると、他方のバッファ・メモリに格納しつつ、２つのリレーション間の演算処理を開始する。従って、マージャは、エンジン・コアへの入力に同期した速度で関係演算を実行する。また、マージャは、その出力部にリレーション（Ａ）あるいはリレーション（Ｂ）に対応したレコードバッファを持ち、マージャでの処理結果が出力される時にレコード・フォーマットの逆変換を行うことにより、エンジン・コアへ入力されたフォーマットに戻す。

本エンジン・コアでは、ソータ、マージャを直結しているので、マージャに入力されるリレーションは、ソータによりあらかじめソーティングされている。マージャは、これを前提として演算を行うので、関係演算処理の性能を著しく向上することができる。以上のように、エンジン・コアは、ＣＰＵの制御プログラムから、演算対象となるリレーションの総レコード数、レコード長、ＫＥＹフィールド長、ＫＥＹフィールドの位置、演算の種類、等の各種パラメータを受取り、各モジュールが連動して動作することにより、全体としてその機能を実現する。

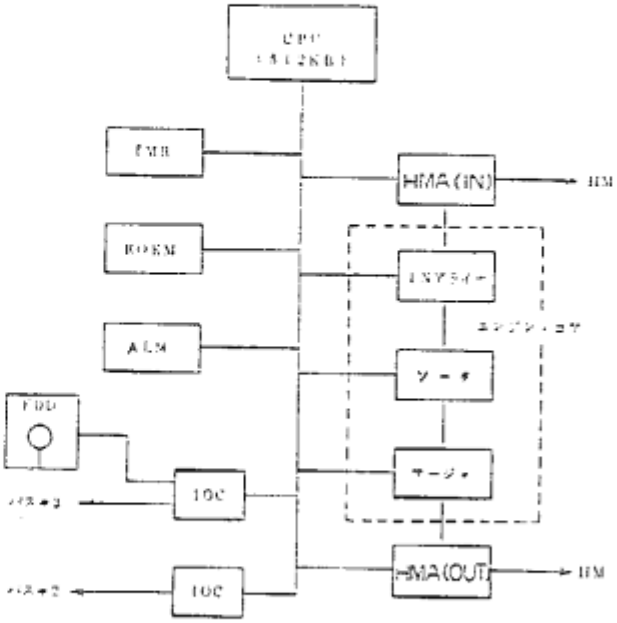
ＨＭアダプタ（以下、ＨＭＡと略記する）は、ＲＤＢＥとＨＭとのインタフェースを制御する機能を有すると同時に、ＨＭＡ（ＩＮ）は、エンジン・コアへのデータの入力ポート、ＨＭＡ（ＯＵＴ）は、エンジン・コアからのデータの出力ポートとして動作する。

ＨＭＡが、エンジン・コアの入出力ポートとして動作するときのデータ転送は、ＨＭ／エンジン・コア間及びＣＰＵのメモリ／エンジン・コア間で可能である

本マージャには算術演算系の機能を設けなかったので算術演算が必要な時には、ＣＰＵのメモリ／エンジン・コア間のデータ転送機能と、ＣＰＵでの算術演算機能を用いてその処理を行うように設計した。

4.　おわりに

本稿では、Ｄｅｌｔａの中核ユニットであるＲＤＢＥのハードウェア構成について述べた。ＲＤＢＥを構成する各モジュールの詳細およびＲＤＢＥの制御ソフトウェアについては、別稿で報告する。謝辞　本研究に関し、ご指導いただきました ＩＣＯＴ　ＫＢＭグループの方々に深謝いたします。



TMR　：計時装置
ROEM：外部固定記憶装置
ALM　：警報装置
ＩＯＣ　：入出力制御装置

図２　ＲＤＢＥのハードウェア構成

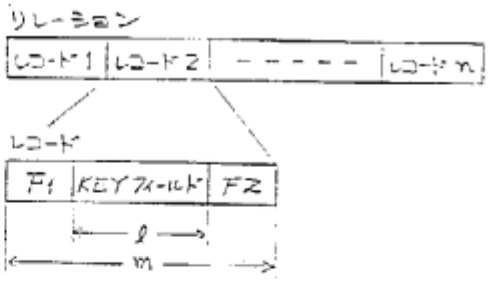


図３（ａ）　入力データ

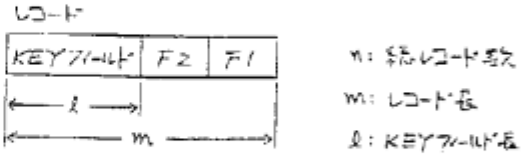


図３（ｂ）　ＥＮアライナの出力

〈参考文献〉
[１] 角田，柴山，横田　他「RDBM Delta(1)-(3)」情報処理学会全国大会予稿，PP4f-6-8, 1983
[２] 神谷，亀井，安部，星野，伊藤　他「関係データベースエンジンの開発（その2-6)」本予稿，PP4f-6-10, 1984

関係データベースエンジンの開発（その２）

－　ソータ　－

4F-6

冷谷　良雄　　岩田　和秀　　酒井　浩　　松田　進

（　㈱東芝　総合研究所　）　　（㈱東芝　青梅工場）

1．　はじめに

通産省第５世代プロジェクトの一環として、関係データベースマシン"Ｄｅｌｔａ"[1]のユニットのひとつである関係データベースエンジン[2]の開発を行った。

本稿では、ｎレコードのソートに対し、
ハード量：Ｏ（$\log_2 n$）
処理時間：Ｏ（ｎ）
入力完了から出力開始までの遅れ：Ｏ（$\log_2 n$）
の性能を持ち、さらに、null値を含み、Ｋｅｙが任意の位置にある最大4096バイト長のレコードに対し、昇順、降順の安定なソート処理ができるソータと、ソート処理がベースとなる関係代数演算と集合演算をソータと同速度でパイプライン的に処理できるマージャのハードウェア構成の概要を報告する。

2．　ソータ

本ソータは、関係データベースの処理に応用するという観点から、以下に述べる点に留意して設計されている。

(a) 関係データベースの処理では、レコードのＫｅｙフィールドを対象としたソート処理と、集合演算のようにレコード全体を対象としたソート処理がある。本ソータは、この両方の処理に対応できるようにするため、ソートの対象となるレコード長を制御プログラムで指定できるようにし、 2～4096バイト長のレコードの昇順、または、降順ソートできるようにした。さらに、Ｋｅｙフィールド長も、 2～4096バイトの長さのレコードに対してソート処理できるようにした。

(b) 関係データベース処理では、Ｋｅｙフィールドの値が同値の時と、null値の扱い方を考慮する必要がある。そこで本ソータでは、Ｋｅｙフィールドの値が同値の場合は、レコードの入力順に出力する安定なソートを行い、null値を含む場合は、正常値のレコードの後にnull値のレコードを入力順に出力するようにした。

(c) 関係データベースの処理では、種々のデータ型式が扱われるが、ソータのハードウェアは、２進数（符号なし）のみを扱うことにしている。そのため、整数と浮動小数点数は、ソータの入力側（ＩＮアライナ）と出力側（マージャ）で、２進数へのデータ型式変換と、その逆変換を行うようにした。

(d) パイプライン方式でソート処理を行うためには、レコードの先頭にＫｅｙフィールドが位置している必要がある。そこで、Ｋｅｙフィールドでソートする場合、ソータの入力側（ＩＮアライナ）と出力側（マージャ）で、Ｋｅｙフィールドをレコードの先頭に移動させる処理と、元に戻す処理を行わせることにした。

このように、本ソータは、入力側にＩＮアライナを、出力側にマージャを接続して、これらを連動動作させてソート処理を行わせている。そこで、２－ｗａｙマージ・ソート方式[3][4]を採用し、sequential input/sequential output型のソータを構成した。

ソータ＆マージャのブロック図を図１に示す。

本ソータは、12段のソート・セルからなり、これにより、最大4096レコードをソートできる。

また、ソート・チェッカは、ソート結果が正しいか否かを監視する。

3．　ソート・セル

ソート・セルは、前段のソート・セルまででソートされている2^{n-1}個のレコードを２組入力し、2^n個のレコードをソートし、後段のソート・セルに出力する機能を有している。ここで、ｎの値は、ソート・セルが置かれている入力側からの段数であ

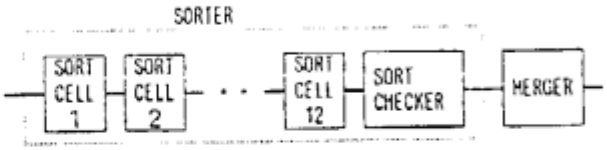


図1　ソータ&マージャのブロック図

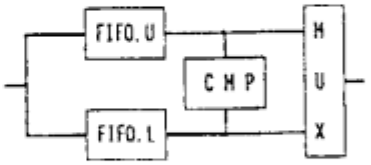


図2　ソート・セルのブロック図

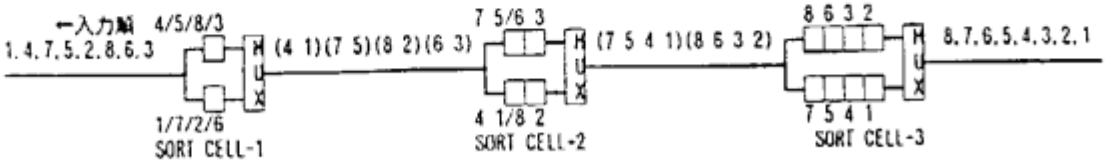


図3　ソート・セルの動作例

る。ソート・セルのブロック図を図２に示す。

ここで、FIFO.U/Lは、FIFO（First-In First-Out）機能を有するメモリで、さらに、現在読み出し中のレコードの先頭から再度読み出し可能な機能も有している。n段目ならば、それぞれ、2^{n-1} レコードを記憶できる容量を持つ。

CMP は、FIFO.U/Lから読み出されて来た値を比較する比較回路である。この比較結果により、いずれか側のレコードが選択回路MUX で選択され、後段のソート・セルに出力される。

次に、図３に８レコード（レコード長＝２バイト）からなるリレーションの並列ソートの動作例を示す。

４．　マージャ

マージャは、ソート処理をほどこされている１つないし２つのリレーション間で関係代数演算や集合演算の基本となる命令セットを高速に実行する専用ハードウェアである。マージャのブロック図を図４に示す。

４．１　演算部

演算部は、ＣＰＵから指示された命令セットを実行するメイン・コンポーネントである。

Buffer.U/Lは、それぞれが１つのリレーションを記憶できる容量を持っている。２リレーション間の命令では、各レコードは同期して読み出されＫｅｙフィールドの比較が行われる。

Buffer.U/Lは、レコード更新制御回路により制御されている。レコード更新制御回路は、FIFO機能と、現在読み出し中のレコードの先頭から、最初のレコードから等、前に遡って読み出す機能を実行する。

CMPS出力制御は比較回路と出力制御回路とからなり、後者は命令の種類と比較結果によりBuffer.U/Lのどちらか、あるいは、両方のレコードを出力すべきか否かを出力選択部に知らせる。

４．２　出力選択部

出力選択部は、演算部の補助動作として、演算部が出力対象指示を行ったレコードのみを選択出力する。出力する際、２進数に変換されているデータ型式を元のデータ型式、つまり、整数や浮動小数点数に戻し、Ｋｅｙフィールドの位置を元に戻す。

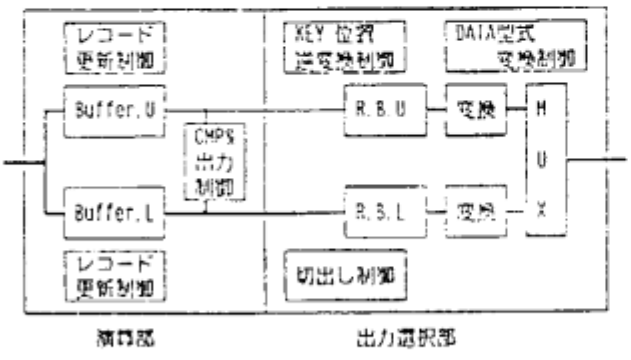


図４　マージャ・ブロック図

また、出力結果は、元のリレーションのレコードの全ての属性を必要とせず、その一部のみでよい場合が多い。そこで、必要な属性のみを切り出す処理「切出し制御」も行う。この指示がレコードの入力時にくるので、１レコード分のバッファR.B.U/L でバッファリングする。

このように、マージャは演算部も、出力選択部も、ソート・セルと類似の回路で実現できるので、ソート・セルと同じ実行速度でパイプライン処理を行う。

４．３　命令セット

マージャで処理する命令セットを以下に列挙する。なお、算術演算、たとえば四則演算は、ＣＰＵが処理する。[2)]

(a) ＰＡＳＳ系命令

対象リレーションをマージャに入力するか、出力するか、単に通過させるか、あるいは、重複を除いて出力する命令である。

(b) ＳＯＲＴ系命令

ソータの容量を越えるソート、あるいは、重複を取除く命令である。

(c) ＲＥＳＴＲＩＣＴ系命令

ひとつの対象リレーションの内から、条件に合うレコードのみを出力する命令である。

(d) ＣＯＭＰＡＲＥ系命令

２つのリレーション間で順に比較し、条件に合うレコードの組のみを出力する命令である。

(e) ＪＯＩＮ系命令

２つのリレーション間でレコードの総当りを行い、条件に一致するレコードの組を全て出力する命令である。

５．　おわりに

以上のハードウェアー式は、TTL のMSI/SSI を用い、基板１６枚で構成しており、既に完成し納入済みである。現在は、8000GateのGate Arrayを用いてソート・セルのＬＳＩ化を計っている。

