(9)

AI86－24

演繹データベースにおける推論実行方式の評価

吉田　敦・土井晃一
田中英彦
（東大）
嶺野和夫
（富士通）

1986年10月24日

社団法人 電子通信学会

演繹データベースにおける推論実行方式の評価

Evaluation of Inference Execution Method in Deductive Database

吉田 敦† 土井晃一† 嶺野 和夫†† 田中 英彦†

Atsushi Yoshida Kouichi Doi Kazuo Mineno Hidehiko Tanaka

東京大学 工学部† 富士通株式会社††

University of Tokyo Fuculty of Engineering FUJITSU LIMITED

1. はじめに

エキスパート・システム，CAD，自然言語処理等，大量の知識を扱う知識処理においては，問題解決に必要な知識の量が膨大になるため，従来のように必要な知識全てが主記憶上にあると仮定したシステムでは対応できなくなる。従って大量の知識に対して効率良い検索や管理を実現する機能が必要である。これを実現する方法として，独立した推論機構に，関係データベースをつなぐだけの方法と，関係データベースの機能を拡張し，その上で推論機能を実現する方法がある。前者は，論理データベースとして古くから研究が行われてきたが，処理効率の向上には限界があり，これにかわる方法として後者の方法が注目されつつある。後者のように関係データベースの機能を拡張し，ruleとfactが扱えるようにしたものを演繹データベースと呼ぶ。我々は後者の方法を採用し，演繹データベースの上で論理型言語を実行する方法について検討・評価してみた。

2. 融合方式と結合方式

本稿では，独立した推論機構に通信路を介して関係データベースを接続する方式を結合方式と呼ぶ。これに対して，関係データベースの機能を拡張し，その上で推論機能を実現する方式を融合方式と呼ぶことにする。両者の概念図を図1に示す。本章では，結合方式に対して，融合方式の方が優れている理由を簡単に述べる。

2.1 結合方式

独立した推論機構に，独立した関係データベースマシンを，通信チャネルを介して接続することにより，知識ベースマシンを構築しようとする方式である。結合方式では，既存の推論機構，関係データベースにほとんど手を加えることなく利用できる。しかし，次のような欠点

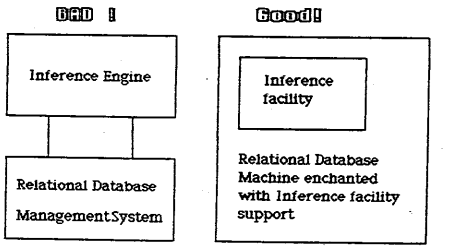

Fig1 2 approaches for constructing Deductive Database System

がある為，実用的なシステムは，実現困難であると言ってよい。

通信オーバヘッドを考慮すると，質問を関係演算の系列に直してから，その系列を関係データベースに送る方式を取ることになる。しかし，このような方式では，factを評価することで得られる変数に対する束縛を用いた定義の絞り込みができないため探索空間の絞り込みができない。さらに実行言語として，pure Prolog を仮定しているため評価可能述語や，副作用を伴う述語が効率良く扱えない。

また，推論機構と関係データベースとの処理速度が異なるため，推論機構側では，動作時間の殆どが，関係データベースの処理待ちになり，推論機構の稼働率は低くなる。

フォン・ノイマンの隘路の解消をすることがデータベースマシンの目的の一つである。同様に知識ベースマシンにおいてもフォン・ノイマンの隘路の解消が解決されるべき問題になる。しかし，結合方式ではこの問題を回避しているにすぎない。従って，結合方式で構築されたマシンは推論機構を関係データベースのフロント・エンド

としてバッチ処理的に使用する場合にしか使いものにならないと言ってよい。

2.2 融合方式

関係データベースの機能を拡張し，その上で推論機能を実現しようとする方式である。融合方式では，結合方式のように推論を実行する環境が二分されてないので，通信オーバヘッドを考慮することなく，処理方法を柔軟に選択することが可能である。即ち，同じ処理を実行する場合，融合方式の方がより最適化された処理が実行可能になる。また評価可能述語や定義更新述語などが入ってもそれに対応できる。さらに，稼働率の極端に低いユニットと，稼働率の極端に高いユニットが存在するような，負荷のアンバランスが生じない。更に質問の前処理や，知識に対するエディタ等のツールを使い，デバッグをしながら応用システムの開発をすることを考えると，融合方式の方が優れたプログラミング環境を与えるはずである。以上のことから，融合方式の方が，結合方式より実用システムが実現しやすいといえる。しかし，次の章で述べるような各技法の開発が必要なため，実際のシステムはまだ存在していない。

