ICOT Technical Report: TR-257

TR-257

# 演繹データベースにおける制約付最小不動点

宮崎収兄(沖電気),
伊藤英則

May, 1987



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

# Restricted Least Fixed Points in Deductive Databases

Nobuyoshi Miyazaki* and Hidenori Itoh**

*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

## Abstract

It is important to reduce the amount of computation by using selection conditions effectively in processing recursive queries. It is known that this requirement can be satisfied in the framework of top-down processing. The performance of deductive database systems could be improved if this requirement is fulfilled by bottom-up methods, because they can be easily combined with methods used in traditional database systems. This paper proposes a method to restrict the range of computation of least fixed points (lfp). This method is an extension of Aho and Ullman's method which moves the selection condition into lfp operation. Instead of moving the selection condition itself, we move a special operator called a restrictor into lfp operation. A way to compute a restrictor is discussed, and it is shown that this restrictor can be used to reduce the amount of computation even for complex queries involving mutual recursion and "not" predicate.

# 演繹データベースにおける制約付最小不動点

宮崎収兄（沖電気工業株式会社）
伊藤英則（ＩＣＯＴ）

## 梗概

演繹データベースにおける再帰問い合せの処理では問い合せ中の選択条件を有効に利用して演算量を減少させることが重要である．既にトップダウン処理に基づいた方式ではこのような方法が知られている．従来のデータベースと親和性の良いボトムアップ処理に基づいた方式でこれを可能にすれば，システムの性能の改善をさらに図ることができる．本稿では最小不動点を制約された空間で計算することにより効率化を図る方法を提案する．この方法は Aho と Ullman の提案した選択条件を最小不動点演算中に移動させる方法の拡張であり，選択条件そのものではなく制約子と呼ぶ演算子を移動させる．制約子の計算方法を議論し，この制約子が相互再帰や否定を含む問い合せにも有効であることを示す．

## 1．まえがき

演繹データベースにおける問い合せ処理の研究は従来のデータベースにはなかった再帰型問い合せの実現方式を中心に行われ，現在までに多くの方法が提案されている[1]．単純な再帰問い合せについては各種の方法が提案されその効率的処理方法がほぼ確立している．これに対し，複雑な問い合せ，特に相互再帰や否定を含む場合の処理の研究も最近盛んになってきた．問い合せ処理の方法は大きく２つに分類される．１つはボトムアップを基礎にした最小不動点(least fixed point: lfp)演算法の拡張またはその改良であり，もう１つはトップダウンを基礎にした方法である．

lfp演算法は問い合せを関係演算に変換し処理する．相互再帰や否定を含む問い合せを拡張lfp演算により処理する方法やその改良が提案されている[2][3][4]．我々は演繹データベースを将来の知識ベースシステムの有力なモデルと考え検討を行っており[5]，lfp演算法を基礎にした実験システムの試作も行った[6][7]．しかしlfp演算の改良により複雑な問い合せについても制約条件の伝播を行い仮想リレーション全体の計算を避ける方法はまだ知られていない．トップダウン方式ではタプル置換法の拡張であるQSQ法が提案されている[8]．この方法では複雑な問い合せでも制約条件の伝播により必要なタプルのみを計算できるという利点がある．しかしデータベース技術の研究で蓄積された各種の高速化手法を活用するにはボトムアップ方式が適しているためボトムアップを基礎として仮想リレーション全体を計算するような不効率さをなくせば最適な方法になることが期待できる．

本稿では制約子と呼ぶルール集合を用いてlfp演算を定義域全体でなく制限された領域で行うことにより，仮想リレーションの大きさを制約し演算の効率化を図る制約lfp演算を提案する．本方式は相互再帰や否定を含む問い合せにも適用できる．空間制約にルール集合を用いる方法としてはマジックセットを用いる方法が提案されているが[9]，本方式のほうが制約効果が大きい．

