A Very Large Rule/Fact Database based on rule-goal graphs

3P-2

OHMORI, Tadashi and TANAKA, Hidehiko
Univ. of Tokyo, Information Eng. Course

1 Introduction

We have been developing a deductive database with a very large database of rule-clauses [1]. In order to retrieve rule-clauses fast, we use a variant of relational algebra, named *Relational Algebra extended with Unification* (RAU). Among operators of RAU, *I-operator* issues a global query [2] to a database of fact-clauses. $\overset{u}{\bowtie}$ *operator* retrieves pairs of rule-clauses.

This paper presents an algorithm of I-operator by a rule-goal graph and that of $\overset{u}{\bowtie}$-operator by hash.

2 Large Rule/Fact Database

A deductive database in this paper has two very large databases; a database of fact-clauses (factDB) in a relational database and that of rule-clauses (ruleDB). A rule-clause (*rule*) is a view definition of a relation. A fact-clause (*fact*) is a tuple of a relation. As in [3], functor symbols are allowed. Compound terms are regarded as complex objects.

Two cases enlarge a ruleDB; either there are many rules each of which has a different head-predicate, or many different rules have a common head-predicate.

In the first case, an indexed file on a head-predicate can retrieve necessary rules fast. Assume that this ruleDB, *Idb1*, is resident in a main memory. Idb1 with a factDB is a conventional deductive database.

It is enough for us to concentrate on the fast retrieval of rules in the second case, where many rules have a common head-predicate. Let's call this large ruleDB *Idb2*. Idb2 is stored in a disk or a large main memory. In Idb2, necessary rules are retrieved only through unification.

Ex.1.

Idb2 has two rules 'r(f(a,X)):-p1(X).', 'r(f(X,b)):-p2(X).' and also two rules 't(h(X,a)):-q1(g(X),c).', 't(f(X,d)):-q2(X).'. Rules with a common head-predicate are view definitions of a common relation. They operate different relations differently in the body according to complex objects in their arguments. Then, a query is given, "How to do 'r(X) and t(X)' ?". The answer is a set of rules which defines a view m(X) = 'r(X) $\wedge$ t(X)'; i.e. 'm(f(a,d)):-p1(d),q2(a).'. □

If we know 10^3 rules of 'r(X)' and 't(X)' in Ex.1, a naive method tries to unify 10^6 pairs. Hence we need a mechanism to retrieve necessary pairs of rules fast from Idb2.

Our approach is as follows [1]; At first, we compile in advance a body-part of each rule in Idb1 and Idb2 into a program of Idb1. So neither Idb1 nor Idb2 call execution of rules in Idb2.

Secondly, we make a set of rules with a common head-predicate. This set is called a *meta-relation*. It is expressed literally by a set of tuples { *tuple1, tuple2,...* }. In Ex.1, the set R with a scheme $[A, B]$ is made as a set of rules 'r(A):-B.'.

i.e. $R[A,B]$ = { (f(a,X),p1(X)), (f(X,b),p2(X)) }. By the definition of R, a tuple (f(a,X), p1(X)) in R is interpreted as a rule 'r(f(a,X)):-p1(X).'.

In the same way, the set T with a scheme $[A, C]$ is a set of rules 't(A):-C.'. i.e. $T[A,C]$ = { (h(X,a), q1(g(X),c)), (f(X,d), q2(X)) }. For Idb2, each meta-relation will have thousands of tuples.

Thirdly, three set-operators $\overset{u}{\bowtie}$, σ, and *I-operator* are defined on meta-relations. The followings illustrate them on meta-relations R and T defined above.

1. $R[A,B] \overset{u}{\bowtie} T[A,C]$ with a scheme $M[A,B,C]$ is a natural join of R and T, but unification occurs in the join attribute 'A'.

e.g. $M[A,B,C]$ = { (f(a,d), p1(d), q2(a)) }. $M[A,B,C]$ is a set of rules 'm(A):-B, C', which defines a view m(X) = 'r(X) $\wedge$ t(X)'. The single tuple in M is a rule 'm(f(a,d)):-p1(d),q2(a).'. Thus $\overset{u}{\bowtie}$ operator retrieves necessary pairs of rules.

2. σ T[A,C] is a selection of rules in T, which is restricted by unification.

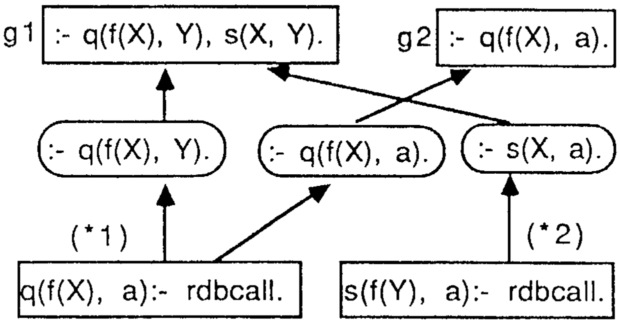
e.g. $\sigma_{A = h(b,\_)} T[A,C]$ = { (h(b,a), q1(g(b),c)) }

3. $I_B\, R[A,B]$ is a set of facts satisfying a rule in R.

e.g. if only p1(i1) and p1(i2) are true and p2(X) has no answers in Idb1, $I_B\, R[A,B]$ = { (f(a,i1),p1(i1)), (f(a,i2), p2(i2)) }.

