TM-0108

ICOT 前期成果報告会資料

May, 1985

©1985, ICOT

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

ICOT前期成果報告会資料

昭和60年5月22日（水）

東京農林年金会館

（財）新世代コンピュータ技術開発機構

目　次

6．関係データベース・エンジンの開発

柴山茂樹（ＩＣＯＴ）

12:00 ～12:30

関係データベース・エンジンは、関係データベースマシンDelta の主要な演算ユニットである。ソート処理を主体とした関係演算用の専用ハードウェア、また算術系の演算と全体の制御を行うソフトウェアについて、処理アルゴリズム、構成及び評価結果について報告する。

7．自然言語処理（ＤＵＡＬＳ：談話理解実験システム）

向井国昭（ＩＣＯＴ）

13:30 ～14:00

会話文を含む複数の文で構成された文章の意味を理解、その内容に関する質問に答えることができるシステムDUALS について報告する。構文の解析にはLFG を用いた。意味論と語用論の考え方は状況意味論に従っている。例題は小学校３年の国語のドリルから選んだ。

8．Prologプログラムの検証システム

金森　直（三菱電機）

14:00 ～14:30

Prologのプログラムが一階論理式で示される性質を満たすことを機械的に証明するシステムの概要を報告する。特に論理プログラミングのパラダイムにより、この論理式をあたかも通常のゴールのように実行して推論できる。

9．知的ＣＡＤ実験システム

丸山文宏（富士通研究所）

14:30 ～15:00

仕様に基づきVLSI用回路の設計全過程を支援する知的CAD 実験システムの試作、評価結果について報告する。論理型言語を用いれば、これらの過程で用いる設計のノウハウを簡明に表現し利用できることが明らかになった。

10．ＰＳＩマイクロインタプリタの性能評価

横田　実（ＩＣＯＴ）

15:20 ～15:50

逐次型推論マシンPSI の実行性能をいくつかのベンチマークプログラム、応用プログラムを用いて評価した結果をもとに、PSI マイクロインタプリタの特性について報告する。

１１．ＥＳＰのプログラミング環境　　近山　隆（ＩＣＯＴ）

15:50 ～16:20

逐次型推論マシンPSI 上のプログラミング／オペレーティング・システムSIMPOSは、対象指向機能を取入れた論理型言語ESP で記述されている。SIMPOS上でのESP プログラム開発環境について報告する。

１２．ＳＩＭの拡張機構：高速プロセッサモジュール　　梅村　護（日本電気）

16:20 ～16:50

逐次型推論マシン(SIM) の研究開発においては、パーソナル型のマシンPSI の能力を越えるような大規模の知識情報処理の応用問題に対処するために、PSI の５倍以上の能力を持つ、高速プロセッサモジュールを開発している。このプロセッサの特徴について報告する。

核言語第一版　ＫＬ１

竹内彰一　（ICOT　第２研究室）

１　序

核言語第一版（以下、ＫＬ１）はＦＧＣＳプロジェクト中期で開発される並列推論マシンＰＩＭの機械語（あるいはそれより若干上位の抽象マシンの機械語）として位置付けられている。はじめにＫＬ１の設計をする際に背景にあったいくつかの思想について述べる。

- ＫＬ１は並列性を陽に記述する並列プログラミング言語である。

ＫＬ１は協調問題解決システムのような複数の並列動作するものをもつシステムを構築する際の出発点となるべき言語である。従って、根本的に並列性を陽に記述する並列プログラミング言語でなくてはならない。

- ＫＬ１は並列実行する上での問題点が少なく、効率の良い実現が可能な言語でなくてはならない。

ＫＬ１は並列推論マシン上に実現される。一般に並列マシン上での並列処理は多くの困難な問題を内包している。従って、ＫＬ１はその実現を考えたときにこれらの困難な問題をできるだけ回避でき、かつ、効率の良い実現が可能な仕様になっていなくてはならない。

- ＫＬ１は単純で一様な構造の言語でなくてはならない。

ＦＧＣＳプロジェクトの一環として、ＫＬ１で協調問題解決システム等の並列ソフトウェアを開発するわけであるが、並列ソフトウェアを設計すること、および、並列ソフトウェアを並列マシン上で動かすことが、両者共に、経験の蓄積の少ない分野であり、解決しなくてはならない問題も多い。ＫＬ１の仕様はこの２つの問題がＫＬ１のプログラマに分離された形で現れるようなものに当面すべきであり、２つの問題に対する経験を個別に蓄積するのが良いであろう。このためには、ＫＬ１の言語仕様の中で、プログラムを並列マシン上でいかに走らせるかを指定するための言語要素を独立させ、残りの基本部分は並列マシンで走らせるという観点からは単純かつ一様な構造をしており、特に並列マシンの構造からは独立、中立であるのが望ましい。

- ＫＬ１は知識情報処理の土台でなくてはならない。

そのためには、柔軟なソフトウェア・システムを実現できるだけの十分な記述力が必要である。特にメタ・プログラミング機能は不可欠である。

２　ＫＬ１(84)　から　ＫＬ１(85)へ

ＫＬ１の概念仕様は昭和58年末に『核言語第一版概念仕様書』[1]　としてまとめられた（以下、これをＫＬ１(84)と呼ぶ）。ＫＬ１(84)は、(1) 並列推論機能、(2) 集合抽象化機能、(3) メタ推論、(4) モジュール化機能の４つの機能を柱として設計されている。

昭和59年度はこの概念仕様をもとに、仕様に関する多くの議論および実現方式の検討が行なわれた。実現方式の検討に関しては、ＫＬ１(84)のサブセットである Concurrent Prolog　のインタプリタおよびコンパイラを開発する過程で行なわれた[2,3,4,5,6]　。仕様の検討および実現方式の検討を行なった結果、ＫＬ１(84)にはいくつかの問題点があることが認識され、昭和59年度末に旧仕様が改訂されることとなった（この改訂版ＫＬ１のことを以下、ＫＬ１(85)と呼ぶ）。

３　改訂の方針

ＫＬ１(84)を改訂する上で考慮した点は以下の２つである。

(1) 並列処理に適した並列プログラミング言語であること。
(2) ＫＬ１(84)で列挙された言語要素のレベルの整理およびハードウェアとの関係の明確化

(1) はＫＬ１を並列推論マシン上の言語とする位置付けから明らかである。ＫＬ１(84)の並列推論機能のベースとなっていた Concurrent Prologはこの観点から見直され、いくつかの問題点が指摘された。それらは、 Concurrent PrologのOR並列を実現する際に必要となる多環境に関連して生じる環境間の整合性の検査の問題、および、大きなユニフィケーションをより小さな並列実行可能なユニフィケーションに分解しようとする際に現れる限界の問題などである[7]　。これらの問題を効率良く解決する方法を模索する過程で、これらの問題点をもたず、かつ、Concurrent Prolog と同等の記述力をもつ言語モデル Guarded Horn Clauses （以下、ＧＨＣ）[8]　を得、ＫＬ１(85)では Concurrent Prologにかわり、ＧＨＣを並列推論機能のベースとすることにした。

(2) については、ＫＬ１(84)で提唱された言語要素それぞれの実現されるべき言語レベルを再検討し、基本となる言語要素および基本言語要素上で実現されるべき上位レベルの言語要素を調べた。また、ＫＬ１のプログラムの論理的な実行モデルと、実マシン上での実際の実行との間の対応を記述する方法についても検討を行なった。これらを検討した結果、ＫＬ１(85)では、全体を、ＫＬ１－ｃ(core)、ＫＬ１－ｐ(pragma)、ＫＬ１－ｕ(user)の３つに分割し、各部の役割をより明確にすると同時に、各部分相互の関係

も明らかにし、ＫＬ１全体としての複合的役割の鮮明化を行なった。

4　ＫＬ１(85)の概要[9]

ＫＬ１(85)の全体構成を図１に示す。図にあるように、ＫＬ１はＫＬ１－ｕ、ＫＬ１－ｃ、ＫＬ１－ｐの３つの部分からなる。

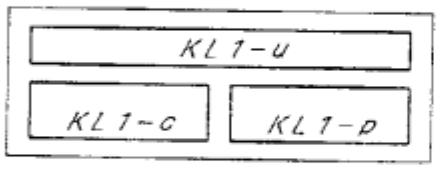


図１　ＫＬ１(85)の全体構成

ＫＬ１－ｃはＫＬ１(85)の言語としての基本部分であり、ＧＨＣに基づいており、ＫＬ１(84)のAND 関係、モジュール化機能およびメタ推論機能に対応する機能を実現する。ＧＨＣは、プログラムがガードつきHorn節で構成されるという点で、Concurrent Prolog やPARLOG[10]と良く似た言語である。ＧＨＣとConcurrent Prolog との相違はＧＨＣは多環境を必要とせず、またread-only annotationをもたないことにあり、ＧＨＣとPARLOGの相違はＧＨＣはモード宣言をもたないことにあるといって良い。ＧＨＣのプログラムは次の形をしたガードつきHorn節の集まりである。

H :- G1, ..., Gm | B1, ..., Bn.　(m>0, n>0)
（G1, ..., Gm：ガード　B1, ..., Bn：ボディ）

ＧＨＣにおける同期はConcurrent Prolog ともPARLOGとも異なったメカニズムで実現される。すなわち、述語を呼出したとき各節のガードを実行している間は呼出し元から観測できるようなbinding を生成しようとしたユニフィケーションは中断・待機する。ＧＨＣの特徴は以下のようにまとめられる。

①単純さ

(1) 述語呼出し、ユニフィケーションの不可分な断片、節の選択操作という少数の小さな基本操作

(2) 条件分岐、非決定性、同期などが記述できる単純で強力なガード機構

②記述能力

既存のConcurrent Prolog プログラムのＧＨＣへの書き直しが可能あり、特に、ＧＨＣでＧＨＣ自身のメタ・インタプリタを書くことができる。

③実現の容易さ

多環境の管理が不要なこと、及び同期に関する多くの情報が静的に得られるためコインパイル度の向上が期待できるなど実現がConcurrent Prolog と比べると容易である。

ＫＬ１－ｐは、ＫＬ１(85)で新たに導入されたものであり、マシン独立なＫＬ１－ｃのプログラムを並列マシン上で実行する際に、並列マシンのもつ物理資源（プロセッサ、時間）をプログラムに合わせて効率的に活用するための補助的言語（プラグマ）である。ＫＬ１－ｐは並列マシンの抽象イメージを提供し、その上でプログラムを空間的に分割したり時間的に分割したりする言語要素を提供する。ＫＬ１－ｕは、ＫＬ１－ｃ，ｐの上に位置するものである。ＫＬ１－ｕはＫＬ１－ｃのもつ機能の他にＫＬ１(84)のOR関係、集合、モジュール等の上位レベルの言語要素を含み、ＫＬ１－ｐの機能も一部備えたシステム・プログラミング向き言語であり、ＫＬ１－ｃ，ｐにコンパイルされて実行される。

5　今後の課題

ＫＬ１の今後の課題としては、第一にＫＬ１－ｃの並列マシン上での実現方式を確立することがある。ＫＬ１－ｐについては、複数の案についてシミュレータ上で実験・評価をする必要がある。そしてこれらをベースに大規模なアプリケーションを開拓し、その結果をＫＬ１－ｕの仕様へとフィードバックさせ、強力でかつ使い易い並列プログラミング言語へとＫＬ１を発展させていく予定である。

文献

[1] 古川他：核言語第一版概念仕様書，ICOT TR-054 (1984).

[2] T.Miyazaki, A.Takeuchi, T.chikayama : A Sequential Implementation of Concurrent Prolog based on the Shallow Binding Scheme, to appear in American Logic Programming Symposium 1985.

[3] K.Ueda, T.Chikayama : Concurrent Prolog Compiler on top of Prolog, 同上

[4] 宮崎他： Concurrent Prologのシーケンシャル・インプリメンテーション－－shallow binding 方式による多環境の実現－－，日本ソフトウェア科学会第１回大会論文集，1984.

[5] 佐藤他： Concurrent Prologのシーケンシャル・インプリメンテーション－－deep binding 方式による多環境の実現－－，同上

[6] 田中他： Concurrent Prologのシーケンシャル・インプリメンテーション－－lazy copying方式による多環境の実現－－，同上

[7] K.Ueda : Concurrent Prolog Re-examined, ICOT TR-102.

[8] K.Ueda : Guarded Horn Clauses, ICOT TR-103.

[9] 核言語第一版説明資料、ICOT, 1985, 5月。

[10] K.Clark, S.Gregory : PARLOG: Parallel Programming in Logic, Research Report DOC84/4, Imperial College, 1984.

リダクション方式並列推論マシン：ＰＩＭ－Ｒ

尾内　理紀夫　（ＩＣＯＴ　第一研究室）

1．はじめに

ICOTは述語論理型言語をベースにした知識情報処理システムの研究開発を行なっている．述語論理の基本操作は推論であるから，そのハードウェアは推論マシンと呼ばれ，また，推論が基本操作であるから，このマシンは並列動作が基本となる．並列推論方式あるいは述語論理型言語として，現在いくつかのものが提案されているが[11],[12],[13],[1],[14],[15]，本マシンの対象言語としては，ICOTが核言語第１版(KL1) のベース言語として位置づけているPrologとConcurrent Prolog [1] を選択した．並列推論方式としては，reduction概念に基づきPrologをOR並列に，Concurrent Prolog をAND 並列に処理する方式を採用した．

次にreduction ベースのマシンを考えるに至った動機について簡単に述べる．　式 7+3のreduction を考えた時，この式は加算に関するruleを用いて自らをmodifyし，10となる．そして式10はこれ以上reduction(modification) できない（既約）ので解となる．すなわち，reduction は，self-modificationとみなせる[2] ．一方，PrologあるいはConcurrent Prolog の実行過程は，親clause (goalとclause) からのresolvent の生成過程であり，resolvent として空節が導出されれば，それが解となる．これは，goalがclause群という一種のruleを用いて自らをmodifyする過程とみなせる．このように，Prologあるいは ConcurrentPrologプログラムの実行過程と，reduction との間に親和性の良さを見出すことができる．これがreduction 概念に基づく並列推論マシンPIM-R (Reduction-Based Parallel Inference Machine)の研究開発の動機であり，resolvent生成過程の並列実行がPIM-R の基本動作となる．

PIM-R はstructure-copy方式を採用しているが，copyおよびそれに伴う処理量の低減およびnetwork 通過packet数とpacket幅の低減のためreverse compaction[3]，only-reducible-goal-copy[8],[10]，独特のプロセス構成法[8],[10]を採用している．アーキテクチャとしては，並行プロセス間チャネル用分散化共有メモリ(Message Board) の導入，効率的packet分配のためのIntelligent なNetwork Nodeの導入，Ground Instance である長い構造体データ格納のための構造体メモリ[4] の導入をその特徴としている．本稿では，PIM-R におけるKL1 ベース言語の並列実行方式，PIM-R アーキテクチャ，ソフトウェアシミュレーション，ハードウェアシミュレータ試作について述べる．

2．KL1 ベース言語の並列実行方式

PIM-R では，PrologをOR並列（応用問題(BUP等）によっては高い並列度が期待できる[7] ）に，Concurrent Prolog をAND並列（ただしgoalが並列AND operatorで結合された時のみ）に実行する．PIM-R においては，goal(body側リテラル）が複数あった時（それら複数goal全体を親プロセスと呼ぶ．但し，並列AND 関係にあるgoalはAND forkして別々のプロセスとなる）は，その内のreducible なgoal（どれがreducible かは各operatorにより指示される）のみをreduceし，生成されたresolvent(子プロセス）から親プロセスへポインタが張られる．このポインタにより子から親へ解が返される．すなわちPIM-R でのProlog，Concurrent Prolog の処理過程はプロセス木の伸張，縮小の過程であり，処理が終了した時，木は論理的に消滅する．Prologの場合と異なり，Concurrent Prolog の場合は，OR関係にある子プロセスは異なる処理要素に分配されるのではなく，まず同一処理要素(IM)内に格納して，そのguard 処理をする．一方，並列AND operatorで結合されたgoalは異なるIMへ分配し並列実行する（AND 並列実行）．またConcurrent Prolog では，１つのプロセスがcommit処理に成功した時に，兄弟プロセスを殺しにはいかずに，遅れて成功した時に自殺する方法をとる．

3．PIM-R アーキテクチャ（図１）[8],[10]

PIM-R は二種類のHodule(Inference Module とStructure Memory Module)とそれらを結ぶNetwork から構成される．

3.1 Inference Module (IM) （図２）

Unification UnitとProcess Pool Unit から構成される．

3.1.1 Unification Unit(UU)

(1)Clause Pool

各Clause Pool は同一clause群を格納する(1語32ビット)．

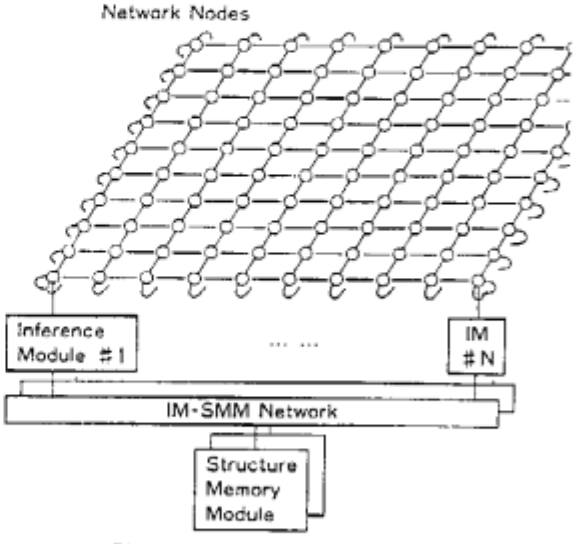


図１．　PIM-R アーキテクチャ構成

(2)Matcher

PPU から送られてきたgoalの第一引数のデータタイプによりunifiable なclause候補の絞り込みを行なう.

