AI 86－13

関係データベースを用いた論理型言語実行系の試作と検討

吉田　　敦・鈴木寿和
大森　　匡・田中英彦
（東　　大）

1986年6月24日

社団法人 電子通信学会

関係データベースを用いた論理型言語実行系の試作と検討

An Experimental Logic Programming Language Interpreter
Using Relational Data Base

吉田敦・鈴木寿和・大森匡・田中英彦

Atsushi Yoshida Toshikazu Suzuki Tadashi Ohmori Hidehiko Tanaka

東京大学 工学部

Faculty of Engineering University of Tokyo

1 はじめに

自動翻訳システム，エキスパート・システムなどの構築には，大量の知識を効率良く扱える知識ベース・システムが不可欠である。このようなシステムでは，問題解決に必要な知識の量が数百ないし，数万のオーダーになることが予測される。このため知識の探索に要する時間が処理性能に大きな影響を与えるようになる。そこで，大量の知識の中から必要なものを効率良く取り出してくる機構が必要になる。このような機構として，関係データベースマシンは有力な候補である。本研究では，関係データベースマシンの環境で推論機能を実現する知識ベースマシンの構築方法を検討する。本論文では，このためのアプローチの第一歩として，Horn節を扱う推論機能を関係論理を用いて実現する方法について検討と考察を行う。本論文の構成は，次のようになっている。第2章では，関係データベースと論理型言語の関係について述べ，第3章では，関係データベースを用いた推論機能の実現方法について述べる。第4章では，関係データベース上で実現されたPrologインタプリタ（RDB－Prolog）について述べ，第5章で，これに対する考察と検討を行う。第6章では，まとめと今後の課題をあげる。

2 関係データベースと論理型言語

知識の表現形式には，一階述語論理，プロダクション・ルール，フレーム，意味ネットワークなど，様々なものがある。知識ベースマシンは，知識処理におけるデータベースマシンとして位置付けられるものであり，その意味で，様々な知識を統一的に扱う機能が必要になる。本研究の最終目標は，複数種類の知識を統一的に扱う機能を有する知識ベースマシンの実現であるが，そのような機能を実現するためには，まだ，考えるべきことが数多くある。そこで，本研究では，第一ステップとして，関係データベースと相性の良い一階述語論理を扱う知識ベースマシンの実現方法を考えることにする。

2．1 関係データベース

関係データベースはE．F．Coddが提案した関係モデルに基づくデータベースである。関係データベースにおける処理単位は，いくつかの値の組であるタプルの集合，つまり，関係である。関係に対する基本演算として，射影，選択，結合があり，このほかに，集合和，集合積，直積などの操作や，関係に対する更新操作が関係データベースにおける操作として実装される。知識ベースマシンの中核となるデータベースに，関係データベースを選んだ理由は，次のとおりである。

関係に対する演算は，集合指向の操作であり本質的な並列性をもっている。この並列性を，知識に対する操作が本来持っている並列性の実現に利用できる。

関係モデルや論理型言語は，その数学的基礎が確立されている。また，関係の各タプルは，論理型言語におけるファクトに，関係に対するビューは，ルールに対応するといったように，

両者の間には強い親和性が存在する。また，Ｐｒｏｌｏｇにおける推論処理は，関係デ―タベ―スにおける探索および更新と本質的に同じである。

関係デ―タベ―ス実装技術・効率化の手法の研究は，現在までに充分な研究が行われている。これらの成果を，知識ベ―スの効率良い実装に活かすことができる。

例えば，関係演算の効率化は，ハッシュとソ―トにより関係を分割し，各分割要素毎に関係演算を実行することにより実現できる。この方法を支援するハ―ドウェアは〔１〕で提案されており，このア―キテクチュアを用いて知識ベ―スを構築できる。また，質問処理の前に，質問に対する処理に関連するグラフを考え，このグラフの解析をとおして質問処理の最適化を行うといったような方法が効率化の手法として考えられる。