## 2．制約 l f p 演算の原理

最初に基本演算である拡張 l f p 演算を簡単に紹介する[3)4)]．この方法は naive 評価法とも呼ばれる[1)]．問い合せQがホーン節で表されているとする．Qは1つのゴール述語と0または複数のルールからなると仮定する．外延データベース（ファクト）はリレーションとして関係データベースに格納されているとする．この時，Qは関係演算で表されるリレーションの定義式の集合に変換できる．即ち，ホーン節問い合せが与えられた時，構造体がない等の制限のもとで関係演算を用いてその解を得ることができる．通常の関係代数では再帰表現はないが，再帰表現を含む式を許すこととする．この時，問い合せは一般に次のように表される．

$$\text{answer} = \sigma_F(r_1)$$
$$r_k = f_k(r_1, \ldots, r_n) \quad k=1,2,\ldots,n \quad (1)$$

ここで$r_k$はホーン節問い合せのヘッド部に表われる述語に対応する仮想リレーションである．$r_1$はゴール述語に対応する．また$\sigma_F$はゴール中の定数に対応する論理式Fを条件とする選択演算を表す．$f_k$は$r_k$をヘッドとする各節を関係演算に変換し，それらの和をとった関係代数表現である．

［例1］　先祖についての問い合せ

```
:-ancestor(taro,X).
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-ancestor(X,Z),ancestor(Z,Y).
```

ここでparentリレーションは外延データベース（EDB）にあるとすると，この問い合せは，

$$\text{answer} = \sigma_{1=\text{taro}}(\text{ancestor})$$
$$\text{ancestor} = \text{parent} \cup \pi_{1,4}(\text{ancestor} \times_{2=1} \text{ancestor})$$

に変換される．ここで$\pi$は射影を∪は和，×は結合を表す．

問い合せQが与えられた時，式（1）を満たす$r_k$の組$\langle r_1, \ldots, r_n \rangle$をQの（同時）不動点と呼ぶ．不動点を求めればQの解を求めることができる．不動点が複数存在する時，どの不動点にも含まれる不動点

が存在すればそれを最小不動点と呼ぶ．ここで組の包含関係はその要素の包含関係によって定まる．最小不動点は次のアルゴリズムによって求めることができる．

アルゴリズム１　（拡張ｌｆｐ演算）

let $r_k^0$ be {} for k=1,2,...,n;
j:=0;
repeat;
 $r_k^{j+1}$ := $f_k(r_1^j,...,r_n^j)$ for k=1,...,n;
 j:=j+1;
until $\langle r_1^{j+1},...,r_n^{j+1}\rangle=\langle r_1^j,...,r_n^j\rangle$;
answer:= $\sigma_F(r_1)$;

アルゴリズム１を拡張ｌｆｐ演算と呼ぶ．拡張ｌｆｐ演算では制約条件が最後に使われるため，繰り返し演算の中で大きなリレーションを計算してしまったり冗長な計算が有るなどの問題がある．このためこれらの欠点を改良することが問題となる．本稿では制約条件の用いかたを検討する．

まず$\sigma_F$とｌｆｐ演算が可換な場合は次のように効率化できることが知られている[10]．１つの再帰述語のみを含む問い合せanswer=$\sigma_F(r)$, r=f(r)が与えられた時，ｆをｒを含む部分とそれ以外に分離しf(r)=g∪h(r)と書くことができる．この時，$\sigma_F(h(r))$=h($\sigma_F(r)$)が成りたてば，

$$\sigma_F(r)=\sigma_F(g)\cup\sigma_F(h(r))$$
$$=\sigma_F(g)\cup h(\sigma_F(r))$$

である．従って，r'=$\sigma_F(r)$とおけば，

answer=r'
r'=$\sigma_F(g)\cup h(r')$

なる式を得る．この式にアルゴリズム１を適用すると，ｒ全体を計算することなく$\sigma_F(r)$を計算することができ，計算の効率化ができる．これは関係データベースで”選択優先”の処理により効率化ができるのと同様である．

この性質をを利用した効率化は一般の場合には適用できないが，この考えを拡張することを検討しよう．

可換な場合には上式と同時に次の式も成り立つことが容易に確かめられる．

$r' = \sigma_F(f(r'))$

$answer = \sigma_F(r')$

一般にはこのような関係は成りたたないが一方の$\sigma_F$の代りに選択演算$\sigma^*$を考えて，もし

$r' = \sigma^*(f(r'))$　かつ

$\sigma_F(r') = \sigma_F(r)$

を満たす$\sigma^*$が存在すればrの代りにr'を計算すれば問い合せの解を得ることができる．Ωをrの定義域とすれば$\sigma^*(r) = \sigma^*(\Omega) \cap r$の関係があるので，$\sigma^*(\Omega)$をr*と書くことにすれば，仮想リレーションに対応する述語が複数ある問い合せでは，