(3)Unifier

Matcher から送られてきたgoalとClause Pool からコピーしたclauseとの間のunification あるいは組込み述語の実行を行ない，結果をPPU へ返す．PrologはOR並列実行するので，新たに生成された子プロセスは，分配方式に従って自IM内PPU あるいは，network 経由で他IMへ分散させる．Concurrent Prolog の場合，unification の結果は，常にいったん自IM内PPU へ戻す． またサスペンドした場合には，将来の再unification のために.unificationしようとしたclauseの格納場所，サスペンドの原因となったgoal側のチャネル等の情報を自IM内PPU へ返す．

3.1.2 Process Pool Unit(PPU)

PPU は，二種類のメモリ (Process PoolとMessage Board)と，二種類のコントローラ (Process Pool Controller とMessage Board Controller) から構成される．

(1)Process Pool Controller(PPC)

PPC は以下の各処理を行なう．

- fork処理 (OR/AND fork 数の設定，OR/AND fork 失敗処理)
- 子プロセス処理 (子プロセスの成功，失敗，生成，サスペンド処理および，親プロセスへの結果報告処理)
- commit処理
- reducible プロセスのUnification Unitへの転送処理
- deadプロセス処理

(2)Message Board Controller(MBC)

MBC は次の各処理を行なう．

- チャネルのためのセルの確保
- チャネル値の読み出し処理とチャネル値の格納処理
- サスペンドプロセスへのactivate処理

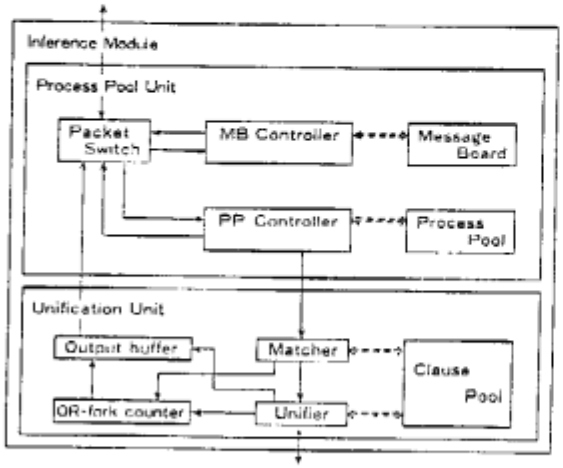


図2 Inference Module構成

(3)Process Pool(PP)

PPはプロセスを格納するメモリであり，1語32ビットである．structure-copy方式に起因するcopy量とIM間の通信をなるべく少なくするようなプロセス構成法とプロセス内部形式[3] を採用している．reducible なプロセスはready キューに繋がれ，先頭からPPC により取り出され，UUへ送られる．また，各プロセスはreduction レベル(プロセス木の根からの深さ) を持ち，これをもとにready キュー内のプロセスを入れかえることにより，IM内depth-first ，あるいは，IM内breadth-first の処理を実現できる．

(4)Message Board(MB) [5],[6]

Concurrent Prolog における並行プロセス間チャネル用のリモートアクセス可能な分散化共有メモリである．

3.2 Structure Memory Module(SMM) [4]

SMM は，Ground Instance である長いリストやベクタ等の構造体データを格納し，UUからの要求に対し，unification に必要な構造体データをUUへ転送する．

3.3 Network

IM間Network は，近傍PPU 内Packet Switch の入力バッファ長等の情報により子プロセスの各IMへの分配を動的に制御できるNetwork Nodeを格子状に配置した構成である． IM-SMM間Network は共有バスによる実現を考えている．

4．ソフトウェア・シミュレーション

4.1 基本動作確認ソフトウェアシミュレーション[9],[10]

台数効果，アーキテクチャの基本的検証等，PIM-R の基本動作確認のために，Prologで記述されたソフトウェアシミュレータを開発し，テストプログラムを走行させ，各種データを収集した．その結果，PIM-R が，Prolog,Concurrent Prologに内在する並列性を引き出すこと，MBC の導入効果，Network Nodeによる子プロセスの動的分配効果，PPC におけるdead処理 ,fork down packet生成／転送処理の休止効果，reduction レベルによる疑似depth-first 処理効果を確認した．

4.2 詳細ソフトウェアシミュレーション

PIM-R データ内部形式等のPIM-R 詳細構造を正確に反映したシミュレーション，PIM-R のInference Module16~64台以上のシミュレーションを主目的として，詳細ソフトウェアシミュレータの開発と行なっている．記述言語として，並行プロセス記述と通信記述機能を持つ言語Occam を採用し，現在までに，IM ,SMMの単体試作と結合を完了し，SMM の導入効果等を確認している．本シミュレータは，PIM-R のUnification Unitにおける1unificationのシミュレーションを高速に実行でき，各種詳細なシミュレーションの実行が期待できる．

5．ハードウェアシミュレータ試作

PIM-R の詳細構造シミュレーションと並列環境シミュレーションを主目的として，ハードウェアシミュレータを試作した．(図3) 早期開発，柔軟性という開発条件から，68000 搭載のシングルボードマイクロコンピュータ(SBCと略す) を用いて，

PIM-R のInference Moduleを実現することとした．そして，PIM-R の各種モジュールは，ソフトで実現し，各モジュールの変更改良に容易に追従できるようにした，また，network は，格子状構造を出発点としたが，応用分野あるいは，KL1 の変更改良によっては，異なるnetwork をも検討してみる必要がある．そこで，固定的なハード構成ではなく，柔軟な共有メモリによる構成とし，その上で各種network をシミュレートすることとした．並列環境のシミュレーションを可能とするために，SBC を実際に複数台（当初8台，将来16台）配置し，SBC と共有メモリとを結ぶ共有バスには，ハード化したバスアービタを導入した．ハードウェアシミュレータ上での各種応用プログラムの走行を可能にするため，各SBC には，十分なローカルメモリを付けることとし，10M バイトまで実装可能とした．また，SBC 群と共有メモリのスーパーバイザとしてVAX11/780 を導入した．この結果，network については，二種類の格子状構造と鎖状構造についてシミュレーションを行なうことができ，また，PIM-R がProlog,Concurrent Prologに内在する並列性を引き出すことも確認した．

6．おわりに

PrologをOR並列に，Concurrent Prolog をAND 並列に実行する並列推論マシンPIM-R アーキテクチャとシミュレーションについて述べた．なお，新たなKL1 ベース言語とされたGHC も，三種類の新データタイプを導入することにより，PIM-R におけるConcurrent Prolog 実行機構を変更することなく，実現可能であると考えている．今後は，詳細ソフトウェアシミュレータによるIM16～64台以上のシミュレーションによるデータ収集，8 台から16台に拡充したハードウェアシミュレータによるデータ収集を行ない，PIM-R の検証を行なう予定である．

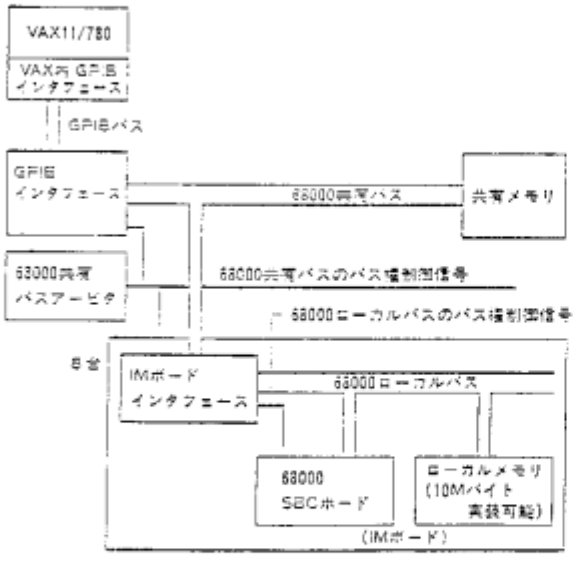


図3　PIM-R ハードウェアシミュレータ構成

最後に，本研究開発は，（株）日立製作所（杉江節，米山貢，岩崎正明，坂部俊文，古住誠一各氏）の御協力を得て，ICOT第一研究室 PIMグループ（尾内理紀夫，麻生盛敏，清水肇，益田嘉直，松本明）により行なわれた．

また，御懇談いただいた渕一博ICOT研究所長，村上国男前第一研究室長，御指導，御討論いただいた東京大学　元岡達教授（プロジェクト推進委員長），田中英彦助教授（ワーキング・グループ1主査）ならびに関係各位に深謝する．

［参考文献］

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

[2] D.A.Turner:A New Implementation Technique for Applicative Languages,Software-Practice and Experience, No.1,Vol.9,1979 .

[3] 清水他: 並列推論マシンPIM-R のプロセス内部表現，第30情処全国大会　1985.

[4] 益田他: 並列推論マシンPIM-R の構造体メモリの一構成法，第30回情処全国大会　1985.

[5] 尾内他: 並列推論マシンにおけるGuard と入力annotationの制御機構，第27回情処全国大会　1983.

[6] 尾内他: 並列環境におけるConcurrent Prolog の実現法，第29回情処全国大会　1984.

[7] 尾内，清水他: 逐次型Prologプログラムの解析，Logic Programming Conference 84' 東京，1984

[8] 尾内他: 並列推論マシンPIM-R のアーキテクチャ，第30回情処全国大会　1985.

[9] 尾内他: 並列推論マシンPIM-R のソフトウェアシミュレーション，第30回情処全国大会　1985

[10]Onai,R.,et al, :Architecture of a Reduction-Based Parallel Inference Machine:PIM-R, New Generation Computing, Vol.3,No.2, 1985. (to be published)

[11]Moto-oka,T.,Tanaka,H.,Aida,H.,et al, : The Architecture of a Parallel Inference Engine -PIE- ,FGCS '84,ICOT,1984.

[12]Ito,N. and Masuda,K., :Parallel Inference Machine Based on the Data Flow Model, Proc. of the International Workshop on Highlevel Computer Architecture 84 1984.

[13]久門他：並列推論処理システム－改良型節単位処理方式－，第30回情処全国大会　1985.

[14]Clark,K.L. and Gregory,S., :PARLOG:Parallel Programming in Logic", Research Report DOC 84/4, Dept. of Computing, Imperial College London, 1984.

[15]Pereira, L.M. ,et al, :DELTA-PROLOG: A Distributed Logic Programming Language", FGCS '84,ICOT,1984.

データフロー方式並列推論マシン：ＰＩＭ－Ｄ

伊藤徳義　（ＩＣＯＴ　第一研究室）

１．　はじめに

PIM-D(Parallel Inference Machine Based on the Data Flow Model) は並列型核言語KL1(Kernel Language 1)を目標言語とするデータフロー方式並列推論マシンである．PIM-D の基本処理モデルであるデータフローモデルは，KL1 に含まれる２つのタイプの並列型Prolog（ＯＲ並列型Prolog及びＡＮＤ並列型Prolog）と密接に関連しており，これらの言語に内在する並列性を実現する処理系としての可能性を与える．本プロジェクトの前期３年間でその処理方式の検討，ソフトウェアシミュレータやモジュールの試作，及び評価を進めてきたのでその結果を報告する．

２．　PIM-D の特徴

PIM-D は以下のような特徴を持っている．

①データフローモデルをベースとする

並列型Prologは，ゴールが与えられたときにプログラムの実行が開始されるというゴール駆動型の特性を持っている．一方，並列処理の実現を自然な形で実現可能なデータフローモデルは，必要なデータが揃った時点で処理（命令）が起動されるというデータ駆動型の特性を持っており，並列型Prologのゴール駆動型の特性と密接な関係にある．これによってPIM-D は並列型Prologに内在する並列性を自然な形で実現することができる．

②ＯＲ並列，ＡＮＤ並列，引数間の並列統一化の実現

ＯＲ並列性，ＡＮＤ並列性，及び，引数間の並列統一化を実現し，プログラムに内在する並列性を最大限に生かすようにした．ＯＲ並列型Prologに対しては，不可視ストリーム(invisible stream)を用いた並列実行を，ＡＮＤ並列型Prologに対しては可視ストリーム(visible stream)を用いた並列実行を実現しており，両並列型Prologを統一的に処理している．

③構造データ共有方式の採用

PIM-D の目標とする知識情報処理の分野では複雑な構造データの操作が必要となる．このようなプログラムを並列マシン上で実行するとき，構造データコピー方式を採用すると，構造のコピーのためのネットワーク負荷が大きくなると共に，構造を格納するための資源（メモリ）の消費量が増加し，場合によっては，このための資源消費によってシステムデッドロックに陥る恐れがある．従って，PIM-D では構造データをプロセス間で共有しておき，必要が生じたときに構造にアクセスを行う構造共有方式を採用した．

３．並列型Prologの処理方式

並列型Prologに内在する並列性については，大別してＯＲ並列性とＡＮＤ並列性があるが，これらのうちいずれを重視するかに応じて２つのタイプの言語がある．ＯＲ並列型Prologは，論理和（ＯＲ）関係にある節の間の並列性（ＯＲ並列性）に重点を置いた言語であり，主として与えられた問題に対する全ての解を並列探索するのに使用される．起動された各ＯＲプロセスは，与えられたゴール（質問）に対する解を互いに独立に探索する．これに対してＡＮＤ並列型Prologは，論理積（ＡＮＤ）関係にあるゴールリテラルの間の並列性（ＡＮＤ並列性）に重点を置いた言語であり，起動されたＡＮＤプロセス間で論理変数を共有させると共に変数に結合されるインスタンスのインタラクティブな通信機能を備えている．

ＯＲ並列型Prologは純PrologにＡＮＤリテラル間の並列又は逐次実行を制御するオペレータを導入した言語である．一方ＡＮＤ並列型Prologに関しては，PARLOG，Concurrent Prolog，又は，GHC(Guarded Horn Clauses) 等が提案されている．これらはいずれも，プログラムをガード節の集合として表現している．このうち，主としてConcurrent Prologの処理方式を検討してきた．

並列型Prologプログラムは，ゴールが与えられたとき起動され，ゴール内のリテラル（ゴールリテラル）とプログラム中の各節の頭部リテラルとの統一化（頭部統一化）が試みられる．このとき，統一化が成功する可能性のある節は，頭部リテラルの述語名がゴールリテラルのそれと同一で，しかもそれらのリテラルの引数の数が同一であるものに限定される（このような節の集合をその述語に対する定義と呼ぶ）．各定義はデータフローグラフにコンパイルされる．このとき，定義中の節に現れる変数をそのインステンスが渡されるパス名として解釈することによって，データフローグラフに変換する．図１に，以下の節が与えられたときの解析グラフを示す．コンパイラはこの解析グラフをもとにマシンコードを生成する．

```
p(X,Y) :- q(Y,Z) & r(Z,X) & ...
```

ＯＲ並列型Prologにおいては，与えられたゴールリテラルと各々の節の頭部リテラルの間で統一化が試みられ，この頭部統一化が成功した節はその本体（があれば）を呼出す．これによって得られた解（頭部リテラルのインスタンス）はゴール側に戻される．一般にＯＲ並列型Prologにおいては，得られる解は複数個在し，それらの解をゴール側に返す順序は非決定的である．即ち，求められた順に解をゴール側に返すことができる．PIM-D ではこのような解の集合をnon-strictな構造データであるストリームとし

