Third Int. Conf. on Flexible Query Answering Syst., FQAS'98

Optimization of Logic Queries* with MIN and MAX Predicates

Sergio Greco,[1] Carlo Zaniolo[2] and Sumit Ganguly[3]

[1] DEIS, Univ. della Calabria, 87030 Rende, Italy
greco@si.deis.unical.it
[2] Computer Science Dept., Univ. of California, Los Angeles, CA 90024
zaniolo@cs.ucla.edu
[3] Dept. of Computer Science, Rutgers University, New Brunswick, NJ 08903
sumit@cs.rutgers.edu

Abstract. We propose an algorithm for pushing *min* and *max* aggregates into recursive predicates, while preserving query equivalence under certain monotonicity constraints. The transformed query is often safe when the original one is not, and more efficient than the original query when this is safe.

1 Introduction

The program in Example 1, below, gives a declarative formulation of the shortest path problem. Given a weighted directed graph represented as a base relation `arc`, where all edge weights are nonnegative, the predicate `path` computes the set of all triples `(X,Y,C)` such that there is a path from node `X` to node `Y` whose cost is `C`. The predicate `sh_path(X,Y,C)` is intended to yield all the triples `(X,Y,C)` such that `C` is the least cost among all paths from node `X` to node `Y`.

Example 1. Stratified shortest path.

$$\mathtt{sh\_path(X,Y,C)} \leftarrow \mathtt{min(C,(X,Y),\ path(X,Y,C))}.$$

$$\mathtt{path(X,Y,C)} \leftarrow \mathtt{arc(X,Y,C)}.$$
$$\mathtt{path(X,Y,C)} \leftarrow \mathtt{path(X,Z,C1),\ arc(Z,Y,C2),\ C = C1 + C2}. \qquad \Box$$

A precise semantics can be assigned to our program, by simply viewing the first rule containing the *min* predicate as a short hand for the following two rules:

$$\mathtt{sh\_path(X,Y,C)} \leftarrow \mathtt{path(X,Y,C),\ \neg a\_lesser\_path(X,Y,C)}.$$

$$\mathtt{a\_lesser\_path(X,Y,C)} \leftarrow \mathtt{path(X,Y,C'),\ C' < C}.$$

where `a_lesser_path` is a new predicate symbol not appearing elsewhere in the program. This has formal semantics, inasmuch as by rewriting the min predicates by means of negation, we get a stratified program [1]. However, a straightforward evaluation of such a stratified program would materialize the predicate

* The work of the first author has been supported by the project MURST 40 % "".

path(X,Y,C) and then choose the smallest cost tuple for every X and Y. There are two problems with this approach: first, it is very inefficient, and second, the computation could be non-terminating if the relation arc is cyclic. To solve these problems, one begins by observing that all minimum paths of length $n+1$, because the weights are nonnegative, can be generated from the minimum paths of length n. Thus, the *min* predicate can be pushed into recursion, in such a way that the generation of new paths is interleaved with the computation of shortest paths, yielding the program of Example 2.

Example 2. Unstratified shortest path.

$$\begin{array}{l} \mathtt{path(X,Y,C)} \leftarrow \mathtt{arc(X,Y,C)}. \\ \mathtt{path(X,Y,C)} \leftarrow \mathtt{sh\_path(X,Z,C1)},\ \ \mathtt{arc(Z,Y,C2)},\ \ \mathtt{C = C1 + C2}. \\ \mathtt{sh\_path(X,Y,C)} \leftarrow \mathtt{min(C,(X,Y),path(X,Y,C))}. \end{array}$$ □

Unfortunately, as we attempt to give a meaning to Example 2 by rewriting it using negation, as in the previous example, we obtain a non-stratified program, with all the open semantic and computational issues therewith. These problems, widely recognized by deductive database researchers, have been the subject of recent works [2, 3, 5, 7, 8, 9, 10, 14, 16, 18, 21]. Although our discussion deals explicitly only with *min* programs, the symmetric properties of *max* programs follow by duality.

Here we use Datalog to study how min and max aggregates are propagated down into queries, since this simplies the presentation; however, similar techniques can be applied directly to the optimization of recursive SQL queries.

Observe that stratified aggregation has very high expressive power on ordered domains [11] and, therefore, queries should be written by using stratified aggregates which have clear and intuitive semantics. Moreover, the computation of queries with stratified *min* and *max* aggregates tend to be inefficient; however, the computation of queries with unstratified min and max aggregates can be very efficient, since useless branches can now be pruned early in the computation. In particular, a very efficient algorithm for the computation of monotonic queries with unstratified min and max aggregates was presented in [5]. Therefore, we are seeking techniques to rewrite stratified queries into equivalent unstratified ones.

2 Syntax and Semantics

The notion of a minimum naturally assumes the existence of a domain of constants over which a total order is defined. Formally, we assume the existence of an alphabet of constants, functions and predicates symbols. We assume also that there are a built-in set of constants D, called *cost domain*, two built-in binary predicates $<$ and $\leq$, and built-in functions whose arguments are elements of D. In our examples, we will use the cost domain of real numbers, where the built-in predicates $<$ and $\leq$ are the usual operators defined for real numbers and the built-in functions are the usual arithmetics operators ($+$, $-$, $*$, etc.).

We assume the basic concepts of terms, atom, Horn program, (Herbrand) interpretation and (Herbrand) model of logic programming.

Definition 1. *A* special atom *is of the form* $min(C, S, Q)$ *where:*

1. *Q is an atom of first order logic, called* minimized atom*,*
2. *S is a set of variables, appearing as arguments Q, called* grouping variables*,*
3. *$C \notin S$ is a variable, appearing as argument of Q, called the* cost variable*. Cost variables can only take values from the cost domain.* □

A *min atom* is either a special atom or an atom of first order logic (standard atom). A min rule is of the form $A \leftarrow B_1, ..., B_n$ where A is a standard atom and $B_1, ..., B_n$ are min atoms. A *ruleset* is a finite set of min rules. A *(min) program* is defined by a ruleset P, its underlying cost domain D_P plus an additional constraint defining the subrange of the cost domain whose values can be assigned to cost arguments; this subrange will be called the *valid cost sub-domain*. Both Examples 1 and 2 are min program.

