

演繹 DB に対する質問処理を効率化するプログラム変換手法

堤 富士雄

(財) 電力中央研究所

演繹データベースに対するトップダウン処理とボトムアップ処理を融合する効率的なプログラム変換手法としてマジックセット法が提案されている。我々はマジックセット法では効率改善が望めないタイプのプログラムに対して、EDB 述語の束縛を有効に利用することによって効率改善を果たす変換手法を提案した。本論文ではその手法を否定と関数を含まないホーン節プログラム一般に対し適用できるように拡張する。我々の手法は従来の bf, bcf 修飾とは異なる新たな修飾を使って効率的な変換を可能にしている。本論文ではまた我々の手法による変換後のプログラムを、効率的に評価する新たな評価法について述べる。

A Program Transformation Method to Optimize Computation of Recursive Queries on Deductive Databases

Fujio Tsutsumi

Central Reserch Institute of Electric Power Industry

Otemachi Bldg., 1-6-1 Otemachi, Chiyoda-ku, Tokyo 100, Japan

We propose a program transformation method for a class of function free, negation free and range restricted Horn clause program. The class contains programs that can not be handled efficiently by Magic-Sets which is a famous transformation method combining top-down and bottom-up evaluation. Our method is a extended version of a method that we have proposed before this paper. Our method uses a new adornments those are different from conventional bf or bcf adornments. This paper also presents an effective evaluation method for the transformed programs by our method.

1 はじめに

演繹データベースに対する質問処理を効率化する様々な手法が提案されている [Ullman 89]. なかでもマジックセット法 [Beerli 87] はトップダウン処理の効果をボトムアップ処理に取り入れた手法として有名である. しかし, マジックセット法によっては効率改善が望めないプログラムが存在する. 例を示す.

例 1

```
r1.1 query(X,Y) :- test(X,Y),anc(X,Y).
r1.2 anc(X,Y) :- par(X,W),anc(W,Y).
r1.3 anc(X,Y) :- par(X,Y).
```

ここで述語 *test* および *par* はデータベースに陽に定義されている関係を表すとする. このような述語を EDB (extensional database) 述語と呼ぶ.

また, 質問には束縛が与えられていないとする. その場合, 我々はマジックセット法の効果を得るために, 次の SIPS (side information passing strategies) [Beerli 87] を使う. SIPS とはルール中のサブゴール間で, どのようにデータをやりとりするかというルールの評価戦略を表すものである. SIPS の表記は [Beerli 87] に従う.

SIPS 1.1

```
sips1.1(for r1.1) {test} →→XY anc
sips1.2(for r1.2) {anch} →→X par,
                  {par} →→WY anc
sips1.3(for r1.3) {anch} →→XY par
```

ルール r1 に対する $\{test\} \rightarrow\rightarrow_{XY} anc$ は, EDB 述語 *test* を最初に評価し, その束縛値を述語 *anc* の計算過程に利用するという評価戦略を示している. この SIPS に従ってマジックセット法の変換を行なった後のプログラムは次のようになる.

```
queryff(X,Y) :- test(X,Y), m_abb(X,Y),
                  ancbb(X,Y).
ancbb(X,Y) :- m_abb(X,Y), par(X,W),
              m_abb(W,Y), ancbb(W,Y).
ancbb(X,Y) :- m_abb(X,Y), par(X,Y).
m_abb(X,Y) :- test(X,Y).
m_abb(W,Y) :- m_abb(X,Y), par(X,W).
```

変換後のプログラムは, 関係 *test* および関係 *par* から作られる 2 引数関係 m_a^{bb} によって, 関係 *anc* の計算量を減らすプログラムとなっている. この変換は, 関係 m_a^{bb} をつくり出す計算量が大きくなければ効率的な変換といえる.

しかし関係 m_a^{bb} の大きさは関係 *test* および関係 *par* に依存しており, 特に関係 *test* が大きい場合には関係 m_a^{bb} が非常に大きくなる可能性があり, このよ

うな変換は効果的ではない. □

マジックセット法では, このように大きな EDB 関係を束縛が渡されない場合に効率的に扱うことができない. 例 1 の場合は, ルール r1 に対する $sip1.1 \{test\} \rightarrow\rightarrow_{XY} anc$ を使用せずに, *anc* の計算を先に行ない, そのあとで関係 *test* とジョインを行なうという次の SIPS 1.2 をとる方が効率的である. しかし, この方法では関係 *test* が関係 *anc* の計算にまったく生かされていない.

SIPS 1.2

```
sips1.2 {anch} →→X par,
        {par} →→W anc
sips1.3 {anch} →→X par
```

我々が [Tsutsumi 90] において提案した変換手法は, 例 1 における述語 *test* のような EDB 述語を関係 *anc* の計算において有効に使うことで効率改善を果たす手法である. 本論文で提案する変換手法 (Propagating Test predicates, 以下 PT と呼ぶ) は, [Tsutsumi 90] において提案した手法を否定と関数を含まない領域制限された (range restricted) ホーン節プログラム一般に対して適用できるように拡張したものである.