て表現している．ＯＲ並列型Prologでは，ストリームの構造はプログラマに対して陽に“見えない”．このようなストリームを不可視ストリームと呼んでいる．

これに対してConcurrent Prolog においては，プログラマが陽に意識してストリームの制御を行う．このようなストリームを可視ストリームと呼ぶ．ＯＲ並列型Prologと異なりすべての節がガード節であり，コミットオペレータによって節の間の排他的制御が行われる．ゴールリテラルが与えられると定義中の全ての節の頭部リテラルとの統一化，及びガード呼出しが起動される．これらが成功した節は，コミットオペレータを実行する．コミットオペレータは，定義内の節間の排他制御を行う機能と，それまでの処理で得られた共有変数に対する結合環境を他プロセスに輸出(export)する機能によって実現される．

4．　マシンアーキテクチャ

マシンは図２に示すように複数のPEM(Processing Element Module)，複数のSMM(Structure Memory Module)，及び，これらを結ぶネットワークから構成される．PEM は図１で示したようなデータフローグラフをロードし，その解釈・実行を行う．SMM は節の引数として現れる構造データを格納し管理するために使用される．前述のように構造データはPEM で実行中のプロセス間で共有されており，各PEM は構造データへのポインタを持っている．プロセスは構造データの内容をアクセスする必要が生じたときにSMM へデータアクセス指令を送出する．特定のSMM へ構造データアクセス要求が集中するのを回避するために，構造データは全てのSMM 間で分散して格納される．

各PEM はさらに，トークン（データを運ぶパケット）を一時的に格納するPQU(Packet Queue Unit)，命令の発火制御を行うICU(Instruction Control Unit)，命令の解釈・実行を行うAPU(Atomic Processing Unit)群から成る．これらの機能ユニットは互いに独立に動作し，データフローグラフをパイプライン的に実行する．

5．　評価結果

上述のマシンを評価するためにソフトウェアシミュレータを開発し，ＯＲ並列型Prolog及びＡＮＤ並列型Prologプログラムに対する評価を行った．シミュレータは図２に示したような機能ユニットを独立のプロセスとして実現しているイベント駆動型のシミュレータである．各機能ユニットの処理時間は適当なハードウェア構成を仮定して決定した．図３はマシンのモジュール数（PEM 及びSMM の台数）を最大６４台まで変化したときのＯＲ並列型Prologプログラムに対する推論性能の様子を示す．ここで推論性能は単位時間当りに頭部統一化の行われた回数HUPS(Head Unification Per Second) で表している．図４は同様にモジュール数を変化したときのConcurrent Prolog プログラムの推論性能の様子を示す．

現在このソフトウェアシミュレータによる評価と並行してマシン実験機による評価も進めている．

6．　おわりに

PIM-D における並列型Prologの処理方式やマシンアーキテクチャを示し，その評価を行った．これによると十分な台数効果が得られることが判明した．今後，引続きその詳細な検討を進めていく予定である．

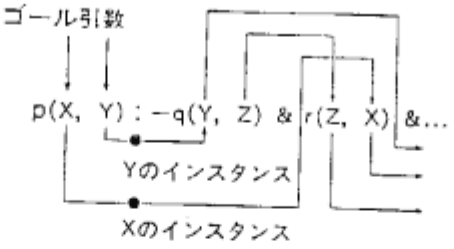


図１．インスタンスの受渡し関係

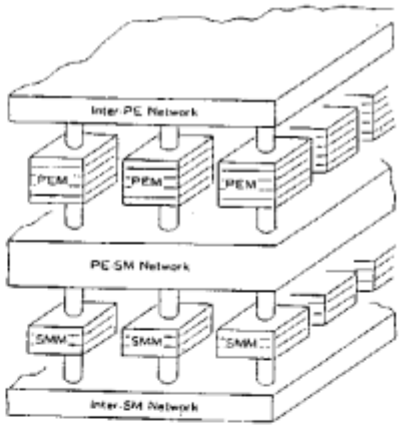


図２．マシンアーキテクチャ

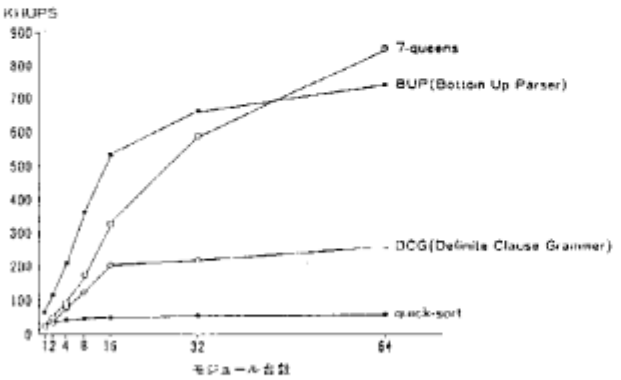


図３．ＯＲ並列型プログラムの推論性能

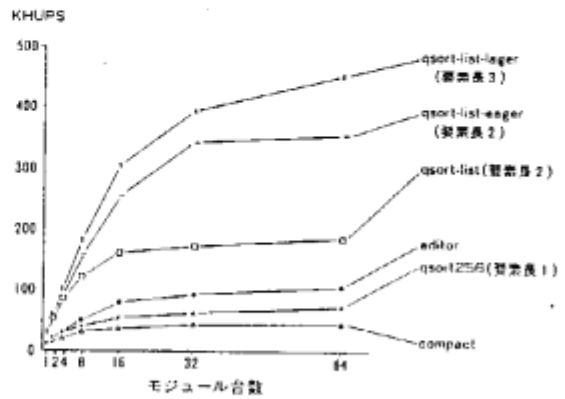


図４．Concurrent Prolog プログラムの推論性能

知識ベース管理実験システム(KAISER)

宮地泰造 (ICOT 第二研究室)

1. はじめに

大規模な知識ベース管理システムを構築する第一歩として、KAISER (Knowledge Acquisition-oriented Information SupplieR) と呼ばれる大規模な知識ベース管理システムを、関係データベース管理機能をも含めて研究試作した。

KAISERの基本機能は、次の3つである。

①知識の獲得
②知識の蓄積
③知識の利用

KAISERの基本機能は次の5つのモジュールにより提供される。

①知識獲得モジュールKAM (Knowledge Acquisition Module)
②知識操作モジュールKMM (Knowledge Manipulation Module)
③知識蓄積モジュールKSM (Knowledge Storage Module)
④知識インタフェース・モジュールKIM (Knowledge Interface Module)
⑤知識対話モジュールKCM (Knowledge Conversation Module)

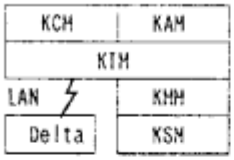


図1. KAISERの構成と環境

図1は、KAISERのモジュール間の階層関係およびDelta (RDBM) との関係を示している。概略的に言えば、それぞれ、知識獲得機能はKAM 、知識蓄積機能はKMM とKSM 、知識利用機能はKCM により実現されている。

また、大規模な知識ベースを逐次型推論マシンPSI 上で利用可能にするために、ホーン節に対する推論システムと関係データベースとをLAN を介するソフトウェア/ハードウェアの両方で結合することが試みられた。このために前期三年間かけてKMM , KSM とKIM の三モジュールがDEC-10 Prolog/ESP 上で研究試作された。KCM は、KIM やKSM 上に構成される自然言語インタフェースであり、利用者はKCM を介して知識ベースを容易にアクセスできる。KCM は、昭和59年度において、PSI 上のESP 言語を用いて開発された。KAM の基本機能の検討は高度な研究課題であったので、前期三年間をかけてDEC-10 Prolog により研究試作された。

2. 知識獲得機能[1,2,3,4,6)]

知識獲得の問題に対して、KAM では論理的観点から設計が行われた。ピアジェの発生的認識論に基づく分析の結果、知識獲得のプロセスは知識同化プロセス、知識調節プロセス、知識均衡化プロセスからなることが分った。そこでKAM では、オブジェクト知識やメタ知識の知識同化や知識調節といった知識ベース管理のための基本機能を持ち、知識ベースへの知識獲得支援を行う機能を実現している。

KAM の実現にとって本質的なのはオブジェクト知識の使い方に関する知識、すなわちメタ知識を管理し利用する推論機構である。このメタ推論機能を用いて、与えられた知識ベースに新しいオブジェクト知識を無矛盾に追加することを支援する知識同化機構、知識ベースに新たな知識を追加した際、それを説明するように知識ベース内に知識を追加したり、他の知識を修正していく知識調節機能、等が実現されている。オブジェクト知識の同化や調節に比べて、メタ知識の同化や調節の実現は難しい。KAM においては、あいまいでない知識 (前提型知識) とあいまいな知識 (仮定型知識、信念) がオブジェクト知識として混在する世界で、メタ知識の追加に対して両者の間に矛盾が生じないように、あいまいな知識の修正 (信念の翻意) を支援するメタ知識同化機能も実現されている。たとえば、KAM はシェークスピアの『真夏の夜の夢』の愛情関係において、オブジェクト知識である愛情関係がメタ知識に矛盾しないように調節し、「悲劇が起きてはいけない」というメタ知識を同化することができる。これは、KAM が持っている汎用的な真実性維持 (truth maintainance) の機能により処理されている。

さらにKAM は、知識同化時に、知識ベースの状態や関連するメタ知識間の連鎖作用を考慮して知識ベースが無矛盾であるように管理する機能を持っている。この高度な知識同化機能は、実世界の存在物である個々のオブジェクトを管理することにより実現される。すなわち、個々のオブジェクトが満足すべき制約条件 (constraint) を管理することにより、オブジェクトおよび知識ベースが無矛盾であるように管理するのである。ここで、各オブジェクトは、大別して2種類の制約条件を持っていると考えられている。1つは、オブジェクトが対象とする世界に存在するための存在制約 (Existential Constraint) であり、もう1つは、対象の世界におけるオブジェクトの活動のための活動 (動作) 制約 (Action Constraint) である。

3. 知識インタフェース機能

大規模な知識ベースを逐次型推論マシンPSI 上で利用可能とするためのインタフェース機能は重要である。

KIH は、PSI 上の応用プログラムで必要とされる知識がどこの知識ベースに存在しているかを調べ、適当な知識ベースに問題を送信してその解答を得ることにより、応用プログラムに有用な知識を提供する機能を実現している。具体的にはKIH は、PSI マシンのESP 言語とPSI の関係データベース・マシンDelta の関係データベース操作用言語とのインタフェースをとる。すなわち、KIH は、ESP で記述された応用プログラム（この中にはKCH やKAN も含む）からPrologのゴール列を受け取り、その個々のゴールを解くことができる知識ベースを捜し、当該知識ベースへの関係データベース操作言語に翻訳ゴール列を送る。個々のゴールはPSI やDelta の知識ベースにより解答を与えられる。KIH は、この解答を当該知識ベースから受け取り、ゴール列を出した応用プログラムへ返す。

ここで、他のマシン（計算機）の知識ベースが利用される場合には、Prologのゴール列やその解答は、LAN (Local Area Network) を介して転送される。

KIH は、前期においては、PSI 上のESP により研究試作された。

4. 知識対話機能[5)]

多種類の知識ベースに蓄積されている大量かつ複雑あるいは専門的な知識を、利用者が容易に利用できることは非常に重要である。KCH は利用者と知識ベースとの自然な対話を実現するために、主に次の４つの機能を提供している。

1)制限された日本語文（半自然言語）による入力
2)複数文の表現を許す（代名詞、接続詞、省略表現を制限して許す）
3)話題転換を許す対話管理
4)協調的な応答のための検索機能（検索結果の表示機能も含む）

誰でも容易に使える言葉は自然言語であることは万人の認めるところであろう。しかし、自然言語の文章には、文章を作る人の表現能力に依存して、複雑であったり、曖昧である‘かかりうけ’が存在する。また、多くの情報を１つの文中に表現しようとするために、少し考え込んだり、表現が正しいことをチェックする努力も必要となる。そのような努力が費やされていても自然言語の文の意味は誤解される場合がある。そこで、これらの理由から、個々の文は簡明な表現にして、複数文による表現を許す対話文を用いることとした。これにより、複雑なかかりうけや曖昧なかかりうけを大幅に削減でき、表現および理解の容易な対話が実現可能になる。

また、KCH では、柔軟な対話の進行や目標の高水準の結果への到達のために、話題転換を許す対話管理を行っている。すなわち、予備知識が不足している利用者がそれを調べたり、利用者が高水準の結果へ到達するための関連知識を調べる機会を与えることを可能にしている。

協調的な応答はKCH の主要な目標である。KCH では、そのための検索機能として、次の５つを提供する。

1)条件に合致するデータの検索
2)検索可能の判定　　3)検索結果の説明
4)協調的な返答　　5)絞り込み機能

1)は一般的な検索機能である。KCH では、2)、3)、4)の機能のためにpresumption （思い込み）の処理を行っている。これは、利用者が勝手に思い込んでいたり、入力ミスを行った場合にそれをチェックして適当な指示を利用者に返すための処理である。また、協調的な返答のために関連情報の付加機能も非常に有効である。絞り込み機能は、利用者に大雑把な質問を許すことができるので、利用者を解答数の予測作業から解放する。

5. おわりに

以上に、知識獲得、インタフェース、対話機能を説明した。この他に、知識の蓄積機能も実現されている。これらの研究試作は、学習機能、分散知識ベース機能、知的対話機能、知識蓄積機能の基礎を築くものであると言える。

参考文献

1) H. Kitakami, S. Kunifuji, T. Miyachi, K. Furukawa, A. Takeuchi, T. Miyazaki, S. Ishii, T. Takewake, H. Ohki : Demonstrationb of the KAISER System at the ICOT Open House in FGCS'84, ICOT TM-0081 (1984)

2) T. Miyachi, S. Kunifuji,F. Kitakami,K. Furukawa : A Knowledge Assimilation Method for Logic Databases, New Generation Computing, 2-4, 385/404 (1984)

3) H. Kitakami, S. Kunifuji, T. Miyachi, K. Furukawa : A Methodology for Implementation of A Knowledge Acquisition System, Proc. of the 1984 International Symposium on Logic Progrramming, Atlantic City, U.S.A., 131/142 (1984)

4) 宮地泰造、國藤　進、古川康一、北上　始：Constraintに基づく論理データベース管理について。情処　知識工学と人工知能研究会、1984年 9月。

5) 宮地泰造、伊草ひとみ、近藤省造、太細孝、古川康一：話題管理機能を持つ対話システムの試作。情処知識工学と人工知能研究会、1985年 1月。

6) 國藤　進、北上　始、宮地泰造、古川康一：知識工学の基礎と応用　第４回　Prologにおける知識ベースの管理、計測と制御、Vol.24 No.6 （昭和60年 6月）

関係データベース・マシン（Ｄｅｌｔａ）

角田　健男　（ＩＣＯＴ第一研究室）

１．はじめに

知識ベースマシンは、並列推論マシンと並んで第五世代コンピュータシステムの中核となるハードウェア要素である。前期３年間における知識ベースサブシステムの研究開発は、知識ベース管理機能が論理型言語実行機能と関係データベース管理機能との統合機能であるとする第１次近似モデルに基づいて展開された。すなわち論理型言語におけるFactは、関係データベースとして取扱うことが容易であるため、大量のFactの検索・格納・更新を効率良く処理できるハードウェアとして関係データベースマシンを開発し、一方論理型言語に基づく推論機構を有する逐次型推論マシン(PSI) 内で動作する知識ベース管理ソフトウェア、つまり大量のFactへのアクセスを管理したり、知識の獲得等を管理するソフトウェア(KAISER)の開発が並行して進められた。前期に開発した関係データベースマシン（以降Deltaと呼称する）は、今後PSI との接続に次いで知識ベース管理ソフトウェアとの接続を経て、中期に開発予定の知識ベースマシンに必要な基礎技術を確立させるために各種基礎実験に使用するとともに、複数台のPSI とLAN 経由で接続したソフトウェア開発ツールの一構成要素として中期に使用される[Murakami83]。

本報告書では、Delta で採用したアーキテクチャの概要とDelta のハードウェアおよびソフトウェア構成、昭和59年度中に実施した機能拡充項目およびDelta の試験評価の概要などについて報告する。

