TM-0151

情報処理学会第32回(昭和61年前期)
全国大会発表論文集
35 件

January, 1986

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

データフロー方式並列推論マシン
―実験機の動特性―

六沢 一昭　大原 輝彦　来住 晶介　**伊藤 徳義
（　神　電　気　）　（ＩＣＯＴ）

1．はじめに

我々は、データフロー方式並列推論マシン実験機（PIM-D)を試作し、現在評価を行っている。本稿では、PIM-D の動特性について述べる。

2．PIM-D の構成

PIM-D は、複数台の演算エレメント（PE）、構造メモリモジュール（SM）、及び1台のサービスプロセッサ（SVP)、及びこれらを接続するネットワークからなる（図1）。

PEは、データフローグラフを解釈／実行する。PE は、トークンのバッファリングを行う PQU（Packet Queue Unit）、オペランドの待合わせを行う ICU（Instruction Control Unit）、発火した命令を実行する APU（Atomic Processing Unit）、及びローカルメモリ LM からなる（図2）。

SM は、構造データを格納する共有メモリである。SVP は、PIM-D の初期化、保守、及び計測データの収集を行う。

3．PIM-D の動特性

評価プログラムとして 7-queens を用いて測定した動特性を以下に示す。

3－1．発火制御

ICU では、オペランドメモリの検索／書込みにハッシングを用いている。また、ハッシュ衝突の起きた場合にそなえ、各エントリにチェインフィールを設け、再ハッシュ(rehash)によりエントリを連鎖する方式をとっている。例を図3に示す。図中、searchとあるのはチェイン探索のことである。また、↓の数が"チェインの長さ"である。

測定データを表1に示す。表の値からハッシュ衝突の確立を求めると、

再ハッシュ回数／（2オペランド命令トークン数／2）＝0．18

となり、およそ２０％（以下）である。オペランドメモリ使用率が最大で１２％であるので、まだハッシュ関数に改良の余地がある。

確立を２０％として、衝突を起こしたトークン１つあたりのチェイン探索回数を計算すると、

チェイン探索回数／（2オペランド命令トークン数／2＊２０％）＝0．74

となり、１回以下である。これより、チェインの長さは0～1であるといえる。

3－2．稼働率

ICU 及び APUの稼働率（idle率）を表2、図4に示す。表中、待時間の2行目、3行目はそれぞれ総クロック数に対する割合、1回のパケット入出力あるいは T-BUSアクセスあたりの時間数である。図4では稼働率（idle率）を1パケットあたりのマイクロステップ数（＝クロック数：1ステップは1クロックで実行）で示した。

PE1～3台の範囲ではidle率（待時間）は一定である。

稼働率は低い（APU 51%,ICU 35%)。これは、APU がパケット出力あるいは LM アクセスを行う際、T-BUS のバス権獲得に非常に長い時間がかかっているためである（6クロック／1アクセス、総クロックの３５％）。図2からわかるように、T-BUS には多くのユニットが接続しており、これがネックになっている。

表1．発火制御に関する計測データ

受信トークン数	714Kﾄｰｸﾝ	再ハッシュ回数	42K回
2オペランド命令	474Kﾄｰｸﾝ	チェイン探索回数	35K回
1オペランド命令	203Kﾄｰｸﾝ	トークンキュー（PQU:容量16Kﾄｰｸﾝ)使用率	
特殊パケット	37Kﾄｰｸﾝ	最大長	1.6Kﾄｰｸﾝ(10%)
送出トークン数	477Kﾄｰｸﾝ	オペランドメモリ（容量32Kﾄｰｸﾝ)使用率	
		最大数	3.7Kﾄｰｸﾝ(12%)

図1．PIM-D の構成

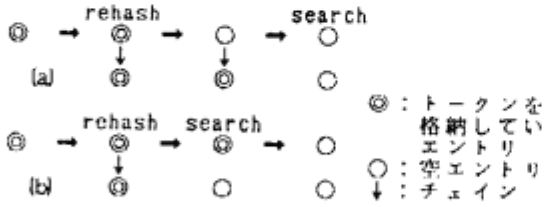


図3．チェイン操作 － チェインの長さ0～1
（(a)，(b)ともrehash 1 回、search 1 回）

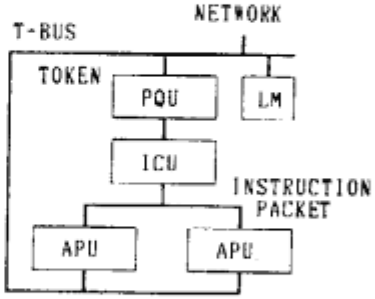


図2．PE の構成

*1986年1月より ICOT 、**1986年1月より神電気

3－3．実行命令解析

OR 並列及び GHCのプログラムを実行させた時の主な命令の実行回数などを表３に示す（実行された命令の総数は、OR並列が21種類440K回、GHC が20種類661K回）。表中、D換算マイクロステップは、１回の T-BUSアクセスに６ステップ（クロック）かかるとして換算した値（B+C*5)。Eの（）内は、総マイクロ実行時間（T-BUS アクセス待時間を含む）に対する割合。Gは、マイクロ実行における T-BUSアクセスに費やす時間の割合（C*6/D)。

OR並列、GHC とも実行ステップ数の上位（Eの上位）３命令で総マイクロ実行時間の40～50%を、また表に示した命令（総種類数の１／４）だけでほぼ２／３をしめている。これらは link を除いて T-BUSアクセスの割合が大きい（平均41%=17/(24+17) ：図４より）。このこともネックが T-BUSであることを示している。

4．まとめ

PIM-D の動特性について述べた。結果は T-BUSがネックになっていることを示している。 LM は APUによってのみアクセスされ、そのアクセスが全 T-BUSアクセスの大きな比重をしめていることから、 LM を T-BUSから切離し、別のバスを APUとの間に設けることで大きな改善が期待できる。

謝辞 ICOT 関係各位に感謝する。

［参考文献］

1)伊藤,"並列型 Prolog のデータフローマシンによる実行方式,"日経エレクトロニクス,1984.11.5号.
2)来住　他,"データフロー方式並列推論マシン実験機の構成,"情報処理学会第30回全国大会 6C-2.
3)久野、大原　他,"データフロー方式並列推論マシン　－実験機の性能評価－，－実験機のプロセッサ割り付け方式－,"本大会 2R-3,4.

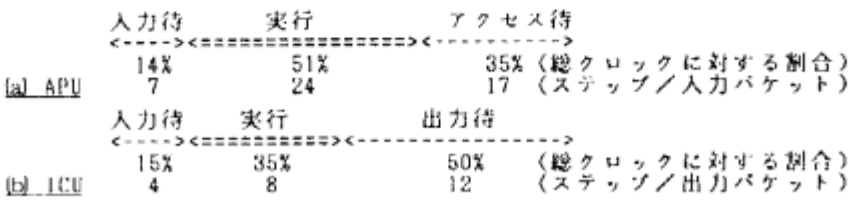


図４．APU,ICU の稼働率

表２．ICU,APU の　idle率

PE台数	総クロック数（クロック）	APU パケット入力待時間（クロック）	APU 入力パケット数	APU T-BUSアクセス待時間（クロック）	APU T-BUSアクセス回数	ICU パケット入力待時間（クロック）	ICU 入力パケット数	ICU パケット出力待時間（クロック）	ICU 出力パケット数
1台	11,409K	1,580K 14% 6.6	239K	4,160K 36% 6.2	668K	1,671K 15% 2.3	714K	5,956K 52% 12.5	477K
2台	6,012K	806K 13% 6.5	124K	2,082K 35% 6.1	334K	878K 15% 2.5	356K	2,957K 49% 11.9	248K
3台	4,044K	552K 14% 6.5	85K	1,341K 33% 6.0	223K	615K 15% 2.5	248K	1,975K 49% 11.7	169K

表３．実行命令

	命　令	A 実行回数	B マイクロステップ数	C T-BUSアクセス回数	D 換算マイクロステップ数	E D＊A	G T-BUSアクセスの割合
OR並列	call（成功）	10K	100	18	190	1,900K(19%)	57%
	copy	54K	15	3	30	1,620K(16%)	60%
	link	66K	20	0.5	22.5	1,485K(15%)	13%
	append-stream	7K	50	8.5	92.5	648K(7%)	55%
	return	5K	70	11.5	127.5	638K(6%)	54%
GHC	call（成功）	12K	100	18	190	2,280K(15%)	57%
	return	17K	70	11.5	127.5	2,168K(14%)	54%
	copy	58K	15	3	30	1,740K(11%)	60%
	switch-by-type	33K	20	4	40	1,320K(9%)	60%
	link	53K	20	0.5	22.5	1,193K(8%)	13%
	create-global	14K	35	7	70	980K(6%)	60%

2R−6

データフロー方式並列推論マシン実験機のプロセッサ割付け方式

大原茂之　久野英治　宗佐昌介　伊藤徳義
（沖電気総合システム研究所）

1．　はじめに

ＰＩＭ－Ｄは第５世代コンピュータ・プロジェクトの一環として研究，開発されたデータ・フロー方式の並列推論マシン実験機である。ＰＩＭ－Ｄ実験機のアーキテクチャはデータ・フローグラフを解釈，実行する演算エレメント・モジュール（ＰＥ）と，共有メモリとして用いられ構造体データを保持する構造メモリ・モジュール（ＳＭ）、及びこれらを結合している階層バスからなっている。階層バスはこれらのモジュール間の通信用トークンの交換網となっている。また４ＰＥ，４ＳＭを一つのクラスタとしてこのクラスタを同一の階層バスで多数結合できるようになっている。ＰＩＭ－Ｄ実験機ではＰＥとＳＭとに機能が分割されていることにより両者の処理が並列に行なえ、かつプロセスの割付けと構造データの割付けが独立して容易に行なうことが可能となっており、それらの割付け戦略を検討することにより，リソースの有効利用，処理時間の高速化が期待できる。

筆者らは、現在ＰＩＭ－Ｄ実験機の試作を終え評価を行っている。本稿では，ＯＲ並列型Ｐｒｏｌｏｇで記述されたプログラムについて評価を行なったものを報告する。

2．　ＯＲ並列型Ｐｒｏｌｏｇの実行方式

Ｐｒｏｌｏｇにおいては，１つのゴール・リテラルと、その述語に対する定義が与えられたとき，そのゴールと定義と定義に含まれるすべての節の間でユニフィケーションが試行される。通常の逐次型推論システムの処理系での節の呼出は、プログラム・リスト上で現われた順に、ゴール・リテラルの呼出は左から右に行う。　ゴールが与えられたときその一番左のリテラルが選ばれ、その述語に対する定義中の最初の節との間でユニフィケーションをおこなう。　これが失敗したとき、バックトラッキング機構による制御により次の節を呼び、ユニフィケーションの再試行をおこなう。これに対しＯＲ並列Ｐｒｏｌｏｇ方式ではこのユニフィケーションを並列に行なうことによって，推論の高速処理を可能にしようとするものである。すなわちゴールが与えられたとき対応する定義中の各節に関するユニフィケーション・プロセスを並列に生成することにより実現できる。ユニフィケーションが成功した場合、一般に出力される解の順序は一意には定まらない。そこでＰＩＭ－Ｄ実験機にインプリメントされたＯＲ並列型Ｐｒｏｌｏｇにはストリームの概念が導入されており，ＯＲ並列で動作しているユニフィケーションプロセスと，その結果を使用するプロセスとの非同期通信手段をこれにより提供している。ストリームはnon_strictなデータ構造である。ここで言うnon_strictなデータ構造とは構造の要素が必ずしもすべて確定していない時点で生成されるような構造のことである。この構造は節の起動と同時にＳＭ上に生成され、その要素は解が得られた順に書き込まれる。解を消費するゴール側はすべての解が生成されるのを待つことなくそれらを参照することが可能である。［１］［２］

プロセッサの割付けはプログラムの実行前に静的に決定される。ＰＥ内のＬＭにはプロセスを実行するＰＥとそのプロセス管理テーブルのポインタが格納されている領域がある。ＰＥがプロセスを起動する際にはこの領域をサーチして，実行モジュール、プロセス管理テーブル、及びプロセス識別子を決定している。（色付きトークン方式）したがってこのテーブル内のポインタを操作することによってプロセッサとプロセスの割付けを容易に制御できる。

3．　評価プログラムによる実験

評価用プログラムとして７queensのプログラムを選んだ。これは探索型のプログラムとして典型的なものであり，並列度が高くプロセッサ割当効果がよく現われることが期待できると思われたからである［３］。現ＰＩＭ－Ｄ実験機は２つのクラスタから構成されており評価実験は１クラスタの場合と２クラスタの場合について行った。

図１には１クラスタ内でＰＥ，ＳＭの台数をパラメータにとったときの実行マシンサイクル数が示されている。また図２にはＰＥを４台としＳＭの台数をパラメータにとったものが示されている。図１と図２を比較すると１クラスタ内ではＰＥ，ＳＭを共に同じ台数ずつ増加させたほうが実行マシンサイクルが減少することがわかる。ＳＭが１台である場合は４ＰＥからのアクセスが集中し１ＰＥ，１ＳＭの場合と実行サイクルはあまり変らないが，ＳＭの台数を増加してゆくとそれが減少してゆくのがわかる。図３には２クラスタの場合の実行マシンサイクルを示した。ここではクラスタ間のプロセッサ割付け比を1:8から1:64まで変化させたものをプロットしてある。ここで言うプロセッサ割付け比とは、起動するプロセスをどのくらいの割合でクラスタ外のプロセッサに割り当てるかを表したものである。ここではプロセッサ割付け比を小さくすると実行サイクルが減少するがあまり小さくするとクラスタ間階層バスの通信量が大きくなり、ここのバスがデッドロックを起こした。これは今後の大きな課題である。

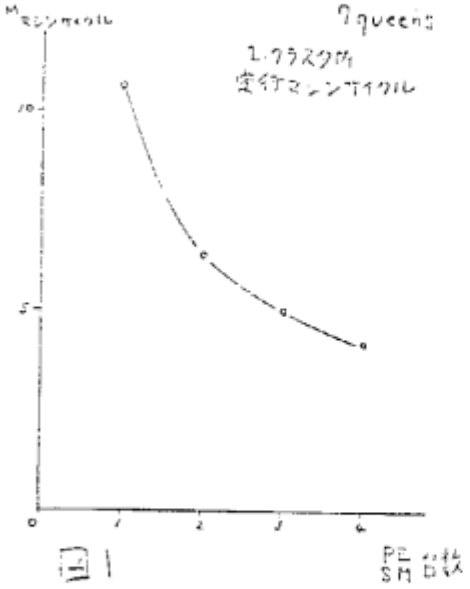


図1

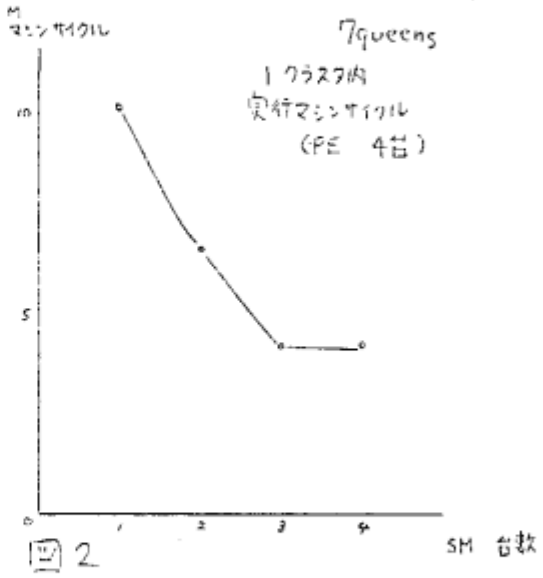


図2

4. おわりに

ＯＲ並列型Ｐｒｏｌｏｇで記述された7 queensプログラムにより，プロセッサ割付け方式が実行マシンサイクルにどのように変化を与えるか実験してみた。実験機のＳＭにはＰＥのＲＰＱに相当するバッファがないためＳＭへの通信（ＰＥ－ＳＭ間，ＳＭ－ＳＭ間）に際して交換網の閉塞率が高くなりオーバーヘッドとなることがわかった。このため算術演算など部材唯一しか存在しないことがあらかじめわかっているような場合はストリームを利用しないことによりＳＭへのアクセスを減らすなどの工夫が必要であると言える。またハードウエアの点からはＳＭにバッファをもうける、あるいはＳＭ間通信用のパスを設けること，などを検討する必要があろう。

今後はさらにいろいろなプログラムを用いて詳細な評価をおこなっていく予定である。

謝辞

ＩＣＯＴ関係各位に感謝の意を表す。

〔参考文献〕

1）伊藤　他，「データ・フロー方式推論マシンのアーキテクチャ」
Proc. of Logic Programming Conf. ICOT Mar 1984

2）伊藤、　「並列型Prologのデータ・フローマシンによる実行方式」
日経エレクトロニクス. 1984年11月5日号.

3）久野　他，「データ・フロー方式並列推論マシンにおけるAND/OR並列効果のシミュレーションによる測定」
情報処理第30回全国大会予稿集

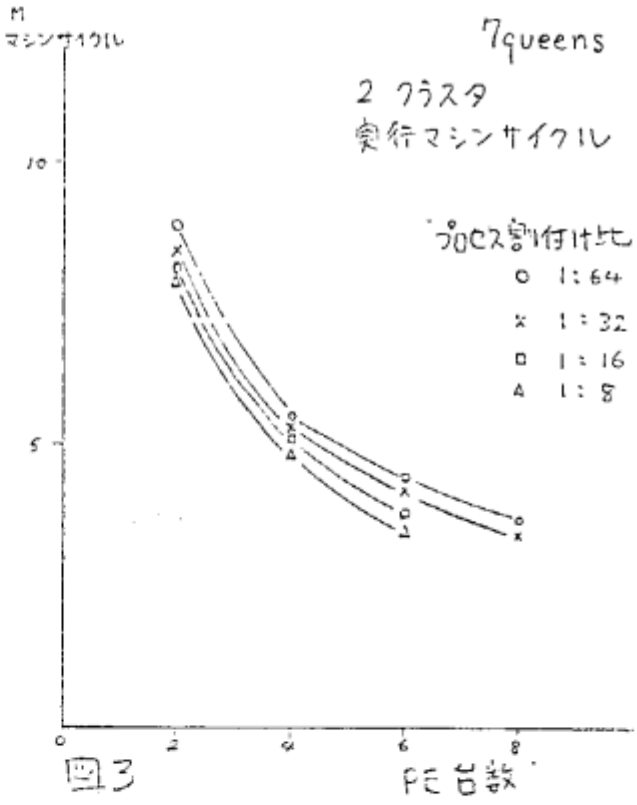


図3

2R-3

データフロー方式並列推論マシン

—— 実験機の性能評価 ——

久野 英治　　大沢 一昭*　　舌沢 方之　　伊藤 徳義**
（沖電気総合システム研究所）　　（ＩＣＯＴ）

1．はじめに

PIM-D (Parallel Inference Machine based on Dataflow model) は第５世代コンピュータプロジェクトの一環として、ICOTと共同で開発している並列推論マシンの１つであり、データフローモデルをベースとしているところに特徴がある[2]-[4]。PIM-Dの目標言語は並列型論理プログラミング言語であり、現在、AND並列型Prologの代表言語であるGHC (Guarded Horn Clauses) [1]とOR並列型Prologの２つの異なる言語をサポートしている。著者等は、PIM-D実験機を試作し、その性能評価を行ってきた。本稿では台数効果を中心としてその評価結果を報告する。

2．PIM-Dの概要

PIM-Dは、４台の処理要素ＰＥ (Processing Element) と４台の構造メモリＳＭ (Structure Memory) を共通バスで結んだクラスタを構成単位とし、このクラスタを上位の階層バスで結んだ構造をしている[2]。
各ＰＥは、データフローグラフで表現されたプログラムを並列に実行し、必要に応じて他ＰＥへのプロセスの分散を行う[3]。構造データは、ＳＭ群に分散して格納され構造データ操作が必要なとき、ＰＥからＳＭへ操作指令が転送される。
プロセスの割当てや構造データの割当ては、ＰＥで制御される。このため、各ＰＥは、全ＰＥの空きプロセス番号の管理表やＳＭの空きセルの管理表を持っており、プロセス起動命令や構造データ生成命令が実行されたときに、これらの管理表から新プロセス番号や構造データセルのアドレスを取出す。

現実験機では、各ＰＥのプロセス番号を前後のＰＥ番号が昇順になるように格納している。ＳＭの空きセルの管理表も同様である。従って、プロセスや構造データはＰＥ群やＳＭ群にほぼランダムに分散して割当てられる。この条件でＰＥ（ＳＭ）数を１～４台と変化させて実験機の性能測定を行った。

3．データ収集

表１はデータ収集のために用いたサンプルプログラムとノード（命令）あたりの平均実行マシンサイクル数である。同表に示すように、この値はプログラムによって大きな変化はなく、ほぼ２５サイクル程度である。

次に、実験機の台数効果について示す。台数効果は、プログラムを実行する（全解を求める）のに要したマシンサイクル数、及び、マシンの性能指標の１つであるRPS (Reductions Per Second) で示す。ここで、RPSは単位時間当りに成功したゴール呼出し (Reduction) の回数である。

図１はOR型Prologプログラム（プログラム番号１～５）に対する台数効果を、図２はOR型Prologプログラムと GHCプログラム（プログラム番号１，２，５，６及び７）の台数効果の比較結果を示す。

表１．平均命令実行サイクル数

番号	プログラム	言語	命令当りの実行サイクル
1	7-queens	OR型	25.93
2	7-queens (決定的宣言有)	OR型	24.12
3	BUP	OR型	26.26
4	DCG	OR型	21.16
5	qsort	OR型	25.57
6	7-queens	GHC	27.52
7	qsort	GHC	23.56

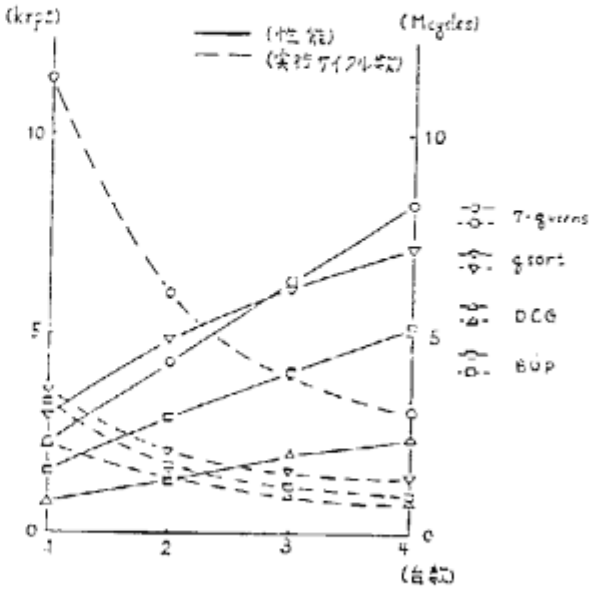


図１．OR並列型Prologプログラムの台数効果

* 1986年1月よりICOT　　** 1986年1月より沖電気

ソフトウェアシミュレーション結果で示したように、プログラムに内在する並列性の大きいものほど性能の台数効果が顕著である[5]が、実験機でも同様の結果を得た。例えば、7-queensは並列性が高く台数の増加に伴ってほぼリニアに性能が向上するのに対して、qsortは性能が飽和する傾向にある。この理由はqsortに関して並列性を引き出すためにAND並列を実現したが、台数の増加によるＰＥ間転送のオーバヘッドのほうが大きいためである。また、ＤＣＧの性能が低い理由は他と比べて１ゴール呼出し当りの起動される節の数は多いが、このうちの成功する節の数が小さいためである。

OR型PrologとGHCとを比較するにあたっては並列性の高い7-queens（決定的宣言の有・無について測定）と低いqsort(256個の要素の並べ換え)を選択した。そのときのＰＥの総実行命令数とＳＭの総操作指令数を表２に示す。

決定的宣言はゴール呼出しを行って得られる解がたかだか一つとわかっているときに明示的に用いることができ、ストリームを介した通信を行うのでなく解を直接親ゴールへ返す。従って、ＳＭへのアクセス回数を大幅に減少でき、決定的宣言を行わなかった場合に比して命令総数で２５%、指令総数で５０%減少する。これによりマシンサイクルが２０%短縮し、性能が５０%程向上した。

GHCプログラムはOR並列型より命令総数が５０%、実行サイクル数も同様に５０%程増加したが、性能は50%高くなっている。これはストリーム操作をユーザ定義述語として明示的に行っているためである(OR並列型では命令レベルでストリーム操作を行う)[4]。

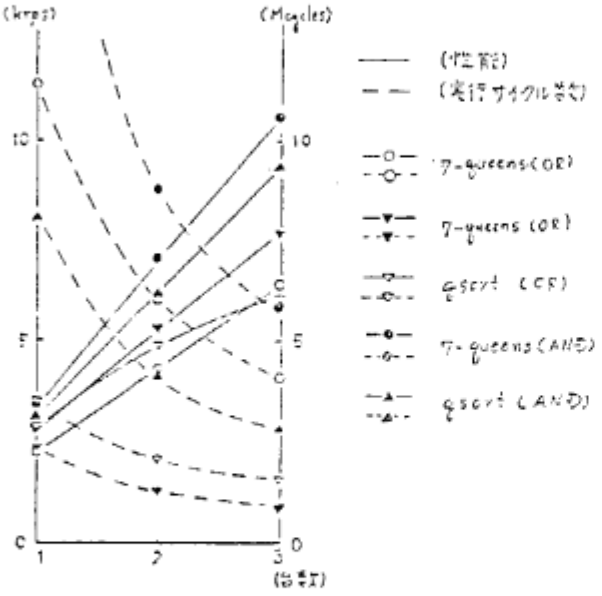


図2. OR並列型PrologとGHCプログラムの台数効果

qsortに関してはGHCの方がOR並列型Prologよりも高い台数効果が得られる。これはOR並列型がゴールレベルでプロセス間の同期をとるのに対して、GHCでは、ストリームを引数レベルで直接消費者プロセスに渡しているためである。

4．おわりに

並列推論マシン研究の一環としてPIM-D実験機を試作し、いくつかのサンプルプログラムに対する評価を行った。それによると、GHC及びOR型プログラムとも命令当りの平均所要サイクル数はほぼ一定である。ＰＥ台数を増加した際の並列処理効果はプログラム特性の影響を受ける。並列性の高いプログラムでは台数効果が得られ、処理方式の有効性が確認できた。今後、実験機の規模をＰＥ16台、ＳＭ15台まで拡張する予定であり、更に大規模システムの評価を行う予定である。最後に、日頃ご指導いただくICOT第４研究室の内田室長はじめとする関係各位に感謝する。

表２．命令実行数とＳＭ操作指令数

プログラム	言語	命令数	指令数
7-queens	OR型	440050	190163
7-queens (決定的宣言有)	OR型	336829	96949
7-queens	GHC	660752	205145
qsort	OR型	138283	42906
qsort	OR型	103243	36380

【参考文献】

[1]. Ueda, K., "Guarded Horn Clauses." ICOT TR-103, June, 1985.

[2]. 来住 他、「データフロー方式並列推論マシン実験機の構成」、情処３０全、６Ｃ－２．

[3]. 大原 他、「データフロー方式並列推論マシン」、情処３２全、２Ｒ－４．

[4]. 伊藤、「並列型Prologのデータフローマシンによる実行方式」、日経エレクトロニクス、１９８４年１１月５日号．

[5]. 佐藤 他、「データフロー方式並列推論マシン」、情処３２全、２Ｒ－１．

データフロー方式並列推論マシン

2R-1

―― ソフトウェアシミュレータによるGHCプログラムの評価――

佐藤 正俊　　伊藤 秘義 *　（ICOT）
末住 晶介　　久野 英治　　六沢 一昭 **　（沖電気）

1. はじめに

論理型言語Prologを並列で高速に実行するコンピュータの研究が現在さかんに行われている。ICOTでは、並列マシン上での実行言語としてAND並列型論理プログラミング言語を選択し、'84はConcurrent Prolog(CP)をベースに、'85は Guarded Horn Clauses(GHC)をベースに検討を進めてきた。筆者等はデータフロー方式に基づく並列推論マシン:PIM-Dの研究を行っており、PIM-D上でOR並列型Prolog及びCPの処理方式を検討すると共に、そのソフトウェアシミュレーションによる評価を行ってきた[Kumo 85]。新たに、PIM-D用GHCコンパイラを作成し、PIM-Dシミュレータをもとに OR並列型 PrologとGHC及びCPとGHCの比較評価を行ったので報告する。

2. PIM-D における各言語の特徴

OR 並列型 Prolog, CP, GHC はそれぞれ PIM-D のプリミティブをノードとするデータフローグラフにコンパイルされ PIM-D シミュレータ上で 実行される。このため各言語の特徴はデータフローグラフの違いとして現れ、この相違が実行時間に影響すると思われる。以下各言語の相違点の概略を述べる。

i) OR 並列型 Prolog

複数の節から成る定義を呼び出す場合、節の間の実行順序関係が規定されていなければ起動された節は OR 並列に実行される。この時これらの節の実行によって得られた解はシステムが用意したストリームを介して消費側のプロセスに渡される。

ii) CP

AND 並列型 Prolog であり AND 関係にあるゴールの間で論理変数を共有させると共に、ガード付コマンドと読出し専用変数の機能を用いてこれらのゴールにより起動されたプロセス間で共有変数に対する結合情報をインタラクティブに通信できる。即ち、呼出された節の頭部またはガードにおける統一化で、ゴールから渡された共有変数が非変数項に結合された場合、コミットオペレータの実行を契機として結合情報をゴール側に知らせる。このため共有変数に関する結合環境は、節の頭部及びガードを実行している間はその節でローカルに保存する必要がある。

iii) GHC

CP と同様に AND 並列型 Prolog であるが GHC においては頭部またはガードにおいてゴール変数に対する結合を許さずその結合は中断されるためローカルな結合環境を作る必要はない[Ueda 85]。従って、CP より処理効率が良いと期待できる。

3. PIM-D シミュレータ上での言語の特性

a) 評価プログラム

典型的な探索型の問題であり、その特性がよく知れている点で N queens プログラムを評価対象として選択した。Nの値はシミュレートする仮想マシンの規模がモジュール数 1 .. 64 台の範囲内であるので、この範囲内で並列性がほぼ飽和する 6 を選んだ。

b) 評価

6 queens プログラムの実行時間の台数効果について OR 並列型 Prolog, CP, GHC の測定結果を図1に示す。また PE 数が 16 台のときの 6 queens プログラムの実行時間（単位はマシンサイクル[Ito 84]）と実行命令数の比較を表1に示す。

表1　実行時間と実行命令数の比較

	実行時間	実行命令数
OR 並列型 Prolog	57446 (0.8)	100912 (0.6)
CP	185165 (2.5)	327449 (2.1)
GHC	73552 (1.0)	156116 (1.0)

( )内は GHC を 1 とした時の値

以下シミュレーションデータに基づいて CP と GHC, OR 並列型 Prolog と GHC の比較を行う。

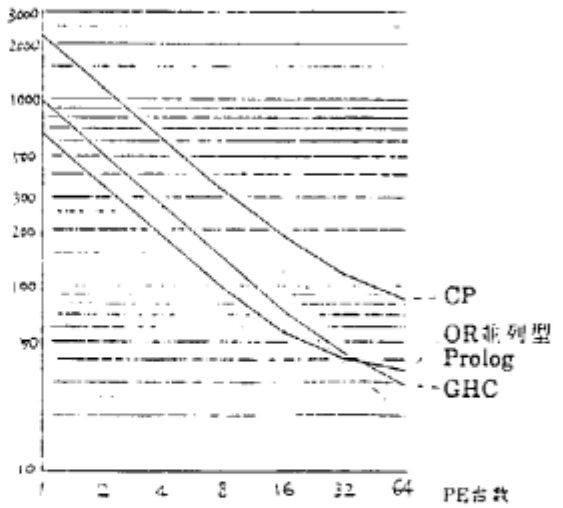


図1　6 queens プログラムの実行時間比較

* 1986年 1月より沖電気　** 1986年 1月よりICOT

i) CP と GHC の比較

CP, GHC ともに AND 並列型 Prolog であるが, 主な相違点はローカル環境を作成するか否かにある。つまり CP においてはローカル環境を LM (Local Memory) 上に作成するため, LM アクセス及びその制御のための処理がオーバヘッドとなっている。PE 数が 16 台のときのローカル環境を作成のための LM アクセス時間と生成トークン総数の比較を表2に示す。表2よりローカル環境を作成のための影響がわかる。

表2 LM アクセス時間と生成トークン総数の比較

	LM アクセス時間	生成トークン総数
GHC	0	156116 (1.0)
CP	217303	482874 (3.1)

( )内は GHC を 1 とした時の値

ii) OR 並列型 Prolog と GHC の比較

OR 並列型 Prolog と GHC の相違は探索処理をOR 並列型で実行するか AND 並列型で実行するかの違いであるが PIM-D では, いずれの処理もストリーム通信を介したプロセス間の並列処理モデルによって統一的に実現している。但し, OR 並列型 Prolog ではシステムがシステムストリームを使用しプリミティブレベルで暗黙的に実行するのに対して, GHC ではユーザが明示的にプログラムしなければならない点で異なる。この相違は表3に示すようにゴールの呼出し回数及び実行命令数として現れる。

表3 ゴールの呼出し回数及び実行命令数の比較

	呼出し回数	実行命令数
GHC	5616 (1.0)	156116 (1.0)
OR並列型Prolog	4252 (0.8)	100912 (0.6)

( )内は GHC を 1 とした時の値

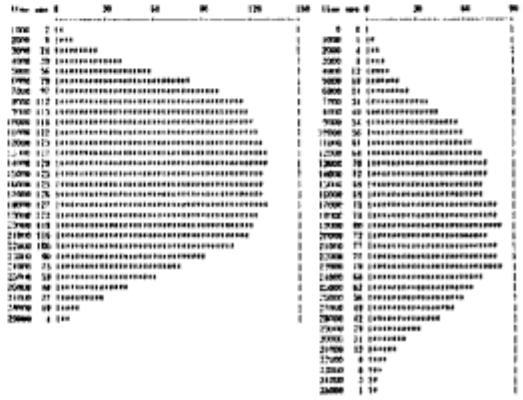
GHC の解の操作のための述語は merge述語と append述語でありこれらのゴールの呼出し回数(1576) を GHC の全ゴールの呼出し回数より差引いた値(4040)はほぼ OR 並列型 Prolog のゴールの呼出し回数(4252) と等しいと考えられる。また実行命令数については merge述語, append述語ともに約20 ステップであり, 再帰実行時の命令ステップはともに約30 ステップである。これより merge述語と append述語の実行命令数はそれぞれ merge 述語で 7970 , append述語で 38790 となり合計で 46760 と推定できる。つまり実行命令数についても GHC の解の操作以外の実行命令数と OR 並列型 Prolog の実行命令数がほぼ等しいと考えられる。

iii) 飽和の傾向について

図1が示すように飽和の傾向は PE 台数 16 ぐらいから始まるが言語によってその傾向がやや異なる。特に OR 並列型 Prolog と GHC の傾向が異なっている。以下 OR 並列型 Prolog と GHC の場合を考察する。

OR 並列型 Prolog と GHC ともに PE 台数による実行命令数の変化はない。ただし, GHC の場合は PE 台数 64 の場合 PE 台数 16 に比べて若干増加しているが 0.1% 未満の増加でありほぼ同じと考えられる。この GHC での PE の台数による実行命令数の変化は merge 述語の実行回数が多少変るためである。即ち, 主としてネットワークの遅延により, merge 述語における非決定性が実行回数の変化として現れるからである。

実行命令数の時間的変化を図2に示す。この図は PE 台数が 64 のときの 1000 マシンサイクルあたりに各 PE で実行された平均命令数(ops)を 1000 マシンサイクル経過毎に表示している。図2より GHC と OR 並列型 Prolog のシミュレーション上での並列性の現れかたの違いがわかる。つまり PIM-D においては GHC は引数レベルで OR 並列型 Prolog はゴールレベルで同期をとっているため並列性が PE 台数に比べて低い場合には並列性の現れかたの違いが顕著に現れてしまい, GHC の実行時間が短くなっている。



i) GHC　　ii) OR 並列型 Prolog

図2　実行命令数の時間的変化

4. おわりに

3種類の並列型論理プログラミング言語についてその処理性能をPIM-Dシミュレータ上で 6 queens プログラムに対して評価した。その結果GHCはCPに比べて約2倍またOR並列型Prologに比べて約0.8倍の処理性能であった。またここでは触れなかったが他のプログラムについてもほぼ同様の結果を得ている。

今後はストリーム処理における最適化及びプリミティブレベルの引き上げによる最適化等によりPIM-DにおけるGHCの処理方式をさらに検討して行く予定である。

最後に, 日頃ご指導をいただく内田第4研究室長はじめ並列推論マシングループ諸氏に感謝の意を表する。

(参考文献)

[Kuno 85] 久野他, 「データフロー方式並列推論マシンにおけるAND/OR並列効果のシミュレーションによる測定」情報処理第30回全国大会予稿集.

[Ito 84] 伊藤, 「並列型Prologのデータフローマシンによる実行方式」日経エレクトロニクス,1984年11月5日号.

[Ueda 85] Ueda.K., "Guarded Horn Clauses," ICOT TR-103, June, 1985.