3．融合方式の実現

2で述べた融合方式の実現にあたっては，解決するべき問題が多い。この章では，これらの問題について触れてみる。

3.1 関係演算の拡張

関係データベースの上で推論機能を実現する場合，推論操作の実行に用いられるデータを関係で表現することが必要になる。この時，関係上で変数やファンクタの表現が必要になる。従って，関係は，フィールド数を可変にする必要がありうるか，または1フィールドあたりのレコード長が可変になるよう拡張する。また，関係に対する探索も，単に等価性だけではなく，単一化可能性を条件とする探索が必要になる。また，単一化可能な項に対しては，単一化代入子による項の書換え（reduction）が必要になる。これらの単一化・縮退を伴う関係演算を，ＥＸＴＲＡと名付けることにする。ＥＸＴＲＡでは，探索条件が単一化可能性に拡張されるので，選択，結合操作が拡張され，これを単一化選択，単一化結合と呼ぶ。射影については探索条件の拡張は影響しないので，拡張をしない。これと同じ演算がＩＣＯＴからも提案されている〔1〕。ＩＣＯＴの方式は，項に対して全順序を考え，これに基づく整列を実行する。これに対して我々のＥＸＴＲＡは，項に対してhashベクタを計算し，hashベクタでクラスタ分割する方法をとる。sortの手間を考えなければＩＣＯＴの提案した方式より我々の方式の方が速い処理が可能である〔2〕。

3.2 質問の前処理

知識ベースにおける質問処理は，データベースにおける質問処理と対応する。データベースにおいては，与えられた質問に対し最適な処理手順を決定するために質問の前処理が行われる。従って，知識ベースにおいても，与えられたゴールに対しその処理手順を最適化するために前処理実行することを考える必要がある。推論機能として論理型言語の一つであるPrologの実行を考えるので，Prologの質問に対する前処理を考えることにした。前処理の方法としては，ruleやfactの間の呼出し関係，定義の数，単一化が成功する確率等を考慮し，与えられた質問に関連するliteral の処理方法を選択していく方法を取る。ここでは質問の前処理のうち，特定の質問によらない部分を質問を受ける前に実行しておくことで，質問を受けてからの処理時間の短縮を図る。

3.3 質問処理の実行

融合方式の長所は，質問に対する処理方法が通信のオーバヘッド等を考えることなく柔軟に選択できる点である。ここでは，質問に対する処理として，変数に対する束縛情報を生成し，これを用いて候補の絞り込みを実行する単一化・縮退の実行と，質問（ゴール）に対する処理を等価な関係演算の系列に展開したものを予め用意しておき必要に応じて選択条件の決定を行ったその系列を実行する方法の２つを考える。実際の質問に対してはこれらの方法を，前処理で決めた組合せ方で実行する方法をとる。処理の実行方法については，次章で述べる。

3.4 知識の管理

関係データベースにおける，データ依存関係のようなものを知識についても考えることは，知識の獲得（学習）や管理の機能を実現する上で重要なテーマである。知識は，そのクラス毎に別の関係に格納される。この関係の間の相互関係，同一関係内の各知識間の関係を，関係に対するビューや，関数従属性，多値従属性などの，関係データベースにおけるデータの相互関係や正規形の関係の性質あるいはその拡張を用いて管理する。これにより，知識の間の冗長性や矛盾のチェックを行い，知識の追加，学習の機能を実現することが可能になる。これらの諸機能を取り入れた総合的なシステムの構成は図２の様になる。

3.5 並列処理

我々は，知識ベースマシンを実現するアーキテクチャとして，hashとsortに基づく関係代数を高速に実行するデータベースマシン〔3〕を基礎とするアーキテクチャ

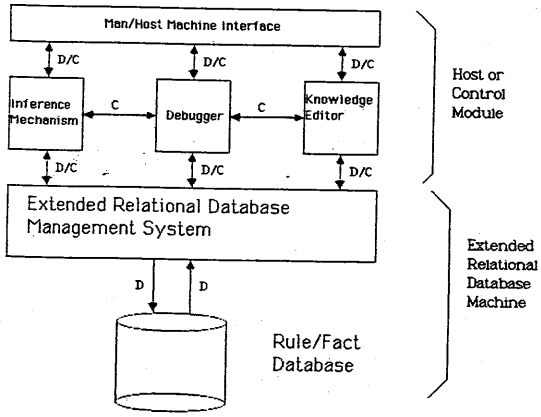

Fig 2 Conceptual View of DDB System for Expert system and so on

を考えている（図3）。このアーキテクチャにおいて，複数台のprocessing module の群を一つのゴールに割り当てるものとし，このような割当てを複数のゴールに対して行う（ＭＩＭＤ処理）なら，ＡＮＤ並列やＯＲ並列が実現可能である。但し，assertやretract のように，定義の集合の変更を伴う述語の実行に対しては，定義の集合に対しlockをかける，または，時刻印を用いた制御などの並行性制御が必要になる。尚，１つのゴールに対する処理を１つのtransaction とみなすと１つのtransaction がさらに子供のtransaction を呼び出す形になるのでこの推論処理はnested transactionになる。