Those operators are called *Relational Algebra extended with Unification* (RAU). Note that I-operator issues a set of queries to Idb1 with a factDB. $\overset{u}{\bowtie}$ and σ are the same as those in [4].

A query to the rule/fact database is expressed as a tree-form of RAU-operators. e.g. a query "retrieve a set of facts which satisfy both a rule in R restricted by $F1$ and a rule in T restricted by $F2$" is expressed by $\pi\, \mathrm{I}\,[(\sigma_{F1}R) \overset{u}{\bowtie} (\sigma_{F2}T)]$. It retrieves necessary pairs



- original rules in Idb1 $\begin{cases} \text{q(X, a):- rdbcall.} \\ \text{s(f(X), Y) :- rdbcall.} \end{cases}$
- g1, g2 are goal clauses to Idb1
- → :the direction of dataflow.

Figure 1: a rule-goal graph

of rules from Idb2 and issues a global query to Idb1 with a factDB.

3 I-op. by rule-goal graphs

Given a global query , then a rule-goal graph is generated in the *depth first order* resolution with common expression sharing [2] (Figure.1).

We use the simplest rule-goal graph [5]. It consists of *goal nodes* and *rule nodes*. A *goal node* is a goal literal which appears during the depth-first resolution. It has *rule nodes* as child-nodes. A *rule node* is an instance of an original rule, which is a unified form with its parent goal-node. e.g. in Figure.1, a rule node (*1) is a unified form of an original rule. Due to the depth first order, the node *g1* has a goal node ':- s(X,a)', not ':- s(X,Y)'.

If a goal node ':-G.' finds an already generated rule node 'H:-B.' such that G = H θ (θ : substitution), the rule node is declared as a *common expression* and a reduction of ':-G.' succeeds. In Fig.1, the goal *g2* succeeds and a rule node (*1) is a common expression. If *g2* is ':-q(X,a).', the common expression (*1) is replaced by 'q(X,a):-rdbcall.' and is still shared.

The generated rule-goal graph is executed by a relational database after materializing all rule-nodes which are declared as common expressions.

Our rule-goal graph is unique in these two points.

1. variable-bindings are propagated in the depth first order. e.g. in Fig.1, a binding {X/f(Y)} at the node(*2) is not propagated to the node(*1). It is much simpler than the exhaustive propagation in [6].

2. temporal relations are needed only for the declared common expressions. The dataflow approach in [6,7] needs to store temporal results in all nodes in the graph for common expression sharing.

4 $\stackrel{u}{\bowtie}$-op. by hash

This section presents a hash based algorithm for $\stackrel{u}{\bowtie}$ operator. We have already proposed an algorithm for it by hash and sort [1,8]. The sort is not, however, always useful if all operands are resident in a main memory.

For simplicity, in $A \stackrel{u}{\bowtie} B$, operands A and B are sets of terms which have a specified tree structure *Tree*. e.g. let *Tree1* be *functor(atom1, atom2)*, expressed by [*functor, atom1, atom2*] or [*n0, n1, n2*]. Then A is { f(X,a), h(a,b),... }.

Distinct hash-tables *htbl(Bid, Key)* are made according to (Bid, Key); *Bid* of a tuple t is a sequence of 0 or 1. It tells t has a variable or a constant symbol respectively on nodes of Tree. e.g. a tuple t1 = f(X,a) has Bid1 = [1,0,1] on Tree1. We say t1 has a value [f,a] on [n0,n2].

Key in a given Bid is a sequence of constant nodes whose values must be equal when unification succeeds. e.g. Key is [n0] or [n0,n2] for Bid1. If a tuple t2= f(b,a) is unified with any tuple t3 whose Bid is Bid1, both t1 and t3 must have an equal value on Key1 = [n0,n2]. Key1 is called a *Key between t1 and Bid1.*

A tuple t is given a hash value *hv(t)* = ***hash-function(t's value on Key)*** in htbl(Bid, Key). e.g. hv(t1) = hash-function(f,a) in htbl([1,0,1], [n0,n2]), and hash-function(f) in htbl([1,0,1], [n0]).

The hash-based algorithm for $A \stackrel{u}{\bowtie} B$ is as follows;

step1. for all tuple t in A and Key in Bid of t, insert t to the entry 'hv(t)' in htbl(Bid,Key).

step2. for all tuple t in B and Bid in A, do the following operations; get Key between t and Bid, and next try unification of t and tuples in the entry 'hv(t)' in htbl(Bid, Key). □

5 Concluding remarks

An experimental (not large) rule/fact database has been developed in CProlog. I-operator is implemented by the method in this paper. $\stackrel{u}{\bowtie}$ and σ are supported by nested loop and simple hashing, respectively.

The algorithm for $\stackrel{u}{\bowtie}$ in this paper can be improved; When terms in A and B have nodes N which are not on Tree, some coding methods are useful. One way is to append another hash table of SSCW [9] on such nodes N to each entry of our hash table.

[ref.] [1]. Ohmori. IWDM'87, pp.291-304. [2]. Sellis. SIGMOD'86, pp.191-205. [3]. Zaniolo. VLDB'85, pp.458-469. [4]. Morita. VLDB'86, pp.52-59. [5]. Ullman. ACM-TODS. 10(3) '85. [6]. Kifer. ICDT'86, pp.186-202 in LNCS243. [7]. Gelder. SIGMOD'86, pp.155-165. [8]. Ohmori. 32nd.A.C. 1M-6, IPSJ. [9]. Morita. 33rd.A.C. 6L-8, IPSJ.