A *(min) query* Q is a pair $\langle g(X), P\rangle$ where P is a min program program, g is a predicate symbol in P, say with arity $n \geq 0$, and X is a list of n variables; the atom $g(X)$ is said query-goal. Given a database D, the *answer* of Q on D, denoted by $Q(D)$, is the set of relations on g denoted as $A_g = \{M(g)|M$ is a (minimal) model defining the semantics of $P \cup D\}$. We say that two queries Q_1 and Q_2 are equivalent ($Q_1 \equiv Q_2$) if for each database D the answers of Q_1 and Q_2 on D are the same. Given a query $\langle g(X), P\rangle$ and a predicate p appearing in P, we say that p is reachable from g if g depends on p. A predicate g depends on a predicate q if there is a rule r such that g appear in the head and p in the body, or there is a predicates q such that g depends on q and q depends on p. Moreover, we say that a rule r is reachable from a predicate g(or equivalently, g depends on r) if the head predicate symbol of r is reachable from g.

For the sake of simplicity we consider min programs without negation. The extension to programs with stratified negation is straightforward.

Without loss of generality we assume that min rules are either standard Horn rules or non-Horn rules of the form

$$H \leftarrow min(C, S, Q)$$

where H and Q are standard atoms.

The semantics of a min program is defined by taking a min program P and defining a first order formula $foe(P)$, called the first order extension of P, obtained by replacing every occurrence of special atoms

$$min(C, S, q(\bar{Y}, C))$$

by the pair of goals

$$q(\bar{Y}, C), \quad \neg a\_lesser\_q(\bar{Y}, C)$$

The distinguished a_lesser_q predicate is defined by the additional rule of the form:

$$a\_lesser\_q(\bar{Y}, C) \leftarrow q(\bar{X}, C'),\ C' < C.$$

where an argument variable in $\bar{X}$ coincides with the corresponding variable in $\bar{Y}$ if this variable appears in the grouping set S and is a new variable otherwise. The variable C' is new and corresponds to the variable C.

For instance, the meaning of Example 2 is defined by the following program:

Example 3. The standard program defining the meaning of Example 2 is

$\texttt{path(X, Y, C)} \leftarrow \texttt{arc(X, Y, C)}.$
$\texttt{path(X, Y, C)} \leftarrow \texttt{sh\_path(X, Z, C1)},\ \texttt{arc(Z, Y, C2)},\ \texttt{C} = \texttt{C1} + \texttt{C2}.$

$\texttt{sh\_path(X, Y, C)} \leftarrow \texttt{path(X, Y, C)},\ \neg\texttt{a\_lesser\_path(X, Y, C)}.$

$\texttt{a\_lesser\_path(X, Y, C)} \leftarrow \texttt{path(X, Y, C')},\ \texttt{C}' < \texttt{C}.$ □

Since we view a min-program simply as a shorthand of $foe(P)$, we will say that M is an interpretation (resp. a (minimal) model) for P, when M is an interpretation (resp. a (minimal) model) for $foe(P)$. Therefore, we can define the semantics of min-programs by using concepts such as stratification, well founded model and stable models, developed for logic programs with negation.

We can now proceed and define classes of min programs which, because of their syntactic structure and stratification conditions induced by the underlying cost-domain, have a total well-founded model which coincides with the unique stable models and with the perfect model which defines the semantics of (locally) stratified programs.

We will assume that certain database predicates arguments called *cost arguments* can only take values from the cost domain, or a subset of this domain called a *valid cost sub-domain*. In most examples of this paper, we will assume that the set of nonnegative numbers is our valid cost sub-domain. The presence of database goals with cost arguments, and the fact that built-in predicates can be true only if their arguments are from the cost domain, implies that certain arguments in the derived predicates also belong to the cost domain.

We point out that the notion of valid cost domain is motivated by the fact that programs satisfy some particular properties only if we consider a subset of the cost domain.

A *valid instantiation* of a rule r in a program $foe(P)$ is a ground instance of r that is obtained by replacing all the variables of r with ground terms from $U_{foe(P)} \cup D'_P$ where (i) D'_P is a valid cost subdomain, and (ii) each goal corresponding to a built-in predicate is true. Thus valid instantiations will not contain a goal such as $3 > 4 + 1$ which is false and thus inconsequential. The valid instantiation of program $foe(P)$ is simply the set of all valid instantiations of all rules of $foe(P)$. The *valid dependency graph* of a program $foe(P)$, denoted G_P, is a directed graph whose nodes are ground atoms of $foe(P)$. In G_P, there is an arc from a node A to another node B if there is a valid instantiation of some rule r in $foe(P)$ such that A appears in the body and B appears in the head of r; moreover if A appears negated then the arc is marked, and will be said to be negative. The *immediate consequence operator* T_P is defined as

$$T_P(I) = \{q \mid \exists q \leftarrow p_1, ..., p_n \in ground(P) \text{ and } p_1, ..., p_n \in I\}.$$

We now formally introduce the definition of cost argument.

Definition 2. *Let P be a min program, and let $det(P)$ denote the program obtained from $foe(P)$ by*

1. *removing all the* `a_lesser` *rules and*
2. *removing all* `a_lesser` *goals from the bodies of the rules.*

The i-th argument of a predicate q of P, is said to be a cost argument *if the i-th argument of every q-atom in $T^{\infty}_{det(P)}(\emptyset)$ belongs to the cost domain. Moreover, a predicate q that has a cost argument will be called a* cost predicate*.*

□

Thus, given a min program P it is possible to divide the predicate symbols appearing in P into the two distinguished sets of cost predicates and standard predicates. We will only consider programs where each recursive set of predicates consists of cost predicates (called *min clique*) or standard predicates (called *standard clique*). Standard cliques are computed by using the classical fixpoint computation whereas min cliques are computed by using special algorithms such as the greedy fixpoint of [5]. Predicates that appear inside special atoms will be called *minimized predicates*.