２．２　論理型言語

知識ベ―スにおける知識表現形式には，様々なものがある。これらの知識を統一的に扱うためには解決するべき問題が数多くある。例えばオブジェクト指向の考え方を導入し様々な知識表現形式をクラス，各知識をクラスのインスタンスとして扱い，知識に対する処理は，メッセ―ジのやりとりで実現し，その中で，関係演算を用いた処理をするなどの方法が考えられる。本研究では，様々な知識を統一的に扱う知識ベ―スマシンを関係デ―タベ―スマシンを基本として構築するのが目的であるが，この目的を実現する第一歩として，一階述語論理のサブセットであるＨｏｒｎ節を知識表現として扱うことにした。その理由として，２．１で述べたことの他に，以下のようなことがあげられる。

論理型言語は，言語としての歴史が浅いため，その仕様はかなり流動的である。従ってエキスパ―ト・システムや言語理解システムといった応用の面からの仕様に対する要求を反映させる余地および，必要がある。また，従来のエキスパ―ト・システムや言語理解システムの大部分が，慣習的にＬＩＳＰやプロダクション・システムを用いており，論理型言語（またはその拡張）を用いた例は数少ない。従って，論理型言語で記述された応用システムの研究が早急に望まれる。

３　関係デ―タベ―スを用いた知識ベ―ス

関係デ―タベ―スを知識ベ―スの構成要素として用いる場合，２通りのアプロ―チが考えられる。１つは，関係デ―タベ―スに，これとは独立な推論機構を接続する方法，もう１つは，関係デ―タベ―スの環境下で推論機能を実現する方法である。ここでは，前者を結合アプロ―チ，後者を融合アプロ―チと呼ぶ。

３．１　結合アプロ―チ〔２〕

知識ベ―スは，推論機構と関係デ―タベ―スの２つの要素から成るという発想に忠実に，それぞれ独立した推論機構と関係デ―タベ―スを通信チャネルを介して接続する方式である。（図１－ａ）

推論機構側では，内部データベースに格納されたル―ルを用いて，質問の展開を実行し，必要に応じて，関係デ―タベ―スにコマンドを転送して処理を実行してもらう。この方式では，ル―ルが推論機構側に，ファクトが関係デ―タベ

図１　知識ベ―スマシン・アーキテクチュア

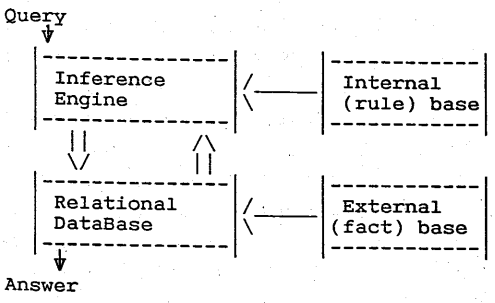


図１－ａ　結合アプローチ

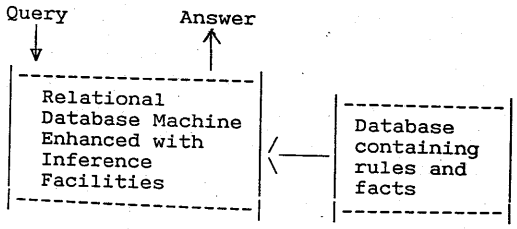


図１－ｂ　融合アプローチ

―ス側にあるため，２つのユニットの間で通信が必要である。このようなシステム構成を取ると，通信のオ―バヘッドが問題になってくる。通信のオ―バヘッドを回避するためには，通信の回数を極力減らす必要があり，推論処理方式が極めて制限される。極端な場合には，ル―ルを用いた，ファクト系列への展開だけを，推論機構側で先に実行し，ファクト系列から関係演算を生成して，関係デ―タベ―スにコマンドとして送ったあと，関係デ―タベ―スで答を生成させる方法を取る。

３．２　融合アプロ―チ

関係デ―タベ―スという一つの環境で，推論機能の実現をめざすアプロ―チである。推論機構として特殊なハ―ドウェアを必要とはしないのが特徴である。（図１－ｂ）推論処理はすべて，関係演算を用いて記述され，定義節の探索や，コンシステンシ―・チェックは関係演算により並列的に実行される。ル―ル，ファクトはともに関係デ―タベ―スに格納され，これらに対する処理はすべて関係デ―タベ―スの中で実行されるので，通信のオ―バヘッドは考える必要がない。