２．Delta アーキテクチャ

２．１　ホストからの問合せ処理の流れ

ホストからDelta への問合せをしたときの処理の流れを図 1に示す。ホスト上のユーザはPrologでプログラムを記述し、外部データベースへ問合せを行うものとする。PSI は、ユーザの問合せ内容とPrologプログラムとからDelta への問合せプランを生成し、次いでDelta コマンドに変換し、パケット形成をしたのちLAN インタフェースを介してDelta コマンドを抽出・解析の結果、検索コマンドと判定すると、データベースから所要のデータを検索し、再びLAN インタフェースを介してホストへ情報を送出する。

PSI とDelta の処理の具体例を図２に示す。PSI とDelta には図2(1)のようなPrologの面とリレーションが格納されているとする。いま、PSI より?-息子(S,mary)という問を出したときのPSI とDelta の処理フローを図2(2)に示す。PSI はまずプラン 1と 2を生成し、次いでDelta コマンド①～④に変換し、Delta へ送出する。Delta はデータベースを検索し結果をPSI へ転送する。

この例からもわかるように、Prologで書かれたプログラムの外部データベースへのアクセスの特徴は、次のように考えることができる。

(1) リレーションの内の属性（アトリビュート）への平等なアクセスがなされる。

(2) リレーションへの問合せはselection,join,union等の関係代数演算に変換され、しかも演算負荷の重いjoinやunion の数が多い。

(3) リレーションを構成する属性数は、あまり多くないケースが多い。

２．２　アーキテクチャの特徴

Delta は論理型言語環境下で使用されるため、そのアーキテクチャの設定においては、前節で述べたアクセス特性を考慮に入れ、以下のアーキテクチャ上の特徴を持たせた[Kakuta83],[Shibayama,82,83,84a,84b]。

(1) アトリビュート型内部スキーマ方式

データベースマシンの性能に大きく影響する内部スキーマには、タプル型格納方式とアトリビュート型格納方式とがあるが、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後者の格納方式を以下に示す理由から採用した[Miyazaki83]。

(a) Prologをはじめとする論理型プログラミング環境下でのDelta へのアクセスパターンは、どのアトリビュートも平等にアクセスされる可能性が大きいこと。

(b) 問合せに対し、必要なアトリビュートのみアクセスすればよいこと。

これに対しタプル方格納方式の場合は不必要とするアトリビュートもアクセスの対象となってしまう欠点がある。

(c) マージ・ソートアルゴリズムで実現したREの処理効率、性能上適したデータ格納方式であること。しかし、アトリビュート型格納方式を採用した結果、次に示すデメリットもある。

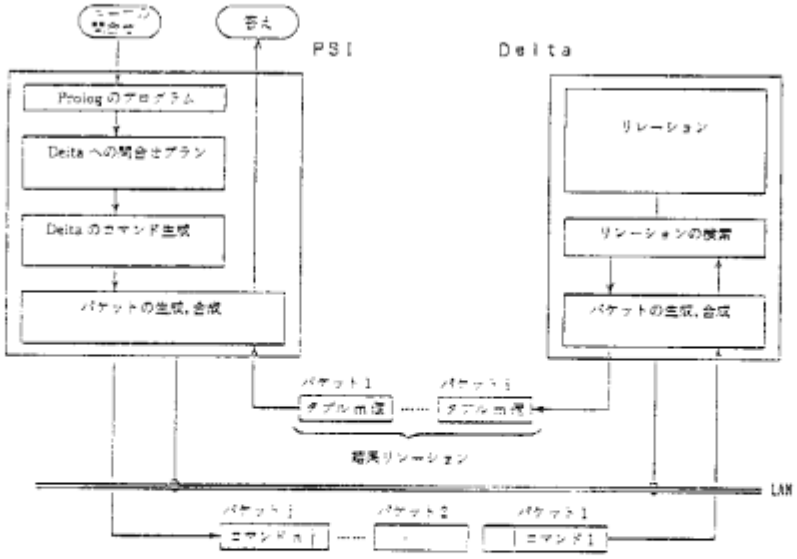


図 1 PSIユーザの外部データベース問合せ処理フロー

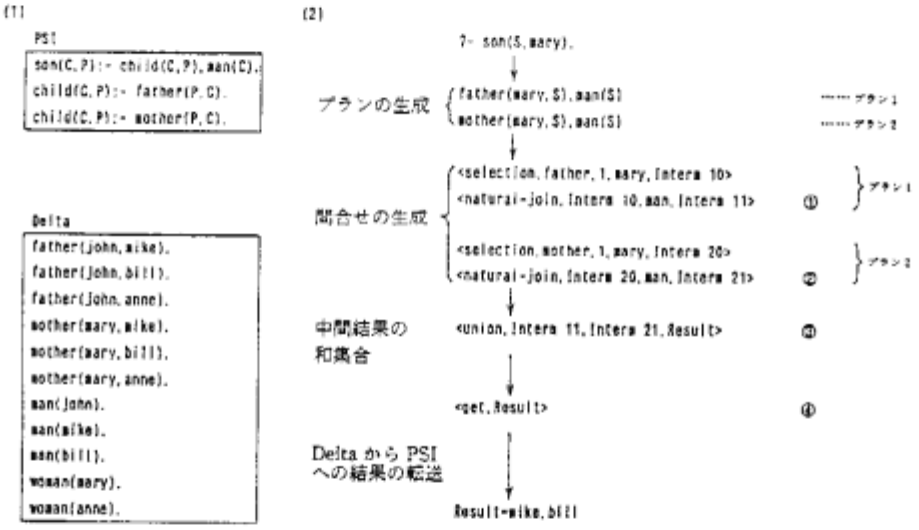


図 2 PSIからDeltaへの問合せ処理例

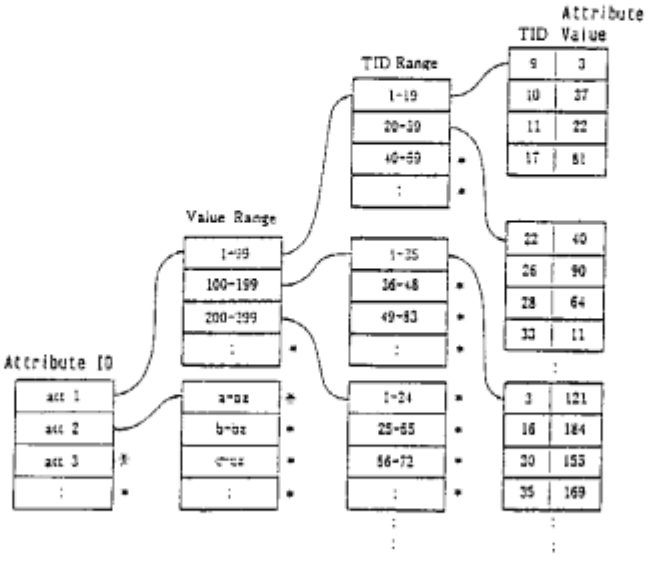


図3 Deltaの内部スキーマ　　注：＊はポインタを示す。

(i) タプル再構成とアトリビュート分割処理が必要であること。

(ii)タプル識別子(TID)を各アトリビュート毎に付してタプルを識別させるため、記憶空間の有効使用が悪いこと。

Delta ではアトリビュート情報のサーチスペースを削減するため、アトリビュート識別番号(Attribute Identifier)、アトリビュート値の範囲とTID の値の範囲をサーチキーとする2段階クラスタリングを採用した。図 3にDelta の内部スキーマを示す。

(2) 関係代数型のコマンドインタフェースの実現

ホストとの論理的コマンドインタフェースとして、関係代数型コマンド(Deltaコマンドと呼称)を選択した。また、ホストとのデータ転送形式はタプル型である。上記Delta コマンドとタプル型データは、LAN プロトコルに従ってホストとDelta 間をパケット形式で転送される。

(3) 関係代数演算用専用エンジンの具備

関係データベースエンジン(RE)は、パイプライン・2ウェイマージ・ソート方式による12段ソート・セルと関係代数演算処理用マージャとから構成され、最大 4台のREがControl Processor(CP) の制御下で並列動作する[Sakai84] 。

(4) 高速大容量メモリユニットを備えた階層構造メモリ

検索・更新対象リレーションのアクセス性能を高めるためリレーションを属性毎に分割格納する可動ヘッドディスク(MHD) 、ディスクキャッシュ・エリアとRSP 間転送用バッファ等から成るデータベースメモリユニット(DMU) 、RSP からの指令によりDMU 上にMHD より読出した属性情報の入るエリアを確保する等のメモリエリア管理や属性ータプル形式変換、属性のクラスタリングインデックス管理等を行いHM制御ソフトウェアが動作するプロセッサ部(HMCTL) 等から構成される。

3. Delta の全体構成

Delta は、全体の制御・監視と演算処理を担当するRSP (RDBM Supervisory and Processing)サブシステムと関係データの格納、検索、変更を担当するHM(Hierarchical Memory) サブシシステムとから構成される。図 4にRDBM Deltaの構成を示す。

HMとRSP とは、合計11本のチャネルインタフェースで接続される。

3. 1 RSP サブシステム構成

RSP サブシステムは、CP,IP,MP,RE から成り、それぞれハードウェアとソフトウェアから構成されている。

(1) RSP ハードウェア構成

各ユニットとも基本部分は 1プロセッサと1MB のメモリとから構成されており、REではソータおよびマージャ専用ハードウェアを、またCPとIPではメモリ容量拡張のため15MBのICバルクメモリを備えている。MPにはDeltaシステム監視用ディスプレイ操作卓、RSP ログ情報収集用HTU が接続される。RSP の各ユニット間は、 3本のIEEE 488バスによって相互接続される。

また、HMとのインタフェース用ハードウェアとして、CP,IP,MPの各ユニットは各々 1個のHMアダプタ(HMA) を、REでは入力、出力用に各1HMAを備えている。

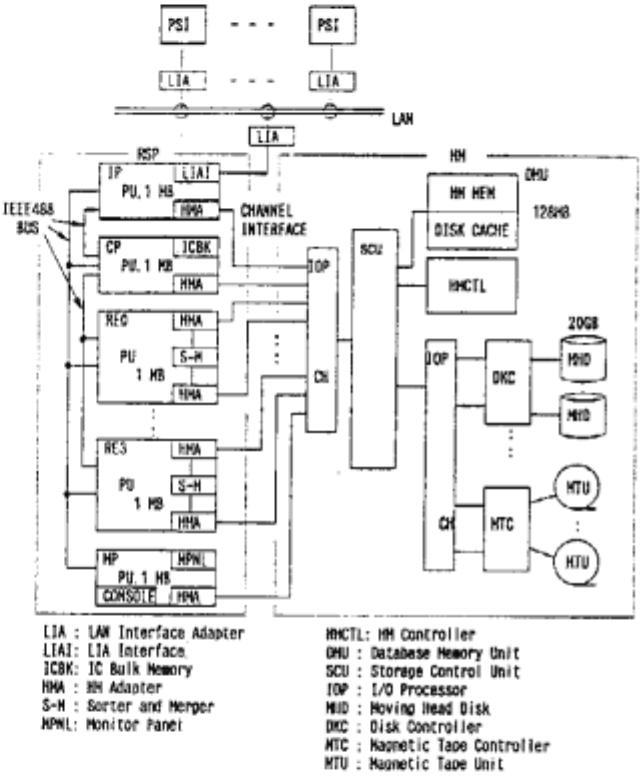


図 4 Delta の構成

(2) RSP ソフトウェア構成

RSP の各ユニットのソフトウェアが分担する機能を以下に記す。

(a) CP

(i) トランザクション管理
トランザクション実行管理とDelta コマンド解析、サブコマンド生成・実行を行う。

(ii) 関係データ管理情報の管理

(iii) 対ＩＰ／ＲＥ／ＭＰ間通信制御

(iv) リカバリ管理
トランザクションを単位としたデータリカバリを行う。

(b) IP

(i) LAN インタフェース制御

(ii) Delta コマンド、データ抽出制御

(iii) コマンド木並列受信制御

コマンド木は、複数子のDelta コマンドから成り、1つの意味のある処理を行う単位である。本制御ソフトウェアは、受信パケット列からトランザクション毎のコマンド木を識別制御する。

(iv) データ形式変換

ホストからデータ入力時にはタプル識別子(TID)を付加し、逆に出力時はTID を削除等の処理を行う。

(v) HM間データ転送

CPがIPとHMとのデータ転送用バッファを確保を指示し、確保完了後は、IPがHM間のデータ転送を行う。

(c) RE

(i) RE制御

CPからのサブコマンドを解析しは、エンジン各部のシーケンス制御とHMとの入出力動作を制御。

(ii) 関係演算サポート

マージャハードウェアでは処理できない算術演算等の演算を行う。

(d) MP

(i) Delta システム状態監視、構成管理

(ii) Delta システム開始、終了制御

(iii) データベースロード・ダンプ制御

(iv) 統計情報収集制御

(v) オペレータコマンド、メッセージ管理

3．2 HMサブシステム構成

HMサブシシステムは、RSP サブシステムの指示のもとにDelta システムで取扱うデータの格納領域への効率良い書込み、および読出し管理を行うためのHMハードウェアとHMソフトウェアとから成る。

(1) HMハードウェア構成

HMハードウェアは、最終構成で128Mバイトの記憶容量を持つ不揮発化した高速DMU と総容量が20G バイトの磁気ディスク装置とこれを制御するディスク制御装置とHM制御プログラムの実行を担当するHMコントローラなどから成る。

(2) HMソフトウェア構成

HMソフトウェアが実行する機能を以下に記す。

(a) HMサブコマンド処理

RSP から指示されたHMサブコマンドにより以下の処理を行う。

(i) アトリビュート定義、操作

アトリビュート定義の生成、消滅およびタプル構成、アトリビュート分割処理を行う。

(ii) 更新処理

アトリビュートに対する挿入、削除、更新処理を行う。

(iii) クラスタリング操作

(iv) データ転送管理

HM－RE間のストリーム・データおよびHM－IP、CP間のパケット・データの転送処理を行う。

(v) RSP バッファ管理

HM内RSP バッファの確保、リリース処理を行う。

(b) メモリ・リソース、MHD スペース管理

アトリビュート情報、ディレクトリ情報のDISKとDMU間転送制御と、これらの情報を格納するメモリ・リソースの管理を行う。

(c) データリカバリ処理

CPからのリカバリ系サブコマンドにより、アトリビュート情報のリカバリ処理を行う。

4．開発システムの前期成果のまとめ

昭和57年 8月より開発を開始した関係データベースマシンDelta は、59年 4月末ハードウェアのICOT搬入ののち、ソフトウェア詳細設計・テストを経て59年11月には検索系コマンドを中心とした基本的動作が稼動できるようになった。その後更新系動作、データベースリカバリ処理、統計情報収集機能等を制御するソフトウェア開発・テストを進めている。

4．1 59年度におけるDelta の機能拡充項目

(1) HMのメモリ容量の拡張

DMU : 16MB → 128MB 、MHD : 5GB → 20GB

(2) 電源障害時のデータベース破壊防止のための無停電電源装置を増設

(3) 関係データベースエンジンを1台から4台に増設

(4) チップ化ソート部を関係データベースエンジンに実装

(5) RSP の各プロセッサの高速化と外部記憶容量の拡張

(6) 障害処理、データベースリカバリ処理などのRAS 機能の強化

(7) 統計情報収集機能の追加

5．Delta システムの評価

Delta の第一次評価として、Delta をLAN を介した代替ホストマシンと接続し、代替ホストからDelta 内データベースへ問合わせを行ない、基本的機能が満たされているかどうかのテストを行った。

5．1　テスト環境

(1) テストシステム構成

代替ホストマシンとして3台のパーソナルコンピュータを使用し、そのうちの1台で評価プログラムを動作させ、残り2台にDelta へのアクセス及び結果等の表示をサポートするプログラムの処理を分担させた。

(2) 評価用プログラム

Prologで記述した2種類の評価プログラムを作成した。

①評価プログラムI

一般的な関係データベース問合せ言語IQL(Interacive Query Language) による問合せを行うプログラム。

②評価プログラムII

PrologにおけるFactをDelta に格納し、Prologの問合せに対して代替ホストでDelta コマンド列を生成して推論を進めるプログラムで、このプログラムが動作するシステムをIRIS(Inference/Relational-database Interface System)と名づけた。

(3) 評価用データベース

機能試験を主目的とするため現実のデータベースに近いものとして観光地データベース（4リレーション）と検索結果の画像表示用として地図データベース（2リレーション）を準備した。またPrologからの問合せでは再帰的な問合せについての評価が重要であるため、各観光地間の隣接関係を示す“neighbor”リレーションを設けた。

