Semantic Optimization: What are Disjunctive Residues Useful for?

Wolfgang L.J. Kowarschick

Research Group Knowledge Bases
Bavarian Research Center for Knowledge Based Systems (FORWISS)
Orleansstr. 34, W-8000 München, Germany
email: kowa@forwiss.tu-muenchen.de

Abstract

Residues have been proved to be a very important means for doing semantic optimization. In this paper we will discuss a new kind of residues - the disjunctive residues. It will be shown that they are very useful to perform subformula elimination, if, in addition, a powerful reduction algorithm is available.

1 Introduction

The notion of *Semantic Query Optimization* for relational databases was introduced in the early 80's by King [79, 81]; Hammer and Zdonic [80] independently developed a very similar optimization method. The key idea of King, as well as of Hammer and Zdonik, is that integrity constraints may not only be utilized to enforce consistency of a database but also to optimize user queries. During the last decade, many efforts were made to improve this technique and to generalize it for deductive databases (Chakravarthy et al. [84-90], Lee and Han [88], Shenoy and Ozsoyoglu [87, 89], Siegel [89], Kowarschick [91, 92a-c], etc., etc.). Especially the notion of residues, introduced by Chakravarthy et al. [84-90], was a great progress in this area.

In Kowarschick [91, 92a] Semantic Optimization was introduced formally as program transformation.[1] In particular, Chakravarthy's definition of residues was generalized (we call them conjunctive residues) and a new kind of residues, the disjunctive residues, was introduced. But since the application of a disjunctive residue always adds a new rule to a program, one question arises: Can disjunctive residues be utilized to "optimize" logic programs? Or, briefly: What are disjunctive residues useful for? The answer to this question is topic of this paper. It will be shown that disjunctive residues are mainly used to *delete* subformulas from rule bodies, although this seems to be a contradiction to the property stated above. Especially, it is much simpler to describe join elimination by means of disjunctive residues than by conjunctive residues, as, e.g., Chakravarthy et al. [90] did.

2 Notation and Definitions

The semantic optimization technique described in Kowarschick [91, 92a-c] is not restricted to untyped logic programs w.r.t. Herbrand pre-interpretation (Lloyd [87]) - it can be applied to any hierarchically typed logic program w.r.t. any pre-interpretation I^{pre}. We will not go in details here; the interested reader is referred to Kowarschick

[1] Note the use of the term *Semantic Optimization* instead of *Semantic Query Optimization*. We do this for two reasons: Firstly, many of the transformation steps can be performed fully independently from any query, and secondly, queries can be viewed as special rules, i.e., it makes no difference whether a "normal" program rule or a query is optimized.

[91, 92a]. In the following $\hat{L}$ shall be a hierarchically typed first order language with equality. In $\hat{L}$ four kinds of predicate identifiers will be distinguished: evaluable (e.g. "<", "is-a"), extensional, intensional, and auxiliary predicate identifiers. Furthermore, I^{pre} shall be an arbitrary pre-interpretation of $\hat{L}$ (e.g. the Herbrand pre-interpretation) and $\mathfrak{I}$ shall be the set of all I^{pre}-interpretations (e.g. the set of all Herbrand interpretations). Note that a pre-interpretation I^{pre} interprets all $\hat{L}$-identifiers except for the non-evaluable predicate identifiers; i.e., especially the interpretation of the evaluable predicate identifiers is already determined by I^{pre}. As usual, $\models$ denotes the standard logical consequence operator (Enderton [72], Kowarschick [92a]). But, we also need the operator $\models^{\mathfrak{I}}$, which only allows for I^{pre}-interpretations:

$$F \models^{\mathfrak{I}} G :\Leftrightarrow \bigwedge I \in \mathfrak{I}, v \in V(I): F \models_{I,v} G$$

(where F, G are sets of formulas and V(I) is the set of variable assignments w.r.t. I)

Instead of *program clauses*, we will use *program rules* to build up logic programs (see Lloyd [87], §17). A program rule $A \leftarrow B$ consists of an atom A, which is called *rule head*, and an arbitrary formula B, which is called rule body. Depending on the head predicate identifier, a program rule is termed extensional, intensional, or auxiliary (evaluable program rules will not be considered). A *fact* is an *extensional* program rule with empty rule body. A logic program is a finite set of rules, an extensional logic program (or a database) is a set of facts, an intensional logic program is a set of intensional and auxiliary rules, and a database logic program is a pair consisting of an extensional and an intensional logic program (observe that a database logic program $L_{db} = (F,G)$ can be viewed as standard logic program $L = F \cup G$). Since we use the perfect model semantics, we consider only (locally) stratifiable programs (Przymusinski [88]; note that it is possible to generalize the perfect model semantics for non-normal $\hat{L}$-programs, Kowarschick [91, 92c]).