$$r_k' = r_k^* \cap f_k(r_1', \ldots, r_n') \quad k = 1, \ldots, n$$

$$answer = \sigma_F(r_1') = \sigma_F(r_1) \qquad (2)$$

なる式を考えることができる．式（２）を満たす$r_k^*$の組をQの制約子，各$r_k^*$を$r_k$の制約子と呼ぶ．また$r_k'$の組$\langle r_1', \ldots, r_n' \rangle$をQの制約付最小不動点（rlfp）と呼ぶ（以下この組を$\langle r_k' \rangle$と略記する）．これは$r_k^* \cap r_k' = r_k^* \cap f(r_k')$であるので$\langle r_k' \rangle$が$\langle r_k^* \rangle$上の最小不動点に対応しているからである．式（２）が与えられた時アルゴリズム１によってQの解を求める演算を制約lfp演算と呼ぶ．問い合せの解に貢献する各$r_k$の部分集合を$r_k^+$とすると制約子をうまく設定して$r_k^+ \subset r_k' \subset r_k^* \subset r_k \subset \Omega_k$とし，$r_k^*$を小さくすることができれば計算の効率化が図れる．

自明の制約子として次の２つがある．まず$\langle \Omega_1, \ldots, \Omega_n \rangle$はQの制約子である．また１つの再帰述語のみを含む可換な問い合せではσF(Ω)はQの制約子である．この２つの例からわかるように与えられた問い合せに対して一般に複数の制約子が存在する．式（２）より次の性質が成り立つ．

［性質１］$p_k$が$r_k$の制約子なら$s_k \supset p_k$となる$s_k$は$r_k$の制約子である．

また制約子が与えられた時，可換の場合と同様に全体を$\sigma^*$で制約するのでなく仮想リレーションを含ま

ない部分のみを$\sigma^*$で制約して r' を求めても $\sigma_F(r)=\sigma_F(r')$ が成り立つ．

## 3．制約子の決定

問い合せQの形からできるだけ小さな制約子を決定する方法を検討しよう．一般に制約子を前以って与えられたリレーションまたは選択演算子として決定することは困難なので，制約子を関係代数やルール集合によって表現することを考える．式（２）のように関係代数で表現すると演算順序などの問題があり議論が複雑になるので元のホーン節で考える．また式（２）が成り立つ限り $r_k$ と $r_k'$ の区別は不要なので，以下では必要ない限り" ' "は省略する．

Qの各節は（引数を省略し）ボデイを $h_k$ で表すと，

$$r_k :- h_k(r_1, \ldots, r_n) \quad k=1, \ldots, n$$

の形をしている．式（２）を満たすようにこれを変形すると，

$$r_k :- r_k^* \wedge h_k(r_1, \ldots, r_n) \quad (k=1, \ldots, n) \quad (3)$$

となる．この式を関係代数に変換し同一の $r_k$ に対する定義節の和をとると式（２）になる．$r_k^*$ の引数はヘッド部の $r_k$ の引数と（変数名も含めて）同じとする．次に $r_k^*$ を定義する節を検討する．式（３）のボデイ部に $r_k$ が現われる節に着目し $h_i$ は述語の and の形をしているので $r_k$ を一番右に移すと，

$$r_i :- r_i^* \wedge g_i(r_1, \ldots, r_n) \wedge r_k.$$

の形をしている．問い合せの処理をトップダウンで行う場合，$r_i^* \wedge g_i(r_1, \ldots, r_n)$ の処理を行ってから $r_k$ の処理を行うと考えれば $r_i^* \wedge g_i(r_1, \ldots, r_n)$ の処理によって決定された値により $r_k$ に対する部分問い合せを発行することになる[8]．これをボトムアップで解釈すると，$r_k$ の解がどのようなものであっても，その解のうち $r_i^* \wedge g_i(r_1, \ldots, r_n)$ と両立するもののみが全体の解に貢献することになる．即ち，$r_i^* \wedge g_i(r_1, \ldots, r_n)$ が $r_k$ を限定すると考えてもよい．従って $r_k^*$ の定義節としては以下のものが考えられる．

$$r_k^* :- r_i^* \wedge g_i(r_1, \ldots, r_n)$$

このようにして以下のアルゴリズムが導かれる．

アルゴリズム２

（１）　初期条件

$r_1^*$:-$r_1^*$_init.