３．３　融合アプロ―チの利点

２つのアプロ―チを比較すると，融合アプロ―チの方が，以下の点で優れている。

融合アプロ―チは，通信のオ―バヘッドを考えない分，推論処理方式に対する選択の柔軟性が高い。従って，結合アプロ―チに比べると，より徹底した最適化が可能である。また，融合アプロ―チで考えるシステムの基本的ア―キテクチュアは，関係デ―タベ―スマシンのア―キテクチュアに基づくので，関係デ―タベ―スの実装や効率化の技術が結合アプロ―チにくらべて適用しやすい。

結合アプロ―チは，従来までは，演繹データベースに関する研究の主流であったが，通信オ―バヘッドと推論処理の効率化とのトレ―ド・オフによる限界がほぼ明らかになってきた。それに対して融合アプロ―チは，これまでほとんど研究されていないため，様々な検討や考察を行う余地が充分にある。

４　ＲＤＢ－Ｐｒｏｌｏｇ処理系の試作

本研究では，融合アプロ―チに基づいた知識ベ―スの第１版として，関係論理を用いて記述されたＰｒｏｌｏｇインタプリタ〔３〕を試作した。試作に用いたデータベースは，ＩＮＧＲＥＳ　Ｖｅｒ７．１０で，インタプリタは，ＩＮＧＲＥＳに対する操作を行うプログラムとして，Ｃ言語で記述した（図２参照）。本章では，

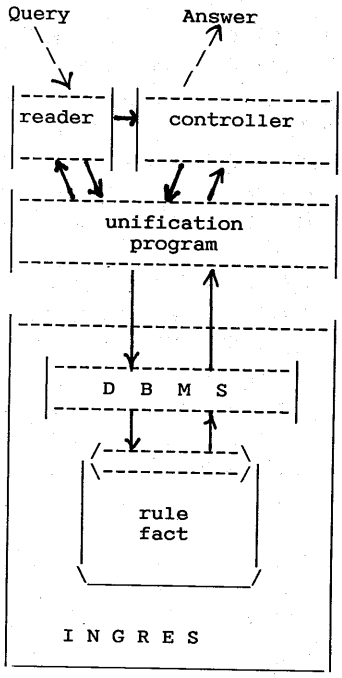


図2　Ｐｒｏｌｏｇインタプリタ

このインタプリタについて説明する。

４．１　仮定

関係デ―タベ―スを用いた，Ｐｒｏｌｏｇインタプリタの第１版では，関係モデルの拡張は極力行わない方針をとった。関係論理だけの範囲での記述力を評価し，充分な記述力を得るには，関係デ―タベ―スに対してどのような拡張を行えばよいかを見るためである。従って，デ―タ・タイプとして，変数は扱うが，ファンクタは，ｌｉｓｔ以外は扱わないことにした。実際のプログラムをみても，ファンクタとして使われるものの大半がｌｉｓｔであり，残りのファンクタも，大部分が，ｌｉｓｔの形で表現可能であり，実用的には問題がない。　ｌｉｓｔは，処理の時に二進木に展開して扱うようにした。（展開部分は関係演算の範囲外になる）

このシステムでは，ルールも，ファクト共に関係データベースの中で扱うので，ファクトだけでなくルールも関係のタプルとして表現する。変数は，Dec-10 Prologの記法に習い，頭一文字が大文字の文字列として表現するが，実際には，タグを用いる表現が望ましい。

4．2　ルールの表現形式

ファクトは，各引数に対して属性を割り当てることにより，関係のタプルとして表現可能である。ここでは，ルールの表現形式について述べる（図3参照）。

ルールの表現で最も問題となるのは，body部のリテラルの数が一定でないことである。これに対する対策には，次のようなものが考えられる。

1）単一化の対象となるhead部に対しては各引数毎に一つの属性を割当てて格納する。body部に対しては属性は一つとし，body部全体を文字列で表現する。

2）body部はリテラル毎に分割し，リテラル毎に一つの関係に格納する。この時，同じルールに属するhead部リテラル，及び，body部各リテラルがhead部リテラルより参照可能になるようにポインタで繋ぐ。