謝辞　本開発に関し御指導いただきましたＩＣＯＴＫＢＭグループの方々に深謝いたします。

＜参考文献＞

1)角田、栗山、横田他、「RDBM "Delta" (1)-(3)」情報処理学会第26回全国大会予稿,1983,4F-6～-8

2)松田、酒井、安部、伊藤、屋野他、「関係データベースエンジンの開発（その１，３－６）」情報処理学会第29回全国大会予稿,1984,4F -5,-7～ -10

3)Todd " Algorithm and Hardware for a Merge Sort Using Multiple Processors" IBM J.R&D,vol 22,no5,September,1978

4)比田井他、「ソートをベースにしたデータベースマシン」情報処理学会第23回全国大会予稿,1981,4F-9

関係データベースエンジンの開発（その３）
～ハードウェアの制御方式～

°笹井　浩　　岩田　和秀　　安部　公朗　　神谷　良雄
（株式会社　東芝　総合研究所）

1　はじめに

通産省第５世代プロジェクトの一環として，関係データベースマシン"Delta"[1]のユニットの一つである関係データベースエンジン（RDBEと略記する）の開発を行った[2][3]。

本稿ではRDBE全体としての機能とそれを実現するためのハードウェア（HMアダプタとエンジン・コア）の制御方式について述べる。

2　RDBEの機能

RDBEの機能は，関係代数演算（JOINなど）の他，データのクラスタ化，更新などDelta に必要な処理のうち，エンジン・コアによる高速処理が可能なもの，および利用可能ではないがDelta での複数プロセサによる機能分散の設計方針からしてRDBEで処理すべきものを選定している。

RDBEの機能は，次のような５つ組で表現される。

（演算種別，演算対象，
　レコード切出し，TID 付加，CPU による演算）

演算種別は，文字どおり演算の種類を表わすものである。表１にその一覧を示す。表１からわかるように演算種別は，関係代数演算そのもの，あるいはそれと同程度に論理的な演算からなる。そして，その大部分は，RDBEのエンジン・コアで高速処理される。ただし，エンジン・コアの機能上の制約により，Joinをはじめとする多くの演算では一度にひとつの比較条件しか処理できない。従って条件が複数の場合は，ひとつの条件による演算に引続いて，別の条件による演算を行うようにするか，あるいはCPU による演算の部分に，ふたつめ以後の条件を記述する必要がある。

演算対象は，REで処理するレコードを格納しているHMのバッファID（高々ふたつ），演算対象フィールドの情報（データ型など），および結果を格納すべきHMのバッファID（常にひとつ）からなる。

レコード切出しは，射影演算に相当するものである。例えば一般にJOINという演算を行なう場合，もとのリレーションのすべての属性が必ずしも必要というわけではなく，そのうちの一部が必要であることが多い。そこでRDBEでは図１に示すように，もとのリレーションの一部だけを切出すことが，エンジン・コアでJoinと同時に行なえるように設計されており，ここでは切出す範囲を指定する。なお，片方のレコードを切出さないように指定することにより，SEMI-JOIN が実現できる。

演算種別	説　　明	実現手段
Pass	CPU による演算等に使用	HW
Restrict	属性と定数との比較による選択	HW
Compare	属性どうしの比較による選択	HW
Join	θ-Join を行なう	HW
Sort	ひとつの属性について並べかえ	HW
Unique	ひとつの属性について重複除去	HW
M-Sort	複数の属性について並べかえ	HW
M-Unique	複数の属性について重複除去	HW,SW
Set	集合演算	HW,SW
CHECK	集合の包含関係の判定	HW,SW
Aggregate	集約演算	SW
Zone-Sort	クラスタ化に使用する。	HW
Delete	データの更新に使用する。	HW

表１　　演算種別の一覧表

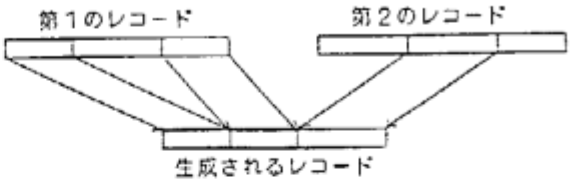


図１　　レコード切出し機能

TID 付加は，演算の結果生成される個々のレコードに，一連の通し番号を付加する機能を制御するものである。TID 付加の有無とTID の初期値が指定できる。

CPU による演算は，エンジン・コアの機能を補う目的で用意されている。詳細は[2] で報告する。

3　ハードウェアの制御方式

3．1　HHアダプタとエンジン・コアの制御方式

RDBEの機能は，エンジン・コアがほとんど処理しているので，エンジン・コアの制御方式だけを示せばよいようにみえる。しかし，実際にはエンジン・コアは入口に渡したレコードを処理し，演算結果を出口に準備するだけであるので，それをHHとの間で転送するHHアダプタも含めて制御方式についても述べる必要がある。それらは次のとおりである。

HHアダプタ(IN)は，HHからデータを受取り，それをエンジン・コアへ送る。JOINなどの演算では，演算対象となるふたつのリレーション中のレコードを，ブロック・マルチプレクスして転送する。つまり，まず第1のリレーション中のレコードを一定個数まとめて受取り，それをエンジン・コアに渡す。そしてエンジン・コアが全部のレコードを受取り終わると，次に第2のリレーション中のレコードを受取り，それをエンジン・コアに送るという方式である。その切替えは，RDBE制御プログラムがHHアダプタ(IN)を制御することにより行なう。

エンジン・コアは，HHアダプタ(IN)から一定個数まとめて渡されたレコードを，処理対象となるフィールドの大小の順に並べかえて内部のバッファに格納したり，あるいは，先に内部のバッファに格納しておいたレコードと，今渡されて並べかえたレコードを順次大小比較することにより，HHアダプタ(IN)からまとめて渡されたレコードについて，Joinなどの演算を行なう。従ってエンジン・コアは，常にHHアダプタ(IN)と同期して制御すればよい。

HHアダプタ(OUT)は，エンジン・コアからデータを受取り，一定個数ずつまとめてHHに送る。エンジン・コアから出力されるレコード数は，演算の種類やデータの値の分布によって変わるので，HHアダプタ(OUT)の動作はエンジン・コアとは非同期となる。なお，HHアダプタ(OUT)は通常エンジン・コアからの出力をひとつのストリームとしてHHに送ればよいが，非常に大量のデータの並べかえの場合に限り，2ウェイ・マージを行なう都合により，2つの中間結果をブロック・マルチプレクスして転送する必要がある。その切替えは，RDBE制御プログラムがHHアダプタ(OUT)を制御することにより行なう。

3．2　大量データの処理方式

ここでは，エンジン・コアで一度に処理できない大量のデータの処理方式について述べる。処理方式は，次に示すように演算の種類に応じて4通りある。

(1) Pass型

レコード間の比較を必要としない演算では，エンジン・コアで処理できる量ごとにブロック化して区切ってHHから入力することにより処理する。すなわち，

```
for i = 1, 2,....,n
        A_i の処理
```

ただし，A_i はi番目のブロックを表わしている。

(2) Join型

ふたつのリレーションに関する演算で，それぞれのリレーションの各要素のすべて対について調べればよい演算は，それぞれのリレーションをエンジン・コアで処理できる量ごとにブロック化し，すべてのブロックの対について処理する。すなわち，

```
for i = 1, 2,....,n
  for j = 1,2,....,n
        A_i とB_j の処理
```

(3) Difference型

差集合（A－B）を求めるような演算では，Bが大量の場合，Bをエンジン・コアで処理できる量ごとにブロック（B_1）に分け，（（A－B_1）－B_2）－… －B_n）を求めることにより処理する。

(4) Sort型

大量データの並べかえは，まずエンジン・コアの容量（最大 128KB）ごとに並べかえを行なう。次に，2ウェイマージを繰返すことにより並べかえを行なう。

4　結論

RDBEは，関係代数演算をはじめとする関係データベースの処理を，主にハードウェアで高速処理する。そしてその指定方法は，ハードウェアの制御を意識する必要がないという意味で非常に論理的である。

RDBEのハードウェア制御方式では，エンジン・コアだけでなく，HHアダプタとの関連が重要である。特にエンジン・コアの容量を超える大量データについては，HHアダプタ(IN)がエンジン・コアで処理できる量ごとにわけて受取ることにより処理する。

謝辞

本開発に関し，ご指導頂いたICOT K8Hグループの方々に感謝いたします。

参考文献

[1] 角田，栗山，横田他，「RDBH Delta（Ⅰ）～（Ⅲ）」，情報処理学会第26回全国大会予稿，4F-6～8，1983

[2] 安部他，「関係データベースエンジンの開発（その4）」，本予稿，4F-8，1984

[3] 岩田他，「関係データベースエンジンの開発（その1，2，5，6）」，本予稿，4F-5,6,9,10，1984

TM－078－27

関係データベースエンジンの開発（その４）
ＣＰＵによるデータ処理方式

安部公朗　酒井　浩　岩田和秀　神谷茂雄　（以上，東芝総合研究所）
松田　進　伊藤文英　（以上，東芝青梅工場）

１　はじめに

通産省第５世代プロジェクトの一環として，関係データベース・マシン”Ｄｅｌｔａ”のユニットの一つである関係データベース・エンジン（ＲＤＢＥと略記する。）の開発を行った。

ＲＤＢＥは，ＣＰから送られてくるコマンドに基づいて，処理の対象となるデータベース・データをＨＭアダプタ（ＩＮ）を用いてＨＭから入力して，エンジン・コアというハードウェアで処理を行い，処理結果をＨＭアダプタ（ＯＵＴ）を用いてＨＭに出力する。また，エンジン・コアだけでは処理できないものについては，ＲＤＢＥのＣＰＵでも処理を行う。（図１参照）

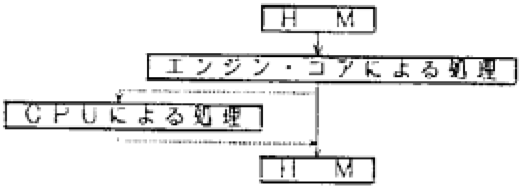


図１　ＲＤＢＥの処理の概要

本報告では，ＲＤＢＥのＣＰＵによるデータ処理とその実現方法について述べる。

２　ハードウェアとソフトウェアの機能分担

ＲＤＢＥは，ＣＰからコマンドを受取ると，通常は，処理対象となるデータベース・データをＨＭから入力し，エンジン・コアで処理を行い，その結果をＨＭへ転送する。しかし，ＣＰから送られてくるコマンドをすべてハードウェアで実現しているわけではなく，次に述べる機能については，ＲＤＢＥのＣＰＵで処理している。

(1) ひとつのタプルに含まれる複数のフィールドについて Null 値かどうかの判定
(2) 算術演算（２，４，８バイト整数，および単精度浮動小数点の四則演算）
(3) AND/ORで結合された複数の比較演算
(4) 集約演算（グループ分けを行い，各グループごとに最大値や最小値を求める演算）

これらの演算をハードウェアでなく，ＲＤＢＥのＣＰＵで処理するようにした理由は，新規に開発するハードウェアの量を大きくしないためと，これらの演算は使用頻度が小さいと推測されたからである。

これらの演算をハードウェアで実現するためには，現在のエンジン・コアにさらに次に述べる機能が必要となる。

(1) については，少なくともすべてのフィールドについて，そのタグの位置を記憶するためのメモリが必要である。ひとつのタプルに含まれるフィールドの個数は，Ｄｅｌｔａの機能仕様で最大２５６と規定されている。

(2) の演算は，桁上げを伴うのでアイテムの処理はＬＳＢ側からＭＳＢ側へ向かって行う必要がある。しかし，現在のエンジン・コアでは，アイテムをＭＳＢ側からＬＳＢ側へ２バイト単位で順次処理するようになっているので，アイテムを蓄えるためのバッファが必要となる。また，Ｄｅｌｔａの機能仕様では，８バイトの整数どうしの乗除算や Null 値の処理も規定されており，それらを処理するハードウェアの開発は困難である。

(3) については，複数の比較演算をするためには，比較演算の種類とそのオペランドを記憶するメモリが必要である。

(4) については，まずグループ分けをする行うために隣りあったアイテムについて，グループ分けの対象となるすべてのフィールドについて，その値が等しいかどうかの判定が必要である。そして各グループについて，最大値や平均値を求める必要がある。このため，(1)，(2) を実現するための機能はもちろんのこと，グループ毎の最大値や平均値を求めるために中間結果を格納すべきメモリが必要である。

３　ＣＰＵによる処理

ＣＰＵによる処理には，次のような種類がある。

(1) UNIQUE演算（ Null 値同志は互いに異なる値として扱う）。
　　これは，２の(1) に対応する。
(2) アイテム内のフィールドの値の変更。
　　これは，２の(2) に対応する。
(3) アイテムの指定された条件での絞り込み。
　　これは，２の(2)，(3) に対応する。
(4) 集約演算。
　　これは，２の(2)，(3)，(4) に対応する。

４　実現方法

４．１　コンパイラ方式の採用

ＣＰから受けとったコマンドを解析してＲＤＢＥのＣＰＵで直接実行可能なオブジェクト・コードを生成するコンパイラを作成した。これは，エンジン・コアから出力されたアイテムは，ＲＤＢＥの主記憶装置上に相当数連続配置されるので，それをなるべく高速に処理するためである。