関係データベースマシンにおける
3C-10 関係代数演算専用エンジンの並列処理評価

伊藤 英則　　安部 公朗　　角田 健男

（財）新世代コンピュータ技術開発機構

1．はじめに

ICOTの第5世代コンピュータ・プロジェクトは、推論機構と知識ベース機構を兼ね備え、高度な知識情報処理システムの構築を目指している。前期3年間において、知識ベース機構については、Prologなどの論理型言語との親和性を生かし、バックエンド型関係データベース・マシン"Delta"を開発した[1][2]。本稿では、関係代数演算専用エンジンを複数台持つような"Delta"を抽象化したバックエンド型関係データベース・マシンを想定する。それらのエンジンが並列に動作して関係代数演算を行う時に、実際の処理系の実測データ・経験データを用いて、シミュレーションにより評価を行い、応答特性について考察する。

2．関係代数演算専用エンジンの関係代数演算処理方式

関係代数演算を高速に実行するためには、演算の対象となる属性を前もってソートしておくと効率が良い。処理負荷の重いJOINなどは、特にその効果が大きい。そこで我々は、ソート処理を高速に行うために、ソートすべきデータをストリーム状に流し、ストリームの転送時間とオーバラップしてデータをソートするPipelined 2-way merge sortアルゴリズムを採用し、専用ハードウェア"ソータ"を開発した。これにより、ソーティングのオーダをO(NlogN)からO(N)に落すことが可能となった。また、ソートされたストリームに対して、関係代数演算を高速に実行するために、ソータの後段にその専用ハードウェアである"関係代数演算処理部"を置いた。ソータは、さらにn段のプロセッサと、各プロセッサの持つメモリ部からなる。実際に実現したソータは、n = 12で、最終段のプロセッサのメモリ部のサイズが64KBである。従って、64KB以下のデータを一度にソート出来る。関係代数演算処理部は、既にソートされたリレーションを格納する2つのメモリと、条件を満たすレコードだけを選択出力するプロセッサからなる。JOINやSELECTIONなどの関係代数演算は、関係代数演算処理部で実行される。実際に実現した関係代数演算処理部の2つのメモリサイズは64KBであるので、64KB以下のリレーションと64KB以下のリレーション（あるいは定数）のJOIN,SELECTIONが一度に実行できる。

3．データ分割並列処理方式

3．1　データ分割ソート演算

大量データをソートする時、ソートすべきデータを空きエンジンの台数分で分割し、各分割データを空きエンジンに割当て並列実行することができれば、実行時間を著しく短縮することができる。これをデータ分割ソート演算と呼ぶ。データ分割ソート演算は次のステップで実行される。

① ソートすべきデータの長さをLとし、空きエンジンの台数をmとした時、データをm分割してL／mのデータをそれぞれのエンジンに割当て、並列にソートを行う。

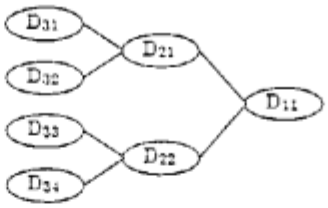
② ソート済みのデータを2個づつ集めてP組のペアーを作り、エンジンの関係代数演算処理部で2-way mergeを行うことにより、P個のソート済みのデータを出力する。この処理を空きエンジンで並列に行い、Lの長さのデータがソートされるまで繰返す。

3．2　データ分割JOIN(SELECTION) 演算

2つのリレーションのデータサイズをそれぞれL1, L2とした時、両者のJOINをL1⋈L2と表わす。これをm台の空きエンジンで並列実行させるためには、片方のリレーションをm分割し、L1/a⋈L2のJOINを1台のエンジンに割当することを考える。これを、データ分割JOIN演算と呼ぶ。同様な方法で、SELECTIONを並列実行させる場合をデータ分割SELECTION演算と呼ぶ。

3．3　データ分割サブコマンド木

データ分割ソート／JOIN／SELECTION演算は、図1のようなデータ分割サブコマンド木に変換され実行される。



D_{31}〜D_{34}は、2MBのソート処理に対応。
D_{21}, D_{22}は、2MBと2MBのマージ処理に対応。
D_{11}は4MBと4MBのマージ処理に対応。

図1　8MBのソートのデータ分割サブコマンド木(n = 4)

データ分割サブコマンド木は、葉から根に向かって処理が進み、同じ深さのノードは並列に実行可能である。各ノードは、前のノードの実行が全て終了しないと、実行を開始できない。

4．関係データベースマシン・シミュレーションモデル

本稿で対象としている関係データベースマシンの概念図を図2に示す。インタフェース部は、ホストマシンから問合せコマンド$\mathcal{C}$を受取り、$\mathcal{C}$をコントロール部に送る。また、$\mathcal{C}$のレスポンス・データをホストに返す。コントロール部は、インタフェース部より送られてくる$\mathcal{C}$を解析しソート処理やJOINなどの関係代数演算レベルのコマンドCに変換し、スケジュール部に送る。スケジュール部は、空きエンジンの状況を監視して、エンジン並列制御戦略によりエンジンの並列実行制御を担当する。また、それにより選ばれたCをデータ分割サブコマンド木に変換する。階層構造メモリ(HM)部はリレーションの格納場所であり、演算の

対象となるリレーションをディスクからバッファメモリ上へのステージング、リレーションの分割処理を担当する。

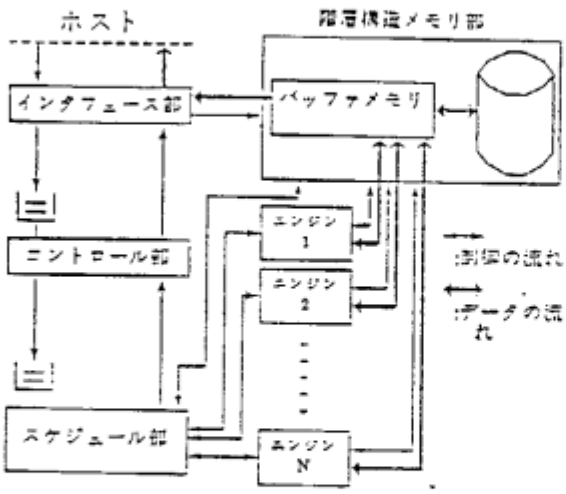


図2　関係データベース・マシンの概念図

5.　エンジン並列制御戦略

エンジン並列制御戦略として、以下の3つを考える。

(1) データ非分割先着優先戦略

どのコマンドもデータを分割せずに、スケジュール部のキューに到着した順にm台の空きエンジンを割当てる。

(2) データ分割先着優先戦略

スケジュール部のキューから先着順にコマンドを1つづつ取り出し、空きエンジンの台数分データを分割し、データ分割サブコマンド木に変換し、実行する。ただし既にデータ分割サブコマンド木に変換され、処理が根にまで至っていない木が存在するならば、それらの木のうち到着の早い木のノードの処理を優先して割当てる。

(3) データ最多分割先着優先戦略

(2) でデータを分割する際、実装されているエンジンの総台数分で分割する。あとの条件は(2) と同じ。

6.　目的関数

目的関数として、次の2つを考える。

(1) 平均待ち時間（平均レスポンスタイム）

(2) 平均HHメモリ使用量

(3) 平均エンジン稼動率

7.　シミュレーションの条件

(1) 資源の条件

① インタフェース部、コントロール部、スケジュール部、HH部は、5MIPS の処理能力を持つ1CPU構成とし、エンジンは、その付加プロセッサとする。

② エンジンの台数は、16台とする。

③ HHのバッファメモリサイズは、128MB とする。

④ HHメモリとエンジン間、及びHHメモリとディスク間データ転送速度は3MB/S とする。また、HHメモリとエンジン間は、入力1本、出力1本、合計2本のチャネルを具備するものとする。

(2) 処理時間に関する条件

Delta で実測可能な時間は、実際の処理系の時間を使用する。それ以外の処理時間は、実測値からの推定、ダイナミックステップ数の見積りにより求めた。

(3) ホストマシンからの問合せコマンドに関する条件

問合せコマンドの到着は、ポアソン到着を仮定し、ソート、JOIN, SELECTIONが等確率に1つづつ到着するものとする。また、演算の対象となるリレーションのサイズは、1MB から16MBまでランダムに選ぶ。

8.　評価結果

シミュレーション結果を図3、図4、図5に示す。

9.　考察とまとめ

図3より、トラフィックが疎の時には1つの演算をより多くの台数で実行した方が効率が良いが、密の時にはデータを分割しないで実行した方が良い効率が得られた。この傾向は、図4についても同様である。これらは、トラフィックの疎密により戦略を変える必要があることを示す。

今後、この評価結果をもとに、中期に開発を予定している知識ベースマシンのユニフィケーション・エンジン並列制御方式の検討を行う予定である。

［参考文献］

[1]Kakuta T., et al. The Design and Implementation of Relational Database Machine Delta, Proc. IWDM'85,(1986)

[2] 安部他，関係データベースエンジンのハードウェア構成とその制御方式，情報処理学会研究会42-2(1985)

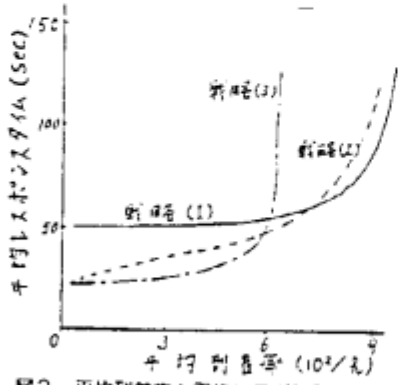


図3　平均到着率と平均レスポンスタイム

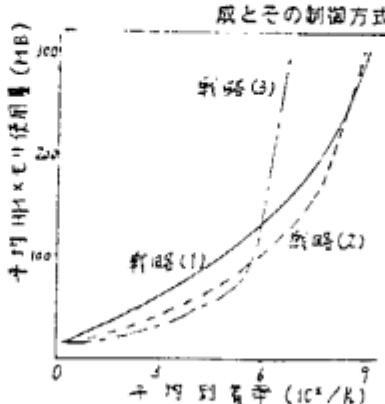


図4　平均到着率と平均HHメモリ使用量

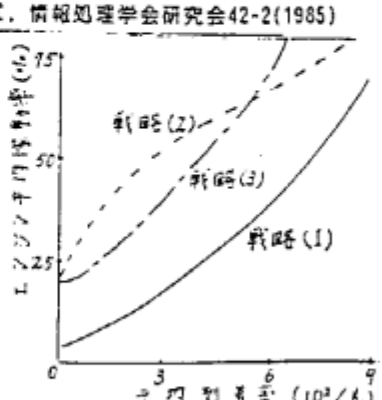


図5　平均到着率と平均エンジン稼動率

単一化エンジンを用いた知識ベース・マシンの構成方法

横田　治夫　　村上　昌己　　森田　幸伯　　伊藤　英則
(財) 新世代コンピュータ技術開発機構

1. はじめに

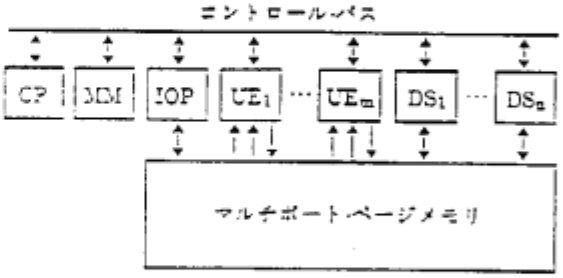
計算機で知識を取り扱う研究が，注目されている。そのような知識情報処理系では，大量の知識の中から必要な知識を効率良く検索する機能と，検索された知識を用いて高速に推論する機能とが重要となる。第五世代コンピュータシステム研究開発プロジェクトでは，それぞれの機能実現のため，知識ベース機構と推論機構の構築をめざしている。我々は，知識ベース機構実現の一つのアプローチとして，関係型知識ベース・モデルを提案した[1]。本稿では，関係型知識ベース・モデルに基づく知識ベース・マシンの構成方法と，そのマシンの中心的な構成要素であるストリーム処理を基本とする単一化エンジンの実現方法を提案する。

2. 関係型知識ベース・モデル

データと知識の違いは，データが必要な情報をそのまま表現しているのに対し，知識は相互に関連し合った必要な情報を適当な構造に置き換えて表現しているものといえる。我々は，大量の知識を格納するため，関係データベースの各アイテムを定数から明確に定義された構造体である項に拡張して，単一化(unification) を用いて項を検索するモデルを提案した。項が格納されたテーブルのことを項関係，関係データベースにおける結合および制約演算のイクオリティ・チェックを単一化操作に拡張した演算のことをそれぞれ単一化結合(unification-join)，単一化制約 (unification-restriction)，それらの検索処理全体をＲＢＵ(Retrieval By Unification) と呼ぶ[1]。

3. 知識ベース・マシンの全体構成

関係型知識ベースをサポートするハードウェアとして，図１に示すような構成の知識ベース・マシンを提案する。２次記憶に格納された項関係に対し，高速にＲＢＵ操作を行うため，ディスク・システムから単一化エンジンに項の集合をストリームとして流すことを前提としている。図中のマルチポート・ページメモリは，複数の処理装置からページ単位で同時にアクセスしても競合が起こらないように設計された共有メモリである[2]。本システムでは，ディスク・キャッシュとして，あるいは単一化エンジン用のバッファ・メモリとしてマルチポート・ページメモリを用いている。紙面の関係上，マシンの全体構成の詳細は割愛するが，全体アーキテクチャについては[3] を，システム全体の制御については[4] を参照されたい。



UE: 単一化エンジン(Unification Engine)
DS: ディスクシステム(Disk System)
CP: 制御プロセッサ(Control Processor)
MM: 主メモリ(Main Memory)
IOP: 入出力プロセッサ(I/O Processor)

図１．知識ベース・マシンの構成

4. 単一化エンジン

4.1 エンジンの構成

単一化エンジンの構成を図２に示す。単一化エンジンは，単一化結合を前提として構成するため，マルチポート・ページメモリとの間で，同時に２つの入力ストリームと１つの出力ストリームを処理できるよう，３本のデータパスを持つ。単一化結合での単一化の項の組合せは，本来２入力ストリーム中の項の数の積になる。項間にある順序付けを行うことにより，この組合せの数を減らすことができる。可変長ソータとペア・ジェネレータはそのためのユニットである[5]。単一化ユニットは，ペア・ジェネレータにより生成された２項間の最汎化作用素(mgu)を求める。また，前処理ユニットは項関係のタプルから処理対象のアイテムを切り出し，後処理ユニットは単一化ユニットで得られたmgu をもとのタプルに適用するためのユニットである。

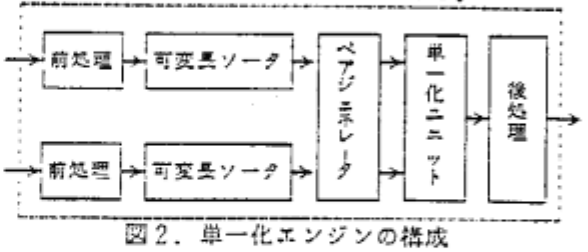


図２．単一化エンジンの構成

4.2 単一化アルゴリズム

一般的なPROLOG処理系などでの単一化のアルゴリズムは，構造体共有方式にしろ構造体コピー方式にしろ，変数ごとにバインドすべき相手をポインタでつなげてゆくことを基本としている。そのようなポインタをたどる操作は，ストリーム処理を基本とする専用ハードウエアには向いていない。

もともと，単一化アルゴリズムは図3のように書くことができる[6]。図3のアルゴリズムの繰り返しの部分(Step2—Step4)をハードウエア化し（単一化エレメントと呼ぶ），それを直列に接続して（図4），複数の項の組合せを流すことにより，単一化処理をパイプラインで行うことができる。

Step1: $k=0, W_k=W, \sigma_k=\varepsilon$
　ここでWは単一化の対象となる項の集合。
Step2: W_kが1つの項のみからなる場合
　—単一化終了(σ_kがmgu)
　それ以外の場合
　—W_kの中の食い違い集合D_kを求める。
Step3: D_kの中の要素v_kとt_kについて，
　v_kがt_kの中に含まれない変数の場合
　—Step4へ
　それ以外の場合
　—単一化不可能
Step4: $\sigma_{k+1}=\sigma_k\{t_k/v_k\}$,
　$W_{k+1}=W_k\{t_k/v_k\}$とする。
Step5: $k=k+1$としてStep2へ。

図3．単一化アルゴリズム

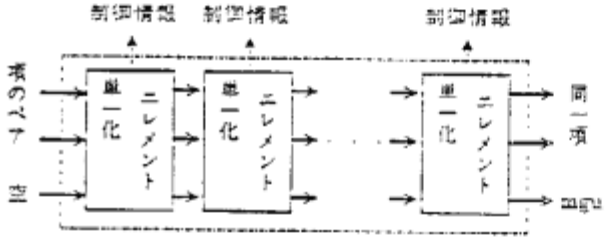


図4．単一化アルゴリズムのハードウェア化

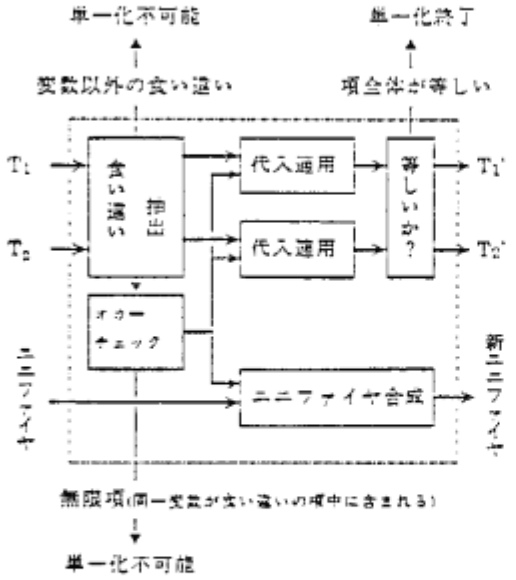


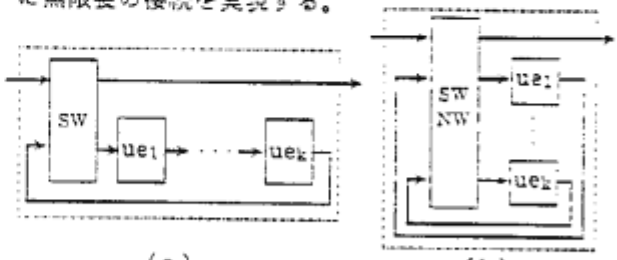
図5．単一化エレメントの内部構成

4.3 単一化エレメント

図4中の1つの単一化エレメントの内部構成を図5に示す。これは，図3のアルゴリズムをそのままハードウエア化した形になっている。Stepに対応したブロック構成で，単一化エレメント中もパイプライン化される。

4.4 単一化ユニットの構成案

実際に単一化ユニットを構成する場合，項中の変数の数に制限がないと，無限個の単一化エレメント(ue)をつなげる必要がある。これは実質的に不可能なため，図6のaやbのように切り替えスイッチやスイッチング・ネットワークなどを使って，仮想的に無限長の接続を実現する。



(a)　(b)

図6．単一化ユニットの構成案

5．おわりに

関係型知識ベース・モデルに基づく知識ベース・マシンの構成方法と，その基本構成要素の単一化エンジンの実現方法について述べた。全体構成の方針は，二次記憶に格納された大容量の処理対象を専用装置で処理するようなシステム一般に適用できる。また，単一化エンジンも，知識ベース・マシンだけでなく，各種の知識情報処理システムで利用できる。今後は，知識ベース・マシンの実現方法についてより詳細に検討を進めていく。

[謝辞] 本検討を進めるに当たり，有益な御示唆を頂いた知識ベースマシンWGのメンバの方々に感謝します。

[参考文献]

[1] 横田，他：単一化による知識ベース検索，60年度通信学会部門別全国大会，S12-2.
[2] 横田，他：マルチポート・ページメモリの構成方法，60年度通信学会部門別全国大会,602.
[3] Yokota and Itoh: A Model and an Architecture for a Relational Knowledge Base, ICOT TR-144 (submitted to the 13th Int. Sympo. on Computer Architectuer).
[4] 物井，他：マルチポート・ページメモリを用いた大規模知識ベースの制御方式，情処第32回全国大会予稿，5Q-5.
[5] 森田，他：単一化結合の処理方式，情処第32回全国大会予稿，1M-7.
[6] Chang and Lee: Symbolic Logic and Mechanical Theorem Proving, Academic Press.

単一化結合の処理方式

1M-7.

森田　幸伯、横田　治夫、西田　健次、伊藤　英則

（財）新世代コンピュータ技術開発機構

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中期の目的の1つとして知識ベースマシンの開発があげられる。知識処理を大規模なシステムで考えた場合、データ処理におけるデータベースのように、知識を効率的に管理・共有できるサブシステムが必要である。本稿では、そのようなサブシステムを効率よく実現するマシンを知識ベースマシンと呼ぶ。知識ベースマシンは、さまざまなニーズ／ホストが利用するため、その概念構造は柔軟性に富んでいることが望ましい。[1]で提案されている関係型知識ベースモデルは、知識ベースマシンに対する非常に柔軟な概念モデルとなっている。しかし、そこで提案されているＲＢＵ演算、特に単一化結合を単純に行うとマシン側の処理量が非常に大きくなる。本稿では、単一化結合を効率よく行うための処理方式およびその一実現方式について報告する。

2．関係型知識ベースモデル

関係データモデルでは、関係の各属性(attribute)の領域(domain)をアトミックなものの集合に制限している。

それに対して関係型知識ベースモデルは、関係の各属性の領域として項の集合を許した項関係を扱う。項関係上の操作としては、関係代数のイクォリティチェックを単一化操作（unification）に拡張した単一化結合(unification-join)、単一化制約(unification-restriction)を用いる。これらをＲＢＵ(Reirieval By Unification)演算とよぶ。射影(projection)は関係代数のそれと同じである。

3．単一化結合の処理方式

云々は、非常に大量な知識が知識ベースマシンに格納されていることを仮定している。その格納場所としては2次記憶を想定している。

最も単純に考えれば、単一化結合は2つの項関係のタプル(tuple)のすべての組合せに対して指定されたアイテムが単一化可能などうかをを調べればよい。しかし、すべての組合せを生成することは、Ｉ／Ｏ回数も増え非常に大きな処理負荷がかかる。そこで項関係のタプルを何等かの順序で順序付けることにより組合せを省略したい。

項を順序付けるるためにジェネラリティの概念を導入する[1]。ただ、ジェネラリティの順序付けは、全順序ではないため、ジェネラリティの順序付けを保存しつつ、全順序にする方法を考える。そこで、項が木で表現されることに注意し、その木の線形化しその辞書的順序を用いることにする。

木の線形化としては、深さ優先のものと、レベル優先のものがある（図1）。本稿では深さ優先のものをleftmost方式の順序と呼び、レベル優先のものをoutermost方式と呼ぶ。[1]で定義した順序はleftmost方式の1つである。但し、関数記号はそのアリティにより区別され、関数記号どうしは適当な順序があり、変数はその出現位置によって順序を付け、関数記号と変数は変数の方がよりジェネラルであるとする。

f(g(Y),a)	f2,g1,Y,a	f2,g1,a,Y
項	深さ優先	レベル優先

図1．項の表現の例

このように（ジェネラルな順に）順序付けられた項の列に対して、文字列どうしの比較操作だけで単一化可能な可能性のある項のペアを選びだしたい。

項を文字列で表現したとき単一化可能なものは先頭から見てどちらかに変数が現れるまで文字列は一致しているはずである。文字列が順序付けられているので、変数より前方の文字列が一致しなくなるまで、その項をとっておきペアをつくりだす。異なったらそれ以降の項の列に対しては、単一化は不可能なので組合せを省略できる。

省略の仕方は順序付けの方式によって異なる。

組合せの省略の例を図2に示す。■は単一化可能な項の組を、□はペアとして選びだされたが単一化不可能な組合せを、◇は省略された組合せをしめす。図2－aがleftmost方式、図2－bがoutermost方式である。

オーダリングの概念は、2次記憶の項のクラスタリングなどにも有効な概念とおもわれる。

4．単一化エンジンの一実現方式

4．1．単一化エンジンの構成

単一化エンジンの構成を図3に示す。単一化エンジンは、次の5つのものからなる。

項関係のタプルから処理対象のアイテムを切り出す前処理ニット、オーダリングに従い項をソートする可変長ソータ、ソートされた2つの項の列を受け取り、単一化の可能性のある項の組（ペア）を生成するペアジェネレータ、生成された項の組の最汎化作用素（ｍｇｕ）を求める単一化ユニット、さらに最汎化作用素をもとのタプルに適用する後処理ニットである。

本稿では、上記のうちソータおよびペアジェネレータについて述べる。この2つのユニットで、前節で述べた方式に従って組み合わせの省略を行う。

単一化ユニットに関しては、別稿[2]を参照されたい。

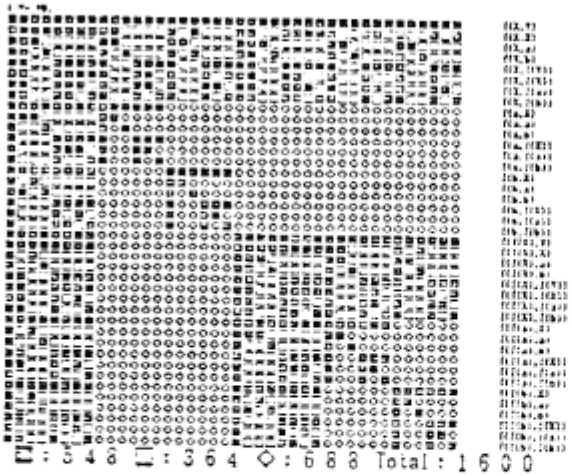


図２－ａ．組合せの省略の例(leftmost方式)

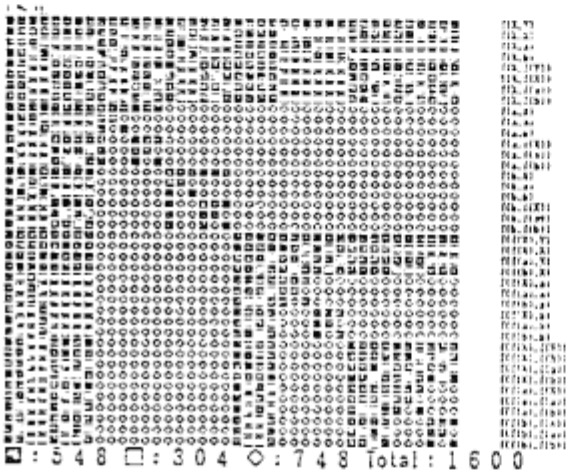


図２－ｂ．組合せの省略の例(outermost方式)

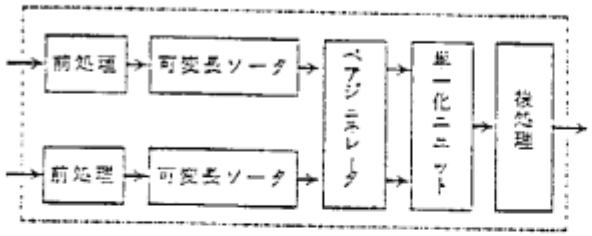


図３．単一化エンジンの構成図

４．２．ソータ

ソート部は、可変長の文字列を扱うソータで、方式としては２ウエイのマージソートを用いる。ハードウェアソータの方式としては、マージソート、バイトニックソート、ヒープソート等があるが、我々は可変長の２ウェイマージソート方式を採用する。項のソートのためには可変長文字列のソートを行わなければならない。可変長のハードウェアソータとしてはパイプラインドヒープソータ[3]が提案されているが、ストリーム型のデータ転送を基本と考えたときには、マージソータが適していると思われる。可変長の文字列を文字単位に巻戻しなしにストリームタイプで処理するために、文字列の線形化Ｔｒｉｅ表現を用いる[3]。

線形化Ｔｒｉｅ表現とは、文字列を文字と位置情報の組の列で表現し、直前の文字列と同じものは省略して表わす方式である。項の列を線形化Ｔｒｉｅ表現で表した例を図４．に示す（例における項の順序は、leftmost 方式である）。

```
f(X,Y),f(X,g(X,b)),f(X,g(a,b)),f(a,g(a,b))

1f2,2X,3y,3g2,4X,5b,4a,5b,2a,3g2,4a,5b
```

図４．項の列の線形化ｔｒｉｅ表現

４．３．ペアジェネレータ

ペアジェネレータの構成を図５に示す。ペアジェネレータは、必要な項を単一化の可能性が無くなるまで、スタックに積んでおき、入力された項と比較し文字列の比較で単一化不可能とされるもの以外の項のペアを全て出力する。

なお、出力の際には、項の表記方法を単一化ユニットで取扱いやすい形に変換する。

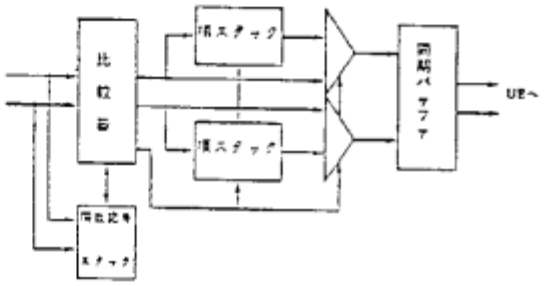


図５．ペア生成部の構成図

５．まとめ

ＲＢＵ方式に基付く処理方式および実現方式の提案した。今後は提案したアルゴリズムやエンジンについてシミュレーション等により評価を行っていく予定である。

［謝辞］本検討を進めるに当たり、有益なご示唆を頂いた知識ベースマシンＷＧのメンバの方々に感謝いたします。

［参考文献］

［１］Yokota,H. "A Model and an Architecture of a Relational Knowledge Base", submitted to the 13th in.Simp. Computer Architecture,1986.

［２］横田他、「単一化エンジンを用いた知識ベースマシンの構成方式」第32回情報処理学会全国大会（1986）

［３］Tanaka,Y."A VLSI Algorithm for Sorting Variabl e-Length character Strings.", New Generation Computin g,1985.

単一化による関係型知識ベース操作

−関係型知識ベースにおけるホーン節集合の更新−

Manipulation of Relational Knowledge Base based on Unification Operation

− Update of Horn Set in Relational Knowledge Base−

坂間 千秋' 村上 昌己" 横田 治夫"

Chiaki SAKAMA Hasaki HURAKAMI Haruo YOKOTA

(株)東芝 情報システム研究所' (財)新世代コンピュータ技術開発機構(ICOT)"

1. はじめに

第五世代コンピュータプロジェクトでは、大量の知識を効率的に管理、運用するための知識ベース・メカニズムの開発を進めている。

その方法として、知識を関係モデルの上で表した関係型知識ベースモデルが提案されている[1]。また関係型知識ベースの上の演算として、単一化制約、単一化結合を導入し、これらの演算を用いて関係型知識ベースに格納されたホーン節集合の上の入力演繹が実現できることが示されている。

関係型知識ベースにおける更新を考えたとき、その基本操作としての関係演算が必要となる。本稿では、新たに単一化可能抽出、単一化可能例なる関係演算を導入し、これらを用いた関係型知識ベースにおけるホーン節集合の更新について述べる。

2. 関係型知識ベース演算

知識を関係モデルの上で表現した、関係型知識ベース(以下、RKB と略す)は項関係

$T \subset K_1 \times K_2 \times \cdots \times K_m$ (Ki:Term, T:term relation)

の集合と定義され、項関係は項タプルの集合

$T = \{(t_1, \ldots, t_n) \mid t_i \in \text{Term}\}$

と定義される。(ここで、Termは項の集合とする。)

RKB に対する演算は関係データベースに対する演算である関係代数の拡張として、関係における各要素間のイクオリティ・チェックを単一化に拡張したものが考えられている。以下では、関係代数における集合演算の単一化可能性による拡張として2つの新しい演算を導入する。

Def.1 単一化可能抽出(unifiable extraction)

$R \triangleright S = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R, {}^{\exists}(y_1, \ldots, y_n) \in S, {}^{\exists}\lambda \in \text{Sub}, \lambda(x_1, \ldots, x_n) = \lambda(y_1, \ldots, y_n)\}$

Def.2 単一化可能例(unifiable instance)

$R \blacktriangleright S = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in R, {}^{\exists}(y_1, \ldots, y_n) \in S, {}^{\exists}\lambda \in \text{Sub}, (x_1, \ldots, x_n) = \lambda(y_1, \ldots, y_n)\}$

ここで R,Sは項関係、$(x_1, \ldots, x_n), (y_1, \ldots, y_n)$ は項タプル、Sub は置換の集合を表すものとする。また、項タプルの単一化はタプル中の項が単一化可能であるときに定義される[2]。

3. ホーン節集合の更新

関係型知識ベースにおいては、ホーン節はヘッド、ボディからなる2属性の項関係によって格納される。即ち、ホーン節のうち確定節(Definite clause)の集合S

$$S = \{P_1 :- Q_{11}, Q_{12}, \ldots, Q_{1m_1}.,\ P_2 :- Q_{21}, Q_{22}, \ldots, Q_{2m_2}.,\ \ldots,\ P_n :- Q_{n1}, Q_{n2}, \ldots, Q_{nm_n}.\}$$

は項関係、

$$S = \{([P_1 \mid X_1], [Q_{11}, Q_{12}, \ldots, Q_{1m_1} \mid X_1]), ([P_2 \mid X_2], [Q_{21}, Q_{22}, \ldots, Q_{2m_2} \mid X_2]), \ldots, ([P_n \mid X_n], [Q_{n1}, Q_{n2}, \ldots, Q_{nm_n} \mid X_n])\}$$

と表され、ゴール節(Goal clause) G

$$?- G_1, G_2, \ldots, G_n.$$

は項関係、

$$G = \{([G_1, G_2, \ldots, G_n])\}$$

と表される。

以下では、前節で定義した関係演算によるRKB におけるホーン節集合の更新について述べる。

3.1 追加

いまホーン節集合Sに、ある限定節を追加してS'とするとき、

$${}^{\forall}R\ (S \vdash_v R \Rightarrow S' \vdash_v R) \quad (\vdash_v : \text{valid})$$

であることが必要条件である。

即ち、Sから演繹可能なすべての導出形RはS'からも演繹可能でなければならない。項関係S に項タプルCを追加して S'とするときのアルゴリズムは以下のようになる。

```
insert(S,C,S'):
 begin
  P←true
  for all T included in S do
   begin
    if (deducible(S-T,T)=false
        or deducible((S-T)∪ C,T)=false)
     then P←false
   end
  if P = true
   then S' ←S ∪ C
  else
   S' ←S
 end
```

ここで、∪、−は関係代数における集合和、集合差を表わす。

上で deducible(S,T) はS からT が演繹可能なとき true、そうでないときは falseとする。そのアルゴリズムは以下の通り。

```
deducible(S,T):
  begin
   S0 ←S
   S0' ←S
   R0 ←φ
   i=0
   if T▶S0 ≠φ
    then deducible(S0,T)=true
   else
```

```
    while S_{i+1} ≠ S_i do
     begin
      R_i ← π_{...}(S_i' ⋈_{...} S_i)
      S_{i+1} ← S_i ∪ R_i
       if  T ▶ S_{i+1} ≠ φ
        then deducible(S_{i+1}, T)←true
       else
          S'_{i+1} ← S_i, i ← i-1
      end
   end
```

ここで、⋈、πはそれぞれ関係代数における結合、射影を表し、 A⋈B は Aと Bの単一化を表すものとする。

いま、RがSに対して冗長であるとは

$$S \vdash_R \quad \vee \quad S \vdash_{\bar{R}} \qquad (\ \vdash:\text{satisfiable})$$

を満たすものとすると、追加によって項関係 Sに冗長性が発生したとき、S から冗長性を除去して S´とするアルゴリズムは以下のようになる。

```
redundancy(S, S´):
 begin
  S´←S
   for all T included in S do
    begin
     if deducible(S-T,T)=true
      then S´ ← S´ − T
     else if satisfiable(S-T,T)=true
      then S´ ← S´ − (S▶T)
    end
 end
```

上で satisfiable(S,T) はS においてT が充足可能なときtrue、そうでないときは falseとする。そのアルゴリズムは、deducible(S,T)において単一化可能射を単一化可能抽出に置き換えることによって得られる。

なお、冗長性の除去については処理効率の問題で知識ベース管理者の判断に委ねられる。

3.2 削除

いまホーン節集合Sから、確定節Cを削除してS´とするとき、

$$\forall R\ (S \vdash_R \wedge\ S - C \vdash_R \Rightarrow\ S' \vdash_R)$$

であることが必要条件である。

即ち、Sからの演繹過程においてCを定理として必ず用いるようなすべての導出形Rは、S´からは演繹不可能でなければならない。項関係S から項タプルC を削除して S´とするときのアルゴリズムは以下のようになる。

```
delete(S,C, S´):
 begin
  S´ ← S−C
  for all T included in S do
   begin
    if(deducible(S-T,T)=true
       and deducible((S-T)-C,T)=false)
     then S´ ← S´−T
    else
      S´ ← S´
 end
```

3.3 変更

いまホーン節集合Sの中のある確定節Cを変更してSをS´とするとき、

$$\forall R\ (S \vdash_R \wedge\ S - C \vdash_R \Rightarrow\ S' \vdash_R)$$
$$\wedge\ \forall R\ (S - C \vdash_{R'} \Rightarrow\ S' \vdash_{R'})$$

であることが必要条件である。

即ち、Sからの演繹過程においてCを定理として用いるような導出形RでS´からは演繹不可能なものが存在し、それ以外のすべての導出形R´はS´からも演繹可能でなければならない。項関係S 中のある項タプル Cを C´に変更して Sを S´とするときのアルゴリズムは以下のようになる。

```
change(S,C, C´,S´):
 begin
  S´←(S−C)∪ C´
  for all T included in S do
   begin
    if(deducible((S-T,T)=true
       and deducible((S-T)-C,T)=false)
     then if deducible(((S-T)-C) ∪C´,T)=false
           then S´ ← S´−T
          else  S´ ← S´
    else  S´ ← S´
   end
 end
```

なお、実際のRKB の更新にあたっては、回復、その他の諸問題を考える必要がある。

4. おわりに

関係型知識ベース操作として、従来の関係代数に加えて単一化可能性に基づく関係演算を導入することによって、関係型知識ベースにおけるホーン節集合の更新が実現できることを示した。

今後は、これらの操作をさらに拡張するとともに、実現方法について検討を進めていく予定である。

［謝辞］本研究を進めるに当たり、有益な討論をして頂いた研究室、並びにKBM会議メンバの皆様に感謝します。

［参考文献］

[1] Yokota,H.et al: A Model and an Architecture for a Relational Knowledge Base,ICOT Technical Report,TR-144(1985).

[2] Murakami,H.et al: Formal Semantics of a Relational Knowledge Base,ICOT Technical Report,TR-149(1985).

[3] Miyachi,T.et al: A Knowledge Assimilation Method for Logic Databases,New Generation Computing,Springer,pp385-404(1984).

