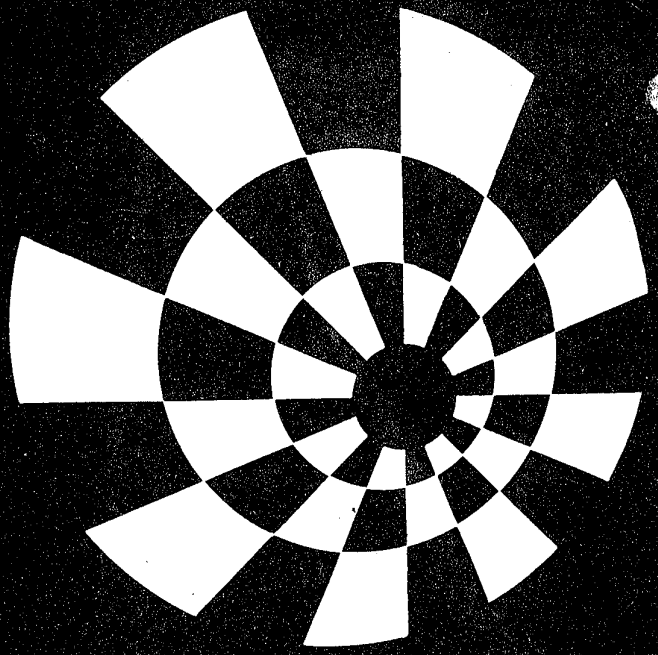

3º SIMPÓSIO BRASILEIRO de BANCO de DADOS

23.25 de março de 1988
Recife - Pernambuco

ANAIS

PROCESSAMENTO DE CONSULTAS RECURSIVAS: UMA DISCUSSÃO

Rosana de Saldanha da Gama Lanzelotte

Depto.Informática - PUC - RJ

SUMÁRIO

Os Sistemas de Bancos de Dados Relacionais são dotados de linguagens de consultas limitadas, através das quais não é possível expressar consultas recursivas. Por outro lado, os bancos de dados dedutivos envolvem um conhecimento implícito, frequentemente expresso através de regras recursivas. Torna-se necessário então estender os processadores de consulta no sentido de atender à recursividade. O presente trabalho apresenta tres propostas para o processamento de consultas recursivas, e mostra o seu funcionamento através da aplicação a um exemplo típico.

1. Introdução.

As linguagens de consulta oriundas do cálculo relacional, as mais usadas hoje em Sistemas de Gerência de Bancos de Dados, se caracterizam pela facilidade de uso e por não exigirem do usuário informações sobre como os dados devem ser recuperados.

Essas linguagens apresentam tambem alguma capacidade dedutiva, através do mecanismo de definição de visões.

Apesar do seu vasto escopo de aplicação, existem exemplos de consultas que não podem ser expressos através dessas linguagens. Um exemplo clássico dessa limitação é o de consultas que envolvem o fecho transitivo de uma relação binária:

-a partir da relação filiação (Filho, Pai),. quem são todos os ancestrais de João?

-se a base de dados de uma empresa registra quem é o sucessor de determinado projeto, ou seja, aquele que pode ser começado a partir do término do referido projeto, quais são todos os projetos afetados pelo adiamento do projeto P1?

Para expressar essas consultas seria necessário dispor de uma linguagem de consultas recursiva, que permitisse a composição recursiva de uma relação com ela própria.

Independente da sintaxe escolhida para acolher a recursividade, existe o problema do processamento dessas consultas, o que é um dos pontos fundamentais para a construção de sistemas de base de conhecimentos.

Existem diversas propostas de solução para o processamento de consultas recursivas, a mais conhecida delas é a solução PROLOG. Um limitante de quase todas as propostas é o domínio de sua aplicação: que categoria de consultas é garantidamente respondida por um dado método.

Este trabalho pretende expor alguns dos métodos propostos e discutir questões relativas à sua implementação.

2. Classificação das estratégias.

O problema que se pretende resolver consiste de uma consulta expressa através de um predicado. O predicado pode dizer respeito diretamente a uma relação da base de dados, caso em que a consulta é solucionada apenas através de recuperação. Por outro lado, o predicado pode ser derivado, ou seja, é definido por uma ou mais regras lógicas.