なお，コンパイラは，ＵＮＩＸのソフトウェア開発のツールであるＹＡＣＣ（Yet Another Compiler-Compiler ）を用いて作成した。

４．２　データ処理方式

ここでは，条件に合うアイテムを出力する絞り込みの処理を例にとり，ＣＰＵによるデータ処理の大まかな流れを示す。

(1) ＣＰから受取ったコマンドを解釈して，ＣＰＵでの処理が必要かどうかを判定する。必要な場合には，コンパイラでオブジェクト・コードを生成する。
(2) ＨＭから，ＨＭアダプタ（ＩＮ）を用い，処理

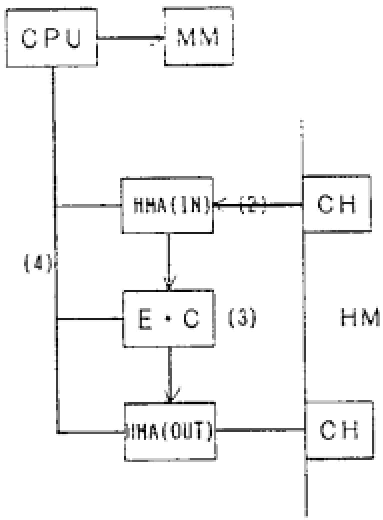
対象となるデータベース・データを入力する。
(3) エンジン・コアを用いて，処理対象となるデータベース・データに関係代数演算を実行する。
(4) (3)の結果をHMアダプタ（OUT）を用いてRDBEの主記憶装置に転送する。（図2参照）
(5) 主記憶装置上に格納されたデータに対して，(1)で生成したオブジェクト・コードを実行する。（図3参照。ここでは，整数型のあるフィールドF1と整数型のあるフィールドF2において，F1＞F2となるようなアイテムを絞り出す場合を例にとってある。）
(6) (5)の実行結果を図4に示すように，主記憶装置よりHMアダプタ（OUT）を用いて，HMに転送する。

5　おわりに

以上，RDBEのCPUによる処理とその実現方法について述べた。
CPUによるデータ処理においては，対象とする同一形式をした多数のアイテムに関して，同じ論理を何回も繰り返し行えばよいので，CPから受取ったコマンドをコンパイルして，CPUで直接実行可能なオブジェクト・コードを生成する方式をとった。これにより，インタプリタ方式に比べて処理の高速化をはかっている。

謝辞

本開発に関し，ご指導戴いたICOT　KBMグループの方々に深く感謝致します。



HMA(IN)　：IN側のHMアダプタ

E・C　：エンジン・コア

MM　：主記憶

HMA(OUT)：OUT側のHMアダプタ

CH　：ブロック・マルチプレクサ・チャネル

図2 CPU処理以前のデータの流れ（その1）

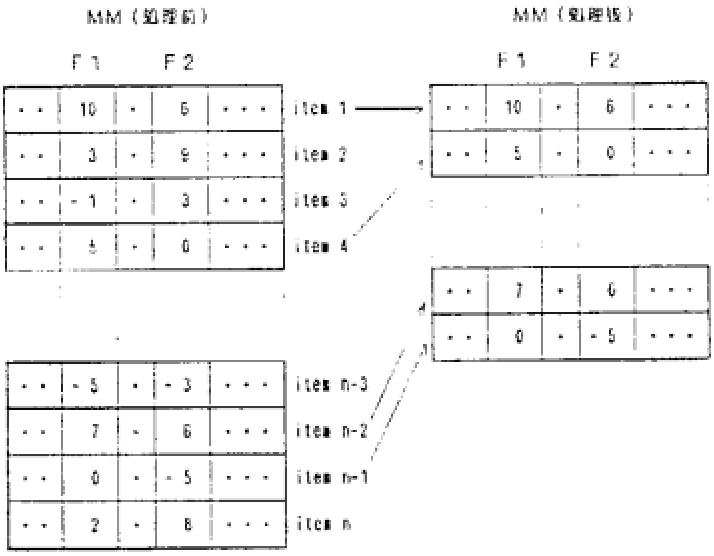


図3　CPUによる処理の例

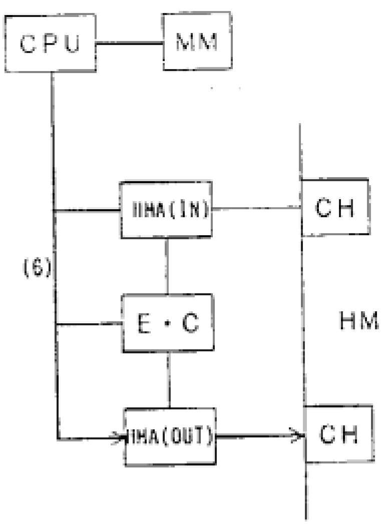


図4 CPU処理後のデータの流れ（その2）

関係データベースエンジンの開発（その５）
～関係データベースエンジンに対する処理指定～

伊藤　文英　　星野　恭夫
（株式会社東芝　青梅工場）

福井　格
（株式会社東芝　総合研究所）

１．　はじめに

通産省第５世代プロジェクトの一環として，関係データベースマシン“Delta”[1] のユニットの一つである関係データベースエンジン（RDBEと略記する）の開発を行った．[2]

本稿では，Delta に対するホストからの問合せに基づき，コントロールプロセッサ（CPと略記する）からRDBEに対して，どのような処理が指定されるかについて報告する。Delta では関係を属性方向のデータの集まりとして格納するため，RDBEに対する処理指定にもその特徴が反映されている。

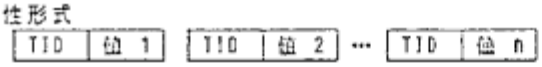
２．　処理されるデータの形式

RDBEが処理するデータのアイテムの構造を図１に示す。データの形式には次の２種類がある。

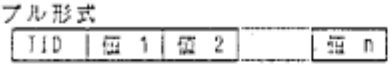
①　　属性形式
②　　タプル形式

図１　データの形式

属性形式



タプル形式



各アイテムには，属性形式のデータの間でのタプル方向のつながりを示すために，タプル識別番号（TID と略記する）がつけられている。TID はDelta 内でのみ使用され，ホストからは意識されない。

属性形式とタプル形式の間でのデータ形式の変換は，CPの指示により，データの置かれている階層構造メモリサブシステム（HMと略記する）でおこなわれる。ただし，属性形式からタプル形式に変換する場合は，属性形式の各データはTID で同順に並べられていなければならない。

３．　RDBEにおけるデータ処理

RDBEにおけるデータ処理は次の２種類に大別される。

①　　ホストからのデータベース問合せに基づく演算
②　　上記演算の入力となるデータ，あるいはホストに出力するためのデータを作成する演算

３．１　データベース問合せに基づく演算

Delta がホストに提供するコマンドと，それを実現するためのRDBEに対する命令の対応を表１に示す。各演算は次のようにおこなわれる。

(1) 制約系演算

①　　定数との比較条件による制約では，条件となる定数をCPから与え，属性形式のデータの中から条件をみたすアイテムを選択する。
②．　属性同士の比較条件による制約では，TID で同順に並べられた２つの属性形式のデータのアイテムを順次比較し，条件をみたすアイテムの組を選択する。
③　　算術演算の結果による制約では，条件中にふまれる属性を全て含むタプル形式のデータの中から，条件をみたすアイテムをCPU での処理により選択する。
④　　単結合では，対象および条件となる２つの属性形式のデータを用意し，対象となる属性形式

表１　データベース問合せに基づく演算

コマンド	操　作　概　要	RDBEに対する命令	入力データの形式
SELECTION	定数との比較条件による制約	定数とのJoin	属性形式
〃	属性同士の比較条件による制約	Compare	属性形式
〃	算術演算の結果による制約	Pass＋CPU	タプル形式
JOIN	θ結合，自然結合	Join	属性形式
SEMI-JOIN	単結合	Restrict	属性形式
集合演算系	和集合，共通部分集合，差集合	Set	タプル形式
統計処理系	総和，最大値，最小値，平均	Aggregate ＋CPU	属性形式
算術演算系	属性と定数との演算	Pass＋CPU	属性形式
〃	属性同士の演算	Compare ＋CPU	属性形式
検査系	集合の包含関係	Check	タプル形式
SORT	複数属性による並べかえ	H-Sort	タプル形式
UNIQUE	重複タプルの削除	H-Unique	タプル形式

のデータの中から条件をみたすアイテムを選択する。

(2) 結合系演算
θ結合，自然結合では，対象となる2つの属性形式のデータから，条件をみたすアイテムの組に新しいTID（結果リレーションのTIDとなる）をつけたものが出力される。

(3) 集合演算，検査，SORT，UNIQUE
これらの演算はタプルを1つの要素とみなすので，タプル形式のデータに対し，それぞれの演算をおこなう。

(4) 統計処理
属性形式のデータに対し，CPUで統計処理をおこなう。

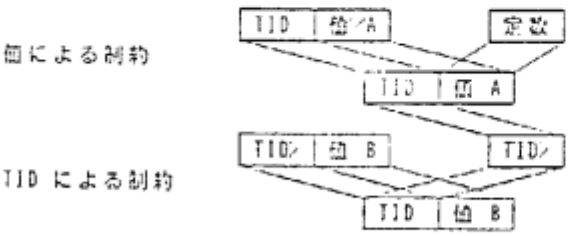
(5) 算術演算
① 属性と定数の演算の場合は，定数をCPから与え，属性形式のデータに対してCPUで演算をおこなう。
② 属性同士の演算の場合は，TIDで同順に並べられた2つの属性形式のデータのアイテムを順次入力し，CPUで演算をおこなう。

3．2 データを作成する演算

データを作成する演算には次のものがある。
① TIDによる制約
② TIDによる結合
③ TIDによるソート

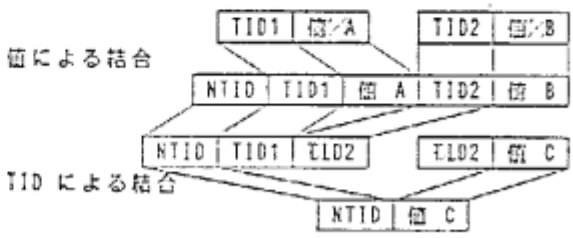
(1) TIDによる制約
データベース問合せに基づく制約系演算は，条件となる属性のデータのみを使っておこなわれる。ある属性のデータにかかった制約を他の属性のデータに反映する場合は，制約がかかった属性のデータのTIDを条件として，他の属性のデータをTIDで制約する。
この様子を図2に示す。あるリレーションの属性Aにかかった制約を他の属性Bに反映させる場合は，制約後の属性AのTIDを使って属性Bを制約する。

図2 TIDによる制約
////：比較フィールド



(2) TIDによる結合
データベース問合せに基づく結合系演算は，条件となる属性のデータのみを使っておこなわれる。結果として，条件をみたすアイテムのTIDの組に新しいTID（結果リレーションのTIDとなる）をつけたものが出力される。結合条件に含まれない属性の結果リレーションのデータを作る場合は，このTIDの組とその属性のデータの間でTIDによる結合をおこなう。
この様子を図3に示す。2つのリレーションは属性Aと属性Bに関する条件で結合されている。属性Bと同じ対象リレーションに含まれる属性Cの結果リレーションのデータを作る場合は，TIDの組と属性Cのデータの間でTIDによる結合をおこなう。

図3 TIDによる結合
////：比較フィールド



(3) TIDによるソート
TIDで同期に並べられた2つ以上の属性形式のデータを作成する場合は，それぞれのデータに対して，上記のTIDによる制約，結合をおこなったあと，TIDでソートする。タプル形式のデータを作成する場合は，全属性についてこの処理をおこなったあと，CPからHMに対してデータ形式の変換が指示される。

4． まとめ

CPからRDBEに指定される処理には，次の特徴がある。
① Deltaでは関係を属性方向のデータの集まりとして格納するため，演算は必要な属性のデータのみを使っておこなわれる。
② ホストからのデータベース問合せに基づく値による演算と，データを作成するためのTIDによる演算がある。
また，Deltaは知識ベースマシンのデータベースとして使われるので，知識ベース特有の問合せに対して効率のよい処理をおこなう方法を検討中である。

謝辞
本開発に関し，御指導頂いたICOT KBHグループの方々に深謝いたします。

参考文献

[1] 角田，栗山，横田他，「RDBH Delta（I）～（III）」，情報処理学会第26回全国大会予稿，4F-6～8，1983
[2] 松用，神谷，酒井，安部，星野他「関係データベースエンジンの開発（その1～4，6），本予稿，4F-5～8，10，1984

関係データベースエンジンの開発（その６）

～エンジンの利用制御演算系の位置づけ～

4F-10

渡野 康元　　伊藤 文英
（（株）東芝 青梅工場）

酒井 浩
（（株）東芝 総合研究所）

1. はじめに

通産省第５世代プロジェクトの一環として、関係データベースマシン"Delta"[1]のユニットの一つである関係データベースエンジン[2]（RDBEと略記する）の開発を行った。

本稿では，関係データベースエンジンについての報告の（その６）として，RDBEについてのまとめの意味で，関係データベースマシンDelta システム全体の中でのデータおよびその演算子群の概念的構成について述べ，RDBEの機能的位置づけの説明としたい。

Delta は、ホストとのインタフェースプロセッサ（IP）、DB制御プロセッサ（CP）、エンジン（RDBE）、階層構造メモリ（HM）、Delta 全体の監視プロセッサ（MP）を構成要素とする機能分散と、エンジンを複数個装備することによる負荷分散を組合せたシステムである。