単一化検索言語による知識ベースソフトウエアの記述

1M-6

村上 昌己　　横田 治夫　　伊藤 英則

(財)新世代コンピュータ技術開発機構

1. はじめに　筆者らは先に、知識ベースに大量に格納された知識を効率的に扱う方法として、関係型知識ベースの単一化検索による方法を提案した(1)。しかし関係型知識ベースモデルは、言わば「知識表現のアセンブラ」である。すなわち、高級な知識表現言語が関係型知識ベースモデルの上で記述されなければ、アプリケーション間での知識を関係型知識ベースに格納し活用できない。

本稿では、意味ネットワーク的知識表現ＤＣＫＲ(2)の一部を関係型知識ベースの上に実現し、それらの保守・更新メソッドとして、has _a インヘリタンスの推移性の保持が関係型知識ベースの上で定義する関係演算を用いて記述できることを示す。

2. 関係型知識ベース　関係型知識ベースにおける基本的なデータ構造は関係である。

〔定義1〕変数記号の集合をVar,関数記号の集合をFun とする。ここでFun は有限集合で各元は固有のアリティをもつものとする。またFun とVar は共通部分をもたないものとする。Var の元とFun の元から通常のように定義される項の集合をTermであらわす。Termの元に対する代入の集合をSub であらわす。

t1,t2 ∈Termの単一化(unification) を通常通り定義する。

〔定義2〕t1,t2,…,tn ∈Term。 fをFun に含まれない特別なn属性関数記号とする。このとき、f(t1,t2,…,tn)をn属性項タプルとよぶ。以下ではf を省略して、

(t1,t2, …,tn)

と書く。項タプル同士の単一化は次のように定義される。

〔定義3〕 t=(t1,t2,…,tn) , t'=(t1',t2', …,tn') のとき t と t' が単一化可能であるとは、f(t1,t2,…,tn)とf(t1',t2',…,tn')の二つの項にmost general unifier: σ∈Sub が存在することである。このとき,(σt1, σt2, …, σtn) をｔとｔ' の最汎単一化タプルとよびmgu(t,t') と書く。

〔定義4〕n属性項関係(n-attribute term relation) とはn項タプルの有限集合である。

〔定義5〕n属性項関係Ｔ1，Ｔ2の単一化積(u-intersection)を次のように定義する。

u-intersection(Ｔ1, Ｔ2) = {t ; t1∈Ｔ1, t2∈Ｔ2 が存在し、t=mgu(t1,t2) }

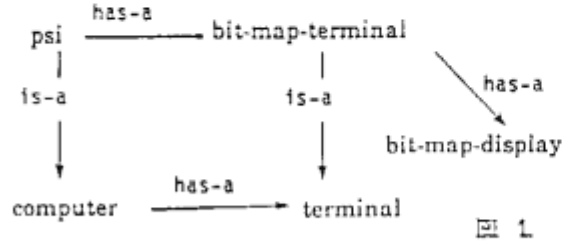


図 1

〔定義6〕n属性項関係Ｔ1，m属性項関係Ｔ2のi番目とj番目(1≦i ≦n,1 ≦j ≦m)の属性による単一化結合(u-join)を次のように定義する。

u-join$_{i,j}$(Ｔ1, Ｔ2) = {(σti, …, σti, …, σtn, σt1',…, σtj-1, σtj-1, …, σtm') | (t1,…,ti,…,tn) ∈Ｔ1,(t1',…,tj', …,tm') ∈Ｔ2が存在し、かつあるσ∈Sub について、σti=σtj' ,}

また項関係に対する射影(projection)，和集合(union) は関係代数演算の場合と同様に定義する。項関係に対する演算を総称して項関係演算とよぶ。

項関係演算は、関係代数演算の拡張として定義されている。関係データベースモデルのタプルでは、対象とするドメインの要素は定数記号によってのみ表現された。したがって関係代数演算ではタプルの各要素に対して施される操作を表現することはできなかった。関係型知識ベースモデルでは、タプルの各要素を表現するのに項を用いることを許し、対象とするドメインの上での操作を表現できるようにしたものである。

3. ＤＣＫＲ　意味ネットワーク的知識表現ＤＣＫＲ(definite clause knowledge representation) (2) は、Prologの上にsem 述語：sem(X, P) を定義することによってis_a 階層及びhas _a 階層を表現している。sem(X, P) は直観的には「Ｘというオブジェクト（の典型的なインスタンス）はＰという性質をもつ」と読む。例として図1 に示した意味ネットワークは図2 のように定義されたsem 述語によって表現される。図2 で第1 行目はis_a インヘリタンスの停止節である。 2〜6 行目は図1 の各枝に対応する。7 行目以下はhas _a 関係のインヘリタンスに推移性をもたせるために用いられる。図3 に「psi は何を持っているか? 」という質問によって検索した例を示す。

ここでは、sem 述語の定義の更新の例として、図1 の意味ネットワークに次の様な枝を付け加える場合について考える。

```
sem(A,is_a(A)).
sem(psi,A) :- sem(computer,A).
sem(computer,has_a(terminal)).
sem(psi,has_a(bit_map_terminal)).
sem(bit_map_terminal,has_a(bit_map_display)).
sem(bit_map_terminal,A) :- sem(terminal,A).
sem(computer,has_a(A)) :- sem(terminal,has_a(A)).
sem(psi,has_a(A)) :- sem(computer,has_a(A)).
sem(psi,has_a(A)) :-
   sem(bit_map_terminal,has_a(A)).
sem(psi,has_a(A)) :-
   sem(bit_map_terminal,is_a(A)).
sem(bit_map_terminal,has_a(A)) :-
   sem(bit_map_display,has_a(A)).
sem(bit_map_terminal,has_a(A)) :-
   sem(bit_map_display,is_a(A)).
```

図 2

terminal —has_a→ input _device

terminal —has_a→ keyboard

keyboard —has_a→ key

これらの枝に対応して次のような節が付け加えられる．

```
sem(keyboard,A) :- sem(input_device,A).
sem(terminal,has_a(keybord)).
sem(keybord,has_a(key)).
```

図　4

さらにhas _a 関係の推移性の保持のために次のような節がassertされなければならない．

```
sem(terminal,has_a(A)) :- sem(keybord,has_a(A)).
sem(terminal,has_a(A)) :- sem(keybord,is_a(A)).
sem(keybord,has_a(A)) :-  sem(key,has_a(A)).
sem(keybord,has_a(A)) :-  sem(key,is_a(A)).
```

図　5

これらの節に対して次のプログラムを実行することによって求められる．

```
comp :-
    clause(sem(A,has_a(B)),true),
    assert((sem(A,has_a(C)):-sem(B,has_a(C)))),
    assert((sem(A,has_a(D)):-sem(B,is_a(D)))),
    fail.
```

4．関係型知識ベースの上でのDCKRの実現　文献(1)で示した関係型知識ベースの上でHorn 節を表現する方法を用いて，DCKRを関係型知識ベースの上で表現し，項関係演算を用いて検索を行うことができる．

ここでは前節でふれたような更新時の has_a 関係の推移性の保持が項関係の演算によって実現されることを示す．更新される前の知識ベースを項関係で表現したものをKBとする．簡単なプログラムによって図2の例から得たKBの例を図6に示す．付け加えられる枝に対応する節を表現する項関係をK1とする．図4の例の場合，K1は図7のようになる．このとき推移性保持の節に対応する項関係K2はK1より次のような項関係演算により求めることができる．

```
yes
| ?- sem(psi,has_a(X)).

X = terminal ;

X = terminal ;

X = bit_map_terminal ;

X = bit_map_display ;

X = bit_map_display ;

X = bit_map_terminal ;

X = terminal ;

no
```

図　3

```
KB={([sem(psi,A)|B],[sem(computer,A)|B]),
    ([sem(bit_map_terminal,A)|B],[sem(terminal,A)|B]),
    ([sem(computer,has_a(A))|B],[sem(terminal,has_a(A))|B]),
    ([sem(computer,has_a(A))|B],[sem(terminal,is_a(A))|B]),
    ([sem(psi,has_a(A))|B],[sem(bit_map_terminal,has_a(A))|B]),
    ([sem(psi,has_a(A))|B],[sem(bit_map_terminal,is_a(A))|B]),
    ([sem(bit_map_terminal,has_a(A))|B],[sem(bit_map_display,has_a(A))|B]),
    ([sem(bit_map_terminal,has_a(A))|B],[sem(bit_map_display,is_a(A))|B]),
    ([sem(A,is_a(A))|B],B),
    ([sem(computer,has_a(terminal))|A],A),
    ([sem(psi,has_a(bit_map_terminal))|A],A),
    ([sem(bit_map_terminal,has_a(bit_map_display))|A],A)}.
```

図　6

```
K1={([sem(keybord,A)|B],[sem(input_device,A)|B]),
    ([sem(terminal,has_a(keybord))|A],A),
    ([sem(keybord,has_a(key))|A],A)}.
```

図　7

```
TEMPLATE={([sem(A,has_a(B))|C],[sem(A,has_a(D))|E],[sem(B,has_a(D))|E]),
          ([sem(A,has_a(B))|C],[sem(A,has_a(D))|E],[sem(E,is_a(D))|E])}.
```

図　8

```
K2={([sem(terminal,has_a(A))|B],[sem(keybord,has_a(A))|B]),
    ([sem(terminal,has_a(A))|B],[sem(keybord,is_a(A))|B]),
    ([sem(keybord,has_a(A))|B],[sem(key,has_a(A))|B]),
    ([sem(keybord,has_a(A))|B],[sem(key,is_a(A))|B])}.
```

図　9

K2 =projection$_{2,3}$
(u-join$_{1,1}$(TEMPLATE, u-intersection
(K1, ((sem(X, has_a(Y))], [ ])))))

ここでTEMPLATEは図8に示す項関係である．

図7の例からK2を求めた例を図9に示す．これは図3に対してcompを実行した結果得られる節に対応する項関係と一致する．更新された結果に対応する項関係KRは次の式で求められる．

KR=union(union(KB, K1), K2)

5．知識コンパイラ　文献(1)では関係型知識ベースの「知識表現のコンパイル・コード」としての可能性を検索操作について検討した．本稿では新たに「知識表現の保守・更新操作のコンパイル・コード」としての可能性について見通しを得た．

関係型知識ベースは効率的な処理アーキテクチャの実現をめざすだけでなく，数学的定式化の可能性も追求しており，真論による定式化が与えられている(3)．ゆえに，高級な知識表現で書かれた知識ベース及びその上の操作を，関係型知識ベースで実現したものは，もとの知識のセマンティクスと考えることができる．我々は高級知識表現言語で記述された知識及びその操作を，項関係及びその上の演算処理に変換する知識コンパイラの開発を計画している．この知識コンパイラは高級知識表現言語と関係型知識ベースの間の意味写像を実現したものとしてとらえることが出来る．

謝辞　有益な助言を頂いたKBM会議及び知識ベースWGのメンバの皆様に感謝します．

参考文献　(1)Yokota,H.,et.al:A Model and an Architecture for Relational Knowledge Base, ICOT TR-144(1985). (2)田中，小山:Definite Clause Knowledge Representation, Logic Programming Conference 85 (1985). (3)Murakami,M.,et.al: Formal Semantics of a Realational Knowledge Base, ICOT TR-149(1985).

KBMS PHI (1)
2M-5 分散知識ベースシステムのシステム構成方式

伊藤 英則, 森田 幸伯, 大場 雅博, 山崎 晴明
(新世代コンピュータ技術開発機構) (沖電気工業)

1. はじめに

第三世代コンピュータシステム研究開発プロジェクトでは、知識情報処理に向いたコンピュータの開発のため、推論マシン・知識ベースマシンの実現を図っている。本研究は、複数の推論マシンと複数の知識ベースマシンがＬＡＮで結合された分散知識ベースシステムを対象とし、その実現に必要な分散制御技術や知識管理方式、協調問題解決方式などの技術開発を行うことが狙いである。

本稿では、分散知識ベースシステムとして試作を計画しているＰＨＩ (Predicate Logic Based Hierachical Knowledge Management System)システムの構成方式について述べる。

2. 分散知識ベースシステムの研究課題

高度な知識情報処理のためには、広域に分散した大量の知識を効率よく管理する必要がある。また、知識は高価な資源であることから、その共有化も重要な要求となる。それらの要求を満たすことを目標として複数台の推論マシンと複数台の知識ベースマシンをネットワークで結合して分散知識ベースシステムを構築することがひとつの大きな研究課題である。

このような環境では、分散化された知識を統合化して、高度に共有化し、大量で高価な知識を高度利用できることが望まれる。

そのためには、以下の条件を考慮する必要がある。

(1)分散している知識の一貫性の保持----
知識の中には相互に関連するものがあり、それらが、ある状態（一貫性）を保持する事を保証する必要がある。すなわち、一方を変更する場合、他方も変更を要求するような制御が必要である。

(2)ロケーショントランスペアレンシー----
分散知識ベースシステムの物理的構成の変更や、共有している知識の格納サイトの変更をおこなっても、他のユーザに影響させたくない。そのためには知識ベースシステムの物理構成を意識する事なく、知識をアクセスできることが必要である。

(3)協調問題解決----
知識は、分散して格納されているので、一つの問題を各サイトで協調しながら解かなければならない場合がある。整合性のとれた解を得るため協調問題解決が必要である。

(4)アクセス権・セキュリティー----
知識の参照・変更は、それぞれ許されたユーザのみに限られ必要がある。

(5)ロバストネス----
あるサイトが障害を発生した際に、分散知識ベースシステム全体が機能を喪失することは望ましくない。障害に対する強靭性が必要である。

以上の他にも、分散知識ベースシステム特有の問題も多く存在するであろう。それらを明確にし、ユーザにとって応答性の良い、信頼性の高い、分散制御方式の体系を開発することが本研究の狙いである。

3. ＰＨＩシステムの全体構成

ＰＨＩシステムの全体構成は、図１に示す通り本夫の推論機能を果すＰＳＩ（ホストＰＳＩ）および知識ベースマシンの機能を果すＰＳＩ（ＰＨＩ）とがＩＣＯＴ−ＬＡＮにより結合された構成とする。

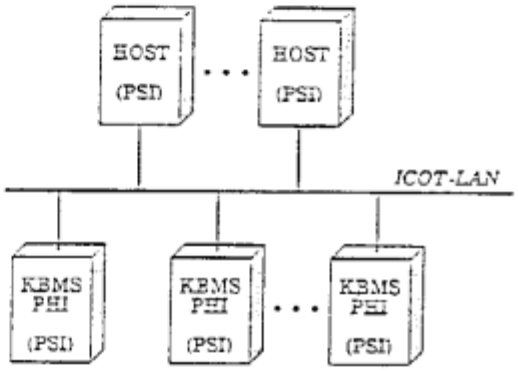


図1. PHIシステムの全体構成

我々は、前期３年間で、上記の環境を目標に、その第一歩として関係型データベースマシンＤｅｌｔａを開発し、知識のうちファクトをデータベースに格納し、それと推論マシンを接続するいくつかの方式を検討した[1]。しかしながら、推論マシンと関係代数コマンドで接続するインタフェースをとると両者の間で非常に多くのインタラクションを必要とし効率的でない。ＰＨＩでは、推論マシンとのインタフェースをより高級なものにする。推論マシンとのインタフェースとしては、推論マシンが論理型言語を用いていることを考慮して、ホーン節（図２）に設定する。

また、格納する知識の範囲も拡張し、扱う知識の対象をホーン節とする。知識ベースマシン側でもホーン節を処理することから、知識ベースマシンも推論機能をもつ必要がある。従って、ＰＨＩシステムにおいては、知識ベースマシンを推論マシンＰＳＩ上に実現する。

また、ネットワークに関しては、ＩＣＯＴ−ＬＡＮを用いる。ＩＣＯＴ−ＬＡＮは、その特徴として、任意のメンバからなるグループを形成しそのグループ内で同報通信を行うグループ通信機能を持つ。

ＰＨＩにおいては、知識ベース管理等の機能は、ＰＳＩ上にソフトウェアで実現するが、知識ベース演算の高速化を図るために一部の機能をハードウェア化する。この専用ハードウェアを、知識ベース演算エンジンと呼ぶ（図3）。前期の関係代数エンジンも知識ベース演算エンジンのひとつのタイプと考えることができる。

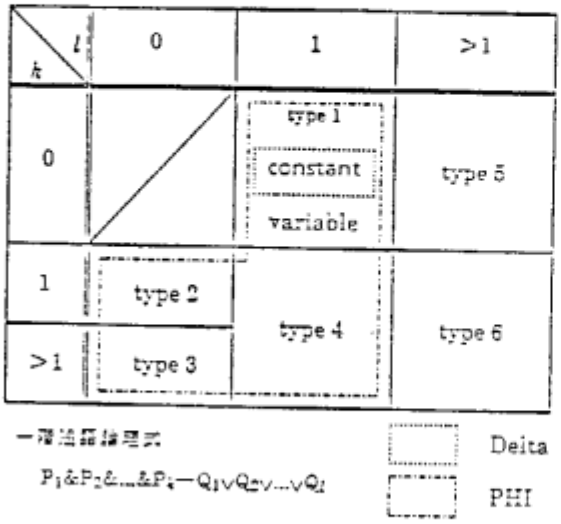


図2. 対象とする知識

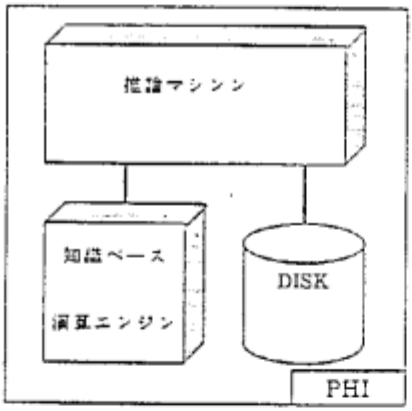


図3. PHIのハードウエア構成

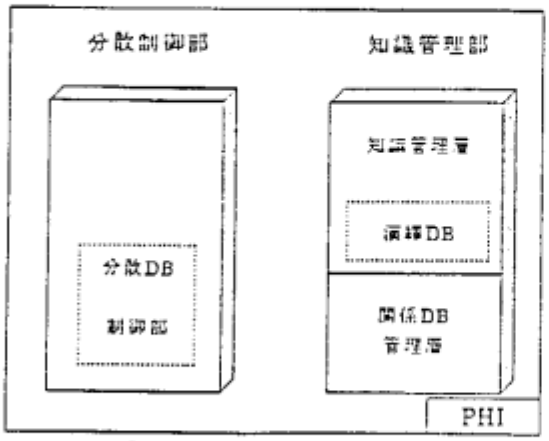


図4. PHIのソフトウエア構成

4. ＰＨＩの機能構成

ＰＨＩの機能構成は、知識ベース管理機能を実現する知識ベース管理部と、各サイトの知識ベース管理部を統合し、ユーザに対してひとつの知識ベースシステムとして見せる分散知識ベース制御部からなる（図4）。

ＰＨＩでは、ファクトとルールを分離して取り扱う。ファクトに対しては関係データモデルを用いる。つまり、知識ベース管理部は、知識管理層と関係データベース管理層の2階層で構成する。ルールを取り扱う知識管理の機能については、さまざまなアプローチでいくつかの研究がなされているが、まだ、全容は見えていない。ＰＨＩでは、知識管理部の核として演繹データベース管理層を置き、それを基礎に知識管理方式の拡張を図っている。

分散制御部に関しては、知識ベース管理部の構成に対応して、分散データベース制御部を置き、それを核にして分散制御部の実現をすすめてゆく。

なお、知識ベース管理部の関係データベース管理層と分散データベース制御部をあわせて分散（関係）データベースを構成できる。

5. まとめ

分散知識ベースシステムＰＨＩでは、現在関係データベース管理層、分散データベース制御部、および知識ベース管理層内の演繹データベース管理層を中心にして実現に向けた検討を行っている[3]-[5]。今後は、知識管理層および分散制御部についてさらに検討を進め、ＰＨＩの試作・評価をとおして分散知識ベースシステムを実現する。

［謝辞］本検討を進めるに当たり、有益なご示唆を頂いた知識ベースマシンＷＧのメンバの方々に感謝いたします。

［参考文献］

［1］Yokota,H. et.al. :"A Model and an Architecture for a Relational Knowledge Base",submitted to the 13th Int. Simpo. Computer Architecture,1986.

［2］Yokota,H. et.al. :"Deductive Database System based on Semantic Resolution.", to appear in Procedding of the 2nd In. Conf. Data Engineer.,1986.

［3］安原，他："知識ベースマシンDelta2"，昭和60年度信学全大

［4］富塚，他："KBMS PHI(2) 知識とデータの扱いに関する一考察"，第32回情処全大，2M-6,1986.

［5］吉田，他："KBMS PHI(3) 分散知識ベース制御方式"，第32回情処全大，2M-7,1986.

［6］山中，他："KBMS PHI(4) 知識ベースマシンシミュレータの機能概要"，第32回情処全大，2M-8,1986.

KBMS PHI (2)

2M－6　知識とデータの扱いに関する一考察

宮崎　収兄*　　横田　治夫**　　阿比留　幸民*
*沖電気工業株式会社　**財団法人　新世代コンピュータ技術開発機構

1．はじめに

第五世代コンピュータ・プロジェクトにおける知識ベース機構の研究では前期3年間は関係データモデルを基礎にした研究を行った。中期4年間での目標は知識ベースマシンプロトタイプの開発であり、知識ベースの扱いに関しては以下の2つのアプローチによる研究を行っている。

(1) Term (項) を単位とする関係型知識モデルによる知識とデータの統一的処理 [1]。
(2) 関係データベース上に知識管理層を追加し2つの階層をあわせ知識ベース管理機能を実現。

この2つのアプローチは両者とも前期のアプローチの発展形態である。KBMS PHIでは後者を中心とした検討を行っているので知識ベース管理部の基本的考え方と構想を述べる。

表1．論理型言語処理系と関係データベース管理システムの比較

機能		論理型言語処理系	RDBMS
知識表現	データ	ファクト	リレーション(タプル)
	知識	ルール	(ビュー)
	再帰問合せ	○	×
	否定	negation as failure	意味不
	構造体	○	×
	変数	○	×
	問合せの表現	ゴール	問合せ言語
	知識の構造化	△	×
管理機能	格納/変更	プログラム言語アサート/リトラクト	更新言語
	共有時のコンカレンシィ	×	○
	リカバリ	×	○
	セキュリティ	×	○
	インテグリティ	×	△
処理方式	基本演算	ユニフィケーション	値の比較
	実行方式	演繹推論により1つずつ求める(トップダウン)	関係演算による全解探索(ボトムアップ)
	実行制御	カット等で制御可能	システムが行う
	実行最適化	×	○
	大規模化(2次記憶使用)	×	○
その他	マルチメディアデータ	×	△

2．論理型言語処理と関係データベース

使用する側からは、知識表現は問題に適したものを使えることが望ましい。他方システム作成者の側からは、できるだけ共通の処理方式によって実現できることが望ましい。また知識ベースの大規模化や共有化を図る必要がある。これらの要求を満たす知識ベース管理システムの構成を検討するため、理論的基礎が明確な述語論理型言語処理を基礎に考える。

知識ベース管理システムに要求される機能には知識の表現能力や処理効率化、共有制御などの知識管理などがある。述語論理を基礎とするシステムでこれら要求の一部を満たすものにPrologに代表される論理型言語の処理系と関係データベース管理システム(RDBMS)がある。知識ベース管理システムに対する要求条件についてこの2つのシステムを比較したものを表1に示す。ここで表現能力の比較はホーン節と関係データモデルの自然な対応関係、即ちホーン節のファクトと関係データモデルのタプルの1対1対応を基礎とした。

表1から明らかなように、表現能力の面では論理型言語処理系が、管理機能の面ではRDBMSが優れている。処理方式による効率についてはそれぞれ利害得失があるが知識ベースが大規模な場合はRDBMSが優れている。これは現在Prolog等では処理の順序が定まっているのに対し、RDBMSでは柔軟な最適化ができることや、2次記憶に対する効率的アクセスメソッドが使用できることによる。また並列処理による高速化を行うシステムでも処理対象の範囲を限定して最適化を図れるRDBMSの方が全体の計算量を減らせる可能性が大きい。

3．知識ベース管理システムの構成

前節の考察から知識ベース管理システムの構成として論理型言語によるプログラムとRDBMSの利点を生かし、両者を結合したものを近似モデルとして考えることができる(図1)。このモデルを実現するシステムはICOTなどで試作されたが[2]、以下のような問題が残されている。

(1) ユーザプログラムと知識ベース管理システムの関係の明確化。
(2) 知識の構造化や他の知識表現との関係。
(3) 知識の管理、特に知識の格納方法や知識のインテグリティ管理。
(4) 実行効率の改良。最適化処理や制御構造に関する実行機構のメタ知識による制御。

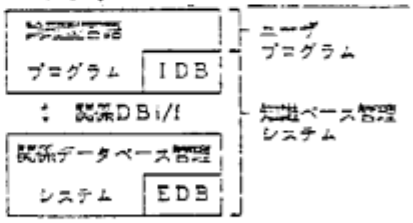


図1　論理言語＋関係データベース管理システムによる知識ベース管理システムモデル

これらの問題を解決するには知識ベース管理システムでデータ以外の知識も管理することが必要である。KBMS PHIでは知識ベース管理システムをデータベース管理層と知識管理層の2階層から構成する（図2）。

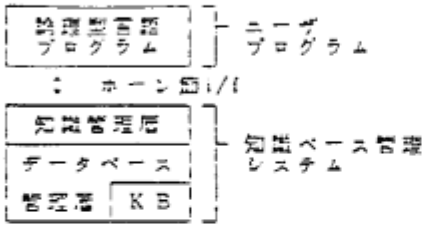


図 2 KBMS PHIのモデル

知識ベースはインテンションデータ部（IDB）、エクステンションデータ部（EDB）およびこれらを管理するメタ知識よりなりデータベース管理層に格納される。IDBはホーン節のルールに、またEDBはホーン節のファクトに対応するものをそれぞれリレーションに格納したものである。

データベース管理層は知識ベースの検索や管理を関係データモデルに従って処理する。データベース管理層は関係演算を基本とするためIDBの内容についても単純なデータとして扱い、ホーン節ルールとしての意味をもった処理は行わない。データベース管理層での処理はホーン節処理のうち以下の制限をつけたものと同等である。

(a) （構造体を含まない）単ホーン節の述語および not、比較、算術演算のみを含む。
(b) 再帰定義を含まない。
(c) 任意のゴールに対し解は、"yes/no"、またはファクト中に現われる値の組み合わせの集合のみである。

知識管理層は知識ベース管理システムに対する要求を処理するのに必要な知識をIDBより取りだし推論を用いた処理を行う。この処理に必要なデータがEDBにある場合は関係演算問合せによりデータをデータベース管理層に要求する。知識管理層の機能には以下のものがある。

(a) 演繹データベース機能（再帰的定義を含むホーン節問合せ（＋IDB）の関係演算への変換およびデータベース管理層による検索実行の制御）。
(b) 構造体を含むファクトとEDBのリレーションとのフォーマット変換。
(c) 検索述語の制限や制御条件付検索の処理および最適化。
(d) IDBのリレーションへの格納と管理、IDBの検索、及び検索に用いたIDB内容の出力。
(e) 知識ベース管理に必要なシステム述語の処理

例えば知識リレーションancestorの処理にPHIでは概念的に以下のようになる。

[プログラムと知識ベース]

```
プログラム: ・・・, get(ancestor(X, Y)), ・・・
IDB   :  ancestor(X, Y) :-parent(X, Y).
         ancestor(X, Y) :-parent(X, Z),
                          ancestor(Z, Y).
         parent(X, Y) :-edb(parent(X, Y)).
EDB   :  parentリレーション
```

[処理]

① プログラム実行時の問合せ発行、
"?-get(ancestor(taro, Y))"。
② 知識管理層による関係演算生成
$ancestor_{\$2}$ =
$\pi_{\$2}((\sigma_{\$1=taro}(parent)) \cup$
$\pi_{\$1,\$4}(\sigma_{\$1=taro}(parent \times_{\$2=\$1} ancestor)))$
③ 知識管理層の繰り返し制御下でのデータベース管理層による解の生成
④ 知識管理層での後処理後、解のプログラムへの受渡し

4．PHIの特徴

KBMS PHIでは知識ベース管理システムを階層化し、知識ベースをIDBとEDBに分離して管理する。この結果論理型言語処理系のもつ知識表現機能とRDBMSの持つ管理機能の利点をあわせたシステムを構成可能である。さらに処理効率の面でも知識の性質に即した処理方式が適用できる。PHIでは知識ベース処理の階層化により以下の特徴の実現を図っている。

(1) 知識ベースのプログラムからの分離による知識の共有化・大規模化の実現。
(2) 論理型知識表現を基礎とし、単ホーン節に加え構造体処理や検索制御機能を実現する。
(3) IDBに対するトップダウン 理とEDBに対するボトムアップ処理の組み合わせによる処理効率化。
(4) IDBとEDBの論理的分割による分散知識ベース処理の容易化。
(5) ホーン節インタフェースによるPSI（ホスト）プログラミング言語との統一、およびユーザプログラムと知識ベース管理システムのインタラクション減少。（知識ベースマシンや分散知識ベース管理システムではインタラクション減少による効率化が必須条件である。）
(6) データベース管理層によるIDBしぼり込みを用いた知識管理層の検索空間の制限。
(7) IDBとEDBをあわせた知識ベースのインテグリティ管理の実現。

5．まとめ

KBMS PHIにおける知識ベースの扱いとシステム構成の基本的考え方を述べた。現在データベース管理層および知識管理層の核部分（演繹データベース＋構造体＋検索制御）の実現方式の検討を行っている。今後はこれらの詳細化および知識ベースの構造化やIDBの分散化などの検討を行う予定である。

[参考文献]

[1] 柳澤他："知識ベースマシンDelta2"，昭和60年度信学会情報システム部門全国大会，昭和60年11月
[2] H. Yokota, et. al.: "An Enhanced Inference Mechanism for Generating Relational Algebra Queries", Proc. 3rd ACM Symp. on Principles of Database Systems, Apr. 1984

KBMS PHI (3)
2M-7 分散知識ベース制御方式

吉田 勇* 高杉 晋朗* 大場 稔博**

*沖電気工業株式会社 **財団法人 新世代コンピュータ技術開発機構

1．はじめに 知識情報処理に不可欠な技術として大量な知識の効率的管理及びその共有化がある。これらの技術課題の解決の基礎技術としてＩＣＯＴでは、複数知識ベースの統合化技術の確立を目指しＰＨＩの試作を行っている[1]。現在我々はＰＨＩの知識ベースのうちデータ部分の分散化のための方式検討を行っている。その具体的課題には、ロケーション・トランスペアレンシィ、サイトの自律性、分散問合せ処理、分散サイト間のコンカレンシィ制御などがある。本稿では上記項目を中心にＰＨＩの分散知識ベース制御方式について報告する。

2．ＰＨＩの構成 ＰＨＩは物理的には複数台のサイト（逐次型推論マシンＰＳＩ）とこれらを結合するブロードカスト機能及びグループ内の通信グループを形成する機能を有する高速ネットワーク（ＬＡＮ）とからなる[2]。また論理的には上記サイト内のトランザクション・マネジャ（ＴＭ）とデータ・マネジャ（ＤＭ）からなる（図１）。ＴＭとＤＭの両方が１つのサイトにあっても良い。ＰＨＩではサイト間の通信目的に合せ以下の３つの通信グループを形成する。(1)システム通信グループ，(2)トランザクション通信グループ及び(3)ＤＭ間セッションである（表１）。これら３つの通信グループを適宜使用しトランザクション処理を行う。１つのトランザクションはユーザ端末が接続されているサイト（ユーザサイト）内のＴＭとそのトランザクション内で使用されるリレーションを含むサイト内のＤＭとで構成されるトランザクション通信グループ内で処理される。

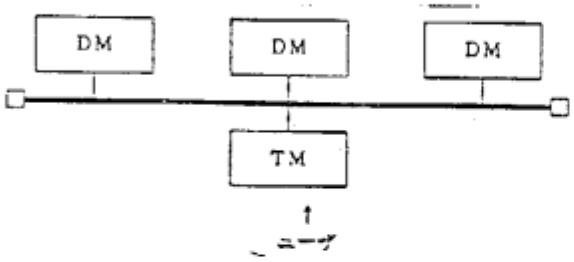


図１ ＰＨＩの論理構成

表１ 通信グループ

名称	メンバ	形成時期
システム通信グループ	システム内の全サイト	立上げ時
トランザクション通信グループ	当該トランザクションを受付ける１つのＴＭと関連するＤＭ	トランザクション発生時
ＤＭ間セッション	１対のＤＭ	立上げ時

3．ＰＨＩの分散知識ベース制御方式

3.1 ディレクトリ管理 ＰＨＩでは知識ベースのデータ部分はリレーショナルモデルによって扱う。またデータの分散配置はリレーション単位とし、重複しての配置は行わない。ロケーション・トランスペアレンシィを実現するため、各々のリレーションにはユーザ毎に定義できる、ユーザ定義名と（格納サイト識別子ーサイト内管理識別子とからなる）システム内一意のシステム内ＩＤとの対応表（ディレクトリ）が付与される。このためユーザ毎に各自のリレーション名が定義できる（図２）。図２において３つのリレーションR_1, R_2, R_3がサイトS_1に格納されており、ユーザ１とユーザ２に同じリレーションに対して異なったユーザ定義名を付与している。このディレクトリは各サイトが当々に自身のリレーションについてのみ管理する、非重複分散管理であるとする。この方式では、ＬＡＮのブロードカスト機能を活用することにより管理オーバヘッドが少なく、且つ高度なサイト自律性を提供することが可能となる。

ユーザ１

ユーザ定義名	システム内ＩＤ
従業員	$S_1 \cdot R_1$
部門	$S_1 \cdot R_2$
給料	$S_1 \cdot R_3$

ユーザ２

ユーザ定義名	システム内ＩＤ
EMP.	$S_1 \cdot R_1$
DIST.	$S_1 \cdot R_2$
SALA.	$S_1 \cdot R_3$

図２ ディレクトリ例

3.2 トランザクション管理 通常、トランザクション実行時にはシステム内全サイト中、実行すべきトランザクションに関連する（該当するリレーションを保持する）サイトはユーザサイトを含み数つかのサイトに限定される。これら関連するサイトで構成されるグループで１つのトランザクション処理を行うことにより他サイトに対するブロードカスト通信時の余分な負荷を軽減することが可能である。ＰＨＩではこのグループをトランザクション通信グループとしてトランザクション開始時（スタート・トランザクション時）にユーザサイト内ＴＭの指

示により形成する。このトランザクション通信グループは問合せ処理が終了するまで保持され、終了と同時にユーザサイト内ＴＭの指示により消滅する。

トランザクションの管理はスタート・トランザクション、エンド・トランザクション及びアボート・トランザクションにより規定される。

3.3 分散問合せ処理　ユーザからの問合せはＴＭからトランザクション通信グループに対してコマンド木群としてブロードカストされる。この問合せは単一サイト内で処理可能なものと、複数サイトにまたがる処理（分散問合せ処理）とに分類される。前者についてはサイト内で閉じた処理であるため一意の処理であるが、後者についてはサイト間のリレーションの転送等を必要とするため、分散データベースにおける問合せ処理の性能向上の為にはこの分散問合せ処理の最適化が不可欠である。従来この最適化についてはセミジョイン方式などが用いられているが[3]、ＰＨＩでは事前にセミジョイン結果等の情報を管理していないため従来の方法では問合せを効率よく処理できない。そこで事前に得たリレーション情報をもとに問合せ処理時に最適な処理手順を決定する、以下の２つのダイナミックなアルゴリズムを検討した[4]。

(1) 個々のサイトが各分散問合せ中のコマンド木毎にその対象のリレーションのサイズを比較することによりリレーションの転送方向を決定する。

(2) 分散問合せを複数のステージからなるコマンドグラフに展開し、各ステージにおいてサイト間で最適な転送プランを作成する。この方式(2)はさらに細かく分類でき、a)は各ステージ内全体に対する最適なリレーション転送プランを各サイトが個々に作成するもので、b)は自サイトに関するコマンド木のみの最適な転送プランを作成しその後ステージ内で各転送プランが矛盾しないことを確認するものである。

この(1)、(2)のいずれかで作成したリレーション転送プランを各サイトにおいて実行する。

3.4 コンカレンシィ制御　従来よりコンカレンシィ制御には、２フェーズロック方式とタイムスタンプ方式とがあるが[5]、ＰＨＩにおいてはこれらの内ブロードカストＬＡＮの機能を活用するため２フェーズロック方式、特に事前宣言を用いた方式を採用する。

デッドロックの対処としてはDetection法、Prevention法、Avoidance法があるが、ＰＨＩにおいてはブロードカスト機能を主に用いるのでメッセージの到着順序の逆転が稀である為これらの中でDetection 法を採用する。Detection 法は更に細かく集中型、分散型、階層型に分類できるが、単一型ＬＡＮであり分散システムである点を考慮すると分散型がよく適合する。ＰＨＩにおけるコンカレンシィ制御は具体的には、スタート・トランザクション時にユーザサイト内ＴＭはトランザクション内で使用するリレーション名を付加してシステム内全体にブロードカストする。システム内全サイトはこのメッセージの内容を解析して、自サイト内に該当するリレーションがある場合該当するリレーションに対してロックを掛けその後、このリレーションに関する情報（ディレクトリ、そのリレーションのサイズ等）をユーザサイト内ＴＭのみならず、システム内全サイトにブロードカストする。ユーザサイト内ＴＭはこのメッセージの内容をもとに必要とするサイトのみでトランザクション通信グループを形成する。このサイトがブロードカストするリレーション情報により各サイトは他サイト内でのトランザクションのリレーション使用状況及びロック待ち情報を知ることができる。それによって各サイトがグローバルなトランザクション待ち状態グラフ（ＴＷＦＧ）を作成・管理し、グローバルなデッドロックを検出することができる。デッドロックを検出した場合、ひとつのトランザクションをアボートすることによりデッドロックを解消する。問合せ終了時（エンド・トランザクションもしくはアボート・トランザクション時）にはユーザサイト内ＴＭは、トランザクション終了もしくは中止の旨をシステム全サイトにブロードカストし、その後この通信グループの消滅を行う。

4．まとめ　ＰＨＩにおける分散知識ベース制御方式についてデータ分散化を中心に報告した。本方式によればブロードカスト通信機能を有効に利用することにより効率的な問合せ処理、コンカレンシィ制御等が可能となる。現在は機能設計段階であり、今後はさらにこれらの事項について詳細化するとともに、知識全体の分散化の制御方式についても検討を加えてゆく予定である。

［参考文献］

[1]伊藤 他：“ＫＢＭＳ　ＰＨＩ(1) 分散知識ベースシステムの構成方式”，第３２回情処全大，１９８６年３月

[2] A. Taguchi et al.: "INI: Internal Network in the ICOT Programming Laboratory and Its Future". ICCC. Sydney. Australia. Oct. 1984

[3] B. Williams. et al.: "Introduction to a System for Distributed Databases (SDD-1)". ACM TODS. Vol. 5 No. 1 pp. 1-17. 1980

[4]吉田 他：“ＫＢＭＳ　ＰＨＩにおける分散問合せ処理方式”，昭和６１年度信学全大，１９８６年３月

[5] C. Mohan.: "Recent and future trends in distributed data management". Proc. NYU Symposium on New Direction for Data Base Systems. May 1984

KBMS PHI(4) 知識ベースマシン シミュレータの機能概要

2M-8

山中 幸隆* 山下 祥司* 森田 幸伯**

*沖電気工業株式会社 **財団法人 新世代コンピュータ技術開発機構

1. はじめに

第三世代コンピュータ・プロジェクトでは既に前期過程を終り、その成果として逐次型推論マシンPSIと超大規模データベースマシンDeltaが開発されている[1][2]。後期4年間においては知識ベースマシンのプロトタイプの開発がひとつの目標となっているが、筆者等はこの目標に沿いKBMS PHIの開発を進めている[3]。そして、現在知識ベース管理部の基本となるデータベース管理層として、Deltaの経験に準じた関係データベース管理システム(RDBMS:PHIでは知識ベースマシンシミュレータと呼んでいる)の開発を行っている。本稿ではこのRDBMSの機能概要について述べる。

2. PHIと関係データベース

PHIでは、

・プログラムからの知識の独立
・知識の共有
・大量知識の取り扱い

等を図っていて、その知識ベース管理部の構成を、

(1) 知識管理層(証明部)
(2) データベース管理層(基本部)