$r_1^*$_initは$\sigma_F(\Omega_1)$に対応するユニット節であり，その引数はゴール節と同じ内容である．

（２）　$r_k$の定義節：各$r_k$について以下の変形を行う．

Ｑの各ホーン節 $r_k$:-$h_k(r_1,\ldots,r_n)$　を変形し，

$r_k$:-$r_k^* \wedge h_k(r_1,\ldots,r_n)$　　．．．（３）

を得る．$r_k^*$の引数は変数名も含めヘッド部の $r_k$ と同じ．

（３）　$r_k^*$の定義節：各$r_k$について以下の処理を行う．

各$r_j$の定義節中，ボデイ部に $r_k$ が現われるものに着目し以下のような $r_k^*$ を構成する．ボデイ部に $r_k$ がm個あれば１つの節からm個の $r_k^*$ 定義節が生成される．

ボデイ部より $r_k$ を１つ選ぶ．

$r_j$:-$r_j^* \wedge g_j(r_1,\ldots,r_n) \wedge r_k$．　の形をしている．$r_k^*$をヘッドとし $r_j^* \wedge g_j(r_1,\ldots,r_n)$をボデイとする節を作る．この時$r_k^*$の引数は変数名も含め元の節の $r_k$ と同じとする．

アルゴリズム２で求められる $r_k^*$　の定義節の集合が制約子を与える条件を検討しよう．$r_k^*$ 定義節のコンシステンシイを以下のように定義する．

まず$r_k^*$：-$r_j^* \wedge g_j$（$r_1$，…，$r_n$）の依存関係を以下のように定義する．この節を生成する元の節

$r_j$：-$r_j^* \wedge h_j$　（$r_1$，…，$r_n$）

についてｈ$_j$は述語の結合の形をしている．ボデイの述語をすべて一意に識別できるように同一名の述語には仮想的に番号をふるとする．この時これから生成される節ではヘッドの述語$r_k^*$がボデイにあらわれる述語に依存している．$r_k$は$r_k^*$に依存するので$r_k$がボデイの述語に依存すると考える．ヘッドに対応する$r_k$は仮想的な番号を継承する．このような依存グラフを考える．

同一の節から生成される$r_k^*$の定義節の集合をＳと

する．Ｓがコンシステントまたは順序づけ可能であるとは，Ｓの節の依存グラフを重ね合せた時，そのグラフにサイクルがないことを言う．Ｓがコンシステントであることは対応する元の節の評価がある順序に従って行われ実行可能であることに対応する．$r_k^*$（k=1,...,n）の定義節の集合がコンシステントであるとはこのようなＳのすべてがコンシステントであることをいう．この時，次の定理が成り立つ．

［定理］　アルゴリズム１で解の求められる問い合せＱに対しアルゴリズム２で与えられる節の集合の（拡張）ｌｆｐをアルゴリズム１または他の方法で求めその解を$\langle r_k, r_k^* \rangle$とする．$r_k^*$の定義節の集合がコンシステントなら$\sigma_F(r_1)$はＱの解となる．即ち$\langle r_k^* \rangle$はＱの制約子である．

証明は付録１を参照されたい．

［系］ $r_k^*$の集合がコンシステントでない時，節のボデイからいくつかの述語を削除しコンシステントにすれば$\langle r_k^* \rangle$はＱの制約子になる．

変換後の問い合せを評価する時，初期条件中の変数はその変数の取りうるすべての定数の集合を表現すると考える．即ちその定数の数だけのタプルがあると考える．実際のインプリメントはこのままでは困難なので，関係代数を拡張して変数を含むリレーションを扱ったり，演算の結果に影響を与えない引数を削除する等の工夫が必要である．

なお問い合せ中にPrologと同様に否定（ｎｏｔ）述語を許し，ｎｏｔの引数が仮想リレーションになっている場合は上記のアルゴリズムでその述語についてはｎｏｔをはずしてヘッドにすることにより制約子を構成するように修正すれば良い．