2. Delta が保持するデータの単位と演算系

Delta では関係データに関する関係演算にソート・マージアルゴリズムを用いるという目的から次のような特徴を持つデータ単位・演算系を持つ。（図１，表１参照）

(1) カラムデータ

Delta では，各リレーションを行方向（関係方向）に見てタプルを並べたデータ形式にするのではなく，カラム方向すなわち属性方向に分割して各属性値のまとまりを単位として扱っている。

この様な分割により、各属性ごとに独立したバリュークラスター化が可能となり、またそれぞれ独立してソート処理が行え、属性と属性の間でのマージ処理などにも効果的となる。このことは、属性を単位として単純化されたハードウェア向きアルゴリズムの演算子系が用意できることを意味する。

ただし、その見返りとしてタプルとしてのまとまりが論理的に保存されるように、それぞれの属性のそれぞれのバリューごとにタプル番号（タプルID）が必要となることも事実であり、"タプル形式への再構成"のような演算が増えることになる。しかしながら、これはハードウェアによる演算装置作成による高速化が可能になることから、パフォーマンスの点では十分補償されてしまう面分である。

このカラムデータは，階層構造メモリ装置に記憶される。他の制御系からはカラムデータの物理的構造とある程度の低レベルの論理構造を意識せずに、その生成、消滅、参照等の制御ができるようにしたい。そのために，階層構造メモリ自身がそのインテリジェンスとして、それらの制御用演算子系を装備することになった。これによりHM以外の制御系からは各属性に対応するカラムデータが抽象データ型として扱える。

(2) ディレクトリデータ

階層構造メモリは，リレーションを分解した一段プリミティブなカラムデータを抽象データ型化して扱っている。したがって、リレーションとしての意味単位に再構成できるためにはさらにカラムデータとリレーションとの対応関係や、リレーション自身に関する情報が必要であり，これらをまとめてディレクトリデータと呼ぶことにする。

ディレクトリデータに対する直接のアクセスは、その物理構造に依存した検索、生成、更新、消去等の操作であり、これらの操作用演算子系はDB制御プロセッサであるCP上に用意されている。

(3) リレーションデータ

(1)、(2) よりカラムデータとディレクトリデータに対するアクセスは独立した演算系によることになっているが、関係データベースマシンとして扱うのはリレーションデータであり、カラムとディレクトリの双方のデータを同時に関連させて扱うことにより始めてリレーションとしての意味的にまとまったデータ操作が行える。

したがってリレーションというデータ単位でアクセスを行うような演算系がDELTA 内部においても必要である。この演算系は(1)、(2) で述べた２つの演算子系を組合わせさらに意味処理を行うことでリレーションという１段上のレベルのデータ型を抽象化している。

以上のデータはDelta 内に保持され，ホストからの要求により消去されない限り存続するという意味で，パーマネント・データである。

3. 関係データベースエンジン

3.1 演算の過程で生成、操作するデータ

(1) サブタプル

ホストからDelta への関係演算指令の一つ一つに対してDelta 内では種々の演算を組合せた演算子シーケンスを実行する．その実行過程では必要最小限個数の属性、即ち、カラムのみを組み合わせながら進むので，演算の最終目的の完成タプルから見ればサブタプルと呼ぶべきデータ形式が現れる。

サブタプルデータこそが関係データベースエンジンの演算対象となるものであり、サブタプル型を抽象データ型化する演算子のセットを提供している。(3.2節参照)

(2) サブディレクトリ

このサブタプルデータは演算の過程で作られる一時的なものではあるがホストからの指示によっては新しいカラム型データとしてパーマネント化されたり、ホストからの次の関係演算指令で演算対象と指

図１　Delta 内データ・演算系の構成

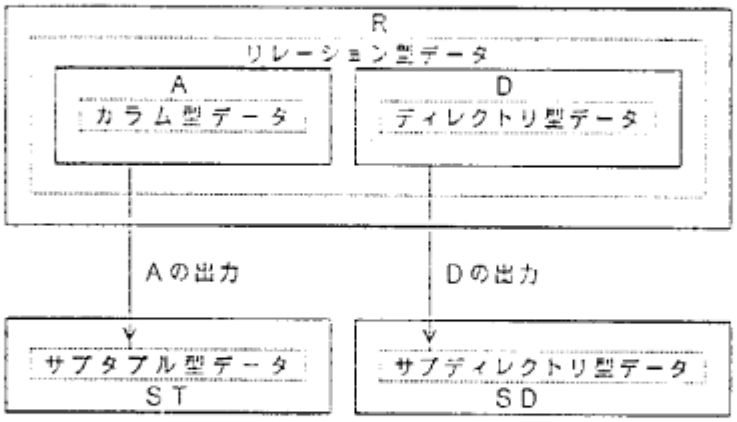


表１　各抽象データタイプの演算系の機能と演算担当プロセッサ

抽象データタイプ		機能	演算担当プロセッサ
A	カラム型	属性定義系，属性参照系	階層メモリプロセッサ（HM）
D	ディレクトリ型	ディレクトリ操作系	ＤＢ制御プロセッサ（ＣＰ）
R	リレーション型	リレーション定義系，定義参照系	〃
ST	サブタプル型	関係演算用ストリーム演算系	関係データベースエンジン（ＲＥ）
SD	サブディレクトリ型	サブディレクトリ操作系	ＤＢ制御プロセッサ（ＣＰ）

定されたりする。

このためにこのサブタプルデータに対してもそれ自身のスキーマ構造や、元になったリレーション、演算の履歴等の情報を保持しておかなければならない。これは、２章で述べたディレクトリデータを元にして作成されるものであり、サブディレクトリと呼ぶことにする。

サブディレクトリにはディレクトリデータと同様のスキーマ情報も含まれるが、前述のような固有の情報があり、ディレクトリデータとは異なる操作用演算子が用意されている。

以上のデータ型は、２章で述べたパーマネントデータを元にして生成されたデータであり、トランザクションという仕事単位の間だけ存在するテンポラリデータである。

３．２　エンジン演算系

エンジンは，このサブタプル型データを抽象化する演算制御系プロセッサとして位置づけられている。（図１，表１参照）

ホストからDelta へ送られた各関係演算コマンドはDelta 内部演算子の列（すなわち図１，表１の演算子の列）に変換される。DB制御プロセッサがその演算子列の実行制御を行い，各演算子をそれの属するプロセッサに指令して実行させる。エンジンに対しては，関係演算の下位レベルであるストリーム演算系の演算子が指定され，階層構造メモリ上のサブタプル型データに対する操作を行う。すなわちRDBEとHMとを合せると、演算手系とデータ実体の双方を包含しているという意味で、サブタプル抽象データ型マシンと呼べる。

４．　おわりに

Delta 内でのデータと演算セットの組み合わせをデータタイプの抽象化の視点から整理してみた。現実のインプリメントでは関係演算系以外の種々の制御や例外処理的な機能をサポートするためにデータタイプの抽象化の考え方から逸脱した部分もあるが，全体としてはほぼ本稿の内容に沿ったものとなっている。

謝辞

本開発に関し、御指導頂いたICOT KBMグループの方々に深謝致します。

参考文献

[1] 角田，柴山，横田他　「RDBM Delta（１）－（３）」，情報処理学会第２６回全国大会予稿，pp 4F-6-8 ，1983

[2] 松田，神谷，酒井，安部，見野他　「関係データベースエンジンの開発（その１－５），情報処理学会第２９回全国大会予稿，pp4F-5-9 ，1984

PROLOGソースレベル・オプティマイザ

2F-5 ― 決定性の検出とその利用 ―

沢村一・竹島卓・加藤和彦

(富士通国際研)

1. はじめに

Prologプログラムをソースレベルで最適化（改良）するPrologソースレベル・オプティマイザを試作した〔1, 2〕。 Prologはバックトラック機構に基づく非決定性プログラミング言語であるためプログラムのデータ及び制御の流れは極めて複雑である。

それゆえ、Prologプログラムの複雑なふるまいを反映したプログラムの最適化規則には様々な前提条件が必要になることが多い。 このような前提条件の中で、オプティマイザを構成する上で最も重要であったのは述語の決定性である。

しかしながら、述語の決定性を一般的に検出することは不可能と思われるので、本稿では決定性の部分クラスとしてのc－決定性なる概念を定義し、それの次のような最適化手法への利用について述べる：

（i） 部分評価（インライン展開、局所的最適化手法）

（ii） カットの自動挿入

2. 決定性とc－決定性

〔定義 1〕 決定的述語

述語が決定的であるとは、その定義節の高々一つの節で成功するのみで、バックトラックしたとき再び成功することはないことをいう。

〔定義 2〕 c－決定性

述語がc－決定的であるとは次の条件を満たすときをいう：

（i）組み込みの決定的述語はc－決定的述語である。

（ii）述語名H に関する定義体を次のものとする：

```
H1 :- Γ1.
 ⋮
Hi :- Γi, [!,] Δi.,  ここで、!は最右とする
 ⋮
Hn :- Γn.
```

このとき、次の条件が成立するならばH はc-決定的述語である； 各i に対して、

① Hi の本体にカットが存在しないとき：

Γi,Δi はc-決定的で、さらにHiは定義節の最後の節である。

② Hi の本体にカットが存在するとき：

Δi はc-決定的である。

〔系〕 c－決定的述語は決定的述語である。

〔注意〕 c－決定性はインライン展開〔1, 2〕によって保存される。

〔例 1〕 c－決定的述語の例

```
p (a ) :-  write (a1) ,nl, !. write  (a2) .
p (b ) :-  q,write (b ) .
```

3. c－決定性の最適化への利用

我々のPrologオプティマイザではインライン展開を次のような自然な方策の下に行っている：

「述語の呼びを、呼出機構を表す等式を付随した代替節の選言によって置き換える」。

しかしながら、これが安全に行えるのは呼ばれる側の述語の定義体の中にカット記号が現れないときである。 というのは、バックトラックが起こったとき、展開前と展開後のプログラムではカットによってその振る舞いに違いが生じるからである。 たとえば、

```
展開前：  p :- q,a,r.
          p.
          a :- b, !,c.
          a :- d.
展開後：  p :- q, (a = a, b, !,c ; a =a,d ) ,r.
          p.
          a :- b, !,c.
          a :- d.
```

において、展開前のプログラムでcが失敗したとすると制御はqに移るが、他方展開後のプログラムでは制御はpの呼びを失敗させることになる。

(1) インライン展開における利用

このような問題を避け、上記のようなインライン展開を可能にする一つの方法はc－決定性の概念を用いることである。 すなわち、次の最適化図式が成立する。

```
a1 :- Γ1.
 ⋮
ai :- Γi, p (A ) , Δi.
 ⋮
an :- Γn.
p1 (A1) :- Θ1. ,  Θi (1 ≤i ≤m ) のいずれかに
 ⋮                 カットが含まれているものとする
pm (Am) :- Θm.
```

```
a1 :- Γ1.
 ⋮
```

```
ai :- Γi, ( p (A) = p1 (A1') , Θ1' ; ・・・ ;
          p (A) = pm (Am') , Θm') , Δi.
  ⋮
an :- Γn.
p1 (A1) :- Θ1.
  ⋮
pm (Am) :- Θm.
```

ここで、Ai', Θi' はそれぞれAi, Θi をリネームしたものを表す。 さらに次の条件を満たす；

(i) Γi の中にカットが存在しないとき；

Γi の中のすべての述語はc－決定的であり、aiは定義節の最後の節である。

(ii) Γi の中にカットが存在するとき；

Γi の中の最も右にあるカットの右側にあってΓi に含まれるすべての述語はc－決定的である。

［例 2］ カットを含む定義体によるインライン展開

```
q (a) :- !, write (a) ,p (X) .
q (b) :- write (b) ,p (Y) ,write (Y) .
```

は例1のpによって次のように展開が可能となる。

```
q (a) :- !, write (a) , ( p (X) = p (a) ,
         Write (a1) ,nl, !, write (a2) ;
          p (X) =p (b) ,q, write (b) ) .
q (b) :- write (b) , ( p (Y) = p (a) ,
         write (a1) ,nl, !, write (a2) ;
         p (Y) = p (b) ,q, write (b) ) ,
         write (Y) .
```

(2) カットの自動挿入における利用

バックトラックしたとき再び成功することがないことが分かっているとき、適切な位置にカット記号を挿入しておけば不必要なredoを避けることができる。 上の図式に関連して次のようなカットの自動挿入が可能となる。

(i) 上の図式において条件がみたされているとき：

Γi => Γi, !

とする。 すなわち、次の最適化図式が成立する

```
a1 :- Γ1.
  ⋮
ai :- Γi, p (A) , Δi.
  ⋮
an :- Γn.
p1 (A1) :- Θ1.
  ⋮
pm (Am) :- Θm.
```

```
a1 :- Γ1.
  ⋮
ai :- Γi, !, ( p (A) = p1 (A1') , Θ1' ;
      ・・・ ; p (A) = pm (Am') , Θm') , Δi.
  ⋮
an :- Γn.
p1 (A1) :- Θ1.
  ⋮
pm (Am) :- Θm.
```

(ii) さらにpがc－決定的であるとき：

Δi => !, Δi

とする。 すなわち、次の最適化図式が成立する

```
a1 :- Γ1.
  ⋮
ai :- Γi, p (A) , Δi.
  ⋮
an :- Γn.
p1 (A1) :- Θ1.
  ⋮
pm (Am) :- Θm.
```

```
a1 :- Γ1.
  ⋮
ai :- Γi, !, ( p (A) = p1 (A1') , Θ1' ;
      ・・・ ; p (A) = pm (Am') , Θm') ,
      !, Δi.
  ⋮
an :- Γn.
p1 (A1) :- Θ1.
  ⋮
pm (Am) :- Θm.
```

以上の最適化図式の妥当性のチェックは現在のPrologのオペレーショナルな意味（インタプリタ）の下に確認している。 さらに形式的な意味論の下での同値性の証明は今後の研究をまたねばならない。

4. あとがき c－決定性は日常のPrologプログラミングにおいてもよく現れ、そのチェックも容易に行える決定的述語の一つのクラスを定義している。一般に、述語の決定性は単にそれがPrologのような非決定性プログラム言語の最適化（改善）に有効になるばかりでなく、効率の良いプログラムを書くときの一つの指針ともなるものである。

本研究の一部は第5世代コンピュータ・プロジェクトの一環として行われたものである。

日頃御指導、御鞭撻いただく北川敏男会長に感謝いたします。

参考文献

［1］沢村・竹島・加藤：PROLOGソースレベル・オプティマイザ―最適化手法のカタログ―, Prolog Conf. '84.