4．質問処理の実行

我々は，演繹データベースにおける推論機能として，

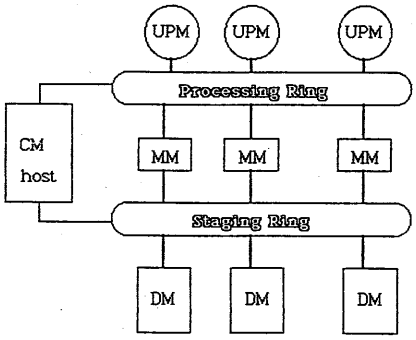

Fig 3 Architecture of Deductive Database

最初に，論理型言語の一つであるＰrolog のreduction を選んだ。その理由として，関係データベースと論理型言語の相性が良い，即ち，関係データベースにおける各概念と，論理型言語における各概念の対応がよいこと，論理型言語における定義の探索は，基本的には，関係データベースでの探索と同じであることが挙げられる。

本章では，Prologにおける推論動作を演繹データベースの上で実現する方法について述べる。

4.1 遅延評価方式

ruleの定義のみを用いて，質問をfactの系列に展開し，そこから関係演算の系列を生成して実行する方法を遅延評価方式と呼ぶ。結合方式による知識ベースでは，通信のオーバヘッドが問題になるので，推論機構と関係データベースの間の通信が少なくなる遅延評価方式または部分評価方式が実装される。ここでは，遅延評価方式の，融合方式での実現方法を述べるが，ここで挙げる方法は，束縛情報を用いた絞り込み方式と比較するため融合方式の上で遅延評価方式を実行する方法を考えたものである。

融合方式では，ゴール用のスタックと，fact系列用のキューを用意し，次のような処理を行う

Ｐhase1 Ｐhase2 を，ゴールスタック，fact系列キューがともに空になるまで繰り返す。

Ｐhase1 ゴールスタックが空でなければ，以下の処理を実行

① factの系列のみになっていたらfact系列のキューに入れる。
② ゴールの中からliteral を一つ選択する。
③ リテラルがfactなら，最右literal にして，ゴールをスタックに返す。
④ リテラルがruleなら，ruleの定義を取り出し，リテラルパターンで単一化選択を実行し，残った候補から，新しいゴールを生成し，ゴールスタックに入れる。

Ｐhase2 fact系列のキューが空でなければ，以下の処理を実行

① リテラルを一つ選択し，そのfactに対する定義を探索する。
② 探索結果から，変数の束縛を示す関係を生成する。
③ 既に，そのfact系列に対する束縛があれば，単一化結合により，consistency check を実行する。
④ リテラルがなくなったら，射影操作により，解集合を生成する。既に，他の解集合があれば，その解集合との和集合を作る。

ここで述べた遅延評価方式は，結合方式における遅延評価方式とは，以下の点で異なる。

結合方式における遅延評価方式は，完全に深さ優先の

探索である。従って，全解探索を実行するためにはback trackingを行う必要がある。ここで述べた方式ではback trackingの必要はないが停止性には注意が必要である。停止性が保証されている問題に対しては，処理にかかる手間は，結合方式における遅延評価方式とほぼ同じである。

4.2 束縛を用いた絞り込み方式

遅延評価方式のようにfactの処理を遅延させないで，即座に実行し，得られた変数の束縛を，束縛情報と呼ばれる関係に格納し，後の定義探索において，この束縛情報による候補の絞り込みを実行する。候補が早期の内に落されるため，全体としての処理量は，遅延評価方式に比べると少なくなる