ここでは，ゴールとbody部リテラルの扱いが統一的になる2）の方を採用した。

リテラル間の参照を可能にする方法には，次のようなものが考えられる。

a）head部に対して，body部最左リテラルの述語名，キー，アリティを属性としてもたせ，これをポインタとして使うようにする。body部の各リテラルに対しても，次に来るリテラルの述語名，キー，アリティを属性としてもたせる。

b）head部リテラルの格納された関係に，その節のリテラルの数を属性として追加する。別の関係に，各節毎にbody部リテラルの数だけ節番号とリテラルの述語名，キー，アリティを値とするタプルを格納する。

前者は，逐次型のインタプリタの実現に用いる。後者は，関係演算の持つ並列性を利用したAND並列とOR並列を実現するインタプリタの実現に用いる。

図3　Horn節の表現

```
friend(X,Y) :- friends(X,Y).
friend(X,Y) :- friends(Y,X).
friends(john,mary).
friends(john,george).
friends(hary,george).
friends(hary,john).
friends(edward,mary).
```

図3－1　Horn節

friend

pnum	term1	term2	nextpred	nextpn	nextar
1	X	Y	friends	1	2
2	X	Y	friends	2	2

friends_g

pnum	term1	term2	nextpred	nextpn	nextar
1	X	Y		0	0
2	Y	X		0	0

friends

pnum	term1	term2	nextpred	nextpn	nextar
1	john	mary		0	0
2	john	george		0	0
3	hary	george		0	0
4	hary	john		0	0
5	edward	mary		0	0

図3－2　表現形式1

friends

pnum	term1	term2	pand
1	john	mary	0
2	john	george	0
3	hary	george	0
4	hary	john	0
5	edward	mary	0

friend

pnum	term1	term2	pand
1	X	Y	1
2	X	Y	1

friend_b

pnum	bpred	bnum	barity
1	friends	1	2
2	friends	2	2

friends_g

pnum	term1	term2
1	X	Y
2	Y	X

図3－3　表現形式2

4．3　束縛の表現

単一化実行後の環境を表す情報（束縛情報）も関係で表現する。

変数名の有効範囲は，各節の定義の中だけである。つまり，同じ変数名に対して同じ識別番号が与えられるのは，同じ定義節の中だけであり，異なる定義節の中では，同じ変数名でも異なる識別番号を与える。また，同じ定義節であっても，別の導出サイクルで呼び出された場合には，同じ変数名に対して異なる識別番号を与える。従って，変数を区別するためには，定義節の識別番号，探索木の深さ，変数名の3つ組が必要になる。

変数に代入されるものとしては，定数，ファンクタ，他の変数がある。定数の代入を表現するには，束縛を表す関係に，定数の値に対する属性を設け，そこに値を代入すればよい。ファンクタについては，基底項のみから成るものについては定数と同じ扱いでよい。変数を含むファンクタについては，変数を取り出す操作と，取り出した変数を束縛に追加するかどうか判定し，必要なら追加する操作が必要になる。他の変数が代入される場合は，変数識別番号の属性に，代入される変数の識別番号を代入する。図4は，束縛を表す関係の例である。

environment

pnum	level	vname	id	const
1	0	X	1	hary
1	1	X	2	john
1	1	Y	3	hary

variable

pnum	level	vname	id	const
1	0	X	1	_1
1	1	X	2	john
1	1	Y	3	_1

term

pnum	pconst	gconst
1	mary	_1
2	george	_1
3	george	_1
4	john	_1
5	mary	_1

図4　束縛の表現

4．4　ファンクタの表現

通常のＰｒｏｌｏｇインタプリタにおいてファンクタを表現する場合，ボ―ランド記法で，スタックに積む要領で連続領域にデ―タを格納するレコ―ド方式と，デ―タをセルに格納し，セルをポインタで繋ぐリスト方式が考えられる。

いずれの方式も，可変長の記憶領域が必要になるが，後者の方法では，関係デ―タベ―スにポインタの概念を大幅に持ち込むことになり，あまり，好ましくない。従って，レコ―ド方式を採用した。ここでは，関係の各タプルは，属性フィ―ルドとして，可変長フィ―ルドを使用できると仮定している。ファンクタは文字列の形で格納し，単一化などの際に展開して用いることにした。