Semantic optimization is commonly done by virtue of residues (Chakravarthy et al. [84-90], Kowarschick [92a, 91]). A residue can be added to or can replace some rule body without effecting the program semantics, i.e., through this transformation a "semantically equivalent" (Kowarschick [92a, 91]) program will be obtained. Usually, residues are computed by means of integrity constraints, but unfortunately, the following definition does not always supply residues, therefore we name them residue *candidates*.

Let G be a locally stratifiable intensional logic program, IC a set of integrity constraints (i.e., a set of closed formulas), IC_0 a subset of IC, $R = H \leftarrow B \in G$ a rule, and B′ an arbitrary formula.

B′ is a general IC_0-residue-candidate of $R \in G :\Leftrightarrow IC_0 \models^{\mathfrak{I}} B \leftrightarrow B'$

B′ is a conjunctive IC_0-residue-candidate of $R \in G :\Leftrightarrow B \wedge B'$ is a general IC_0-residue-candidate of $R \in G$ ($\Leftrightarrow IC_0 \models^{\mathfrak{I}} B \leftrightarrow B \wedge B' \Leftrightarrow IC_0 \models^{\mathfrak{I}} B \rightarrow B'$)

B′ is a disjunctive IC_0-residue-candidate of $R \in G :\Leftrightarrow B \vee B'$ is a general IC_0-residue-candidate of $R \in G$ ($\Leftrightarrow IC_0 \models^{\mathfrak{I}} B \leftrightarrow B \vee B' \Leftrightarrow IC_0 \models^{\mathfrak{I}} B \leftarrow B'$)

Basically, a resiue candidate is a residue, if it does not effect the "satisfaction" of the integrity constraints that were used to compute the residue candidate. For more details, see Kowarschick [92a, 91].

3 Disjunctive Residue Candidates

To compute residue candidates we need a simple lemma.

Lemma 1

Let C←D be a formula, which is a logical $\mathfrak{I}$-consequence of a set IC_0 of integrity constraints: $IC_0 \models^{\mathfrak{I}} C \leftarrow D$. Moreover, let R= H←B be a rule, S an arbitrary formula, and σ a substitution.

If $IC_0 \models^{\mathfrak{I}} S \rightarrow (B \rightarrow D\sigma)$, then $S \rightarrow C\sigma$ is a conjunctive IC_0-residue-candidate of R.

If $IC_0 \models^{\mathfrak{I}} S \rightarrow (C\sigma \rightarrow B)$, then $S \wedge D\sigma$ is a disjunctive IC_0-residue-candidate of R.

Proof

Straightforward (Kowarschick [92a]). □

Lemma 1 can be utilized to develop algorithms for computing residue candidates (a very non-deterministic algorithm is described in Kowarschick [92a]). But here, we think it more intuitive to explain the use of this lemma by virtue of an example.

Example 1

```
extpred   a: ℕ; b: ℕ,ℕ
intpred   p: ℕ,ℕ
var       x,y,z: ℕ

R1:  p(x,y)←a(x,y),¬b(x,y)
ic1: b(u,u)←
ic2: ¬b(u,v)←u>v
```

Observe, without loss of generality we can assume that the free variables of the integrity constraints and of the rules are disjoint. And we should keep them disjoint, since, otherwise, we might be unable to compute all residue candidates.

