IMPLEMENTANDO UMA MAQUINA DE INFERENCIAS ACOPLADA A BANCOS DE DADOS EXTENSIONAIS

Pedro Manoel Silveira

NCE 0587

Abril, 1987

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro, RJ
BRASIL

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

Implementando uma Máquina de Inferências Acoplada a Bancos de Dados Extensionais

Pedro M. Silveira

SUMÁRIO

Este artigo descreve uma implementação experimental onde uma máquina de inferências, acoplada a um banco de dados extensional, foi utilizada na verificação de esquemas e reformulação de consultas. São apresentadas as motivações de tal experiência, a fundamentação teórica e uma descrição lógica da implementação.

ABSTRACT

This article describes an experimental implementation where an inference machine, coupled to an extensional database, was used for schema verification and query reformulation. We present the motivation behind such experience, its theoretical fundaments and logical description.

1. Introducão

O desenvolvimento de metodologias para prova automática de teoremas e manipulacão lógica [1,2] teve forte influência na área de bancos de dados. De um lado, sua própria concepcão teórica foi posta em perspectiva, uma vez que bancos de dados poderiam ser vistos como teorias, em contraposicão ao que acontece na abordagem tradicional onde os dados são um modelo de uma teoria. Por outro lado, várias das técnicas de prova de teoremas puderam ser incorporadas aos SGBD atuais, gerando novas possibilidades de pesquisa.

O objetivo deste artigo é descrever uma implementacão experimental onde uma máquina de inferências foi utilizada em conjunto com um banco de dados extensional para a reformulacão de consultas e verificacão de esquemas.

O presente artigo está estruturado da seguinte maneira. Na Secão 2 nós apresentamos a motivacão para tal aplicacão, mostrando com exemplos uma idéia do que se pode fazer. Na Secão 3 nós apresentamos a linguagem modelo para formulacão de esquemas e consultas e as Secões 4 e 5 cuidam de aspectos mais formais da aplicacão, mostrando algo de sua fundamentacão teórica. A Secão 6 descreve a implementacão em maiores detalhes. Na Secão 7 são discutidas algumas das conclusões de tal experiência e a Secão 8 conclui o artigo.

```
MUSICO    ( Nome/NOME, Ano_Nasc/ANO )
OPUS      ( Compositor/NOME, Titulo/NOME, Ano/ANO )
GRAVACAO  ( Maestro/NOME, Orquestra/NOME, Ano/ANO, Opus/TITULO )
```

Figura 1. Esquema Relacional

2. Caracterizacão do Problema

Nos exemplos abaixo, nós usaremos o sistema INGRES versão 7.10 [3] como referência. Considere o esquema relacional da Figura 1, onde para cada relacão, A/P indica que o atributo A tem P por domínio. INGRES permite limitadas restricões sobre o banco de dados e nós poderíamos ter

```
RANGE OF m IS MUSICO
DEFINE INTEGRITY ON m IS m.Ano_Nasc > 1600
```

afirmando que para todas as tuplas da relacão **Musico**, o valor do atributo **Ano_Nasc** deve ser maior do que 1600. Suponha a consulta

```
RETRIEVE (m.Nome) WHERE m.Ano_Nasc > 1938
```

e a consulta

```
RETRIEVE (m.Nome) WHERE m.Ano_Nasc < 1600
```

onde o domínio da variável **m** são as tuplas da relação **Musico**. Embora a segunda consulta claramente produza uma resposta vazia, ambas levam aproximadamente o mesmo tempo para serem processadas, permitindo-nos concluir que o connecimento prévio (a partir das restricões) sobre as tuplas de **Musico** não foi levado em conta pelo gerenciador do banco de dados. No exemplo, a resposta (i.e., vazia) poderia ser inferida a partir das restricões de integridade. Embora não seja o caso frequentemente, há casos em que o processamento de uma consulta pode ser melhorado sensivelmente. Suponha para o mesmo esquema a restricão "somente músicos nascidos após 1850 podem ter conduzido numa gravacão". Embora não permitida, tal restricão em INGRES seria da forma

```
DEFINE INTEGRITY ON m, g IS
g.Maestro = m.Nome -> m.Ano_Nasc > 1850
```

onde as variáveis **m** e **g** tem as relacões **Musico** e **Gravacao** como domínios. Agora, considere a consulta

```
RETRIEVE (m.Nome) WHERE m.Nome = g.Maestro
```

Sem consultar as restricões, o processamento desta consulta envolveria todas as tuplas de **Musico**, o que poderia ser custoso, uma vez que alguma forma de juncão com a relacão **Gravacao** é necessária. Analisando as restricões do banco de dados, entretanto, nós notamos que a condicão seletiva

```
m.Ano_Nasc> 1850
```

poderia ser incluída, pois sabemos que apenas aqueles músicos nascidos após 1850 poderiam ser maestros. Para um banco de dados onde as restricões mencionadas fossem respeitadas, a consulta acima deveria produzir sempre o mesmo resultado que

```
RETRIEVE (m.Nome) WHERE m.Nome = g.Maestro
                        AND
                        m.Ano_Nasc > 1850
```

com a selecão adicional provavelmente causando uma melhoria sensível na performance da execucão.