4．5　Ｐｒｏｌｏｇインタプリタ

ここでは，最初に，線形導出のみを実行し，バックトラックは行わないものを試作した。これにより，関係演算でＰｒｏｌｏｇにおけるゴ―ルの導出が記述できることを確認した。次に関係演算が本質的に持つ並列性を活かし，ＯＲ並列とＡＮＤ並列を実行するＰｒｏｌｏｇインタプリタを試作した。ＯＲ並列は，サブゴ―ルの導出を関係演算により集合操作的に行うことで実現できる。ＡＮＤ並列は，ゴ―ルの各リテラルに対する導出を並列に実行し各導出の結果に対する関係のｊｏｉｎにより，コンシステンシ―・チェックを実行すればよい。

4．5．1　線形導出の実現

Ｐｒｏｌｏｇにおける推論操作は単一化である。これが，関係演算への展開が可能かどうかを確認する目的で，ゴ―ルに対する導出を関係論理（記述力は，関係代数と等価である）で記述してみた。ここでは，１つのゴ―ルから複数のサブゴ―ルが導出されてもその中の１つしか実行しない。そのアルゴリズムは次のようになる。

ゴ―ルが空節でない間，リテラルを１つ選択し，以下の処理を実行する。

リテラルの各引数に対し，以下の処理を実行する。

各項の間の単一化をテストし，成功すれば，代入を実行，失敗したら全体を失敗させる単一化のアルゴリズムについては，後で述べる。

4．5．2　並列実行のシミュレータ

ゴールの集合が空になるまで，以下の処理を実行

ゴールの1つを選択し，そのゴールに対する単一化を実行

単一化で得られた各ゴールに対し，それぞれ次の処理を実行する。

各リテラルに対してそれぞれ単一化を実行し，リテラルでの単一化が失敗したゴールは放棄する。すべてのリテラルに対して単一化が成功したものについては，各リテラルに対する単一化の結果の，ｊｏｉｎをとって，コンシステンシー・チェックを行う。ここで失敗したゴールも放棄する。

各ゴールに対する結果の和集合を取る。

4．5．3　単一化の実現

・リテラルに対する単一化

ゴールを形成する各リテラルは，関係〔述語名〕－ｇの中に格納し，ｈｅａｄ部リテラルは，関係〔述語名〕の中に格納する。後者のコピーを作り，各引数に対し，項の単一化を実行する。項の単一化に失敗したリテラルは，即，消去される。

残りは，単一化可能なリテラルだけとなる。

・項の単一化

ｈｅａｄ部リテラルの問題としている項に対して以下の処理を実行する。

問題となる項のうち，定数となるものを取出し，関係ｔｅｒｍの属性ｐｔｅｒｍの値として格納する。

問題となる項が，初めて出現した変数なら関係ｖａｒｉａｂｌｅへ格納する。

問題の項がその他の変数なら，関係ｖａｒｉａｂｌｅからその変数の属性ｃｏｎｓｔの値を取出し関係ｔｅｒｍの属性ｐｔｅｒｍへ格納する。

次に，ゴールリテラル側の項に対し，以下の処理を実行する。

関係〔述語名〕－ｇの中の問題となる項が定数ならそれを関係ｔｅｒｍの属性ｇｔｅｒｍへ格納。初めて出現する変数なら関係ｖａｒｉａｂｌｅへ格納する。その他の変数なら，関係ｖａｒｉａｂｌｅの中から属性ｃｏｎｓｔの値を取出し関係ｔｅｒｍの属性ｇｃｏｎｓｔへ格納する。

env1

pnum	level	vname	id	const
1	0	X	1	goerge
1	1	X	2	john
1	1	Y	3	_3
2	0	X	1	_1
2	1	X	2	john
2	1	Y	3	_1

env2

pnum	level	vname	id	const
1	0	X	1	hary
1	1	X	2	john
1	1	Y	3	edward
2	0	X	1	_1
2	1	X	2	_2
2	1	Y	3	mary