3 Propagation of min predicates

In this section, we present an algorithm for transforming monotonic-min programs into *query-equivalent* programs which can be more efficiently evaluated using the semi-naive greedy operator G_P [5].

We associate adornments to minimized cost predicates by assigning to each of their arguments, one of the three labels u, e, or m as follows:

1. the cost argument is labeled m
2. an argument not containing in the set of grouping variables is labeled e
3. an argument appearing in the set of grouping variables is labeled u.

Consider, for instance, the min atom $min(C, (X, Y), p(X, Z, Y, C))$. The adorned atom associated with the minimized atom is $p^{\text{ueum}}(X, Z, Y, C)$.

Observe that u-variables and e-variables are, respectively, universally and existentially quantified in the foe expansion of the min predicates. Observe also that grouping variables correspond, in non-recursive queries, to arguments appearing in the "GROUP BY" section in SQL queries.

Predicates that are not minimized will then be assigned the default adornment ϵ. For instance, in Example 1, the adornment for both *sh_path* is ϵ whereas the adornment for *path* is *uum*.

Thus, we use the concept of *adornments* to abbreviate minimized predicates. The basic component of the transformation algorithm is the propagation of an adornment for the head of a rule into the body of the rule. We discuss this for Horn rules and non-Horn rules next.

3.1 Propagating Adornments into Horn Rules

Consider the following rule for which we wish to propagate the adornment *uum* for $\mathtt{p}$ into the body of the rule.

$$\mathtt{p(X,Z,C)} \leftarrow \mathtt{q(X,Y,C1)},\ \ \mathtt{s(Y,Z,C2)},\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$

In this case, a particular tuple $\mathtt{p(X,Z,C)}$ is a *possible* candidate member of $\mathtt{p}^{\mathrm{uum}}$ provided that the conjunction $\mathtt{q}^{\mathrm{uum}}(\mathtt{X,Y,C1}) \wedge \mathtt{s}^{\mathrm{uum}}(\mathtt{Y,Z,C2})$ is true. Clearly, this condition is necessary but not sufficient, since, the combination of two tuples from $\mathtt{q}^{\mathrm{uum}}$ and $\mathtt{s}^{\mathrm{uum}}$ may not result in a tuple for $\mathtt{p}^{\mathrm{uum}}$. This motivates us to introduce a new *surrogate* predicate cp^{uum} (the name for the surrogate predicate symbol is obtained by prefixing c to the name of the predicate and by superscripting the adornment) whose definition is intended to collect the set of candidate minimum tuples for the given adornment. Thus the propagation of the adornment *uum* in the above rule results in the following rule:

$$\mathtt{cp}^{\mathrm{uum}}(\mathtt{X,Z,C}) \leftarrow \mathtt{q}^{\mathrm{uum}}(\mathtt{X,Y,C1}),\ \ \mathtt{s}^{\mathrm{uum}}(\mathtt{Y,Z,C2}),\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$

If there are multiple rules that define the predicate p, we generate one such adorned rule for each of the rules. Consequently, we define $\mathtt{p}^{\mathrm{uum}}$ as the minimum over $\mathtt{cp}^{\mathrm{uum}}$ as follows:

$$\mathtt{p}^{\mathrm{uum}}(\mathtt{X,Z,C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X,Z}),\mathtt{cp}^{\mathrm{uum}}(\mathtt{X,Z,C})).$$

The above rule is called the *minima definition* of $\mathtt{p}$, in terms of the surrogate predicate $\mathtt{cp}^{\mathrm{uum}}$ for the adornment *uum*. In this manner, we generate the *adorned definition* of a predicate for a given adornment, which is the union of the adorned rules for the predicate and the minima definition of the predicate in terms of the surrogate predicate for that adornment.

Adorning variables by u

In the above example, the variable $\mathtt{Y}$ is shared by both $\mathtt{q(X,Y,C1)}$ and $\mathtt{s(Y,Z,C2)}$ in the body of the rule. It is safe to adorn $\mathtt{Y}$ as u in both the $\mathtt{q}$ and $\mathtt{s}$ predicates in the body. However, in general, it is not safe to adorn $\mathtt{Y}$ as e in either or both the predicates. Suppose that, for instance, we adorn $\mathtt{Y}$ by e in $\mathtt{q(X,Y,C1)}$ and by u in $\mathtt{s(Y,Z,C2)}$. This would give the following adorned rule

$$\overline{\mathtt{cp}}^{\mathrm{uum}}(\mathtt{X,Z,C}) \leftarrow \mathtt{q}^{\mathrm{uem}}(\mathtt{X,Y,C1}),\ \ \mathtt{s}^{\mathrm{uum}}(\mathtt{Y,Z,C2}),\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$

It should be clear that $\overline{\mathtt{cp}}^{\mathrm{uum}}$ may be a proper subset of $\mathtt{cp}^{\mathrm{uum}}$ since corresponding to the $\mathtt{Y}$ values in $\mathtt{q}^{\mathrm{uem}}(\mathtt{X,Y,C1})$, there may not be any $\mathtt{Y}$ values in $\mathtt{s(Y,Z,C2)}$ (and therefore in $\mathtt{s}^{\mathrm{uum}}(\mathtt{Y,Z,C2})$[4]). Hence *shared variables in the body of a rule are adorned as u.*

Furthermore, consider the variables $\mathtt{X}$ and $\mathtt{Z}$ in the body and whose adornments are prescribed as u in the head. These variables are not shared in the body. In general, *variables that are adorned as u in the head are adorned as u in the body*, irrespective of whether they are shared or not.