Uma outra frente diz respeito à existência de um modelo para o esquema ou, em outras palavras, a possibilidade de satisfazer-se as fórmulas restritivas. No esquema hipotético que temos até aqui, uma restricão adicional como

```
DEFINE INTEGRITY ON m IS
 not (m.Ano_Nasc > 1600)
```

seria silenciosamente aceita pelo INGRES, embora condenando a relacão **Musico** à eterna vazitude, uma vez que tupla nenhuma poderia satisfazer esta e a primeira restricões simultaneamente.

Embora tenhamos tomado INGRES como exemplo, a maioria absoluta dos sistemas gerenciadores de bancos de dados não dão suporte

a ítens como os exemplificados acima. Entretanto, mesmo que um sistema gerenciador não seja capaz de fazer valer as leis de integridade, a especificação das mesmas deveria existir, uma vez que 1) esquemas mais precisos seriam definidos, e 2) o módulo de processamento de consultas poderia tirar proveito das restrições como informações adicionais sobre os estados do BD.

3. Ambiente Formal

A aplicação de uma máquina de inferências requer um ambiente formal para expressão das fórmulas lógicas. No nosso caso, onde bancos de dados apresentam várias disciplinas de modelagem de dados e linguagens de consulta, foi necessário uniformizar tais diferenças através de uma linguagem comum para expressão de esquemas e consultas. Tal linguagem foi batizada **L+**.

L+ é baseada no Cálculo de Predicados e não é possível neste espaço dar uma descrição geral da sintaxe e semântica da linguagem. Nas figuras 2 e 3 há exemplos de esquemas em **L+** e o leitor pode ter uma ligeira idéia de sua estrutura e estilo. No primeiro esquema, **NOME**, **TITULO** e **ANO** são conjuntos primitivos, cujas extensões são fixas e definidas. O primeiro é o conjunto de strings alfabéticas com até 40 caracteres e o terceiro o conjunto de números de 1500 até 2000. A seguir, **MUSICO**, **OPUS** e **GRAVACAO** são definidos como abstrações, cada um com um conjunto de funções definidas sobre eles. Note que a função **Compositor** definida sobre **OPUS** retorna um objeto em **MUSICO**, que é uma abstração, enquanto que a função **Ano** retorna objetos do conjunto primitivo **ANO**. Nos exemplos há também uma série de restrições de integridade, as quais são fórmulas lógicas sem variáveis livres.

L+ compreende uma variada gama de declarações cuja descrição e sintaxe detalhadas aparecem em [5], inclusive funções de agregação. Quanto à consultas, **L+** tem como forma geral

$$t_1,\ldots,t_n \quad \text{WHERE} \quad v_1 \in D_1 \ \ldots \ v_k \in D_k$$
$$\text{WHEN} \quad \phi\,(v_1,\ldots,v_k)$$

onde $t_1,\ldots,t_n$ é a lista-alvo, $D_1 \ \ldots \ D_k$ são os domínios das variáveis v_1 até v_k e ϕ é uma fórmula onde as únicas variáveis livres são $v_1,\ldots,v_k$. Quantificadores universais e existenciais podem aparecer em ϕ.

O uso de linguagens de definição de dados baseadas no Cálculo de Predicados não é novidade e pode ser encontrado em trabalhos como [6] e [7]. Também, pode-se argumentar que tal escolha pode conduzir a implementações não factíveis. Entretanto, no contexto deste trabalho, onde um ambiente formal claro e preciso é necessário, **L+** parece-nos adequada, uma vez que 1) **L+** provê maneiras simples para obtenção de facilidades de modelagem presentes em outros modelos e 2) **L+** vai além daquelas facilidades na sua capacidade de prover um ambiente formal para a aplicação de uma máquina de inferências. Em [5] nós apresentamos uma descrição detalhada de **L+**, além da comparação com outros modelos e linguagens, entre eles DAPLEX [8].

4. Reformulação de Consultas

Seja Q uma consulta da forma

$$t_1,\ldots,t_n \quad v_1 \in D_1 \ldots v_k \in D_k \quad \phi(v_1,\ldots,v_k)$$

Tomando Q como uma relação hipotética, contendo apenas todas as tuplas formando a resposta à consulta, nós podemos dizer que

$$\forall v_1 \in D_1 \ldots \forall v_k \in D_k \quad \phi(v_1,\ldots,v_k) \rightarrow Q(t_1,\ldots,t_n) \qquad (1)$$

e, complementarmente

$$\forall v_1 \in D_1 \ldots \forall v_k \in D_k \quad Q(t_1,\ldots,t_n) \rightarrow \phi(v_1,\ldots,v_k) \qquad (2)$$

Expressando (1) e (2) na forma clausal [4], nós temos um conjunto S_1 de cláusulas da forma

$$Q(t_1,\ldots,t_n) \quad L_1$$

derivadas de (1), e um outro conjunto S_2 de cláusulas da forma

$$Q(t_1,\ldots,t_n) \quad L_2$$

derivadas de (2), onde L_i é uma disjunção (possivelmente vazia) de literais, t_i é um termo e todas as variáveis em cada cláusula são universalmente quantificadas em seus respetivos domínios. Em L_i, não há literais em Q. As cláusulas das formas acima são chamadas respostas definitivas de Q e complementos definitivos de Q, respectivamente.