Dá-se a denominação de base de conhecimentos ao conjunto de todas as relações da base de dados, dita base de dados explícita, acrescido do conjunto de regras que definem os predicados derivados, dita base de dados implícita.

As diversas estratégias propostas diferem entre si por vários aspectos.

A primeira subdivisão que pode ser estabelecida diz respeito ao tratamento global aplicado à consulta e à base de conhecimento. Duas categorias se distinguem [CERI87]:

-estratégias lógicas, que manipulam a consulta e as regras no domínio da lógica;

-estratégias algébricas, que convertem as regras e a consulta para a álgebra e aplicam manipulações algébricas ao conjunto de equações resultantes.

As estratégias lógicas podem ser distintas entre si pelo fato de exigirem ou não um passo prévio de reescrita das regras [BANB6a]:

-estratégias de avaliação, em que não ocorre nenhuma reordenação prévia das regras, sendo a consulta aplicada diretamente à base de conhecimento;

-estratégias de reescrita de termos, em que as regras são rearrumadas de modo a que a sua avaliação seja mais eficiente, e em seguida é aplicado um método de avaliação.

Do ponto de vista da interveniência das relações de base ou derivadas, as estratégias se dividem em:

-compiladas, que consistem de um passo preliminar de "compilação", em que são consideradas apenas as relações derivadas (base da dados implícita), que gera um "programa objeto", posteriormente executado sobre as relações base (base de dados explícita);

-interpretadas, em que tanto fatos como regras são considerados intermitentemente durante o processo de "execução" da consulta.

As estratégias de reescrita de termos são compiladas, pois o passo de reescrita das regras considera apenas a base da dados implícita.

Quanto à execução, as estratégias se subdividem em iterativas ou recursivas, conforme o controle do interpretador ou programa objeto.

Uma última diferenciação diz respeito ao sentido da pesquisa:

-top-down, se a estratégia parte da consulta e a expande através da aplicação de regras às relações derivadas;

-bottom-up, se a estratégia parte das relações base e gera tuplas para as relações derivadas, até gerar os resultados da consulta.

É esperado que as estratégias "top-down" sejam mais eficientes que as "bottom-up", pois estas últimas geram mais resultados do

que o necessário por não levarem em conta a consulta durante o processo. Porem, as estratégias "bottom-up" são de mais fácil compreensão e implementação [BAN86a].

Um ponto comum em quase todas as estratégias conhecidas hoje é a preponderância das regras sobre os fatos durante o processamento de consultas. Em geral, não há grande preocupação em direcionar o processamento no sentido de otimizar a recuperação de fatos na base de dados explícita. Apenas o projeto PROBE [DASM86] tenta conciliar a busca da solução indutiva com a eficiência da recuperação de fatos.

A seguir serão discutidos alguns métodos de processamento de consultas recursivas. O exemplo utilizado consiste da seguinte base de dados, CAT, em que é usada a notação PROLOG:

Base de Dados Explícita: composta pelas relações

```
disc  (CodDisc, NumCred)
prereq(CodDisc, CodPrereq)
```

que armazenam cada disciplina, seu número de créditos, e as disciplinas que são pre-requisitos de alguma disciplina. Os fatos presentes na base de dados explícita são

```
disc(calc1, 4).
disc(calc2, 4).
disc(calc3, 4).
disc(calc4, 4).
disc(fis1, 5).
disc(fis2, 5).
disc(fis3, 5).
disc(fis4, 5).
disc(fisnuclear, 6).
prereq(calc2, calc1).
prereq(calc3, calc2).
prereq(calc4, calc3).
prereq(fis2, fis1).
prereq(fis2, calc1).
prereq(fis3, fis2).
prereq(fis4, fis3).
prereq(fis4, calc3).
prereq(fisnuclear, fis4).
prereq(fisnuclear, calc4).
```

Base de Dados Implícita: composta pelas regras

```
r1      depende(X,Y) :- depende(X,Z), prereq(Z,Y).
r2      depende(X,Y) :- prereq(X,Y).
```

3. Método de Avaliação Ingênua.

Esse é o método mais simples e imediato já proposto. Consiste de uma estratégia lógica de avaliação, que utiliza a abordagem compilada, iterativa e bottom-up. É aplicável a um conjunto de regras "bottom-up" avaliáveis.