[4] unless $\mathtt{s}$ is total in its first argument.

Adorning variables by e

Consider the following rule and the adornment uem to be propagated in the body:

$$\mathtt{p(X,Z,C)} \leftarrow \mathtt{q(X,U,C1)},\ \ \mathtt{s(W,Z,C2)},\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$

Clearly, X must be adorned as u since it is so adorned in the head. Let us now consider the variables U and W. These variables are neither shared nor do they appear in the head. We adorn these variables by e, since their values are inconsequential. The variable Z in the body is also adorned as e since, it appears only once in the body and is adorned as e in the head. Thus, *variables that appear only once in the body and do not appear in the head are adorned as e. Variables that appear only once in the head as e and appear only once as in the body are adorned as e.*

Monotonicity of the cost predicate in the body

We now consider the constraints that the cost predicate must satisfy in order to propagate adornments. Consider the following rule, which is identical to the one considered previously except for the cost predicate, and the adornment uum (same as before).

$$\mathtt{p(X,Z,C)} \leftarrow \mathtt{q(X,Y,C1)},\ \ \mathtt{s(Y,Z,C2)},\ \ \mathtt{C} = \mathtt{C1} + (\mathtt{C2\ mod\ 5}).$$

However, in this case, the adornment cannot be propagated into the body predicate s(Y, Z, C2) since the minimum value of C2 (as an integer) may not necessarily imply the minimum contribution of (C2 mod 5) to C. However, the cost variable for the head is a non-decreasing monotonic function of the cost variable for the q-predicate and hence the adornment can be propagated into the q-predicate. The adorned rule for this case is

$$\mathtt{cp}^{\mathrm{uum}}\mathtt{(X,Z,C)} \leftarrow \mathtt{q}^{\mathrm{uum}}\mathtt{(X,Y,C1)},\ \ \mathtt{s(Y,Z,C2)},\ \ \mathtt{C} = \mathtt{C1} + (\mathtt{C2\ mod\ 5}).$$

This example shows that some body predicates may stay unadorned and are called unadornable predicates. All adornments to unadornable predicates resulting from the application of the previous rules are ignored.

Totality of the cost predicate in the body

We now consider the issue of totality of the cost predicate that is raised by the following example.

$$\mathtt{p(X,Z,C)} \leftarrow \mathtt{q(X,Y,C1)},\ \ \mathtt{s(Y,Z,C2)},\ \ \mathtt{C} = \mathtt{log}(\mathtt{C1} + \mathtt{C2} - 5).$$

Suppose that (as before) we wish to propagate the adornment uum into the body. In this case, however, it is not possible to propagate the adornment into any of the body predicates. Suppose that for a certain X, Y and Z, the minimum value for C1 is 2 and for C2 is 2. However, $\mathtt{C} = \mathtt{log}(-1)$ is undefined. Hence, the adorned rule for this example is

$$\mathtt{cp}^{\mathtt{uum}}\mathtt{(X,Z,C)} \leftarrow \mathtt{q(X,Y,C1)},\ \ \mathtt{s(Y,Z,C2)},\ \ \mathtt{C} = \mathtt{log}(\mathtt{C1} + \mathtt{C2} - 5).$$

We formally define total mappings below.

Definition 3. Let P be a min program with cost domain D_P and valid cost subdomain C_P. Let $f(X_1, \ldots, X_n)$ be an n-ary function on the cost domains. We say that the equality goal $Y = f(X_1, \ldots, X_n)$ defines a *total monotonic* mapping from X_i, $(1 \leq i \leq n)$ to Y when the following two conditions are satisfied:

totality: if $X_1, ..., X_n \in C_P$ then $f(X_1, \ldots, X_n) \in C_P$, and
monotonicity: if $X_1, ...X_n, U, V \in C_P$ and $V \leq U$ then
$f(X_1, \ldots, X_{i-1}, V, X_{i+1}, \ldots, X_n) \leq f(X_1, \ldots, X_{i-1}, U, X_{i+1}, \ldots, X_n)$. □

The rules for propagating adornments into the body of Horn rules are summarized below. We use the following running example:

Example 4.

$$\texttt{cp}^{\mathrm{uem}}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{q}(\texttt{X},\texttt{Z},\texttt{N1},\texttt{C1}),\ \texttt{r}(\texttt{Z},\texttt{Y},\texttt{N2},\texttt{C2}),\ \texttt{s}(\texttt{Z},\texttt{W},\texttt{C3}),$$
$$\texttt{N1} = \texttt{N2+1},\ \ \texttt{C} = \texttt{C1+C2+(C3 mod 5)}.$$

where the last goal in the body of the rule define a total monotonic mapping from `C1` and `C2` ro `C`; the mapping from `C3` to `C` is not monotonic. □

Propagation of adornments

Given a rule r with head predicate $p(\bar{X}, C)$, we denote with $B(r)$ the set of cost predicates in the body of r and with $TC(r)$ the set of cost predicates in $B(r)$ such that for each $q(\bar{Y}, C') \in B(r)$ there is a total-monotonic mapping from C' to C and C' is a variable not appearing in any other predicate in $B(r)$. In our example $TC(r) = \{\texttt{q}(\texttt{X},\texttt{Z},\texttt{N1},\texttt{C1}), \texttt{r}(\texttt{Z},\texttt{Y},\texttt{N2},\texttt{C2})\}$ and $B(r) = TC(r) \cup \{\texttt{s}(\texttt{Z},\texttt{W},\texttt{C3})\}$.

Given a variable X we denote with $\phi(X)$ the set of variables containing X plus all variables whose values depend on the value of X, i.e., all variables which are connected to X through some relational condition. For instance, in Example 4 we have $\phi(N1) = \{N1, N2\}$.