［2］竹島・沢村・加藤：Prologソースレベル・オプティマイザ―インライン展開を中心として―, 本大会予稿.

4P-6 PROLOGソースレベル・オプティマイザ —インライン展開を中心として—

竹島 卓・沢村 一・加藤和彦
(富士通国際研)

1. はじめに

Prologプログラムをソースレベルで最適化（改良）するPrologソースレベル・オプティマイザを試作した〔1, 2〕。Prologのような論理型言語の最適化に対する研究は未だ少ないので，我々がPrologプログラムをソースレベルで最適化するに当たって考察した事をまず論じ，ついで具体的方針としてのインライン展開を中心とした最適化について述べる。

2. Prologの非論理性に起因する最適化の困難性

Prologプログラムを最適化しようとするとき，Prologの次のような実行メカニズム，言語要素は論理的な最適化をかなり制限する原因となっている。

① Prolog特有の実行メカニズム： 上→下，左→右，バックトラック，とくにcutによるバックトラック制御：これらは論理式レベルでの自由な最適化を阻害する。

② 組み込みのevaluable 述語： side effectを持つ述語（入出力述語），メタ述語，カット，否定，repeat（トートロジー），etc.；これらはオブジェクトとメタ言語をともに考慮に入れた最適化を必要とする。

3. 最適性の判断規準

最適化の有用性の尺度，すなわち何を規準にして最適化されたかということを明確にしておかねばならないことは言うまでもない。次の三点から考察する。

(1)外的規準（処理方式，環境） (2)内的規準（表現）
(3) 計算方式

最適化方法といわゆる複雑性の研究は目的とするところはよく似ている。しかしながら，複雑性の研究では(1), (3)は問題とならない。プログラムの最適化問題ではオーダよりむしろ，係数の改良に主眼が置かれるので，(2)のみでなく(1), (3)も最適化を考えるときの考察対象とされるべきであろう。

4. プログラム変換論とプログラムの最適化

狭い意味においては，プログラムの変換論とプログラムの最適化の目的は同じと考えられるが，広義には，プログラムの変換論とは一つのプログラミングparadigmであって，stepwise refinement (Dijkstra,Wirth) の概念にこれまでの伝統的な最適化技術を結びつけたものを意味することが多い。そして，このparadigmの下では，我々は非効率的であるかもしれないが，明確で正しいプログラムを書き，それから，それを同値性を保存する変換を経て，明確さには欠けるかもしれないがより効率のよいプログラムへと変換する。

以下で述べられるProlog言語の最適化はPrologのプログラミングの規範を示すことでもなく，またeurekaを多用するdrastic な変換でもない。それはどちらかと言うと伝統的な従来言語の最適化の精神に通じるものと言える。
従って，プログラムの変換論とは明瞭に立場が異なっていることを強調しておかなければならない。

5. 最適化の方針（インライン展開を中心として）

我々のPrologの最適化問題に対する取組方は実用及び経験主義的である（少なくとも演えき的ではない）。

従来言語の最適化技法の中でインライン展開（サブルーチン展開）は最も重要な最適化法として知られている。その大きな目的は， ①起動機構（サブルーチンの呼出機構）の解消，および ②他の最適化技法の適用可能範囲の拡大，にあるとされる。
一方，Prolog言語に対してのインライン展開はつぎのような特徴をもっている。

（i）Prologは述語（手続き）の列からのみ書かれる言語であって，すべての実行文が展開の対象となってしまう。

（ii） Prologではそもそも呼び出し機構は述語定義の探索とユニフィケーションの二つからなり，その呼び出し機構を完全に解消することは一般に不可能である。すなわち，述語の呼び出し機構を呼び側の述語とその述語定義の単一化可能性にすりかえる必要がある。

これらはProlog特有の言語形式，計算方式に依っている。（i）はPrologの最適化を考えるさいインライン展開がその出発点となるべきことを暗示するし，（ii）は起動機構を新たな表現によりプログラムテキストの中に表現することを要求する。

実際，最も素朴な場合のゴールGの展開形式はつぎのようになる。

$$\begin{array}{lll} & H1 & :-\Gamma 1. \\ & H2 & :-\Gamma 2. \\ & \vdots & \\ G & Hn & :-\Gamma n. \end{array}$$

$G=H1',\ \Gamma 1';\ \cdots\cdots;\ G=Hn',\ \Gamma n'.$

ここで，H'はHの renaming term である．呼出履歴の一部はG＝H'として保存する．

このようなインライン展開によって変容させられたプログラムに対していく種類かの最適化手法が適用されることになるわけであるが（展開して式を長く保っておくと他の最適化を受けさせるのにいつも都合がよいというわけではないが），適用の順序が問題になり，個々の最適化手法が有効であってもその順序を誤るとせっかくのインライン展開の効果は半減してしまうことになりかねない．

オプティマイザに関する研究の現状からいって，2節で論じた最適性に関する判断規準すべてに適合する最適化システムを構成することは不可能に近い．ここでは理想的な最適化システムの実現に向けて，first stepとしての方針を取り，つぎのような点に留意してシステムを試作した．

（a）展開レベル

現在我々のいく種類かの最適化手法を総合的に見ると，インライン展開は帰納的なプログラムの場合1レベルの展開で十分であると考えられる．それ以上展開しても他の最適化手法によってdrastic にプログラムが最適化されるという期待は薄いようである．また展開のし過ぎはメモリーの過大な消費による負荷が高くなる．

（b） 時間 vs． 空間効率

現段階の研究の現状からいって，時間効率にのみ焦点を合わせた最適化システムの構成とし，空間効率の評価はシステムの使用経験を踏まえて考察していくことにする．

（c）interactive vs．automatic （システム構成に当たって）

Prologオプティマイザへの入力としてfullなPrologのソースプログラムを対象とするために，適時最適化のための指示を使用者が与えるという方式にする．

5．Prologソースレベル・オプティマイザの概要

Prologソースレベル・オプティマイザはつぎの3つの主要機能からなる．

① 最適化に必要な情報をプログラムテキストから抽出する機能

② 各種最適化手法に対応する機能

③ 入出力機能

オプティマイザの全体の処理はユーザからの指示に従って，直線，終端再帰，一般の帰納的プログラムの各インライン展開 ~~（3．2節参照）~~ の前後に各種の個別の最適化手法（3．3節参照）を適用する形で進行する ~~（図2）~~ ．

直線プログラムのインライン展開以外のインライン展開では，そこでさらに最適化を進めるか，終了させるかの判断を人間にもとめるようにしている．

ここで各種の最適化手法とはつぎのものである．

i．ユニフィケーションの部分実行 等式の形をしたゴールの計算を部分的に進行させる．

ii．ゴールの簡単化 選言中あるいは連言中の重複ゴールの除去，冗長なtrue，fail の除去，不実行部分の除去，共通ゴールのくくりだし．

iii．変数除去 冗長変数除去，等式代入．

iv．ゴールの統合化 複数の等式ゴールを統合する．

v．節の分解 選言をもつ節を分解する．

これらの最適化手法は無条件で適用できるとは限らず，所望の条件をみたしている必要がある． 基本的な条件はプログラムの形式（直線，終端再帰，一般）に依存するものでありこの形式の認識が必要である． とくに重要な条件はプログラムの決定性である．決定性情報によらない最適化は極く小さなプログラムのクラスにかぎられる． 決定性検出とその利用については別に本大会で詳述している．

7．あとがき

主として経験的かつ直観的にPrologプログラムのソースレベルでの最適化手法について検討した結果，多くの有効な最適化手法が見いだされた． それらは従来言語の最適化手法と比べて論理的には複雑である． 我々はPrologプログラムの変換論とは異なる立場から検討してきたが，各種最適化手法が同値性を保証していることの形式的な証明については今後の課題としてまだ触れていない．

我々のPrologプログラムのインライン展開の方法とBurstallらのプログラムの変換論におけるいくつかの概念との間にはいくつかの表面的な類似点が見いだされることについて触れておかねばならない． 例えば，プログラム変換論におけるunfolding はその行為だけではインライン展開と同じであるが，unfoldingは後に abstraction によってfolding されることを意図して行われるのでインライン展開の主要目的①，②とは明らかに異なる．またfolding は我々の最適化手法では等式ゴール列の統合化（3．5節）といくらか似ているが，我々の方法では現在のところabstraction という発見的手法を用いることはしていない．

他方，インライン展開もunfolding も形式面からみると共に計算を部分的に進行させているわけであるから，partial evaluation の異なる形態であるとも見なせる．

本研究の一部は第5世代コンピュータプロジェクトの一環として行われた．

参考文献

〔1〕沢村・竹島・加藤：PROLOGソースレベル・オプティマイザー最適化手法のカタログ―，Prolog Conf．'84．

〔2〕沢村・竹島・加藤：Prologソースレベル・オプティマイザー決定性の検出とその利用―，本大会予稿．

PROLOGインタプリタの構成

服部 彰　丹羽 雅司　梅本 剛　木村 康則　岸本 光弘
（富士通株式会社）

1．はじめに

ＬＩＳＰマシンＡＬＰＨＡ〔3〕の上に，ＰＲＯＬＯＧマシンアーキテクチャの研究と高性能ＰＲＯＬＯＧ処理環境の実現を目的として，ＰＲＯＬＯＧインタプリタシステムを開発したので報告する。

2．開発のねらい

このＰＲＯＬＯＧインタプリタシステムの主な開発のねらいは次の通りである。

（1）ＰＲＯＬＯＧの高速処理（高速インタプリタ）

ＰＲＯＬＯＧの実行に適した内部表現やデータ型を導入し，インタプリタの実行部やガーベジ・コレクタをファームウェア化することにより，高性能インタプリタを実現すること。特に，4項で述べる各種の高速化技法を用いて，実用プログラム開発に必要な高速性を実現すること。

（2）ＰＲＯＬＯＧマシンアーキテクチャの評価

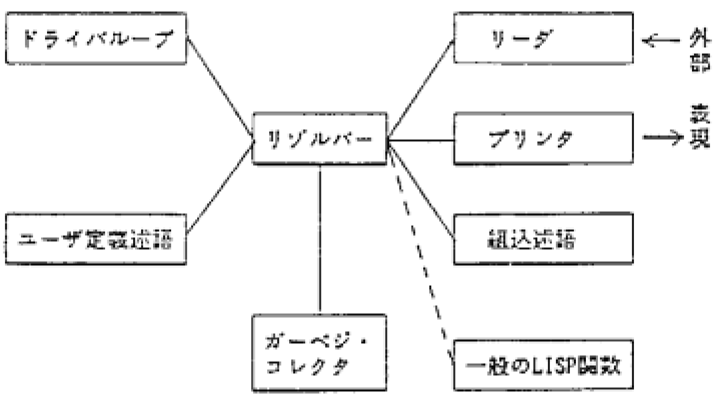
ＰＲＯＬＯＧマシンアーキテクチャ（実行制御，メモリ制御等）に関する基礎データを収集，評価すること。

（3）ＬＩＳＰシステムとの最適インタフェース

ＰＲＯＬＯＧの世界からＬＩＳＰ機能が述語と同様に使えること。

（4）ＤＥＣ−10ＰＲＯＬＯＧ互換

標準的ＰＲＯＬＯＧであるＤＥＣ−10ＰＲＯＬＯＧとデバッグ機能も含めて構文，機能の互換性があり，実数もサポートすること。



図．1　ＰＲＯＬＯＧ処理系のシステム構成

3．処理系のシステム構成

図．1にＰＲＯＬＯＧ処理系のシステム構成を示す。リーダ，プリンタ及び組込述語の大部分はＬＩＳＰで記述されている。ＰＲＯＬＯＧの実行部（リゾルバーと呼ぶ）やガーベジ・コレクタはファームウェア化されている。ＰＲＯＬＯＧドライバループ及びユーザ定義述語は図．2のようなベクトル型に内部表現されている。

（1）ＰＲＯＬＯＧの起動

ＬＩＳＰシステムから，ＰＲＯＬＯＧのドライバループを初期ゴールとしてリゾルバーに渡すことにより，ＰＲＯＬＯＧシステムがスタートする。

（2）入出力系

リーダはバックトラックを含む変形ＳＬＲ（k）パーザでＤＥＣ−10ＰＲＯＬＯＧの全構文をサポートしており，端末やファイルから外部表現を読込んで，高速処理に適した内部表現へ変換する。

プリンタは内部表現を外部表現へ変換して端末やファイルへ出力する。

（3）リゾルバー

リゾルバーはユニフィケーション，組込述語コール，リターン，バックトラック等の機能を持ち，内部表現に変換されたドライバループや入力ゴールを組込述語やユーザ定義述語を用いてリダクション処理するマイクロルーチン（2Ｋ語）である。

