1N-8

演繹データベースにおける再帰質問のボトムアップ処理のためのプログラム変換手法

堤富士雄　鈴木孝彦　高木利久　牛島和夫
(九州大学)

1．はじめに

論理型プログラムを生成／検査型プログラムとみなし，効率を改善するプログラム変換手法がこれまで研究されている［1］．しかしそれらの多くは，トップダウン処理を前提とした手法であり，ボトムアップ処理を前提としたものではない．そこで本稿では，ボトムアップ処理を前提とするプログラム変換手法について述べる．本手法は，展開／畳み込み変換を用いるという点では従来のトップダウンのための手法と同じである．本手法の特徴は次の４点である．

（１）ボトムアップ処理を前提とすることで，これまで改善できなかったタイプのプログラムに対しても改善できる．
（２）プログラムの修飾（adornment）を用いて，適用できるプログラムのクラスを簡単に特定することができる．
（３）従来の展開／畳み込みに加えて，存在引数（existential argument）を考慮することで効率的な変換を可能にしている．存在引数とは，全解探索を必要としない引数である．存在引数にたいしては効率的な処理法が研究されている［2］．
（４）マジックセット法［3］との親和性がある．

2．ボトムアップ処理と生成／検査型プログラム

演繹データベースにおけるボトムアップ処理では，まずルールのボディの述語を満たすファクト（タプル）がデータベースに存在するか否かを評価する．そしてボディの述語が全て満足されたならばらば，ヘッドの述語に相当するファクトを新たに生成する．この操作を，新しいファクトが生成されなくなるまで繰り返す．

この処理は，広い意味での生成／検査型処理と見なすことができる．生成／検査型処理の効率改善方針は，解の候補となるデータをすべて作り出す前になるべく早く検査を行い，解の候補を絞り込むことである．再帰ルールをボトムアップ処理する際には，多量のファクトが生成されるため，この生成をいかに少なくするかが重要である．

今回の手法は、データベースに定義されているＥＤＢ(extentional database)述語に対する変数束縛をなるべく早く求め，それを検査することによってファクトの生成を抑えることを目的とする．

3．ボトムアップ処理とプログラム変換

本稿の目的は次のようなプログラム変換を定式化することにある．

例Ａ．変換前
```
A-1. query(X,Y):-a(X,Y),t(Y).
A-2. a(X,Y):-p1(X,Y).
A-3. a(X,Y):-p(X,Z),a(Z,Y).
```
t,p,p1はＥＤＢ述語，質問は?-query(X,Y).である．

これは，関係aのグランドインスタンス（以下ファクトaと呼ぶ）を生成し，tによって検査するプログラムとみなすことができる．このプログラムを以下のように変換することによって効率の改善可能である．

変換後
```
A-1'. query(X,Y):-a'(X,Y).
A-2'. a'(X,Y):-p1(X,Y),t(Y).
A-3'. a'(X,Y):-p(X,Z),a'(Z,Y).
```
変換後のプログラムは変換前と一様質問等価(uniformly query equivalent［2］)である．

この変換は，ファクトaをすべて生成し，その後tの検査をしていたプログラムを，ファクトa一つ生成する度にtを検査するように書き換えることによって，効率を改善している．なお，この改善はt,p,p1の関係の大きさに依存しない．本手法の基本戦略は例Ａのように，ＥＤＢ述語による検査をファクトの生成過程（特に再帰定義された）において，なるべく早く行おうとするものである．しかしこの変換がいつも効率を改善するとは限らない．次の例を考えてみよう．

例Ｂ．変換前
```
B-1. query(X,Y):-a(X,Y),t(X).
B-2. a(X,Y):-p1(X,Y).
B-3. a(X,Y):-p(X,Z),a(Z,Y).
```
t,p,p1はＥＤＢ述語，質問は?-query(X,Y).