1. Predicates in $B(r) - TC(r)$ are adorned with the empty adornment ϵ. We call such predicates *non adorned predicates*.
2. Predicates in $TC(r)$ are adorned with the an adornment $\alpha \neq \epsilon$. We call such predicates *adorned predicates*. Each predicate $q(X_1, ..., X_n, C) \in TC(r)$, where C is the variable denoting the cost argument, is adorned as follows:
   (a) The cost argument C is labeled with m ($C1$ and $C2$ in Example 4);
   (b) Each variable labeled as u in the head is labeled as u in the body (X in Example 4).
   (c) Each variable X_i $(i \leq 1 \leq n)$ is labeled as u if there is a variable in $\phi(X_i)$ appearing in some other non built-in goal ($Z, N1$ and $N2$ in Example 4).
   (d) Each variable X_i $(i \leq 1 \leq n)$ which has not be labeled by m or u in steps $a - -c$ is labeled as e (This is the case of Y in Example 4.)

The adorned rule for the Example 4 is:

$$\texttt{cp}^{\mathrm{uem}}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{q}^{\mathrm{uuum}}(\texttt{X},\texttt{Z},\texttt{N1},\texttt{C1}),\ \texttt{r}^{\mathrm{ueum}}(\texttt{Z},\texttt{Y},\texttt{N2},\texttt{C2}), \texttt{s}(\texttt{Z},\texttt{W},\texttt{C3}),$$
$$\texttt{N1} = \texttt{N2+1},\ \ \texttt{C} = \texttt{C1+C2+C3+(C3 mod 5)}.$$

Minima Definition of a predicate in terms of its surrogate predicate Given a Horn clause with head predicate symbol p, the above rules generate the adorned version of each rule defining p, for a given adornment α. The adorned version of each rule defines a surrogate predicate cp^{α}. We complete the minima definition of p in terms of cp^{α} for the adornment α by introducing the following rule in the transformed program:

$$\mathtt{p}^{\alpha}(\mathtt{X_1}, ..., \mathtt{X_n}) \leftarrow \mathtt{min}(\mathtt{X_i}, \mathtt{W}, \mathtt{cp}^{\alpha}(\mathtt{X_1}, ..., \mathtt{X_n}).$$

where X_i is the cost argument and W is the set of variables adorned as u in α. The above rule is denoted by $mindef(p, cp^{\alpha}, \alpha)$ and is read as the minima definition of p in terms of cp^{α} as determined by the adornment α.

3.2 Propagating Adornments into Non-Horn rules

In this section, we discuss how to propagate an adornment into the body of a non-Horn rule.

Consider the following non-Horn rule.

$$\mathtt{p}(\mathtt{X}, \mathtt{Y}, \mathtt{C}) \leftarrow \mathtt{min}(\mathtt{C},\ (\mathtt{X}),\ \mathtt{q}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{C}).$$

The above rule may be rewritten as

$$\mathtt{p}(\mathtt{X}, \mathtt{Y}, \mathtt{C}) \leftarrow \mathtt{q}^{\mathrm{ueem}}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{C}).$$

where the definition of $\mathtt{q}^{\mathrm{ueem}}$ is obtained using a surrogate predicate $\mathtt{cq}^{\mathrm{ueem}}$ and the rules for propagation (see Section 3.1). Now suppose that the adornment for the head to be propagated is uem. Then, the adorned rule is

$$\mathtt{p}^{\mathrm{uem}}(\mathtt{X}, \mathtt{Y}, \mathtt{C}) \leftarrow \mathtt{q}^{\mathrm{ueem}}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{C}).$$

This example shows that if all variables adorned as u in the body are also adorned as u in the head, then no change is made to the body. This further implies that if the adornment for the head is uum, then the body of the adorned rule remains unchanged, as follows:

$$\mathtt{p}^{\mathrm{uum}}(\mathtt{X}, \mathtt{Y}, \mathtt{C}) \leftarrow \mathtt{q}^{\mathrm{ueem}}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{C}).$$

Indeed, while the adornment uum requires the minima for every value of `X` and `Y`, the body of the original rule, whose adornment is $ueem$, returns minima for `X` alone. Hence, `Y` is adorned as e in the body. If the adornment is eum, then the adorned rule is

$$\mathtt{p}^{\mathrm{eum}}(\mathtt{X}, \mathtt{Y}, \mathtt{C}) \leftarrow \mathtt{q}^{\mathrm{eeem}}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{C}).$$

The rules for propagating an adornment α into a non-Horn rule

$$r : p(X_1, \ldots, X_n) \leftarrow min(Y_i,\ W,\ q(Y_1, \ldots, Y_n).$$

are as follows:

1. First, the non-Horn rule is replaced by the following rule

$$p^{\alpha}(X_1, \ldots, X_n) \leftarrow q^{\beta}(Y_1, \ldots, Y_n).$$

   where β is the adornment corresponding to the min-predicate of the body.
2. Then, we construct the adornment γ for the body of the rule as follows: the variable Y_i denoting the cost-argument is labeled as m; a variable is labeled as e in γ if it is labeled as e in α or in β; the remaining variables are labeled as u[5]. The adorned rule is

$$p^{\alpha}(X_1, \ldots, X_n) \leftarrow q^{\gamma}(Y_1, \ldots, Y_n).$$

3.3 Adorned definition of predicates

So far, we have discussed how to propagate an adornment for the head of a rule into the body to generate the adorned version of the rule.

For derived predicates, the *adorned definition* of a predicate p, for an adornment α, is the set of all rules defining the surrogate predicate cp^{α} and the minima definition of p in terms of cp^{α} for the adornment α.

If p is a base predicate, then the adorned definition of p w.r.t. the adornment α is the minima definition p in terms of p for the adornment α, i.e.,

$$p^{\alpha}(X_1, \ldots, X_n) \leftarrow min(X_i, W, p(X_1, \ldots, X_n)).$$

where X_i is the variable adorned as m in α and W is the set of variables adorned as u in α.