5．2　評価結果

上記の評価システムにおいて問合せ処理が正しく実行されることを確認した。またDelta コマンド体系については、その表現能力や簡略さなど妥当性を確認した。

昭和59年度時点におけるDelta の処理性能特性を図 5に示す。問合せQ1～Q8に対する評価システム全体の応答時間とDelta の内部処理時間の相対値とサブコマンドの数との関係はほぼリニアに増加することがわかった。Deltaの基本機能が完全に稼動した時点で更に大きなデータベースに対しての問合せに対しての性能を今後評価してゆく予定である。

6．今後の課題

現在Delta を構成する両サブシステムの制御ソフトウェアの統合試験を継続中であり、今後は、

(1) Delta システム性能評価及び性能改善、機能拡充

(2) LAN 経由でのPSI 接続試験、特にKAISERとの接続試験の実施

(3) Delta アーキテクチャの妥当性の評価及び改善提案

(4) 4台のRDBEの並列動作アルゴリズム実験及び評価

(5) PSI-Delta の密結合インタフェース実験

(6) Delta にFactとRuleを格納し、推論エンジンを直結した知識ベース管理機能についての実験環境の作成並びに実験[Yokota 84a,84b]

などを行う予定である。

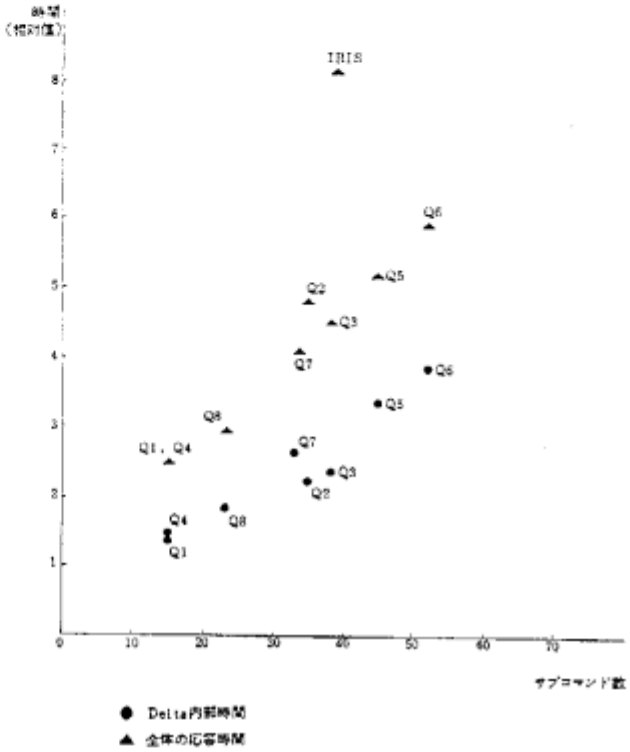


図5　Delta の性能特性

参考文献

[Kakuta 83] Kakuta, T. et al. RDBM Delta (I),(II) and (III), Proc. of 29th National Conf. of IPSJ, Mar. 1983 (in Japanese), and also ICOT Technical Memorandum TM-0008.

[Miyazaki 83] 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、情処学会第27回全国大会、1983.10

[Murakami 83] Murakami, K. et al. A Relational Database Machine:First step to Knowledge Base Machine, Proc. of 10th Symposium on Computer Architecture, Stockholm, Sweden, June 1983. Also available as ICOT TR-012.

[Sakai 84] Sakai H. et al. Disign and Implementation of the Relational Database Engine, Proc. of Int'l Conf. of Fifth Generation Computer Systems 1984, Nov. 1984, and also ICOT TR-063.

[Shibayama 82] Shibayama, S. et al. A Relational Database Machine"Delta", ICOT TM-0003, Nov. 1982.

[Shibayama 83] Shibayama, S.et al. On RDBM Delta's

Relational Algebra Processing Algorithm, Proc. of 27th National Conf. of IPSJ, Oct. 1983, and also ICOT TM-0023.

[Shibayama 84a] Shibayama, S. et al. A Relational Database Machine with Large Semiconductor, Disk and Hardware Relational Algebra Processor, New Generation Computing, 2, No.2,1984.

[Shibayama 84b] Shibayama, S. et al. A Relational Database Processing on an Attribute-based Schema, Proc. of IPSJ, Sep. 1984.

[Yokota 84a] Yokota, H. et al. An Enhanced Inference Mechanism for Generating Relational Algebra Queries, Proc. of 3rd ACM SIGACT-SIGMOD Symposium on Priciples of database systems, Apr. 1984, and also ICOT TR-026.

[Yokota 84a] Yokota, H. et al. Unification in a Knowledge Base Machine, Proc. of 29th National Conf. IPSJ. Sep. 1984.

関係データベース・エンジンの開発

柴山茂樹（ICOT第一研究室）

1．はじめに

関係データベース・エンジン（RDBE）は、関係データベースマシン Deltaの主要な演算ユニットであり、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なう事により、関係データベースの処理が効率よく行なわれるよう設計されている。関係データベースの処理を高速に実行するためには、ソート、セレクトやジョインなどの関係データベース演算を高速に行なう必要がある。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なうハードウェアを設計し、これを汎用CPU の制御プログラムで制御する方式を採用した。階層的な構造をもつ階層構造メモリ・サブシステム(HM)との結合により、その高速性が一部確認されている。

2．RDBEの設計思想

RDBEは、以下の点を重要なポイントとして設計された。

・ハードウェアによる高速な関係演算処理

ジョインや、セレクションなどの、処理負荷が重い演算をハードウェアで効率良く処理すること。

・ストリーム処理をベース

関係は、アトリビュートに独立して格納されている。各アトリビュートは、ストリームの形で階層構造メモリとRDBEの間でやりとりされる。

・現実の関係データベース処理のサポート

単なるアルゴリズムの検証用ではなく、現実のデータベース処理に供する事ができること。

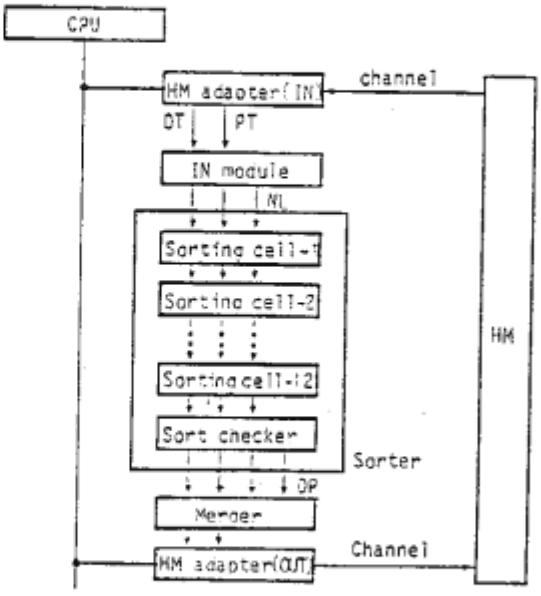
3．処理アルゴリズム

ジョインやプロジェクションは、対象アトリビュートのストリームがソートされていると処理が簡単になる。また、ソーティングは、パイプライン・ソーティングなどの効率的な（ハードウェア）アルゴリズムが知られている。従って、関係データベース処理を、まず対象となるストリームをソートしてから行なうことは自然な発想であると考えられる。これが現実的に可能になる条件は、(1) データをオーダ Nの時間でソートできるアルゴリズムが存在すること、(2) 関係データベース処理のアルゴリズムが、ソートされたデータ列に対して、自然なものであること、(3) ストリームを入出力する高速のバッファが用意されていること、等が挙げられる。我々は、pipeline 2-way merge sort [1] のソーティング・アルゴリズムをインプリメントしてソータ部を構成し、その後段に、ソーティング部のセルと類似のマージャを置き、ソート部より送られるストリームに対してディレイなくデータベース処理を行なえる構成を取った。こうする事により，種々のセレクションは、ストリームを一度RDBEを通して流すことで処理でき、ジョインも、一方のアトリビュートがマージャの内部メモリに納まるときは、もう一方のアトリビュートの長さにかかわらず、オーダ Nの時間で処理できる特性を持つ。RDBEに対するバッファ部は、 HM の、データベースメモリユニット（大容量半導体バッファ）が分担する。

4．RDBEの構成

RDBEの構成を図．1に示す。ソータとマージャはRDBEの基本演算処理部である。HMに接続するためのアダプタが、入出力独立に備えられており、同時動作が可能である。INモジュールは、ソーティング用の前処理回路であり、パイプラインソートが可能な用にキー・フィールドをタプルの先頭に持ってくる機能、データ・タイプの変換機能、NULL値を判別する機能を実現している。汎用CPU 部は、RDBEへのコマンドの解釈を行ない、ハードウェア部を制御する。また、ハードウェアでサポートされないコマンドは、このCPU で処理される。そのようなコマンドには、算術演算、複雑な条件を持つセレクション、集約演算、等がある。

この構成で特徴的なのは、HMアダプタ、INモジュール、ソータ、マージャ、HMアダプタが直列に接続され、モジュール毎のパイプライン処理を可能にしている事である。我々が、



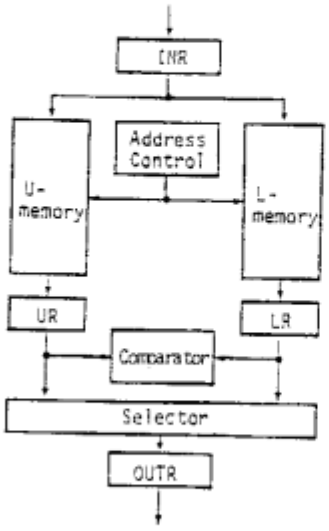
図．1　RDBEの構成

パイプライン・ソートを採用した最も大きな理由は、この構成のようにカスケードに処理モジュールを並べることによって、データを一度RDBEに通すことで処理を行なうことが可能だからである。各モジュール毎では機能レベルのパイプライン処理が行なわれる。機能レベルのパイプライン処理とは、例えばメモリへのリード、ライト、および比較などである。

INモジュールでは、入力されたデータをキー・フィールドが先頭に来るようにならべかえを行なっている。これは、このアルゴリズムを採用してタプル・ソートをするために必要な前処理である。キー・フィールドが先頭にないと、ソート・セルのマージ動作でそのタプルをマージする際に、先頭部分にある、キーでない部分を一時的に格納するバッファがセル毎に必要になってしまう。また、ソータのハードウェアを簡単にするために、扱うデータ・タイプは絶対値の整数に限っている。これ以外のデータ・タイプのストリームは、やはりINモジュールでデータ変換される。これらの逆変換はマージャの出力制御部で行なわれる。RDBEで扱う事のできるデータ・タイプは、絶対値整数、符号付整数、浮動少数点数、文字(ASCII,JIS)、である。Delta ではさらに幾つかのデータ・タイプをサポートしているが、絶対値整数と同じ扱いにできるものはソータでは区別する必要がない。

4．1　ソータ

ソータは、12段のソーティング・ステージとソート・チェッカにより構成されている。各々のソーティング・ステージは、ソーティング・セルと呼ばれる特殊なハードウェアユニットと、 RAMにより実現された2個のFIFOメモリで構成されている。ソーティング・セルの構成を図．2に示す。ソーティング・セルは比較器および制御部により構成されている。ソーティング・セルは全部が同期して動かされる。 i番目のセルは一般に2^{i} の長さの2つのストリームを入力され、マージによって1つの2^{i+1} の長さのストリームを生成し、次のステージに送りだす。



図．2　ソーティング・セルの構成

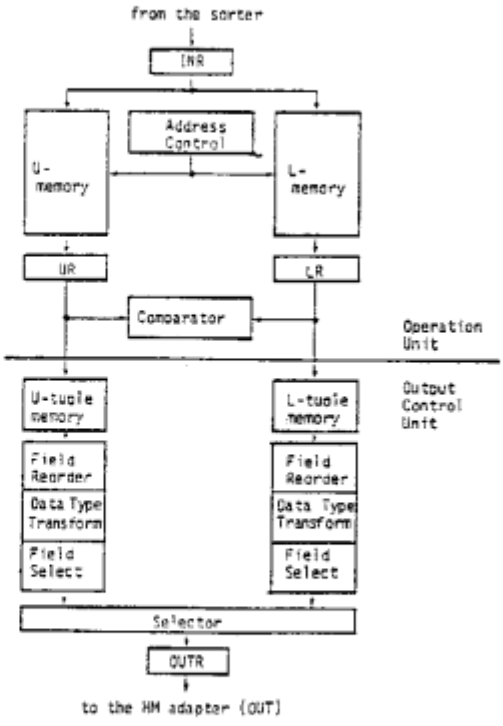
最初の1段目のセルの入力はINモジュールに接続され、最後の12段目のセルの出力はマージャに接続されている。すべてのセルとチェッカはI/O バスによりCPU の制御下にあり、制御パラメータの転送、ハードウェアの起動が行なわれる。パラメータの中にはタプル長、キー・フィールドの長さ、タプル数、動作モードの設定などがある。

内部ソート可能な、最大ソート長は2^{12} = 4096 （アイテム）となる。しかし、最大ソート長は、FIFOメモリの大きさにも依存する。実現値は最終段のFIFOメモリが64KBなので、アイテム数が4096より小さくても、その容量が64KBをこえる物については一度にソートはできない。

4．2　マージャの構成

マージャの構成を図．3に示す。マージャは比較部と出力制御部に分れている。比較部は2つの64KBのメモリと、比較器、制御部よりなる。比較部はソート・セルと類似の構成であり、コマンドにより2つのメモリから同時にデータを読みだし、条件を満たすタプルを、次段の出力制御部に送る。出力制御部では、2つの16KBのタプル・メモリ、フィールド再配置回路、フィールド選択回路、データ・タイプ変換回路、新TID 付加回路があり、前段より送られてきたタプルのフィールドの選択、フィールドの切出し、データ・タイプの逆変換、必要ならば新TID の付加を行い出力をする。これらの回路の制御は制御ROM により行なわれる。

マージャの関係データベース演算の本質は、マージ動作（比較動作）により選択されたストリームを、出力を制御して選び出す事にある。例えば、セレクションは、マージする一方のストリームを比較する条件となるデータと考える事に



図．3　マージャの構成

より、実行できる。ジョインの場合は一方のストリームが1つのアトリビュートであると考えれば、マージの演算自身はセレクションと同じに扱える。出力制御部は比較によって選ばれて送られてきたストリームを、演算の種類、出力条件にマッチするように整形、変換する。ジョインの場合、2つのストリームの中で重複した値が現れると、定義によりその重複した値どうしのデカルト積を出力しなくてはならない。この為に用いられるのが重複フラグである。重複フラグは、ソート・チェッカで立てられ、DPラインによって伝達される。この時、出力がボトルネックになるので入力側のストリームは一時、停止される。

マージャの演算の基本は述べたようにソートしたストリームに対して行なわれるが、常にあるアトリビュートやリレーションをすべてソートしなければいけない訳ではない。セレクションは、一方の条件となるストリームが、マージャのFIFOメモリに格納できれば、選択されるストリームは、ソータで内部ソートできるだけの容量づつ送られてくれば実行できる。ジョインも、一方の短いストリームがマージャのFIFOメモリに格納できれば、もう一方の長いストリームは、ソータの容量づつ送られる事により、一回だけRDBEに通すだけでジョインが完了する。一方のストリームがFIFOメモリに入り切らない時は、ジョインの演算はのオーダは、O(N*M) となる。つまり、マージャの演算時間は、ストリームが何回RDBEを通るかによって決まる。

5. 性能予測

RDBEの性能はDelta の演算の性能を予測する上で基本と考えられる。Delta 全体の性能評価は、中期の一つの課題と考えているが、RDBEの性能は用いるアルゴリズムから典型的な場合には比較的容易に算出できる。最も基本的なソート処理の性能予測をここで行なう。まず、ソートのパラメータを次に示す。

N: ソーティング・セル数
M: マージャのメモリ・キャパシティ
C: ストリーム中のタプル数
L: タプル長
T: 1 byte の転送スピード

ソートの処理時間はデータ長により、このソータに対して内部ソートになるか外部ソートになるかによって異なる。

まず、有効ソータ・キャパシティ,E, を次のように定義する。

$E = \min(2^N * L, M)$

E は内部ソートに有効に用いられるソータの容量を表わしている。次に、マージ回数,F, が次のように定義できる。(対数の底は、2である。)