束縛情報を用いた絞り込み方式では，推論は次のようにして実行される。ここでは，pure Prologの実行方法を示す。

ゴールは，処理待ちのためのゴールスタックに格納され，処理は，このゴールスタックが空になるまで以下の操作を繰返し実行する。

① ゴールスタックからゴールを一つ取り出し，その中のリテラルを一つ選択する。

② リテラルの定義情報を調べ，それに基づいて，次のような処理を実行する。

factのみの場合

(1) factに対する定義を取り出してくる。

(2) 束縛情報を基に，各束縛組をゴールに代入した，ゴールインスタンスの集合を作る。

(3) (1), (2)で得られた関係の間の単一化結合による定義の探索を実行する。

(4) (3)の結果から，そのリテラルに対する束縛情報を生成する。

(5) 束縛情報があれば，共通変数がある時は，単一化結合によるconsistency check を実行，共通変数が無い場合は，束縛情報同士の直積をとる。

ruleのみの場合

(1) ruleに対する定義を取り出す。

(2) 束縛情報を基に，各束縛組をゴールに代入した，ゴールインスタンスの集合を生成する。

(3) (1), (2)の関係の間で単一化結合を取ることにより，定義探索を実行する。

(4) (3)の結果から，新しいゴールを生成する。（ゴールの定義によりあらたに変数に対する束縛が生じる場合には，束縛情報との間で，consistency check を実行する。

rule, factともに定義がある場合

ruleに対する処理を実行したあと，factに対する処理を実行する。

pure-Prolog に範囲を限定すると，処理は上のようになる。

組込み述語が入った場合，定義更新述語が入った場合については，あとで述べる。

4.3 関係演算系列の実行

質問に対する処理が，再帰形式でない等，簡単な形の場合には，その質問（ゴール）に対する処理は，拡張関係演算の系列の形に変換できる。この変換は，質問に対する前処理で，ゴールに対する処理パターンを生成しておき，実際のゴールが来た時に，このパターンに具体的な値を代入して実行する。単一化・縮退を実行する方法に比べると，余計な束縛情報を生成・処理しない分だけ処理量が少なくなる。

Prolog に於ける質問の全てが関係演算の系列に置換え可能なわけではない。再帰形式の質問の場合，収束判定には，関連する定義の数の多項式オーダの時間がかかることが知られている〔4〕。従って，現実的には，関係演算の系列に展開可能な質問のクラスは限られたものとなる。以上のことから，関係演算の系列に展開できる部分は，関係演算の系列に展開したものを，単一化・縮退の中に埋め込む形で実行する。

4.4 組込み述語の実行

エキスパート・システム等の応用を考えると，算術・比較演算や，副作用を伴う，入出力述語等が必要になってくる。これらの述語を組み込んだ場合の解析は，極めて複雑になるため，従来の演繹データベースでは，全く考えられてこなかった。我々の研究では，複雑な解析は考えないことにして，組込み述語を導入した場合，処理の手間を少なくするための，組込み述語実行戦略について検討する方針を取る。

組込み述語としては，四則・比較演算のほか，論理型言語に対する新しいデータ構造として，集合の拡張である関係データを導入し，関係の生成，関係演算，集合演算を取入れ，入出力述語等も取り入れることを検討中である。

4.5 定義更新述語

Prolog におけるassertやretract は，定義の集合自体に対する更新操作である。但し，実際の応用プログラムの多くは，これらを，大域変数の更新操作にしか使用していないのが現状である。しかし，知識の更新を伴うような応用，たとえば，学習機能を持つエキスパート・システムには，本質的に必要なものとなってくる。そこで，我々は，更新系述語の導入に対して，質問処理をデータベースに於けるnested transactionと考え，更新系述語の実行にあたっては，必要な定義集合に対しlockをかける，又は，時刻印による制御を導入するなど，同時実行制御の技術を適用することを検討中である。

5．実行方式の評価

4章で述べた，演繹データベースにおける論理型言語の実行方式について，簡単な評価を行ってみた。ここでは，pure- Prolog の範囲の評価を行うことにする。

5.1 評価の方針

演繹データベースでは関係に対する操作や，関係のタプルと他のデータ型の間の変換が，全体の処理時間に対し支配的な処理となると仮定する。実際のデータベースマシンでは，ディスクへの入出力が支配的な処理であるとされているが，ここでは，入出力の手間については，後の検討課題とした。論理型言語の実行を考えた場合，関係に対する操作としては，ゴールインスタンスの集合と定義の集合の間の単一化結合，consistency check のための，束縛情報間の単一化結合，解集合生成のための射影や合弁（和集合）操作がある。データ変換操作としては，ゴールインスタンスの生成，新しいゴールの生成，局所的な束縛の生成がある。関係操作とデータ変換とでは，関係操作の方が支配的であると仮定する。

評価の方法としては，いくつかの形式的な例題を考え，パラメータを変えて前述の各処理の手間の変化を調べることにした。パラメータとして，各rule，fact間の単一化成功率と，リテラルの評価順序をとった。これにより，前処理による戦略決定に必要なデータも得られる。単一化成功率については，次のような量であると定義した。