env

pnum	level	vname	id	const
1	0	X	1	george
1	0	X	1	hary
1	1	X	2	john
1	1	Y	3	edward
2	0	X	1	hary
2	1	X	2	john
2	1	Y	3	hary
2	1	Y	3	edward
3	0	X	1	hary
3	1	X	2	john
3	1	Y	3	edward
3	1	Y	3	hary
4	0	X	1	mary
4	1	X	2	john
4	1	Y	3	mary

ans

ans	ans1	ans2
1	1	1
2	1	2
3	2	1
4	2	2

図5　コンシステンシーチェック

ｐｔｅｒｍ，ｇｔｅｒｍがともに定数で，異なる値なら，単一化不可能であるので，ｔｅｍｐよりそのリテラルを消去する。ｐｔｅｒｍが定数で，ｇｔｅｒｍの値が変数の場合は，関係ｖａｒｉａｂｌｅの属性ｃｏｎｓｔの値が，ｇｔｅｒｍの値に等しくなるすべてのタプルの属性ｃｏｎｓｔの値をｐｔｅｒｍの値に置き換える。

ｐｔｅｒｍの値が変数なら関係ｖａｒｉａｂｌｅの属性ｃｏｎｓｔの値が，属性ｐｔｅｒｍの値と等しいすべてのタプルの属性ｃｏｎｓｔの値をｇｔｅｒｍの値におきかえる。関係ｖａｒｉａｂｌｅには束縛の情報に格納される。

４．５．４　コンシステンシ―・チェック

２つの環境に共通な変数に，矛盾する値が代入されていないかどうかを確認する操作がコンシステンシ―・チェックである。これは２つの環境から新しい環境を作るための対応表ａｎｓを作り，この表を基に，結合操作を実行することで，実現される（図５参照）。

５　検討と考察

５．１　関係モデルの拡張

ここで試作したＰｒｏｌｏｇインタプリタでは，変数やファンクタの表現に文字列を使用していた。従って，単一化や代入の際に，文字列の展開が必要であった。通常の関係デ―タベ―スではタプルの値として定数しか扱わないからである。変数やファンクタを効率良く扱うためには関係モデルに対する拡張が必要になってくる。拡張の案の１つとして，型のチェックを効率良く実行するために，タグ付の値の導入が考えられる。また，ファンクタの表現を効率良く実現するためには，タプル値フィ―ルドを可変長レコ―ドにする必要がある。また，ファンクタ導入に際しては理論上の問題点，例えば，正規形の問題も考える必要がある。

関係デ―タの拡張に伴い，関係演算も拡張が必要になる。変数やファンクタの導入により，結合や挿入における条件は，単なる等価性ではなく，単一化可能性に拡張されなければならない。条件のチェックは，タプルの値を木構造に展開して，再帰的に木構造をチェックする形になる。また，単一化可能な値同士では代入が行われる。このように拡張された関係演算（単一化結合，単一化選択，射影）を，拡張関係演算（ＥＸＴｅｎｄｅｄ　Ｒｅｌａｔｉｏｎａｌ　Ａｌｇｅｂｒａ）と呼ぶ。

５．２　質問の前処理

Ｐｒｏｌｏｇにおける推論処理は，リダクションである。これは，基本的には，データベースの探索と更新で記述できる。但し，関係演算でリダクションをそのまま記述すると，関係演算以外の部分もかなり含まれるので，純粋に関係演算のみで実行するのに比べると効率は落ちる。従って，関係演算の系列に展開できる部分は，質問の前処理で検出し，関係演算の系列に展開する方法をとる。この方法では，再帰的定義に対する展開が問題になる。再帰的定義に対する質問木の最適化には，部分的な解決策がいくつか提案されており，これらを統合して扱うような方法を考える必要がある。

関係演算の系列で質問処理を実行する方法は関係デ―タベ―スの環境下で，質問処理を行う方法としては効率が良い。その反面，再帰的定義の処理が苦手である。これに対して，リダクションを関係演算を用いて記述する方法は，再帰的定義の処理はできるが，効率の点で関係演算のみで処理を行う方法に劣る。質問の前処理ではこの２つの方法を使い分けるような実行方法を生成する。