変換後
```
B-1'. query(X,Y):-a'(X,Y).
B-2'. a'(X,Y):-p1(X,Y),t(X).
B-3'. a'(X,Y):-p(X,Z),t(X),a(Z,Y).
B-2. a(X,Y):-p1(X,Y).
B-3. a(X,Y):-p(X,Z),a(Z,Y).
```
例Ａでは，畳み込みによって述語aを消すことができた．しかし例Ｂでは，ルールB-3'にtと共有変数を持たないリテラルaが現れているため，消去できない．変換後のプログラムでもやはりファクトaをすべて生成してしまうため，効率は改善できていない．ところが，例Ｂのプログラムをトップダウン処理すると，変換後のプログラム（制御を考慮してルールB-3'のtとaの位置を入れ換えている）はtの検査を通過したものに関してのみaを展開するようになっており，探索空間が小さくなっている．一方，例Ａのプログラムに関してはトップダウン処理の効率は改善されていない．

これらの例からわかるようにボトムアップ処理とトップダウン処理とで変換によってプログラムの効率が改善できるクラスが異なる．

4．本手法が適用できるクラス

再帰定義されたプログラムを展開／畳み込みを使って変換する手法では畳み込みが可能かどうか判定する一般的な手法はない（例Ｂではできなかった）．しかし，ボトムアップを前提とする本手法においては，プログラムに修飾（adornment）を施すことで，本手法が適用可能か否かを，実際に手法を適用する前に判断することができる．修飾例を示す．

A Program Transformation for Bottom-up Computation of Recursive Queries on Deductive Databases
Fujio TSUTSUMI, Takahiko SUZUKI, Toshihisa TAKAGI, Kazuo USHIJIMA
KYUSHU UNIVERSITY

例C．
C-1. query(X,Y):-a(X,Y),b(X,Y).
C-2. a(X,Y):-p(X,Z),a(Z,Y).
C-3. a(X,Y):-p1(X,Y).

b,p,p1はＥＤＢ述語である．bの検査をaの生成過程において早く行うことを目的として，aに修飾を行う．

まず，ルールC-1においてボディのaとbが共有変数を持っているので，

C-1'. query(X,Y):-a^{b1b2}(X,Y),b(X,Y).

と修飾する．a^{b1b2}とはaの第１，２引数がそれぞれbの第１，２引数によって検査される必要があることを示している．この修飾をC-2のヘッドに適用し，ボディのaを共有変数を考慮して修飾する．

C-2'. a^{b1b2}(X,Y):-p(X,Z),a^{*b2}(Z,Y).

a^{*b2}の*はbによって検査される必要が無いことを示している．同様にC-3を修飾する．

C-3'. a^{b1b2}(X,Y):-p1(X,Y).

修飾によって新しくa^{*b2}が生まれたので，この修飾を再びC-2,3に適用し

C-2''. a^{*b2}(X,Y):-p(X,Z),a^{*b2}(Z,Y).
C-3''. a^{*b2}(X,Y):-p1(X,Y).

となる．新しい修飾が出現しなかったので，修飾を終る．プログラムの修飾を終った時点で，a^{**}が出現しないかぎり本変換手法は適用可能である．a^{**}ができると，bによる検査を行う前にファクトを全て生成しなければならないからである．修飾による本変換手法が適用可能か否かを検査するアルゴリズムは，否定と関数を含まないホーン節プログラム一般に対して適用でき，また必ず停止する．

５．変換手法

変換は**step1-5**（表１）に従って行う．

例Ｃのプログラムを用いて変換手法を以下に説明する．例Ｃの修飾後のプログラムを再掲する．

D-1. query(X,Y):-a^{b1b2}(X,Y),b(X,Y).
D-2. a^{b1b2}(X,Y):-p(X,Z),a^{*b2}(Z,Y).
D-3. a^{b1b2}(X,Y):-p1(X,Y).
D-4. a^{*b2}(X,Y):-p(X,Z),a^{*b2}(Z,Y).
D-5. a^{*b2}(X,Y):-p1(X,Y).

まず**step1**を適用する．例ではｔはb，ｐはaであり，共有変数はX,Yである．a^{b1b2}とbを組み合わせて

D-6. a'^{b1b2}(X,Y):-a^{b1b2}(X,Y),b(X,Y).