例 1 (続き)

例 1 のオリジナルのプログラムに SIPS 1.2 が与えられたとする. その場合, PT は次のように変換する.

```
query(X,Y) :- par(X,W), anc'(W,Y),
              test(X,Y).
query(X,Y) :- par(X,Y), test(X,Y).
anc'(X,Y) :- par(X,W), anc'(W,Y).
anc'(X,Y) :- par(X,Y), test(W,Y).
```

変換後のプログラムでは関係 *test* とのジョインが関係 *anc'* の計算量を減らすことに役立っている. □

PT の基本的なアイデアは, 再帰定義された述語の計算過程においてインバリアントになっている引数を見つけ出し, その引数に対する EDB 述語の束縛を計算過程のなるべく早い段階で利用する, というものである.

マジックコンディション法 [Mumick 90] は, プログラム中の算術比較述語による制約を有効に利用する変換手法である. この手法では修飾として従来の bf に加えて新たに c を導入し, 算術比較の制約をマジックファクトの生成ルールに伝播することで, 効率改善を果たしている. この手法の特徴は変換において領域制限性 (range restricted property) を保存している点である. 例 1 も述語 *test* をあたかも算術比較述語であるかのように使用することで, マジックコンディ

ション法による変換が可能である。ただしそのままでは例1のプログラムには具象化サブゴールとして再帰述語 anc^{bc} が現れてしまうため、[Mumick 90] 中のマジックコンディション法の変換可能なクラスではない。論文では、この型のプログラムに対しても拡張可能であるとしている。しかしその拡張手法は算術比較述語を対象としたものであり、EDB 述語とのジョインを一般的に扱えるものではない。

また、本論文の PT と類似のプログラムのクラスを、プログラム変換を使って効率改善する手法として [Kemp 89] において C 変換が提案されている。しかし、C 変換は複数の引数を持つ EDB 述語を対象として効率化することができない。これは、C 変換がもとと算術比較述語を対象とした変換であるためである。PT の変換は次の3つのステップで構成されている。

1. 与えられたプログラムと SIPS から、修飾された述語記号の集合を得る。
2. 得られた述語記号の集合から変換に利用するグラフを作成する。
3. グラフを利用した変換を行なう。

以下の構成を述べる。まず、第2章で幾つかの基本的な概念について説明する。第3章では、新しく拡張した修飾の定義とその伝播アルゴリズムを示す。第4章では変換に利用するグラフ (FA グラフ) の説明とそれを利用した変換アルゴリズムを示す。第5章では、変換後のプログラムを効率的に評価する評価手法について述べる。最後に第6章で今後の課題について述べる。

2 従属性グラフおよびクリークの定義

以降の章で述べる変換手法に使用する幾つかの概念について定義を示す。

定義 1. 従属性グラフ D(dependency graph)

プログラム P の従属性グラフ D とは次を満たす有向グラフである。ノードは P 中の IDB (intentional database) 述語記号である。IDB 述語とはデータベース中に陽に定義されておらずルールによってのみ定義されているものを言う。P 中のあるルールにおいて、ヘッダの述語記号が p でボディに述語記号 q が存在する場合、ノード q からノード p への有向枝 ($q \rightarrow p$) を引く。

定義 2. クリーク [Kemp 89](stratum[Mumick 90])

1. クリークとは有向グラフの最大強連結成分である。
2. クリーク Q がクリーク P よりも上のレベルにあるのは、P 中のノードの一つから Q 中のノードの一つへ有向枝で達することができる場合である。($P \rightarrow \dots \rightarrow Q$)
3. 従属性グラフ中のクリークを D クリーク、FA グラフ中のクリークを FA クリークと呼ぶ。

定義 3. 検査述語

検査述語とは、ルールに現れる EDB 述語の内、そのルールに対する sips において矢 ($\rightarrow$) の根元に現れないものを言う。

検査述語は第1章において述べた、マジックセット法では効率的に扱えないタイプの EDB 述語である。PT はこの検査述語を対象に変換を行なう。

3 修飾

この章ではまず新しい修飾を定義し、与えられたプログラムから修飾された述語記号の集合を得るアルゴリズムを示す。なお以降のアルゴリズムでは、同じ変数が一つの述語の違う引数位置に同時に現れた場合に、修飾中に eq 述語を適時導入するものとする。eq 述語とは項の間の等価関係を表す述語である。本論文では紙数の都合によりアルゴリズム中の eq 述語を扱う部分を省略しているが、これはアルゴリズムの一般性を失うものではない。

PT では、従来の bf もしくは bcf 修飾とは異なる修飾を新たに導入し使用する。新しい修飾は従来の bf (bcf) 修飾が評価時の各引数の束縛状態のみを示していたのに対し、各引数を満たす値が評価においてどの検査述語のどの引数の束縛を受けるのか、まで記述するものである。この修飾は PT の変換が、再帰計算過程におけるインバリアントに対する EDB 述語の束縛を利用するのに対応している。

定義 4 (修飾)

検査述語の修飾型とは検査述語の引数を、番号もしくは * で置き換えたものである。番号は IDB 述語の引数を表し、* は存在引数

[Ramakrishnan 88] を表している。

例) $man(1, 3, *)$

修飾集合とは、複数の検査述語の修飾型の集合である。

例) $\{man(1,3,*), salary(2,*)\}$

修飾述語記号とは、添字として修飾集合を修飾した述語記号である。述語記号 p に修飾集合 a を修飾した修飾述語記号を p_a とする時、 p_a を p の修飾バージョンと呼び、 p を p_a のオリジナルバージョンと呼ぶ。