$F = \lceil \log(CL/E) \rceil$

F は外部ソートの際のマージの回数を表わす。

(1) F = 1 の場合

このとき、ストリームは一回の内部ソートで処理でき、

Tsort = (2CL + Llog(C))T となる。

(2) F = 1 の場合

一回の外部ソートが必要だが、最初のストリームは、ソートされた直後にマージャのFIFOメモリにロードできるので、再転送は省略できる。

Tsort = (3E + log(E) + 2(CL-E) + log(CL-E))T

(3) F > 1 の場合

二回以上の外部ソートが必要となり、またマージの時間はデータの内容に依存するので、概略の時間は、

Tsort = 2.5CLT + 1.5CLTF となる。

これらの値は、 HM の半導体バッファに、ストリームが用意されてからのものであり、かつRDBEのCPU によるコマンドの解釈、ハードウェアの起動の時間などのいわゆるオーバヘッドを含んでいないので、ソート時間の下限を表わしている。図. 4に、N = 12、 M = 64KB 、 L = 16 、 T = 1/(3MB/sec) = $0.33*10^{-6}$の場合の、性能予測のグラフを示す。

実験的なRDBEと HM を用いたソート時間の測定の結果では、データ量が小さい時のオーバヘッドは上記の式に対して10msecのオーダ、データ量が多い時は、30から40% となっている。この評価は中間的なものであり、これからDelta 全体の評価の一環としてよりシステマティックなRDBEの評価も行なっていくつもりである。

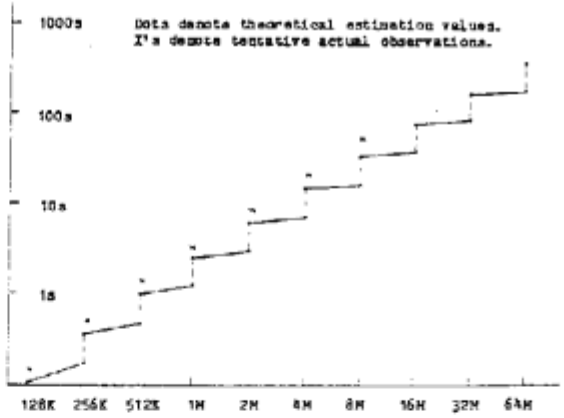


図. 4 ソートの性能予測

6. 結論

Delta の演算ユニットであるRDBEについて、そのアルゴリズム、構成、実現、性能の予測について述べた。本格的な評価はこれからの課題であるが、仮の評価でもその高速性が一部確認されている。

7. 謝辞

Delta の開発に当たって、ICOTとのメンバと協力し、かつインプリメントを担当された東芝、日立の研究者、技術者の各位に深謝する。

[参考文献]

[1]Todd, S., "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM J. Res. Dev., 22, 1978

自然言語処理（DUALS：談話理解実験システム）

向井国昭　（ICOT　第3研究室）

1．概要

会話文を含む文章の意味を理解し、その内容に関する質問に答えるシステムDUALSを試作実験したのでそれについて報告する。構文解析はLFG(Lexical Functional Grammar)を用い、意味解析と語用解析の考え方は状況意味論（Situation Semantics)を用いた。解析の例題は小学校3年の国語の文章読解ドリルの中から選んだ。

本試作の目的は文脈理解までを含めた意味解析モデルを作ることと、それをロジック・プログラミングで記述することである。状況意味論がこのふたつを結び付ける有効なアイデアであることを確認した。

2．状況意味論

状況意味論は自然言語の新しいモデル論的な意味論であり、自然言語のもつ文脈依存性を非常に良く説明する。会話の関連性（relevancy)、coherency 、presuppositionなどの文脈処理に対しても有効である。以下に状況意味論の基本的なアイデアを示す（詳細は“J. Barwise & J. Perry, Situations and Attitudes, MIT Press, 1983”を参照のこと）。

① 世界は状況の集りである。
② 文の意味〖文〗は状況間の関係である：
　〖文〗（談話状況，話者結合状況，記述状況）
③ 状況は事実の集合である。
④ 事実は次の形である：
　$(\ell, r, a_1, \cdots\cdots, a_n, p)$
　ℓ：時空領域　a_i：オブジェクト
　r：n項関係　p：1または0
⑤ 状況は“タイプ”を持つ。
⑥ （一般の）“タイプ”は次の形で表せる：
　〔不定項｜条件〕

3．システムの構成

（略、別資料参照）

4．構文解析

文法記述に、LFG を拡張したものを使った。これは拡張ユニフィケーションを持つ論理型言語と考えられる。構文解析にはBUP を使用している。

文法は、規則数60、語彙数 100程度のもので、日本語の基本的な構造を記述している。文法の特徴としては、受動化・使役化あるいはコントロール現象などを語彙レベルで一般的に記述した点と、日本語文に対しフラットな構造を仮定した点が挙げられる。文法がカバーする範囲は、一部の法助動詞・否定を含む基本的な日本語文で、疑問文・一部の等位接続文・一部の条件文を含んでいる。名詞句に関しては、一部の限量化名詞句・複合名詞句・派生名詞などを扱っている。

今後の課題としては、一般的な等位構造や条件文の扱い、法助動詞・副詞句・再帰代名詞などの扱い、敬語表現の扱いなどがある。

5．意味解析

意味の表現は、状況意味論のタイプ・システムに基づいており、文法規則に付記されたタイプ間のユニフィケーションにより、構文解析中に意味表現を合成する。

固有名詞・代名詞（省略を含む）・一部の限量化名詞句・一部の複合名詞句・派生名詞などの名詞類の意味解析と、平叙文・疑問文・一部の条件文・一部の等位接続文などの基本的な文構造の意味解析を行う。また、副詞や法助動詞の意味解析も一部行っている。

今後の課題としては、法助動詞・副詞句一般の解析、一般の条件文の解析、あるいは、名詞句のさまざまな用法や心的態度にからむ動詞の意味の解析が挙げられる。

6．オブジェクト同定

省略（ゼロ代名詞）と照応（代名詞）の同定処理を行なう。日本語の特徴のひとつとしてゼロ代名詞が頻繁に使用される。状況意味論の枠組で言えば、資源状況および話者結合状況の生成と管理が処理内容である。出力は等式のリストである。それは何と何が同じものであると同定されたかを示す情報である。

処理方式は“仮説を生成せよ・そしてテストせよ”である。仮説の生成にはfocus centerの理論を用いている。目は顔の一部であるという全体－部分に関する知識やシソーラス情報を用いたり、文脈を参照したりする。次はゼロ代名詞の文脈参照用法の例である：

太郎は花子を見た。（ゆは）本を読んでいた。

一方、テストは、coherency のチェックによっている。

7．談話構造解析

談話構造は入力文章が記述するオブジェクトと、それらに関する状況とから成る情報を表す記憶構造であり、一文ごとに追加・更新される。本実験においては談話構造は簡単のためにオブジェクト同定モジュールの出力の集合と同一視している。しかしながら談話構造にテンプレートを定義しておき、それに従って予測的に解析する試みは独立して行なった。そのテンプレートの内容は次のとおりである。

① 主人公は行動目的を持つ。
② その目的の障害イベントが発生する。
③ 主人公は目的と計画を変更する。
④ 主人公は計画を実行する。

テンプレートには、与えられた文脈からこれらの情報を見出すためのヒューリスティクスが記述されている。

8．問題解決

問題解決のベースとなる知識は約40のホーン節で表現されている。それに加えて「風が吹けば桶屋が儲かる」式の因果関係が次のルールで表現されている：

```
causal_chain(A,B) :-  cause(A,B).
causal_chain(A,B) :-  cause(A,C),
                      causal_chain(C,B).
```

飛行機の下が海であれば飛行機は海の上を飛んでいることや、乗客にエンジンから煙が吹き出したことを知らせれば乗客がパニックになることが推論できる。後者の推論にはAがBにPを知らせればBはPを知るなどの、情報に関する一般知識も用いている。

9．文合成

動詞の格パターンに表層文のパターンを記述しておくことにより表層文を生成している：

「吹き出す」→ "AがBから吹き出す"

10．シソーラス

単語を上位－下位の階層関係で分類した辞書である。オブジェクト同定のひとつの基準として、与えられたふたつのオブジェクトが同じ意味カテゴリに属すか否かを調べるのに用いられている。

12．CIL

この実験を通じてタイプと不定項が談話解析の記述において有効であることが分った。そこでPrologにこのふたつを導入すれば強力な談話解析用の記述言語が得られるはずである。このような予想のもとにCIL なるPrologの拡張言語を試作した。CIL は構文、意味、語用処理および知識表現言語としてのPrologの記述パワーを大きく向上させるものである。なお、DUALS の談話構造解析および問題解決モジュールはCIL の初期の版により記述されている。

13．実現

試作版はDEC-2060 Prolog ですべて作成された。システム全体がコアに入り切らず、LFG パーザ部分とその他の部分とを別々のシステムとし、データはファイル渡しとした。

1文当りの所要CPU 時間は以下のとおりであった：

構文・意味解析（LFG）	……	約10秒
オブジェクト同定	……	約 1秒
談話解析・問題解析・文合成	……	約 1秒

例題として用いた物語と質問は次のとおりである：

つぎのお話を読んで、あとのといに答えなさい。

あと1時間でマニラに着こうという時、どうしたことか、急に、エンジンから白いけむりがふき出しました。これを発見したき長のロールさんは、はっとしました。もしも火でもふこうものなら、ひこうきは、ばく発してしまいます。下は広々と広がる太平洋です。そうしたら、50人の乗客のいのちはどうなることでしょう。

ロールさんは、急いで、スチュワーデスのふちがみさんをよびました。そして、いざというときの用意をするようにめいじました。ふちがみさんの顔がひきしまりました。「お客様にお知らせするんですか。」「いや、なんとか、このままとんでみる。お客様には、知らせないほうがいい。」ふちがみさんは、そうじゅう室を出ると、にっこりとほほえみながら、「みなさん、これから、きゅうめいぐをつけるくんれんをいたします。」と言いました。

(1) いま、ひこうきは、どこの上をとんでいますか。
(2) 乗客は何人ですか。
(3) き長さんは、だれですか。
(4) スチュワーデスは、だれですか。
(5) ロールさんは、何を発見したのですか。
(6) ロールさんが「はっとした」のはなぜですか。
(7) ロールさんが、お客に知らせないほうがいいといったのは、なぜだと思いますか。

「小学3年全国標準テスト…国語」
受験研究社刊より

14．結論

状況意味論ベースの文脈モデルが有望であることを、談話理解システムDUALS の試作実験により確かめた。そのモデルが構文、意味、語用論そして知識表現までも含めてロジック・プログラミングと親和性が高いことが確認できた。

Prologプログラムの検証システムについて

金森　亘（三菱電機中央研究所）

要約．論理プログラミングのパラダイムにより、Prologプログラムの検証はデバッギングの延長上の一般的なプログラム確認の方法である。

1．はじめに

人はしばしば誤ったプログラムを書く。例えばリストを逆順に並べるプログラムを次のように書いてしまう。

```
reverse([],[]).
reverse([X | L],M):-reverse(L,N),append(N,X,M).
```

図１．プログラマの意図に反したプログラム

どうすればプログラムが意図したものであることを確かめることができるだろうか。世の中にはデバッギングと検証という対照的な印象を与える方法が知られている。デバッギングでは適当なテストデータを与えて実行し、おかしな所がないかを調べる。より多くのテストデータについておかしな所がなくなれば（だいたい）より良くプログラマの意図に合っているが、非形式的で泥臭い感じをもたれている。これに対し、検証は何をするか(what)を示す仕様をいかに計算するか（how)を示すプログラムに対して数理論理的に証明すると言う。仕様が証明されればそれで正しいことがわかるが、形式的・数学的で機械にやらせるものと思われている。一見したところ全く隔たったこの二つのアプローチは論理プログラミングでは実は同一線上にある。

2．Prologプログラムのデバッギング

図１のPrologプログラムをトレースしてみよう。トレーサと呼ばれるデバッギング・ツールが次のように応答する。

```
| ?- reverse([2,1],M).
   (1) 0 Call : reverse([2,1],_40) ?
   (2) 1 Call : reverse([1],_105) ?
   (3) 2 Call : reverse([ ],_117) ?
   (3) 2 Exit : reverse([ ],[ ])
   (4) 2 Call : append([ ], 1,_105) ?
   (4) 2 Call : append([ ],1,1)
```

図２．Prologプログラムのデバッギング例

図中であるゴールが成り立つには下の（Callで呼ばれる）ゴールが成り立てばよい。図２のトレースからプログラマは次のような意図に合ったプログラムに修正する。

```
reverse([],[]).
reverse([X | L],M):-reverse(L,N),append(N,[X],M).
```

図３．プログラマの意図に合ったプログラム

結局デバッギングとは“プログラマが頭の中でいだいているモデルとプログラムの動作にくい違いがないかを確認する”ことである。

3．論理プログラミングのパラダイム

ここで第５世代計算機システムの中心的理念である論理プログラミングのパラダイムを思い起こしてみよう。“Prologプログラムの実行は論理的証明である”。上図の?-reverse([2,1],M). の実行は$\exists$M reverse([2,1],M)の証明である。一般にPrologプログラム“ B:- B_1 ,B_2 , …,B_n ”に対して次の推論規則が対応する。（横線の下が成り立つには上が成り立てばよいと読む。）

$$\text{実行}\quad \frac{\sigma\,(\,B_1 \wedge B_2 \wedge \cdots \wedge B_n\,)}{\sigma(B)}$$

図４．実行のための推論規則

証明したい論理式に対して下の方から証明図が作られるが(下図参照)トレーサはそれを上下逆様に表示する。

$$\cfrac{\cfrac{\text{reverse}([],[]) \wedge \text{append}([],1,1)}{\text{reverse}([1],1) \wedge \text{append}(1,2,?)}}{\text{reverse}([2,1],?)}$$

図５．実行に対応する証明図

このパラダイムによればデバッギングで注目する“動作”とは $\exists Y_1\ Y_2 \cdots Y_n\ (A_1 \wedge A_2 \wedge \cdots \wedge A_k)$ の形の性質を証明するときの様子に他ならない。

4. Prologプログラムの検証

この見方を延長してプログラムの検証を“プログラマが頭の中でいだいているモデルとプログラムの論理的性質にくい違いがないかを確認する”ことと考えてみよう。ここで論理的性質とは例えば“リストを逆順に並べた後、さらに逆順に並べると元に戻る”

$\forall$ L,H(reverse(L,H)$\supset$reverse(H,L))

など $\forall X_1\ X_2 \cdots X_n\ \exists Y_1\ Y_2 \cdots Y_m$ (n,m≥0) の形の論理式で示されるものする。注目すべきはここでは仕様は必ずしもプログラムが何をするか全体を示したものでなくともよいこと、成り立つべきより多くの性質を検証すれば、(だいたい)より良くプログラマのいだいているモデルと一致することである。これはデバッギングの印象に近い。

しかしこのように検証の定義を単に言い換えただけでは意味がない。このような論理式を普通のPrologシステムは実行することはできない。しかし下図のような推論規則を採用すると実はこのような論理式も“実行”できる。この推論規則が我々の新しいアイデアである。

拡張実行

場合分けの規則(3つ)

正のゴールの実行 $\dfrac{\sigma(G[B_1 \wedge B_2 \wedge \cdots \wedge B_m])}{\sigma(G_+[B])}$

負のゴールの実行 $\dfrac{\sigma_1(G[B_{11} \wedge B_{12} \wedge \cdots]) \cdots \sigma_k(G[B_{k1} \wedge B_{k2} \wedge \cdots])}{G_-[A]}$

真偽による単純化 $\dfrac{\sigma(G(\mathrm{true}))\quad \sigma(G(\mathrm{false}))}{G}$

帰納法

図6. 検証のための推論規則

reverse(H,[X | L1])∧append(L1,[X],L)
 ⊃ reverse([Y | H], [X | L])
↓ 正のゴールの実行
reverse(H,[X | L1])∧append(L1,[X],L)
 ⊃∃L2(reverse(H,L2)∧append(L2,[Y],[X | L]))
↓ 正のゴールの実行 (L2←[X | L1'])
reverse(H,[X | L1])∧append(L1,[Y],L)
 ⊃∃L1'(reverse(H,[X | L1'])∧append(L1',[Y],L))
↓ 真偽による単純化 (L1' ←L1)
append(L1,[Y],L)⊃append(L1,[Y],L)
↓ 真偽による単純化
true

図7. Prologプログラムの検証例

拡張実行のうち正のゴールの実行は先ほどの実行とよく似ている。実際、普通の実行ゴールは全く同じように“実行”され、一般化になっている。拡張実行についてはあとは図7の検証システムの応答の一部に説明を譲る。先ほどのトレースと同様に図中のゴールが成り立つにはその下のゴールが成り立てばよい。推論規則が増えただけ、どの規則を使うかに多少の工夫(ヒューリスティクス)が要る。

最後の推論規則“帰納法”について一言説明しよう。先ほどの性質はどんな長さのリストについても成り立つが、そういった性質を示すには帰納法が要る。よく知られた自然数の帰納法で $\forall X$ (自然数(X)⊃ 性質(X))を示すには

Base Case : 性質(0)
Induction Step : 性質(X)⊃ 性質(X+1)

を示せばよいがこれは次の自然数の定義で“自然数”を“性質”で置き換えたものである。

```
自然数(0).
自然数(X+1):- 自然数(X).
```

図8. 自然数を定義するPrologプログラム

同様に$\forall$ L,H(reverse(L,H)⊃ 性質(L,H))を示すには図3のreverse のプログラムで“reverse ”を“性質”で置き換えればよい。

5. むすび

デバッギングと検証はともにプログラムがプログラマのいだいているモデルと一致するかを確かめる方法である。検証は少し間接的な性質を証明し、そのProverは言わばプログラムにさしこむProberである。昨年度開発されたPrologプログラムの検証システムはそのような性質を自動的に証明するものである。(詳細は以下の文献を御覧下さい。)

文献

[1] Kanamori,T.and H.Seki, “ Verification of Prolog Programs Using An Extension of Execution ” , ICOT TR-096,1984.