（4）ＬＩＳＰシステムとのインタフェース

リゾルバーからＬＩＳＰ関数がコールでき，関数ＵＮＩＦＹを用意している。それにより，ＬＩＳＰ関数との論理変数を介した通信が高速に行える。

（5）ガーベジ・コレクタ

ガーベジコレクタはファームウェア化（約2Ｋ語）されており，ＬＩＳＰ，ＰＲＯＬＯＧ両システムに共通である。ＬＩＳＰオブジェクトの他にグローバルスタック，トレイルスタックの回収，コンパクション機能を持つ。

（6）スタック

ＤＥＣ−10ＰＲＯＬＯＧと同様にローカル，グローバル，トレイルの3種のスタックを用い，構造データの表現はStructure-Sharing法〔1〕を用いている。ローカルスタックは

使用頻度が高く，又TRO技法も使えるのでCPU内蔵のハードウェアスタック（実容量8K語，仮想容量64K語）に割付けている。グローバル，トレイル両スタックは大容量のため主記憶に割付けている。

（7）述語及びクローズの内部表現

述語は述語名シンボルの属性リストに引数個数インデックスして格納している。ユーザ定義述語の各クローズは図2のようにベクトルとして内部表現している。一方，大部分の組込述語はLISPのSUBR関数となっている。

4．高速化技法

このPROLOGインタプリタは，次のような方法を用いて高速化をはかっている。

（1）ローカルスタックのハードウェアスタック割付

（2）CONSをLISPと同様のリスト表現

他のスケルトンはベクトル型の内部表現を用いているが，CONSは特に使用量が多いのでLISPと同様のリスト表現を用いてアクセスの高速化とメモリ使用量の効率化をはかっている。

（3）定数CONS，定数スケルトンの導入

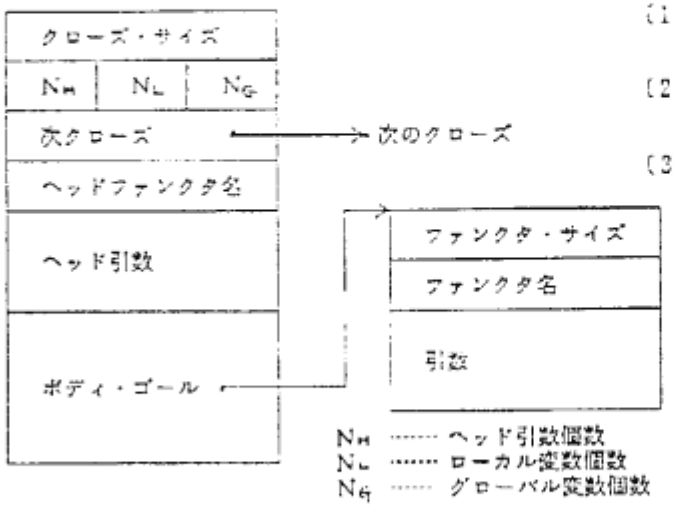
変数を含まない構造体専用のデータ型を導入し，不要なモレキュール作成を省略している。

（4）トレイルスタックへの不要なプッシュの省略

ローカル，グローバル両スタックのバックトラック点をレジスタ上に保持することにより，バックトラック点以降の変数への束縛時のトレイルプッシュを省略している。

（5）Deterministic 述語の高機能リターン処理

Deterministic 述語が終端ゴールの場合，次に実行すべきゴールの親フレームとバックトラックフレームの内，新しい方のフレームの直下に次ゴール用のフレームを作成する。終端ゴールでない場合には，親フレームを流用する。これにより，終端ゴールでのローカルスタック長の短縮と非終端ゴールでの高速化をはかっている。



図．2　クローズ及びファンクタの内部表現

5．性能

本PROLOGインタプリタによるD.H.D.Warrenの2種のベンチマークプログラムの実行時間を下表に示す。

NATIVE REVERSE-30	35.9 msec
QUICK SORT-50	45.6 msec

この性能はDEC-10PROLOGインタプリタ（DEC2060）に比べ，5倍以上高速である。

尚，4項で述べたCONSのリスト表現及び定数CONS，定数スケルトン型の導入は上記プログラムの場合，6～7%高速化に寄与している。4項（4），（5）の効果は更に大きな応用プログラムでの評価が必要である。

6．おわりに

LISPマシンALPHA上に高速PROLOGインタプリタシステムを開発した。今後，各種応用プログラムでの推論マシンアーキテクチャに関する基礎データの収集，評価を行う予定である。

尚，本研究は第5世代コンピュータプロジェクトの一環としてICOTの委託で行ったものである。

謝辞

日頃御指導頂く渕様部長，林室長並びに本研究に御協力頂いた関係諸兄に深謝致します。

参考文献

〔1〕D.H.D.Warren : Implementing Prolog.
D.A.I.Research Report no.39-40 (1977) .

〔2〕G.M.Roberts : An Implementation of PROLOG.
A thesis of Waterloo University (1977) .

〔3〕服部，林，秋元，丹羽，品川，森木，木村，佐藤
：ALPHA-高性能LISPマシン，情報処理学会
記号処理研究会資料23-1 (1983) .

PROLOGの並列処理システム

坂敷 昇弘　　岩藤 健　　増沢 秀徳
(富士通株式会社)

1. はじめに

我々はプロログを並列に処理する方式を研究している。前回の全国大会では、プロログを節単位に処理するモデルを提案した[1]。その後、このモデルに基づいた並列処理システムを試作した。

今回は、試作した並列処理システムについて、特徴・構成・結果を報告する。

2. モデルの特徴

モデルの特徴については文献[1]に詳述してあるので、ここでは列挙するに留める。

- OR並列処理方式を採用する。AND関係は逐次に処理する。
- リテラルとユニフィケーションが成功する節単位で、UNIFY プロセスを作る。
- UNIFY プロセスに解を保持する機構を付ける。

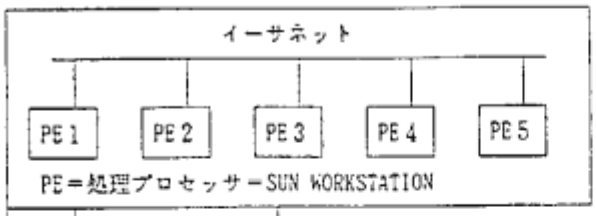


図1　ハードウェア構成

3. システムの構成

今回試作したシステムは、プロセッサ要素としてパーソナルコンピュータを複数台使い、互いに他の総てのプロセッサとデータを通信できる形態とした。

具体的には、SUNワークステーション5台をイーサネットで結合した。（図1）

各SUNには、同一の並列処理系を搭載した。処理系は互いにデータをやりとりしながら、全体としてプロログを処理する様にした。

4. 並列処理系の構成

ここでは、各プロセッサ上に搭載した並列処理系について述べる。図2に示す処理系の構成を参照しつつ、動作の概略を説明する。

(1)処理系の基本動作

①質問メッセージに対する処理

メッセージプールに質問メッセージ？-A.B. が在るとする。

ユニフィケーション制御部は、？-A.B. を取り出し、その最も左のリテラル、すなわち Aについて推論ベースを使用してユニフィケーションを行う。

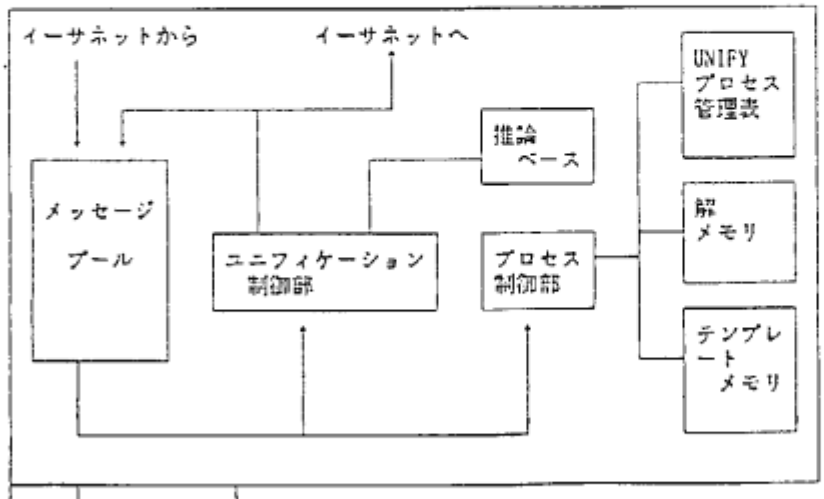


図2　処理系の構成

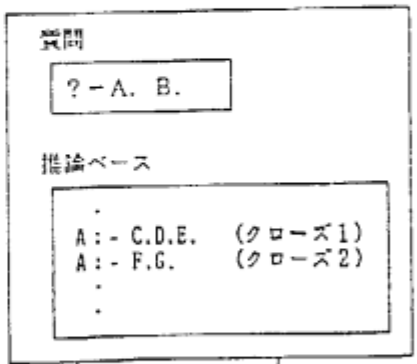


図3　質問と推論ベース

・一つ以上のクローズとユニファイに成功した場合：

質問メッセージ（?- A.B.）を制御する為の情報をUNIFYプロセスとしてUNIFYプロセス管理表に登録し、またそのテンプレート（A.B.）をテンプレートメモリに登録する。

例えば、クローズ1でユニファイが成功したとすると、そのボディを質問メッセージ?-C.D.E.として、メッセージプールへ、或いはイーサネットを介して他のプロセッサへ転送する。

・一つのクローズも成功しなかった場合：

親のUNIFY プロセスにfailメッセージを転送する。

③解メッセージに対する処理

プロセス制御部はその解メッセージから対応するUNIFY プロセスを見いだし、そのUNIFY プロセスの制御情報から得られるテンプレート（A.B.）を取り出す。

そして、その解をテンプレートに反映させて作成したB に関して、ユニフィケーション制御部は①と同様の処理を行う。

④メッセージの選択

メッセージプールからのメッセージの取り出し順は、種類によらず、受け付け順に取り出すことにした。

(2)処理の分配

他のプロセッサに依頼する仕事の単位を大きくする事を目指し、UNIFY プロセスを次の戦略で分配した。

・質問が組み込み述語かファクトなら、自プロセッサのメッセージプールに入れる。

・質問がボディのあるルールなら、自分のプールを見て、ルールがあったら、別のプロセッサに質問を送る。自分のプールにルールが無かったら、そのプールに質問を入れる。

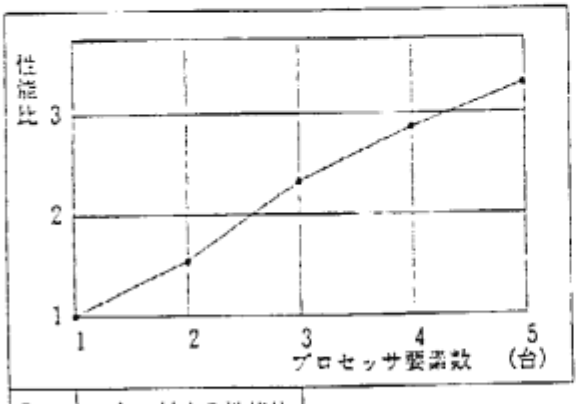


図4　1台に対する性能比

5．実行結果

試作したシステムで、プロセッサ台数の効果を調べた。例としてタクィーンのプログラムを実行した時の結果を図4に示す。図中の値は、1台のプロセッサの性能に対する複数プロセッサを使用した場合の性能比である。

グラフより以下の事が言える。

・3台のプロセッサで実行したら、1台のときの2倍以上（3×0.77）、5台では3倍以上（5×0.65）の性能向上が見られる。

今回の試作に使った処理系の性能と通信路の性能はどちらも充分なものではなかった。処理系のユニフィケーション当りの時間と、1回の通信に要する時間の比は3対1程度と近接していた。しかし、5台の場合の実行結果では、ユニフィケーションが約8000回、通信が約 100回であり、全処理時間に占める総通信時間は無視できる値であった。

しかし、グラフを見ると、台数効果が十分には出ていない。その原因は、処理の分配方式が適切でなく、プロセッサの稼働率に偏りがある為と考えている。

6．おわりに

試作したPROLOGの並列処理システムについて述べた。

節単位処理モデルに基づく並列処理システムを複数のパーソナルコンピュータを使用して試作した。その試作実験により、並列処理効果を確認した。

今回の実験においては、処理系が初期版であり、通信路もイーサネットという様に、各要素の性能はあまり高くない条件での結果であった。処理系に改良が加えられ、そして更に高速の通信路が使用される場合の効果については今後の調査が必要である。

なお本研究は第5世代計算機プロジェクトの一環として、ICOTの委託で行ったものである。

ここで、日頃から様々の指導を賜っている、渕一博部長と相馬室長に感謝の言葉を捧げたい。

参考文献

(1)佐藤ほか「節単位処理モデルの提案」情報処理第28全国大会5H-8.P.1141

並列環境における Concurrent Prolog実現法

尾内理紀夫　　麻生盛敏
（財）新世代コンピュータ技術開発機構

1．はじめに

我々は，リダクション方式並列推論マシン[ICOT 84] の研究開発を行う過程において，ソフトウェアシミュレータを開発し，その上で，Prologプログラムを走行させ，各種データを収集している。このシミュレータ上では，Prologプログラムだけでなく，Concurrent Prolog (CP) [Shap 83]プログラムをも実行できる。これは，いわば並列環境でのCPの一種の処理系を実現している訳で，これについて述べる。

2．並列環境

並列環境と，本稿において前提とする事項について述べる。

①プロセス：CPのClauseは，次のような形をしている。

head :- guard | body.

guard,body はリテラル列である。プロセスは，（結合情報＋guard リテラル列＋bodyリテラル列） からなる。また，Parallel And Operator で結合されたリテラル列がreduction 可能となった時は，各々のリテラルを独立したプロセスとして生起させる。(reduction可能かどうかは，Parallel And Operator, Sequential And Operator,Commit Operator により指示される。) そして，プロセスを単位として処理ユニット（並列環境だから処理ユニットは複数ある）に割りつける。

本並列環境では，reduction 可能なリテラルのみをeager copyしてreduction する。例えば，bodyリテラル列が，

b1&b2&b3　　('&'はSequential And Operator)

のとき，b1のみをcopyし，reductionする。その結果，子プロセスが生成されれば，子プロセスから親プロセス b1&b2&b3 へ，backポインタを張る。これは，子プロセスから親プロセスへ結合情報を返す時に使用する。以後，reduction 可能なguard リテラル，bodyリテラルをreducible subgoal と呼ぶことにする。

プロセスの持つ各種tag 内にC-tag(Commit tag) がある。プロセス生成時，C-tag はOFF であり，最初にguard が成功した子プロセスがこれをONにしてcommitする。guard が成功した子プロセスはcommitする前に親プロセスのC-tag をチェックし既にONならば，消滅する。(C-tagがONになった直後に他の兄弟プロセスを殺しにはいかない。)

②チャネル：

規定1：clause内のParallel And Operator で区切られたリテラル間の共有変数はチャネルである。

(例) go:-p1(X),p2(X),p3(X).
(','はParallel And Operator)

規定2：clause内のリテラル間の共有変数のうちの少なくとも一つにread only 変数がある時，これら共有変数はチャネルである。

(例) go:-p1(X),c1(X?),c2(X?).