## ４．制約子の性質

前節で導入した制約子の性質を検討するためまず制約子の例を示す．アルゴリズム２を例１の問い合せに

適用すると以下を得る．

```
ancestor(X,Y):-ancestor*(X,Y),parent(X,Y).
ancestor(X,Y):-ancestor*(X,Y),ancestor(X,Z),
                    ancestor(Z,Y).
ancestor*_init(taro,X).
ancestor*(X,Y):-ancestor*_init(X,Y).
ancestor*(X,Z):-ancestor*(X,Y),
                    ancestor(Z,Y).
ancestor*(Z,Y):-ancestor*(X,Y),
                    ancestor(X,Z).
```

このルール集合をアルゴリズム１により評価する時，ancestor*は徐々に大きくなりancestorはつねにこれにより制約されているので無駄な計算を避けることができる．

ここでancestor*の定義節はコンシステントでない．即ち２番目の節は$ancestor_{right} \rightarrow ancestor_{left}$を，３番目の節は$ancestor_{left} \rightarrow ancestor_{right}$を表している．従ってどちらかのボデイのancestorを削除する必要がある．どちらを削除すべきかは２つの場合の結果を比較すれば容易に判定できる．即ち３番目の節のancestor(X,Z)を削除するとこの節は
ancestor*(Z,Y):-ancestor*(X,Y)となり初期条件の定数は第１引数に入っているのでancestor*が定義域全体になってしまい制約力が無くなる．従って第２の節を変形すべきであるが，この節は変形するとancestor*(X,Z):-ancestor*(X,Y)となり計算に影響しなくなり節全体を削除できる．コンシステント化はancestorが大きなリレーションであると仮定しルール／ゴールグラフを用いる方法[11]と同様に変数のバインド関係をとらえることによっても行うことができる．これらの考察からこの問い合せは下記と同等である．

```
ancestor(X,Y):-ancestor*(X,Y),parent(X,Y).
ancestor(X,Y):-ancestor*(X,Y),ancestor(X,Z),
                    ancestor(Z,Y).
ancestor*_init(taro,X).
ancestor*(X,Y):-ancestor*_init(X,Y).
ancestor*(Z,Y):-ancestor*(X,Y),ancestor(X,Z).
```

なお，この問い合せの実行時 ancestor* の第２引数はつねに定義域全体を表現しているので，計算に関与しない．従って，ancestor* の第２引数を削除し１引数の述語として扱うことも可能である．問い合せのゴールがancestor(taro,X)でなく　ancestor(X,jiro)の場合はアルゴリズム２によって例１とはancestor*_initのみが異なった節集合を得るので制約子は上記とは異なる．

この例のように制約子の計算中に一部の引数がバインドされないままになったり，ボデイ部の述語が計算に寄与しない場合，制約子から一部の述語や節を削除できる．このような修正を加えた制約子の定義節は簡単な問い合せに対しては多くの場合マジックセット[9]と同等になる．しかし，例１のようにボデイに同じ再帰述語が２度現われたり，次の例のように相互再帰型の場合はマジックセットには制約効果がないので制約子の方がより強力である．ここで述べたコンシステント化や簡略化は多くの場合容易であるが厳密な定式化は今後の課題である．

［例２］相互再帰型問い合せ

```
p(X,Y):-a(X,Y).
p(X,Y):-p(X,Z1),a(Z1,Z2),q(Z2,Y).
q(X,Y):-b(X,Y).
q(X,Y):-p(X,Z1),c(Z1,Z2),q(Z2,Y).
```

で a,b,c がＥＤＢにあるとする．これを変換すると，p と q の定義は明らかなので省略し，初期条件も省略すると，

```
p*(X,Z1):-p*(X,Y),a(Z1,Z2),q(Z2,Y).
p*(X,Z1):-q*(X,Y),c(Z1,Z2),q(Z2,Y).
q*(Z2,Y):-p*(X,Y),p(X,Z1),a(Z1,Z2).
q*(Z2,Y):-q*(X,Y),p(X,Z1),c(Z1,Z2).
```

を得る．この問い合せの制約子と実行例を付録２に示す．

仮想リレーションに対応する述語が再帰型でない場合にその制約子を構成すると他の述語の影響で再帰型になることがある．一般に非再帰な述語はその都度計算したり，あらかじめ問い合せ中から消去するなどの