[2] Kanamori,T.and H.Fujita, “Formulation of Induction formulas in Verification of Prolog Programs” ICOT TR-094,1984.

[3] Kanamori,T.and K.Horiuchi, “ Type Inference in Prolog and Its Applications,” ICOT TR-095,1984, to appear IJCAI-85.

[4] Seki,H.and T.Kanamori, “ Incorporating Generalization Heuristics into Verification of Prolog Programs” , ICOT TR-108,1985,to appear IJCAI-85.

知的ＣＡＤ実験システム

丸山文宏（富士通株式会社）

1. はじめに

知的ＣＡＤ実験システムは，ハードウェア論理設計の分野を対象とする知識ベースシステムであり，知識利用システムの研究の一環として，Prologによる試作及びその評価を行ってきた。

コンピュータを始めとするハードウェアの設計はおよそ次のように行われると見ることができる。まず，仕様を決定し，その仕様を満たすようなハードウェアの動作を設定する（機能設計）。この動作を実際に実現する回路の設計を行い（回路設計），配置・配線等の実装設計を経て製造工程に進む。本システムは機能設計と回路設計を含せた論理設計の過程をカバーする。このようなシステムにおいては，設計者のノウハウを有効に利用するとともに，従来のＣＡＤシステムによって支援されてきたアルゴリズムに基づく処理もうまく組み入れることが重要である。本研究の目的は，論理型言語の上に知識ベース及びアルゴリズミックな処理を統合した知的ＣＡＤシステム実現の可能性を探ること，及び，それを通して論理型言語の評価を行うことである。

2. システム構成

システム構成を図１に示す。

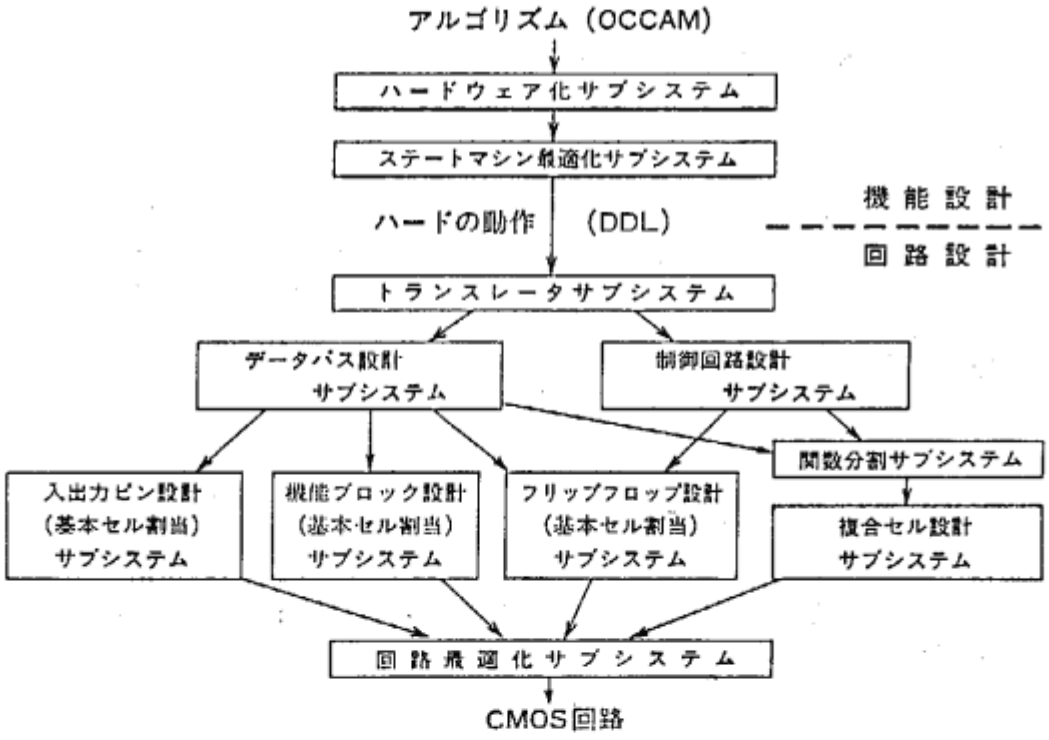


図１　システム構成

本システムでは，並列性を表現し易いように設計されたプログラミング言語occam で仕様となるハードウェアの動作アルゴリズムを記述する。この段階では，ユーザはアルゴリズムとそれを実現するハードウェアとの対応を意識する必要はなく，設計の高水準化が可能になる。ハードウェア化サブシステムは，このアルゴリズムからできる限り並列性を抽き出しハードウェアと対応付ける。この際，知識ベースに格納されたハードウェア機能設計に関する知識が利用される。ステートマシン最適化サブシステムによって最適化された情報はハードウェアの振舞いの記述であり，ハードウェア記述言語ＤＤＬによるステートマシンとして表現される。このレベルでは，ハードウェアの動作については明確になっているが，それを実現する回路の構造については何も規定されていない。

トランスレータサブシステムはＤＤＬ記述から回路設計に必要な２種類の情報を抽出・整理する。すなわち，データパス情報及びステートマシンに関するステート遷移情報である。データパス情報はデータパス設計サブシステムに渡され，各ハードウェア要素（フリップフロップ，機能ブロック等）のまわりの回路の設計に利用される。制御回路設計サブシステムは，ステート遷移情報に基づいて，ステートを実現するフリップフロップとそのまわりの制御回路を設計することによりステートマシンを実現する。入出力ピン，機能ブロック，フリップフロップは，基本セル割当サブシステムにより，セル・ライブラリに登録された基本セルを組み合せて実現される。一方，組合せ回路は任意の論理関数を実現できる複合セルによって実現される。実現すべき論理関数は関数分割サブシステムによって遅延，ファンアウト等を考慮して分割され，それぞれが複合セル設計サブシステムによって複合セルのレイアウト・パタンを与えられる。基本セル及び複合セルに関する情報が回路最適化サブシステムに渡され，回路の冗長な部分が除去され，最終的にＣＭＯＳ回路（基本セル，複合セル及びその接続関係）が設計される。

3. 機能設計フェーズにおける推論

機能設計は，今までほとんどコンピュータによる支援が行われていなかった部分であり，設計者のノウハウに大きく依存している。設計者が設計を行う場合，全体の仕様を睨みながら段々と具体的な回路を明らかにしていく。このように徐々に設計が出来上がっていく（または詳細化されていく）過程は，計算機においては，その実行に伴う副作用が環境を変化させていくことにより実現できる。本システムでは，Prologのassertを用いて次々と事実（fact）を書き込んでいくことにより実現した。このために４０個程の述語（predicate）が用意されている。このような前向き推論の他に，設計を行ううえで様々な条件をチェックする必要がある。このためには後向き推論がふさわしい。後向き推論はPrologの実行メカニズムにより自然に実現した。我々は，論理型言語の述語を用いて設計対象間の関係が自然に表現できることに注意し，これを生かすように知識の表現を行った。

4. 回路設計フェーズにおける処理

回路設計における最初の処理は，ハードウェアの機能記述を回路設計のための情報に変換するトランスレーションである。トランスレータサブシステムは，ＤＤＬ記述を解析し，各レジスタ，ターミナル，ステート毎にそれぞれに対するオペレーションをまとめ，１２種類のフレーム型の情報を生成する。例えば，あるレジスタに関して，そのレジスタに値をセットするオペレーション（レジスタ転送オペレーション）をすべて集め，そのソース，実行条件毎にまとめた情報をそのレジスタのビット数等の情報とともにフレームに格納する。Prologにとって言語の解析は得意な分野であり，効率的にトランスレータを実現することができた。制御回路設計サブシステムは，各ステートに関するフレーム情報に基づいてステートマシンを実現する回路を設計する。この際，必要に応じてステート数の削減を行い（設計者が使っているノウハウを利用する），コンパクトな回路を設計する。フリップフロップ，機能ブロックに関するフレームは，入力されるソース・条件に関する情報をすべて持っているから，その入力部分の回路を設計することは容易である。データパス設計サブシステムは，フレーム情報から各レジスタ，機能ブロックのまわりの組合せ回路を設計する。これにより設計対象の全貌が明らかになる。以上は半導体テクノロジに独立な処理である。

ＣＭＯＳ回路の構成要素としては基本セル及び複合セルを用いる。基本セルはフリップフロップ，機能ブロック等を実現するために使われ，セル・ライブラリに登録されたものをそのままあるいは組み合せて使用する。複合セルは任意の組合せ回路を実現する機能を持つ。ところが，複合セルは，それを構成するゲートを並べる順序によって隙間（セパレーション）を多く取らなければならなかったりセパレーションのないコンパクトなレイアウトが可能になったりする。本システムでは，この複合セルのレイアウトにグラフ理論的なヒューリスティックアルゴリズムを導入している。

5. 評価と今後の課題

試作・評価を通じて，知識ベースとアルゴリズミックな処理を論理型言語上に統合した本システムの枠組みが設計型エキスパートシステムの構築に有効であることを確認した。論理型言語の評価に関しては，知識表現の枠組みとして設計対象間の関係を自然に表現できるという良い性質を持っていること，また，知識ベースとアルゴリズミックな処理の統合が容易であることを確認した。

今後の課題としては，知識及び取り扱う設計データの構造化を行うこと，オブジェクト指向などの他のプログラミング・パラダイムを導入すること，知識の獲得・管理に関する研究を行いその機能を盛り込むことなどがある。

最後に，本研究に対して御支援頂くＩＣＯＴ第二研究室古川室長に深く感謝致します。

参考文献　F. Maruyama et al, "PROLOG-BASED EXPERT SYSTEM FOR LOGIC DESIGN" FGCS'84 pp.563-571 (1984).

PSI マイクロインタプリタの性能評価

横田　実（ICOT第三研究室）

1．はじめに

パーソナル逐次型推論マシンPSI は第五世代コンピュータ研究開発プロジェクトの一環として研究，開発された推論専用マシンである．そのハードウェアは昭和58年末に完成し，ファームウェアは翌，昭和59年春に初版がリリースされた．現在，オペレーティングシステムであるSIMPOSの開発とファームウェアの部分的な改良作業が行なわれている．PSI の特徴は機械語KL0(kernel Language Version 0)そのものが論理型言語をベースとした高級言語であり，それをマイクロプログラム化されたインタプリタ（マイクロインタプリタ）と専用アーキテクチャにより高速に実行することである．ユニフィケーションやバックトラックといった論理型言語に特有の処理がファームウェアレベルで効率良く実行される．本プロジェクト前期の最終年度として，昭和59年後半はPSI の実行性能に関する評価とアーキテクチャ評価，及びそれにもとづくマイクロインタプリタの一部チューニングを行なった．結果としてユニファイヤ部で約2倍，基本コントロール部で 1.3倍，全体で約 1.5倍初版より高速化された．まだ改良の余地は残っているが一応の評価結果をまとめ，PSI のマイクロインタプリタの性能特性について明らかになったので以下に報告する．なお，性能評価の比較対象としてはDEC-20Prologコンパイラ Version 1.0 版によりfast-code 指定でコンパイルしたプログラムの性能を用いた．

2．基本実行性能

Prologプログラムの特性は，適応すべき問題の性質やプログラミングスタイルに大きく影響される．従って，まず推論マシンの性能を左右する基本項目としてユニフィケーションの実行時間，述語呼出し処理やバックトラック処理のオーバヘッドなどを選び個々の特性について評価した．測定は例えば以下のような簡単なクローズを 100個並べたものを更に多数回ループさせて実行時間を測定した．

```
.... :- p10(a).
p10(a) :- p11(a).
p11(a) :- ...
```

結果を表１に示す．表の数値はコール／リターンを含む 100 クローズ分全体の処理時間である．表からPSI のユニフィケーション速度はDEC-10Prologと同等か，それ以上であることがわかる．特に，呼出し元に値を返す場合（var <-- atom/var）の処理が早い．これはDEC-10Prologでは呼出し元変数セルへの書込みがあまり高速に処理されていないことによると思われる．さらに，PSI ではユニフィケーションのために固定的な処理を必要とするが，正味の引数当たりのユニフィケーションに要するステップ数は簡単な場合で５～６マイクロ命令と少ない．従って，表１の６引数の例のように引数の数が多い程高速性が発揮される．

実行順序制御に関する性能については表２に示すが，PSI の方が遅い場合もある．PSI では遠隔カットなどの強力な実行順序制御を実現しているのでもともと実行環境の管理が複雑でありその分ハンディがある．例えばスタックの本数はDEC-10Prologに比べて１本多く，またバックトラックに備えて退避される情報量も倍近い．その意味で，充分健闘しているといえるであろう．

BODY vs. HEAD	PSI(msec)	DEC(msec)	DEC/PSI
atom <--> atom	0.89	0.99	1.1
atom ---> lvar	0.70	0.91	1.3
gvar <--- atom	1.06	2.41	2.3
lref <--- lvar	0.82	0.96	1.2
gvar <--- lvar	0.78	1.55	2.0
gvar <--- f(a)	1.14	2.81	2.5
f(a) ---> lvar	0.78	1.06	1.4
f(a) <--> f(a)	1.42	1.69	1.2
6atoms<-->6atoms	1.59	3.62	2.3

Table 1. Unification Speed

control type	PSI(msec)	DEC(msec)	DEC/PSI
det. call	0.56	0.69	1.2
det. call/return	0.94	0.76	0.8
non-det. call	0.97	1.25	1.3
shallow backtrack	0.55	0.49	0.8
deep backtrack	2.73	3.68	1.3

det. : deterministic

Table 2. Execution Control Overhead

3．サンプルプログラムを用いた評価

実際のPrologプログラムの実行性能がどの程度であるかを評価するためにPrologコンテストで選ばれたプログラムの中から幾つか選び実測を行った．これについても測定誤差を少なくするため，cut-and-failにより多数回ループさせて実測した．結果を表３に示す．全般に同等か，もしくはPSI の方が若干遅い．評価対象プログラムはいずれもプログラム規模が小さいうえにリスト処理が多く推論マシンの評価としては偏りがある．例えば，naive reverse のよ

うにコンパイラによる最適化が効果的なプログラムに対してDEC-10Prologは高速である．一方，PSI はそれらのシンプルなプログラムに対して基本処理のオーバヘッドが大きく相対的に遅い結果となった．

test program	PSI(msec)	DEC(msec)	DEC/PSI
nreverse (30)	14.6	9.48	0.65
quick sort (50)	16.1	14.6	0.91
tree traversing	52.2	61.1	1.17
lisp (tarai3)	4050	4360	1.08
lisp (fib10)	375	402	1.07
lisp (nreverse)	174	194	1.11
8 queens (1)	105	97.5	0.93
8 queens (all)	1710	1580	0.92
reverse function	38.9	41.7	1.06
slow reverse (6)	101	89.0	0.88