Supondo que o conjunto de restrições presentes no esquema do banco de dados gere o conjunto S_B de cláusulas, então a aplicação da máquina de inferências ao conjunto $S_B \cup S_1$ resulta num conjunto final onde se pode distinguir o subconjunto S_A, formado pelas respostas definitivas de Q. Analogamente, a aplicação da máquina de inferências a $S_B \cup S_2$ resulta num conjunto de cláusulas onde nós distinguimos S_C, como o conjunto de complementos definitivos de Q.

4.1 Interpretação de S_A

S_A inclui cláusulas oriundas originalmente da consulta Q e cláusulas inferidas pela máquina de inferências. A suposição de que cláusulas resumidas estão presentes em S_A garante que não haja redundâncias. Com isso, cláusulas inferidas que representem uma versão mais simplificada da consulta, com um menor número de literais, provavelmente resumirão claúsulas mais complexas. Adicionalmente, a resposta para Q está completa em S_A, ou seja, a união das respostas representadas por cada cláusula é a resposta total de Q.

Uma estratégia simples para implementar este processo é controlar o conjunto S_A (inicialmente S_A toma o valor de S_1) de modo a

UNICAMP

(i) incluir em S_A as respostas definitivas de **Q** geradas pelo processo de inferência que resumam alguma de suas cláusulas;
(ii) manter S_A livre de cláusulas resumidas;

Desse modo, quando o processo de inferência pára tem-se em S_A um conjunto que, por definição, contém a resposta completa de **Q**, porque
(i) as cláusulas inseridas em S_A foram corretamente inferidas das leis regendo o banco de dados, representando formulações genuínas para $Q(t_1,...t_n)$;
(ii) as cláusulas removidas foram resumidas por cláusulas presentes em S_A.

Como não há cláusulas redundantes, S_A representa o mínimo conjunto de cláusulas proporcionando uma resposta completa de **Q**.

4.2 Interpretação de S_B

Aqui nós trabalhamos com as consequências de (2), viz, sempre que $Q(t_1,...,t_n)$ é verdadeiro então também é verdadeira a fórmula **O**. A idéia é produzir uma variedade de cláusulas que sejam complementos definitivos de **Q** e então escolher aquelas que poderiam enriquecer a formulação da consulta.

Excluíndo-se as respostas vazias, cuja detecção é imediata, o processo de geração de cláusulas complementares é complicado pelo fato de que a utilidade de uma cláusula não é imediatamente compreendida. No caso de S_A, o interesse imediato é a simplificação e qualquer cláusulas resumindo alguma das existentes é interessante. Neste caso, entretanto, a relevância de uma cláusula depende de vários critérios. Para ilustrar esses critérios, nós apresentamos abaixo dois exemplos: a identificação de complementos definitivos de uma consulta **Q** representando seleções e junções.

Seleções

Aqui nós usamos o termo seleção, da Álgebra Relacional, para melhor entendermos a semântica da operação. Basicamente, nós lidamos com expressões de uma variável podendo-se, entretanto, ter mais de uma variável no literal em **Q**. Para uma consulta **Q**, um complemento definitivo da forma

$$Q(t) \mid L$$

representa uma seleção se:
(i) v é a única variável livre em L;
(ii) v ocorre em t;

Todos os termos aparecendo em **L** são ou constantes, ou a variável v ou símbolos funcionais cuja única variável livre é v.

Junções

Expressões com duas ou mais variáveis livres causam o

equivalente à operação de junção da Álgebra Relacional. A introdução de tal operação numa consulta pode ser interessante quando tem-se uma estrutura de índices nas relações objeto, facilitando o acesso, ou quando se trabalha com modelos como Codasyl por exemplo, onde junções coincidindo com as instâncias de sets são eficientemente processadas.

Para uma consulta **Q**, um complemento definitivo da forma

Q(t) L

representa uma junção se:

(i) v_1 e v_2 são as únicas variáveis livres em **L** e tem domínios distintos;

(ii) v_1 e v_2 ocorrem em **t**;

(iii) há um e somente um literal (ou sua negação) em **L**, da forma $t_1 \Theta t_2$, onde Θ é um símbolo relacional.

Embora bastante genérica, a definição acima restringe nossa escolha de junções. Requer-se que v_1 e v_2 apareçam na lista alvo, i. e., elas devem estar diretamente relacionadas ao resultado da consulta. Outra restrição é que apenas uma junção de cada vez seja considerada, uma vez que nós admitimos apenas duas variáveis livres de cada vez. A junção em si é representada pela expressão Θ e os vários tipos de junção (Θ-join, equi-join, etc...) dependem de como o literal é formado. t_1 e t_2 são termos cujas únicas variáveis livres são v_1 e v_2 respectivamente. Essas restrições podem ser relaxadas de várias maneiras, permitindo a composição de várias junções ou junções com seleções.

Os dois critérios apresentados acima apenas ilustram a larga variedade de soluções possíveis na escolha de complementos de uma consulta. Seleções, por exemplo, são casos clássicos de soluções gerais, pois normalmente é vantajoso adicionar seleções de modo a reduzir a cardinalidade dos operandos. Entretanto, em alguns modelos de dados poderão existir soluções particularmente vantajosas, como a introdução de junções para bancos de dados Codasyl, no caso da junção coincidir com o critério de formação de sets.