の2層に分けている。知識管理層では論理型言語のルールに対応するインテンショナルな知識の管理を行う。データベース管理層では知識の大量化を効率的に処理する為、ファクトに対応するデータを管理すると共に、インテンショナルな知識の二次記憶への格納も行う。尚、データベース管理層は単体でも動く事ができる。以下、論点を絞る為に、データベース管理層が単体で動く場合を想定して話を進める。

3. システム構成

想定しているシステム構成を図1に示す。図1で、ホスト・マシン、RDBMSにはいずれも逐次型推論マシンPSIを想定していて、それぞれは高速LANを介して接続されている。

4. コマンド・インタフェース

RDBMSとホスト・マシン上のホストとの間の論理的なインタフェースは、基本的には、RDBMSがホスト側からいくつかのコマンドを受け、RDBMS内で処理を行い、そのレスポンスをホスト側に返す事により構成される。

RDBMSの処理においては、次の様に処理単位を階層化している。

(1) コマンド
関係代数を中心にしたRDBMSの処理を指示する基本単位。

(2) コマンド木
1つの意味のある結果を得る為のコマンドの集り。

(3) コマンド木群
コマンド木の集り。

(4) トランザクション
データベースの一貫性を保つ為の単位。1つもしくは複数のコマンド木群よりなる。トランザクション終了と共にロックも解除される。

5. ESP

RDBMSの記述に、Prologにオブジェクト指向の概念を加えたESP(Extended Self-contained Prolog)言語により行う。ESPはICOT(新世代コンピュータ技術開発機構)により開発された言語であり、RDBMSのインプリメントを行うPSIのシステム記述言語となっている為である。ESPを採用する事により、上位レイヤの知識管理層とのインタフェースが明快になり、上位レイヤの変更に関して大きな柔軟性を持つ利点が生まれる。又、Prolog特有のユニフィケーション、バックトラック等の機構により、システム開発も容易になる。

6. リレーションとデータタイプ

RDBMSでは関係代数的処理を提供する為に、ユーザが扱うリレーションを以下の3種類に分けている。

(1) PR(Permanent Relation)
ユーザにより正規に定義され、トランザクション終了後も残る。

(2) IR(Intermediate Relation)
ユーザの指定は名前のみ行われる。コマンド木に1つ対応し、トランザクション終了時に消去される。

(3) ＴＲ(Temporary Relation)

コマンド木の中だけで定義され、名前の指定はない。コマンド木中の他コマンドからそのＴＲを生成したコマンド番号によって指示される。

又、ＲＤＢＭＳでは、データタイプとして文字型、数値型、無型を扱う。文字型には２バイト漢字ＪＩＳと１バイトＡＳＣＩＩがあり、数値型には４バイト及び８バイト整数と浮動小数点実数がある。無型は演算対象として何にでもマッチする。

7．機能

以下ＲＤＢＭＳの機能概要を述べる。又、一部のコマンド例を表１に記す。

7.1 データ操作機能

(1) 検索

インデクスとしてハッシュ、B*木が利用できる。コマンド木は内部で自動的に最適化される。

(2) 更新

更新情報をリレーションとしてまとめ、リレーション単位で更新を行う。これにより、タプル検索の処理効率を高める事ができる。

(3) 入出力

ホストとＲＤＢＭＳとのデータ転送を行う。一回当り最大4000バイトのデータ転送が可能である。

(4) その他の機能

ソート、コピー等を用意している。

7.2 データ定義機能

リレーションやインデクスの登録、削除等を行う。

7.3 非同期処理

ＲＤＢＭＳの異常状態に対処するもので、他の処理の実行状態にかかわらず優先して非同期な処理を行う。システムの稼働状態検査やトランザクション強制アボート等がある。

7.4 セキュリティ

セキュリティ機能として、ユーザのデータベース使用権をユーザ名とパスワードによりチェックする。パスワードにユーザが任意に変更できる。又、リレーション使用権をリレーション特有のアクセス許可モードによりチェックする。

ユーザはユーザグループとしても登録できるので、幅広い利用が可能となる。又、アクセス権付与コマンドにより、特定のユーザだけにアクセスを許す事ができる。

7.5 コンカレンシィ制御

データの一貫性を保証する為に、ロック機構によるコンカレンシィ制御を行う。ロックの単位はリレーションであり、範囲はリレーション群である。但し、スキーマに関しては階層管理を行ない、ロック単位をタプル群とする。これによりリレーションが違えば検索更新処理と定義処理の競合を許す事ができる。

7.6 リカバリ

データベース利用に関して、トランザクション障害、メディア障害等の障害からデータを守る為にシャドー方式に基づく二重化管理を行い、トランザクション開始時の状態へ回復させる。

7.7 バッファリング

メモリ内での処理に比べて二次記憶との１／０処理は負荷が大きい。そこで、二次記憶との１／０を少なくする為にＬＲＵ機構に基づくバッファリングを行う。

8．おわりに

ＲＤＢＭＳの機能概要について述べてきた。今後は、より詳細な検討を進め、より高機能なシステムの構築を目指す予定である。

［参考文献］

[1] 角田　他："RDBM "Delta"(1)～(111)"，第26回情処全大，1983

[2] 西川　他："パーソナル逐次型推論マシンφ"，第27回情処全大，1983

[3] 伊藤　他："KBMS PHI (1)～(3)"，第32回情処全大予稿集，1986

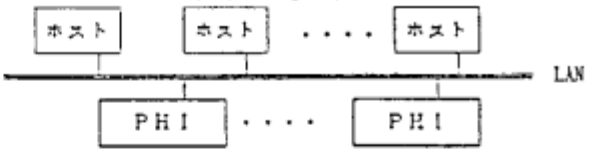


図１．システム構成

機能	コマンド
検索	PROJECTION SELECTION JOIN UNION COUNT SUMMATION ADDITION EQUAL
更新	INSERT DELETE UPDATE
入出力	GET GET-NEXT GET-END PUT PUT-NEXT
特殊操作	GROUP-BY SELECT-GROUP COPY SORT
データ定義	CREATE-RELATION PURGE-RELATION
アクセス権制御	REGISTER-USER AUTHORIZATION
トランザクション制御	START-TRANSACTION END-TRANSACTION
セッション制御	BEGIN-SESSION END-SESSION
非同期	SENSE-STATUS ABORT-PROCESSING

表１．コマンド例

5G-5

マルチポートページメモリを用いた大規模知識ベースの制御方式

物井 秀俊、横田 治夫、西田 健次、伊藤 英則　　酒井 浩
(財)新世代コンピュータ技術開発機構　　(株)東芝

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では、知識情報処理向けのコンピュータシステム開発のため、推論・知識ベース機構の実現を図っている。我々は、その基本要素の一部として知識ベースマシンを開発中である。

知識ベースマシンは、大量の知識をディスク等の二次記憶装置に格納し、ホストから要求された知識の検索を高速に実行する専用マシンとして位置付けられる。このようなシステムでは、処理装置と主記憶間のボトルネックだけでなく、主記憶と二次記憶間のボトルネックが性能上の大きな問題となる。

我々は、この主記憶と二次記憶間のボトルネックを解消するため、複数の二次記憶装置およびデータ処理装置から同時にアクセス可能なマルチポートページメモリ(MPPM:Multi-Port Page-Memory)の、経済的かつ実用的な構成法を提案した[1]。さらに、大量のデータを扱うときに、ネックとなる処理に専用ハードウェア(エンジン)を用いる方式は、データベースマシンの一つの流れとなっている[2]。

本稿では、このMPPMと専用ハードウェアを用いた知識ベースマシンの制御方式について述べる。

2. 関係型知識ベース検索

我々が検討中の知識ベースマシンでは、項関係を構成要素とする関係型知識ベースを検索の対象とする。項関係とは、関係データベースにおける関係操作の対象を項、つまり変数を含む一種の構造体に拡張したものである[3]。

関係型知識ベースに対する演算としては、項間の単一化を基本操作とする単一化結合と単一化制約および射影がある。これは、従来の関係代数の結合や制約演算の等号条件を単一化に置き換えた操作である。

3. アーキテクチャ

複数のプロセッサをバッファメモリを介して二次記憶と結合する場合、一本のデータバスで結合したのでは、データバス上での競合が頻発し、思うような性能を得ることができない。このようなデータ転送時の競合を押えて高速なデータ処理を可能にするため、バッファメモリとしてMPPMを使用する。

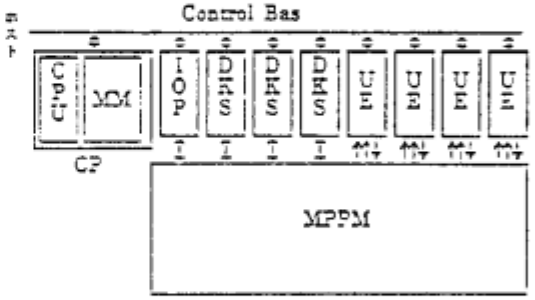
検討中の知識ベースマシンの全体構成を図3.1に示す[4]。各要素は、以下のような動作をする。

・UE－項関係演算を実行するデータ処理専用ハードウェア[5]。MPPMからデータをストリームとして入力し、処理結果をストリームとしてMPPMに出力する。

・DKS－二次記憶とMPPM間のデータ転送を実行。

・IOP－ホストマシンまたはCPのMMとの間のデータ転送を実行。

・CP－システム全体の制御とホストマシン間のインタフェース制御を実行。



CP　:Control Processor
MM　:Main Memory of CP
IOP　:I/O Processor
UE　:Unification Engine
DKS　:Disk System
MPPM　:Multi-Port Page-Memory

図3.1 知識ベースマシンの全体構成

ここで、UEとDKSは独立にMPPMのページがアクセス可能なため、UEを実行させる一方でDKSによるデータの先読みを行うことができ、制御方式によっては効率的なデータ処理が可能になる。

知識ベースマシン全体の制御とホストとの間のインタフェース制御は、CPで行う。UE,DKSおよびIOPの各プロセッサは、CPに対する入出力装置として動作し、コマンドと割り込みにより制御を受ける。ここで、CPと各プロセッサ間の制御情報のやりとりは、コントロールバスを通して行われる。

4. 並列制御方式

機能分散型のマルチプロセッサデータベースマシンでは、各プロセッサのスケジューリングが、性能上の問題となる。このスケジューリング方式としては、例えばDIRECT[2]に見られるように、各プロセッサにスケジューリング機能を持たせる方式が代表的である。しかし、この方法ではトランザクションが増えた時、プロセッサの割当てオーバヘッドが増加する恐れがある。

我々は、MPPMに接続された各プロセッサを入出力装置として動作させることにより、CPにおいて事象駆動型のスケジューリングを行い、効率のよい並列処理を行うことを考えている。

先ず$T_2 \leftarrow T_{0body} \otimes_{head} T_0$(注)という単一化結合演算を考える。ここで、入力となる項関係T,T_0が

$T = \{PT_1, PT_2, PT_3, PT_4\}$

$T_0 = \{PT_{0_1}, PT_{0_2}, PT_{0_3}\}$。

というページに分割でき、その内PT_2,PT_{0_1},PT_{0_3}がMPPM上にステージング済みであるとする。この時、単一化結合はページ毎の演算をUEで行い、図4.1に示すような手順で実行できる。

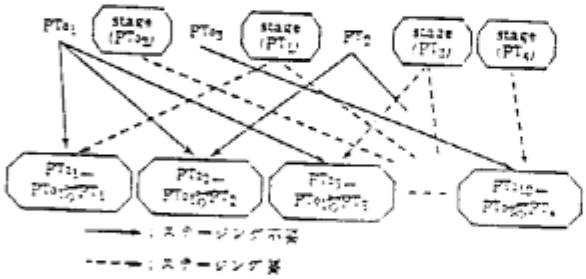


図4.1 単一化結合の実行手順

図4.1の各ノードにおいて、PT_2,PT_{0_1},PT_{0_3}だけを使う単一化結合は、UEで即実行可能である。しかし、他のノードではDKSにおけるステージングの完了を待ち合せる必要がある。このような待ち合せを可能にするため、以下のパケット形式の命令を考える。

条件	演算	後処理

ここで各部は、以下のような目的をもつ。

・条件部 ― パケットが実行可能となる条件。

・演算部 ― UEで実行する演算と使用するページ。

・後処理 ― 演算結果を他のパケットに反映。

このパケットを用いると図4.1の処理は図4.2のように書くことができる。ここで、S(P)はページPの状態を示し、S(P)=Tの時ページPがアクセス可能S(P)=Fの時アクセス不可能を示し、初期値はFとする。また、stage(P)は,ページPのDKSによるステージングを示す。

プロセッサ	条件	演算	後処理
DKS	T	; stage(PT_1)	;S(PT_1)←T
	T	; stage(PT_3)	;S(PT_3)←T
	T	; stage(PT_4)	;S(PT_4)←T
	T	; stage(PT_{0_3})	;S(PT_{0_3})←T
UE	$S(PT_1)=T \& S(PT_{0_1})=T$	;$PT_{2_1} \leftarrow PT_2 \otimes PT_{0_1}$	;S(PT_{2_1})←T
	$S(PT_2)=T \& S(PT_{0_2})=T$	;$PT_{2_2} \leftarrow PT_2 \otimes PT_{0_2}$	;S(PT_{2_2})←T
	$S(PT_3)=T \& S(PT_{0_3})=T$	;$PT_{2_3} \leftarrow PT_2 \otimes PT_{0_3}$	;S(PT_{2_3})←T
	$S(PT_4)=T \& S(PT_{0_3})=T$	;$PT_{2_4} \leftarrow PT_4 \otimes PT_{0_3}$	;S(PT_{2_4})←T
	all $S(PT_{0_i})=T$	; Nop	;S(T_2)←T
CP	$S(T_2)=T$	; <Terminate Process>	

図4.2 パケットによる実行手順

(注)headとbodyという二つの属性を持った項関係T,T_0に対して、組毎に単一化操作を行い、単一化ができた組の集合を項関係T_2に格納する。

以上に述べたパケットは、演算を実行するCP,UEおよびDKS毎に分配し、プロセッサ毎に実行の可不可を判定することができる。ここで実行可となった処理は、他の処理とは独立にDKSやUEを動かして演算部で示された処理を実行し、処理結果を他の処理に伝達する。この制御に対応するフローを、図4.3に示す。

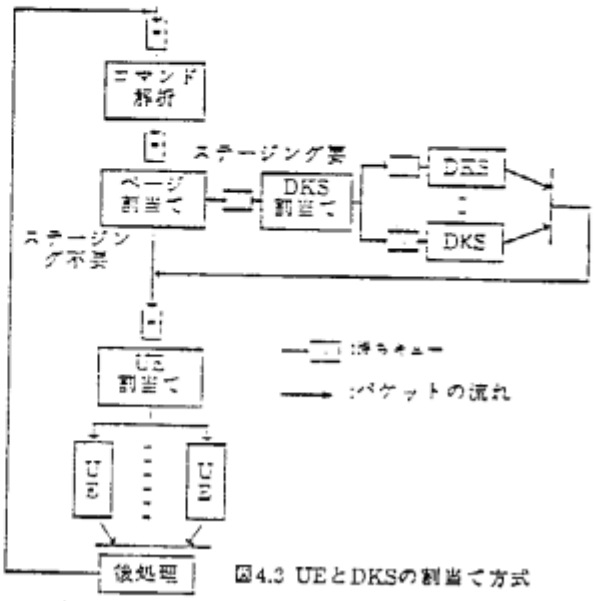


図4.3 UEとDKSの割当て方式

5. まとめ

以上、MPPMとUEを用いた、知識ベースマシンの制御方法について述べた。マルチプロセッサ方式の関係データベースマシンでは、トランザクション毎、関係毎、また関係を形成するページ毎と、並列化のレベルが色々考えられる。本稿では、MPPMの特徴を生かすページ毎の並列処理について述べた。特に、事象駆動型の並列制御方式を取ることにより、プロセッサの割当てオーバヘッドの削減が図れ、効率のよい並列処理が可能となる。今後は、さらに制御方式の詳細化を図り、シミュレータを用いて、処理の単位や並列制御について、定量的な検討をすすめてゆく。

[謝辞]本検討に当たり、有益な御示唆を頂いた知識ベースWGのメンバの方々に感謝いたします。

[参考文献]

[1]横田,他:マルチポートページメモリの構成方法,昭和60年度通信学会部門別全国大会.

[2]Boral,H.,et al.:Implementation of the Database Machine DIRECT,IEEE Trans. on SE,Vol.SE-8, No.6,pp533-543.

[3]横田,他:単一化による知識ベース検索,昭和60年度通信学会部門別全国大会.

[4]森田,他:単一化結合の処理方式,情処第32回全国大会予稿,1M-7

[5]横田,他:単一化エンジンを用いた知識ベースマシンの構成方式,情処第32回全国大会予稿,1M-8

2G-5 GHCプログラムのアルゴリズミック・デバッギングについて

竹内彰一
（財）新世代コンピュータ技術開発機構

1 はじめに

計算機アーキテクチャの発展に伴い、並列計算機が実用化されていくに従って並列プログラムのデバッギングはソフトウェア開発上極めて重要な技術となることが予想される。一方で並列プログラムのデバッギングは逐次プログラムのそれと比較すると非常に困難な仕事であると言われてきた。その理由としては、(1) 複数の計算が並列に行なわれていること、(2) 並列に行なわれている計算が互いに作用を及ぼし合うこと、(3) 逐次プログラムには無いデッドロックのような新種のバグが出現することなどがある。

通常、プログラムのデバッグは実行トレースを基本とし動作を理解する過程で行なわれるが、この方法では多数の計算が錯綜している並列計算のデバッグの際、プログラマの負担が増大し並列プログラミング自身の専門化を招きかねない。これに対しデバッギングのアルゴリズム（バグの存在する範囲の絞り込み）が組込まれているアルゴリズミックなデバッガは、プログラマが必要最小限の情報を与えてやればデバッガ自身がバグの位置を同定することができるという点で、プログラマの能力によらず並列計算の複雑さにも対応できる着実で負担の軽い手法である。

Shapiro は一連の質問をプログラマに発しながらバグを検出するアルゴリズム "Divide and Query" について報告している[2]。本稿ではこの "Divide and Query" アルゴリズムを拡張したものに基づいた並列論理型言語GHC[1]のためのアルゴリズミックなデバッガについて報告する。このデバッガは必要最小限の情報だけをプログラマに提示し、それに対するプログラマの応答に基づいて着実にバグの存在する範囲を狭めていく。またこのデバッガはGHC自身により簡潔に実現されている。

2 GHCプログラムのアルゴリズミック・デバッギング

GHCはユニフィケーションに基づいた非決定的な並列プログラミング言語である[1]。本デバッガは以下に示すような症状を持つGHCプログラムが与えられると、プログラマとの一連の質問応答の後バグを含んだ節のインスタンスを返す。

①ゴールの計算は成功して終了するが出力値（出力ユニフィケーション）が間違っている。

②ゴールの計算がデッドロックに陥る。

プログラマとの質問応答は行なわれた計算がプログラマの意図するものであったかどうかに関するものである。意図と異なる部分が実際に行なわれた計算の中にあれば、そこにバグを含んだ節を用いた計算が行なわれたことになる。プログラムSに対するプログラマの意図としては

（ア）S上で成功終了するすべての計算の入出力関係の集合

（イ）S上でサスペンドするすべての計算の入出力関係

の2つを考え、これをM（S）で表わす。M（S）は非決定的な並列プログラムの意味としては不十分であるが、①、②の症状のデバッギングには十分である。M（S）は陽にすべてを書き表わす必要がなく、デバッギングを行なうプログラマの頭の中にあって、具体的な計算（終了しているか、サスペンドしている）の示す入出力関係がM（S）に含まれるか（意図に合うか）、含まれないか（意図に合わないか）をプログラマが判定できるだけで十分である。

本デバッガではまず最初に①、②の症状を示した計算を木の形で再現する。この木はGHCプログラムのガード部を省略したAND 木になっており、計算木と呼ぶ。計算木は次のような性質を満足する。

(i) 計算木の各ノードは exit(ゴール) または suspended(ゴール) のいずれかである。

(ii)Tを計算木、L0 をそのルート、T1,...,Tn をルートのすぐ下の部分木、L1,...,Ln をそれらのルートとすると

G0 :- G1 , ... , Gn .

はプログラム中に存在するある節のインスタンスである。（ただし、Gi はLi= exit(Gi)あるいはLi=suspended(Gi)を満足するものである）

exit(G) のGはこのノードをルートとする成功した計算の入出力関係を表わしており、 suspended(G) のGは未だ終了してない計算の部分的入出力を表わす。計算木中のノードNの重みw（N）は次のように定義される。

w（N）＝0　if　Nがシステム述語に対応する。

＝1＋w（N1)＋　. . .　＋w（Nm）

otherwise

ただし、Ni （i=1 ～m ）はNの子ノードである。デバッギングのアルゴリズムを以下に示す。

アルゴリズム

入力：　プログラム　S、　ゴール　P

M'≦ M（S）　（最初は空）

出力： Sの節のインスタンスである次のような節

A：－B1，．．．．，Bn．

ただし、A∈M(S)，Bi∈M(S)

ステップ1

ゴールPの計算をシミュレートし計算木を生成する。

ステップ2

計算木のルートの重みwをM´を法として計算する。

もしw＝1ならば

<ルート>：－Q1，．．．．，Qn．

を出力として返す。ただしQ1，．．．．Qnはルートのデノード。w＞1の場合は計算木の中でその重みが［w／2］を超えない最大のノードmを求め、m∈M(S)かどうかをプログラマに問合せる。答がm∈M(S)の場合は同じ計算木についてM´をM´－{m}としてステップ2を再帰的に適用する。答がm∉M(S)の場合は、mをルートとする部分木に対して同じM´のままステップ2を再帰的に適用する。

本アルゴリズムとShapiro のアルゴリズムとの違いはM(S)の意味をGHCのデバッグ向きに拡張した点にある。ノード数がN、最大分岐数がbであるような計算木に対して本アルゴリズムが高々b・logN回の質問を発して後必ず停止してバグを含んだ節のインスタンスを返すことはShapiroの証明と同様にして証明することができる。

3 実現

本デバッガはGHC自身によりGHCのメタ・インタプリタ的なものとして実現されている。デバッガは大きく分けて2つ、質問応答を管理するプロセスと計算木を管理するプロセスとからなっている。前者はプログラマとの質問応答を行ない、過去の質問応答の結果を記憶していて同じ質問を繰返さないようにしている。後者はデバッギングの前半（ステップ1）において計算をシミュレートしながら計算木を木状のプロセス・ネットワークとして生成する。デバッギングの後半（ステップ2）において、この木状のネットワークは木の各ノードの重み値の更新や中点の計算を並列に行なう。現在はデバッガをメタ・インタプリタ的なものとして実現しているが[3] にあるようにこのデバッガをプログラムSに関して部分計算すればSの専用デバッガS´を得ることができる。S´はSに専用なだけもとのインタプリティブなデバッガより効率が良いことが予想される。プログラム・リストを含む詳細は[5] にある。

4 議論

本稿では "Divide and Query" 法にもとづくGHCプログラムのデバッギングについて述べた。高橋らは関数型プログラムを対象に "Divide and Query" 法を基本としながらもプログラムの静的な構造についての情報を用いてデバッギングの効率を改善する方法について報告している[4] 。本稿で示したデバッガについても同様の改善の可能性はあると思われるが、我々はそれを前節で述べた部分計算の延長として行ないたいと考えている。

次のような症状を持つプログラムは現在のデバッガでは対処できないが、適当な拡張を行なえば対処が可能となる。

a 計算が失敗する。

b 無限再帰を含み停止しない。

c ガードの計算中に存在するバグ

次に示すようなバグは本デバッガのアルゴリズムでは原理的に対処できない。

d 非決定的で再現性のないバグ

非決定的で再現性のないバグをデバッグ時に確実につかまえる手段はない。

e スタベーション

スタベーションは一般に不公正なスケジューリングにより生じるが、本デバッガではスケジューリング自身をデバッグの対象とはできない。

f 計算は正しいが途中の動作が間違っている。

一般に非決定的かつ並列プログラムの意味としては、正しい入出力値関係だけでは不十分であることが知られている[6,7] 。しかしながら本デバッガは基本的に正しい入出力関係をベースにしているためこのようなバグはとれない。

これらのバグは非決定的並列プログラムに固有なものであり最も対処の難しいものである。これらはデバッガの対象というよりむしろプログラムの検証系の対象というべきものであろう。

文献

[1] K.Ueda:Guarded Horn Clauses, Proc. of Logic Programming Conf. (Tokyo), 1985.

[2] E.Shapiro: Algorithmic Program Debugging, MIT Press, 1983.

[3] 竹内他：Prologプログラムの部分計算とメタ・プログラムの特殊化への応用， Proc. of Logic Programming Conf. (Tokyo), 1985.

[4] 高橋他：関数依存グラフの変換による関数型プログラムのデバッグ法、情報処理学会ソフトウェア基礎論研究会 WGSF12-3, 1985

[5] A.Takeuchi: Algorithmic Debugging of GHC Programs, ICOT Technical Report, in preparation.

[6] J.B.Brock,M.B.Ackerman: Scenario: A Model of Nondeterminate computation, In Lecture Notesin Computer Science, vol.107, Springer-Verlag, 1981.

[7] J.Staples, V.L.Nguyen: A Fixpoint Semantics for Nondeterministic Data Flow, JACM, vol.32, No.2, 1985

ＧＨＣサブセット逐次型処理系の作成

2G-4

*1 江崎令子　*2 宮崎敏彦　*1 太細孝

*1 三菱電機（株）情報電子研究所　*2 （財）新世代コンピュータ技術開発機構

1．はじめに

ＧＨＣサブセット（ＦｌａｔＧＨＣに、簡単なモジュール構造をサポートしたもの．以下、単にＦＧＨＣ）の処理系を、逐次型推論マシンＰＳＩのヒープ上で実現した．使用した言語は、ＥＳＰである．

ＦＧＨＣとは、ＧＨＣ〔上田85〕に対して以下の制限を加えたものである．

（１）passive partは逐次的に実行する．

（２）guard 部に書ける述語は、組込み述語のみ．

（３）候補節のテストは上から下に逐次的に行なう．

ＦｌａｔＧＨＣのプログラムは、コンパイラにより、Warren流の中間コードに変換される．本稿では、この中間コードを実行するエミュレータの実現方式について述べる．

2．処理の概要

2－1．処理の流れ

あるgoalについての大まかな処理は、次のように進む．

１）goalを ready_queueから取り出す．

２）goalに対応する各clauseの guard部を実行．

- successならば、 trustしてbody部のgoal列を展開し、 ready_queueに入れる．
- suspendならば、goalをsuspension_queueに入れる．
- そのgoalに対応するすべてのclauseを実行して、successも suspendもしないならば、そのgoalはfailなので、ＡＮＤ兄弟をfailさせる．

３）１）へ戻る．

図２－１は、処理の概要を図式的に示したものである．

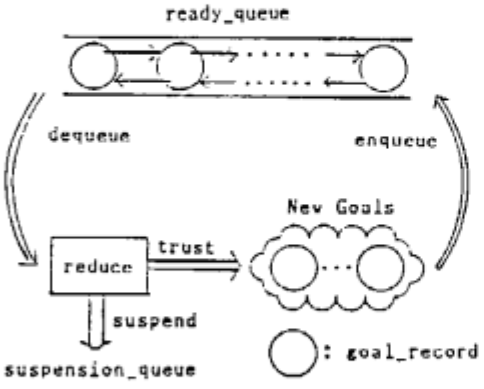


図２－１．処理の流れ

2－2．ゴール・レコード

ユーザ・プログラム中の各goalに対応する．

ゴール・レコードは、スケジューリング用の双方向ポインタ、ＡＮＤ木管理用の双方向ポインタ、各モジュールに対応するclass objectと述語名アトム、引数表、どのメタ・コールに所属しているかを示す識別子、などの要素から成る．

2－3．メタコール・レコード

ユーザ・プログラム中のメタ・コールに対応する．

メタコール・レコードは、スケジューリング用の双方向ポインタ、ＡＮＤ木管理用の双方向ポインタ（兄弟用、子供用）、メタコールの結果を unifyする変数へのポインタ、メタコールの実行をコントロールするコードへのポインタ、識別子、などの要素から成る．

2－4．サスペンション・レコードと サスペンション・キュー

サスペンション・レコードは、goalが未定義変数により suspendした時、そのgoalに対応して作成され、サスペンション・キューを構成する．

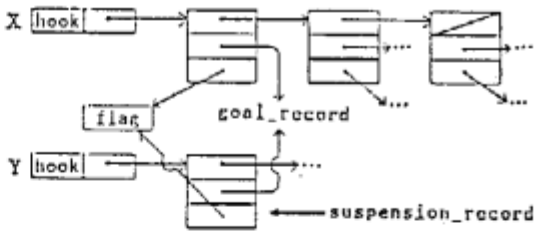


図２－２．サスペンション・キューの構成

サスペンション・レコードは、以下の要素から成る．

（１）同一の変数にhookしているサスペンション・レコード同志をつなぐ単方向ポインタ

（２） suspendしているゴール・レコードへのポインタ

（３）サスペンション・レコードの指しているゴール・レコードが、既にresume（hookしていた未定義変数が instantiateされて、 suspendしていたゴールが、再度レディ・キューに入ること）されたかどうかを示すフラグ

図２－３に、resumeのアルゴリズムを示す．

```
if instantiate(Some_Cell) then
  ( Susp_rec = value(Some_Cell),
    repeat
        Next_record = (next(Susp_rec)),
        Flag = flag(Susp_rec),
        if value(Flag) == off then
          ( enqueue(goal_rec(Susp_rec)),
            value(Flag) := on ),
        die(Susp_rec),
        Susp_rec := Next_record
    until (Next_record =/= null) ).
```

図２－３．resumeのアルゴリズム

３．中間コード

本処理系では、中間コードとして、Warrenの命令セット［Warren83］をＰＧＨＣ用に拡張して使用した。

Warrenのものと基本的に異なるのは、以下の点である。

１）clause中の変数は全てtemporary variableとする。

２）passive part中の unificationを特殊化し、効率化を計っている。

以下に、命令セットの概要を述べる。

３－１．ユニフィケーション系命令

（１） put系命令

引数レジスタから、ヒープ上に取られた引数表に値を格納する。

（２）put_to_reg系命令

引数レジスタから引数レジスタに値をロードする。ＴＲＯのために、 active partの適当なゴールに対して使用される。

（３）get0系命令

clause headの引数の unificationを行なう。

get0_variable以外のget0系命令は、指定された引数レジスタの内容が未定義の場合は、 suspendする。

（４） get系命令

active partの unificationを行なう。基本的にはsuspendしない。

（５）unify0系命令

構造体の要素に対する unification。

get0_structureの後に現われ、 read modeで実行される。 unify0_variable以外のunify0系命令は、指定された構造体の要素が未定義の場合は suspendする。

（６） unify系命令

構造体の要素に対する unification。

put_structureの後に現われ、write modeで実行されるか、または、 get_structureの後に現われ、その結果に応じて、 read modeまたは、write modeのいずれかで実行される。基本的には suspendしない。

３－２．手続き的命令

:alloc_args ・・・ ヒープ上に引数表をとる。

:terminate ・・・ゴール・レコード領域を解放して、ＡＮＤ木の操作を行なう。

:create ・・・ ゴール・レコードの作成と、レディ・キューへの enqueueを行なう。

:first_create,:only_one_create,:last_create など、レコードの再利用を行なうよう最適化されたものもある。

以上の命令の他に、組込み述語などに対応する命令コードが考えられる。

３－３．コンパイルド・コード

ＦＧＨＣプログラムは、コンパイラによって、ＥＳＰのプログラムに変換される。図３－１．は、コンパイルド・コードの例である。

```
%%% FlatGHC source program %%%
sift([P|L],S) :- true | S=[P|S1], shift(K,S1),
                                  filter(P,L,K).
%%% compiled code %%%
:sift(C, Vs, Bnd) :-
    :get0_list(C, 0, Vs),
    Bnd>0,
    :unify0_variable(C, 0, Vs),
    :unify0_variable(C, 4, Vs),
    !,
    :get_list(C, 1, Vs),
    :unify_value(C, 0, Vs),
    :unify_variable(C, 3, Vs),
    :alloc_args(C, 2, As),
    :put_variable(C, 2, Vs, 0, As),
    :put_value(C, 3, Vs, 1, As),
    :only_one_create(C, sift, As, Bnd-1),
                % make goal_record and enqueue
    :put_value_to_reg(C, 4, 1, Vs),
    :filter(C, Vs, Bnd-1) ;
                % direct call
```

図３－１．中間コード・プログラム

図３－１．において、最後のgoal、filterは、レディ・キューに入らず、直接呼び出されることで、高速化が計られている。

４．評価と課題

以上に示した仕様に基づき、逐次型推論マシンＰＳＩ上にＧＨＣサブセットの処理系を作成した。

実行速度は、30要素の naive reverseで、0.87krps.であった。

現在の処理系では、逐次的に処理を行なう事を前提としているが、今後、数台のマシン上で実際に分散処理を行なう処理系へと拡張していく予定である。

「参考文献」

［上田85］Kazunori, Ueda:
"Guarded Horn Clauses"
ICOT Technical Report 1985.

［Warren83］Warren, D.H.D.:
"AN ABSTRUCT PROLOG INSTRUCTION SET"
SRI International .
Technical Note 309, 1983.

［宮崎85］宮崎敏彦：
"ＫＬ１－Ｂ（第０版）の提案"
ＩＣＯＴ　ＫＬ１－Ｂ検討会資料，1985.

4N-12

Analogical Reasoning as Probabilistic Reasoning

Takashi Yokomori
IIAS-SIS, Fujitsu Ltd.

1. Introduction

We note there are various contexts in which people say "one is analogous to the other". The expression "as like as two peas", for example, comes from the analogy of appearance, while "similar story" and "inherited temper" fall under another category of expressions corresponding to another type of analogy. One may classify analogies as follows:

- syntactic analogy (a)
  [form, appearance, symbolism]
- semantic or functional analogy
  - dynamic or operational analogy (b)
    [temper, performance]
  - static analogy (c)
    [situation, sense(color, taste)].

In the study on analogical reasoning, [H85a] discusses and proposes one method for realizing the analogical inference based on its "theory of analogy" which only concerns the analogy of type (a). The key idea there is to formalize the concept of analogy using "inductive generalization" by [P70]. Provided that any object in analogical reasoning can be expressed as a finite set of ground atoms in Horn logic, the analogy between the two objects S_1 and S_2 is defined as follows:

S_1 and S_2 satisfy the partial identity condition if there exists a one-to-one mapping f such that $f(Sg_1) = Sg_2$, where S is the least generalization of S_1 and S_2 for substitutions g_1 and g_2, i.e., S is the least general such that $Sg_1 \subset S_1$ and $Sg_2 \subset S_2$.

In the definition above, f claims a strict one-to-one correspondence between the two objects, which causes some difficulties in dealing with common cases of analogy. It cannot handle analogies between the two objects having distinct domains, further, even if in the same domain, a semantic analogy of the kind (b) or (c) above neither.

In the later paper [H85b] Haraguchi extends his formulation of analogy (partial identity condition) to overcome some of its drawbacks above and to deal with a kind of semantic analogy to some extent. There still remain, however, quite a few problems to be solved in the extended version. For example, the identification of a concept in different formulations cannot be treated, and more typically, no consideration has been given to the problem of handling the "everyday" analogy used in daily life such as "like something".

The goal of this paper is to show that taking Dempster's theory of probability as a formal framework analogical reasoning is regarded as probabilistic reasoning, and to surge on a possible way of handling semantic analogy in that framework.

2. Semantic Analogy and Analogical Reasoning

As mentioned in the previous section, by semantic analogy we mean the types (b) and (c) of analogy. However, our intension here is mainly the type (c), that is, static analogy. In what follows, we simply refer to semantic analogy as analogy, unless any confusion arises. We also assume that the objects in discourse of analogy are described in Horn logic formulas.

Consider the followings that give typical examples of analogy in daily life.

Let parent(X,Y) be a predicate meaning X is a parent of Y, and couple(X,Y) meaning X and Y are a couple. In general, we have

grand-parent(X,Y)<-parent(X,Z),parent(Z,Y) --- (1)

in-law(R,X,Y)<-R(X,Z),couple(Z,Y) --- (2)

where R is a relation name. (For example, in-law(parent, tom, mary) denotes "tom is a parent-in-law of mary".) Further, it holds

transitive(R,X,Y)<-R(X,Z),R(Z,Y).

Then, firstly, (1) is re-expressed as follows:

grand-parent(X,Y)<-transitive(parent,X,Y) --- (1').

This implies that we must take the analogy (in fact, identity) of this kind into account. It is too numerous to count up instances of this type. (The identities of "ancestor" and "transitive closure of parent", of "male" and "antonym of female", etc.)

Secondly, since

in-law(grand-parent,X,Y)<-
grand-parent(X,Z), couple(Z,Y),

we eventually have

in-law(grand-parent,X,Y)<-
transitive(parent,X,Y),couple(Z,Y) --- (3).

Comparing (1') and (3) instead of (1) and (3), it is suggested that there is a similarity in syntax between the two expressions which emerges through the "analogy relation" couple, while the comparison of (1) with (3) can never give us a chance to detect even a syntactic analogy between the two.

Note that the second example provides an interesting view of analogy in which one may say semantic analogy is transformed into syntactic one.

Now, considering analogical reasnoning we encounter in everyday life, it can be, we believe, characterized by :

(i) detecting analogy mostly in terms of subjective criteria,
(ii) non-monotonic reasoning features,
leading to
(iii) approximate reasoning based on incomplete knowledge base.

Thus, it seems quite reasonable to look at analogical reasoning as non-monotonic reasoning in which approximate reasoning using subjective probability plays the central role.

3. Dempster's Probability Theory

In [Sh76] Shafer has discussed the way of treating uncertain knowledge such as "A bird can fly" or "People can see", and given a suggestive view that certainties of those are better thought of not as specific values but as ranges. He has refined Dempster's work [D68] on the theory of probability which extends the classical Bayesian theory so as to cope with a task of treating "ignorance" and "disbelief". (For the details on Dempster and Shafer's theory, see [I83] as well as the two above.)

As mentioned above, if we take the position to look at analogical reasoning as non-monotonic reasoning, then the analogical reasoning system we intend must have a

capability of inference based on "uncertain knowledge" in addition to that of belief revision. As shown below, Dempster's rule has many attractive properties in the sense above.

Here, we follow the notation in [G84a] and [G84b]. The truth values T and F are thought of as probabilities 1 and 0, respectively. A probability interval [a,b] (where $0<a<b<1$) is denoted by the pair (a,b), where $\bar{b}$ stands for $1-b$ and $a+\bar{b}<1$. Intuitively, "a" means the degree to which a given statement is confirmed(to be true) by the available evidence, while "$\bar{b}$" the degree to which it is disconformed(or doubted to be false). Examples are a true statement with the value (1,0), a false statement with the value (0,1).