The integrity constraint ic_1 yields a conjunctive residue candidate (which, in fact, is a residue, since ic_1 consists only of evaluable and extensional predicates):

$$b(u,u) \models^{\mathfrak{I}} u \neq v \leftarrow \neg b(u,v)$$
$$ic_1 \models^{\mathfrak{I}} C \leftarrow D$$

$$\models^{\mathfrak{I}} T \rightarrow (a(x,y) \wedge \neg b(x,y) \rightarrow \neg b(x,y))$$
$$\models^{\mathfrak{I}} S \rightarrow (\text{body}(R_1) \rightarrow D\{u/x, v/y\})$$

As a consequence of lemma 1, $x \neq y$ ($\models \exists S \rightarrow C\{u/x, v/y\}$) is a conjunctive residue candidate of R_1. The second integrity constraint, on the other hand, yields a disjunctive residue candidate:

$$ic_2 = \neg b(u,v) \leftarrow u > v$$
$$ic_2 \models^{\mathfrak{I}} C \leftarrow D$$

$$\models^{\mathfrak{I}} a(x,y) \rightarrow (\neg b(x,y) \rightarrow a(x,y) \wedge \neg b(x,y))$$
$$\models^{\mathfrak{I}} S \rightarrow (C\{u/x, v/y\} \rightarrow \text{body}(R_1))$$

Therefore, $a(x,y) \wedge x > y$ ($= S \wedge D\{u/x, v/y\}$) is a disjunctive residue candidate, which again is even a disjunctive residue. □

As usual, the conjunctive residue $x \neq y$ can be utilized to answer queries such as p(x,x) or p(5,5) without any access to the database. But, also the disjunctive residue may be used to accelerate the computation of particular answers, namely of answers to queries where the first attribute is greater than the second (such as p(7,5) or p(x+3,x), when $\mathfrak{I}$ interprets + as customary addition!). In the sequel we will study in more detail how this can be done.

At first sight it seems as if the application of a disjunctive residue makes the computation of the perfect model more expensive, since, in principle, in this case a new rule is added to the program (both programs $L \cup \{H \leftarrow B \vee Residue\}$ and $L \cup \{H \leftarrow B, H \leftarrow Residue\}$ have the same perfect model - we call them pm-equivalent (see Kowarschick [92a])). But in special cases, a disjunctive residue is, in fact, a general residue, and then the original rule body can be replaced by this residue.

It can be proved that every general Ø-residue-candidate is even a general residue, if the accompanying logic program remains to be locally straitifiable when this residue candidate is added (Kowarschick [91,92a]).

We call such residues *tautological residues*. Tautological residues can always be utilized to reduce, i.e. to simplify rule bodies, since they do not change the program semantics. And rule bodies should be reduced, because in most cases the computation of a reduced rule body is cheaper than the computation of the original rule body.

Reduction Algorithm

(compare to algorithm 3 of Chakravarthy et al. [88])

Let F be an arbitrary formula. Apply the following transformation rules to subformulas of F until no more transformation rules are applicable. The resulting formula reduc(F) is a tautological residue for every rule R with rule body F.

A transformation rule $A_1,...,A_n \rightarrow B$ is to be interpreted as follows: If A_1 or ... or A_n is a subformula of F replace it by B.

There are some trivial reductions which should be done always:

$T \wedge A,\ A \wedge T,\ A \wedge A,\ A \vee A \rightarrow A$
$\bot \wedge A,\ A \wedge \bot,\ A \wedge \neg A \rightarrow \bot$
$T \vee A,\ A \vee T,\ A \vee \neg A \rightarrow T$
⋮
$\neg\neg A \rightarrow A$
$(A \wedge B) \vee B,\ (A \vee B) \wedge B \rightarrow B$
$(A \wedge B) \vee (A \wedge C) \rightarrow A \wedge (B \vee C)$
$(A \vee B) \wedge (A \vee C) \rightarrow A \vee (B \wedge C)$
⋮
$\forall x A,\ \exists x A \rightarrow A$
when x does not occur free in A
$\exists x(A \vee B) \rightarrow (\exists x A) \vee (\exists x A)$
$\forall x(A \wedge B) \rightarrow (\forall x B) \vee (\forall x B)$
⋮
$\exists x(x=t \wedge ...x...x...) \rightarrow ...t...t...$
$\forall x(x=t \rightarrow ...x...x...) \rightarrow ...t...t...$
$\exists x(x=t) \rightarrow T$
$\forall x(x \neq t) \rightarrow \bot$
when x does not occur free in t

But, the special properties of the pre-interpretation I^{pre} should be taken account of, too. For example:

$5<3 \rightarrow \bot,\ 3<5 \rightarrow T,\ x=x \rightarrow T$
$x+3>x \rightarrow T$ (if type-of(x) = ℕ,...)
⋮
$x<y \wedge x>y \rightarrow \bot,\ x<y \vee x=y \vee x>y \rightarrow T$
⋮ □

Now we are capable of utilizing the disjunctive residue $a(x,y) \wedge x>y$ of rule R_1 (Example 1). Suppose, the question $p(z+3,z)$ shall be answered. This can be done by computing the perfect model of program L_1:

L_1: Q: answer(z) ← p(z+3,z)
R_1: p(x,y) ← a(x,y) ∧ ¬b(x,y)

Since $a(x,y) \wedge x>y$ is a disjunctive residue of R_1, we can also compute the perfect model of L_2 to answer the query:

L_2: Q: answer(z) ← p(z+3,z)
R_1': p(x,y) ←
(a(x,y) ∧ ¬b(x,y)) ∨
(a(x,y) ∧ x>y)

This program can be optimized by the standard optimization technique "perform selections as early as possible" (see e.g. Ullman [89]):

L_3: Q: answer(z) ← p(z+3,z)
R_1'': p(z+3,z) ←
(a(z+3,z) ∧ ¬b(z+3,z)) ∨
(a(z+3,z) ∧ z+3>z)

Move formally, we can view the predicate p as auxiliary predicate, since the user is only interested in the predicate answer. It can be proved that the perfect models of L_2 and L_3 differ at most in auxiliary predicates (viz. p) - we therefore call them weak pm-equivalent (Kowarschick [92a]).

Finally, we can use the reduction algorithm to simplify the body of R_1'':

L_4: Q: answer(z)←p(z+3,z).
R_1''':p(z+3,z)←a(z+3,z).

That is, if x is greater than y, then the disjunctive residue a(x,y)∧x>y can be utilized to delete the subformula ¬b(x,y) from the body of R_1. Now we are able to answer the introductory question "What are disjunctive residues useful for?": They are useful for deleting superfluous subformulas of rule bodies; but beside the disjunctive residues, we need a powerful reduction algorithm, too. Note that the presence of an answer rule is not essential for the application of residues, they can also be used to delete redundancy from rules. The fact, that disjunctive residues are more qualified to delete subformulas from rule bodies than conjunctive residues, can be confirmed by applying our method to two examples given by Chacravarthy et al. [90].

Example 2

(Chakravarthy et al. [90], Section 5.1, Q9)

```
extpred
    emp: String,N,String
         (Name,Salary,Department)
    emp-car: N,String,String
             (Tag#,Owner,Color)
```

Every employee car is owned by an employee:

ic_1: emp-car(t,o,c)
→∃s,d emp(o,s,d)

Find all employees who own a red car:

Q: answer(o)←
emp(o,s,d)∧emp-car(t,o,c)

Since the variables s, d, t, and c are not used in the query head, they can be existentially qualified in the query body - and this should be done always in case of semantic optimization (at least, if the accompanying integrity constraints contain existential qualifiers), since a formula with existential qualifiers is weaker than without ($F \models \exists x F$):

Q′: answer(o)←
(∃s,d emp(o,s,d))∧
emp-car(t,o,c)
(Q″: answer(o)←
(∃s,d emp(o,s,d))∧
(∃t,c emp-car(t,o,c))
will do as well)

For Q′ we compute the residue candidate emp-car(t,o,c)∧emp-car(t,o,c), which can be reduced to emp-car(t,o,c). (For Q″ we get (∃t,c emp-car(t,o,c))∧emp-car(t,o,c), which can be reduced to emp-car(t,o,c), likewise.) This residue candidate can be added to the query Q′:

Q‴: answer(o)←
((∃s,d emp(o,s,d))∧
emp-car(t,o,c))
∨ emp-car(t,o,c)

If we reduce the rule body of Q‴, we obtain the same result as Chakravarthy et al. [90], namely that the relation emp can be deleted from our query:

Q⁗: answer(o)←emp-car(t,o,c) □

This example suggests that our method is at least as powerful as that of Chakravarthy et al. [90]. But it is even more powerful, since join elimination or, more generally, subformula elimination has not to be handled in any other way than subformula introduction. This especially means that we, in contrast to Chakravarthy et al. [90], have not to restrict the class of integrity constraints in case of subformula elimination.