あるfactによる束縛情報により，あるruleの候補を探索した場合，その定義全体に対して，探索に成功した定義の数の比率を，そのfactとruleの間の単一化成功率と定義する。fact間，rule間，束縛情報とfactまたはruleの間についても，同様に単一化成功率が定義できる。

比較評価の方針としては，各例題について，遅延評価方式と，束縛による絞り込み方式を比較した。例題によっては，遅延評価方式に対してデータをとらなかったものもあるが，その例題については，リテラルの評価順序が換えられることで，いかに処理の効率化が可能であるかを示すデータであると考える。

評価にあたって，C言語による演繹データベースのシミュレータを作成した。シミュレータは，拡張関係演算を実行するサブルーチン群と，演繹データベース上の，論理型言語実行のためのサブルーチン，メイン・ルーチンからなり，pure- Prolog を実行する部分だけでソースコードにして3000行，オブジェクトのサイズにして256kbyteのものである。シミュレータは，VAX11-730 上のＵＮＩＸ4.2bsdの上で動作する。

5.2 単一化成功率の変化

例題１は，fact f1 から，rule r1,r2に対して絞り込みがある場合で，パラメータとして，f1からr1,r2 に対する単一化成功率を変えて実験を行った。パラメータの変え方として，両方の単一化成功率を同じにしつつ変化させた場合と，両方の単一化成功率を異なる値にして変化させた場合について実験を行った。

各々の結果を，表１，２に示す。

表１の結果より，次のようなことが言える。単一化成功率を変化させると，束縛を用いた絞り込み方式では，単一化成功率にほぼ比例して，ゴールインスタンス生成，束縛情報の生成，新しいゴールの生成，consistency check の手間が増加する。ゴール集合と定義集合の間の単一化結合の手間も，単一化成功率の高い領域では，ほぼ単一化成功率に比例する。これに対し，遅延評価方式では各処理の手間は単一化成功率によらずほぼ一定であった。両方式を比較してみると，ゴールインスタンス生成の手間は，単一化成功率が高くなると，束縛を用いた方法の方が，遅延評価方式より手間が多くなる。新しいゴールの生成については，単一化成功率が100 %になると両方式における手間はほぼ等しくなる。要求・定義間の単一化結合，束縛情報の生成については，束縛を用いた方式では，単一化成功率が100 %でも，遅延評価方式の場合の半分程度の手間になった。consistency check については，束縛を用いた方法では，遅延評価方式の手間より１～２桁小さい値を示した。

このことから，consistency check の手間が支配的な問題では，束縛を用いた方式の方が処理時間が短くなることがわかる。但し，ゴールインスタンスの生成，新しいゴールの生成等の，タプルからゴール，またはその逆のデータ変換の手間が，単一化成功率が高くなると効いてくるので，これらのデータ変換を効率良く実行する方法が必要になってくる。

5.3 評価順序の変更

同じrule定義に対し，そのbody部にあるリテラルの評価順序を変更した場合についても，データを取ってみた。ここでは，fact f1 からrule r1,r2に対して，異なる単一化成功率の場合及び同じ単一化成功率の場合（例題１）と，fact f1,f2からrule r1 に対して異なる単一化成功率の場合（例題２）についてデータを取ってみた。例題１，２いずれに対しても，body部literal 数は３個なので，評価順序は６通り存在する。この結果を，表３～６に示す。これらの結果から，次のようなことが考えられる。

factからruleに対する単一化成功率が低い場合，factを先に実行することによる手間の削減効果が見られる。この効果は，単一化成功率のほかに，そのruleのbody部の処理にかかる手間にも依存する。factからruleに対する単一化成功率が高いと，factを優先実行することによる手間の削減効果はあまり見られない。手間の削減効果の最も大きい処理は，consistency check である。

これらのことから，単一化成功率が低いfactとruleが同じruleのbody部にある場合，factを先に実行するように戦略を決定するのは，手間の削減の効果が大きいので，積極的にこの戦略を用いるのは有効である。全般的に言って，factとruleが同じbody部にある場合，factを先に実行する方が良い。

5.4 再帰処理の例

例題3は，簡単な再帰的定義の入った例題である。これについてデータを取った所，表7のような結果が得られた。この結果を見ると，要求・定義間の単一化結合の手間は，束縛を用いた方がかえって多くなっている。しかし，consistency check については，束縛を用いた方が遅延評価方式に比べると遙かに少なくなっている。その他の処理については，束縛を用いた方が遅延評価方式より少ない。