Table 3. Execution Speed of Sample Programs

4．応用プログラムを用いた評価

実用的な規模の応用プログラムにおけるPSI の総合性能を評価するために実際にPSI の上で稼動しているオペレーティングシステムSIMPOSの一部であるウィンドウシステムとFGCS'84 においてデモンストレーションに使用されたハーモナイザシステム，8パズル，Bottom Up Perser (BUP) の4つのプログラムを用いて評価を行なった．比較すべきデータが無いためここではマイクロインタプリタにおける機能別実行時間比を表4に示す．プログラム動特性の評価からは各応用とも組込述語の利用頻度が高く，ウィンドウシステムでは全述語呼出しの約80% ，最もProlog的なBUP においても60% は組込述語の呼出しである．PSI では機械語KL0 にデータ操作等のための組込述語を豊富に用意したためこの傾向は強くなっている．しかしながら全実行ステップ数に対する組込述語処理の占める割り合いはそれほど多くない．むしろ実行順序の制御に多くの時間を費している．これは既に述べたようにKL0 の実行順序制御そのものが複雑であることが影響している．また組込述語の引数読み出し (get-argument) に時間がかかっている．BUP,harmonizerにおいてユニフィケーションの比率が高いのは構造体データのユニファイに時間を要するためである．

応用レベルでのDEC-10Prologとの比較は言語仕様の違いのため難しいが，一例としてBUP について評価した結果を表5に示す．まずDEC-10Prologで記述されていたプログラムをKL0 と同じにするために書直しを行いさらに実行時のスタック再配置の時間を補正して比較した結果，PSI での実行が 1.2～ 1.4倍高速であるという結果になった．スタック再配置は従来型マシンにおいて一つのメモリ空間に2つ以上のスタックを配置しようとした場合に生じる問題である．PSI ではこれを避けるためエリアベースの論理アドレッシング機構をアーキテクチャに取り入れ，多数のスタックを自由に設定することを可能としている． 従って，PSI ではスタックの再配置が不要である．

5．おわりに

以上，3種類の評価を通じてPSI は当初の設計目標であったDEC-10Prologと同等以上の性能を達成できたといえる．PSI を設計するに当たっては，推論マシンとしての立場からPrologらしさ，即ち，ユニフィケーションやバックトラック処理に重点を置いた．ある程度複雑な処理を念頭に置き，それに充分耐えるように考慮したつもりである．その結果，単純な処理にも，複雑な処理にも比較的フラットな性能を示すことがわかった．PSI がstructure sharing を採用していることもこの傾向を強めている．しかしながら，単純な処理においてオーバヘッドが存在する，及びコンパイラによる最適化が効きにくいといった面があるのも事実である．また，どのような処理を高速化すべきかについては，表4からも分るように個々のプログラムによって特性に大きな差異があり，論理型プログラムとしての一般的傾向をとらえることは難しい．Prologが柔軟であるが故に，これまでのプログラム言語に比べてプログラム特性を予測することは困難であろう．一方，DEC-10Prologとの比較作業は，逆にDEC-10Prolog処理系の特性を明らかにする結果となった．例えば，レジスタを活用できるようなユニフィケーション処理は高速であるが，メモリアクセスが含まれると遅くなる等である．現在，より詳細なPSI の評価結果をもとに，更に小型，高性能の推論マシンへ向けての見直しを行なっている．

最後にPSI の開発ならびに評価を担当してくれたPSI アーキテクチャグループメンバ諸氏，BUP を用いてPSI の評価を行なってくれた平川，奥西両氏に深謝します．

BUP	original	DEC	DEC-SS	PSI	DEC/PSI
A	349	56	52	43	1.21
B	987	256	194	139	1.40
C	1486	540	424	309	1.37

A: "Alice loves Dyna."
B: "Alice chases the march_hare who has a watch."
C: "He who falls in love with himself has no rivals."
SS: stack shifts overhead
Table 5. Comparison between DEC-10Prolog and PSI

program	control	unify	trail	get_arg	cut	built	others
window	31.1%	17.1%	2.0%	13.6%	10.0%	20.0%	6.2%
8 puzzle	27.5	11.0	7.5	22.7	-	30.7	0.6
BUP	26.3	43.0	4.7	5.2	5.6	12.4	2.8
harmonizer	25.5	47.0	5.4	7.0	4.1	8.3	2.7

control : call/return/backtrack　　cut : cut
unify : unification/copy_argument　　built : major builtin predicates
get_arg : get_argument/put_argument
Table 4. Execution Time Ratio of the MicroInterpreter

ESPのプログラミング環境

近山　隆（ICOT 第三研究室）

1. 概要

逐次型推論マシンゆとそのオペレーティング／プログラミング・システムSIMPOSは、論理型プログラムのための良好な開発環境を提供するためのいわゆるスーパー・パーソナル・コンピュータ・システムである。SIMPOSはすべて言語ESPで記述されている。SIMPOS上の応用プログラムも、原則としてESPで記述することを想定している。このゆ／SIMPOS上でのESPを用いたプログラミング環境について報告する。

2. ESP

ESPは、論理型言語と対象指向言語とを融合したプログラミング言語である。

従来のPrologでは、プログラムのモジュール化のための機能がまったく用意されていなかったり、逆に、強力な動的なモジュール機能の導入のために実行時のオーバヘッドが大きくなり、著しい処理効率の低下をきたしていたりする場合が多かった。実用に供し得るような大規模なプログラムを論理型言語を用いて記述しようとする際には、このことが大きな障害になっていた。

ESPは、Prologと同様、論理型の実行メカニズムを基本としているが、対象指向機能を導入することにより、大規模なプログラムの開発に欠かせない、モジュール化の機能を実現している。処理方式の上では、コンパイル時の静的な解析と、実行時の動的なパラメタ化のバランスをとることに留意して設計した。このため、充分な柔軟性をもつセマンティクスを実現しながら、実用的な処理効率をもつシステムを実現できたわけである。

SIMPOSは、最下層のデバイス駆動ルーチン、メモリやプロセスの管理ルーチンから、最上層のプログラミング・システムに至るまで、すべてESPで記述されている。下層では充分な処理速度が、上層では記述性の高さが要求されるOSのような大規模なシステムを、単一の言語で統一的に、しかも短期日の内に開発できたことは、ESPの言語としての優秀性に負うところが少なくない。

こうしたESPの提供する機能は、SIMPOS自体のみならず、その上に構築される応用システムの記述にも大いに有効である。特に、応用システムが大規模化すればするほど、対象指向によるモジュール化機能が真価を発揮するのである。

3. SIMPOS

SIMPOSは、

単一ユーザ
複数プロセス
対話的処理

を基本とするオペレーティング・システム部と、その上で使いやすいユーザ・インタフェースを提供するプログラミング・システム部とからなる。

SIMPOSの最大の特徴は、すべてがESPで記述され、すべての機能がユーザに開放されている点にある。ESPのもつ継承の機構を用いれば、ユーザはOSの提供する機能を自由に組み合せ、さらに自分の必要とする機能をこれに付加することが容易にできる。OS自体もこのような使用法を念頭に置いた設計になっており、標準的な機能の組み合せをいわばレディ・メイドの形で提供する他に、種々の独立した機能を部品としても提供している。

以下に、SIMPOSの提供する種々の機能と、それらを実現しているサブシステムの概略を、ユーザとのインタフェースを中心に説明する。

(1) ウィンドウ

ゆのビットマップ・ディスプレイの画面は、いくつものウィンドウと呼ばれる矩形領域に分けられている。このようなウィンドウのひとつひとつは、それぞれ通常の端末のように動作させることができる。これにより複数プロセスの処理状況を容易に見てとることができる。また、ひとつのプロセスであっても、性質の異なる表示を別のウィンドウに出すことにより、より見やすい表示を行なうことができる。このウィンドウを管理しているのがウィンドウ・サブシステムであり、SIMPOSの対話的プログラム環境の基本となっている。

(2) ログイン

システムの立上げが終わると、画面上にはログイン・ウィンドウが現れる。あらかじめ登録したユーザ名とパスワードを入力すると、そのユーザ用に指定された初期化が行われる。どのユーザについてどのような初期化が必要であるかはユーザ管理システムと呼ばれるサブシステムが管理している。

(3) システム・メニュー

ログイン処理の終了後、マウスの右ボタンを二回クリッ

クすると、システム・メニューが表示される。このシステム・メニューはＳＩＭＰＯＳのコマンド・プロセッサの役割を果すもので、いつでも右二回のマウス・クリッククリックで呼出すことができる。以下に述べるような各システムは、すべてこのシステム・メニューで当該の項目を選ぶことにより起動される。

(4) エディタ

システム・メニューでＥｄｉｔｏｒを選ぶと、Ｅｄｉｐｓと呼ばれるエディタが呼出される。これを用いて、プログラムや任意のテキストの作成や修正、ファイルへの格納などを行なう。カナ漢字変換ユーティリティを用いて、日本語の入力も可能である。

(5) ライブラリアン

プログラムのソースが完成したら、ライブラリアンを呼出し、実行可能な型式への変換を行なう。実行できるＥＳＰのプログラムには、解釈実行コードと機械語コードの二種類がある。解釈実行コードはソース・プログラムの情報がほとんどそのまま残っているので、デバッグに便利である。一方、実行速度の面では機械語が有利である。両者を自由に併用できるので、適宜使い分けるのがよい。

このようなコードやソース・プログラム、そして継承関係の解析に必要な種々の情報を管理しているのがライブラリと呼ばれるサブシステムである。ライブラリアンはこのライブラリのインターフェース部にあたる。ライブラリアンは、登録済のＯＳやユーザ定義のクラスに関する情報の検索機能も提供している。

(6) デバッガ

プログラムのデバッグはデバッガを用いて行なう。デバッガでは、対話的なプログラムの実行や、トレースを出力しながらのステップ実行ができる。ステップ実行のモデルとしては、述語単位のコントロール・フロー・モデルに加えて、クローズ単位のモデルによる実行も可能になっている。

(7) ウィンドウ・マニピュレータ

多数のウィンドウを用いて作業をしていると、最初に指定したウィンドウの大きさや位置を、途中で変えたくなることも多い。このような場合に用いるのがウィンドウ・マニピュレータである。

(8) ファイル・マニピュレータ

ディスク上のファイルや、種々のデバイスは、ひとつの木構造をなすディレクトリの下に管理されている。ファイル・マニピュレータを用いると、このディレクトリの構造を調べたり、ファイルの消去やディレクトリの生成などを指示することができる。また、ファイルに関する種々のユーティリティ・プログラムの呼出しもできる。

(9) ネットワーク

φはローカル・エリア・ネットワークを介してたがいに結合されており、異なるφ上のプロセス同士がネットワークを通じて通信することができる。ネットワーク・マニピュレータは、このネットワークの状態の表示などを行なうプログラムである。

(10) ターミナル・エミュレータ

ターミナル・エミュレータは、φを汎用機の端末として動作させるためのシステムである。また、端末のラインを用いたファイルの転送なども可能である。

4. おわりに

φ／ＳＩＭＰＯＳは、論理型プログラムの開発環境として、最低限必要な基本機能を実現することができた。本当に使いやすいシステムとして完成させるためには、今後の一層の改良・拡張が必要である。

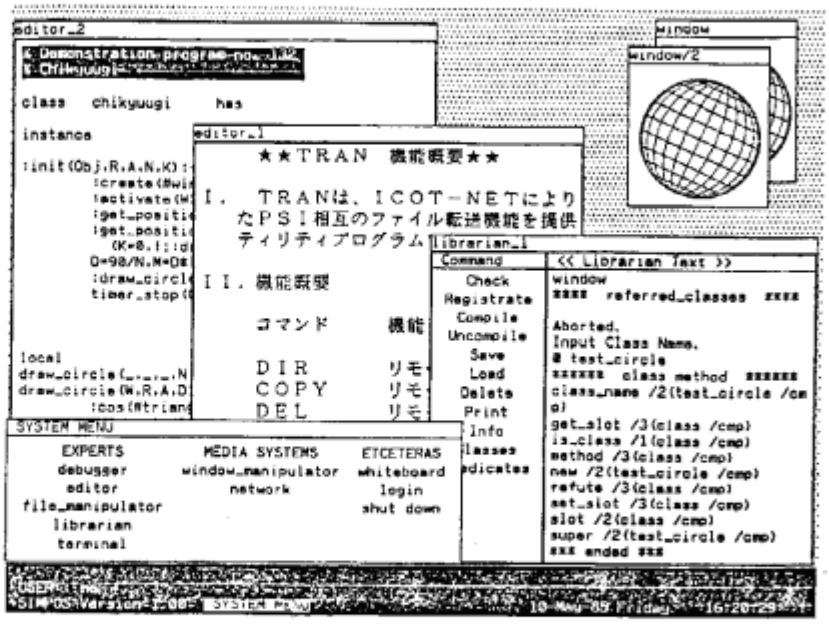


図 1 ＳＩＭＰＯＳのディスプレイ表示画面例

ＳＩＭの拡張機構：高速プロセッサモジュール

梅村　護　（日本電気株式会社Ｃ＆Ｃシステム研究所）

設計の方針

逐次型推論マシン（ＳＩＭ）の拡張機構、高速プロセッサモジュール（ＣＨＩ）の特徴について述べる。本モジュールは、ＦＧＣＳプロジェクトにおけるシステム開発において、既に開発を完了し評価フェーズに入っているパーソナル型ＳＩＭ（ＰＳＩ）の能力を越えるような大規模な知識情報処理の応用問題に対処するためのプロセッサである。ＣＨＩの基本設計方針は以下のとおりである。

①現状で利用可能な最高速の素子を用い、言語処理に専念するバックエンド型プロセッサとする。

②ＰＲＯＬＯＧの基本処理の高速な実行と大規模な問題の実行の為に大容量の実メモリを実装する。

③コンパイラ技術を利用したきめ細かい最適化が可能な独自の機械語セットを設定する。

④専用ハードウェアによる高速実行を意図すると共に、効率良いシステム記述を可能とするＣＨＩ用の拡張ＰＲＯＬＯＧ言語を設定する。

アーキテクチャの特徴

以上の方針の下に設計したＣＨＩは、ＣＭＬ素子（スイッチング速度＝ 0.7ｎｓ）を用い、ＰＲＯＬＯＧを基本とする言語の処理に依存したハードウェア構成と、マイクロプログラムによるＰＲＯＬＯＧ特有の機能の提供によってプログラムの高速実行を可能とした。また、検討の結果、大規模な問題を高速に実行するためには大容量のメモリを備えることが必須であることを確認し、６４メガワード（２５６ＭＢ）の実記憶を実装することとした。

ＣＨＩの基本処理方式は、ユニフィケーション処理にマッチしたレジスタ構成を採用し、それを最大限に利用するようなオブジェクトコードをコンパイラによって発生させることによって、高速な処理を実現するものである。同時に、プログラム制御のために必要な領域の確保や削除、および不要となった形骸の再利用などにおいて、効率のよいメモリ利用を行うことも高速な実行に有効である。ＣＨＩは機械語セットとしてこれらのきめこまかい制御が可能な機能を備えている。また、機械語は３６ビットの固定長であり、コードの先取りや分岐などの制御を行いやすくしている。さらにＰＲＯＬＯＧのデータ属性に応じたタグをメモリフィールドに置くことにより、ユニフィケーションにおけるデータ属性に応じた制御の切り替えを極めて高速に実現するよう設計した。ハードウェアは、ＰＲＯＬＯＧの基本機能に依存してモジュール化し、局部的に並列処理を行って高速化をはかっている。

ＣＨＩ用拡張ＰＲＯＬＯＧ言語はシステム記述が可能なこと、第５世代プロジェクト用言語として充分な環境を提供できることを目標とし、ＤＥＣ１０ＰＲＯＬＯＧを基本として拡張を施した仕様となっている。ＰＳＩ上のプログラムについては、マシンに依存する部分を除いて互換性が保たれる。ＣＨＩ上で実行するためのプログラムをＰＳＩ上で作成するときには、予めＣＨＩの言語仕様を知っておく必要がある。

ＣＨＩの実行速度

ＣＨＩはそのハードおよびファームの基本検査を全て完了し、現在ソフトウェアの実機デバッグを実施中であり、正確な評価は今後の作業に待たねばならないが、机上評価によれば、APPENDの基本繰り返し処理を３．６マイクロ秒程度で実行することを確認した。その他の機能的な見積りとＰＳＩとの比較によれば、当初の目標であるＰＳＩの３ないし５倍の処理性能は充分クリアーできる見込みである。