例) $anc\{man(1,3,*), salary(2,*)\}$

ある修飾述語記号 p_a の修飾集合に含まれる検査述語は、そのオリジナルの IDB 述語 p の計算過程において、 p とジョインを行なわれる検査述語のいくつかである。どの検査述語のどの引数によって束縛されるかは、プログラム中のルールにおいて IDB 述語の引数の値がどのように渡されるかを、トップダウンにたどることで調べる。なお、質問は $query$ をヘッドに持つルールによって表すとする。

最初の修飾述語記号は $query$ をヘッドに持つルールのボディから得られる。これは次の acquire-seed 手続きを使って行なう。acquire-seed によって得られた最初の修飾述語記号 (seed) から始めて、他の IDB 述語の修飾述語記号を、ルールを使って求めていく手続きを修飾伝播と呼び、その後を示す propagate-adornment 手続きによって行なう。

acquire-seed(R, p)

(入力) ルール R , R のボディの IDB 述語 p

(出力) p の修飾バージョン p_a

修飾集合 AS を ϕ とする。 R' のボディの検査述語の内 p と共有変数を持つものを $e_1, \dots, e_m$ とする。

for $i = 1$ to m do

e_i の引数の数を n とする。 e_i の第 j 引数 ($j=1..n$) が p の第 k 引数と変数を共有している場合、 e_i の修飾型 e_i^a の第 j 引数は ' k ' とする。 e_i の第 j 引数が p と共有変数を持たない場合修飾型の第 j 引数は '*' とする。

e_i を AS に加える。

od.

AS を p の修飾とする。 □

例 2

r2.1 $query(X, Y) :- a(X, Y, X), t(X), p(X, W), q(W, Z).$
(述語 t, p, q を検査述語とする。) に対して acquire-seed 手続きを行なうことによって、修飾述語記号 $a\{eq(1,3), t(1), p(1,*)\}$ を得る。 □

次に修飾伝播手続きを示す。

propagate-adornmet(R, p_a)

(入力) ルール R , R のヘッドの述語 p に対応する修飾述語記号 p_a

(出力) ルール R のボディの複数の IDB 述語の修飾バージョンの集合 S_q

入力である修飾述語記号の修飾集合を AS_p とする。ボディに IDB 述語がないときは $S_q = \phi$ を返す。 AS_p 中の要素の引数位置の番号を、ヘッドの述語の対応する変数に書きかえる。* はルール中に現れていない相異なる変数とする。置き換えた検査述語を R のボディに加え、 R' とする。ルール R' のそれぞれのボディ内 IDB 述語 $q_1, \dots, q_n$ に対して、次のようにその修飾バージョンを求める。

for $i = 1$ to n do

$q_i^a = \text{acquire-seed}(R', q_i)$

od.

得られたすべての修飾述語記号 $q_i^a (i = 1..n)$ を要素とする集合を S_q とする。

$S_q = \{q_1^a, \dots, q_n^a\}$

□

例 2 (続き)

r2.2 $a(X, Y, Y) :- b(X, Y, Z), p(Z, W).$ に対して $a\{eq(1,3), t(1), p(1,*)\}$ を与えて propagate-adornment 手続きを実行する。述語 p が検査述語であるとする。その場合、出力の修飾述語記号は $b\{eq(1,2), t(1), p(1,*), p(3,*)\}$ となる。 □

本手法の修飾は、検査述語間の共有変数関係をも表現するように拡張することができる。そうすることによってより詳しい束縛情報を伝播させることができる。その拡張については本論文ではアルゴリズムを簡略に示すために省略する。

与えられたプログラムから修飾述語記号の集合 S を得るアルゴリズムを示す。

produce-apsset($query, P$)

(入力) プログラム P

(出力) P に対応する修飾述語記号集合 S

step 1. $S = \phi$ とする。

query をヘッドに持つルール $R_i (i = 1..)$ とそれぞれの R_i のボディの複数の IDB 述語 $p_{ij} (j = 1..)$ より、それぞれの p_{ij} の修飾述語記号を得る。

$p_{ij}^a = \text{acquire-seed}(R_i, p_{ij})$

得られたすべての修飾述語記号の集合を ΔS とする。

$S = \Delta S$

step 2. ΔS の要素である $p_k^a (k = 1..n')$ に対して、それぞれ p_k をヘッドに持つ複数のルール $R_{kl} (l = 1..n')$ に与えて新たな修飾述語記号を得る。