Incomplete information is represented as follows. "A bird can fly" might have truth value (.9,.02), or "Humans are mammals" has the value (1,0), while "Desk can see" which is an example of extremely incomplete information might have the value (0,0). Thus, generally, the completeness of information represented by the value (a,d) corresponds to the sum $a+d$ called the mass of the truth value.

The rules are associated with their truth values as follows:

Q <- (a,d) $P_1,\ldots,P_n$.

For example, can-fly(X)<- (.9,.02) bird(X).

To perform one inference derivation using a rule, assume that bird(x) has the truth value (a,d). Then, can-fly(x) is drawn with the truth value (.9a,.02a). Note that "d" does not contribute to this value, because the rule cannot be used to draw any conclusion from the fact that "x is not a bird". (In case when a rule has multiple antecedents, a product of their each value should be used as the truth value of the total certainty for them.)

Suppose that

bird(peeko) (1,0)

can-fly(X)<- (.9,.02) bird(X).

Then we have a conclusion

can-fly(peeko) (.9,.02) --- (4).

On the other hand, if isa(peeko,penguin) (1,0) is obtained later, then using a rule

can-fly(X)<- (0,1) isa(X,penguin),

we eventually draw

can-fly(peeko) (0,1) --- (5),

leading to a typical situation (a kind of contradiction) in non-monotonic reasoning.

Dempster has discussed this situation in detail and given a solution to this difficulty. If we denote by (a,b)+(c,d) the inference obtained by combining rather contradictory two inferences (a,b) and (c,d), then Dempster's rule gives us

$$(a,b)+(c,d) = \left(1-\frac{\bar{a}\bar{c}}{1-(ad+bc)},\ 1-\frac{\bar{b}\bar{d}}{1-(ad+bc)}\right).$$

The operation + has many preferable properties for our purpose. Those are

[1] (a,b)+(c,d)=(c,d)+(a,b)

[2] (a,b)+((c,d)+(e,f))=((a,b)+(c,d))+(e,f)

[3] (a,b)+(0,0) = (a,b)

[4] (a,0)+(c,0) = (a+c-ac,0)

[5] (1,0)+(c,d)=(1,0) if (c,d)≠(0,1)

[6] (0,1)+(c,d)=(0,1) if (c,d)≠(1,0)

[7] (1,0)+(0,1) is undefined

[8] if we denote by - the inverse of +, for (c,d) ≠ (0,1) or (1,0), we have

$$(a,b)-(c,d)=\left(\frac{\bar{c}(a\bar{d}-\bar{b}c)}{\bar{a}\bar{d}-\bar{b}c-\bar{a}d},\ \frac{\bar{d}(b\bar{c}-\bar{a}d)}{\bar{a}\bar{d}-\bar{b}c-\bar{a}d}\right).$$

This allows us to retract specific inferences without affecting others and provides many of the same features as a truth mentainance system.

4. Detection of Analogy

We shall discuss how analogical reasoning is incorporated in non-monotonic reasoning using Dempster's probability theory. Although Dempster's theory is suitable for dealing with subjective probability, we feel the detection of analogy should be performed by the mechanism of some kind that is based on objective measures. Such a measure is used for computing the degree of analogy.

Let A(x) and B(x) be predicates having their Herbrand domains Dom(A) and Dom(B), respectively. Assume that there is a one-to-one mapping F from Dom(A) to Dom(B) and all elements of Dom(A)(or Dom(B)) are enumerable in some fixed order. Given an A(x), define

L(A)={w in Dom(A) | A(w) holds true}.

Further, let d_F be defined as follows:

$d_F(A,B)= dist(F(L(A)),L(B))$,

where $dist(L_1,L_2) = norm(L_1 \oplus L_2)$,

$norm(L)= \sum^{\infty}_{i} m_i f_i$, and

f_i = 1(i-th element of the domain is in L)

f_i = 0 (otherwise),

$m_i = 2^{-i}$ (i>1), $\sum m_i = 1$,

$\oplus$ denotes the symmetric difference.

It is easily seen that a mapping dist satisfies the metric axioms, i.e. it provides a metric for measuring the analogy of the two concepts. Note that $d_F(A,B)=0$ iff A=B under F, $d_F(A,B)=1$ iff A and B are complementary, and $0<d_F(A,B)<1$ holds.

Using a metric d_F, analogical reasoning is performed in the framework of probabilistic reasoning as shown below.

Suppose the Domains D_1 and D_2 between which there exists a one-to-one mapping F are given. Further, let S_i be a set of Horn clauses with D_i(i=1,2) in which each fact or rule of S_i is already assigned its truth value (a,d). (If a clause is confirmed true, then (a,d)=(1,0).) Then, consider the union $S= S_1 \cup S_2$ as the whole system for performing reasoning in $D_1 \cup D_2$.

There exist two types of rules to be possibly added to S.

(1) If A is a ground atom in S_1, then

F(A) <- (a,d) A is possibly a rule,

where the truth value (a,d) is determined from F by subjective criteria.

(2) If $d_F(A,B)= t$, then B <- (1-t,t) A is possibly a rule.

Thus, the reasoning system S with additional rules works as a probabilistic reasoning system using Dempster's rules.

This work is part of the major R&D of the FGCP, conducted under program set up by MITI.

References

[D68]Dempster,A.P.,A Generalization of Bayesian Inference, J.Roy.Stat.Soc. B, 30 205-247,(1968).

[G84a]Ginsberg,M.L., Non-monotonic Reasoning Using Dempster's Rule, Proc.AAAI-84, 126-129,(1984).

[G84b]Ginsberg,M.L., Implementing Probabilistic Reasoning, Res.Rep. No.84-31, Stanford Univ.,(1984).

[H85a]Haraguchi,M., Analogical Reasoning Based on the Theory of Analogy, RIFIS-RR, Kyushu Univ.No.105 (1985),

[H85b]Haraguchi,M., Towards Semantics of Analogical Reasoning, paper at Inf.Sys. Symp., IIAS-SIS, Fujitsu Ltd.,Nov.1985.

[I83]Ishizuka,M., Dempster & Shafer's Theory, IECE of Japan, Sep. 900-903,(1983).

[P70]Plotkin,G.D., A Note on Inductive Generalizations, Machine Intelligence, 5, 153-216, (1970).

[Sh76]Shafer,G.,"A Mathematical Theory of Evidence", Princeton Univ.Press,(1976).

4M-6

On Semantics of Logic Programs with Uncertainties

Yasubumi Sakakibara
IIAS-SIS, Fujitsu Ltd.

1. Introduction

A mechanism for dealing with uncertainties plays a central role in knowledge systems like expert systems. Especially, incorporating uncertainties into logic programs (e.g. Fuzzy-Prolog [2],[3]) is a current study. On the other hand, there are few semantic considerations for it. In this paper, we will first attempt to give a semantics for logic programs with uncertainties. This is a further work of [4] which is the only paper dealing with a semantics so far. Our basic definitions of logic programs with uncertainties follow it. Secondly, we will define a proof procedure for logic programs with uncertainties which corresponds to an interpreter for it and present a sufficient condition for certainty functions of clauses to achieve the completeness of the proof procedure.

2. Logic programs with uncertainties

A definite clause is a clause of the form $A \leftarrow B$, where A is an atom and B is a conjunction of zero or more atoms. A certainty factor c is a real number between 0 and 1. A certainty function f is a function from multisets of certainty factors to certainty factors. A logic program with uncertainties P is a finite set of pairs $(A \leftarrow B, f)$, where $A \leftarrow B$ is a definite clause and f is a certainty function.

The certainty function computes the certainty of the conclusion of a clause from the multiset of certainties of solutions to goals in the condition of the clause. Two requirements of f in [4] are that for every multisets S, $f(S \cup \{1\}) = f(S)$, and that f be monotonic increasing, which means that $S \subseteq S'$ implies $f(S) \leq f(S')$, where $\leq$ is the partial order over multisets, defined as follows. Let S and $X = \{x_1, \ldots, x_n\}$ $(n \geq 0)$ be two multisets. Then $X \leq S$ iff there is a multiset $Y = \{y_1, \ldots, y_n\}$ such that $S \supseteq Y$ and $x_i \leq y_i$ for $0 \leq i \leq n$.

3. Semantics for logic programs with uncertainties

An interpretation I of a logic program with uncertainties P is a set of pairs (A,c), where A is a ground atom and c is a certainty factor. An I contains at most one pair (A,c) for any atom A. A ground atom A is true in I with certainty c iff there is a pair (A,c') in I such that $c \leq c'$. An atom A is true in I with certainty c iff for every ground instance A' of A, A' is true in I with certainty c. Let $A \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ be a ground definite clause, f be a certainty function and S be the multiset of certainties $\{c_1, \ldots, c_n\}$ such that (B_i, c_i) is in I for $1 \leq i \leq n$. Then $A \leftarrow B_1, \ldots, B_n$ is true in I w.r.t. f iff A is true in I with certainty f(S). A definite clause $A \leftarrow B$ is true in I w.r.t. f iff any ground instance $A' \leftarrow B'$ of it is true in I w.r.t. f. A logic program with uncertainties P is true in M iff for any pair $(A \leftarrow B, f)$ in P, $A \leftarrow B$ is true in M w.r.t. f. Such an interpretation M is called a model for P.

A partial order $\leq$, intersection $\cap$ and union $\cup$ on interpretations are defined in the natural way as follows. Let I and I' be interpretations. Then $I \leq I'$ iff for any pair (A,c) in I there is a pair (A,c') such that $c \leq c'$. (A,ci) in $I \cap I'$ iff for two pairs (A,c) in I and (A,c') in I' $ci = \min(c,c')$. (A,cu) in $I \cup I'$ iff for two pairs (A,c) in I and (A,c') in I' $cu = \max(c,c')$.

Lemma 3.1 (model intersection property) Let **M** be any nonempty set of models for P. Then $\cap \mathbf{M}$ is also a model for P.

By the above lemma, the least model $\cap M(P)$ for P which is the intersection of all models for P exists.

We define a transformation Tc_P associated with P in the same way as standard logic programs. Let I be an interpretation. (A,c) in $Tc_P(I)$ iff $c=\sup\{f(S) \mid A \leftarrow B_1,\ldots,B_n$ is a ground instance and f is the certainty function of a clause in P such that (B_i,c_i) in I for $1 \leq i \leq n$ and $S=\{c_1,\ldots,c_n\}\}$.

Lemma 3.2 Let I be an interpretation. Then I is a model for P iff $Tc_P(I) \leq I$.

Lemma 3.3 Tc_P is a monotonic mapping.

By the above lemmas and the Knaster-Tarski fixpoint theorem [1], we have the following theorem.

Theorem 3.4 The least model $\cap M(P)$ for P is equal to the least fixpoint of Tc_P ($lfp(Tc_P)$ for short).

We consider $\cap M(P)$ and $lfp(Tc_P)$ as the semantics for a logic program with uncertainties P.

4. Proof procedure for logic programs with uncertainties.

A <u>(P,F)-derivation</u> is a sequence of quadruples $\langle G_i,C_i,\theta_i,F_i\rangle$, $i=0,1,\ldots$ such that (1) G_i is of the form $\leftarrow(B_1,\ldots,B_m)$ where $m \geq 0$ and B_j is an atom $(1 \leq j \leq m)$ and F_i is of the form $\{X_1,\ldots,X_m\}$, (2) C_i is a list of m clauses $(A^{(1)} \leftarrow D_1^{(1)},\ldots,D_{n1}^{(1)}, f^{(1)}),\ldots,(A^{(m)} \leftarrow D_1^{(m)},\ldots,D_{nm}^{(m)}, f^{(m)})$, (3) θ_i is a most general unifier of $(B_1,\ldots,B_m)$ and $(A^{(1)},\ldots,A^{(m)})$, and (4) G_{i+1} is $\leftarrow(D_1^{(1)},\ldots,D_{n1}^{(1)},\ldots,D_1^{(m)},\ldots,D_{nm}^{(m)})\theta_i$ and F_{i+1} is $\{Y_1^{(1)},\ldots,Y_n^{(1)},\ldots,Y_1^{(m)},\ldots,Y_n^{(m)}\}$ where $X_1=f^{(1)}(\{Y_1^{(1)},\ldots,Y_n^{(1)}\}),\ldots$, $X_m=f^{(m)}(\{Y_1^{(m)},\ldots,Y_{nm}^{(m)}\})$. A (P,F)-derivation is <u>successful</u> if it is finite and its last goal (some G_i) is empty. The certainty of a successful (P,F)-derivation is F_0 in it. Then we have the following theorem which means the soundness of a successful (P,F)-derivation.

Theorem 4.1 For a ground atom A, if A has a successful (P,F)-derivation with the certainty c, then A is true in $\cap M(P)$ with c.

However we do not generally have the completeness of a successful (P,F)-derivation. With the restriction on a certainty function f that $f(\{c_1,\ldots,c_m\}) \leq \min(c_1,\ldots,c_m)$, the completeness can now be achieved.

Lemma 4.2 Assume that for each certainty function f, $f(\{c_1,\ldots,c_m\}) \leq \min(c_1,\ldots,c_m)$. Then for (A,c) in $\bigcup_i Tc_P^{\,i}(\phi)$, there exists some N such that (A,c) in $Tc_P^{\,N}(\phi)$.

Theorem 4.3 Assume that for each certainty function f, $f(\{c_1,\ldots,c_m\}) \leq \min(c_1,\ldots,c_m)$. For a ground atom A, if A is true in $\cap M(P)$ with c, then A has a successful (P,F)-derivation with certainty c' such that $c \leq c'$.

We believe this condition (i.e. $f(\{c_1,\ldots,c_m\}) \leq \min(c_1,\ldots,c_m)$) will play an essential role in the validity of quantitative reasoning.

This work is part of the major R&D of the FGCP, conducted under program set up by MITI.

References

[1] K.R.Apt and M.H.van Emden, "Contributions to the theory of logic programming", JACM, 29, 3(1982).

[2] M.Ishizuka and N.Kanai, "PROLOG-ELF INCORPORATING FUZZY LOGIC", Proc. 9th IJCAI, 1985.

[3] M.Mukaidono, "Fuzzy Prolog no kokoromi"(in Japanese), Proc. 27th Conf. JIPS, 1983.

[4] E.Y.Shapiro, "Logic Programs With Uncertainties", Proc. 8th IJCAI, 1983.

On Semantic Analogy in Object-oriented Representation

4M-13

Yuji TAKADA
IIAS-SIS, Fujitsu Ltd.

1. Introduction

Analogy is the very powerful and basic human information processing in reasoning and learning. Therefore, various analogies have ever been discussed, but most of them did not have flexibilities or logical validities. In this paper, we discuss a study of analogy in an **object-oriented language** based on a **logic programming language** like PROLOG, and show some flexibility and validity of this analogy.

In such languages, each object represents an axiom set. For knowledge representation and processing, one of characters of a logic programing language provides some validities, while that of object-oriented language does some flexibilities. Here, we discuss analogies from two different viewpoints. One concerns logical semantics, taht is, the one-to-one correspondence between models. The other concerns more structural relationships between objects.

2. Object-oriented Representation

In this paper, the objects in discourse of analogy are represented in an object-oriented language like **KORE/KR** [K85] or **ESP** [C84] based on a logic programming language.

Class is defined as follows :

```
class <class name> has
    [isa  <is-a definition> ;]
    [part <part definition> ;]
    [slot <slot definition> ;]
      [ <method definition> ]
end.
```

where underline indicates a reserved word and [...] indicates "..." is optional. <is-a definition> comprises a sequence of class names, which form super classes for multiple inheritance mechanism, while <slot definition> a sequence of slot names. <part definition> comprises a sequence of relations of the form P<C, where P is a variable called **part** and C is a class name, meaning that the value of P is an instance belonging to a class C. Instantiating P, together with an instance beloging to the class <class name>, provides a PART_OF relationship. <method definition> is a collection of methods. Method M_i (i=1,....n) is defined as follows :

$$MP_i \Rightarrow MC_{i1},\ldots,MC_{im(i)} \quad (\ m(i)\geq 0\ )$$

where MP_i is a "term" in first-order langunge. MC_{ij} is called "method call" and denoted by I<-m, where m a **message**, I a variable supposed to be instantiated with an instance. We call the right side of => **body.**

Suppose an object represents an axiom set. Then a method call I<-m implies that m is interpreted in the axiom set I. The interpretation of method M_i depends on all instances appearing in method calls of MC_{ij}s, therefore so does the interpretation of an object with M_i. Hence, it is possible to take that PART_OF explicitly represents a dependency of models between objects.

3. Definition of analogy as renaming

Analogy in this paper is based on **renaming**, which gurantees one-to-one correspondence between models. We take mutual recursively definitions in the following.

Definition 1 For given two terms t_1 and t_2, **renaming** F is a non-empty set of pairs recursively defined as follows :

1) If both t_1 and t_2 are constants exclusively or variables, then $F=\{<t_1,t_2>\}$.

2) If $t_1=f(u_1,\ldots,u_n)$ and $t_2=g(v_1,\ldots,v_n)$ then for each i=1,...,n there exists renaming F_i for u_i and v_i, and $F=\{<f,g>\}\cup\bigcup_{i=1}^{n}F_i$ is one-to-one.

Definition 2 Given two classes C_1 and C_2, let S_i be the set of all slot names in C_i (i=1,2). Then, we say C_1 and C_2 are **analogous on slots** iff $S=S_1\cap S_2$ is non-empty. Renaming $F_s=\{<s,s>|s\in S\}$ is called **slot-analogy.**

Definition 3 Let method call MC_i be I_i<-m_i, and suppose an instance denoted by I_i belongs to C_i (i=1,2). Then, we say MC_1 and MC_2 are **analogous on method call** iff :

1) Classes C_1 and C_2 are analogous in the manner below. Let F_c be a renaming for C_1 and C_2. (See definition 5)

2) There exists a renaming F_m for messages m_1 and m_2.

3) $F_{mc}=F_c\cup F_m$ is a renaming.

We call F_{mc} **analogy on method call.**

We say two methods M_1 and M_2 are **comparable** iff both MP_1 and MP_2 have the same arity and m(1)=m(2).

Definition 4 For two comparable methods M_1 and M_2, let $MCS_i=\{MC_{ij}|j=1,\ldots,n\}$, where MC_{ij} is a method call in the body of M_i (i=1,2). Then, M_1 and M_2 are **analogous on method** iff :

1) There exists a renaming F_{mp} for MP_1 and MP_2.

2) If n>0, then there exists a one-to-one correspondence R on $MCS_1\times MCS_2$, where there exists an analogy on method call F_j for each pair in R, and $F_b=\bigcup_{j=1}^{n}F_j$ must be a renaming. In

the case when n=0, i.e. both MCS_1 and MCS_2 are empty, F_b is empty.

3) $F_m = F_{mp} \cup F_b$ is a renaming.

We call F_m a **method-analogy.**

Definition 5 Given two classes C_1 and C_2, let F_s a slot analogy for their slots, F_M a union of some of the method-analogies for their methods. Then, we say C_1 and C_2 are analogous if there exists a renaming $F = F_s \cup F_m$. We call F **class-analogy**, or merely **analogy.**

Semantically, for analogous slots, only correspondence between their models is the predicate since detections of slot-analogy concern no slot-values, while for analogous methods, there exists a strict one-to-one correspondence between their models. Thus, the analogies here have various semantics, which may correspond to those of conditions for analogies like PIC and SPIC in [H85].

4. The role of PART_OF on analogy

On the basis of those analogies mentioned above, there exists another type of analogy, **analogy on parts**, where PART_OF relationships play the central role. This type of analogy conerns more structures of knowledge, rather than logical validities.

Definition 6 Let P_i be the set of all parts in C_i (i=1,2) and let $P \subset P_1 \times P_2$. Then, we say classes C_1 and C_2 are **analogous on parts** iff :

1) For any $q = \langle p_1, p_2 \rangle \in P$, there exists an analogy F_q for classes C_{p1} and C_{p2} which p_1 and p_2 belong to, respectively.

2) P is one-to-one.

3) $F_p = \bigcup_{q \in P} F_q$ is a renaming.

F_p is called **part-analogy.**

Now, taking the part-analogy into account, we extend the definition of analogy for classes so as $F = F_s \cup F_M \cup F_p$ to be a renaming. Suppose PART_OF relationship explicitly represents a dependency between objects. Then the part-analogy concerns the axiom sets an object depends on, rather than its own one.

PART_OF relationships and part-analogies provide some flexibilities. In learning, objects may often have no slot or method but parts. In such cases, PART_OF and part-analogies provide heuristic guides for acquisitions of not only new slots or methods but new classes. For example, consider Example-1 in which water-pipe and electric-chord, water and electricity are analogous. Then pipe-law and ohms-law, which would be acquired from the pressure method (i.e. voltage method) by transforming pipe-law's one, are part-analogy. Similarly, part-analogies provide the same guides in analogical reasoning.

Moreover, in acquiring methods, each of which has only analogous parts in its method calls, there exists a semantic validity in the sense of method-analogy, that is, the acquired method in one class should be precisely method-analogous to the method in the other part-analogous one. Therefore, PART_OF relationship and part-analogy provide guides for acquisitions with that semantic validity.

5. Discussion

Consider an object in this language as the unit of knowledge representation, i.e. the schema like frame. Then, we may take that various semantic levels of analogy like slot- or method-analogy would be due to various qualities of knowledge. Actually, since knowledge has declarative or procedural properties, this fact suggests that it is possible to take slot-analogy as declarative analogy and method-analogy as procedural analogy.

Moreover, for reasoning and learning by analogy, part-analogy may provide some logical validities as well as flexibilities, because PART_OF relationship concerns not only the knowledge structures but dependencies of semantics between objects.

Hence, we believe that the approach discussed here is more fruitful, and developing a theory and a system based on it for analogical reasoning in this knowledge representation is our current work.

This work is part of the major R&D of the FGCP, conducted under program set up by MITI.

References

[C84] Chikayama,T., ESP Reference Manual, ICOT Technical Report, TR-044,1984.
[H85] Haraguchi,M., Towards semantics of Analogical Reasoning, paper at Inf.Sys. Symp., IIAS-SIS, Fujitsu Ltd.,Nov.1985.
[K85] Katayama,Y. et al., Mondai Kaiketsu Sien Kankyou KORE (Sono 3), 5L-10, in this proceedings,1986.

Example-1

Note : *self is a pseudo-variable, getslot() is built-in method call which gets a slot value.

```
class pipe-law has
   part   p<water-pipe, w<water;
      pressure(R*I) =>
         p<-resistance(R), w<-flow(I);
end.
class water-pipe has
   slot   length, area;
      resistance(rho*(L/S)) =>
         *self<-getslot(length,L),
         *self<-getslot(area,S);
end.
class water has
   slot   volume, time;
      flow(V/T) =>
         *self<-getslot(volume,V),
         *self<-getslot(time,T);
end.
class ohms-law has
   part   c<electric-chord, c<electricity;
end.
class electric-chord has
   slot   length, area;
      resistance(phi*(L/S)) =>
         *self<-getslot(length,L),
         *self<-getslot(area,S);
end.
class electricity has
   slot   quantity, time;
      current(Q/T) =>
         *self<-getslot(quantity,Q),
         *self<-getslot(time,T);
end.
```

5L-8

問題解決支援環境ＫＯＲＥ（その１）
――概要――

新谷虎松・片山佳則・平石邦彦・戸田光彦
（富士通㈱国際情報社会科学研究所）

１．はじめに

KORE(Knowledge Oriented Reasoning Environment)はProlog上に構築された汎用の問題解決環境であり，固有なプログラミングパラダイムを提供するためのいくつかのサブシステムにより構成される．これらサブシステムは，共通な内部表現を有することによりKOREとして統合化される．　有益な問題解決機能を提供するためには，システム内に人間の知識・問題解決能力を効果的にモデル化（知識ベース化）する必要がある．我々は，それらをモデル化するための表現技法として対象指向的表現法とルール指向的表現法を用いる．このモデル化に必要な統一的な内部表現技法は，Prologにおける関係テーブル表現技法〔１〕を導入する．このような関係テーブル技法は，システムに自然な知識表現能力と独力な推論機能を提供する一方，知識ベースとデータベースとの自然なインターフェイスの構築を可能にする．

２．ＫＯＲＥの基本構成

ＫＯＲＥは次の基本サブシステムにより構成される．(1)KORE/DB(Data Base subsystem)〔９〕，(2)KORE/IE (Inference Engine subsystem)，(3)KORE/KR (Knowledge Representation subsystem)〔８〕，(4)KORE/EDEN (Extended Dependency network subsystem)〔７〕これらサブシステムは，単独でも独立的に機能し，各サブシステムは内部表現形式において統一化することができ，全体としてＫＯＲＥに様々な問題解決支援機能を提供する．これらサブシステムの基本機能の概要は次のように記述することができる．(1)のKORE/DBはKOREにおける関係テーブルを操作管理する機能を担う一方，単独では関係データベースを操作するためのパーソナルな関係データベースシステムとして機能する〔２〕．　(2)のKORE/IE はKOREにおけるルール指向的問題解決・知識表現機能を担う一方，単独ではOPS5〔３〕等に代表される強力な前向き推論型プロダクションシステムとして機能する．(3)のKORE/KRはKOREにおける対象指向的知識表現・問題解決機能〔４〕を担う一方，単独では柔軟な知識表現言語として機能する．　(4)のKORE/EDEN はKOREにおいて知識を管理・利用するために必要となる知識間の関係情報〔５〕を保存し，知識ベース管理機能の一機能を担う一方，単独ではネットワークを操作するための強力なシステムとして機能する〔６〕．これら基本サブシステム間の関係は図１のように図示できる．

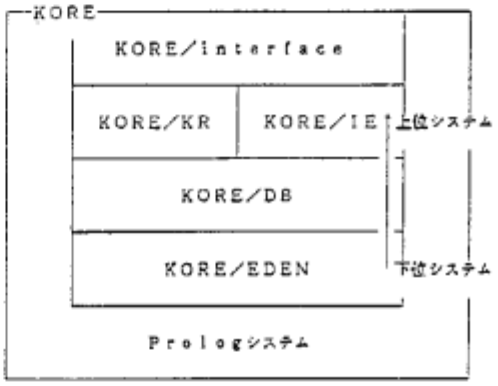


図１．ＫＯＲＥの構成

KOREは図１で示すように各サブシステムはその記述言語であるPrologとのインターフェイスを持つ．これらサブシステムにはレベルがあり，図１で示すような上位―下位概念が存在する．下位システムに位置づけされたサブシステムは，その上位のサブシステムの基本内部表現の管理のための基本機能を担う．例えば，KORE/IE の下位システムであるKORE/DB はKORE/IE におけるグローバルデータベース管理機能を提供し，　KORE/EDEN はKORE/DB の関係情報管理機能を提供する．

KOREにおいて，上位システムは下位システムの機能を利用できるが，下位システムは上位システムをを利用するための機能は与えられない．これは，各サブシステムの働きを明確にすることにより，各サブシステムを用いたアプリケーションの設計・構築法を明らか

にし，KOREの基本的なシステム設計とデバッキングを容易にすることに貢献する．

3．KOREの内部表現技法の概略

KOREは主に4つのサブシステムにより構成され，これらサブシステムは内部表現において共通の関係テーブル表現を持つことにより統合される（図2参照）．

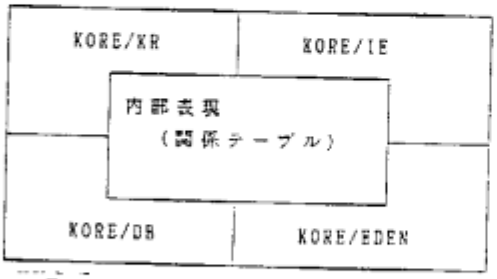


図2．KOREサブシステムの統合

関係テーブル表現は関係表現とその表現力において等価であり〔1〕，Prologにおいては素直に表現ができる．例えば，以下のように関係テーブルはPrologのアサーションにより表現できる．

測定	名前	体重	身長
	太郎	80	172
	次郎	75	180
	花子	55	157

測定（太郎，80，172）
測定（次郎，75，180）
測定（花子，55，157）

このような関係表現における引数の位置はテーブルの列に相当し，各々の関係はテーブルの行に相当する．KOREでは関係テーブルをこのような関係表現により保存し，利用する．

4．サブシステム統合化の例

図3はKORE/KR とKORE/DB が内部表現において統合化され利用されることを示している．図3においてStep 1 ～Step 4 まではKORE/KRにおけるメッセイジ・パッシングフローを示しておりStep 1はクラスcar のロード，Step 2はクラスcar からのインスタンスcivic の生成，Step 3はインスタンスcivic への属性値付加，Step 4はインスタンスcivic の内部状態の表示ために用いられている．Step 5はKORE/DB による関係テーブルの検索が行われ

```
| ?- file <- load,car.    %Step 1
>>  car  loaded.

yes
| ?- car <- new,civic.    %Step 2

yes
| ?- putvalue(civic,weight,90).
                  .                  %Step 3
                  .
yes
| ?- civic <- put,[power,83].

yes
| ?- civic <- pp.          %Step 4

**********civic object **********
     classname car
     >value set<
      value                power  83
      value                weight  90
      value                displacement  [1500,1300]
      value                price  100
      value                color  [blue,red,white]
 end of object
yes
| ?- select * from civic.   %Step 5
[value,power,83]
[value,weight,90]
[value,displacement,[1500,1300]]
[value,price,100]
[value,color,[blue,red,white]]
```

図3．KORE/KR とKORE/DB の実行例

ていることを示している．内部表現は以下のようなテーブルとして表現される．

civic	slot	facet	value
	value	power	83
	value	weight	90
	value	displacement	(1500,1300)
	value	price	100
	value	color	(bule,red,white)

5．おわりに

KOREのサブシステムにおける以上のような関係テーブルを用いた内部表現の統一化はKOREの強力な問題解決支援機能を実現する上で重要な働きを担うことになる．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行った．

参考文献

〔1〕 Kowalski,R. : Logic for problem Solving, Elsevier North Holland, 1977.pp.31-44.

〔2〕 Dasi,T. : On Database Systems Development Through Logic. ACM Transactions on Database Systems, Vol.7,No.1,March 1982.pp.102-123.

〔3〕 Forgy,C.L. : OPS5 User's Manual, CMU-CS-81-135,July 1981.

〔4〕 Bobrow,D.G. and Stefik,M.J. : The LOOPS Manual,XR-VLSI-81-13,1983.

〔5〕 London,P.E. : A dependency-based modelling mechanism for problem solving, AFIPS Conference Proceedings, Vol.47, 1978.pp.253-247.

〔6〕 新谷・片山：知識テーブル（その表現）―知識ベースにおける知識記憶管理方式とその応用―，情報31回全国大会，1P-8,1985.

〔7〕 新谷，他：問題解決環境KORE（その2）―知識記憶制御機構KORE/EDEN とその応用―，情報32回全国大会，5L-9,1985.

〔8〕 片山，他：問題解決環境KORE（その3）―関係表現サブシステムKORE/KR とその応用―，情報32回全国大会，5L-10,1985.

〔9〕 新谷，他：問題解決環境KORE（その4）―関係データベースサブシステムKORE/DB とその概要，情報32回全国大会，5L-11,1985.

問題解決支援環境ＫＯＲＥ（その２）

5L-9

—知識記憶利用機構KORE/EDEN とその応用—

新谷虎松・片山佳則・平石邦彦
（富士通㈱国際情報社会科学研究所）

1．はじめに

最近，知識情報処理研究において，知識獲得や知識表現と同様に効果的な知識利用の重要性が議論されている〔1〕．効果的な知識利用を実現するためには，知識間の関係を効率的に保存し利用する必要がある．よく知られている知識間の関係には，isa 関係，hasa（もしくはpartof）関係，data dependency 等がある．これら関係情報はネットワーク推論，暗黙値推論，信念の錯誤等を実現する上で重要な情報源であり，その形態として，一般に，ネットワーク構造を形成する．KORE/EDEN はこのようにネットワーク化された知識間の関係を効率的に保存し利用するためのネットワーク管理機能を提供する．

2．KORE/EDEN のネットワーク内部表現

KORE/EDEN は知識間の関係ネットワークを知識テーブル〔2，3〕を用いて保存する．知識テーブルの詳細は文献2．3で述べられているが，ここではその概要を紹介する．知識テーブルは関係ネットワークを（関係タイプをその要素とする）関係テーブルとして表現される．関係テーブル表現は素直にPrologで表現できる．関係タイプには(1)隣接関係を表す正整数１，(2)到達可能性を表す正整数２，(3)関係が無いことを表す整数０がある．例えば，図１で示すようなisa 関係ネットワークは，図２で示す知識テーブルにより表現される．

図２におけるテーブルはPrologシステムにおいては以下のようなアサーション群により表現される．

```
tableColumn( [living thing,animal,plant,cat,bird,
              dog,pigeon,sparrow,tree,bush,oak] ).
    table(animal, [1] ).
    table(plant, [1] ).
    table(cat, [6] ).
    table(bird, [6] ).
    table(dog, [6] ).
    table(pigeon, [ 266] ).
    table(sparrow [ 266] ).
    table(tree, [18] ).
    table(bush, [18] ).
    table(oak, [65570 ] ).
```

ここで，第二引数に現れるリスト内の数字は知識テーブルの列要素を２進数列として圧縮したものである．例えば，知識テーブルにおける第４行目のcat の列要素は＂２１０

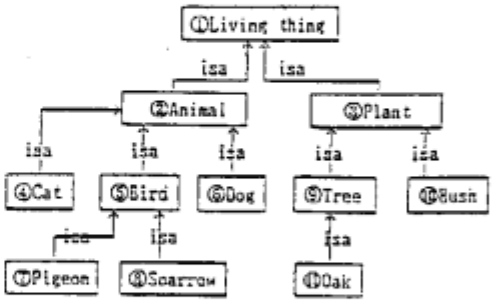


図１．isa 関係ネットワークの例

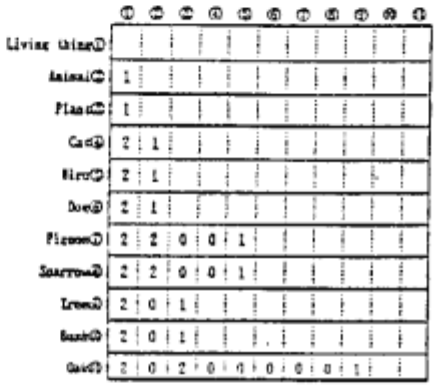


図２．知識テーブルの例

０００００００＂であり，これは２進数列＂0000000000000000000110＂で表すことができる．この２進数列は，第１列を２進数列の第１，２桁で表現し，第２列は第３，４桁で表現するといった繰り返しで得る．ここで，２進数列＂0000000000000000000110＂は十進数６で表すことができ，この十進数がアサーションの第二引数のリストの中に現れる．

3．KORE/EDEN の知識テーブル管理機能

KORE/EDEN の知識テーブル管理機能として以下のものがある．(1)ネットワーク編集機能，(2)ネットワーク検索機能，(3)知識テーブル表示機能，(4)知識テーブル圧縮機能，(5)ネットワークループチェック機能等が存在する．これら知識テーブル管理機能の詳細を図示すると図３のようになる．

(4)の知識テーブルの圧縮機能は，あるネットワークを保存するのに必要とされる知識テーブルの記憶容量を効率良く用いるために用いる．行列転置圧縮は知識テーブルの行列を転置することによりその圧縮機能を達成する（図４参

照），この行列転置圧縮は，主に，テーブルを表現するのに必要なアサーションの数を減らすために用いられる．

図３．KORE/EDEN の管理機能

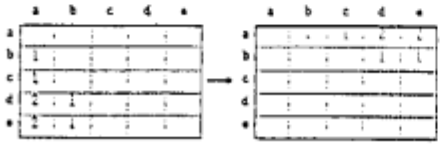


図４．行列転置圧縮の例

図４の例では，行列転置圧縮によりアサーションの数を２つ減らしたことに相当する．つまり，行列転置圧縮前では行ｂ，ｃ，ｄ，ｅに関するアサーションが必要であったが，その後では行ａとｂに関するアサーションを持てば良い．関係再配置圧縮は知識テーブルの行及び列における順番を入れ換えることによりその圧縮機能を達成する（図５参照）．この圧縮は知識テーブルそのものに必要な記憶スペースを削減する（これにより，アサーションの第二引数におけるリストの要素数を減らすことができる）．

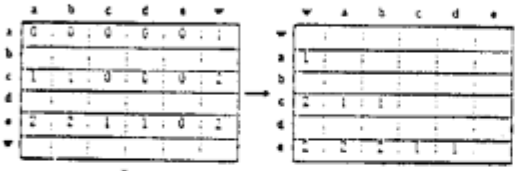


図５．関係再配置圧縮の例

(5)のネットワークループチェック機能は知識テーブルに関係を付加した場合にそこから得られるネットワークにループが有るかどうかをチェックする機能である．このループ発見過程は図６のように示すことができる．

図６は図右上のループを形成するネットワークを知識テーブルに保存して行く過程を示している．ここで，到達可能性セットとは到達可能性を親から伝播させたり，子供に伝播させることをいう．到達可能性の伝播は関係タイプ２を知識テーブル要素に立てることに相当する．これら伝播

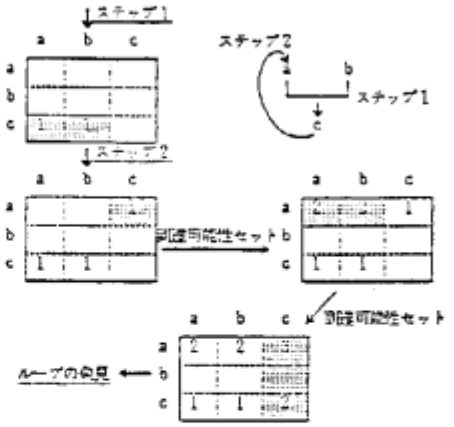


図６．ループ発見過程

操作のための操作（知識テーブル要素に２を立てること）はPrologにおける論理ビットORを用いて行う．そこで，このように自分自身に到達可能性がある知識の知識テーブルの列要素の中には整数３が生じる．なぜなら整数１（隣接関係タイプ）と整数２（到達可能性タイプ）の論理ビットORは整数３であるからである．つまり，このような整数３が生じるかどうかをチェックすることによりネットワークにおけるループ構造の発見を達成する．

４．KORE/EDEN の応用

問題解決において，一般にgenerate and test による問題解決法では，ある推論過程が失敗したら，最も最近の選択点までbacktrackingする必要があり，またそこから他の方法を試す必要がある．これは非常に能率が悪い．しかし，MOLGENで用いられているように推論過程（dependency records)を保存することにより，さらに効果的に失敗を修正することができる．この保存された推論過程に基づくback trackingは，dependency-directed backtrackingとよばれ，信念の維持等を実現するための有用な機能である．推論過程は，一般的に，ネットワーク構造を形成する．

KORE/ EDENはこのような構造を柔軟に管理ができ，暗黙値推論やtruth maintenance system等を構築する上で重要な機能を提供する．

５．おわりに

KORE/EDENはネットワーク構造を効果的に管理するために設計され，知識利用の側面において効果的に機能することが期待できる．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．

参考文献

(1) Hayes-Roth,F. : The Knowledge-Based Expert System: A Tutorial,COMPUTER,September,1984,pp.11-28.