扱いができるので，非再帰な述語の制約子は導入しない方が全体としての効率を上げることができる．

制約子ともとの述語とが相互再帰でない時は制約子を計算してから元の述語をを計算できる．このような時，制約子は分離可能であると呼ぶ．制約子が分離可能ならまず制約子を計算してからもとの述語に対応する部分を計算できるので効率化が図れる．問い合せ中の各ルールのボデイ部に再帰述語が高々１つしかない場合はこの問い合せは（強い意味で）線形であると呼ばれる．非再帰述語の制約子を用いない場合，線形問い合せの制約子を構成すると，そのボデイには元の再帰述語は現われなくなる．従って，線形問い合せの制約子は分離可能である．また線形問い合せではアルゴリズム２の結果はコンシステントなのでそのまま制約子の定義になる．

## ５．処理性能

制約ｌｆｐ演算はアルゴリズム１をそのまま適用するのではなく，差分計算[3)12)]を行うなどにより効率化が可能であるがここではその詳細は議論せず，いくつかの例について他の方式との比較を行う．従来の方式の代表的なものについては既に比較されている[1)]ので，ここではそれを基礎として制約ｌｆｐの性能を検討する．性能の尺度としては中間結果のサイズを用いる．

代表的な問い合せ処理方法としてはＨＮ[13)]，（拡張）ｌｆｐ(naive評価とも呼ばれる[1)])，semi-naive評価[12)](naive評価を差分計算を行うように改良したもの)，ＱＳＱ[8)]（これには再帰型のＱＳＱＲと繰り返し型のＱＳＱｌがある）などがあげられている．ＨＮとＱＳＱはトップダウン型であるが，他はボトムアップ型のアルゴリズムである．なおマジックセット[9)]（ＭＳと略す）は定数伝播が可能な多くの場合制約ｌｆｐと同様に動作する．この他に単純な問い合せで差分計算と定数伝播を行い，ＨＮとほぼ同等な性能になるようｌｆｐを改良したアルゴリズムも知られており[3)4)]，これを改良ｌｆｐと呼ぼう．

先祖に対する問い合せで例１とは異なって ancesto

r(X,Y):-parent(X,Z),ancestor(Z,Y) のように線形の場合は，ＥＤＢの内容によって性能の関係が変化するが大略以下のような関係になる[1]．

｛ＨＮ，改良ｌｆｐ｝≪｛ＱＳＱＲ，ＭＳ｝
≪ＱＳＱＩ≪semi-naive≪ｌｆｐ/naive

ここで｛Ａ，Ｂ｝はＡとＢがほぼ同等なことを示す．また≪は１桁またはそれ以上の差を表す．制約ｌｆｐでは生成されるルールがＭＳと同等である．従ってアルゴリズム１をそのまま適用した時はＱＳＱＩと同等になり，差分計算を行えばＭＳ，ＱＳＱＲと同等になる．なおPrologを演繹データベース処理系とみなした場合，問い合せ中の定数の位置やデータの内容により性能が変化するが，最良でＱＳＱＲと同等で最悪ではｌｆｐよりも効率が悪い．

例１や例２のように非線形や相互再帰型の問い合せではＨＮ，改良ｌｆｐ等の効率の良い方式は適用できない．またPrologも適用できない．マジックセットでは定数伝播ができなくなりsemi-naiveと同等になってしまう．適用可能な方法の比較は大略以下のようになる．

ＱＳＱＲ≪ＱＳＱＩ≪｛semi-naive，ＭＳ｝
≪ｌｆｐ/naive

制約ｌｆｐでは例１の計算量は線形の場合と同じくアルゴリズム１をそのまま適用するとＱＳＱＩと同等で，差分計算を行うとＱＳＱＲと同等になると考えられる．

一般に制約ｌｆｐを用いると直接ｌｆｐを求めた場合で中間結果のタプル数がＱＳＱＩと同じ，差分計算をうまく行った場合でＱＳＱＲとほぼ同じになる．制約ｌｆｐではボトムアップを基本にしているので関係演算はＱＳＱのタプル置換より高速化できる場合があり，制約ｌｆｐの方がＱＳＱより効率を良くできる可能性が大きいと考えられる．