Definição:

Um conjunto de regras é "bottom-up" avaliável se cada uma das regras que o compõe é "bottom-up" avaliável, ou seja,

-cada variável da cabeça da regra aparece também no corpo, e
-cada variável do corpo é segura.

A não "segurança" de uma variável provem de sua ocorrência em predicados aritméticos, ditos avaliáveis, aos quais estão associadas relações com um número infinito de tuplas. Por exemplo, a regra

```
muitoscreditos(X) :- X >= 5.
```

tem no seu corpo um predicado avaliável, ao qual corresponde um relação com infinitos elementos.

Definição:

Uma variável de um predicado do corpo de uma regra é segura se

-aparece em algum predicado não avaliável do corpo, ou
-aparece apenas em um predicado avaliável p, mas p tem um subconjunto de outras variáveis, que são seguras, e que determinam o conjunto finito dos valores da primeira; ou seja, dados os valores para cada uma das variáveis do subconjunto determinante, então existe um conjunto finito de valores da variável do predicado avaliável que corresponde àqueles.

A regra

```
pesada(X) :- disc(X,Y), Y >= 5.
```

só tem no seu corpo variáveis seguras, apesar da ocorrência de um predicado avaliável.

O conjunto de regras da base de dados implícita de CAT é "bottom-up" avaliável, pois todas as variáveis das cabeças de regras aparecem tambem nos respectivos corpos e são seguras.

A consulta que será submetida ao método é a seguinte,

```
r3      consulta(X) :- depende(X,calc2).
```

ou seja, deseja-se saber todas as disciplinas que dependem da disciplina calc2.

A fase de compilação do método da avaliação ingênua gera um programa objeto iterativo, a partir das regras da base de dados implícita, da seguinte maneira:

-seleciona todas as regras que derivam a consulta;
-atribui a cada predicado derivado uma relação temporária;
-inicializa com vazio essas relações temporárias;
-a cada regra não recursiva, é associado um comando que computa o valor do predicado da cabeça a partir dos valores dos predicados do corpo;
-a cada conjunto de regras mutuamente recursivas, é associado um laço que aplica as regras desse conjunto até que nenhuma nova tupla seja gerada.

No caso desse exemplo, as regras que derivam a consulta são r1,r2,r3, e serão geradas duas relações intermediárias, correspondentes aos predicados **depende** e **consulta**.

O programa objeto gerado pela fase 1 do método é o seguinte:

```
inicio
-inicializar a relação depende com vazio;
-inicializar a relação consulta com vazio;
-avaliar (depende(X,Y) :- prereq(X,Y));
-inserir tuplas resultantes na relação depende
enquanto novas tuplas são geradas
     -avaliar (depende(X,Y) :- depende(X,Z),  prereq(Z,Y)) usando
      o valor corrente de depende;
     -inserir o resultado em depende
fim enquanto
-avaliar (consulta(X) :- depende(X,calc2));
-inserir o resultado em consulta
fim
```

A fase 2 do método executa o programa objeto contra a base de dados explícita, como se vê a seguir.

Execução antes do laço:

```
depende = {}
consulta = {}

depende = {(calc2,calc1), (calc3, calc2), (calc4,calc3),
           (fis2,fis1), (fis2,calc1), (fis3,fis2), (fis4,fis3),
           (fis4, calc4), (fisnuclear,fis4), (fisnuclear,calc4)}
```

Execução do laço: primeira vez

As seguintes novas tuplas são geradas em **depende**:

```
          {(calc3,calc1), (calc4,calc2), (fis3,calc1),
           (fis4,fis2), (fis4,calc2), (fisnuclear,fis3),
           (fisnuclear,calc3), (fisnuclear,calc3)}
```

O estado atual das relações consiste de **depende** contendo as tuplas originais e mais as novas e **consulta** vazia.

Execução do laço: segunda vez

Todas as tuplas geradas durante a vez anterior são novamente geradas e mais as seguintes novas tuplas em **depende**:

```
((calc4,calc1), (fis4,fis1), (fis4,calc1), (fis4,calc1),
 (fisnuclear,fis2), (fisnuclear,calc2))
```

A relação **consulta** ainda está vazia.

Execução do laço: terceira vez

Todas as tuplas geradas durante as duas execuções anteriores do laço são regeradas e mais as seguintes novas tuplas em **depende**:

```
((fisnuclear,fis1), (fisnuclear,calc1),
 (fisnuclear,calc1))
```

A relação **consulta** permanece vazia.