(2) 長谷・片山：知識テーブル（その実現）一般言語ベースにおける推論記憶管理方式とその応用一，情報処理全国大会，1P-8，1985．

(3) Shirai,T. : Knowledge Table: An Approach to Knowledge Utilization Mechanism for Knowledge Base,Fujitsu IIAS-SIS Resaearch Report,1986,pp38. (in preparation)

問題解決支援環境KORE（その3）

-知識表現サブシステムKORE/KRとその概要-

5L-10

*片山 佳則　新谷 虎松　平石 邦彦
富士通㈱ 国際情報社会科学研究所

1. はじめに

汎用を目指した問題解決環境を実現するためには，その問題・状況・処理方法などに於ける知識が（機能的にも構造的にも）素直に表現されていなければならない．

このように知識を表現するための表現・利用機構としてオブジェクト表現と論理型表現を中心にしたシステムKORE/KR を開発している．これ自身は，単独な表現システムであるが，問題解決支援環境KOREにおいて，プログラミングパラダイムを提供する基本構成要素として利用可能である．

様々な知識を表現する場合には，その知識が持つ構造的な関係がシステム内で，どのように実現され，それがどのような意味を持つかなどの概念間の関係の意味を明らかにしなければならない．またどのような知識がどこに表現され，どのように利用できるかなどの対応が取れることも重要である．KORE/KR はこれらのための環境も備えている．

本稿では，この表現システムにおける概念間の関係を明らかにし，その機能の実現を中心として，表現・利用の環境について報告する．

2. KORE/KR における関係表現

知識表現を考える場合，その構造を表す為のネットワークとそれが持つセマンティクスが問題となる．システム側の構造に対して，人間が考えている概念を対応させることは容易なことではない．オブジェクト表現に限らず，他の表現においても，表現する知識の間の関係として様々なものを挙げることが出来るが，基本としてIS-A関係とPART-OF関係がある．これらは，一般には，包含関係と部分・全体の関係として考えられている．

KORE/KR でのオブジェクト表現では，概念は，全てクラスオブジェクトとして表現される．IS-A関係については，このクラス間（概念間）の階層関係表現として導入されている．

> KORE/KR におけるすべての関係は，概念の関係を表している．各概念の意味は，その結合関係の概念構造で規定される．

概念のレベルと実体のレベルに分けられるKORE/KR では，次のように基本的な結合関係を与えている．

(1) 階層関係を表す結合は，IS-A関係である．各概念が持つ厳密な継承方法によってその概念自体の意味が決まっている．

(2) 機能的関係を表す結合は，PART-OF 関係である．これにより，概念が持つ操作の受け渡し（分担）方法が決まり，概念自体の意味が表される．

これらの関係を用いて，知識を複数のクラス（概念）に分割して規定し，情報の隠匿などを利用し，矛盾無く表現する．階層関係を表す結合(1)には，多重表現が可能である．

これらの２つの関係は，(1)の結合の中では，満足のいく知識を見つけた時点で終了し，(2)では，結合されているすべてが対象となっていることから，KORE/KR では，IS-A関係がORであり，PART-OF 関係がAND と考えられる．

実際には，これらの意味は，その概念（クラス）が持つ個々の動作・実行内容により規定される．

概念レベルで構成したこれらの結合関係を利用する為には，概念に対応するオブジェクト（インスタンス）を実体レベルに持たなければならない．概念レベルと実体レベルの結合関係はINSTANCE-OF である．この関係によって実体レベルのオブジェクトが概念レベルで規定された機能・動作を利用出来る．包含関係など，全ては概念レベルで表されるため，INSTANCE-OF 関係に多重な関係は必要としない．従って概念レベルでの結合関係で規定された概念オブジェクトの構造が重要となる．

2-1 概念レベルの結合関係

IS-A関係が表す概念の階層によって，知識は必要に応じて継承される．継承の対象となる内容は，上位の概念が持つslot・method内の知識や，PART-OF 関係である．

上位を多重に持っていた場合は，検索の制限に従ってIS-A関係は初めに見つかった内容のみを受け継ぐ．複数の内容を受け継ぎたい場合には，原則としてPART-OF 関係を用いて表現しなければならない．

但し，PART-OF 関係を用いることで，概念間の意味的な対応が取れない場合には，各メソッドに対してIS-A関係での継承（検索）の制限を変更させることができる．

○ 継承（検索）の制限

・IS-A関係の定義によって表現される階層的ネットワークを left-to-right・depth-first で進められる．但し，各methodに対しての指定により次のように変更することができる．

① IS-A関係のネットワークで多重に表現された情報を全て継承する．この場合，継承結果の副作用は全てAND で処理される．

② left-to-right・right-to-left の切り換えをする．

これらの方法は，Flavorのconvine-method[2] を参考にしているが，methodを実行する際にFlavorが持つdemon-methodの働きは持たない．

PART-OF 関係を引き継いだ実体オブジェクトに対してpartに対応するすべての実体オブジェクトが必ずシステム側で生成される．

partとして生成された実体オブジェクトは，全体であるオブジェクトからの機能分担として動作し，他から直接メッセージを受け取ることは出来ない．

2-2 結合関係の実現

オブジェクト間の関係表現は，概念レベルで行われているため，これらのオブジェクトをどう実現するかによってほとんどが決まる．概念レベルのオブジェクトは次のように定義される．

```
defclass   <オブジェクト名>  ::
   super-set <IS-A階層の上位オブジェクト集合>  ;
   parts     <PART関係のオブジェクト表現>  ;
   value-set  < スロット定義>  ;
   attach-method    < 付加手続きの記述>  ;
   method           < メソッドの定義>  .
```

super-set で，IS-A階層の直属の上位オブジェクトをすべて次の例のように記述する．

```
super-set    OBJECT,WINDOW,CHECK ;
```

(英大文字はオブジェクトを表す)

KORE/KR での特徴として，PART-OF 関係の記述は，partsに，各部分をどのオブジェクトで分担するかをペアにする．

```
parts    right-wing  -  WING,
         left-wing   -  WING ;
```

parts を定義したクラスオブジェクト内の表現（メソッド・付加手続きなど）では，parts で定義された各部分のオブジェクト(right-wing,left-wing)を，他のオブジェクトと同様に扱うことが出来る．各部分をどのように利用（機能分担）するかは，parts 定義をしたオブジェクト内で定める．従って各部分への操作は，parts 定義のオブジェクトを通してしか行えない．オブジェクトのスロット・メソッド・付加手続きなども所定の方法で記述する．

3．表現・利用の環境

前回，いくつかの機能を示した[1] が，その他KORE/KRの利用に関していくつか取り上げる．

a) オブジェクト定義で示したようにスロットに付加手続きの記述を行える(attach-method)．メカニズムは，外部からのメッセージによりメソッドが起動されるのと同様，オブジェクト内のスロットから付加手続きの起動がメッセージ送信の要領で実行される．これは，partへの操作と同様にオブジェクト内でのメッセージ送信と考えられる．

b) すべてのオブジェクトは，外部ファイルとして予め作成しておき，file-manipulatorオブジェクトにメッセージを送ることで変換される．その際に副作用としての付加手続きの起動は，様々なオブジェクトへ影響を与える．そのためfile-manipulatorでは，全オブジェクトの変換・作成が終了してから，付加手続きが起動される．これによって未定義オブジェクトへの副作用などのエラーを無くす．

c) KORE/KR は，method-managerオブジェクトが組み込まれており，これによってメソッドに関する情報の管理を行っている．情報はいつでもメッセージにより利用できる．

▷メソッドがどのオブジェクトにあり，どのオブジェクトから利用出来るか．

▷オブジェクトはどのようなメソッドを持ち，その他にどのようなメソッドを利用できるか．

これらの情報により，オブジェクトの概念関係を利用して，メッセージによる実行を円滑に進めることが出来るようになる． method-manager は，file-manipulatorと連結し，メッセージを受けることで各メソッドの状態を記録していく．

現在,C-Prolog 上のKORE/KR がシステムとして持っているオブジェクト集合は次の５つに分類される．

(1)＜META＞：クラスオブジェクト全体を統轄する役割．

(2)＜UTILITY ＞：クラスオブジェクトの階層の中で，上位レベルにあり，様々なユーティリティの機能を持つ．

(3)＜LOAD＞：外部からのオブジェクトの入力を行い，変換をする．

(4)＜METHOD＞：methodの情報を記録・管理する．

(5)＜WINDOW＞：windowシステムの働き．

これらがシステム起動時に組み込みとして入力される．(4)と(5)に対しては，必要に応じて入力を選択可能．

4．おわりに

概念間の関係を明らかにし，KORE/KR の利用環境についてその概略を述べた．詳しい応用まで述べていないが，このシステムは幾つかのProlog上に実装されている．現在，PART-OF 関係を用いた構造の記述を取り上げている．

また，付加手続きを主に利用する表現は副作用としての複雑さを導くことになるため，注意が必要である．

尚，本研究は，第五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．

〔参考文献〕

(1) 片山，新谷 "知識テーブル（その利用）ー知識ベースにおける対象指向表現とその処理機構の試作ー"，第31回情報処理全国大会，(1985) p.1233-1234

(2) D.Weinreb,D.Moon " Lisp Machine Manual " 4th ed., Symbolics, Inc. (1981).

問題解決支援環境ＫＯＲＥ（その４）

－関係データベース・サブシステムKORE／DBとその概要－

5L-11

°平石 邦彦　新谷 虎松　片山 佳則
富士通㈱ 国際情報社会科学研究所

1．はじめに

論理型プログラミング言語であるprologは関係データベースの表現に適しており，それ自体で高度なデータベース問い合わせ言語として見ることができる．本発表ではprologのこのような特徴を利用した関係型データベース操作言語について報告する．

2．prologと関係データベース

関係モデルでは，有限個の順序付けられたデータの組の集合を関係(relation)といい，この関係によりデータベースを表現する．また，各々のデータの組のことをタプル(tuple) と呼ぶ．

まず次のような関係suppliers を考える．

suppliers [sno,sname,status,city] =
{(s1,smith,20,london),(s2,jones,10,paris), ･･･}

ここで，sno,sname,status,city は属性名をあらわしている．この関係はprologの事実(fact)により次のように表現できる［1］．

```
db_suppliers(s1,smith,20,london).
db_suppliers(s2,jones,10,paris).
        :
```

このとき検索は次のように行うことができる．

質問：statusが20より大きくlondonにいるsuppliersのsno は？

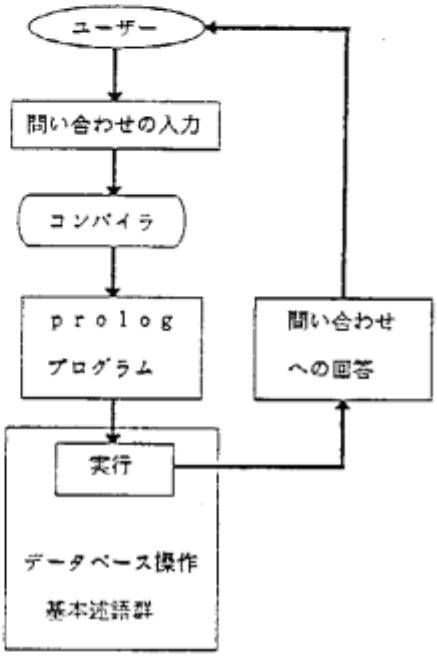
```
| ?- db_suppliers(X,_,Z,london),Z>20.
```

このように，prolog自体でデータベース言語としての機能を持っている．

3．システムの構成

データベースへの問い合わせにおいて，prologそのままではユーザーにとって使いやすいものではない．そこで，本システムではprolog上にＳＱＬ［2］［3］に似た文法を持つデータベース言語をのせ，それをprologプログラムに翻訳して実行するという形式をとっている．したがってユーザーはprologを意識することなくデータベースを操作することができる．

構成はFig.1 のようになっている．コンパイラは，問い合わせのprologへの変換および述語の並び換えによる最適化を行う．データベース操作基本述語群は，タプルの追加・削除・更新など基本的なデータの管理を行う．データベースに対する操作は，すべてこの述語群を通して行われる．



Ｆｉｇ．１　ＫＯＲＥ／ＤＢの構成

4．問い合わせの変換

次の問い合わせを考える．

```
select sno from suppliers
 where city=london
   and status >
         (select status from suppliers
                 where sname=smith)
   and sno in
         (select sno from shipment
                 where qty>300).
```

これは次のようなprologプログラムに変換される．

```
ans(A) :-
        db_suppliers(_,smith,C,_),
        db_suppliers(A,_,D,london),
        db_shipment(A,_,B),B>300,D>C.
```

selectブロックにおけるwhere 以下の条件は連言形をしているものとする．このとき各条件を次のように分類する．

- 等号条件(=,in)　sname=smith，city=london，sno in (select …)
- 非等号条件
  - 独立条件　1つの関係の属性についての条件．qty>300
  - 関連条件　複数の関係の属性を含む条件．status>(select …)

各属性値の取り出しは，述語db_関係名（…）によって行われる．等号条件はdb_関係名（…）の変数に統合化される．独立条件は，各関係の値を取り出す述語の直後に置かれる．関連条件は，すべての属性の値を取り出した後に置かれる．

5．問い合わせの最適化

prologが問い合わせを評価する際の効率は，連言中に言われた目標(goal)の順序に依存する．したがって，問い合わせの目標の順序を入れ換えることにより，問い合わせの最適化を行うことができる．ここでは，述語db_関係名（…）に対し次のような価格の式を導入している［3］．目標となる述語に対応する関係の要素数を，値を代入された引数の定義域の大きさの積で割ることにより価格が求まると仮定する．

$$COST = \frac{|R|}{|D1| \times |D2| \times \cdots \times |Dn|}$$

ここで，|R|：関係のタプル数

|D|：引数がとる定義域Dの大きさ

である．この式によりdb_関係名（…）の順序を動的に入れ換える．その手順は以下の通りである．

① 各db_関係名（…）の価格を計算し，最小のものを選び出す．

② ①で選んだdb_関係名（…）の変数すべてに値を代入し，残りの目標について再びこの手順を繰り返す．

次に評価すべき目標は，その時点での最小価格を持った目標となる．また，1つの目標の選択は残りの目標の選択に影響を与えることになる．

6．規則の利用

prologは事実だけでなく規則(rule)も扱うことができる．ここでは規則を用いることにより実現されている機能について説明する．

(1) ビュー(view)

ビューは関係型データベース内の関係に対して必要な属性やタプルの集合を定義した仮想的な関係である．これはprologの規則を用いることにより簡単に実現できる．たとえば2つの関係

suppliers [sno,sname,status,city]

shipment [sno,pno,qty]

からsupl [sno,sname,pno] という仮想的な関係を定義するには，

```
db_supl(A,B,C) :- db_suppliers(A,B,_,_),
                  db_shipment(A,C,_).
```

とすればよい．これはdefine_view コマンドにより実現される．suplはsuppliers とshipmentの等結合になっており，検索に関しては通常の関係と全く同様に扱うことができる．また，

```
db_a(X,Y,Z) :- db_b(X,Y),Z is (X+Y)/2.
```

のように，値を計算させることも可能である．

(2) 手続きの実行

ビューの場合と同様な方法により，あるタプルを検索したときに，自動的に手続きを実行させることができる．たとえば，

```
db_suppliers(s1,smith,20,london) :-
                    method(s1,london).
```

とすれば，このタプルを検索したときに，手続き method(s1,london) が実行される．

7．おわりに

ＫＯＲＥ／ＤＢはprologをデータベースとして利用する際のユーザー・インターフェースとして設計されており，検索等の処理はすべてprolog自身が持つ機能を使っている．したがって，データベースとしての能力もprolog処理系に依存する．

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．

〔参考文献〕

［1］R.Kowalski,"LOGIC FOR DATA DESCRIPTION," In H.Gallaire(ed.),"LOGIC AND DATA BASES," PLENUM PRESS, New York(1978).

［2］M.M.Astrahan,"System R:Relational Approach to Database," ACM Transactions on Database Systems, Vol.1,No.2,pp97-137(1976).

［3］D．リー，"PROLOGデータベース・システム"，近代科学社(1985)．

［4］M.M.Astrahan,"Implementation of a Structured English Query Language," Communications of ACM, Vol.18,No.10,pp580-589(1975).

5U-4

方式設計段階における性能評価支援の一検討

岩田 悦之　野田 泰徳　桜井 昌俊　平野 建郎
沖電気工業（株）

1．はじめに

論理装置の設計において、方式設計の段階で実機を作る前の正確な性能評価を行う事が、品質向上、設計期間短縮の面から望まれている。本論では、パイプライン処理方式の計算機をターゲットとし、その命令仕様、データパス構成、データパス構成上での命令仕様の実現方式（データ転送時）が決定した設計段階において、それらを用いて当該装置の性能を評価する為のデータを導出し、性能を向上させる為の改善案を見出す際の支援を行うツールについて検討した。更にパイプライン処理方式のモデル化について並列処理記言語を用いて検討した。

2．システム概要

パイプライン処理方式計算機の方式設計段階における性能評価項目の一つとしては、命令実行の並列度が考えられる。並列度の高い処理方式はハードウェアファシリティを有効に活用した、パフォーマンスの高い計算機を実現出来る可能性を持っている。その命令実行の並列度から計算機を評価し、性能を向上させる為の修正点を取り込む為には、図1の様に各命令の各ステージが並列に実行されている状態（図1）を調べ、レジスタの利用に競合が起るクリティカルなデータパスを発見する事が有効であると考えられる。これを基に命令の実行待ちを最小にする修正を命令仕様やデータパス構成に加える事によって性能の向上を図る事が出来る。

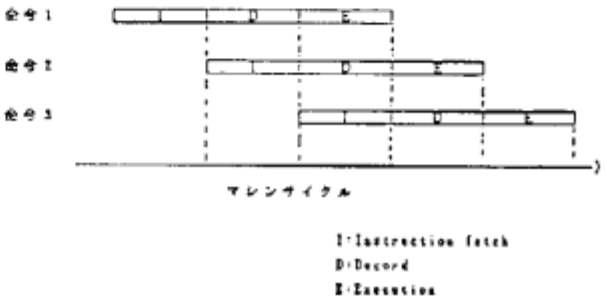


図1　パイプライン処理方式の並列実行

本システムでは、上記の評価と性能の向上を図る際の支援を行うために、入力として、テストプログラム、データパス構成、各命令の各ステージでデータパスを構成するファシリティをどの様に用いるかを記述した命令定義記述を用い、データの流れのシミュレーションを行う。そして、各命令の各ステージが並列に実行されている状態（実行制御データ）と各ハードウェアファシリティにおいて利用競合が起きる状態（競合データ）とを表すタイムチャート、Mips値、テストプログラム実行時の並列度、更に専門家の知識を組み込んだエキスパートシステムによってコンサルテーションされる設計変更の指針等が出力される。

3．システム構成

システムは図2の構成となる。シミュレータ部は入力データを基にシミュレーションを行い、リアルタイムでトレースデータをタイムチャート形式でコンソール等に出力する。post_processorは、シミュレーション終了後にMips値、命令実行の並列度等の統計データを出力する。adviserは設計変更の指針を示す知識処理モジュールである。

4．並列論理型言語によるモデル化

4．1 GHC

GHC(1)は並列に存在するプロセスを動的に生成し、各プロセス間で通信や同期の制御を行う事が出来る。これにより並列実行するパイプラインの動作のモデル化が容易であると考えられる。更に論理型言語であるため、知識を直接表現することが可能である。以上よりパイプライン処理方式のモデル化にGHCを用いた。

4．2 シミュレータの構成

現在試作中のシミュレータ部の構成を図3に示す。命令実行モジュールは、命令定義記述に従って命令を実行し各ステージに於て利用するファシリティをリストにしてデータパス管理モジュールに渡し、実行終了メッセージの到着を待つ。データパス管理モジュールは、受け取ったリストに従ってファシリティの利用競合を解消しながら、データパスのシミュレーションを行い、それが終了すると実行終了メッセージを返す。

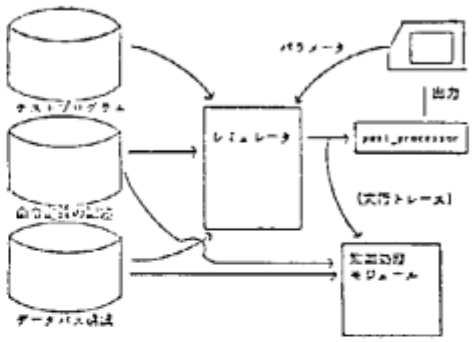


図2 システム構成

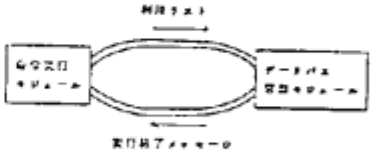


図3 シミュレータ部の構成

4．3 命令実行モジュールのプロセス構成

命令実行モジュールのプロセス構成を図4に示す。stage_processは命令を実行して行く主体、message_managerは各stage_process間のメッセージの管理、timer_processは各stage_processの同期を管理するプロセスである。図は3段パイプラインの例である。これをGHCで記述したのが図5である。シミュレータの起動は述語pipelineの呼び出しによって行う。起動されると、body部に示された5つのプロセスが同時に生成され、処理を行なう。

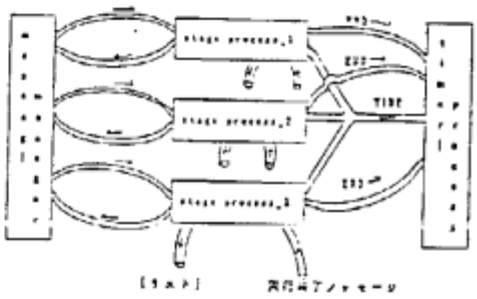


図4 命令実行モジュールのプロセス構成

```
pipeline([list(A,B,C),end(X,Y,Z)]):-true |
        timer(O,T,P1,P2,P3),
        message_manager(in(U,V,W),out(P,Q,R)),
        stage(1,[O|T],P1,U,[[]|P],A,Z),
        stage(2,[O|T],P2,V,[[]|Q],B,Y),
        stage(3,[O|T],P3,W,[[]|R],C,X).
```

図5 命令実行モジュールの記述例

4．4 評価

パイプライン方式の並列実行する各ステージをGHCのプロセスに割り当て、プロセス間の同期をGHCの同期機構を用いて容易にモデル化することができた。

5．まとめ

本システムは複雑な並列動作を行うパイプライン方式計算機の制御部とデータパス機構の設計を支援する事を目標としている。現在シミュレータ部のプロトタイピングを行っている段階であり、他の部分の詳細決定等は今後の課題である。

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行われている。
御検討頂いたICOT第一研究室古川室長及び同研究室諸氏に感謝する。

〈参考文献〉

*1 Kazunori Ueda "Guarded Horn Clause". ICOT Technical Report. TR-103(1985)

知識表現／推論システムPeaceとその故障診断問題への適用

4L-9

古関 義幸* 和田 慎一* 西田 哲朗** 若杉 暢爾*** 近藤 真己***

* 日本電気㈱ C&Cシステム研究所，**同 複合交換開発本部，

*** 日本電気技術情報システム開発㈱

1. はじめに

電子装置の社会的役割は年々高まっており、例えば電子交換機では、障害発生時にはサービスに影響を与えずに速やかに復帰することが要求されている。このために、局用ディジタル式電子交換機は、オンライン障害検出機能、装置の二重化、及び自己診断機能を備えている。しかし、その自己診断の能力には限度があり、診断できない障害が発生した場合には専門家による判断が必要である。また、他の電子装置に比べて寿命が長く、保守の専門家を長期に確保することが困難である。このような背景の中で、保守の専門家と同等の知識を持ち、その代わりとなるシステムの開発が望まれる。本稿では、現在研究開発を進めている電子交換機の故障診断エキスパートシステムについて、その基本となる知識プログラミングシステムを中心に述べる。

2. 故障診断における専門家の知識

電子交換機のような大規模なシステムの障害を完全に診断可能な診断プログラムの作成は困難な問題である。例えば論理ゲートのみからなる組み合わせ回路においてでさえ、障害を構成部品の単一縮退故障に仮定したとしても、完全な故障診断を行なう診断プログラムを作成する問題は複雑な組み合わせ問題となることが知られている。その上、実際のシステムははるかに大規模で複雑な順序回路であり、障害の種類もショート故障、開欠故障等、種々であり、かつ装置外部の状況も障害に影響する。

このように困難な問題に対して、人間は、診断プログラムでは診断できなかったような障害に対しても、障害時の状況、障害の出方、部品の壊れ易さ、回路構成、等を柔軟に考慮しながら種々のテストを行ない、故障箇所を推論していき、最終的には障害を取り除く能力を持つ。このような能力を持った保守の専門家の知識を計算機上に表現し同様の判断を行なうシステムを実現するために、専門家のインタビューを通して診断の実際を調査した。

専門家は、まず、①症状、障害時の状況等の情報を集め、被疑部分を思い浮かべる。次に、②その被疑部分の確定あるいは切り分けのために有効な種々のテスト方法を考え、実行する。その時に、部品の壊れ易さ、テストのサービスへの影響も考慮に入れている。そして、③そのテストの結果をもとに被疑部分の更新を行なう。②、③を、交換可能単位までに切り分けが出来るかあるいはそれ以上切り分けが出来なくなるまで繰り返し行なう。そして、④その被疑範囲の部品の交換を行なう。最後に、⑤その交換により故障が完治したかどうか確認し、⑥完治しない場合は考え直す。このように人間は種々のテストを行ないながら被疑範囲を狭めていき、最終的には部品の交換によって診断を行なっているといえる。このような診断の方法は一般の電子装置について共通であるといえる。

このような診断をするシステムの実現には、従来の医療診断エキスパートシステムのように、症状と原因をルールで記述しただけの知識では不十分である。①の症状についての知識は各症状をオブジェクトとした表現形式が適している。②のテスト選択の知識は、条件部に被疑部分の分布を、動作部に選択するテストを記述した前向き推論ルールが適している。また、③のテスト結果判断の知識にも、条件部にテスト結果を、動作部に被疑部分を記述した前向き推論ルールが適している。④で使用する部品の知識は各部品、装置をオブジェクトとしたネットワーク構造の表現形式が必要である。

3. 知識プログラミングシステムPeace の概要

多様な知識を表現し利用するための知識プログラミングシステムとして、Prolog言語上に Peace (Prolog based Expert AppliCations Environment)

を開発中であり、これを用いて診断知識ベースを試作している。これは、Prolog言語上に、オブジェクト指向の表現、前向き推論プロダクションルールの記述、マンマシンインタフェース作成用のライブラリを実現したものである。Prolog言語上にその自然な拡張として実現してあり、Prolog言語の持つ利点をそのまま利用しながら、これらの表現形式を混在して用いることができる。

3．1　オブジェクト指向の表現

オブジェクトは、一つの概念、事物等に関する性質をPrologの節の集合で表わす。これは他のProlog上のオブジェクト指向表現〔中島84、近山84〕と基本的に同様である。但し、

・　オブジェクト間の任意の関係及びそれを通したマルチプルインヘリタンスの定義を、関係を表わすオブジェクトを使用して自由に行なえる。クラス、インスタンス、メタクラス等の関係も一般的な関係の一つとして扱う。

・　オブジェクト変数及びメソッドの記述をオブジェクト内に定義された一引数の節及びボディ部を持った節に対応させており、Prolog言語との親和性がよい。メソッド起動はそのオブジェクト内に定義されたボディ部を持つ述語の呼び出しに等しい。

・　逆関係の更新を自動的に行なう。

等の特徴を有している。例えば交換機のパッケージについての知識を以下のように記述する事が出来る。

```
equipment_A : category(category_1).
package_of: $inherit(category).
package_1 : a_kind_of # package;
            package_of # equipment_A;
            package_No(pkg001).
package_2 : a_kind_of # package;
            package_of # equipment_A;
            package_No(pkg002).
package : disp_No :- package_No(X) in origin,
                     display(X),nl.
```

ここで例えば、categoryがcategory_1であるようなパッケージを全て求める処理は、Prologの組み込み述語 setofを用いて、

```
?-setof(X,(has_kind # X in package,
           category(category_1) in X), S).
S = [package_1, package_2]
```

のように求まる。ここで、述語inがオブジェクト内の節を操作する述語で、参照、メソッド起動等を行なう。これは、Prologで書かれたPrologインタプリタであるdemo述語〔国藤85〕を拡張して実現された。関係has_kindは関係a_kind_of の逆関係であり、システムにより自動的に作成されたものである。　関係package_ofを通して、オブジェクトequipment_Aのcategory(category_1)という性質がオブジェクトpackage_1 とpackage_2 に継承されている。　関係pacakge_ofはpackage_ofというオブジェクトに定義されており、$inherit(category)により、categoryという述語を継承するように定義されている。

3．2　前向き推論プロダクションルールの記述

前向き推論プロダクションルールは、同時に適用する複数のルールをまとめてルールブロックと呼ぶオブジェクトに記述する。このため、他のオブジェクトと同様に管理でき、また、効率のよい推論が可能になる。ルールの条件部及び動作部はPrologのゴール呼び出しの形式で記述する。例えば②のテスト選択の知識を表わすブロックは、

```
test_selection_rules :
  $control (do_1) ;
  rule1  if   test1 を選択する条件
         then test1 を実行;          .....
```

のように記述できる。ここで、$controlはこのブロック内でのルール実行の制御形態を表わす。

4．おわりに

電子交換機の故障診断における専門家の知識を整理し、その表現形式を考案した。Prolog言語上に、知識プログラミングシステム、及び知識ベースを試作中であり、今後その有効性を確かめていく。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進めているものであり、日頃御指導頂くICOT岩下室長に深謝します。

〔文献〕
〔中島84〕Prolog/KR における知識表現、第28回情処全大
〔近山84〕ESP Reference Manual, ICOT TR-044
〔国藤85〕論理型言語Prologによる知識ﾍﾞｰｽの管理,Logic Prog.Conf85

仮定の修正を行なう推論方式の故障診断への応用

4L－10

和田 慎一* 古関 義幸* 若杉 暢甫**
* 日本電気㈱ C＆Cシステム研究所
**日本電気技術情報システム開発㈱

1．はじめに

複雑な電子装置の保守の支援の必要性から、筆者らは電子装置の故障診断を行なうエキスパートシステムの研究・開発を行なっている[1]．故障診断では症状や装置に対するテスト実行の結果などの情報をもとに故障の原因についての判断が行なわれるが、「大体・・・だろう」という判断も行なわれる．通常はこのような不確かな判断も合わせて推論を進め、誤りがあるとわかったときにそれを修正するという方法がとられる．このような形の推論をエキスパートシステムとして実現するために、通常正しいと思われることを仮定して推論を進め、矛盾の検出された場合に誤った仮定を修正する推論方式の利用が考えられる．本稿ではこのような推論方式の故障診断における必要性、適用の方法、および矛盾が検出された際の仮定の制御の方法について述べる．

2．故障診断における仮定に基づく推論の利用

2．1 故障診断における不確かな判断

故障診断は、①テスト（故障に関するデータ・情報の収集のためのアクション）を選択する、②テストを実行する、③テスト結果として得られた情報をもとに故障原因として考えられる範囲について判断する、という処理の繰り返しとして進められる．この中で③の判断には種々の不確かさが含まれる．

有効なテストの数が限られている故障診断においては、得られたテスト結果のデータをもとにして故障原因として考えられる範囲をより狭く絞り込むために、不確かさを含む判断も合わせて用いる．判断に不確かさが含まれているとしても絞り込まれた範囲に該当する部品を交換して故障が直ればそれで診断を終了することができる．

2．2 不確かさを含む判断の例

不確かさを含む判断の例を挙げる．図1のような構成の装置においてC1, C2, C3が同一な構成の部品であり、A-Ci(i=1,2,3)間では定常的に信号の伝達が行われているものとする．A-Ci間の信号の伝達について異常検出の機能があるが、異常が存在するとともにタイミングや実行状況等の確率的条件が満たされた場合に限って異常が検出されるものとする．このような装置において、A-C3間のみで異常が検出され、A-C1, A-C2間では異常が検出されていない状況を考える．異常に関係している部分Aに関しては「Aが故障の原因であるならばおそらくA-C1, A-C2間でも同様に異常が検出されることになるはず」と考えられるが、これは対象としている状況と合わない．このことから「Aは故障の原因ではない」という不確かな判断が行われる．

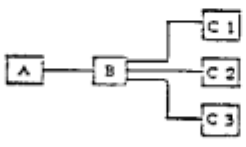


図1 装置の構成

2．3 仮定に基づく推論の利用

不確かさを含む推論を行なう以上、故障原因として考えられる部品を交換しても故障が直らないことがある．その場合への対処のためにdata dependency[2]あるいはtruth maintenance[3]の利用が考えられる．その利用は次のように行なう．

①判断において不確かなことを仮定として表わし、通常はそれを信じ、それに基づいて推論を進める．

②推論実行の履歴を記録しておく．これにより、導かれた各事実がどのような仮定に基づくのか（dependency）を調べられる．また仮定がくつがえされた際にそれに基づいて導かれた事実を消去できる．

③故障を起こしていると考えられる部品を交換しても故障が直らない場合や事実間に起こり得ない関係が生じた場合を矛盾とする．

④矛盾が検出された場合、前提となっている仮定を疑い、くつがえす（dependency-directed backtracking）．これによって矛盾が解消される．

2．4 結果を予想できるテストの実行の省略

仮定に基づく推論を行なうことによって、さらにテスト実行の制御に関して次のような処理を実現することができる．

①実行のコストが大きいテストのうち、その結果について予想できるものがある．通常時はテストの実行を省略し、代わりに予想されることを仮定として信じ、それに基づいて推論を進める．

②矛盾によってその仮定が疑わしくなった時に省略

していたテストを実行し、仮定の真偽を確認する。

2．5 確率的推論に基づく処理との比較

故障診断における不確かさに対処するための方法として、確率的推論を用いて各部分あるいは各部品ごとに疑わしさを表わす値を評価し、その値の大きな部分から交換していく方法も考えられる。しかし、この方法では、推論の不確かな部分に戻ってその確認のテストを行なうこと、検出された矛盾に応じて、事実関係の一貫性を保ったうえで事実を修正することなどが困難である。

3．仮定の制御

3．1 仮定の制御の方針

仮定に基づく推論を採用する場合、検出された矛盾を解消するために、矛盾の前提となっている仮定のどれかをくつがえすことになるが、どの仮定をくつがえすかを決定しなければならない。また、ある仮定をくつがえすことにより、くつがえされていた別の仮定を再び信じることができる（信じても矛盾を生じさせない）場合がある。適用する問題の性質から仮定の制御の方針としては、次の2つに基づくものとする。

①より多くのより確度の大きな仮定を信じる

矛盾が検出された場合でも、互いに独立な仮定を矛盾を含まない範囲でより多く信じることによって、不確かながら有効であるような推論規則がより多く用いられるように保つ必要がある。このため、あらかじめ各仮定に確度を表わす数値を与えておき、矛盾が検出されたときには、矛盾を含まない範囲で、できる限りより確度の大きな仮定をより多く信じるように保つ、つまり確度の和が最大となるようにするものとする。

②疑わしい仮定の真偽について調べるテストの実行

テストの省略（2．4で説明）のために代わりに立てられた仮定が検出された矛盾の原因として疑わしくなった場合、省略していたテストを実行して仮定の真偽を調べ、その結果を利用する。

3．2 仮定の制御の実現法

仮定の制御の方針①の実現法として、単に「矛盾の前提となっている仮定の中で確度の最小のものをくつがえす」という方法が考えられるが、その方法は十分なものでない。このために、同時に信じ得る仮定の最大限の組み合わせ（矛盾を含まない仮定の組み合わせ）の集合を保持し、矛盾が検出された際にはその前提となっている仮定の組み合わせに応じてその集合を更新するという方法をとる。この集合をＣＢＳ（Consistent Belief Set）と呼ぶ。そして、常にその中で確度の和が最大になるような仮定の組み合わせにしたがって仮定を信じるようにする。

この処理の例を図2に示す。推論実行中にSTEP1で仮定A,Bの間で、STEP2で仮定A,Cの間で矛盾が検出された場合の処理が示されている。特にSTEP2において矛盾の前提となっている仮定A,Cのうち確度の小さいのは仮定Cであるが、仮定Aをくつがえし、仮定Bを再び信じることにより、確度の和が最大になるように保たれている。

実行のＳＴＥＰ	<STEP0>	<STEP1>	<STEP2>
矛盾の前提となっている仮定	－	A,B	A,C
ＣＢＳ	{{A,B,C}}	{{A,C},{B,C}}	{{B,C},{A}}
ＣＢＳで確度の和の最も大きくなる仮定の組み合わせ	{A,B,C}	{A,C}	{B,C}
くつがえす仮定／信じる仮定	－／－	B／－	A／B

各仮定の確度は　A：12，B：11，C：10

図2　仮定の制御の実行例

4．おわりに

仮定に基づく推論の故障診断における必要性、適用の方法、そのための仮定の制御の方法について述べた。この方法を用いることにより、不確かなことが疑わしくなったときに、それに関するテストを行わせるという処理が実現できる。現在、Prologを用いて前向きプロダクションシステムおよびその上の知識ベースを試作しているが、処理の効率などに課題がある。また、このような推論の応用として、ユーザの推測する内容を一つの仮定として事実間の関係の一貫性を保った形で受け入れ、利用する、などの可能性も考えられるが、それらについて今後検討していきたい。

なお本研究は第５世代コンピュータプロジェクトの一環として進めているものである。

【参考文献】

[1] 古関他，"知識表現／推論システムPeaceの故障診断問題への適用"，情報処理学会第32回全国大会講演論文集4L-9（昭和61年）1121-1122

[2]Stallman,R.M. and Sussman,G.J., "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis",Artificial Intelligence 9(1977)135-196

[3]Doyle,J.,"A truth maintenance system",Artificial Intelligence 12(1979)231-272

オンラインKWIC検索システムにおける分かち書き処理

9T-4

岡　夏樹　　重永信一　　玉越靖司
（新世代コンピュータ技術開発機構）　（松下電器産業株式会社）

1．はじめに

我々は，実文例中での言葉の使われ方を容易に収集することができ，自然言語処理研究に有効なオンラインKWIC（Key Word In Context)検索システムの開発を行ってきた．

本システムは，与えられたキーワードの検索をオンラインで効率よく行うために，文中に現れる単語の索引ファイルを前もって作成しておく方法をとっている．このための単語抽出は，漢字かな混じりのべた書き文から自動的にバッチ処理で行う．ここでは，この単語抽出用の分かち書き処理について報告する．

2．これまでの研究との比較

これまでに報告されている分かち書きの処理法は，かな漢字変換に関連した，かな入力を対象としたものが多い．いずれも最長一致法を用いた形態素解析を基本にし，２文節最長一致法［１］，文節数最小法［２］，語の出現頻度に基づく方法［３］，助詞に着目する方法［４］等のようにそれぞれ精度を上げる試みがなされている．

これに対し，漢字かな混じり入力では，簡単な方法でもかなりの精度が得られると予想されるので単純な最長一致法を採用した．ただし漢字かな混じり入力の場合は表記のゆれに対処する必要がある．

文献［５］は，字種の変り目に着目して，小さな辞書だけで解析することを提案しているが，これは精度の向上に限界があり，例外の処理が増えてかえって複雑になる恐れがあるので，本システムでは大きな辞書を使用することにした．

従来，文節内の接続可能性と文節を構成する条件とを考慮して解析を進める方法がとられてきた．これに対し今回報告する方式の特徴は，それらを手続き上区別せずまとめて隣接可能性としてとらえ，一貫して辞書と隣接表により処理を進めることで，これにより非常に簡明なプログラムを実現した．

3．処理の概要

本処理は漢字かな混じり文（かな書き文も可能）の単語分かち書きを行い，合わせて各単語の品詞も決定する．

単語辞書および隣接表を用いた形態素解析のレベルで正しいと判断される結果のうち，最初に見つかったものを出力する．

4．処理アルゴリズム

以下のように総当り探索を行う．

１）解析点からの入力文字列と一致する単語すべてを辞書から探して，単語候補リストをつくる．ただし英数字等は字種切りにより単語候補とする．

２）付属語・活用語を優先した最長一致法による優先順位に従い，隣接表を用いて直前の単語との隣接検定を行う．

　隣接可であれば，次の解析点にすすむ．

　隣接不可であれば，次の候補について隣接検定する．

　すべての単語候補について隣接不可であれば，一つ前の解析点に戻る．

　解析開始点または解析確定点まで戻ってもだめだったら，解析済みの部分の次の文字を未登録語とする．

３）一つ解を見つけたら終了する．

5．辞書および辞書引き

かな表記および標準的な漢字かな混じり表記を見出しとし，送りがなのゆれにも見出しを追加する方針で対処した．

辞書引きの処理時間の観点などから辞書は次の３種類に分けた．

・ひらがな１文字表記の自立語辞書（約３００語）
・上記以外の自立語辞書（約３００００語（表記だけが異なる語は含まない））
・付属語辞書（形式名詞・補助動詞などを含む）（約２５０語）

辞書引きは索引表・索引木を使って高速に行っている．

6．隣接表および隣接検定

接続，係り受けなどの関係の有無によらず，２つの単語（記号類なども含む）が隣接しうるかどうか（隣接可能性）のみに着目し，これを１枚の表の形式にまとめたものを隣接表と呼ぶ．

隣接可能であるのは次の場合である．

１）文節内接続可能（自立語－付属語間，付属語－付属語間）
２）複合語可能（自立語－自立語間，接頭語，接尾語，助数詞など）
３）前の単語が文節の末尾に，後ろの単語が文節の先頭になりうる
４）記号類（句読点，括弧，単位記号，文の先頭，文の末尾など）との隣接

7．性能評価

性能評価用入力文として下記の２冊の本からそれぞれ１００文ずつを使用した．

・「日本－その姿と心－」＜第２版＞，新日本製鉄株式会社能力開発部，学生社，1984
・「英語構文の研究」，高梨健吉，英誠社，1975

精度（正しく分かち書きされ，かつ，正しく品詞が与えられた単語数／総単語数）は，９５％であった．品詞を問題にしなければ９７％である．

本プログラムはPascalでインプリメントした．処理速度は，DEC2060 のＣＰＵタイムで，１文（平均23単語）当たり 3.8秒であった．このうち辞書ひきに要する時間が約90％を占める．

8．正しく分かち書きできない原因と考えられる対策

下記の原因のうち１）と２）が主要なものである．なお，辞書中のノイズになる見出しを削除するという場当たり的な方法は，入力対象を限定した場合には有効であるが，本質的な解決にはならない．分類や隣接検定をより細かくする方針をとるべきである．また，本システムはある入力文に対してどのような分かち書きが行なわれるか試験できるコマンドを備え，ユーザのキーワード選択の助けとしている．

１）未登録語の存在

辞書の拡充で対処すべきである．表記のゆれにも対処しなければならない．未登録語が存在してもある程度単語の区切りは予測できると思われる［６］が，本システムでは，その様な処理は行っていない．

２）単語候補の優先順位

処理時間や簡明さを考慮して，最初に見つかった解を一つだけ出力することにしたので，単語候補の優先順位が問題となる．精度の向上のためには，隣接検定を多レベルにすることが簡単でかつ有効な方法であると思われる．最も簡単には，隣接可能（出現頻度が多い），隣接可能（出現頻度が少ない），隣接不可能の３レベルにすることが考えられる．ＫＷＩＣのキーワード抽出という用途からみれば，複数の候補があればそれらを全部出力してやるのもひとつの方法であろう．

３）隣接検定が不十分
４）単語認定基準があいまい

9．おわりに

本システムは改良すべき点はあるが，現状でＫＷＩＣのキーワード抽出用として実用に耐えうるものと考える．

なお，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からの委託により，松下電器在職中に行ったものである．

１０．参考文献

［１］牧野他，“べた書き文の分かち書きとかな漢字変換”，情報処理学会論文誌，vol.20,No.4, pp.337-345,1979 .
［２］吉村他，“文節数最小法を用いたべた書き日本語文の形態素解析”，情報処理学会論文誌，vol.24,No.1,pp 40-46,1983 .
［３］内田他，“自由入力方式のカナ漢字変換”，情報処理学会自然言語処理研究会資料，27-3,pp.1-8,1981.
［４］館林他，“自由文入力・かな漢字変換方式”，情報処理学会第２６回全国大会講演論文集，論文番号2H-1,pp.1165-1166,1983.
［５］植村，“電子計算機による自動索引の研究（上）”，電子技術総合研究所報告第７４３号，1974.
［６］吉村他，“未登録語を含む日本語文の形態素解析アルゴリズム”，九大工学技報，vol.55, No.6, pp.635-639, 1982.