For sake of simplicity, we introduce the symbol ϵ, denoting the empty adornment. The minima definition of any predicate p in terms of another predicate q for ϵ is the empty set of rules. For propagating adornments from the head to the body of a rule, the adornment ϵ is considered as an abbreviation for the adornment that labels all variables (except the cost variable) by u and the cost variable by m.

We illustrate the adorned definition of predicates by the following example.

Example 5. Consider the program of Example 1. The adorned definition of the predicate `sh_path` *w.r.t. the adornment* ϵ *is:*

$$adorn\text{-}pred(\texttt{sh\_path}, \epsilon) = \{\ \texttt{sh\_path}^{\epsilon}(\texttt{X}, \texttt{Y}, \texttt{C}) \leftarrow \texttt{path}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C}).\ \}$$

Thus the adorned definition of the predicate `path` *appearing in the program with the adornment uum is*

$$adorn\text{-}pred(\texttt{path}, uum) = \left\{ \begin{array}{l} \texttt{cpath}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C}) \leftarrow \texttt{arc}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C}). \\ \texttt{cpath}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C}) \leftarrow \texttt{path}^{\text{uum}}(\texttt{X}, \texttt{Z}, \texttt{C1}), \texttt{arc}^{\text{uum}}(\texttt{Z}, \texttt{Y}, \texttt{C2}), \\ \qquad\qquad\qquad\qquad\quad \texttt{C} = \texttt{C1+C2}. \\ \texttt{path}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C}) \ \leftarrow \texttt{min}(\texttt{C}, (\texttt{X}, \texttt{Y}), \texttt{cpath}^{\text{uum}}(\texttt{X}, \texttt{Y}, \texttt{C})). \end{array} \right\}$$

[5] We assume that there are not repeating variables in the head or in the body of canonical non-Horn rules

whereas the adorned definition of the predicate `arc` *appearing in the program with the adornment uum is*

$$adorn\text{-}pred(\texttt{arc}, uum) = \{\ \texttt{arc}^{\mathrm{uum}}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{min}(\texttt{C},(\texttt{X},\texttt{Y}),\texttt{arc}(\texttt{X},\texttt{Y},\texttt{C})).\} \quad \square$$

We now formally define the adorned definition of a predicate p, w.r.t. a given adornment α. In order to do so, we use the following terminology.

1. $adorn\text{-}rule(r, \alpha)$ is the adorned rule obtained by propagating the adornment α for the head of r into the body of r.
2. $mindef(p, q, \alpha)$ denotes the rule that defines p as the minima of q for the adornment α.

The adorned definition of a predicate p w.r.t. an adornment α, denoted by *adorn-pred*(p, α) is equal to

- $mindef(p, p, \alpha)$, if p is a base predicate
- $\{adorn\text{-}rule(r, \alpha) \mid r \text{ defines } p\} \cup \{mindef(p, cp^{\alpha}, \alpha)\}$, otherwise.

3.4 Algorithm for Adorning a Program

In this section, we discuss an algorithm that generates an adorned program from a given min-program and a query predicate.

The algorithm begins by generating the adorned definition of the query predicate, whose symbol we assume to be q. This may generate new adorned predicate symbols. The adorned definitions of these predicate symbols are (recursively) included in the adorned program until no new adorned predicate symbol remains.

General min-programs may have predicates without cost arguments, or predicates whose cost arguments are either non-monotonic or are not total. These predicates may not yield any adorned definitions during the propagation algorithm. Hence, the definition of such predicates must be included in the adorned program.

A simple way of achieving this is to initiate the adornment propagation algorithm with the query predicate symbol q and the adornment ϵ. After all adorned definitions are generated, the set of rules reachable from q^{ϵ} form the adorned program. Finally, in the adorned program, all occurrences of q^{ϵ} are replaced by q. We illustrate the algorithm by means of the following example.

Example 6. Consider the following program P, reproduced from Example 1.

$$\texttt{sh\_path}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{min}(\texttt{C},(\texttt{X},\texttt{Y}),\ \texttt{path}(\texttt{X},\texttt{Y},\texttt{C})).$$

$$\begin{aligned}&\texttt{path}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{arc}(\texttt{X},\texttt{Y},\texttt{C}).\\ &\texttt{path}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{path}(\texttt{X},\texttt{Z},\texttt{C1}),\ \texttt{arc}(\texttt{Z},\texttt{Y},\texttt{C2}),\\ &\texttt{path}(\texttt{X},\texttt{Y},\texttt{C}) \leftarrow \texttt{C} = \texttt{C1} + \texttt{C2}.\end{aligned}$$

Suppose that the query predicate symbol is `sh_path`. We begin the propagation algorithm with the adornment ϵ for `sh_path`. By propagating the adornment ϵ

into the only rule that defines $\mathtt{sh\_path}$, we obtain the following rule, which is included in P.

$$\mathtt{sh\_path}^{\epsilon}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}).$$

However, a new adorned predicate symbol $\mathtt{path}^{\mathrm{uum}}$ is obtained. Hence, we now include the adorned definition of $\mathtt{path}^{\mathrm{uum}}$ into P, which is as follows.

$$\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Z},\mathtt{C1}),\ \ \mathtt{arc}^{\mathrm{uum}}(\mathtt{Z},\mathtt{Y},\mathtt{C2}),\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$
$$\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{arc}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}).$$
$$\mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X},\mathtt{Y}),\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C})).$$

A new adorned symbol $\mathtt{arc}^{\mathrm{uum}}$ is generated, whose adorned definition is included in P.

$$\mathtt{arc}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X},\mathtt{Y}),\mathtt{arc}(\mathtt{X},\mathtt{Y},\mathtt{C})).$$