5. Verificação de Esquemas

Como foi visto nos exemplos acima, uma das possíveis aplicações de uma máquina de inferências acoplada a bancos de dados extensionais é a verificação lógica de um esquema. Formalmente, o problema é o da verificação da possibilidade de satisfazer as restrições de integridade do banco de dados. Neste caso, a máquina de inferências assume o papel de um provador de teoremas: dado um conjunto de cláusulas resultante das restrições de integridade do esquema, o objetivo é procurar gerar a cláusula vazia. Tal processo de inferência pode terminar por uma de três razões:

(i) a cláusula vazia é realmente gerada e neste caso podemos ter certeza de que o esquema é incorreto, pois seria impossível achar um estado não vazio do banco de dados que atendesse a todas as restrições existentes;

(ii) a cláusula vazia ainda não foi encontrada mas não há possibilidade de inferir novas cláusulas das até então geradas. Esta é uma situação incomum, a menos que se trate de esque-

mas com pouquíssimas restrições. De qualquer modo, o esquema deve estar correto no sentido em que há modelos possíveis que satisfaçam todas as restrições;

(iii) após uma quantidade substancial de inferências, a máquina de inferências excede seus recursos disponíveis e pára. Neste caso, assume-se que provavelmente o esquema esteja correto, embora não se possa afirmar conclusivamente sim ou não, uma vez que isto estaria além da capacidade de processamento disponível.

6. Descrição da Implementação Experimental

A Figura 4 mostra a interconexão dos componentes lógicos de nossa implementação. Os componentes aparecem como retângulos. A parte central é delimitada por um retângulo tracejado. Note que esta parte tem uma consulta em L+ como entrada e a consulta na forma clausal como saída. É ali que a maior parte da aplicação acontece. As duas caixas pontilhadas representam a interface para a linguagem QUEL. Na versão atual, as consultas em QUEL são traduzidas para L+ e então submetidas ao parser de L+. Na saída, a consulta em QUEL é construída a partir da forma clausal resultante do Analisador de Consultas. As próximas seções descrevem cada um dos componentes em maiores detalhes.

Compilador de Esquemas L+

É o responsável pela compilação de esquemas expressos em L+. Construído com auxílio do YACC (Yet Another Compiler-Compiler) [9], aceita a sintaxe completa de L+. A saída deste componente é um conjunto de cláusulas representando as restrições do esquema, bem como tabelas com informações a respeito de predicados, funções, expressões derivadas, etc...

Compilador de Consultas L+

Este componente verifica as consultas e compila-as para a forma clausal, conforme descrito acima. Expressões derivadas, permitidas em L+, são aceitas e substituídas por suas definições originais.

Máquina de Inferências

As funções básicas aqui são a simplificação das cláusulas e a procura de resolventes ao lado da aplicação do algoritmo de Paramodulation [10], e um tratamento similar para a relação > [5]. Um ponto interessante é a manipulação de tipos (ou domínios). Bancos de dados são fortemente baseados em domínios e, para cada unificação, a máquina de inferências testa a interseção dos domínios dos termos candidatos. Paralelamente, a compatibilidade de tipos entre predicados e termos é verificada. Num literal como

o domínio de t deve intersectar o domínio de X, ou o literal é sempre falso e pode ser removido de sua cláusula. Vários controles foram introduzidos na tentativa de restringir o crescimento do número de cláusulas, entre eles:
- o número de literais dos resolventes, de modo a não permitir o surgimento de cláusulas muito longas;
- o tamanho das cláusulas contendo igualdades, de modo a restringir a aplicação do algoritmo de Paramodulation;
- nível de unificação para termos envolvendo funções, de modo a evitar a unificação de parâmetros de parâmetros de parâmetros, etc...
- nível de unificação para termos envolvendo funções quando aplicando Paramodulation;
- suposição da não existência de domínios vazios.

Analisador de Consultas

Este módulo é responsável pela melhoria da consulta após a aplicação da máquina de inferências. O objetivo é estudar as cláusulas e estabelecer a melhor formulação de uma consulta. Para cada interface específica, diferentes estratégias podem ser usadas. De um modo geral, há quatro modos de melhorar uma consulta, descritos resumidamente a seguir.

Eliminação de redundâncias, onde o objetivo é eliminar cláusulas que contém literais em excesso, que não alteram o resultado final da consulta. Por exemplo, uma resposta definitiva como

$$Q(nome) \quad nome = f(v_1) \quad nome = f(v_2)$$

pode ser simplificada para

$$Q(nome) \quad nome \; != f(v)$$

sem alterar o resultado final de Q

Simplificacão, onde o objetivo é buscar nas cláusulas derivadas alguma formulação que simplifique o processamento da consulta. Neste caso, o maior benefício é a eliminação de junções, através da geração de cláusulas que resumam as cláusulas originais da consulta.

Complementação, onde o objetivo é buscar informações que produzam o enriquecimento da formulação de uma consulta. Em geral, a identificação de seleções complementares reduz a cardinalidade das relações em operações subsequentes.

Alternativas, onde a estratégia é buscar formulações alternativas que substituam as originais. Na presente implementação, entretanto, tal estratégia não foi efetivada.

Reconstrutor de Consultas L+

A função deste componente é reconstruir uma consulta em L+ a partir de sua representação na forma clausal. Esse processo envolve a remoção de funções Skolem [4].

Compilador de Consultas QUEL

Este é o primeiro componente operando para a interface QUEL especificamente. Sua função é traduzir consultas expressas em QUEL para consultas em L+.