数式を含む日本文の構文解析

4S-4

和田 孝， 西谷 泰昭， 岩元 晃二
日本電気（株）ソフトウェア生産技術研究所

1．はじめに

我々は、プログラム自動生成システムＰＧＥＮを開発中である。ＰＧＥＮは、日本文、図、表で記述されたモジュール仕様書からプログラムを自動生成する実験システムである［1，2］。モジュール仕様書の日本文には、条件や計算を表現する数式が多く含まれる。これらの数式は日本文との明確な区切りなしで混入しているのが普通であり、また数式の中には日本語の表現が含まれる。そのため、日本文の構文解析において、数式の範囲、構造のあいまいさが問題となる。ＰＧＥＮにおける構文解析では、数式を日本文と同じ枠組みで解析する方式をとっている。本稿では、その方式について報告する。

2．構文解析の概要

ＰＧＥＮの構文解析の全体構成を図１に示す。構文解析は入力日本文を、その構造を表現したネットワーク（Ｎ－ネットと呼ぶ）に変換する過程である。

文節情報抽出では、分かち書き解析で認識した文節を、次の係り受け解析での単位となる文節の形式へ変換する。

係り受け解析では文法に従って文の構造を解析し構文木を生成する。日本文の構造のあいまいさは構文木の部分木を同一の位置に複数個持つことにより表現する。［1］

最適木選択では複数の部分木から最適な部分木を選択することにより最適な構文木に絞り込む。最適な部分木の選択は、各部分木に対して各文節の係りの長さの値を並べたベクトルを計算し、それらを比較することにより行なう。［1］

Ｎ－ネット生成では、最適な構文木をルートからたどりながらネットワークを生成する。

3．数式の文法

日本文の中に混入を許す数式を図２のように定めた。二項演算子（ＢＯＰ）と単項演算子（ＵＯＰ）は辞書に登録することによりユーザが自由に定義できる。ユーザは各演算子について優先順位と結合性（左，右，否結合）を指定する。名詞句（ＮＰ－Ｄ，ＮＰ－Ｆ，ＮＰ－Ａ）は日本文の文法に従う。名詞句は、中に数式を含んでもよい。

名詞句を終端記号とみなした場合、図２の文法はあいまいであるが、演算子の優先順位と結合性で解決される。

4．数式を含んだ日本文の構文解析

プログラム自動生成システムの入力としては、ユーザに数式の構造を明示的に指定させないことが望ましい。その場合には、たとえば

(i)　「入力の通帳残高＝元帳の通帳残高でなければ．．．」

のように、数式に含まれる日本語のために数式の範囲があいまいになったり、

(ii)　「Ｎ＋１番目の入出金明細」

のように構造のあいまいさ［（Ｎ＋１）番目の入出金明細　か　Ｎ＋（１番目の入出金明細）か］が生じ得る。

このような日本語に起因するあいまいさを前述の最適木選択を利用して解決することにした。そこで、数式を日本文の場合と同じ枠組みで構文解析する方式をとった。

日本文は文節の並びであり、文節の間の係り受け関係を決定するのが日本文構文解析である。したがって、数式において文節に相当するものを決定し、その文節

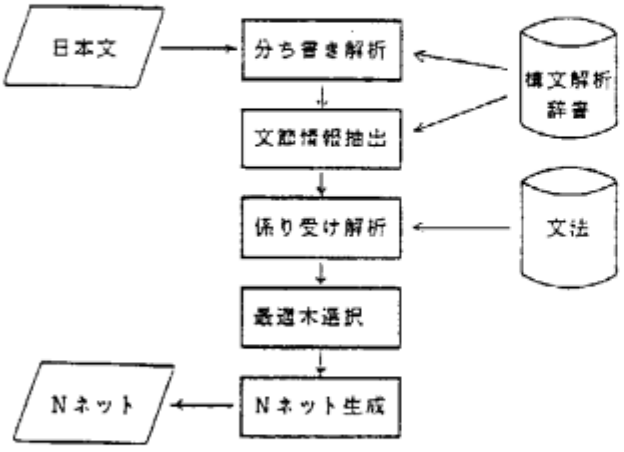


図１．日本文構文解析部の全体構成

```
<式>  ::=
        <式>  <ＢＯＰ>  <式>
    |   <ＵＯＰ>  <式>
    |   '(' <式>  ')'
    |   <ＮＰ－Ｄ>
    |   <ＮＰ－Ｆ>  '(' <式列>  ')'
    |   <ＮＰ－Ａ>  '(' <式列>  ')'
          ＢＯＰ：  二項演算子
          ＵＯＰ：  単項演算子
          ＮＰ－Ｄ：  データを表現する名詞句
          ＮＰ－Ｆ：  関数を表現する名詞句
          ＮＰ－Ａ：  配列を表現する名詞句

<式列>  ::=
              (空)
    |   <式>
    |   <式列>  ','  <式>
```

図２．数式の文法

の間の係り受け関係を定める必要がある。

5．数式における文節と係り受け

式は日本文における句に対応する。日本文の場合、たとえば「以下の処理を行なう」は２つの句、「以下の処理」と「行なう」が「を」という係り受け関係で結合してできた句である。二項演算子に関する式、たとえば

＜式１＞　＝　＜式２＞

において、＜式１＞，＜式２＞は句であり、＜式１＞から＜式２＞へ係り受け関係があり、両者が結合して新しい式、すなわち句となると考える。二項演算子 '＝' は日本文における付属語（「を」等）に相当する。式における係り受け関係を図３に挙げる。

日本文の文節は係り受け関係の対象となる最小単位であり、句を構成する最小単位である（上述の例では「以下の」「処理を」「行なう」）。式においては、その係り受け関係から考えて文節に相当する最小単位は、

(i)　データを表現する名詞、または '(' と ')' で囲まれた式／式列

(ii)　二項演算子（ＢＯＰ）が後ろに付いた(i)

(iii) 単項演算子

(iv)　関数／配列を表現する名詞

である。また、名詞句（ＮＰ－Ｄ，ＮＰ－Ｆ，ＮＰ－Ａ）の部分は、日本文の文節の解釈に従う。

上記の文節の解釈では、１つの文節の中に '(' と ')' で囲まれた式／式列を含む場合、その文節の子として式の文節列が属することになる。すなわちカッコの構造が文節の入れ子構造として表現される。

文節の間の係り受け関係は日本文の場合と同じく、２句の間に係り受け関係があれば、両句のそれぞれ最後の文節の間に係り受け関係があるとする。

図４に式における文節とその間の係り受けの例を示す。

6．構文解析方式

数式は日本文と同一の方式で構文解析する。以下、数式についてのみ述べる。

文節情報抽出では各入力について、数式の部分を前述の文節の列として解析する。数式における '(' と ')' の対応をとるのはこの時点であり、文節の入れ子構造が生成される。

係り受け解析では文法に従って式の構造を決定する。文法は２句の間の係り受け関係を記述したものである。式に関する文法は２句に含まれる演算子（正確には、左側の句の最後に現われる演算子、右側の句のトップレベルの演算子および最後に現われる二項演算子）に着目し、それらの優先順位と結合性の比較により記述されている。

この文法規則により、たとえば、"Ａ＊"，"Ｂ＋Ｃ" は優先順位が '＊' ＞ '＋' であるので結合しない（係り受け関係がない）。また、"Ａ＋"，"Ｂ＋Ｃ" は '＋' の結合性が左であれば結合しない［（Ａ＋Ｂ）＋Ｃが標準形である］。

最適木選択では、前述した数式に関連したあいまいさを除去する。選択は主に、演算子のオペランドのデータ型チェックにより行なわれる。演算子のオペランドおよび結果のデータ型はＰＧＥＮの意味辞書に記述されている。「Ｎ＋１番目の入出金明細」では "Ｎ＋" が、"１" と "入出金明細" のいずれかへ係るので、両者への係りの良さをそれぞれ計算して比較する。'＋' のオペランドが実数であると意味辞書に記述されていれば前者が選択される。最適木選択では部分式のデータ型が計算され、それが上位の式のデータ型チェックに利用される。

7．おわりに

現在、数式に関連したあいまいさの解決は演算子のデータ型チェックに多く依存している。ＰＧＥＮでは構文解析、意味解析の処理形態がバッチ的であり、システムが最適と思うものを一意に決めている。この部分に会話機能を導入することを検討中であるが、これに伴い、数式のあいまいさの解決に、より多くのファクタを考慮してゆくつもりである。

なお、本研究は、ＩＣＯＴからの委託（発注０７０６号）の一環として行なったものである。

参考文献

［１］西谷，和田，岩元「知識利用のプログラム自動生成システム」情報処理学会第３０回全国大会４Ｋ－１０

［２］岩元，西谷，和田，臣司，土田「ＰＧＥＮ－１：仕様書理解をベースとしたプログラム自動生成システム」情報処理学会第３２回全国大会５Ｍ－６

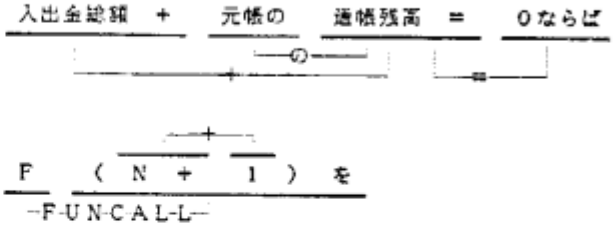


図４．式における文節と係り受けの例

＜式＞	係り受け関係名	係り受け
＜式１＞ ＜ＢＯＰ＞ ＜式２＞	＜ＢＯＰ＞	＜式１＞ → ＜式２＞
＜ＵＯＰ＞ ＜式＞	＜ＵＯＰ＞	＜ＵＯＰ＞ → ＜式＞
＜ＮＰ－Ｆ＞ '(' ＜式列＞ ')'	ＦＵＮＣＡＬＬ	＜ＮＰ－Ｆ＞ → '(' ＜式列＞ ')'
＜ＮＰ－Ａ＞ '(' ＜式列＞ ')'	ＡＲＲＡＹＲＥＦ	＜ＮＰ－Ａ＞ → '(' ＜式列＞ ')'

図３．式における係り受け関係

文書要約システムにおける世界知識の構造[1]

6M−7

安原宏　小松英二　北研二　加藤安彦　山本由紀雄
（沖電気工業株式会社　総合システム研究所）

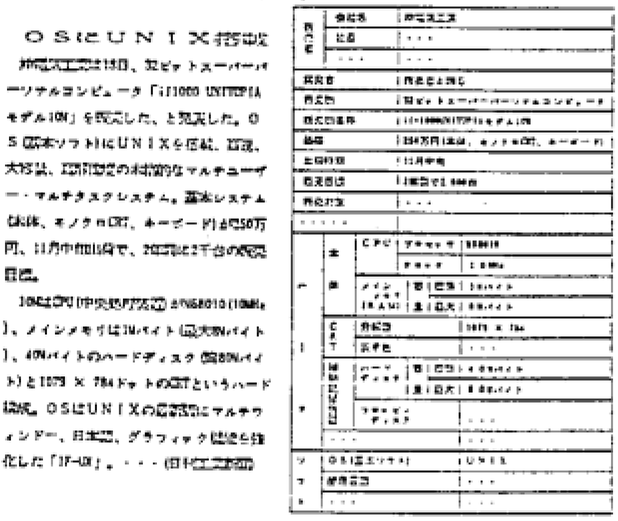
1．はじめに

自然言語処理の１つの応用として文書要約がある。我々は、情報産業関連の新聞・雑誌等の記事を対象とした要約システムを検討している。本稿では、特にコンピュータの新製品発表記事に限定したものについて述べる。

要約手法としては、あらかじめシステムで用意した概念を入力文書中の具体的な値に結合する一種のフレームにおけるスロット代入法を採用した。この考え方に沿った研究として文献［１］で既に小規模な実験が報告されている。そこでは論文抄録を深層の意味表現として構造化したキーワードが用いられている。我々は、そのような概念構造化の他に意味記述も導入することにより、単に記号としてではなく実体として理解，要約ができるシステムをめざしている。

2．要約システム概要

文書要約として一般の知識を扱う文書や長文を対象にすることは、現在の自然言語理解技術では未だむずかしいので、我々は、対象分野を限定し、かつ短文で実験してみることにした。まず、対象分野をコンピュータ関係の新製品発売に関する記事に選んだ。コンピュータの知識は比較的明確化し易く、記事の内容も類型化したものが多い。図１に要約例を示す。


ＯＳにＵＮＩＸ搭載

沖電気工業は15日、32ビットスーパーパーソナルコンピュータ「if1000 UNITOPIA モデル10H」を開発した、と発表した。ＯＳ(基本ソフト)にＵＮＩＸを搭載、…、大容量、…のマルチユーザー・マルチタスクシステム。基本システム(本体、モノクロCRT、キーボード)は約50万円、11月中旬出荷で、2年間に2千台の販売目標。

10HはCPU(中央処理装置)がMC68010(10MHz)、メインメモリは1Mバイト(最大8Mバイト)、40Mバイトのハードディスク(最大80Mバイト)と1079×784ドットのCRTというハード構成。ＯＳはＵＮＩＸの…にマルチウィンドー、日本語、グラフィック機能を強化した「IF-UX」。・・・(日刊工業新聞)

発売者	会社名				沖電気工業
	社長				・・・
	・・・				・・・
開発者					発売者と同じ
開発物					32ビットスーパーパーソナルコンピュータ
開発物名称					if1000UNITOPIAモデル10H
価格					約50万円(本体，モノクロCRT，キーボード)
出荷時期					11月中旬
販売目標					2年間で2,000台
用途対象					・・・
・・・・・					
ハード	本体	CPU	プロセッサ		MC68010
			クロック		10MHz
		メインメモリ(RAM)	容量	標準	1Mバイト
				最大	8Mバイト
	CRT	解像度			1079×784
		表示色			・・・
	補助記憶装置	ハードディスク	容量	標準	40Mバイト
				最大	80Mバイト
		フロッピィディスク			・・・
	・・・				・・・
ソフト	OS(基本ソフト)				UNIX
	使用言語				・・・
	・・・				・・・

図１．記事の要約例

3．世界知識

２．で述べた要約システムで用いる世界知識について特徴的な考え方を述べる。

知識表現の世界では、上位−下位関係、部分−全体関係、属性−属性値関係等が事物の知識を表現するのに一般的に使われている。上位−下位関係では性質の継承が重要な役割を持っているが、本システムでは、性質継承性の代りにマクロ機能を導入し、事物に対して個別に知識を記述する。これはよく使う木構造の知識をマクロで宣言しておき、知識表現中で自由に使えるようにしたもので、記述能力と可読性を高めるものといえる。例えば図２は、パーソナルコンピュータの世界知識であるが、もしその知識が一般のコンピュータと同じものであればマクロでコンピュータを呼び出せばよい。他方、マイクロコンピュータの知識として独自に記述すると、言語やＯＳの細部でコンピュータの知識を特殊化したり、別のモジュールを追加することができる。

知識を記述するときは、上位−下位、部分−全体等の関係が自由に使用できることが望ましい。本システムでは、関係の意味付けに拘泥しないで、事物の知識を木構造で自由に表現することにしている。各ノードに記載する語は代表的な"概念"であり、適宜定義できるものとする。従ってこの知識表現を形式的に記述すると次のようになる。

＜ＷＫ＞＝＜ルート概念＞　→　＜Ｓ＞
＜Ｓ＞＝｛＜マクロ＞｜＜概念＞｜＜概念＞　→　＜Ｓ＞｝…

ここで→は下位概念へのリンクで、ルート概念から木構造で拡がり、その概念特有の知識を表すものである。これ以外に陽に木構造にあらわれないで任意の場所で使えるものがある。例えば価格や名称の知識がそれである。これらは、それが言明された知識に付属させて出力される。

上記概念間を関係づける知識に加えて概念そのものを定義する意味辞書を作成する。従来の知識表現の弱点は意味記述を明記しなかったため、概念の類似性が検出できなくて、システムが fail safe でなかったことである。あらかじめ完璧な知識を記述することは本来不可能であろう。既知のことをすべて記述すれば際限がなく検索効率も悪くなる。例えばコンピュータの知識をトランジスタレベルまで分解する必要はないであろう。また予想したものと別の概念が文中に出現することもある。そのときにもできるだけ既存の知識の中で近いものと関係付けられることが望ましい。意味辞書はこれらのことを

[1] この研究は第５世代コンピュータプロジェクトの一環としてＩＣＯＴからの委託で行われたものである。

119

可能にするために導入したもので、その内容は図３に示すように、数値や図形コードで表現する形態情報、木構造の知識を指す構造情報、および、言葉で表現する機能情報からなっている。形態情報は、ハードかソフトかを区別したり、物の大小関係を判定するのに使え、構造情報は、新しいコンポーネントが宣明されたときに木構造に加えるのに用い、機能情報は、計算するのか、記憶するのか、入出力をするのか等の区別をつけるのに用いる。機能を表現する用語はシステムで解釈が与えられている。またこの意味辞書には固有名詞の知識も記載されており、「○○○の３倍の処理能力」といった場合にも具体的値がとり出せるようになっている。

4．要約処理

要約の出力となる表は、あらかじめ想定した構造知識の中から１つ選定する。この選択方法としては、見出し語の解析、記事の先頭部分の解析、記事中の専門用語の頻度からの推論等が考えられる。対象のルート概念が定まると、それに関する木構造知識をマクロ展開しながら作成して行く。これで、スロットを埋める準備ができたことになり、後は文章を解析しながら、与えられた構造知識との対応を取って行く。もし文章中の言葉（例えばプロセッサ）が木構造知識の中で複数箇所（例えばＣＰＵのプロセッサとグラフィックのプロセッサが）存在すると、現在の文脈に近い方を選ぶ。逆にマクロ展開した木構造知識の中に一つも出現していない概念（例えばファイルサーバ）が文中に出現すると、それの木構造への代入を断念するか、辞書項目を用いて、最も意味的に近い位置に新たにつけ加えるかのいずれかを選択する。文章が全て処理し終ると、表の中で記入された部分のみを出力する。

5．おわりに

要約システムで用いる世界知識の階層構造による表現と意味辞書について述べた。この方法では知識を要素概念で整理しており、変更・拡張に適した構造となっている。本システムは今後Ｐｒｏｌｏｇを用いてインプリメントする予定である。特にポストエディットや知識ベースのデバッグ・更新を容易にするために、要約システムに適した知識エディタの開発も考えている。専門用語の数が増大して行くにつれて、処理速度の低下が予想されるため、高速化の工夫が必要となろう。

最後に本研究の機会を与えて頂いたＩＣＯＴの横井俊夫第２研究室長、研究推進を支援して頂く総合システム研究所似鳥一彦部長、並びに、貴重なコメントを頂いた木下哲男、奥村晃の両室員に感謝する。

文献[1]　猪瀬，斉藤，堀：シナリオを用いる論文抄録理解・作成援助システム，情報処理学会論文誌，pp22-29 vol 24 NO 1. 1983

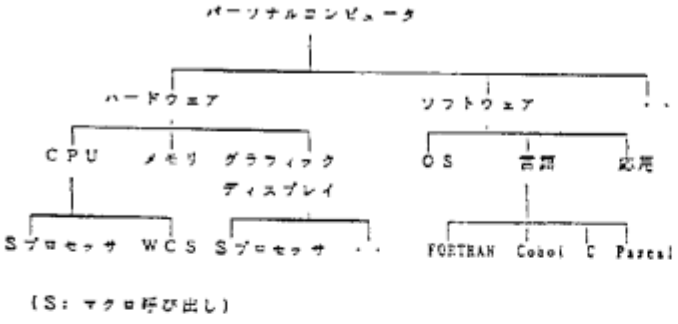
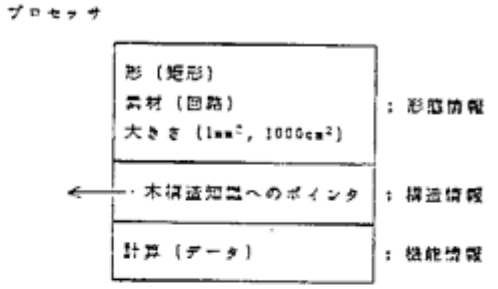


図３．意味辞書の例



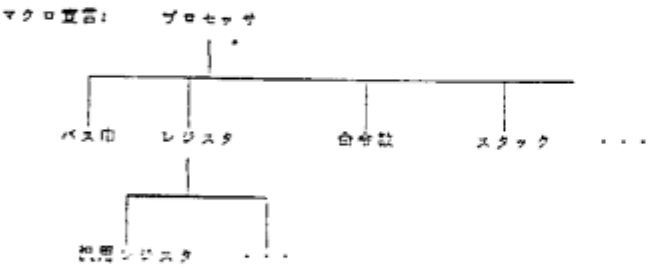


図２．知識表現の例

5M-6

ＰＧＥＮ－１：仕様書理解をベースとしたプログラム自動生成システム

岩元 亮二，西谷 泰昭，和田 孝，庄司 功，土田 賢省
日本電気（株） ソフトウェア生産技術研究所

１．はじめに

人間であるプログラマ向けに書かれたモジュール仕様書は、日本文、図、表の組み合わせで記述されることが多い。我々は、このような自然な形式で表現されたモジュール仕様書を理解しプログラムへ自動変換する方式について研究しており、その第一次プロトタイプシステムＰＧＥＮ－０については既に報告ずみである［１］。その後、図形の意味処理機能の追加、表の表現能力の強化、日本文のあいまいさ解消と省略語補完機能を強化し、文対応でなくモジュール仕様書単位で処理できる実験システムＰＧＥＮ－１を開発した。本稿では、意味処理機能を中心にＰＧＥＮ－１の特徴を述べる。

２．ＰＧＥＮ－１の概要

本システムの開発に当たって、対象分野を銀行オンライン業務プログラムとし、実際に書かれたモジュール仕様書を参考にして出来る限り制限を設けずに処理できるようにする方針をとった。

図１にＰＧＥＮ－１のシステム構成を示す。システムは、図２、図３に示すような入力に対して、ＣＯＢＯＬ／Ｓ（ＮＥＣの構造化ＣＯＢＯＬ）のソースコードを出力する。入力のモジュール仕様書（日本文）は、節・段落構造を持ち、節は節番号（[.]で囲まれている）と節タイトル（省略可）からなる節ヘッダで始まるものである。

日本文、表、図の各入力は、入力処理部で構造解析され、それぞれＮネット、Ｔネット、Ｇネットと呼ばれるネットワークに変換される。これらのネットワークは入力に対応した構造を表現しており、冗長な情報、意味的なあいまいさ、情報の不足などをそのまま表現している。意味処理部は、これらのネットワークを参照して仕様書の意味を表わす意味ネットワーク（Ｍネット）を作成し、更に適当な知識を用いてＭネットをプログラム生成に十分な形に変形、補完していく。Ｍネットが正しく生成された後、プログラム生成部によりプログラムに変換される。

以上の処理では、種々の知識が利用されるが、これらの知識は、対象システムに固有なものとそうでないものに大別され、主に辞書に保持されている。対象システムに固有な知識は、主にデータに関連した情報であり、データの構造と属性、コード表、それらと日本語名との対応などの情報を持つ。これらの知識は、システムのデータ設計結果に対応し、ＰＧＥＮ－１ではデータ型辞書、変数辞書という形でユーザが定義するようになっている。一方、対象システムに固有でない知識として、日本文、表、図の文法的規則、あいまいさ解消や不足情報補完のための手続き的知識、プログラミング言語についての知識、主に用言（「代入」、「加える」など）に対応したプログラム上の意味知識などがある。これらの知識はあらかじめＰＧＥＮ－１に保持されている。

```
[1] 入金記録の計算
      会社が "00" でない入金情報について，以下を繰り返す。
   (1) 総額に入金額を加える。
   (2) 入出金口数をカウントアップする。
[2] 処理キーコードエリアのセット
      有効のとき，以下の処理を行なう。
   [2.1]
      使用区分が1のとき，以下を行なう。
   (1) コード数をカウントアップする。
   (2) コード数が5以下なら，コード数番目のコードを，"321"とする。
       0のとき，検証以外なら，エラーコードを"321"として，リターン。
[3] 処理コードエリアのセット
      有効のとき，以下の処理を行なう。
   [3.1]
      使用区分が1のとき，以下を行なう。
   (1) コード数をカウントアップする。
   (2) コード数が5以下なら，コード数番目のコードを"321"とする。
       0のとき，検証以外なら，エラーコードを"321"としてリターン。
[4]
      利回コードが"20"のとき，摘要コードに"N0"をセットし，
      "22"のときは，"N1"をセット。
[5]
      通帳がマル優限度額より大きいとき，表2に従い摘要コードに値をセット
      してリターンする。
[6]
      処理種別全のとき，表1の組み合せ以外なら，エラーを409としてリターンする。
[7]
      図1のとき，リターンする。
[8]
      図1に従い，値をセット。
```

図２　入力日本文の例

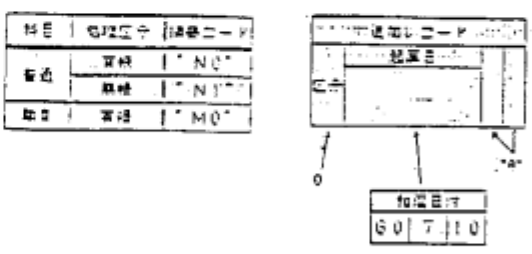


図　３　表、図の入力画面

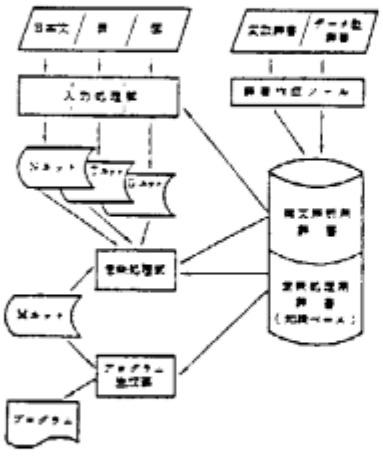


図１　ＰＧＥＮ－１のシステム構成

3．日本文の意味処理

意味処理の入力であるNネットは、格文法の考え方にそったネットワークであり、構文上のあいまいさは構文解析の妥当性ベクトル［1］の考え方により既に除去されている。意味処理部では、冗長表現の削除、意味的あいまいさの解消、不足情報の補完、プログラム上の意味への変換を行なう。

（1）冗長表現の削除

「Xの組み合わせ」という表現は、Xが表や図の参照、あるいは変数の並びを表わす表現のとき、単に「X」という表現で十分であり、「の組み合わせ」は冗長である。日本文ではこの他に、「Xの値」、「X番目」、「Xの処理」などのように、Xの種類によっては単に「X」としてもよい表現がある。PGEN－1では、Xの種類を見極めた上でこれらの冗長表現を削除する。

（2）並列句の解釈

簡潔な表現法として仕様書のなかには並列句が多用されている。例えば、「A，Bにa，bをセット」、「A，Bに1をセット」、「A，Bがa，bである。」などが典型的な例である。構文解析結果のNネットでは、並列句は図4（a）のように表現されている。この例は「A，Bにa，bをセット」に対応し、並列句「A，B」に並列句「a，b」をセットするということを表している。意味処理部ではこれを「Aにaをセットし、Bにbをセットする」という意味のMネット（図4（b））に変換する。また、「A，Bに1をセット」は「Aに1をセットし、Bに1をセットする」を表わす意味ネットに変換する。このように、並列句については、述語の種類とその格の内容の組み合わせによってどのように意味ネットに変換するかを手続的知識として組込んでいる。

（3） あいまいさの解消

日本文の単語に対して、プログラム仕様書ではいくつかの意味が対応する。例えば、「である」（be動詞）に対しては、等しい（eq）、含まれる（member）、代入する（set）、などの意味がある。また、データを示す名詞についても一つに変数を特定できない場合がある。

あいまいさ解消に仕様の論理構造を規定するMモデルを利用する。MモデルはMネットのひな型であり、Mネットのノード間の結合条件を規定している。例えば、Mモデルの一部（図5）により、setのobj-rとしてaddが許されることがわかる。このMモデルによって妥当な解釈の範囲を絞ることができる。この方法は特にif文の条件部と実行部の区別に有効で、条件部にはpredicateが要求され、実行部にはstatementが要求されることからbe動詞のあいまいさを除くことが可能である。

データを表わす名詞のあいまいさ解消には、データの構造と属性に関する知識を利用するが、さらに節タイトルも利用する。例えば、図2の2節の中では、「コード数」を監査キーコードエリアのサブフィールドの「コード数」と同定し、3節の中では注意コードエリアのサブフィールドの「コード数」と同定する。

（4） 省略語の補完

データ型とコード表の知識を利用して、「普通預金のとき」が「科目コードが普通預金（＝21）に等しいとき」の意味に対応するMネットに変換されるのは先に発表した通りである［1］。このほかに、繰返し表現に伴う省略語補完と局所的文脈による補完を行う。

図 4　並列句の表現

図 5　Mモデルの一部

繰返し表現では、図2の4節の例のように同型のパターンが繰返されるときに後のパターンの中で前のパターンと同一のものが省略される現象がみられる。この例では、後のパターンで「科目コードが」と「摘要コードに」が省略されている。PGEN-1では直前に類似のパターンがあるかどうかみてパターンによる補完を行う。

局所的文脈利用の補完の例は、「AとBを加え、Zに代入する」を「AとBの和をZに代入する」の意味への変換である。これは「代入」に対する補語obj-rを補完している。この場合、（3）で述べたMモデルに合致する対象を局所的文脈から同定している。

4．図表の意味処理

図の意味処理では変数に関する知識を参照してデータ構造図としての意味を同定する。関係を表わす線分の両端の接する範囲については、両端でデータ属性が矛盾しない最小の範囲を選択する。関係の種類はeq(等号)、set(代入)のいずれかを決定する。このとき日本文の意味表現を参照し、条件部で図を参照していればeq、実行部で参照しているときはsetと判定する。

表の意味処理では、日本文において条件としてのみ表を参照しているときには(and,orでつなげた)条件の意味表現を、それ以外の場合は(if-then-elseの選題における)条件とアクションの意味表現を生成する。表のアクション部については、見出しと日本文の意味表現との間の関連を同定し、その結果により表の意味内容を日本文の意味表現のどこに埋めるべきかを決定する。

図と表の意味処理部は日本文の意味処理の途中のフェーズで起動され、その結果は日本文の意味表現と結合されてMネットを形成する。このMネットは日本文意味処理部の後段のフェーズによって繰り返し文としての構造化、あいまいさの解消、省略語補完の処理を受ける。

5．おわりに

はじめに述べたように、PGEN－1はできるだけ現状の仕様書を理解できることを目的としているが、今後は確実に理解できる水準を定め実用的なものにすること、及び部品に対応する高水準の慣用表現の定義機能を追加することが重要であると考える。最後に本研究に対し励ましと御指導を下さった日電・ソフトウェア生産技術研究所の赤津所長と藤野支配人に深謝する。尚、本研究は、（財）新世代コンピュータ技術開発機構からの委託（発仕3401－04号）の一環として行ったものである。

文献［1］西谷、和田、岩元「知識利用のプログラム自動生成システム」昭60情処学会全大第30回

並列論理型言語によるＬＳＩのフロアプランシステムの検討

2U-2

－実験システム概要－

吉永和弘　青柳洋介　白水昇
（沖電気工業株式会社）

1．はじめに

ＶＬＳＩの集積度の増大及び機能の多様化が日々進行しており、設計作業の効率化のために知識工学の応用が、又高速化のために並列処理の導入が求められている。そこで、第５世代コンピュータ計画での核言語の１候補である並列論理型言語Ｇｕｒｄｅｄ　Ｈｏｒｎ　Ｃｌａｕｓｅｓ（ＧＨＣ）［１］を用い実験システムの試作中である。特に、従来の手法では、高品質・高速のフロアプランシステムは実現が困難であると考えられている。

本稿では、実験システム構成及び配置機構概略構成について述べる。

2．実験システム構成

フロアプラン決定に関しては、設計者のノウハウに多くを依存しており実現が困難であるので、まず、既存の手法でもある程度自動化されている配置機構について試作した。また、既存ＣＡＤシステム中の設計データの流用や実験システムの評価が必要であることからＤＢインタフェース機構を、大量の図形データを取り扱うためにグラフィクスシステムが不可欠であることからユーザインタフェース機構についても試作した。（図１参照）

2．1　配置機構

設計データを入力することにより、ＧＨＣの並列プロセスとして配置モデルを生成し、配置状況を表示しながら配置処理を実行し、配置結果を出力する。

2．2　ＤＢインタフェース機構

ＣＡＤデータベース中に格納されている大量の設計データをＧＨＣ上へ抽出し、逆にＧＨＣ上の配置結果をデータベース上に併合する。

ＧＨＣ上から直接、データを入出力する直接方式と、データをＧＨＣ形式へ一旦変換し入出力する間接方式が考えられる。直接方式では、ＧＨＣとデータベース間で整合をとることが困難であるので間接方式でのみの試作を行った。

2．3　ユーザインタフェース機構

配置対象の状態を随時グラフィック端末に表示する。

ＧＨＣ上に全体を実現する組こみ方式と、ＧＨＣ上には図形処理コマンドの入出力インタフェースのみを実現する通信方式が考えられる。組こみ方式は、並列性を考慮した効果的インタフェースが可能となるが実現が困難であるため通信方式のみの試作を行った。