Execução do laço: quarta vez

Dessa vez nenhuma tupla nova é gerada. O estado das relações intermediárias é

```
depende = ((calc2,calc1), (calc3, calc2), (calc4,calc3),
           (fis2,fis1), (fis2,calc1), (fis3,fis2), (fis4,fis3),
           (fis4, calc4), (fisnuclear,fis4),
           (calc3,calc1), (calc4,calc2), (fis3,calc1),
           (fis4,fis2), (fis4,calc2), (fisnuclear,fis3),
           (fisnuclear,calc3),
           (calc4,calc1), (fis4,fis1), (fis4,calc1),
           (fisnuclear,fis2), (fisnuclear,calc2),
           (fisnuclear,fis1), (fisnuclear,calc1))

consulta = ()
```

Execução após o laço:

O estado de **depende** permanece o mesmo e as seguintes tuplas são geradas para a relação **consulta**

```
consulta = ((calc3,calc2), (calc4,calc2), (fis4,calc2),
            (fisnuclear,calc2))
```

que contem as respostas desejadas, e a execução do programa objeto é encerrada.

O que se observa imediatamente nesse método é que ocorre, a cada passo do laço, a regeração de todas as tuplas do passo anterior, além da eventual geração de novas. Além disso, todas as possíveis

tuplas de depende foram geradas, mesmo aquelas que não atendem à consulta.

O fato de que todas as tuplas da relação intermediária foram geradas é uma decorrência da estratégia "bottom-up", que não parte da consulta, e sim das relações base.

Entretanto, há uma melhoria a ser efetuada, no tocante a evitar a regeração de todas as tuplas a cada passo do laço. Essa é a proposta do próximo método.

4. Método de Avaliação Semi-ingênua.

Esse método é uma variação do anterior e, portanto, é incluído nas mesmas categorias e atende à mesma classe de aplicações.

O método introduz o conceito de **diferencial** de uma fórmula de primeira ordem, que computaria apenas o conjunto de novas tuplas a cada passo de um laço do programa objeto.

Os laços do programa objeto são gerados para cada conjunto de regras mutuamente recursivas, e todos os predicados cabeça envolvidos nessas regras são avaliados dentro do mesmo laço. Seja p um predicado recursivo, e p1, p2, ..., pn os predicados mutuamente recursivos a p, e q1, q2, ..., qm, relações base ou predicados derivados. Seja

```
p :- f(p1,p2,...,pn,q1,q2,...,qm).
```

a regra que computa o predicado p. No método de avaliação ingênua, todos os qi's são completamente avaliados antes do laço que computa os predicados p e os pi's. Tanto p como os pi's são avaliados no interior do mesmo laço, pois são mutuamente recursivos.

No exemplo anterior, a regra recursiva é

```
depende(X,Y) :- depende(X,Z), prereq(Z,Y).
```

Antes do laço, todas as tuplas de prereq, que é a relação base da regra, são inseridas na relação intermediária **depende**. O valor da relação **depende** antes do início do laço é

```
depende = {(calc2,calc1), (calc3, calc2), (calc4,calc3),
           (fis2,fis1), (fis2,calc1), (fis3,fis2), (fis4,fis3),
           (fis4, calc4), (fisnuclear,fis4), (fisnuclear,calc4)}
```

Na iteração j do laço, o valor corrente de cada relação intermediária correspondente a cada um dos predicados é pi(j), e é computado

```
f(p1(j), p2(j), ..., pn(j), q1, q2, ..., qm)
```

que causa a inclusão de novas tuplas em cada relação pi, além de replicar todas as tuplas já existentes. Seja **dpi(j)** o conjunto de

novas tuplas inseridas em cada pi. Então, o valor das relações pi no início do passo (j+1) é

pi(j) + dpi(j) (+ denota a união)

No caso do exemplo anterior, o valor de dp(2), que é o conjunto de novas tuplas inseridas pelo passo 2 do laço é

((calc4,calc1), (fis4,fis1), (fis4,calc1), (fis4,calc1), (fisnuclear,fis2), (fisnuclear,calc2))

No passo (j+1) do laço é avaliado

f((p1(j)+dp1(j)), ..., (pn(j)+dpn(j)), q1, q2, ..., qm)

que, desnecessariamente, recomputa a expressão computada no passo anterior!