QUEL é uma linguagem baseada no Cálculo de Tuplas [11], cujo comando RETRIEVE não é relacionalmente completo sem a ajuda das construções APPEND e DELETE. Assim sendo, o compilador de consultas QUEL considera como uma consulta qualquer número de comandos APPEND, RETRIEVE INTO e DELETE seguidos de um comando RETRIEVE. A idéia é substituir cada um daqueles comandos por expressões derivadas em L+ e fazer referências a estas no comando RETRIEVE.

Por exemplo, para a consulta

```
RANGE OF m IS MUSICIAN
RETRIEVE INTO TEMP (m.Nome)
RANGE OF t IS TEMP
RANGE OF g IS GRAVACAO
DELETE t WHERE t.Nome = g.Maestro
RETRIEVE (t.Nome)
```

relativa ao esquema da Figura 1, cujo resultado é a lista de músicos que não são maestros, nós teríamos

(i) em TEMP, os nomes de todos os músicos
(ii) com o comando DELETE, a remoção de TEMP dos músicos que são maestros
(iii) restando em TEMP, os músicos que não são maestros.

Para traduzir a consulta acima para L+, o software de conversão inicialmente define o efeito dos comandos RETRIEVE INTO e DELETE como conjuntos derivados e refere-se a eles na consulta final. Assim, nós teríamos

```
set TEMP is { Nome WHEN Exists Musico
                    SUCH THAT Nome = Nome(Musico) }
```

como resultado do RETRIEVE INTO e

```
set TEMP1  s { Temp WHEN not Exists Gravacao
                    SUCH THAT Temp = Maestro(Gravacao) }
```

como resultado do comando DELETE. Finalmente, correspondendo ao comando RETRIEVE nós teríamos a consulta em L+

```
Nome(Musico) WHEN For All Gravacao
                  Nome(Musico) != Maestro(Gravacao)
```

Detalhes de tal tradução podem ser encontrados em [5].

Reconstrutor de Consultas QUEL

Este componente traduz uma consulta na forma clausal para QUEL. Novamente, a não completude do comando RETRIEVE força-nos a uma combinação de comandos QUEL, quando é o caso da consulta

original conter quantificadores universais ou a união de atributos diferentes. Considerando a mesma consulta do exemplo acima, nós temos

```
Nome(Musico) WHEN For All Gravacao
                    Nome(Musico) != Maestro(Gravacao)
```

Isto é equivalente a

```
Nome(Musico) WHEN not Exists Gravacao
                    SUCH THAT Nome(musico) = Maestro(Gravacao)
```

ou seja, os nomes dos músicos para os quais não existe um maestro com o mesmo nome. Mas este é o conjunto de nomes de todos os músicos menos aqueles para os quais existe algum maestro com o mesmo nome.

```
( Nome(Musico) )
  -
( Nome(Musico) WHEN Exists Gravacao
                    SUCH THAT Nome(Musico) = Maestro(Gravacao) )
```

e, daqui, nós podemos produzir os comandos QUEL aparecendo no exemplo acima. Note que esta é uma visão simplificada do problema mas a idéia básica é transformar quantificadores universais em quantificadores existenciais com fórmulas negadas e então aplicar a operacão de diferença de conjuntos. Este processo é aplicado repetidamente, enquanto houver quantificadores universais na formulação da consulta.

Notas Adicionais

O uso de funcões diretamente computáveis torna-se relevante quando um modelo de dados almeja cobrir as facilidades de vários outros. Nesta implementacão nós adotamos uma orientacão de ordem prática para tal fim. Durante a compilacão de um esquema em **L+**, insere-se automaticamente a definicão das funcões mais utilizadas em geral. A idéia é de que a máquina de inferências possa reconhecer algumas dessas funcões e calculá-las sempre que possível. Se necessário, o usuário pode definir funcões adicionais mas, neste caso, a máquina de inferências as tratará como elementos sintáticos apenas, pois a metodologia de cálculo lhe é desconhecida. Em **L+**, as principais funcões diretamente computáveis são do tipo **LOG**, **SQRT**, **MOD**, etc...

Esse uso requer uma estrutura de tipos mais flexível. É irritante para um usuário definir cada funcão computável de acordo com os tipos que aparecem em esquemas específicos. Por esta razão, há dois tipos implícitos em **L+**: **STRING** e **NUMBER**, que representam cadeias de caracteres e números repectivamente. Assim, funcões computáveis podem ser definidas genericamente usando aqueles tipos. Para cada tipo definido no esquema, a compatibilidade deste com os tipos implícitos é estabelecida. Por exemplo, suponha a declaracão

```
ANO IS [1500 .. 2000 by 1]
```

Claramente, **ANO** é compatível com **NUMBER**. Para uma funcão defini-

da como

```
COMPUTABLE Add (n,n) -> n
           WHERE n IN NUMBER
```

o termo Add(5,Ano), onde ano tem domínio **ANO**, é válido.

Outros tipos básicos poderiam ser definidos como, por exemplo, a distinção entre números reais e inteiros. Por ocasião do processo de parsing, os operadores aritméticos +,-,/,* de **L+** são substituídos pelas funções diretamente computáveis **Add**, **Sub**, **Mult**, **Div** respectivamente.

7. Alguns Resultados Experimentais

É virtualmente impossível dar um quadro completo de como a máquina de inferências se porta de uma maneira geral. Mesmo um artigo dedicado integralmente ao assunto não seria suficiente. Entretanto, nós achamos importante passar ao leitor alguns resultados concretos a esta altura.