6．考察

ここでは，処理の手間の数え方として，1つのタプルまたはゴールを処理する手間を1として数えている。また，関係演算にかかる手間に対して，単一化結合は，2つの関係のタプルの数の和に比例し，単一化選択，射影は関係のタプルの数に比例するという仮定をおいた。単一化結合の手間が，通常の結合操作のように2つの関係のタプルの数の積でないのは，単一化結合をhashとsortに基づく高速アルゴリズムで実行〔2〕するという仮定をおいているからである。

要求と定義の間の単一化結合について見ると，1対多で単一化結合を実行している場合が多く見られた。単一化結合の処理の手間は，2つの関係のタプル数の和なので，この場合には，単一化結合ではなく，単一化選択を実行するべきである。また，定義側パターンが一つだけの場合には単一化パターン検索は不要であり，通常のreduction を実行すれば良い。

再帰的処理等の場合，以前実行した単一化パターン検索の結果の再利用が有効である。例えば例題3の場合，前に実行した単一化パターン検索の結果と，その時の要求側パターンを記録しておけば，要求・定義間単一化パターン検索の手間は半分程度まで減らせる。

2つのfactからruleに対する単一化成功率が同じ場合，両方のfactをruleより先に実行するなら，どちらを先に実行しても処理の手間に差は見られない。片方のfactのみをruleより先に実行する場合，両方をruleより先に実行する場合より，consistency check の手間，要求・定義間の単一化結合の手間が増えるが，どちらのfactを先にしても手間は同じである。（後に評価されるfactにより落される処理の分だけ余計に処理が必要になる。）ruleを先に評価すると，絞り込みが全く行われないため，処理にかかる手間は，遅延評価方式より多くなる。

2つのfactからruleに対する単一化成功率が異なる場合，両方のfactをruleより先に実行する時には，どちらのfactが先でも同じ処理の手間になるが，片方のfactを先に実行した場合，単一化成功率の低いfactを先に実行した方が，factによりruleの候補が落される分だけ，処理の手間が少なくなる。単一化成功率の高いfactのみを先に実行した方は，2つのfactをruleより先に実行した場合とほとんど同じだけの手間がかかっている。実際には，単に，一つのfactと，ruleの間の単一化成功率ではなく，ruleを実行する時の束縛情報と，ruleの間の単一化成功率により，処理の手間がほぼ決まると考えた方が良い。この実験結果は，2つのfactの間の単一化成功率の影響をほとんど受けていない。2つのfactの間の単一化成功率の影響は，consistency check にかかる手間にわずかに影響を与えるだけである。また，fact間に共有変数があるかどうかということも，処理にかかる手間にほとんど影響していない。

これらのことから，実行方式に対する戦略として，ruleを実行する際に，そのruleと，その時の束縛情報の間で，単一化成功率が最小になるように実行順序を制御するのが，全体の処理時間の短縮効果が最も大きいといえる。従って，例えば，単一化成功率の低いfactとruleが同じゴールの中にある場合にはfactをruleより先に実行するといった戦略を実行時にとるか，前処理で予め処理順序をそのように決めておけば，質問処理にかかる全体の処理ステップを少なくできる。

7．おわりに

以上のことから，束縛を用いた絞り込みを実行する方法が多くの場合に，遅延評価方式より処理の手間が少なくなることが実証された。このことから，遅延評価だけでなく束縛を用いた絞り込み方式も実行可能な融合方式の方が，事実上遅延評価しか実行できない結合方式より優っているといえる。また，束縛を用いた絞り込み方式で手間の削減効果を上げるには，単一化成功率の低いfactとruleが同じゴールにある時は，factを先に評価するという戦略をとるべきであるといえる。

今後の課題として，現在，組み込み述語の入った場合や，定義更新述語の入った場合の実行方法を検討・評価中である。また，知識の管理，並列処理等についても，検討していき，最終的にはこれらの検討により得られる知見を基にして，知識ベースマシンのアーキテクチャを検討する予定である。

謝辞

この研究を進めるにあたって貴重な意見を提供してくださった東京大学田中英彦研究室ならびに生産技術研究所高木幹雄研究室のＳＩＧＫＢのメンバーの方々に感謝いたします。

参考文献

〔1〕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

〔2〕大森，吉田，田中，「推論機能と関係データベースの融合方式」人工知能と知識処理研究会AI86-21 1986年 7月

〔3〕M.Kitsuregawa ：Relatinal Algebra Machine Based on Hash and Sort：GRACE，東京大学学位論文 1982

〔4〕J.D.Ullman：Implementation of Logical Query Languages for Databases，ＡＣＭ　ＴＯＤＳVol10,no.3　Sep.1985