## ６．むすび

ｌｆｐ演算を制約された空間で行う制約ｌｆｐ演算の基本原理と制約子の構成方法を提案し制約子の例を示した．また制約ｌｆｐの性能について考察を行った．

制約ｌｆｐにより相互再帰や否定(not)を含むような複雑な問い合せを効率良く処理できる．今後は本方式の実現方法の検討を行うとともに他の方式との組み合わせによる問い合せ処理の最適化の検討を行う予定である．

謝辞

本稿の作成にあたり御意見御助言をいただいたＩＣＯＴおよび沖電気工業株式会社のＫＢＭＳ　ＰＨＩ研究の関係者に感謝します．

## 参考文献


1) Bancilhon, F. and Ramakrishnan, R.: An Amateur's Introduction to Recursive Query Processing Strategies, Proc. of ACM SIGMOD, pp16-52 (1986).

2) Yokota, H., Sakai, K. and Itoh, H.: Deductive Database System Based on Unit Resolutions, Proc. of Intl. Conf. on Data Engineering, pp228-235 (1986).

3) Ceri, S. Gottlob, G. and Lavazza, L.: Translation and Optimization of Logic Queries : The Algebraic Approach, Proc. of 12th VLDB, pp395-402 (1986).

4) Miyazaki, N., Yokota, H. and Itoh, H.: Compiling Horn Clause Queries in Deductive Databases: A Horn Clause Transformation Approach, ICOT Technical Report TR-183, ICOT, (1986).

5) Itoh, H.: Research and Development on Knowledge Base Systems at ICOT, Proc. 12th VLDB, pp437-445 (1986).

6) 阿比留幸展　羽生田博美　宮崎収兄　森田幸伯：KBMS PHI：演繹データベース実験システム PHI / K², 第34回 情報処理学会全国大会予稿集, pp1491-1492 (1987).

7) Itoh, H., Takewaki, T., Miyazaki, N. and Mitomo, Y.: Deductive Database System Written in Guarded Horn Clauses, ICOT Technical Report TR-214, ICOT, (1986).

8) Vieille, L.: Recursive Axioms in Deductive Databases: The Query/Subquery Approach, Proc. 1st EDS, pp179-193 (1986).

9) Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J.: Magic Sets and Other Strange Ways to Implement Logic Programs, Proc. of 5th ACM PODS, pp1 15 (1986).

10) Aho, A. and Ullman, J.: Universarity of Data Retrieval Languages, Proc. of ACM POPL,

pp110-120 (1979).
11) Ullman, J.D.: Implementation of Logical Query Languages for Databases, ACM TODS, Vol. 10, No.3, pp289-321 (1985).
12) Bancilhon, F.: Naive Evaluation of Recursively Defined Relations, in (eds.) Brodie, M.L. and Mylopoulos, J., On Knowledge Base Management Systems, Springer-Verlag, pp165-178 (1986).
13) Henschen, L. and Naqvi, S.: On Compiling Queries in Recursive First-Order Databases, JACM, Vol.31, No.1, pp47-85 (1984).

## 付録１　定理の証明

Ｑに直接アルゴリズム１を適用し得られる解と〈$r_k'$〉から得られる解が一致することを示す．

直接ｌｆｐで与えられる解を〈$r_k$〉とする．また各$r_k$のうち論理式$\sigma_F \wedge f_i(r1,\ldots,r_k)$を真とする値の集合を$r_k^*$とする（但し〈$r_k^*$〉⊂〈$r_k$〉）．〈$r_k^*$〉⊂〈$r_k'$〉⊂〈$r_k$〉であれば，〈$r_k'$〉から求める$\sigma_F(r_i')$は$\sigma_F(r_i)$と一致する．ここで〈$s_k$〉⊂〈$t_k$〉は各kについて$s_k \subset t_k$であることを意味する．

（１）〈$r_k'$〉⊂〈$r_k$〉であること：

$r_k'$は式（２）の関係代数表現から得られる．式（２）のボデイ部を論理式として見た時，これが真であれば式（１）のボデイ部も真である．従って$r_k' \subset r_k$．