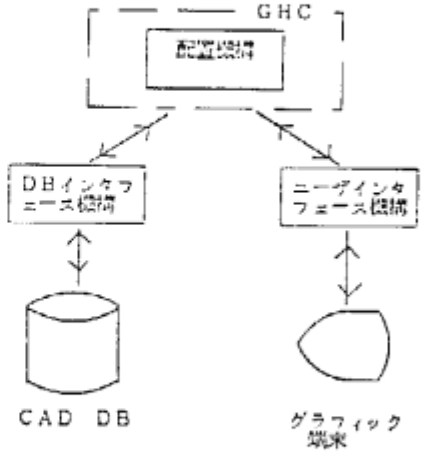


図１．　実験システム構成

3. 配置機構概略構成

GHCは、AND並列を並列プロセスの記述に、OR並列を非決定性プロセスの記述に用いることができる。また、プロセス間の共有変数を通信路として用いプロセスの同期を取ることができる。そのため、Concurrent Prologの場合と同様に、プロセスをtail-recursiveに走らせることでオブジェクト指向的なプログラミングが可能となる。([2],[3])そこで、基本的にオブジェクト指向の考えをふまえ、並列性を導入することにより配置機構のモデル化を行った。(図2参照)

3.1 設計データ

設計データは、プロセスとして生成されると内部情報を出力したただちに消滅する、状態を持たず事実のみを表すオブジェクトとしてモデル化した。

データベース中には、各種ブロック(配置対象、ライブラリ等)がメンバ(ブロック毎の属性で、例えば外形や内部端子等)毎に格納されている。そこで、記述の単位をメンバ毎にし、形式を

db([ブロック名,メンバ名],
　出力):-
　true|出力=内部情報。

に統一しプロセスの生成を単純化した。

3.2 プロセスジェネレータ

プロセスジェネレータは、設計データから受けとった設計情報を適宜入力することで、配置モデル内の全てのプロセスを具体化し、最後に消滅するオブジェクトとしてモデル化した。

具体化は、プロセス毎に、プロセス名、属性(当該プロセスの固定的情報)、通信チャネル、初期状態を設定することにより行われる。

3.3 配置モデル

配置モデルは、ブロック(素子、端子)及びエリアから構成される並列プロセス群が相互に通信を繰り返しながら初期状態から最終状態へ変移し消滅するオブジェクトとしてモデル化した。

なお配置モデルの詳細については、別稿[4]で説明する。

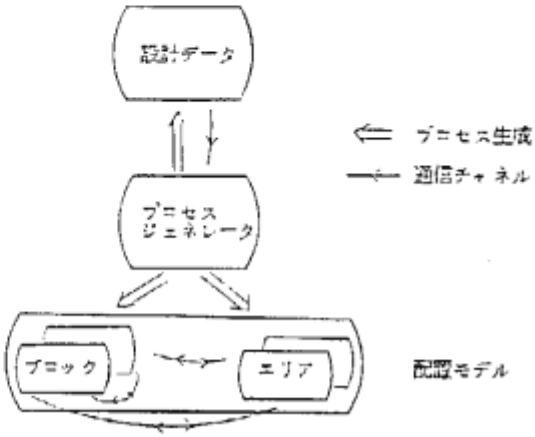


図2. 配置概略構成

4. おわりに

実験システムのプロトタイプが構築できた。しかし、配置モデルの変更に伴いプロセスジェネレータやDBインタフェース機構を変更しなければならない。そこで現在は、これ等に独立性をもたせ効率よく実験できるように検討している。

なお、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行われた。御討論をいただいたICOT第1研究室吉川室長に感謝する。

〈参考文献〉
[1] 核言語第1版説明資料, ICOT
[2] How to solve it in Concurrent Prolog, ICOT
[3] 野田他, 並列処理方式論理シュミレータの一検討, 第30回情報全大
[4] 吉永他, 並列論理型言語によるLSIのフロアプランシステムの検討 ーデータ表現手法ー, 第32回情報全大

6U-3

並列論理型言語による ＬＳＩのフロアプランシステムの検討

－ データ表現手法 －

吉永和弘　青柳洋介　白木昇
（ 沖電気工業株式会社 ）

１．はじめに

ＬＳＩのフロアプランシステムへの並列性の導入に関し，今回並列論理型言語ＧＨＣを用い配置機構について試作，実験を行った。システムをオブジェクト指向的考え方でモデル化し，その構成要素である設計データ，ブロック，エリアなどをＧＨＣの並列プロセスで構成し，その表現方法や通信方法について検討を行った。

本論文では配置機構のモデル化，各プロセスの機能，問題点について報告する。

２．配置機構

配置機構は，初期配置部と配置改善部に分割して考えた。今回は初期配置部について検討を行い試作した。また配置対象はポリセル方式のＬＳＩの最下位層とした。以下，配置対象，初期配置部のモデル化について述べる。

２．１　配置対象

配置対象は，ブロック（端子，素子），内部に含まれるブロックの種類，大きさにより算出された数段のエリアより構成される。

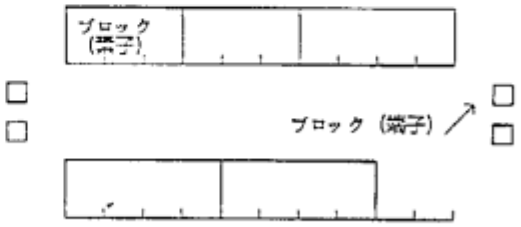


図２．１　配置対象　エリア

２．２　配置機構のモデル化

入力端子側から信号の流れに沿ってセルが配置されてゆく様なモデルを検討した。以下，ブロックモデル，エリアモデルについて述べる。

（１）ブロックモデル

ブロックは隣接ブロックと同期を取るための同期プロセス，移動規則を持ちエリアと通信を行いながらブロックを移動させる移動規則プロセス，及びこれらのプロセスを制御するブロックマネージャから構成される。

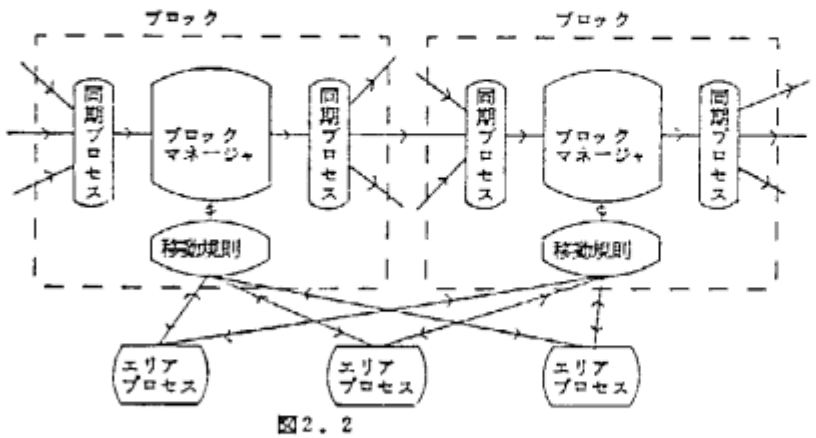


図2．2

① ブロックは入力側の隣接ブロックの移動終了で同期をとり，移動を開始する。移動開始の条件は隣接移動済ブロックとの結合密度の合計値が一定値に達した場合か，隣接ブロックのすべてが移動終了となった場合である。

② エリアと通信を行いながら配置可能な空エリアのうち，

Ｃｏｓｔ＝Σ（距離×結合密度）

を最小にするエリアを占有する。そのとき，他のブロックから参照されているエリアがあれば，そのエリアが解放されるか，配置済になるまで待つ。

③ 移動が終了したら，出力側隣接ブロックにメッセージを送り消滅する。

```
block(BLKNM.St.In.Out.Ar_chs):-
    St=[state(BLKNM.trans.Area.Pos)|Ns]|
    block_move_rule(Ar_chs.Pos),
    block_area_sansho(BLKNM.Pos.
          Ar_chs.Ar_chs_t.Res),
    block_area_haichi(BLKNM.Res.
                Ar_chs_t.Ns),
    Ns=[state(BLKNM.State.Area.
              (X1.Y1))|Ns1] .
    block_st_change(Res.State),
    block_trans_st(Res.Out.Out1),
    block(BLKNM.Ns.In.Out1.Ar_chs).
```

ブロックの記述例

（２）エリアモデル

ポリセル方式のＬＳＩではブロックはその種類により異なる幅を持つ（２～数十グリッド）。そのため，エリアを格子状に規格化された網目（メッシュ）に分割して表現した。各メッシュごとに１つのプロセスが存在する。

① エリアはブロックから参照メッセージを受け取ると自分の状態（配置済，参照中，未配置など）を返す。

② ブロックから配置要求メッセージを受け取ると配置可の場合自分の状態を配置済の状態に変更する。

```
area(GRPNM-Mno.Ar_ch):-
    Ar_ch=[area(GRPNM.Mno.Type.(Y.X).
              State.Mode.Ack)|Ar_ch1].
    Mode=access(BLKNM.haichi).
    State=state(BLKNM.trans)|
    State1=state(BLKNM.done).
    Ack=ack(GRPNM-Mno.State1).
    Ar_ch1=[area(GRPNM.Mno.Type.(Y.X).
            State1.Mode1.Ack1)|Ar_ch2].
    area(GRPNM-Mno.Ar_ch1).
```

エリアの記述例

３．実験結果

今回は図３．１の回路を対象に実験を行った。結果は図３．２の様になった。

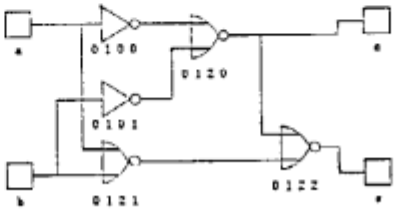


図３．１

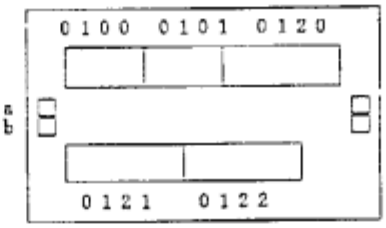


図３．２
配置結果

この実験によりブロックの配置を並列プロセスとして実行可能なことが確認できた。結果を人手による配置と比較すると最適とは言えない。これはブロックが局所的な情報しか知ることができないことと，移動規則が十分でないためであると考えられる。今後の配置改善部には，全体の監視を行うようなプロセスや移動規則の改良が必要である。また，より大規模な実データで実験を行い既存の配線システムにかけるなどの実用評価をする予定である。

なお，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行われたものである。御討論をいただいたＩＣＯＴ第一研究室古川室長に感謝する。

Multi-PSIシステムの概要

50-8

瀧 和男　木村 康則　横田 実　近山 隆　内田 俊一
（（財）新世代コンピュータ技術開発機構）

1．はじめに

日本の第五世代コンピュータプロジェクトでは、論理型プログラムを並列に高速実行する並列推論マシンの研究を主にアーキテクチャの観点から行なってきた[1]が、並列ソフトウェアの面では、多くの問題が未だ未着手のまま残されている。そのおもなものには、

(1) プロセッシング・エレメント間の負荷分散をソフトウェアで制御する方式
(2) 分散環境におけるメモリ管理やガベージコレクション方式
(3) 大規模な並列プログラムをいかにして書くか。
(4) オブジェクトコードの分配、管理方式
(5) 並列推論の世界での入出力や割込みの取扱い
(6) 分散環境におけるプログラムのテストとデバッグの方式

等があり、アーキテクチャの研究を加速するためにも、並列推論に関するソフトウェア上の研究を進展させる事が急務になっている。

そのためのアプローチとして、われわれは研究開発ツール用に別途開発を進めてきたパーソナル逐次型推論マシンPSI[2]を活用することとし、これを複数台接続して実際に並列計算機環境を作り、その上に並列推論言語の実行環境を構築することにした。これをMulti-PSIシステムと呼ぶ。その目的は、並列推論マシン用の、

(1) 言語 KL-1 (Kernel Language version 1)
(2) オペレーティングシステム PIMOS (Parallel Inference Machine's Operating System)
(3) 応用ソフトウェア

の研究開発を進めることであるが、開発にあたっては、ある程度大規模な並列ソフトウェアを書き，デバッグし、走らせるに足るだけの信頼性、使い易さ、性能を備えた並列推論言語の実行およびデバッグ環境を構築するよう、特に注意を払っている。

以下では、Multi-PSIシステムとその上で行なおうとしている研究開発の全体像について概要を述べる。

2．研究開発スケジュール

次の３つのシステムを予定している．

(1) Pseudo-multi-PSI（61年度初版完）：　１台のPSI上に作られるMulti-PSIのシミュレータで、ソフトウェアのデバッグに用いるほか、負荷分散戦略の調整に必要なデータの収集も行なう。処理系は(2)に準ずる。
(2) Multi-PSI(-V1)（61年度完成）：　現在のPSIを入出力装置も含めて 6台程度接続したシステムで、ESPベースのKL1処理系を実装する。PIMOSの各部を試作し、簡単な評価用プログラムを実行できるようにする。分散環境におけるデバッグ方式を検討する。
(3) Multi-PSI-V2（仮称）（63年度完成）：　改良小形化されたPSIのCPU部分を数十台接続し、その上にファームウェアで記述したKL1処理系を実装する。フロントエンドマシンとしてPSIを１台設置する。PIMOSの改良、拡張を行ない、本格的な応用プログラムを実行させ評価を行なう。

3．Multi-PSI（V1）システムの概要

PSIマシン 6台程度を接続ネットワークハードウェアにより格子状に接続した構成をとる。共有メモリは置かず、メッセージパケットの交換によりCPU間通信を行なうが、メッセージはユーザプログラムが陽に取扱うのでなく、言語の実行メカニズムの一部として自動的に生成される。実装予定の言語KL1は並列推論言語 GHC[3]をベースに置き、AND 並列、AND 結合されたゴール間のストリームによるデータ通信、ガード部による同期機能などを特徴とする。他のCPUへのゴール送出や他CPU内のデータとのユニフィケーションは、メッセージパケットの形に変換されて送出される。CPU間の負荷分配がゴール１個の単位で行なわれ得ることから、かなりfine-grainな並列性を取扱うシステムといえる。

Multi-PSI(V1)システムは PSIマシンをベースとして、図１の階層構成図中の下線部分を新たに製作する。主な構成部分について以下に述べる。

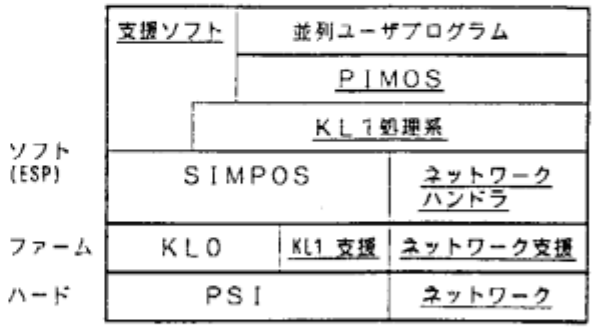


図１．Multi-PSI（V1）システムの階層構成図

4．言語と処理系

前述のGHC をベースとした言語にモジュール化機能を取入れたユーザ言語をKL1-u(user) と呼び、PIMOS および評価用プログラムの記述にはこれを用いる。一方、言語機能を更に分解して機械語相当にしたものをKL1-B(base) と呼び、マシン上の処理系はKL1-B 用に作られる。[4] 分散環境用のユニフィケーションや負荷分散制御のサポート機能はこの中で実現される。これらの他にユーザが負荷分散制御や実行優先レベルを指定するためのnotationでプログラムの意味とは独立に付加できるものをpragma(KL1-P) と呼んでいる。KL1-u で書いたプログラムにpragmaを付加したものが、コンパイラによりKL1-B に変換され実行される。

V1システムでは処理系はPSI のシステム記述言語ESP で記述の予定であり、効率向上の為必要に応じてファームウェアサポートを行なう。

5．PIMOS

並列プログラムを効率良く実行するための基本機能として 1. であげた(1)(2)(4)(5)の項目を実現する。中でも重要な試みは、距離の存在するネットワークすなわち通信のローカリティの概念を持ち込む必要のあるネットワーク上で、ソフトウェアにより負荷分散と通信の局所化を制御することである。これらは言語処理系とも深い関連を持つ。

PIMOS の初版では、並列推論マシン向きの実行管理と資源管理が主な研究テーマであり、ユーザに対するサービス機能はPIMOS の改良の段階で検討してゆく。

6．支援ソフトウェア

V1システム用の支援ソフトウェアは、PSI のプログラミング・オペレーティングシステムSIMPOSを活用してESP で記述し、各PSI 上に同じものを配置する。

(1) Multi-PSI システムモニタ： KL1 処理系と接続ネットワークの初期化、起動、停止、モニタリング、ハードウェア保守サポートを行なう。

(2) 分散環境デバッガ： 各プロセッサに割付けられたゴールや、プロセッサ間の通信に着目した低レベルのデバッガである。

7．接続ネットワークハードウェア

主に実装設計の容易さから格子状のネットワークを選択した。格子状ネットワークはCPU 間距離の概念が存在するネットワークの代表的なものであり、PIMOS における負荷分散制御の実験に適するが、V1システムではCPU の推論速度に比べてネットワークの転送速度が遅いため、異なるタイプのネットワークのシミュレーションにも使用できよう。

ハードウェアは、CPU の内部バスのオプションスロットに実装され、隣接の 4台のCPU とそれぞれ接続する為の 4本のケーブルが引出される。これらの引出し口をそれぞれチャネルと呼んでいる。チャネルには送信用、受信用に各々独立した信号線の組が含まれる。データはパリティビットを含め10ビット単位で並列転送され、チャネルの一方向当たりの転送能力は約500kB/sec である。

データは可変長パケットの形で転送され、先頭には行先CPU 番号の情報を持つ。接続ネットワークは行先CPU 番号を認識し、それが自CPU であればデータパケットを取込む。異なる場合には立ちあげ時にプリセットされたCPU 番号と送出先チャネルとの対応表を引いて再送出する。各々のチャネルから到着したパケットがそれぞれ異なる送出先へ再送され、行き先チャネルがぶつかり合わない場合は、転送は同時並列的に実行される。この方式では、横方向優先などの簡単なルーチング戦略を固定的に用いるだけで、デッドロックが回避される。ネットワークとCPU のインタフェース部分には、4k語の容量をもつ送信用、受信用のFIFOバッファを設置しており、また各チャネルの送出部分にも256語のFIFOバッファを置いた。

8．負荷分散の一方式

プロセッサ台数が非常に多い場合にも適用できること、プログラムからプロセッサ台数を意識する必要性が少ないことを重視しつつ、通信の局所性と負荷バランスの制御をプログラマに書かせる立場をとった方式を検討中である。

プログラマには計算パワーが一様に分布した単位平面を想定させてそれを分割しては部分問題に与えることを書かせ、計算パワー平面上で部分問題間の通信の局所性を保つように分割と部分問題の配置を制御させる。この制御はプログラムにpragmaを付加して行なう。一方システムは、計算パワー平面と物理プロセッサとのマッピングを管理し、部分問題をどの物理プロセッサで実行するかを決める。長い間負荷の不均衡が続く場合には、隣接プロセッサとの通信によりマッピング関係を調整し、プログラマが制御しきれなかった負荷バランスを補正する。

9．おわりに

Multi-PSI システムの概要について報告した。現在V1システム用の接続ネットワークハードウェアを製作中であり、ESP ベースの処理系も製作が進んでいる。今後システムの構成要素の各論につき順次報告する予定である。

文献 [1] Murakami,K et al.:Research on Parallel Machine Architecture for F.G.C.S.,Computer,vol.18,no.6,June'85

[2] 西川他： PSIの性能評価(1),第30回情処全国大会,no.1C-2

[3] Ueda,K: GUARDED HORN CLAUSES, Proc.on the Logic Programming Conf.'85,July 1-3,in Tokyo, pp.225

[4] 江崎他： GHCサブセット逐次処理系の作成，本大会，2G-4

3Q-4

高性能Ｐｒｏｌｏｇマシン（ＣＨＩ）の機械語命令先取り方式についての考察

幅田　伸一　　中崎　良成　　梅村　澄

（日本電気（株）　Ｃ＆Ｃシステム研究所）

はじめに　　通産省第５世代計算機プロジェクトの一環として開発中の高性能Ｐｒｏｌｏｇマシン”ＣＨＩ”［１］は、Ｐｒｏｌｏｇで記述したプログラムの高速実行環境の提供を目的としたＰｒｏｌｏｇ専用マシンである。Ｐｒｏｌｏｇプログラムの高速処理を実現するために、コンパイラによる最適化［２］、［３］とアドレス演算・データ演算・命令解読の並列処理［４］を採用している。この結果、大部分の機械語命令は、実行に必要なステップ数が２から４ステップとなり、機械語命令先取りが性能に与える効果は大きくなっている。本稿では、ＣＨＩの研究開発の際に行なった機械語命令先取り方式の性能評価について報告する。

評価項目　　ＣＨＩにおける機械語命令先取り方式の効果を評価するために、キャッシュメモリの構成とパイプラインのステージ数が異なる５種類のハードウェア構成の性能比較を行なった。パイプライン処理の評価では、命令のアドレス演算、命令の読み出し、ＯＰコードの解読とオペランドの切り出し、オペランド・アドレスの演算、命令の実行をサブ処理とし、１個のマイクロプログラム制御系が全体を制御する集中制御方式を対象とする。

キャッシュメモリの構成　　キャッシュメモリを１個とし、データと命令共用にする場合と、データ用と命令用の２個に分割する場合を比較する。

データと命令共用とする場合、ハードウェア規模を小さくでき、制御も簡単である。しかし、命令の先読みによるキャッシュ・アクセスと命令実行によるキャッシュ・アクセスが同時に発生することによる処理性能の低下が予想される。

キャッシュメモリの構成を比較する理由は、キャッシュメモリを２個にすることで得られる性能が、ハードウェア規模の増加、制御の複雑化に見合うかを、評価することにある。

パイプライン・ステージの段数　　ＣＨＩの機械語命令処理は、命令のアドレス演算、命令の読み出し、ＯＰコードの解読、オペランドの切り出し、オペランド・アドレスの演算、命令の実行からなる。機械語命令の先取りによる実行ステップ数の減少と、パイプラインの各ステージで行なう処理の簡単化によるクロック幅の減少が高速化に与える影響は大きい。そこで、次の５個の処理をパイプラインのサブ処理の候補とする。

１）命令のアドレス演算
２）命令の読み出し
３）ＯＰコードの解読とオペランドの切り出し
４）オペランド・アドレスの演算
５）命令の実行

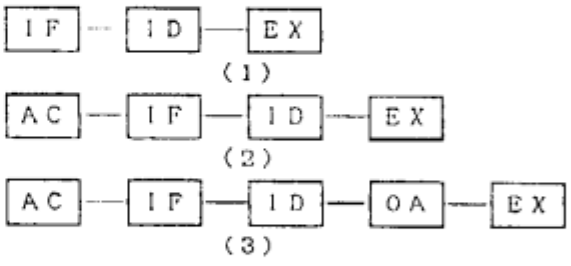
１）から４）のサブ処理は、１ステップで終了する。５）の命令の実行は、命令により必要なステップ数が異なり、２から４ステップ必要である。

ハードウェア規模、性能の２点から、第１図に示す３種類のパイプライン構成を比較する。（１）のパイプライン構成は、ＩＦ，ＩＤ，ＥＸの３個のステージからなる。ＩＦは、サブ処理２）を実行する。ＩＤは、サブ処理３）を実行する。ＥＸは、サブ処理１）、４）、５）を実行する。

（２）のパイプライン構成は、ＡＣ，ＩＦ，ＩＤ，ＥＸの４個のステージからなる。ＡＣは、サブ処理１）を実行する。ＥＸは、サブ処理４）、５）を実行する。ＩＦとＩＤは、（１）のパイプライン構成の場合と同じである。

（３）のパイプライン構成は、ＡＣ，ＩＦ，ＩＤ，ＯＡ，ＥＸの５個のステージからなる。ＯＡは、サブ処理４）を実行する。ＥＸは、サブ処理５）を実行する。ＡＣ、ＩＦとＩＤは、（２）のパイプライン構成と同じである。

ＥＸステージの処理が少ない（３）のパイプライン構成が、最も性能が良いと予想できる。しかし、ハードウェア規模、制御の単純さは、（１）のパイプライン構成が優れている。パイプライン・ステージの段数を比較する理由は、段数を増加することで得られる効果とそのために支払うハードウェア量、制御の複雑さの関係を評価することにある。



第１図　　パイプラインの構成

命令先取り方式　キャッシュメモリの構成とパイプラインの構成の組み合わせから、次の5種のハードウェア構成が考えられる。

a）　キャッシュメモリ　1個、（1）のパイプラインを使用した方式
b）　キャッシュメモリ　1個、（2）のパイプラインを使用した方式
c）　キャッシュメモリ　1個、（3）のパイプラインを使用した方式
d）　キャッシュメモリ　2個、（2）のパイプラインを使用した方式
e）　キャッシュメモリ　2個、（3）のパイプラインを使用した方式

上記5種のハードウェア構成の内で、ハードウェア構成e）が、性能的に、最も優れているのは、明白である。しかし、そのために投入するハードウェア量、制御の複雑さの量を考慮すると、性能の差を明白にする必要がある。

各方式の比較　CHIにおける代表的なユニフィケーション処理用機械語命令の実行ステップ数を比較した結果を第1表に示す。○印は、他の構成より優れていることを示している。△印は、逆に、他の構成より劣っていることを示している。nは、ユニファイ処理の対象である2個のデータのタイプにより、0または、1以上の値となる。

（a）（b）（c）の結果から、キャッシュメモリが1個の場合のパイプライン構成と性能の関係が分かる。まず、（a）だけ性能が悪い機械語命令が12個存在することから、（a）の構成は好ましくない。（a）が（b）（c）に比べて劣る理由は、（b）（c）が、キャッシュメモリ内で命令先取りアドレスの演算を行なうのに対して、（a）では、プロセッサ部の演算器で行なうからである。したがって、命令先取りアドレス演算機能をキャッシュメモリ内に備える必要がある。

（b）と（c）の比較より、オペランド・アドレス先行演算による改善効果が小さいことが分かる。したがって、キャッシュメモリ1個の場合、オペランド・アドレス先行演算を導入する必要がない。同様に、（d）（e）の結果から、キャッシュメモリ2個の場合のパイプライン構成と性能の関係が分かる。キャッシュメモリ1個の場合より、オペランド・アドレス先行演算の効果が大きいことが分かる。これは、オペランド・アドレス先行演算によりキャッシュメモリの未使用ステップが減少したことが原因である。

キャッシュメモリの個数による影響は、（b）と（d）、または、（c）と（e）を比較することで分かる。（b）と（d）の結果から、オペランド・アドレス先行演算を行なわない場合は、キャッシュメモリの未使用ステップが多いため、性能が極端に改善されてはいないことが分かる。しかし、オペランド・アドレス先行演算を行なう（c）と（e）の比較では、個々の性能改善効果が加わった結果、性能の改善効果が大きくなっている。

以上から、オペランド・アドレスの先行演算と2個のキャッシュメモリは、単独では、大きな効果が期待できず、方式（e）の様に、オペランド・アドレスの先行演算と2個のキャッシュメモリを組み合わせる必要がある。したがって、ハードウェア構成としては、ハードウェア構成が比較的簡単な（b）の構成が、命令の先行処理が比較的多い（e）が適している。

まとめ　各命令先取り方式について、CHIにおける機械語命令単位での実行ステップ数の比較を行なった。この結果、中途半端な命令先取りは、あまり効果がないことが分かった。今後、さらに高性能なマシンを実現するため、ベンチマーク・プログラムによる性能評価を行なっていく予定である。

参考文献
[1] Nakazaki et al.:"Design of a High-Speed Prolog Machine"　Computer Architecture,1985
[2] 中嶋　他：高性能ＰＲＯＬＯＧマシン：ＨＰＭ　－　基本アーキテクチャと処理方式　－　情報処理学会第３０回全国大会
[3] Warren.D.H.:"An Abstract Prolog Instruction Set"　Tech.report309,AI Center,SRI International,1983
[4] 幅田　他：高性能ＰＲＯＬＯＧマシン：ＨＰＭ　－　ハードウェア構成　－　情報処理学会第３０回全国大会

	a	b	c	d	e	
put temporary variable	2	2	2	2	2	
put permanent variable	2	2	2	2	(1)	
put temporary value	△	1	1	1	1	
put permanent value	3	3	(2)	3	(2)	
put list	△	1	1	1	1	
put structure	△	2	2	2	2	
get permanent variable	2	2	2	2	(1)	
get temporary variable	△	1	1	1	1	
get permanent value	3 +n	3 +n	(2) +n	3 +n	(2) +n	
get temporary value	△ +n	1 +n	1 +n	1 +n	1 +n	
get list	△	1	1	1	1	
get structure	3	3	3	3	3	
unify permanent variable	3 3	3 3	3 3	3 (2)	3 (2)	read mode write mode
unify temporary variable	2 2	2 2	2 2	(1) (1)	(1) (1)	read mode write mode
unify permanent value	△ +n △	3 +n 3	3 +n 3	3 +n 3	3 +n 3	read mode write mode
unify temporary value	3 +n △	3 +n 2	3 +n 2	(2) +n 2	(2) +n 2	read mode write mode
unify list	△ △	1 2	1 2	1 2	1 2	list の場合 変数の場合
unify structure	3 △	3 3	3 3	3 3	3 3	structure の場合 変数の場合

第1表　実行ステップ数の比較

2G-6 高性能ＰＲＯＬＯＧマシン（ＣＨＩ）における最適化技法

小長谷　明彦　　中嶋　良成　　幅田　伸一　　梅村　護
日本電気株式会社　Ｃ＆Ｃシステム研究所

1．はじめに

近年、新しいＰｒｏｌｏｇのコンパイル方式としてＷａｒｒｅｎの命令セット[1]が注目を集めている。本稿では同方式をベースとした高性能ＰｒｏｌｏｇマシンＣＨＩ[2]における最適化技法をそのコンパイル特性と合わせて報告する。

2．Ｗａｒｒｅｎの命令セット

Ｗａｒｒｅｎの命令セットは仮想的なＰｒｏｌｏｇマシンの命令を定義したものであり、ユニフィケーション命令（ｐｕｔ，ｇｅｔ，ｕｎｉｆｙ）、ゴール呼び出し命令（ｃａｌｌ，ｅｘｅｃｕｔｅ，．．．）、インデクシング命令（ｔｒｙ，ｒｅｔｒｙ，ｔｒｕｓｔ，．．．）より構成される。ＣＨＩの命令セットもこれに基づいている。Ｐｒｏｌｏｇのクローズはこれらの命令の列に展開され、以下の最適化が可能である。

（１）インデクシング
（２）終端再帰呼び出しの最適化
（３）レジスタに割り当てられた変数のアクセス命令の最適化

3．インデクシング

インデクシングはクローズの特定引数に着目してクローズをあらかじめ分類しておき、冗長なバックトラッキングを防止する最適化手法である。この分類基準としてＣＨＩでは現在以下のものを用意している。

（１）データ型（変数、リスト、構造体、アトミック・データ）による分類
（２）アトミック・データの値による分類
（３）構造体のファンクタの値による分類

データ型による分類に関する調査結果を表１、表２に示す。調査対象はＣＨＩ用に作成されたプログラム（資源管理プログラム、インタプリタ、コンパイラ等）である。平均的な手続きのクローズ数は２．７であった。表１より、アトム、構造体を扱うクローズは平均的に長く、変数とリストを扱うクローズは短い。これより、上記の分類は概ね適切であることがわかる。ただし、表２をみると、全てのインデクシング引数が変数であるクローズが非常に多い。この傾向はベクトル等リテラルとして現われないデータ構造を多用する資源管理プログラムにおいて特に顕著となる。このような手続きをよく調べると、

ｐ（Ｘ）：－ｖａｒ（Ｘ），！，．．．
ｐ（Ｘ）：－ｉｎｔｅｇｅｒ（Ｘ），！，．．．

のように変数で受けて、タイプチェックを行なっている例が非常に多い。したがって、変数の型を含めた分類を行なうとより効果的なインデクシングが実現できると予想される。

表３はアトミック・データのインデクシングに関して、インデックス・テーブルを逐次的に検索する場合と、ハッシュして検索する場合の実行時間を比較したものである（ＶＡＸ７８０上でＣで記述されたＣＨＩ仮想語エミュレータを使用）。表より明らかなように、ハッシュ検索と逐次検索の境は少なくデータベースの先頭の方のクローズの検索はかえって逐次検索の方が早い。これはハッシュ値の計算や衝突の処理にオーバーヘッドがかかったためと考えられる。クロスポイントはデータにもよるが１０～２０であり、ＣＨＩのようにアトム番号をハッシュ値とせず、アトム名から計算したハッシュ値をアトム番号とは別に持つようなシステムではクローズ数が少ない場合は逐次検索を用いた方が効果的である。また、コンパイル時間は逐次検索の方がハッシュ検索よりも２～２．５倍早い。

4．終端再帰呼び出しの最適化

Ｐｒｏｌｏｇの最後のゴールの呼び出しは制御を戻す必要がないためサブルーチン呼び出し（ｃａｌｌ）ではなく、ジャンプ命令（ｅｘｅｃｕｔｅ）で実現することができる。このためには、ゴール引数の退避が必要であり、Ｗａｒｒｅｎ命令セットではゴール呼び出しの前にゴール引数を引数レジスタにセットし（ｐｕｔ命令）、呼ばれた側では引数レジスタの値とユニフィケーションする（ｇｅｔ命令）ことによりこれを実現している。引数レジスタを用いることにより以下の利点が生じる。（１）ユニットクローズ、本体部にゴールが１つしかないクローズ（以下トランジェント・クローズと呼ぶ）では変数領域をスタック上に確保する必要がない。（２）引数レジスタを用いたアクセスの最適化（次節）ができる。このうち、性能的には前者の効果が大きい。実際、Ｗａｒｒｅｎの命令セットを用いると他のベンチマークプログラムに比べ、ａｐｐｅｎｄプログラムが非常に速い[３]が、これは、引数型によるインデクシングの効果とともにトランジェント・クローズであるため制御情報を格納するためのメモリアクセスが全く不要となる効果が大きい。

表４はＣＨＩインタプリタのクローズのタイプを

調べたものである。カット等の組込述語を通常のゴールと同じように扱うと終端再帰呼び出しの最適化の効果はあまり期待できないが、組込述語をインライン展開するとトランジェント・クローズが非常に多くなることがわかる。組込述語のインライン展開は変数分類を複雑にするが［３］、Ｐｒｏｌｏｇの場合は単なるインライン展開の効果以上の高速化が期待できる。

５．一時変数の最適化

値領域をローカルスタック上にとらずに済む変数は一時変数と呼ばれる。この値領域を引数レジスタ上にとるとユニフィケーション命令が省略できる場合があり、これは一時変数の最適化と呼ばれる。

単純な例としては、

ｐ（Ｘ，Ｙ）：－ｑ（Ｘ，ａ）．

のＸのようにヘッドゴールと同一位置にある変数やヘッドゴールに現われるボイド変数．

ｐ（＿，ａ）．

があるが、レジスタのライフタイムを調べることにより、より高度な最適化が可能である。例えば、

append([XiY],Z,[XiYZ]):-append(Y,Z,YZ).

では、Ｚに対するユニフィケーション命令（ｇｅｔ命令、ｐｕｔ命令）だけでなく、本体部のＹ，ＹＺの引数レジスタへのセット命令（ｐｕｔ命令）も省略できる［２］。

一時変数は任意のゴールに存在するが最適化可能な一時変数はヘッドゴールと本体部の第一ゴールのトップレベルに現われる最初の出現に限られる。現在ＣＨＩのコンパイラで行なっている一時変数の最適化条件を次に示す。

１）ヘッドゴールでのｇｅｔ命令の省略
- ①ヘッドゴールのみに出現する変数
  例　ｐ（Ｘ，Ｙ，［Ｙ］）．
- ②本体部での出現より後ろにある変数
  例　ｐ（Ｙ，ａ，Ｘ）：－ｑ（Ｘ，Ｘ，Ｙ）．
- ③同一位置に本体部の出現がある変数
  例　ｐ（ａ，Ｘ，ｂ）：－ｑ（Ｘ，Ｘ，Ｘ）．

２）本体部でのｐｕｔ命令の省略
- ①ヘッドゴールでの出現と同じか前にある変数
  例　ｐ（Ｘ，Ｙ，Ｘ）：－ｑ（Ｘ，Ｙ）．
- ②ヘッドゴールでの同一位置にボイドがある変数
  例　ｐ（Ｘ，＿）：－ｑ（ｂ，Ｘ）．
- ③ヘッドゴールの引数より後に出現する変数
  例　ｐ（Ｘ，Ｙ）：－ｑ（ａ，ｂ，Ｘ）．

一時変数の最適化は様々なパターンがあるが、省略できる命令はレジスタ転送命令のため、実行性能に対する効果はあまり大きくなく、ａｐｐｅｎｄの例でも１０％以下でしかない。

６．まとめ

Ｐｒｏｌｏｇにおける最適化の基本はいかに冗長なメモリアクセスを減らせるかにあり、インデクシング、終端再帰呼び出しの最適化の利点もバックトラック情報や制御情報を格納するフレームの生成を省略できる点にある。最適化に関する今後の課題としては変数の型を含めたインデクシングや、本稿では述べなかったが、冗長なカットの除去やクローズ内ＯＲの効率的な実現がある。ただし、これらを実現するにはソースプログラムの解析だけでは不十分な点があり、より効果的な最適化を実現するためには変数の型宣言や引数の入出力指定等の導入が必要と思われる。

最後に、本稿を記述するにあたって種々のデータを収集してくれた神戸日本電気ソフトウェア株式会社の阪野氏に感謝いたします。

参考文献

[1] Warren :"An Abstract Prolog Instruction Set" TR309, AI Center, SRI International, 1983
[2] Nakazaki etal.:"Design of a High-Speed Prolog Machine"Inin'l. Symp. on Comp. Archi. 1985
[3] Roy P. V.:"A Prolog Compiler for the PLM" UCB/CSD 84/203, Berkeley 1984

表１　データ型によるインデクシング後の型別平均クローズ数（単純平均クローズ数２．７）

変数	１．７
構造体	１２．３
リスト	１．１
アトミック	６．７

表２　クローズのインデックス引数の手続き内での出現パターン

全て変数	７４％
全て非変数項	１２％
変数／非変数項	１％
非変数項／変数	１２％
それ以外	１％

表３　データベース検索問題（２２０クローズ）におけるインデクシング方式の実行速度比較

	ハッシュ検索	逐次検索	非決定的検索
先頭	０．９	０．５	０．９
中間	１．３	１．５	４３．５
最後	１．０	２．６	８４．３

表４　インタプリタのクローズ型の比率

組込述語展開	無し	有り
ユニット	５％	５％
トランジェント	１８％	５９％
その他	７７％	３６％