Para isso, nós tomaremos como exemplo a verificação do esquema mostrado na Figura 2 inicialmente e, depois, de uma versão mais sofisticada do mesmo esquema, mostrada na Figura 3. O leitor pode extrapolar esta análise para o caso de consultas, uma vez que o problema é análogo. A ação crítica é atingir um certo objetivo. Para a verificação de esquemas este objetivo é a descoberta da possibilidade ou não de satisfazer o esquema; para reformulação de consultas é a obtenção ou não de certas transformações. Para ambos os propósitos, o exemplo deve fornecer uma idéia do número de cláusulas envolvidas.

Para o esquema da Figura 2, as seguintes cláusulas existiriam:

```
 1.  nome(f1(gravacao)) = maestro(gravacao)
 2.  titulo(f2(gravacao)) = opus(gravacao)
 3.  nome(f3(opus)) = compositor(opus)
 4.  opus(gravacao) != titulo(opus) | ano(gravacao) > ano(opus)
       | ano(gravacao) = ano(opus)
 5.  maestro(gravacao) != nome(musico)
       | ano(gravacao) > ano_nasc(musico)
 6.  compositor(opus) != nome(musico)
       | ano(gravacao) > ano_nasc(musico)
 7.  titulo(opus) != opus(gravacao)
       | nome(musico) != compositor(opus)
       | ano(gravacao) > ano_nasc(musico)
 8.  titulo(opus) != opus(gravacao)
       | nome(musico) != compositor(opus)
       | ano(gravacao) > ano_nasc(musico)
 9.  ano_nasc(musico) > 1600
10.  nome(musico) != maestro(gravacao) | ano_nasc(musico) > 1850
```

onde f_1, f_2, f_3 são funções Skolem.

Note que este esquema não explora totalmente o poder de expressão de **L+**. Na verdade, ele se parece bastante com o modelo relacional. A máquina de inferências trabalha mal com este conjunto de cláusulas e não consegue descobrir se o esquema pode ser satisfeito ou não.

Suponha que o usuário cometa um engano e digite < em vez de > na restrição **Compositor-opus**. A máquina de inferências gera exatamente 300 cláusulas para descobrir que o esquema torna-se inválido. Este número de cláusulas ocorre pela alta incidência de igualdades e relações > nas cláusulas. Essa deficiência vem da semelhança com o modelo relacional presente no esquema, com sua característica inabilidade para lidar com expressões funcionais. Para relacionar duas tuplas, introduz-se igualdades.

Agora considere o exemplo da Figura 3. Aquele esquema é equivalente ao que nós consideramos acima exceto que, desta vez, o potencial de **L+** é melhor aproveitado. As restrições referentes a "foreign keys" não são mais necessárias e as conexões entre relações distintas ocorrem pelo uso de expressões funcionais. As cláusulas para aquele esquema estão listadas a seguir:

```
1.  ano(gravacao) > ano(opus(gravacao))
        | ano(gravacao) = ano(opus(gravacao))
2.  ano(gravacao) > ano_nasc(maestro(gravacao))
3.  ano(opus) > ano_nasc(compositor(opus))
4.  ano(opus(gravacao)) > ano_nasc(compositor(opus(gravacao)))
5.  ano_nasc(musico) > 1600
6.  ano_nasc(maestro(gravacao)) > 1850
```

Desta vez é possível notar-se uma menor incidência de igualdades. A notação funcional é bastante apropriada para este tipo de aplicação. Com o conjunto acima, a máquina de inferências gera apenas 52 cláusulas para descobrir que o esquema pode ser satisfeito. Para o engano de inverter-se a relação na restrição **Compositor-opus**, a máquina de inferências faz bonito: apenas 19 cláusulas são necessárias para detectar a inconsistência.

Os controles usados neste experimento não afetaram grandemente a performance global. A restrição de que um dos resolventes tenha no máximo dois literais não foi necessária, uma vez que as cláusulas não são longas. O nível de unificação para Paramodulation e para a relação > tiveram efeito entretanto, uma vez que em alguns casos a cadeia de substituições poderia ir além do limite de 3.

Algumas conclusões podem ser observadas dos parágrafos acima. Primeiramente, o número inicial de cláusulas, por si só, não leva necessariamente ao desastre. A natureza das cláusulas é também importante. Segundo, **L+** proporciona uma notação mais compacta para a expressão de restrições de integridade quando comparada, por exemplo, com a notação relacional. Terceiro, é bastante difícil prever ou caracterizar o comportamento da máquina de inferencias com segurança. Provas que parecem triviais podem explodir, enquanto que outras, aparentemente complicadas, fluem ràpida e suavemente.

8. Conclusões

Este artigo apresentou a descrição da aplicação de uma máquina de inferências acoplada a um banco de dados extensional. Há muitas conclusões úteis a ressaltar num trabalho dessa natureza e nos parágrafos seguintes nós analisamos e recapitulamos alguns deles.

O primeiro refere-se à definição de **L+**, que é uma linguagem

destinada a prover um ambiente lógico para aplicações como a presente. Esquemas e consultas formulados em outros modelos são traduzidos para L+ de modo a padronizar e compatibilizar as informações com os padrões da máquina de inferências.