（２）〈$r_k^*$〉⊂〈$r_k'$〉であること：

（ａ）　〈$r_k^*$〉⊂〈$r_k^*$〉である．

$r_k^*$の構成アルゴリズムの前半部から$\Omega_1$を$r_1$の定義域とし$r_1^* \supset \sigma_F(\Omega_1)$．

さらに$r_1^* \supset \sigma_F(\Omega_1)$であれば$r_k^*$の構成アルゴリズムの後半部は$\sigma_F(f_1)$を真とする可能性のあるすべての$r_k$の値についてその値を$r_k^*$の元とするよう構成されている．これらの元が実際に$r_k$の元としてｌｆｐ演算で生成されるには$r_k :- f_k$の少なくとも１つの節のボデイが真となり，しかもボデイに現われる各$r_j$の値が$r_j^*$の元になっていることが条件である．$r_k^*$の定義節の集合がコンシステントであれば$r_k^*$の構成アルゴリズムはこれらの$r_j$の値が$r_j^*$の元となるよう保証している．従って〈$r_k^*$〉⊂〈$r_k^*$〉．

（ｂ）　〈$r_k^*$〉⊂〈$r_k'$〉である．

$r_k^*$の元は$r_k :- f_k$の形の節のどれかから生成される．$r_k'$を求める定義では$r_k :- r_k^* \wedge f_k$がこれに対応しており，（ａ）から$r_k^* \subset r_k^*$だから$f_k$

が真ならば $r_k^* \wedge f_k$ も真である．従って $r_k^*$ の各元は $r_k'$ の元であり，$\langle r_k^* \rangle \subset \langle r_k' \rangle$ である．

Q．E．D．

## 付録２　相互再帰問い合せの実行例

本文の例２の問い合せの実行例を示す．なお，制約Ｉｆｐはまだインプリメントしてないため演繹データベース実験システムＰＨＩ／ｋ[2] [6]に制約子のルールを与えて結果の確認を行った．

ＥＤＢには以下のデータがあるものとする．

```
a(i,j). a(j,h). a(h,k). a(t,s). a(k,m).
b(h,i). b(k,t). b(j,h). b(s,o).
c(j,k). c(t,h). c(o,j).
```

:-p(j,X). をゴールとして直接Ｉｆｐ（アルゴリズム１）を実行すると５回目の繰り返しで収束し，下記の結果を得る．

```
p(i,j). p(j,h). p(h,k). p(t,s). p(k,m).
p(i,i). p(j,t). p(i,h). p(j,o). p(i,t).
p(i,o).
q(h,i). q(k,t). q(j,h). q(s,o). q(i,t).
q(j,i). q(i,i). q(i,h).
```

この結果，この問い合せの解は X={h,t,o}である．アルゴリズム２の結果をコンシステントになるように変形し，さらに簡略化を行うと以下のようになる．なおここで p* と q* の第２引数は計算に影響しないので削除し両者を１引数の述語とした．

```
p(X,Y):-p*(X),a(X,Y).
p(X,Y):-p*(X),p(X,Z1),a(Z1,Z2),q(Z2,Y).
q(X,Y):-q*(X),b(X,Y).
q(X,Y):-q*(X),p(X,Z1),c(Z1,Z2),q(Z2,Y).

p*_init(j).
p*(X):-p*_init(X).
p*(X):-q*(X).
q*(Z2):-p*(X),p(X,Z1),a(Z1,Z2).
q*(Z2):-q*(X),p(X,Z1),c(Z1,Z2).
```

このルールをアルゴリズム１で計算すると以下の結果を得る．なお各ループで新しく生成される結果のみ示し，"－"は変化のないことを意味する．

| | p* | p | q* | q |
|---|---|---|---|---|
| 第１ループ | p*(j) | { } | { } | { } |
| 第２ループ | － | p(j,h) | { } | { } |
| 第３ループ | － | － | q*(k) | { } |
| 第４ループ | p*(k) | － | － | q(k,t) |
| 第５ループ | － | p(j,t)<br>p(k,m) | － | － |
| 第６ループ | － | － | q*(s) | － |
| 第７ループ | p*(s) | － | － | q(s,o) |
| 第８ループ | － | p(j,o) | － | － |
| 第９ループ | － | － | － | － |

この結果をまとめると，

p*(j). p*(k). p*(s).
p(j,h). p(k,m). p(j,t). p(j,o).
q*(k). q*(s).
q(k,t). q(s,o).

となり :-p(j,X) の解は X={h,t,o} である．