$S_{q-kl} = \text{propagate-adornment}(p_k^a, R_{kl})$

得られたすべての S_{q-kl} の和集合を newS とする。

$S' = \text{newS} \cup S$

$\Delta S = S' \setminus S$

$S = S'$

step 3. $\Delta S = \phi$ ならば、終了。さもなければ、step 2. へ。

□

例 3

次のプログラムに対して produce-apsset 手続きを実行する。

r3.1 $\text{query}(X, Y) :- a(X, Y), t(X).$

r3.2 $\text{query}(X, Y) :- b(X, Y), t(Y).$

r3.3 $a(X, Y) :- b(X, W), p(W, Y).$

r3.4 $b(X, Y) :- p(X, W), a(W, Y), t(W).$

r3.5 $b(X, Y) :- p1(X, Y).$

述語 $t, p, p1$ を EDB 述語、その内述語 t, p を検査述語とする。

step 1. まず、query をヘッドに持つルール r3.1, r3.2 より、seed を得る。

$\Delta S = \{a_{\{t(1)\}}, b_{\{t(2)\}}\}$

$S = \Delta S$

step 2. 述語 a, b をヘッドに持つルール r3.3, r3.4, r3.5 より新たな修飾述語記号を得る。

$\text{newS} = \{a_{\{p(\cdot, 1), t(1), t(2)\}}, b_{\{p(2, \cdot), t(1)\}}\}$

$S' = \text{newS} \cup S$

$\Delta S = S' \setminus S = \text{newS}$

$S = S'$

step 3. ΔS が空でないので、step 2 に戻る。

step 2. 再び ΔS とルール r3.3, r3.4, r3.5 より、新しい ΔS を求める。

$\Delta S = \{a_{\{p(\cdot, 1), t(1), p(2, \cdot)\}}, b_{\{p(2, \cdot), p(\cdot, 1), t(1)\}}\}$

step 3. ΔS が空でないので再び step 2.

step 2.

$\Delta S = \{\}$

step 3. 手続き produce-apsset を終了する。

$S = \{a_{\{t(1)\}}, b_{\{t(2)\}}, a_{\{p(\cdot, 1), t(1), t(2)\}}, b_{\{p(2, \cdot), t(1)\}},$

$a_{\{p(\cdot, 1), t(1), p(2, \cdot)\}}, b_{\{p(2, \cdot), p(\cdot, 1), t(1)\}}\}$

□

4 FA グラフの作成およびグラフを利用した変換手法

PT では、検査述語による束縛の伝播関係を表すグラフを作成し、それを変換に利用する。そのグラフを FA (full adornment) グラフと呼ぶ。FA グラフの定義を示す。

定義 5 (FA グラフ)

FA グラフのノードは修飾述語記号集合 S の要素および query 述語記号である。有向枝は次の 2 種類である。

1. produce-apsset 手続きの step 1. で acquire-seed 手続きを呼び出した時に、入力としてルール R , 出力として p_a を得たならば、 $(p_a \xrightarrow{R} \text{query})$ なる有向枝を引く。
2. produce-apsset 手続きの step 2. で propagate-adornment 手続きを呼び出した時に、 $S_q = \text{propagate-adornment}(p, R)$ であって、 $q \in S_q$ であるならば、ノード q からノード p への有向枝 ($q \xrightarrow{R} p$) を引く。

□

実際には、FA グラフの作成は第 3 章の produce-apsset 手続きを行なっている時になされる。

例 6 に対応する FA グラフを図 1 に示す。

次に PT の変換の中の、新しいルールを作成する部分の手続きを示す。

produce-define-rule(p_a)

(入力) 修飾述語記号 p_a

(出力) 述語 p_a を定義するルール R

ルール R のヘッドの述語記号は入力として与えられた修飾述語記号とする。ルール R のヘッドの述語の引数はすべて異なる変数とする。ルール R

のボディは入力の変修述語記号のオリジナルバージョンの述語 p と、修飾集合中の検査述語の連言であるとする。ボディのオリジナル述語 p の引数はヘッドと同じとする。検査述語の引数は、番号に関しては、番号が i の場合はヘッドの第 i 番めの引数の変数に置き換える。* はルール中に現れていない相異なる変数とする。なお、修飾集合が空の場合は、検査述語の連言はつけない。

□

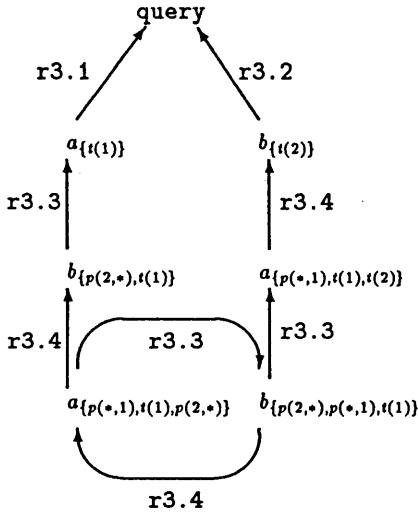


図 1. FA グラフ

PT の変換では、良く知られた展開 / 畳み込み変換 [Tamaki 84]、とゴール挿入変換 [Tamaki 85] を組み合わせて使用している。ただしゴール挿入変換に関しては PT で必要である変換に限定して、畳み込みと共に使用している点で [Tamaki 85] と違う。変換で使用するゴール挿入と畳み込みを一組にしてゴール挿入畳み込みとし、その手続きを示す。

insert-goal-fold(C, D)

(入力) 畳み込まれるルール C 、畳み込むルール D (D は produce-define-rule によって作られたルール)

(出力) ゴール挿入と畳み込みをなされたルール C''