O ideal seria computar a expressão apenas sobre as **novas** tuplas, ou seja,

df(p,dp,q) = f(p+dp,q) - f(p,q)

Trata-se, então, de determinar a fórmula para **df**, o que é um problema, já que as expressões de primeira ordem não tem o operador diferença.

Caso se consiga uma fórmula para **df**, o programa objeto pode ser gerado, associando-se a cada predicado recursivo **p** quatro relações intermediárias, p.antes, p.depois, dp.antes, dp.depois. Os laços correspondentes aos conjuntos de regras mutuamente recursivas seriam gerados como se segue.

```
enquanto o estado muda

     para todos os predicados mutuamente recursivos p
          -iniciar dp.depois com vazio;
          -iniciar p.depois com p.antes;
     fim para

     para cada regra mutuamente recursiva
          -avaliar df(p1,dp1,p2,dp2,...,pn,dpn,q1,q2,...,qm)
            usando os valores correntes de pi.antes para pi e
                                          de dpi.antes para dpi;
          -inserir as tuplas resultantes em dp.depois;
          -inserir as tuplas resultantes em p.depois;
     fim para

     p.antes := p.antes + p.depois;
     dp.antes := dp.depois;

fim enquanto
```

O problema de encontrar uma fórmula para **df** não está completamente resolvido. Em [BAN86] há propostas para **df** em termos de álgebra relacional.

É importante observar que a avaliação de **df** nem sempre é equivalente a computar f apenas sobre as novas tuplas. Isso só é válido no caso de regras **lineares**, ou seja, aquelas que só tem em seu corpo um e apenas um predicado mutuamente recursivo com o predicado da cabeça.

No exemplo aqui usado, só há regras lineares. Então a aplicação da avaliação semi-ingênua equivaleria a aplicar a avaliação ingênua apenas às novas tuplas, a cada passo do laço. Entretanto, se houvesse na base de dados a regra

```
depende(X,Y) :- depende(X,Z), depende(Z,Y).
```

e a cada passo só se aplicasse a avaliação às novas tuplas, a resposta à consulta só conteria as disciplinas distanciadas de dois entre si.

Os dois métodos descritos até agora apresentam uma ineficiência considerável, devida ao fato de que ambos calculam a relação que atende à consulta por inteiro. Já que a consulta normalmente envolve alguma constante, seria desejável restringir o espaço de pesquisa na base de dados explícita, de modo a considerar apenas os fatos **relevantes**, ou seja, aqueles que derivam a consulta. No entanto, a determinação dos fatos relevantes pode envolver um esforço tão grande quanto o necessário para atender à consulta.

Alguns métodos propõem uma estratégia para a avaliação do conjunto de fatos relevantes, a partir de modificações no conjunto original de regras, e portanto, são classificados como estratégias de reescrita de termos.

5. O Método dos Conjuntos Mágicos.

Essa é uma estratégia lógica de reescrita de termos, classificada ainda como compilada, iterativa e bottom-up. É aplicável a um conjunto de regras "bottom-up" avaliáveis.

O objetivo da fase de reescrita da base de dados implícita consiste em acrescentar regras capazes de pré-calcular o conjunto de fatos relevantes para determinada consulta.

O primeiro passo consiste em gerar, a partir do conjunto original de regras, um **sistema de regras adornadas** [ULLM85], de modo que cada regra seja transformada em um conjunto equivalente de regras em que, em cada nova regra, algumas variáveis do predicado cabeça sejam consideradas **amarradas** e as demais livres.

Um predicado **adornado** será denotado por um sufixo constituido por uma sequência de b's e f's, que indica quais os argumentos amarrados ("bound") ou livres ("free"). O comprimento dessa sequência corresponde ao número de argumentos do predicado. Por

exemplo,

p.bff...f

é uma das possíveis versões adornadas do predicado **p**, em que o primeiro argumento está amarrado("bound") e todos os demais livres ("free"). O comprimento da cadeia bf...f corresponde ao número de argumentos do predicado **p**.

A partir do conceito de variável **amarrada**, pode-se definir o conceito de argumento **distinto** em um predicado do corpo de uma regra, que,

-ou está amarrado no predicado cabeça,
-ou é constante,
-ou aparece em uma ocorrência de algum predicado base do corpo da regra que tem uma variável distinta.