X はチャネルであり，X?を特にreadチャネルと呼ぶ。

このように，チャネルとして使用される変数（チャネルと呼ぶ）と，それ以外の変数（変数と呼ぶ）とを区別する。また，チャネルの性質は実行時に動的にinherit する。たとえば，clause側引数内変数は，reducible subgoal 側のチャネルとunify されるとチャネルとなる。（３頁の例を参照）

③Message Board：チャネル用のメモリとしてMessage Board (MB)を設ける[Onai 83] 。producerプロセスとconsumerプロセスとは，MBを介してメッセージをやりとりする。MB内の一つのチャネル用セルの論理構成は次のとおり

チャネルの値	suspend プロセス リスト

suspend プロセスリストは，このチャネルが原因でsuspendしたプロセスが登録され，値が書き込まれた時，その値がsuspend プロセスに送られる。

④OR並列実行：reducible subgoal とclauseとのunify はOR並列に実行される。

⑤AND 並列実行： Parallel And Operatorで区切られたリテラルはそれぞれ独立したプロセスとしてAnd 並列に実行する。

3．Concurrent Prolog 実現法

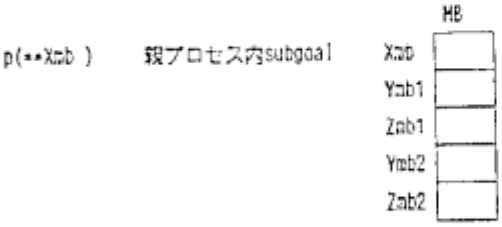
チャネルは，論理的に二語の "チャネル情報" でデータ表現される。第一語は，MB内該当チャネルあるいは親プロセスのチャネル情報へのポインタであり，第二語は，local データ領域である。guard が成功し，commitした時に，このlocal データをMB内該当チャネルセルあるいは親プロセスのチャネル情報内local データ領域に書き込む。以下，例を用いて説明する。

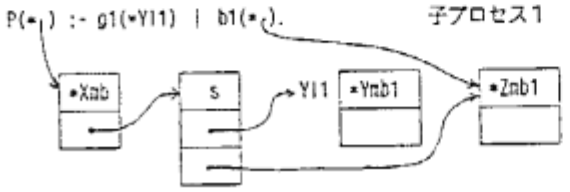
(例 3.1)　p(*,)　　reducible subgoal 側

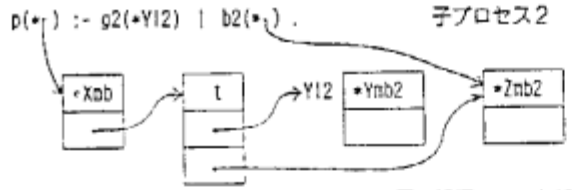
*Xmbは，MB内セルへのポインタ
local データ領域 （以後略して p(**Xmb)）

```
p(s(Y,Z)) :- g1(Y)  | b1(Z).     OR関係にあるclause側
p(t(Y,Z)) :- g2(Y)  | b2(Z).     （まだY,Z はチャネルで
                                   はない）
```

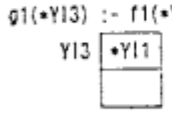
unify 時にチャネルの性質はinherit するので，各Y,Z は，チャネルとなり，OR clause ごとにHB内にセルが確保される。一方，g1,g2 のY に関しては，並行関係にあるプロセスが存在しないので，このg1,g2 のY は，子プロセスレベルではチャネルとしては使用されない。よって，このg1,g2 のY のために，HB内に新たにセルは確保せず，headチャネルとチャネル情報を共有する。（本並列環境では，reduction 時には引数はeager copyされる） reduction 結果は，次のとおり。

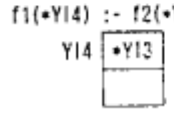




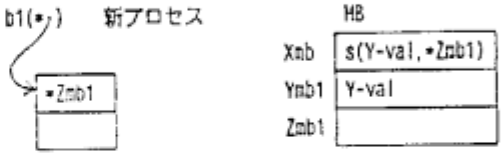


これら，同一の親を持つプロセスでも，同一処理ユニットに存在するとはかぎらないから，g1 ,g2がそれぞれ成功すると，これらのプロセスは，親プロセスのC-tag チェックを行う。そしてC-tag を先にCNにしたプロセスのみがcommitし，local データをHB内のXnb に書き込む。以下のようにguard g1がネストする場合は，f1,f2 には，やはり並行関係にあるプロセスはないので，チャネル情報をheadチャネルと共有する。各local データの内容は，プロセスが成功した時，結合情報として親プロセスのlocal データ領域に返されていき，g1(*Y11)が成功し，commitする時に，local データがHB内のXnb ,Ynbに書き込まれる。ネストの先で，プロセスがsuspend した時は，チャネル情報を逆にたどり，HB内チャネルセルにアクセスし，値が既に書き込まれていれば，取り込み，まだならばsuspend プロセスリストに登録する。

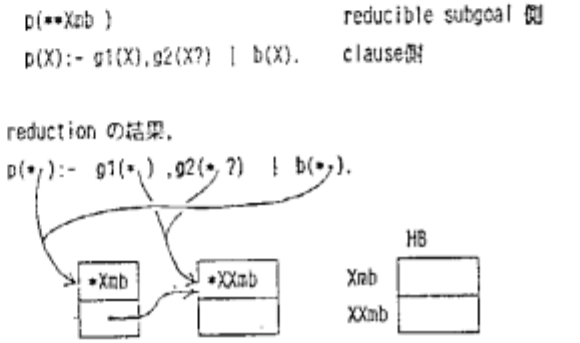




本例で，例えば，子プロセス1のguard が先に成功し，その時Y11 のlocal データ値がY-val とすると，commitした後はHB への書き込みが走り，次のようになる。



（例 3.2） guard 内のあるチャネルに関し並行プロセスが存在し，各並行プロセス内チャネルのうちに，少なくとも一つreadチャネルでないチャネルがある時は，そのチャネルのために新たにHB内にセルを確保する。



となる。g1とg2は独立したプロセスとなり，HB内のセルXXnbを通じてメッセージをやりとりする。そして，g1,g2 が共に成功し，commitした時にXXnbの値をXnb に書き込む。

なお複数producerプロセスが存在する時，各プロセスは，あるチャネルへの書き込みをする。この時，該チャネルのHB内チャネルセルは，一つであるから，同一値の書き込みは許されるが，異なる値の書き込みは，failになる。

4．おわりに

チャネル用メモリ(Hessage Board) およびreducible subgoal のeager copyによる，並列環境でのCPの実現法について述べた。これはリダクション方式並列推論マシンのソフトウェアシミュレータに組込まれており，シミュレーション結果については，機会を改めて報告する。最後に日頃御指導いただく村上国男第一研究室長，御討論いただいたICOT研究員の皆様に深謝する。

［参考文献］

[Shap 83] E.Y.Shapiro : A Subset of Concurrent Prolog and Its Interpreter , ICOT Technical Report TR-003 ,1983

[ICOT 84] 電子計算機基礎技術開発成果報告書 推論サブシステム編 （財）新世代コンピュータ技術開発機構，1984

[Onai 83] 尾内，麻生：並列推論マシンにおけるGuard と入力annotationの制御機構，昭58後期情報処理全国大会

ＣＡＤ向けＰｒｏｌｏｇインタープリタ
ＣＡＤＬＯＧ

光本　圭子　　森　　啓　　藤田　友之　　後藤　敏
（日本電気（株）　Ｃ＆Ｃシステム研究所）

１．　はじめに

最近、人工知能、あるいは、知識工学の研究が活発になり、そのプログラム言語として、Prologが注目され、実用化に向けて、改良、拡張の検討が行なわれている[1,2]。ここで、大規模問題を取り扱うシステムを、Prologですべて表現することを考えると、現在のハードウェア環境や、Prologの処理系では、実行速度、メモリ使用量の点から見て問題がある。著者らは、FORTRAN 言語との結合機能をPrologに付加することによって、実行時間の短縮、蓄積されたソフトウェアの有効な活用を可能にした。

本稿では、Prologの実用化の１つとして、FORTRAN 言語とのリンク機能を持つPrologインタープリタ ＣＡＤＬＯＧ について報告する。

２．　ＣＡＤＬＯＧの特長

ＣＡＤＬＯＧの構文規則は、DEC-10版 Prolog[3]に類似している。文字は、大文字、数字、記号の３種であり、変数は '$'を付加して記述する。

２．１　FORTRAN 言語とのリンク機能

手続き的処理が多い問題に対する記述しやすさや、プログラムの実行効率の面から考えると、Prologに手続き型言語(FORTRAN) との結合機能を付加することは、有効な手段である。FORTRAN で書かれたプログラムとのリンクは、外部ファンクション述語と呼ばれる述語によって対応づけられる。Prologの中で、この述語は頭文字に '$'を付加して他の述語と区別している。FORTRAN 言語では、Prologのような引数の評価ではなく、必ず入出力を規定しなければならない。したがって、ＣＡＤＬＯＧでは，外部ファンクション述語の引数の入出力の整合がとれなければ、FAILとして、バックトラックが生じる。

例えば、FUNCという節を下記のように表現する。

```
FUNC($X,$Y):-$PROC1($X,$Y).   ····· サブルーチンPROC1(X,Ȳ)に対応する外部ファンクション述語
FUNC($X,$Y):-$PROC2($X,$Y).   ····· サブルーチンPROC2(X̄,Y)に対応する外部ファンクション述語
FUNC($X,$Y):-$PROC3($X,$Y).   ····· サブルーチンPROC3(X,Y)に対応する外部ファンクション述語
```

（注）　X は入力変数、X̄ は出力変数を表わす。

このFUNC述語を用いて、SUB_GOAL1 という節を次のように定義する。

```
SUB_GOAL1($X,$Y,$Z):-FUNC($X,$Y),FUNC($Z,$Y).
```

この節において、第１引数が定数、第２、３引数が変数とユニファイされた例を示す（図１）。

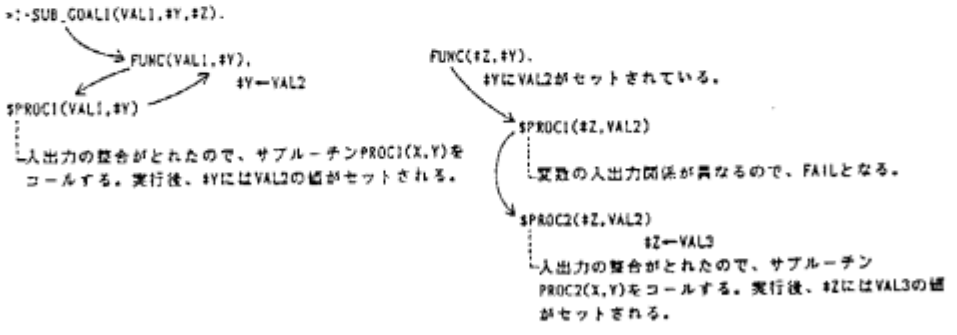


図１　外部ファンクション述語に対応するFORTRAN サブルーチンがコールされる様子

２・２　FORTRAN 言語によるシステムのデータ構造内のデータの整合性

FORTRAN 言語とのリンク機能を持っているため、この言語で構築されているシステム内のデータ構造を使用することができる。これによって、現在Prologでは実現不可能な大規模データ処理が効率よく処理でき、実行時間も短縮される。FORTRAN で構築されているシステム内のデータ構造を用いる場合、このデータ構造を作っている変数と、Prologプログラムの中に表われる変数とで、システム全体の状態を表現することになる。そこで、Prologのバックトラックが生じた時に、Prologプログラムの変数に対してのみユニファイされる前の状態に戻す処理を行なうと、Prologの変数が表現する状態と、データ構造内のデータが表現する状態とにくい違いを生ずる。したがって、プログラマは、どの外部ファンクション述語がデータ構造内のデータを更新するか常に意識し、バックトラックが生じると、どのような処理を行なわなければならないか考えて、プログラムを組む必要がある。これは、プログラマに負担をかけるばかりでなく、プログラムを読みにくくする。ＣＡＤＬＯＧでは、データを更新する外部ファンクション述語に対応するFORTRAN サブルーチンに、常に更新前のデータを保持する処理を組み込んでおき、バックトラックが生じてこの述語まで戻った時に、保持しているデータを用いて、データの改復処理を行なう機能を持っている。例を図２に示す。このように、FORTRAN で構築されたシステム内のデータ構造を用いる場合でも、ＣＡＤＬＯＧはデータの改復処理用サブルーチンを組み込むことにより、常にデータ構造内のデータの整合性を保つことができる。

```
UPP(1).
UPP(2).
>:-UPP(#X),RECON,BLOCKING(#X,#Y),$DELETE(4Y),$CONNECT(#X),$CONNECT(4Y),RECOFF,FAIL.
```