C を $A :- K, L$ の形、 D を $B :- K', E'$ の形のルールとする。なお、2つのルールは変数を共有しないように、必要ならば前もって前もって名前替えを行なうものとする。ここで K, K' は引数の異なる同じ IDB 述語とし、 E' は検査述語の連

言とする。入力の 2つのルールが次の条件を満たしている時に変換することができる。

条件 (1) $K'\theta = K$ を満たす $mgu\theta$ が存在する。

条件 (2) $K^{a1} = \text{acquire-seed}(C, K)$, $K^{a2} = \text{acquire-seed}(D, K')$. K^{a1}, K^{a2} の修飾集合をそれぞれ $a1, a2$ とする。その時 $a1 \subseteq a2$ である。

条件を満たしている時、次の変換を行なう。

step 1. $E'\theta$ をルール C にゴール挿入する。

$C' : A :- K, E'\theta, E, L$.

step 2. ルール D をルール C' の K & $E'\theta$ の部分に畳み込む。

$C'' : A :- B\theta, E, L$.

□

上記の変換ではルール C 中の述語は K 以外は消去されていないが、畳み込みによって L 中の幾つかの検査述語を消去することができる。本論文では、その削除可能な述語の判別手続きに関しては紙数の都合で省略する。

変換で使用するグラフに関する概念を新たに 2つ示す。

定義 5. (接続ノード集合 SC_N)

FA グラフ中のノードの内ではいずれの FA クリークにも含まれないノードの内 query ノード以外の集合を接続ノード集合 SC_N とする。

定義 6. (孤立ノード)

FA グラフ中で SC_N に属するノードの内、 SC_N 中の他の要素に入る有向枝を持たないものを孤立ノードとする。

変換手続きを示す。

PT-transform(F, P_0)

(入力) プログラム P_0 と、対応する FA グラフ F

(出力) 変換されたプログラム P_1

step 1. 接続ノード集合 SC_N 以外のすべてのノードの表す修飾述語記号に対して produce-define-rule 手続きを使ってルールを作成する。生成されたすべてのルールの集合を D とする。

step 2. 孤立ノード s に対して、ノード s へ入ってくる有向枝として (有向枝群 1) $l_1 \xrightarrow{R_{11}} s, \dots, l_n \xrightarrow{R_{1n}} s$ が存在し、ノード s を出て行く有向枝として (有向枝群 2) $s \xrightarrow{R_{01}} o_1, \dots, s \xrightarrow{R_{0m}} o_m$ が存在するとする。

ルール $R_{0j} (j = 1..m)$ を、ルール $R_{1i} (i = 1..n)$, および R_{1i} と同じヘッドを持つルールで展開し、 R_{1i} による展開後のルールを R_{ij} とする。

ノード s および有向枝群 1, 2 を FA グラフからすべて消去し、それぞれのルール $R_{ij} (i = 1..n, j = 1..m)$ に対応して有向枝 $l_i \xrightarrow{R_{ij}} o_j$ を加える。

step 3. step 2 において、 $l_1, \dots, l_n$ の内、いずれかの FA クリークに属しているノード l_k に対しては ($l_k \xrightarrow{R_{1k}} s$), R_{1k} で展開したルール $R_{kj} (j = 1..m)$ に l_k を定義しているルールをゴール挿入畳み込みする。

step 4. step 2 で書き換えた FA グラフに孤立ノードが存在するならば step 2 に戻る、孤立ノードがなければ step 5 へ進む。

step 5. F 中のすべての FA クリークに属するノード q に対して、ノード q に入る枝の内、ラベルが R で根元のノードがいずれかのクリークに属している有向枝

$$p_1 \xrightarrow{R} q, p_2 \xrightarrow{R} q, \dots, p_{n'} \xrightarrow{R} q$$

がある場合には、 q を定義しているルールを R , および R と同じヘッドを持つルールで展開し、 R で展開したルールに、 $p_1, \dots, p_{n'}$ を定義しているルール $D_i (i = 1..n')$ をゴール挿入畳み込み (insert-goal-fold) する。それぞれの畳み込んだルールを R_i とする。グラフ中の有向枝のラベルを R からそれぞれ R_i に置き換える。

□

例 3 (続き)

変換を例 3 のプログラムを使って説明する。

なお、 $a_{\{t(1)\}} = a_1, b_{\{t(2)\}} = b_1, a_{\{p(*,1),t(1),t(2)\}} = a_2, b_{\{p(2,*),t(1)\}} = b_2, a_{\{p(*,1),t(1),p(2,*)\}} = a_3, b_{\{p(2,*),p(*,1),t(1)\}} = b_3$ と略記する。