(note que essa definição é recursiva).

A construção do novo conjunto de regras se dá da seguinte forma:

-para cada regra da base de dados implícita, é criado um novo conjunto de regras, em que cada uma tem como cabeça uma das possíveis versões adornadas do predicado cabeça da regra original;

-em cada nova regra, substituir todos os predicados derivados do lado direito pelas suas versões adornadas, em que adorno é determinado pelas variáveis distintas do predicado.

Considerando o exemplo utilizado anteriormente, o novo conjunto de regras seria,

```
depende.bf(X,Y) :- depende.bf(X,Z), prereq(Z,Y).
depende.fb(X,Y) :- depende.fb(X,Z), prereq(Z,Y).
depende.bf(X,Y) :- prereq(X,Y).
depende.fb(X,Y) :- prereq(X,Y).
consulta.f(X) :- depende.fb(X,calc2).
```

Na primeira regra, o adorno do predicado depende.bf(X,Z) resulta diretamente do adorno da cabeça da regra. Na segunda regra, o adorno do predicado depende.fb(X,Z) estabelece que a variável Z é amarrada, o que decorre do fato de que Z é distinta por aparecer no predicado base prereq(Z,Y), em que Y é distinta.

Desse novo conjunto de regras, apenas um subconjunto deriva a consulta e, portanto, será o subconjunto considerado doravante, ou seja,

```
depende.fb(X,Y) :- depende.fb(X,Z), prereq(Z,Y).
depende.fb(X,Y) :- prereq(X,Y).
consulta.f(X) :- depende.fb(X,calc2).
```

A partir desse conjunto de regras, um novo conjunto será gerado da seguinte forma,

-para cada ocorrência de um predicado derivado no corpo de uma regra adornada, é gerada uma regra mágica, através do seguinte algoritmo:

(i) escolher um predicado adornado p no corpo de uma regra adornada r;

(ii) apagar todos os outros predicados derivados do corpo;

(iii)na ocorrência do predicado derivado escolhido, substituir o nome do predicado por magico_p.a, onde a é o adorno, e apagar todas as variáveis não distintas;

(iv) apagar todos os predicados base não distintos;

(v) na cabeça da regra, apagar todas as variáveis não distintas, e trocar o nome por magico_p1.a', onde p1 é o predicado da cabeça e a' é o adorno do predicado p1;

(vi) trocar de posição os dois predicados mágicos.

O seguinte conjunto de regras mágicas será gerado para o exemplo em questão:

```
magico.fb(Z) :- magico.fb(Y), prereq(Z,Y).
magico.fb(calc2).
```

Não foi necessário nesse exemplo distinguir dentre os predicados mágicos gerados, e por esse motivo foi simplificada a sua denominação.

-para cada regra adornada será gerada uma regra modificada da seguinte maneira:

para cada regra cuja cabeça é p.a, acrescentar ao seu corpo o predicado mágico magico_p.a(X), onde X é a lista de variáveis distintas nessa ocorrência de p.

O seguinte conjunto de regras modificadas será gerado,

```
depende.fb(X,Y) :- magico.fb(Y), depende.fb(X,Z), prereq(Z,Y).
depende.fb(X,Y) :- magico.fb(Y), prereq(X,Y).
consulta(X) :- depende.fb(X,calc2).
```

Esses dois novos conjuntos de regras, o de regras mágicas e o de regras modificadas, serão avaliados através de um dos métodos de avaliação já propostos. O programa objeto gerado agora conterá um laço a mais que os já discutidos, pois há agora uma nova regra recursiva, que é

```
magico.fb(Z) :- magico.fb(Y), prereq(Z,Y).
```

O laço do programa objeto que calcula as tuplas da relação intermediária **magico** nada mais faz do que selecionar os fatos relevantes para a consulta corrente.