を定義する．

step2を適用する．bに存在引数が無いので，そのままD-1にD-6を畳み込む．

D-1'. query(X,Y):-a'^{b1b2}(X,Y).

step3を適用する．例ではD-6をD-2,3で展開する．

D-2'. a'^{b1b2}(X,Y):-p(X,Z),a^{*b2}(Z,Y),b(X,Y).
D-3'. a'^{b1b2}(X,Y):-p1(X,Y),b(X,Y).

step4を適用する．例ではD-2'のボディに②の形があらわれたので**step1**にもどり

D-7. a'^{**b2}(X,Y):-a^{*b2}(X,Y),b(W,Y).

を定義する．

再び**step2**を適用する．変数Wは存在引数であるのでD-2'のボディに述語b(W,Y)を追加し，

D-2''. a'^{b1b2}(X,Y)
:-p(X,Z),a^{*b2}(Z,Y),b(W,Y),b(X,Y).

とし，D-2''にD-7を畳み込む．

D-2'''. a'^{b1b2}(X,Y):-p(X,Z),a'^{**b2}(Z,Y),b(X,Y).

step3を適用する．D-7をD-4,5で展開する．

表１　変換アルゴリズム

step1. ＥＤＢ述語ｔと，ルールｒi(i=1..n)のボディにおいてtと共有変数を持つ再帰定義された述語ｐを組合わせて，新しい述語ｐ'を定義する(定義節ｒ')．ｐ'の引数はｐのそれと同じにする．
step2. すべてのルールｒi(i=1..n)にｒ'を畳み込む．定義節ｒ'のｔに存在引数がある場合は存在引数を含むリテラルｔをｒiに追加して，畳み込む．
step3. ｒ'内のｐを，ｐの定義節で展開する．
step4. 展開することによってできたｐ'の定義節ｒi'(i=1..m)においてｐが現れる場合（再帰）は
①ｔと共有している変数がルールｒ'と同じ引数位置である時，step5へ進む．
②違う引数位置だが共有変数はある時，step1へもどる．
step5. ｒi'にｒ'を畳み込みｐの出現を取り除く．

D-4'. a'^{**b2}(X,Y):-p(X,Z),a'^{**b2}(Z,Y),b(W,Y).
D-5'. a'^{**b2}(X,Y):-p1(X,Y),b(W,Y).

D-4'に**step4**の①の形が現れたので**step5**の処理をする．

step5を適用する．D-7をD-4'に畳み込む

D-4''. a'^{**b2}(X,Y):-p(X,Z),a'^{**b2}(Z,Y).

変換を終了．変換後のプログラムは

D-1'. query(X,Y):-a'^{b1b2}(X,Y).
D-2'''. a'^{b1b2}(X,Y):-p(X,Z),a'^{**b2}(Z,Y),b(X,Y).
D-3'. a'^{b1b2}(X,Y):-p1(X,Y),b(X,Y).
D-4''. a'^{**b2}(X,Y):-p(X,Z),a'^{**b2}(Z,Y).
D-5'. a'^{**b2}(X,Y):-p1(X,Y),b(W,Y).

ルールD-5'における変数Wは，存在引数であるため１つだけ解を求めればよい．これにより述語bによる検査の効率化がはかれる．

ここで述べた変換手法は相互再帰的に定義された述語を含むプログラムに対しても拡張可能である．

６．おわりに

再帰質問をボトムアップ処理を前提としてプログラム変換する手法を述べた．本変換手法を適用できるプログラムのクラスは簡単に特定できる．しかしＥＤＢ述語による検査をファクトの生成過程に融合するという戦略は，４節で述べた適用可能なクラス以外のプログラムに対しても，有効な場合があり，それをも含むように手法を拡張することを検討している．

謝辞

本研究に対し有益な助言をいただいた九州大学工学部吉田紀彦助手に感謝いたします．

参考文献

［１］淵一博監修　古川康一　溝口文雄共編：プログラム変換，知識情報処理シリーズ７，共立出版株式会社，1987.
［２］Raghu Ramakrishnan and Catriel Beeri and Ravi Krishnamurthy:Optimizing Existential Datalog Queries,Proc.7th PODS.,Austin,1988,pp.89-102.
［３］Catriel Beeri and Raghu Ramakrishnan:On the Power of Magic,Proc. 6th PODS.,San Diego,1987, pp.269-283.