(step 1.) クリークを構成するノードの表す各修飾述語記号に対して、ルールを作成する。

$$r3.6 \quad a_3(X, Y) :- a(X, Y), p(V1, X), t(X), p(Y, V2).$$

$$r3.7 \quad b_3(X, Y) :- b(X, Y), p(Y, V1), p(V2, X), t(X).$$

(step 2.) 図 1 の FA グラフにおいて孤立ノードは a_1, b_1 である。それぞれのノードから出ていくルールを入ってくるルールで展開する。この場合は $r3.1$ を $r3.3$, $r3.2$ を $r3.4, r3.5$ で展開する。

$$r3.8 \quad query(X, Y) :- b(X, W), p(W, Y), t(X).$$

$$r3.9 \quad query(X, Y) :- p(X, W), a(W, Y), t(W), t(Y).$$

$$r3.10 \quad query(X, Y) :- p1(X, Y), t(Y).$$

(step 3.) a_1, b_1 それぞれに入ってくるノード b_2, a_2 は FA クリークに属していないので、何もしない。

(step 4.) step 2 において書き換えられた FA グラフに孤立ノード b_2, a_2 が存在するので、step 2 に戻る。

(step 2.) $r3.8$ を $r3.4, r3.5$ で、 $r3.9$ を $r3.3$ で展開する。

$$r3.11 \quad query(X, Y) :- p(X, V), a(V, W), t(V), p(W, Y), t(X).$$

$$r3.12 \quad query(X, Y) :- p1(X, W), p(W, Y), t(X).$$

$$r3.13 \quad query(X, Y) :- p(X, W), b(W, V), p(V, Y), t(W), t(Y).$$

(step 3.) ノード b_2, a_2 に入ってくるノード a_3, b_3 は共に FA クリークに属しているので、 $r3.11, r3.13$ にそれぞれ $r3.6, r3.7$ をゴール挿入畳み込みする。

$$r3.14 \quad query(X, Y) :- p(X, V), a_3(V, W), p(W, Y), t(X).$$

$$r3.15 \quad query(X, Y) :- p(X, W), b_3(W, V), p(V, Y), t(Y).$$

(step 4.) step 2 において書き換えられた FA クリークには、孤立ノードがないので、step 5 へ進む。

(step 5.) クリークを構成するノードに対して変換を行なう。 a_3 を定義している $r3.6$ を $r3.3$ で、 b_3 を定義している $r3.7$ を $r3.4, r3.5$ で展開する。

$$r3.16 \quad a_3(X, Y) :- b(X, W), p(W, Y), p(V, X), t(X), p(Y, V2).$$

$$r3.17 \quad b_3(X, Y) :- p(X, W), a(W, Y), t(W), p(Y, V1), p(V2, X), t(X).$$

$$r3.18 \quad b_3(X, Y) :- p1(X, Y), p(Y, V1), p(V2, X), t(X).$$

展開したルール $r3.16, r3.17$ に、 b_3, a_3 を定義しているルール $r3.7, r3.6$ をそれぞれゴール挿入畳み込みする。

$$r3.19 \quad a_3(X, Y) :- b_3(X, W), p(W, Y), p(Y, V).$$

$$r3.20 \quad b_3(X, Y) :- p(X, W), a_3(W, Y), p(V, X), t(X).$$

変換を終了する。変換後のプログラムと FA グラフを示す。

$query(X, Y) \text{ :- } p1(X, Y), t(Y).$
 $query(X, Y) \text{ :- } p1(X, W), p(W, Y), t(X).$
 $query(X, Y) \text{ :- } p(X, V), a_3(V, W), p(W, Y),$
 $t(X).$
 $query(X, Y) \text{ :- } p(X, W), b_3(W, V), p(V, Y),$
 $t(Y).$
 $b_3(X, Y) \text{ :- } p1(X, Y), p(Y, V1), p(V2, X),$
 $t(X).$
 $a_3(X, Y) \text{ :- } b_3(X, W), p(W, Y), p(Y, V).$
 $b_3(X, Y) \text{ :- } p(X, W), a_3(W, Y), p(V, X),$
 $t(X).$

□

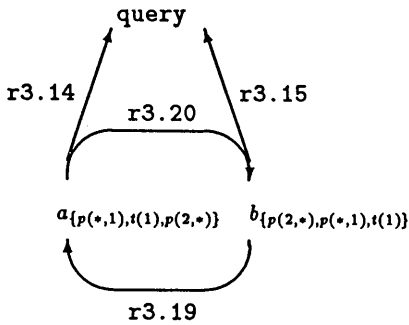


図 2. 変換後の FA グラフ

PT-transformation 手続きに対して、次の定理が成り立つ。

定理 1.

プログラム P を入力として、PT-transformation 手続きによりプログラム P_t を得たならば、P と P_t は質問等価 (query equivalent [Ramakrishnan 88]) である。□

証明は [Tamaki 84] の変換の正当性証明のテクニックを利用している。特徴はゴール挿入量込み込みが重み完全性 [Tamaki 84] を保存することを示している点である。

5 変換後のプログラムの評価手法

論理プログラムをボトムアップ評価する代表的な評価法としてセミナティブ法 [Bancilhon 85] が提案されている。しかし、PT の変換後のプログラムはそのままセミナティブ法で評価すると効率が悪く場合がある。そこで我々は PT によって変換されたプログラムを効率的に評価する手法を考案した。本章ではその評価法について述べる。

本論文で提案した PT は、再帰的な生成手続きを分割してゆく手法と考えることができる。これは、解の生成過程において一般的な関係を生成した後に必要な関係を抽出するのではなく、最終的に必要な関係の範囲がある程度わかるならば、生成される関係の範囲を限定して、分割して生成を行なう方が効率的である、という考え方に基づいている。