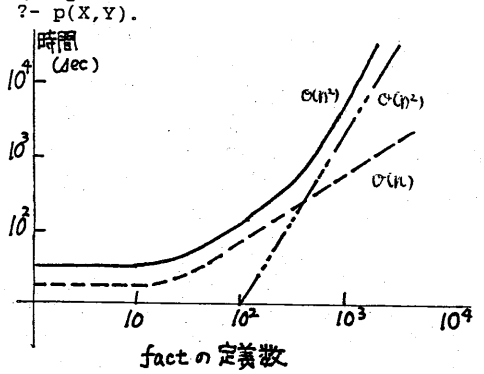
５．３　マシン・ア―キテクチュア

５．１でのべた拡張関係演算を高速に実行するには，ハッシュとソ―トを用いたアルゴリズムが有効である。例えば，結合操作は普通に実行した場合の処理の手間は，関係のタプル数の積に比例する。これをハッシュとソ―トによりクラスタに分割し，各クラスタ毎に独立に処理すれば，処理実行の手間は，２つの関係のタプル数の和に比例することが知られている。図６は関係デ―タベ―スとして通常のデータベースを用いる場合と，関係デ―タベ―スマシンを用いる場合の比較である。例題中のファクトの定義数を変更して，ファクトの定義数に対する処理時間の変化を測定してみたものである。図の実線が，通常の関係デ―タベ―スシステム，破線が関係デ―タベ―スマシン，二点鎖線がＣＰｒｏｌｏｇである。この例題では，質問処理を関係演算で実行した場合，関係演算の回数は，結合３回，選択４回，射影５回が実行される。図

より判るとおり，関係データベースマシンを使用しないと，CPrologで実行する場合より速くなるには，ファクトの定義数がかなり大きい場合に限定される。従って，関係データベースマシンを使用しないと現実的な範囲での有効性が出ない。上で述べたアルゴリズムを支援

```
definitions
p(X,Y) :- p1(X,Z),p2(Z,Y).
p1(X,Y) :- q1(X,Z),q2(Z,Y).
p2(X,Y) :- q3(X,Z),q4(Z,Y).
factp(q1(_,_)).
factp(q2(_,_)).
factp(q3(_,_)).
factp(q4(_,_)).

query
?- p(X,Y).
```



———— INGRES

－－－－－ 関係データベースマシン

－・－・－ CProlog

図6　処理性能曲線

(ファクトの定義数に対する処理時間)

する関係データベースマシン・アーキテクチュアとしては，例えば，〔1〕のようなものが考えられている。このようなアーキテクチュアの中で質問処理を行う方が，独立した推論機構と関係データベースをつなぐ方法より，処理の柔軟性その他の点ですぐれている。

6　おわりに

Prologにおける推論処理であるリダクションは，ファンクタのない場合に関係演算を用いて記述できることがわかった。ファンクタのある場合や，変数を扱うには，既存の関係データベースでは，これらの項を，文字列の形出格納し，利用する時に，木構造などの扱いやすいデータ構造に展開する必要がある。ファンクタや変数を含んだ場合に効率良い処理を実現するためには，関係データ及び関係演算に拡張が必要である。関係データに対しては，可変長フィールドやタグの導入が必要になり，関係演算に対しては，単一化を含む操作への拡張が必要になる。このように拡張された関係データや，関係演算を支援するためには，〔1〕のようなアーキテクチュアを基本としたアーキテクチュアを考えるのが，妥当である。また，単一化を伴う関係代数演算は，ICOTからも提案されている〔4〕が，その実現に際しては，解決するべき点が多く残されている。

今後の課題として，拡張関係演算の実装，推論処理方式の検討，質問前処理の実現方法，エキスパート・システムなどの応用に関する検討などがあげられる。

参考文献

〔1〕M. Kitsuregawa: "Relational Algebra Machine Based on Hash and Sort: GRACE", 東京大学大学院博士論文　1982年3月

〔2〕H. Yokota et. al: "Deductive Database System based on Unit Resolution", ICOT TR-123

〔3〕鈴木寿和：「関係論理を用いたProlog インタプリタの試作」，東京大学卒業論文　1985年3月

〔4〕横田他，「単一化結合の処理方式」第32回情報処理学会全国大会1M-7　1986年3月