We now obtain the set of rules in P that are reachable from $\mathtt{sh\_path}^{\epsilon}$, which is listed below.

$$\mathtt{sh\_path}^{\epsilon}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}).$$
$$\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Z},\mathtt{C1}),\ \ \mathtt{arc}^{\mathrm{uum}}(\mathtt{Z},\mathtt{Y},\mathtt{C2}),\ \ \mathtt{C} = \mathtt{C1} + \mathtt{C2}.$$
$$\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{arc}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}).$$
$$\mathtt{path}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X},\mathtt{Y}),\mathtt{cpath}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C})).$$
$$\mathtt{arc}^{\mathrm{uum}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X},\mathtt{Y}),\mathtt{arc}(\mathtt{X},\mathtt{Y},\mathtt{C})).$$

Thus, the original rules which are not reachable from $\mathtt{sh\_path}^{\epsilon}$ are not included in the adorned program. The final adorned program is obtained by replacing $\mathtt{sh\_path}^{\epsilon}$ in the above program by $\mathtt{sh\_path}$. □

The algorithm which propagates min predicates is reported in Figure 1 where the variables S and T contain, respectively, the set of predicate symbols to be adorned and the set of predicate symbols already adorned; $body(r)$ denotes the set of atoms in the body of a rule r.

3.5 Query Equivalence of Original and Adorned Programs

In this section, we characterize the query equivalence between the given program and the adorned program obtained as the output of Algorithm 1.

Note that Algorithm 1 includes the adorned definition of predicates into a given program P at each iteration. Hence, until termination, there is at least one adorned predicate symbol in P which is not defined in P, making it "incomplete". In order to prove the equivalence of the adorned program and the original program, we begin by completing the definition of such incomplete programs.

The *adorn-completion*(P), for a given set of rules P, is P union with $mindef(p, p, \alpha)$, for every p^{α} which appears in the body of some rule in P but is undefined in P, that is, for each undefined n-ary predicate symbol p^{α} a rule of the form

Algorithm 1. *Propagation of Min Predicates*
Input: Min program P and a goal $q(t)$.
Output: Adorned min program;
var $S, T, \textit{new-pred}$: set of predicate symbols;
  p : predicate symbol;
  α: adornment;
begin
  $S := \{q^\epsilon\};\ \ T := \emptyset;$
  while $S \neq \emptyset$ **do**
    Choose an arbitrary predicate $p^\alpha \in S$;
    Include $\textit{adorn-pred}(p, \alpha)$ in P;
    $\textit{new-pred} = \{p^\alpha \mid p^\alpha \in \textit{body}(r)$ for rule $r \in \textit{adorn-pred}(p, \alpha)\} - T$;
    $S := S \cup \textit{new-pred}$;
    $S := S - \{p^\alpha\};\ \ T := T \cup \{p^\alpha\}$;
  end while
  $Q :=$ set of rules reachable from q^ϵ;
  return Q after replacing q^ϵ in Q by q;
end.

— **Figure 1.** *Propagation of Min Predicates* —

$$p^\alpha(X_1, ..., X_n) \leftarrow min(X_i, W, p(X_1, ..., X_n))$$

where X_i is the cost argument and W is the set of variables adorned as u in α is added to *adorn-completion*(P).

Lemma 4. *Let P be a set of rules with query predicate symbol q and a predicate symbol p reachable from q. Let $Q = P \cup \textit{adorn-pred}(p, \alpha)$. Then*

$$\textit{adorn-completion}(P) \equiv_q \textit{adorn-completion}(Q)$$

Proof. Follows from the construction of the adorned rules. □

The following lemma discusses the special case when the adornment for propagation is ϵ.

Lemma 5. *Let P be a set of rules with query predicate symbol q. Let Q be the set of rules obtained by determining the set of rules in $P \cup \textit{adorn-pred}(q, \epsilon)$ that are reachable from q^ϵ, and then replacing q^ϵ by q. Then*

$$\textit{adorn-completion}(P) \equiv_q \textit{adorn-completion}(Q)$$

Proof. Follows from the conventions for the ϵ adornment and the construction of adorned rules. □

Essentially, Lemma 2 states that query equivalence is preserved at the first iteration step of Algorithm 1 and Lemma 1 states that query equivalence is preserved at each subsequent iteration step of Algorithm 1. The following theorem formally states the query equivalence of the original and the adorned program.

Theorem 6. *Let $Q_1 = \langle q, P_1 \rangle$ be an aggregate stratified query and let $Q_2 = \langle q^\epsilon, P_2 \rangle$ be the query generated by Algorithm 1, then*

1. *$foe(P_2)$ has a total well-founded model, and*
2. *$Q_1 \equiv Q_2$ under the well-founded semantics.* □

Observe that $foe(P_1)$ in the above theorem is stratified and its perfect model coincide with the (total) well-founded model and with the unique total stable model. Moreover, $foe(P_2)$ has a total well-founded model which coincide with its unique total stable model. In the following example we present the completion of the adorned program at the various steps.

Example 7. Consider the program of Example 1 and the query $sh\_path(X, Y, C)$. In the first step the program P_1 consists of the rules in P plus the adorned rule;

$\mathtt{r_1}: \ \mathtt{sh\_path}^\epsilon(\mathtt{X,Y,C}) \leftarrow \mathtt{path}^{\mathtt{uum}}(\mathtt{X,Y,C}).$

In the adorn-completion of P_1 consists of P_1 plus the rule

$$path^{uum}(X, Y, C) \leftarrow min(C, (X, Y), path(X, Y, C)).$$

At the nest step, the algorithm generates the program P_2 ($adorn{-}completion(P_1)$) obtained by adding to P_1 the rules