Custo/benefício vem em consideração quando se lida com métodos para otimização de consultas. Neste artigo, entretanto, nós não tratamos tal aspecto. A suposição básica é de que as consultas são pequenas quando comparadas às extensões dos bancos de dados em questão. Uma forte razão contudo, é a inexistência de um bom modelo analítico, ou a dificuldade de formulá-lo. Outros autores como King [12] discutem o uso de estimativas de custos para as diversas formulações de uma consulta. Isto é feito depois de iniciado o processo. Estimar o custo do processo propriamente dito contra seu benefício potencial é, neste artigo, um problema em aberto.

Outro ponto importante e não explorado no presente artigo são os efeitos colaterais do uso da presente metodologia. Em [13] nós mostramos a equivalência da forma clausal com os métodos para teste de contenção de expressões relacionais tais como Relational Tableaux [11]. Nós acreditamos que outros métodos de otimização de consultas baseadas na manipulação lógica sejam facilmente obtíveis numa estrutura como a que foi aqui sugerida.

Outros autores tem trabalhado nesta área, normalmente conhecida como otimização semântica de consultas. Os principais trabalhos são Hammer & Zdonik [14] e King [12].

9. Referências

1. Bundy, A. The Computer Modelling of Mathematical Reasoning Academic Press, 1983.

2. Chang, C. & Lee, R. C. Symbolic Logic and Theorem Proving Academic Press, 1973.

3. Stonebraker, M. ; Wong, E. ; Krèps, P. ; Held, G. The Design and Implementation of INGRES. ACM TODS 1, 3 (Setembro 1976).

4 Nilsson, N. J. Problem Solving Methods n Artificial Intelligence, McGraw-Hill, 1971.

5. Silveira, P. M. Database Design and Query Reformulation With an Inference Machine, PhD Thesis, University of Kent, Inglaterra, 1985.

6. Abrial, J. R. Data Semantics, in Database Management, Klimbiee, J. & Koffeman, K. (eds), North-Holland, 1974.

7. Ch lds, D. L. Description of a Set-Theoretic Data Structure, AFIPS Conf. Proc. 33 Part i, 1968, AFIPS Press.

8. Shipman, D. W. The Functional Data Model and the Data Language DAPLEX, ACM TODS 6, 1 (Marco 1981).

9. Jonhson, S. C. Yacc: Yet Another Compiler-Compiler, Computing Science Technical Report N. 32, Bell Laboratories, 1975.

10. Darlington, J. L. Automatic Theorem Proving with Equality Substitutions and Mathematical Induction, in Machine Intelligence 3, Michie, D. (ed), Edinburgh Univ. Press, 1968.

11. Ullman, J. D. Principles of Database Systems, Computer Science Press, 1982 (2. ed).

12. King, J. QUIST: A System for Semantic Query Optimization, Proc. 7th VLDB, 1981.

13. Silveira, P. M. On the Subsumption of Relational Tableaux by Clause Form, submetido ao Congresso SBC, 1987.

14. Hammer, M. & Zdonik-Jr, S. B. Knowledge-Based Query Processing, Proc. 6th VLDB, 1980.

```
        set NOME      s !A-Z,a-z,!40;
        set TITULO    s !A-Z,a-z,0-9!50;
        set ANO       s [1500 .. 2000 by 1];

        set MUSICO  is abstraction;
              Nome -> Nome;
         Ano_Nasc -> Ano;

        set OPUS    is abstraction;
             Titulo -> Titulo;
          Compositor -> Nome;
                Ano -> Ano;

       set GRAVACAO is abstraction;
            Maestro -> Nome;
         Orquestra -> Nome;
                Ano -> Ano;
               Opus -> Titulo;

Constraint Foreign-key-maestro for Gravacao:
           EXISTS Musico SUCH THAT
                  Nome(Musico) = Maestro(Gravacao);

Constraint Foreign-key-opus for Gravacao:
           EXISTS Opus SUCH THAT
                  Titulo(Opus) = Opus(Gravacao);

Constraint Foreign-key-compositor for Opus:
           EXISTS Musico SUCH THAT
                  Nome(Musico) = Compositor(Opus);

Constraint Nao-tao-velho for Musico:
           Ano_Nasc(Musico) > 1600;

Constraint Maestro-moderno for Gravacao, Musico:
           IF   Nome(Musico) = Maestro(Gravacao)
           THEN Ano_Nasc(Musico) > 1850;

Constraint Opus-gravacao for Opus, Gravacao:
           IF   Opus(Gravacao) = Titulo(Opus)
           THEN Ano(Gravacao) > Ano(Opus);

Constraint Maestro-gravacao for Gravacao, Musico:
           IF   Maestro(gravacao) = Nome(Musico)
           THEN Ano(Gravacao) > Ano_Nasc(Musico);

Constraint Compositor-opus for Opus, Musico:
           IF   Compositor(Opus) = Nome(Musico)
           THEN Ano(Opus) > Ano_Nasc(Musico);

Constraint Compositor-opus-gravacao for Musico, Opus, Gravacao:
           IF   ( Nome(Musico) = Compositor(Opus)
           AND    Titulo(Opus) = Opus(Gravacao) )
           THEN (Ano(Gravacao) > Ano_Nasc(Musico)
           AND   Ano(Opus) > Ano_Nasc(Musico) );
```