表１．単一化成功率の変化

```
問題：r(X,Y,Z):-f1(X,Y),r1(X,Z,W),r2(Z,W,Y).
r1(s1,X,Y):-f11(X,Y).        r2(X,Y,t1):-f20(X,Y,Z),r3(Y,Z).
r1(s2,X,Y):-f12(X,Y).        r2(X,Y,t2):-f21(X,Y,Z),r3(Y,Z).
r1(s3,X,Y):-f13(X,Y).        r2(X,Y,t3):-f22(X,Y,Z),r3(Y,Z).
r1(s4,X,Y):-f14(X,Y).        r2(X,Y,t4):-f23(X,Y,Z),r3(Y,Z).
r1(s5,X,Y):-f15(X,Y).        r2(X,Y,t5):-f24(X,Y,Z),r3(Y,Z).
                             r3(X,Y):-f4(X,Z),f5(z,Y).
```

fact f1 とrule r1,r2の間の単一化成功率を同じ値にしつつ変化させた時の結果を下の表に示す。ここでf1-s5はfactであり、その定義は５個ずつとした。各factの間の単一化成功率は０％または１００％である。

表内において／の右側の値が遅延評価方式での値、左側の値が束縛を用いた絞り込みを行なう方式での値である。

単一化成功率	２０％	４０％	６０％	８０％	１００％
推論サイクル数	21/238	28/238	46/238	63/238	81/238
goal instance 生成	29/57	39/57	61/57	83/57	105/57
要求・定義間単一化結合	91/550	95/562	157/574	219/586	281/598
束縛情報の生成	16/250	21/265	32/275	43/280	54/285
新しいゴールの生成	12/56	18/56	29/56	40/56	51/56
consistency check	10/175	24/308	42/382	60/426	78/440

表２：例題１において単一化成功率の値を異なる値にした場合の実験結果

単一化成功率f1-r1|f1-r2	20%|80%	40%|60%	60%|40%	80%|20%
推論サイクル数	21/238	28/238	41/238	27/238
goal instance 生成	28/57	39/57	56/57	46/57
要求・定義間単一化結合	79/550	95/562	157/562	127/550
新しいゴールの生成	12/56	18/56	24/56	15/56
束縛情報の生成	16/260	21/270	32/270	31/260
consistency check	18/294	24/338	42/338	36/294

例題２：body部リテラルの評価順序を変更した場合（２つのfact間の共有変数無し）

```
query(X,Y,Z,W):-f01(X),f02(Y),r01(X,Y,Z,W).
r01(ef51,ed70,X,Y):-f0(X,Z),f1(Z,Y).
r01(ef52,ed71,X,Y):-f2(X,Z),f3(Z,Y).
r01(ef53,ed72,X,Y):-f4(X,Z),f5(Z,Y).
r01(ef54,ed73,X,Y):-f6(X,Z),f7(Z,Y).
r01(ef55,ed74,X,Y):-f8(X,Z),f9(Z,Y).
```

単一化成功率、リテラル評価順序の組み合わせ表（表３）

番号	リテラル評価順序	単一化成功率f01-r01	単一化成功率f02-r01
1	f01 - f02 - r01	４０％	６０％
2	f01 - r01 - f02	４０％	６０％
3	f01 - f02 - r01	２０％	８０％
4	f01 - f02 - r01	２０％	８０％
5	f01 - r01 - f02	２０％	８０％
6	f02 - r01 - f01	２０％	８０％
7	f01 - f02 - r01	６０％	４０％
8	f01 - f02 - r01	８０％	２０％

表4 例題２に対する実験結果

条件	1	2	3	4	5	6	7	8
推論ステップ数	8	9	6	6	6	15	8	6
ゴール数最大値	2	2	1	1	1	4	2	1
goal instance 生成	54	31	43	43	19	55	54	43
要求・定義間単一化結合	78	80	47	47	43	154	78	43
新しいゴールの生成	3	3	2	2	2	5	3	2
束縛情報の生成	70	47	50	50	26	86	70	50
consistency check	92	84	61	61	42	135	92	61
射影	20	20	10	10	10	10	20	10
和集合	20	20	0	0	0	0	0	0

表5 例題３ 実験パラメータ

例題番号	単一化成功率f01-f02	単一化成功率f01-r01	単一化成功率f02-r02
3－1	２０％	４０％	８０％
3－2	２０％	８０％	４０％
3－3	４０％	２０％	８０％
3－4	４０％	８０％	２０％
3－5	８０％	２０％	４０％
3－6	８０％	４０％	２０％