$\mathtt{r_2}: \ \mathtt{cpath}^{\mathtt{uum}}(\mathtt{X,Y,C}) \leftarrow \mathtt{arc}^{\mathtt{uum}}(\mathtt{X,Y,C}).$
$\mathtt{r_3}: \ \mathtt{cpath}^{\mathtt{uum}}(\mathtt{X,Y,C}) \leftarrow \mathtt{path}^{\mathtt{uum}}(\mathtt{X,Z,C1}), \mathtt{arc}^{\mathtt{uum}}(\mathtt{Z,Y,C2}), \mathtt{C} = \mathtt{C1{+}C2}.$
$\mathtt{r_4}: \ \mathtt{path}^{\mathtt{uum}}(\mathtt{X,Y,C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X,Y}),\ \mathtt{cpath}^{\mathtt{uum}}(\mathtt{X,Y,C})).$

The program $adorn - completion(P_2)$ is obtained by adding to P_2 the rules

$$arc^{uum}(X, Y, C) \leftarrow min(C, (X, Y), arc(X, Y, C)).$$

At the third and final step, the algorithm generates the new program P_3 by adding to P_2 the rule

$\mathtt{r_5}: \ \mathtt{arc}^{\mathtt{uum}}(\mathtt{X,Y,C}) \leftarrow \mathtt{min}(\mathtt{C},(\mathtt{X,Y}), \mathtt{arc}(\mathtt{X,Y,C})).$

The adorn-completion of P_3 coincides with P_3 since all adorned predicates are defined. Observe that at each step the queries $(\mathtt{sh\text{-}path}^\epsilon(\mathtt{X,Y,C}), \mathtt{P_i})$ and $(\mathtt{sh\text{-}path}(\mathtt{X,Y,C}), \mathtt{P})$ are equivalent. □

4 Conclusion

In this paper, we have presented techniques for the propagation of extrema predicates into recursive logic programs. The main contribution consists of a precise algorithm for the propagation of *min* and *max* predicates into logic programs. The propagation of such meta-level predicates permits the efficient computation, by using a variation of the seminaive fixpoint, called seminaive greedy fixpoint [5], of logic programs with *min* and *max* predicates. In [5] it has been shown that the computation of the single source shortest path program of Example 1, after the propagation of the *min* predicates, has the same complexity of the celebrated Dijkstra's algorithm. The technique presented in the paper can be also extended to programs presenting different forms of monotonicity such as XY-stratification [22] and modular stratification [13].

References

1. K. Apt, H. Blair, and A. Walker. Towards a theory of declarative programming. In (*Foundations of Deductive Databases and Logic Programming* (Minker, ed.), 1988.
2. M.P. Consens and A.O. Mendelzon. Low complexity aggregation in graphlog and Datalog. Theoretical Computer Science, 1993.
3. S. W. Dietrich. Shortest Path by Approximation in Logic Programs. *ACM Letters on Programming Languages and Systems* Vol 1, No, 2, pages 119–137, June 1992,.
4. S. J. Finkelstein, N. Mattos, I.S. Mumick and H. Pirahesh. Expressing Recursive Queries in SQL. *ISO - IEC JTC1/SC21 WG3 DBL MCI Tec. Rep.*, March, 1996.
5. S. Ganguly, S. Greco, and C. Zaniolo. Extrema predicates in deductive databases. *J. Computer and System Science*, 1995.
6. M. Gelfond and V. Lifschitz. The stable model semantics of logic programming. In *Proc. of the Fifth Intern. Conf. on Logic Programming*, pages 1070–1080, 1988.
7. S. Greco, S. Ganguly, and C. Zaniolo. Greedy by Choice. *Proc. Eleventh ACM PODS Conf.*, pages 109–118, 1991.
8. S. Greco, D. Saccà, and C. Zaniolo. Dynamic Programming optimization for Logic Programs with Aggregates. *Proc. Int. Logic Progr. Symp.*, pages 109–118, 1993.
9. D. Kemp and P. Stuckey. Semantics of Logic Programs with Aggregates. In *Proc. of the 1991 Intern. Symposium on Logic Programming*, pages 387–341, 1991.
10. I.S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *Proc. 16th Conf. on Very Large Data Bases*, pages 264–277, 1990.
11. I.S. Mumick, and O. Shmueli. How Expressive is Stratified Aggregation. In *Annals of Mathematics and Artificial Intelligence*, 1995.
12. H. Przymusinska and T.C. Przymusinski. Weakly perfect model semantics for logic programs. In *Proc. Fifth Intern. Conf. on Logic Progr.*, pages 1106–1120, 1988.
13. K. Ross. A Procedural Semantics for the Well-Founded Negation in Logic Programs. In *Proc. ACM Symp. on Principles of Database Systems*, pages 22-33, 1989.
14. K. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Databases. *Proc. Eleventh ACM PODS Conf.*, 1992.
15. D. Saccà and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proc. of the Ninth ACM PODS Conf.*, pages 205–217, 1990.
16. S. Sudarshan and R. Ramakrishnan. Aggregation and relevance in deductive databases. In *Proc. of the 17th Conf. on Very Large Data Bases*, 1991.
17. R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL: Control, Relations and Logic. In *Proc. of Intl. Conf. on Very Large Data Bases*, 1992.
18. S. Sudarshan, D. Srivastava, R. Ramakrishnan and C. Beeri, Extending Well-Founded and Valid Semantics for Aggregation, In *Proc. of the Int. Logic Programming Symposium*, pages 591-608, 1992.
19. J. Ullman. *Principles of Data and Knowledge-Base Systems*, volume 1 and 2. Computer Science Press, New York, 1988.
20. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.
21. A. Van Gelder. Foundation of Aggregation in Deductive Databases In *Proceedings of the Third Int. Conf. on Deductive and O-O Databases*, Phoenix, AZ, 1993.
22. C. Zaniolo, N. Arni, and K. Ong. Negation and aggregates in recursive rules: the LDL++ approach, In *Proc. Int. Conf. od Deductive and Object Oriented Databases*, pp. 204-221, 1993.