しかし分割したことによって重複した計算を行ってしまう場合がある。

例 4

オリジナルのプログラムが関係 a のみを生成しているのに対し、変換後のプログラムは

$a_{t(1)}$, $a_{t(2)}$ という 2 つの関係を生成するとする。この 3 つの関係はベン図で示すと次のような関係にある。

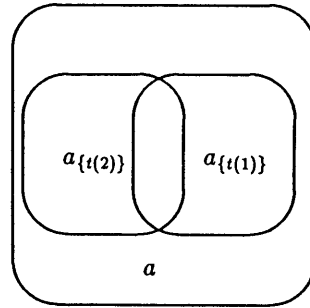


図 3

変換後のプログラムが計算するそれぞれの関係

$a_{t(1)}$, $a_{t(2)}$ は、オリジナルの関係 a よりも小さいが、2 つは別々に計算されるので、共通部分 $a_{t(1)} \cap a_{t(2)}$ を重複して計算してしまう。この重複する計算量が多い場合にはセミナティブ評価法をそのまま適用すると効率的でない。□

例 4 のような重複した計算を行なわないようにする方法の 1 つとして、共通部分を先に計算しその後で残りの部分を計算するという方法が考えられる。

我々の考案した評価法は、例 4 のような場合にはまず共通部分 $a_{t(1)} \cap a_{t(2)}$ つまり $a_{t(1),t(2)}$ を生成するルールを作成し、先に評価しておき、後で残りの部分を計算する。この評価法を我々は階層的評価法と呼ぶ。

なお、ルールの階層レベルは、関係 A と関係 B を生成するルールと、その共通部分の関係 $A \cap B$ を生成するルールがある場合、前者は後者より 1 レベル上の階層とする。

階層的評価法は次の 3 ステップで構成されている。

step 1. D グラフのそれぞれの D クリークに対応する修飾述語記号に対し、その下層レベルの修飾述語記号の集合を作成する。

step 2. step 1 で作成した下層レベルの修飾述語集合に属する修飾述語記号の表す関係を生成するルール、および生成した関係を一つ上のレベルに受け渡すルールを作成する。

step 3. 階層化されたルール群を下層より順にセミナイブ法によりボトムアップ評価する。

階層的評価法が重複した計算を行わないのは、各層のルール間では再帰ルールのインバリエントに対して束縛が異なるという理由による。

本論文では、階層的評価法のアルゴリズムは紙数の都合で省略し、例を使って概略を説明するにとどめる。

例 5

$query(X, Y, Z) :- a(X, Y, Z), t(X, Y, Z).$
 $a(X, Y, Z) :- p(X, W), a(W, Z, Y).$
 $a(X, Y, Z) :- pl(X, Y, Z).$

t, p, pl を EDB 述語とし、その内 t, p を検査述語とする。

PT はこのプログラムを次のように変換する。

$query(X, Y, Z) :- p(X, W), p(W, V), a_3(V, Y, Z),$
 $t(X, Y, Z).$
 $query(X, Y, Z) :- pl(X, Y, Z), t(X, Y, Z).$
 $query(X, Y, Z) :- pl(W, Z, Y), p(X, W),$
 $t(X, Y, Z).$
 $a_3(X, Y, Z) :- p(W, X), p(X, V), a_4(V, Z, Y).$
 $a_3(X, Y, Z) :- pl(X, Y, Z), p(W, X),$
 $t(V, Y, Z).$
 $a_4(X, Y, Z) :- p(W, X), p(X, V), a_3(V, Z, Y).$
 $a_4(X, Y, Z) :- pl(X, Y, Z), p(W, X),$
 $t(V, Z, Y).$

変換後のプログラムは $a_3(a_{\{t(*,2,3),p(*,2)\}})$ と $a_4(a_{\{t(*,3,2),p(*,1)\}})$ を別々に計算するプログラムになっている。

この変換後のプログラムを階層的評価法は次の 3 つのルール群に変換する (step 1, step 2)。

ルール群 1

$query(X, Y, Z) :- p(X, W), p(W, V),$
 $a_3(V, Y, Z), t(X, Y, Z).$
 $query(X, Y, Z) :- pl(X, Y, Z), t(X, Y, Z).$
 $query(X, Y, Z) :- pl(W, Z, Y), p(X, W),$
 $t(X, Y, Z).$

ルール群 2

$a_3(X, Y, Z) :- p(W, X), p(X, V), a_4(V, Z, Y).$
 $a_3(X, Y, Z) :- pl(X, Y, Z), not(a_{5-root}(X, Y, Z)),$
 $p(W, X), t(V, Y, Z).$
 $a_4(X, Y, Z) :- p(W, X), p(X, V), a_3(V, Z, Y).$
 $a_4(X, Y, Z) :- pl(X, Y, Z), not(a_{5-root}(X, Y, Z)),$
 $p(W, X), t(V, Z, Y).$

ルール群 3

$a_5(X, Y, Z) :- p(X, W), a_5(W, Z, Y), p(V, X).$
 $a_{5-root}(X, Y, Z) :- pl(X, Y, Z), t(V, Y, Z),$
 $t(W, Z, Y), p(U, X).$
 $a_5(X, Y, Z) :- a_{5-root}(X, Y, Z).$
 $a_3(X, Y, Z) :- a_5(X, Y, Z).$
 $a_4(X, Y, Z) :- a_5(X, Y, Z).$