表6 例題３ 実験結果

以下の各実験結果において実行時間は、elasped timeであり、精度は低い。

例題３－１：評価順序

１：f01-f02-r01　２：f01-r01-f02　３：f02-f01-r01　４：f02-r01-f01

５：r01-f01-f02　２：r01-f02-f01

評価順序	1	2	3	4	5	6
推論ステップ数	6	9	6	15	19	21
ゴール最大数	1	2	1	4	5	5
goal instance 生成	19	31	19	55	64	66
要求・定義間単一化結合	47	80	47	154	197	209
新しいゴールの生成	2	3	2	5	6	6
束縛情報生成	26	46	26	86	103	105
consistency check	37	73	37	135	133	155
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	--	--	188	213	197

例題３－２：評価順序

１：f01-f02-r01　２：f01-r01-f02　３：f02-f01-r01

４：f02-r01-f01　５：r01-f01-f02　６：r01-f02-f01

評価順序	1	2	3	4	5	6
推論ステップ数	6	15	6	9	21	19
ゴール最大数	1	4	1	2	5	5
goal instance 生成	19	55	19	61	66	64
要求・定義間単一化結合	47	154	47	80	209	197
新しいゴールの生成	2	5	2	3	6	6
束縛情報生成	26	86	26	46	105	103
consistency check	37	135	37	73	155	133
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	190	--	74	225	210

表6　例題3 実験結果（続き）

例題３－３：評価順序

1：f01-f02-r01　2：f01-r01-f02　3：f02-f01-r01
4：f02-r01-f01　5：r01-f01-f02　6：r01-f02-f01

評価順序	1	2	3	4	5	6
推論ステップ数	6	6	6	15	18	21
ゴール最大数	1	1	1	4	5	5
goal instance 生成	20	19	20	55	63	66
要求・定義間単一化結合	47	43	47	154	191	209
新しいゴールの生成	2	2	2	5	6	6
束縛情報生成	27	26	27	86	102	105
consistency check	38	42	38	135	122	155
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	--	--	186	201	230

例題３－４：評価順序

1：f01-f02-r01　2：f01-r01-f02　3：f02-f01-r01
4：f02-r01-f01　5：r01-f02-f01　6：r01-f01-f02

評価順序	1	2	3	4	5	6
推論ステップ数	6	15	6	6	18	21
ゴール最大数	1	4	1	1	5	5
goal instance 生成	21	55	21	19	63	66
要求・定義間単一化結合	47	164	47	43	191	209
新しいゴールの生成	2	5	2	2	6	6
束縛情報生成	28	86	28	26	102	105
consistency check	39	135	39	42	122	155
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	186	--	--	200	225

例題３－５：評価順序

1：f01-f02-r01　2：f01-r01-f02　3：f02-f01-r01
4：f02-r01-f01　5：r01-f02-f01　6：r01-f01-f02

評価順序	1	2	3	4	5	6
推論ステップ数	6	6	6	9	19	18
ゴール最大数	1	1	1	2	5	5
goal instance 生成	22	19	22	31	64	63
要求・定義間単一化結合	47	43	47	80	197	191
新しいゴールの生成	2	2	2	3	6	6
束縛情報生成	29	26	29	46	103	102
consistency check	40	42	40	73	133	122
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	--	--	--	200	225

例題３－６　評価順序

1：f01-f02-r01　2：f01-r01-f02　3：f02-f01-r01
4：f02-r01-f01　5：r01-f02-f01　6：r01-f01-f02

評価順序	1	2	3	4	5	6
推論ステップ数	6	9	6	6	19	18
ゴール最大数	1	2	1	1	5	5
goal instance 生成	22	31	22	36	64	63
要求・定義間単一化結合	47	80	47	43	197	191
新しいゴールの生成	2	3	2	2	6	6
束縛情報生成	29	46	29	26	103	102
consistency check	40	73	40	42	133	122
射影	10	10	10	10	10	10
和集合	0	0	0	0	0	0
実行時間（秒）	--	--	--	--	211	203

例題３のプログラム

```
query(X,Y,Z,W) :- f01(X,Y),f02(Y,Z),r01(X,Y,Z,W).
```

r01 引数の一部は定数になっている。
f01 定義の数は５つとした。
f02 定義の数は５つとした。

例題４　再帰的定義を含んだもの

```
anc(X,Y) :- par(X,Y).
anc(X,Y) :- anc(X,Z),par(Z,Y).
```

推論方式	束縛を用いた方式	遅延評価方式
推論サイクル数	23	27
goal instance生成	27	14
要求・定義間単一化結合	522	302
束縛情報生成	16	176
新しいゴールの生成	18	8
射影	8	14
和集合	16	24
ゴール数最大値	12	--
consistency check	14	294

表７．再帰的定義のある例題に対する結果