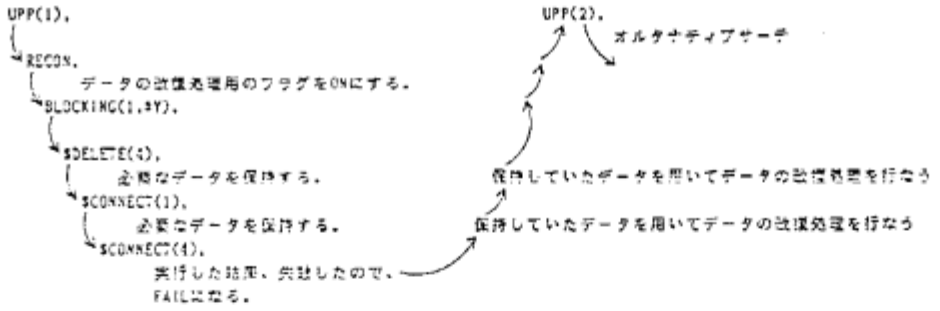


図２　データの改復処理機能の例

２．３　外部ファンクション述語の登録、変更、削除、表示

外部ファンクション述語を新たに登録したり、対応するFORTRAN サブルーチンに変更が生じたり、不必要な外部ファンクション述語を削除するために、ユーティリティが用意されている。このユーティリティを用いて、ユーザは、外部ファンクション述語の登録、変更、削除が容易にできる。

３．　おわりに

FORTRAN 言語とリンク機能を持つPrologインタープリタ　ＣＡＤＬＯＧ　について報告した。この機能により、大規模問題を取り扱うシステムでも、蓄積されているソフトウエアを有効に活用しながら、人工知能の手法を取り入れたシステムの構築を可能にする。現在、ＣＡＤＬＯＧは、NEC MS上でインプリメントされ、ＬＳＩ配線設計エキスパートシステム[4] を構築する言語として、使用している。

参考文献

[1] 中島，" Prolog/KR の概要" ，情処，記号処理，18-5，1982.

[2] 梅村，中嶋，小長谷，幅田，島津，横田，" 文字処理を導入したProlog:Shapeupについて" ，Proc. of the Logic Programming Conference，1983.

[3] W.F.Clocksin and C.S.Mellish，" Programming in Prolog " ，Springer-Verlag ，1981.

[4] 森，光本，藤田，後藤，" 配線エキスパートシステム" ，情処第28回全国大会，pp 1073-1074，1984.

Study of a Newspaper's Treatment of Proper Names

Satoru Ishii

Institute for New Generation Computer Technology

1. Introduction

People's names are important keywords in sentences, and care must be taken that they be correct. However, errors in names are difficult to detect. For this task, computerized name-checking system may be of great assistance in avoiding errors, and may reduce the proofreader's workload.[1]

With a view toward designing a computerized directory-consultation system, this study used one day's output of a typical Japanese newspaper to determine the following characteristics of the use of personal proper names:

(1) Style in which names appear
(2) Level of sentence understanding required
(3) Types of directories required.[2]

2. Sample Data

The morning and evening editions of the July 19, 1983 Asahi Shimbun constituted the study's sample data. All sentences, including headlines and tables, were analyzed. Advertisements, TV and radio listings and serials were ignored. Assumed names such as pen names and screen names were treated as conventional names.

The data contained a high volume of sports reporting, especially, stories about amateur and professional baseball; this bias has not been overlooked.

3. Style of Name Usage

We established four categories of style,[12][13] and tabulated names falling into each category. The categories and the number of appearences are as follows.

Names are used:	
with an organization and a post	79 times
with a post	227 times
with a title or an honorific	272 times
with none of the above	1293 times

The first category mainly included names of statesmen. The second category mainly included names of statesmen and sports figures. The third category included names of various persons falling into no paticular grouping. The last mainly included names of sports figures and entertainers.

There were 12 names that could not be assigned to any of these four categories; these were ignored.

4. Contextual Understanding Required

In a computerized consultation ststem, in the case of government officials for instance, three fields must be generated through contextual understanding: the individual's name, the organization he belongs to and the post he holds. We divided the contextual understanding required into six levels and counted names appearing in each.

The levels, from easy to difficult, have been tentatively established as follows. Figures in parentheses indicate the percentage of appearences in each level.

Understanding of BUNSETSU required(10%): In the BUNSETSU in which the name appears, the organization and post also appear. Therefore, understanding of the BUNSETSU is required. BUNSETSU is a concept unique to the Japanese language, and has no English equivalent.-Ed.

Understanding of text format required(43%): The text is based on a well-defined format, the understanding of which is required.[9] A typical example is a table of baseball scores.

Understanding of relationship to another appearence required(8%): The name appears in the same text in a more detailed style, and understanding the relationship between these appearences is required.

Understanding of context required(14%): The organization and the post appear in the text, therefore, the context must be understood.

Understanding of text domain required(24%): The text domain, for example, sports or science, is necessary.

Impossible(1%): The name forms part of another word. In such cases, it is impossible to distinguish the name.

Understanding of the BUNSETSU is sufficient for only 10%of the appearances. For 90%of the appearances, more in-depth understanding is required.

5. Effectiveness of Directories

We divided names into nine classes, selected appropriate directories for each class and determined the effectiveness of each. The results are shown in Table 1.

Names of government officials and foreign statesmen were effectively located. Names of professors and directors could be located; however, since such names were few, relative to the size of the corresponding directories, confirmation of names in these classes would be inefficient in a computerized system. Names of creative artists were difficult to locate. Names of sports figures were effectively located, but this depends on the sports involved. To locate the names of entertainers, an extensive directory was required.

Of the entire sample, about 75%of the appearences could be located using a number of directories whose entries listed approximately 200,000 persons.

6. Conclusion

Using one day's output of a large Japanese newspaper, we studied the usage patterns of personal proper names. This work represents the first stage in the design of a computerized directory-consultation system.

(1) We divided the names into four categories and found that names unaccompanied by titles appeared most frequently.
(2) For generating database queries, the understanding of BUNSETSU was sufficient for only 10%of the appearences.
(3) Using adequate directories with approximately 200,000 entries, about 75%of the appearences could be located.

Table 1. Effectiveness of Directories

A: Class	B: Appearances (%)	C: Directories Used	D: Entries (in 1000s)	E: Located	F: E/B (%)	G: E/D (%)	H: F*G
Government officials	248(13)	[3]	5	219	88	4	352
Foreign statesmen	61(3)	[3]	1	36	59	4	236
Professors	65(4)	[4]	90*	65	100	0.07	7
Directors	37(2)	[5]	30**	23	62	0.08	5
Creative artists	123(7)	[3]	6	36	28	0.6	17
Sports figures	1035(55)	[7][8][10][11][14]***	50	976	94	2	188
Entertainers	71(4)	[6]	9	57	80	0.6	48
Newspapermen	27(1)	---	--	0	0	---	---
Others	199(11)	---	--	0	0	---	---
Total	1871(100)		191	1412	75	0.7	53

* This figure represents only professors and assistant professors. Names of other university instructors were assumed not to have been listed.
** This figure represents only directors. Names of other company officials were assumed not to have been listed.
*** Only the number of KOUKOU YAKYUU(high school baseball) teams was found. Names of players were assumed to have been listed.

References

[1] Ishii, S.: "Study of Proofreading Techniques Used at a Japanese Newspaper", Proc. of the 28th National Conf. (II) 2H-7, pp.1205 ~1206, IPSJ (1984)
[2] 石井 暁："新聞における人名の使い方の調査"、TM-0062、新世代コンピュータ技術開発機構(1984)
[3] 堀山定編："朝日年鑑1983年版"、朝日新聞社 (1983)
[4] 大学野球記録刊行会編："1983全国大学野球年鑑"、鹿鳴社 (1982)
[5] ダイヤモンド社編："ダイヤモンド会社職員録（全上場会社版）1983年"、ダイヤモンド社 (1982)
[6] 著作権資料協会編："出演者名簿1983年版"、著作権資料協会 (1982)
[7] "棋士名鑑"、昭和57年版将棋年鑑、pp.335~ 381、日本将棋連盟 (1982)
[8] "棋士名鑑"、1982年版囲碁年鑑、pp.389~ 441、日本棋院 (1982)
[9] "記事のフォーム"、記者ハンドブック第4版第3刷、pp.327~ 338、共同通信社 (1982)
[10] "12球団新陣容"、朝日新聞1983年3月28日号、朝刊、pp.28 ~29、朝日新聞社 (1983)
[11] "昭和五十八年度九月場所東西幕内十両力士星取表"及び"昭和五十八年度九月場所東西幕下以下力士星取表"、相撲1983年10月号、pp.182~ 185、ベースボール・マガジン社 (1983)
[12] "人名、年齢の書き方"、記者ハンドブック第4版第3刷、pp.359~ 363、共同通信社 (1982)
[13] "人名等の書き方"、新版記事スタイルブック一新用字用語集一新版2刷、pp.443~ 452、時事通信社 (1983)
[14] 朝日新聞1983年7月17日号、朝刊、p.16、朝日新聞社 (1983)

知的プログラミング環境の実現へ：第５世代プロジェクトにおけるアプローチ

横井　俊夫
（財）新世代コンピュータ技術開発機構・研究所

１．（知的）プログラム言語

プログラム言語のプログラミング手法やプログラミング環境への関わり方は、その時代、時代で様々な形をとってきた。コンパイラの制作が自動プログラミングのすべての話題であった時代から、手法や環境は、言語と独立して論論できる、そうすべきことが強調されさえしたソフトウェア工学の時代へと移る。最近はもう少し言語を重視しようという風潮がみられるようになったが。そして、第５世代コンピュータは、プログラム言語を再び主役に引き立てる。

なぜ、主役の座を去るに至ったかの過去をたどる。

(1) プログラム言語の研究に大きな展開が無くなった。様々な言語の模索はCOBOL, FORTRAN, PL/I等の商用汎用言語に集約された。新しい要求事項も、それらの言語の部分的な手直しにとどまり、根底からおびやかすような事態は起らなかった。また、その傾向は、互換性維持の立場からの強い要求によって、一層増長されることになる。

(2) ソフトウェアの基本的枠組を規定するのは、プログラム言語だけでなくなった。オペレーティング・システム、プログラミング・システム、データ・ベース・システム等を大きく前提として考慮せねばならなくなった。

第５世代における復活は、この２つの事態に大きな変化が生じた、生じつつあることに基づく。

(1) 知的機能を実現するための土台としての記号処理機能、モジュール化の新しい仕組であるオブジェクト指向機能、これらは、従来の言語機能を大きく越え、新しい枠組を求める。

(2) 新しい枠組に立つ言語は、従来言語の機能をその一部に含み、新しい機能は高度なオペレーティング・システム、プログラミング・システムの構成を容易にし、データ・ベース機能が本来の仕組として組込まれており、ソフトウェアを総合的に規定する土台となる。

この新しい言語を方向づける基本的プログラミング・スタイルとして論理型言語を採用する。

２．（知的）ワーク・ステーション

新しい言語に則った高機能なワーク・ステーションは、プロジェクト全体の共通の研究・開発ツールであるが、それは、また、知的プログラミング機能を事実に即して明らかにする場，またシステムを具体的に検証する場として必須のものである。逐次型推論マシンの名のもとに、高機能パーソナル・コンピュータ：PSI(マシン・サイクル２００ns，主記憶１６M語）を開発し、現在、ソフトウェア・システム：SIMPOSを作成中である。オブジェクト指向機能を論理型言語上に融合したESP を使用し、新しいモジュール分割によるシステム開発を実験している。モジュール・ライブラリの知識ベース化、知的Help機能等、具体的事例に即しての知的プログラミング・システムの実現を計画して居り、また、ハード、ソフト、言語の改良、拡張を努力的に続け、プロジェクト全期をかけ大きな土台に成長させる予定である。

３．知的プログラミング・システム

まだ、小規模な実験システムの試作や理論的検討の域を出ていない仕様記述・検証技術への本格的な取組み、実用化と称しうるレベルのシステムの開発を目指して展開される。これを基調とし、ソフトウェアの日本語化、さらに自然言語化とソフトウェア開発・[illegible]作業支援の統合化が絡む。仕様記述、検証技術に向っては、様々な側面からの様々な試みが必要であるが、大きく仕様記述の側からと検証・自動証明システムの側からアプローチする。対応するメーカの研究グループの努力に加え、仕様記述に関しては、東工大（根本教授）TELL/NSLプロジェクトとの共同研究、自動証明に関しては、基礎理論ワーキンググループ（広瀬教授主査）における研究活動が、大きな支えとなっている。