ルール群 3 は a_3 と a_4 の共通部分である a_5 を生成するルールおよび、その生成した関係を上のレベルに渡すルールより成っている。ルール群 2 はルール群 3 の後を受けて、関係 a_3 と a_4 を生成するルールである。ルール群 1 は生成された関係 a_3 やその他の EDB 述語から、質問の解を生成するルールである。

階層化の後のプログラムに a_{5-root} なる述語が現れている。これは、各レベルの再帰ではないルールが、重複した計算を行なうのを防ぐために導入した述語である。

階層的評価法の step 3 で、階層化されたプログラムを下のレベルのルール群 (ルール群 3) から順にセミナイブ法によって評価して行く。こうすることによって重複した計算をさけることができる。□

6 おわりに

本論文では、与えられたプログラム中の再帰定義された述語の計算過程で、インバリエントになっている引数に対する、EDB 述語の束縛を計算の早い段階で使用するようプログラムを変換することによって、効率を改善する手法 (Propagating Test predicates, PT) を提案した。

また階層的評価法と呼ぶ、変換後のプログラムを効率的に評価する評価法について述べた。

ここでもう 1 つ階層的評価法以外の評価時の工夫の必要性について述べる。例 1 の PT による変換後のプログラムにはルールのボディに述語 *test* の現れる回数が増えている。この例に限らず、PE による変換後のルールにはボディに検査述語の現れる回数が増える。この増えた検査述語とのジョインの計算量が大きければ PT による変換後のルールは効率的ではない。

しかしこの問題に対しては次のように考えることが

できる。まず、検査述語とのジョインは増えているが、各回のジョインされるタプル数はPTの変換の性質から、もとの検査述語とのジョインに比べて少なくなる。しかし、ジョインされるタプルの総数が多い場合はありうる。ただしそのような場合にも、第1に新たに増えた検査述語とのジョインは変換中の存在引数の導入によって、必ずセミジョインになっている。セミジョインは一般的なジョインに比べて少ない計算量で実現できる[Ullman 89]。第2に、検査述語の表す関係の各引数には、インデックスを施すと仮定することができる。インデックスジョインは非常に高速に実現できる[Ullman 89]。

最後に現在の手法の問題点と課題を述べる。

PTによる変換では展開によってもともと1つであった述語が複数に分解される。これは好ましくない効果を変換後のプログラムに対し与えることがある。例えば捕捉マジックセット法[Beeri 87]では捕捉述語を導入して計算の重複を防いでいる。このような効果を現在のPTは阻害してしまう。PTをマジックセット法をはじめとする他の効率改善手法と融合して行くためには、このミスマッチを取り除く必要がある。

またPTを実際の演繹データベースシステムにおいてテストすることで、その効果や問題点を明らかにすることがもう一つの今後の課題である。

謝辞

九州大学情報処理教育センターの高木利久助教授には、変換手法の拡張の方向に対して重要な助言を、また本論文の基本的な構成に示唆を頂いたことに感謝致します。また九州大学大学院の鈴木孝彦氏には、変換後のプログラムの評価法について重要な意見を、さらに手法の問題点を指摘して頂いたことに感謝致します。そしてこの論文に対し活発な意見を交わして頂いた高木ゼミのみなさんに感謝します。

参考文献

- [Bancilhon 85] Bancilhon, F.: Naive Evaluation of Recursively Defined Relations, On Knowledge Base Management Systems - Integrating Database and AI Systems, Brodie and Mylopoukos, Eds., Springer - Verlag.
- [Beeri 87] Beeri, C. and Ramakrishnan, R.: On the Power of Magic, Proc. 6th PODS., pp.269-283, San Diego, 1987.
- [Kemp 89] Kemp, D.B. and Ramamohanarao, K. and Balbin, I. and Meenakshi, K.: Propagating Constraints in Recursive Deductive Databases, North American Logic Programming Conference, 1989.
- [Mumick 90] Mumick, I. S. and Finkelstein, S.J. and Pirahesh, H. and Ramakrishnan, R.: Magic Conditions, Proc. PODS., 1990.
- [Tamaki 84] Tamaki, H. and Sato, T.: Unfold/fold Transformation of Logic Programs, Proc. 2nd International Logic Programming Conference, pp.127-138, Uppsala, 1984.
- [Tamaki 85] Tamaki, H. and Sato, T.: Equivalence Preserving Logic Program Transformation and its Application to Program Synthesis, J.IPS Japan, Vol.26, No.11, pp.1423-1431, 1985.(in Japanese)
- [Ramakrishnan 88] Ramakrishnan, R. and Beeri, C. and Krishnamurthy, R.: Optimizing Existential Datalog Queries, Proc. 7th PODS., pp.89-102, Austin, 1988.
- [Tsutsumi 90] Tsutsumi, F. and Takagi, T. and Ushijima, K.: An Effective Program Transformation of Logical Recursive Queries in Deductive Databases, Proc. of the FAR-EAST Workshop on Future Database Systems, pp.13-25, Melbourne, 1990.
- [Ullman 89] Ullman, J.D.: Principles of database and knowledge-base systems vol. 1 vol. 2, Computer science press, 1989.