Figura 2. Esquema L+

```
        set NOME      is !A-Z,a-z!40;
        set TITULO    is !A-Z, a-z, 0-9!50;
        set ANO       is [1500 .. 2000 by 1];

        set MUSICO    is abstraction;
                Nome -> Nome;
            Ano_Nasc -> Ano;

        set OPUS      is abstraction;
          Compositor -> Musico;
              Titulo -> Titulo;
                 Ano -> Ano;

        set GRAVACAO is abstraction;
             Maestro -> Musico;
          Orquestra -> Nome;
                Ano -> Ano;
               Opus -> Opus;

Constraint Opus-gravacao for Gravacao:
           Ano(Gravacao) >= Ano(Opus(Gravacao));

Constraint Maestro-gravacao for Gravacao:
           Ano(Gravacao) > Ano_nasc(Maestro(Gravacao));

Constraint Compositor-opus for Opus:
           Ano(Opus) > Ano_Nasc(Compositor(Opus));

Constraint Compositor-opus-gravacao for Gravacao:
           Ano(Opus(Gravacao)) >  Ano_Nasc(Compositor(Opus(Grava-
           cao)))
              AND
           Ano(Gravacao) > Ano_Nasc(Compositor(Opus(Gravacao)));

Constraint Nao-tao-velho for Musico:
           Ano_Nasc(Musico) > 1600;

Constraint Maestro-moderno for Gravacao:
           Ano_nasc(Maestro(Gravacao)) > 1850;
```

Figura 3. Esquema L+

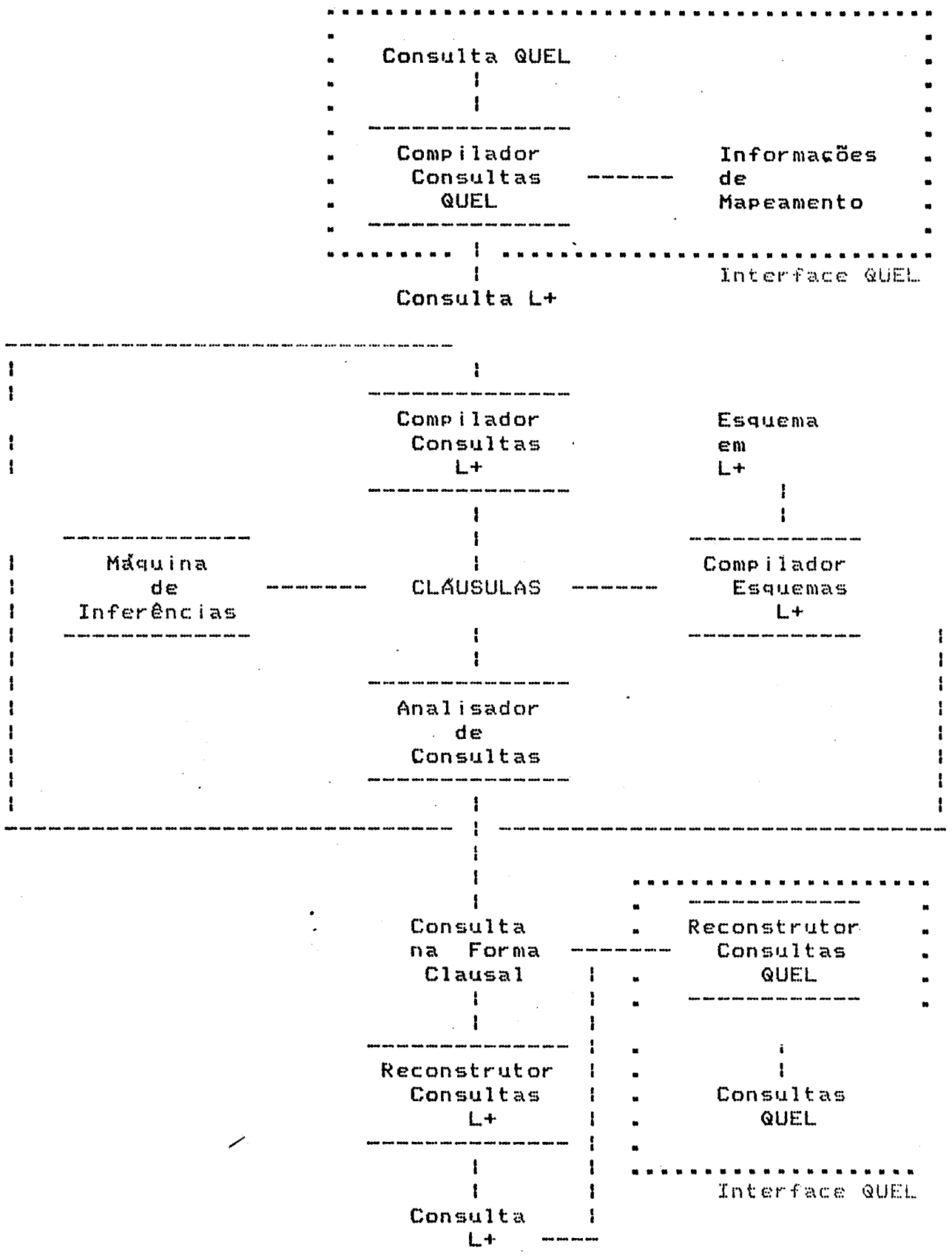

Figura 4. Implementação Experimental