Supondo que o método de avaliação usado é o método ingênuo, o seguinte programa objeto será gerado, durante a fase de compilação (elimina-se o sufixo **fb** por questão de simplicidade).

```
início
-inicializar a relação magico com {(calc2)};
enquanto novas tuplas são geradas
     -avaliar (magico(Z) :- magico(Y), prereq(Z,Y)) usando
      o valor corrente de magico;
     -inserir o resultado em magico
fim enquanto

-inicializar a relação depende com vazio;
-avaliar (depende(X,Y) :- magico(Y), prereq(X,Y));
-inserir tuplas resultantes na relação depende
enquanto novas tuplas são geradas
     -avaliar
         (depende(X,Y) :- magico(Y), depende(X,Z), prereq(Z,Y))
      usando o valor corrente de depende;
     -inserir o resultado em depende
fim enquanto

-inicializar a relação consulta com vazio;
-avaliar (consulta(X) :- depende(X,calc2));
-inserir o resultado em consulta
fim
```

Como o conjunto de regras é linear, pode-se aplicar a avaliação de regra dentro de um laço apenas às novas tuplas, para evitar duplicação de resultados.

A execução do programa produzirá os seguintes resultados:

Antes do laço 1: magico = {(calc2)}

Laço 1 - primeira vez: magico = magico U {(calc3)}

Laço 1 - segunda vez: magico = magico U {(calc4), (fis4)}

Laço 1 - terceira vez: magico = magico U {(fisnuclear)}

Laço 1 - quarta vez: nenhuma nova tupla é produzida, e a relação mágico armazena os valores

magico = {(calc2), (calc3), (calc4), (fis4), (fisnuclear)}

Antes do laço 2:

depende = {(calc3,calc2), (calc4,calc3), (fis4,calc3), (fisnuclear,calc4), (fisnuclear,fis4)}

Laço 2 - primeira vez:

depende = depende U {(calc4,calc2), (fis4,calc2),
(fisnuclear,calc3)}

Laço 2 - segunda vez:

depende = depende U {(fisnuclear, calc2)}

Laço 2 - terceira vez: nenhuma nova tupla é produzida, e a relação depende armazena os valores

depende = {(calc3,calc2), (calc4,calc3), (fis4,calc3),
(fisnuclear,calc4), (fisnuclear,fis4),
(calc4,calc2), (fis4,calc2),
(fisnuclear,calc3),
(fisnuclear, calc2)}

Execução após o laço:

O estado de depende permanece o mesmo e as seguintes tuplas são geradas para a relação consulta

consulta = {(calc3,calc2), (calc4,calc2), (fis4,calc2),
(fisnuclear,calc2)}

Comparando-se o volume de resultados intermediários gerados dessa vez com os dos métodos anteriores, nota-se uma redução considerável, devida ao fato de que apenas os fatos relevantes para a consulta foram capturados na relação magico.

6. Conclusão.

Os tres métodos aqui apresentados são uma pequena amostra do trabalho que vem sendo desenvolvido para atender ao processamento de consultas recursivas.

Um problema ainda em aberto é a questão da não aplicabilidade de um método a qualquer conjunto de regras.

Outra questão que se coloca consiste da pouca atenção dada à recuperação eficiente de fatos na base de dados explícita. A pesquisa é sempre subordinada à sequência de execução de cada método, sem que haja uma preocupação global de minimizar os acessos em função de determinada consulta. Um exemplo de como uma estratégia global poderia melhorar essa pesquisa fica evidente no exemplo explorado nesse trabalho. Já que o conjunto de regras envolve o fecho transitivo, a relação prereq poderia ser previamente ordenada pelo atributo que é constante na consulta. Dessa forma, todas as pesquisas a essa relação seriam otimizadas.

É compreensível que as pesquisas iniciais tenham se detido principalmente no aspecto de viabilizar a solução de determinadas consultas, e que em um passo posterior haja a preocupação em tornar os métodos mais eficientes.

BIBLIOGRAFIA

[BAN86] Bancilhon F, "Naive Evaluation of Recursively Defined Relations", em "On Knowledge Base Management Systems", ed. Brodie M e Mylopoulos J, Springer-Verlag, 1986.

[BAN86a] Bancilhon F, "An Amateur's Introduction to Recursive Query Processing", ACM, 1986.

[CERI87] Ceri S, Notas de aula do curso "Database Design", RJ, 1987.

[DASM86] Dayal U, Smith J M, "PROBE: A Knowledge-Oriented Database Management System", em "On Knowledge Base Management Systems", ed. Brodie M e Mylopoulos J, Springer-Verlag, 1986.

[ULLM85] Ullman J, "Implementation of Logical Query Languages for Databases", ACM Tods, Vol. 10, n. 3, set 1985.