Example 3

(Chakravarthy et al. [90], Section 7.4, Example 12)

```
extpred
    parent:  person,person
    sibling: person,person
```
ic_1: sibling(x,y)←sibling(y,x)

Note that Chakravarthy et al. call this integrity constraint "recursive," but what is a "recursive" formula? A formula always may be transformed into a "non-recursive" form (e.g., ⊥←¬sibling(x,y),sibling(y,x)) only in case of *program rules* such a transformation will be forbidden, since it would change the program semantics (namely the perfect model).

Q: answer(r)←
    parent(r,s)∧
    sibling(s,t)∧sibling(t,s)

If this query is optimized by Chakravarthy's method, both atoms sibling(s,t) and sibling(t,s) may be deleted from the rule body, what, indeed, is wrong. But with our method this problem does not occur. The integrity constraint ic_1 yields the following disjunctive residues:

Res_1:=parent(r,s)∧sibling(s,t)∧
    sibling(s,t)
Res_2:=parent(r,s)∧sibling(t,s)∧
    sibling(t,s)

They can be reduced to:

Res_1:=parent(r,s)∧sibling(s,t)
Res_2:=parent(r,s)∧sibling(t,s)

These conjunctive residues can be applied separately or jointly to the query, in all cases an equivalent query will be the result:

reduct(query-body∨Res_1) = Res_1
  = parent(r,s)∧sibling(s,t)
reduct(query-body∨Res_2) = Res_2
  = parent(r,s)∧sibling(t,s)
reduct(query-body∨Res_1∨Res_2)
  = parent(r,s)∧
    (sibling(s,t)∨sibling(t,s)) □

4 Conclusion

In this paper we have introduced the notion of disjunctive residues which were demonstrated to be very useful for subformula elimination. But to do so, we need a powerful reduction algorithm, because, otherwise, a disjunctive residue would be nothing else than an additional program rule which only increases the computation costs. We believe that disjunctive residues (together with a reduction algorithm) are much more natural to describe subformula elimination than conjunctive residues. As a next step, we will implement a prototype of a semantic optimizer to study the relationship of conjunctive and disjunctive residues in more detail.

References

U.S. Chakravarthy, D.H. Fishman, and J. Minker [84]. Semantics Usage and Query Optimization in Deductive Databases. Technical Report 1413, University of Maryland, 1984.

U.S. Chakravarthy [85]. Semantic Query Optimization in Deductive Databases (Vol. I and II). Ph.D. Thesis, University of Maryland, 1985.

U.S. Chakravarthy, D.H. Fishman, and J. Minker [86]. Semantic Query Optimization in Expert Systems and Database Systems. In L. Kerschberg (ed.), Expert Database Systems, Proceedings from the First International Workshop, pp. 659-674, California, 1986.

U.S. Chakravarthy, J. Grant, and J. Minker [88]. Foundations of Semantic Query Optimization for Deductive Databases. In J. Minker (ed.), Foundations of Deductive Databases and Logic Programming, pp. 243-273, California, 1988.

U.S. Chakravarthy, J. Grant, and J. Minker [90]. Logic-Based Approach to Semantic Query Optimization. In ACM TODS, Vol. 15, No. 2, pp. 162-207, 1990.

H.B. Enderton [72]. A Mathematical Introduction to Logic. Academic Press, California, 1972.

M.M. Hammer and S.B. Zdonik Jr. [80]. Knowledge-Based Query Processing. In Proc. 6th VLDB, pp. 137-147, Montreal, 1980.

J.J. King [79]. Exploring the Use of Domain Knowledge for Query Processing Efficiency. Technical Report HPP-79-30, Stanford University, 1979.

J.J. King [81]. Query Optimization by Semantic Reasoning. Ph.D. Thesis, Stanford University, 1981. (also Michigan, 1984)

W.L.J. Kowarschick [91]. Semantische Optimierung rekursiver, insbesondere Δ-transformierter Logikprogramme. Doctoral dissertation, University of Technology Munich, 1991.

W.L.J. Kowarschick [92a]. Semantic Optimization: A Theoretical Approach. Internal report, FORWISS, Munich, 1992.

W.L.J. Kowarschick [92b]. Semantic Optimization of Recursive Logic Programs. Internal report, FORWISS, Munich, 1992.

W.L.J. Kowarschick [92c]. The Semantics of Δ-Transformed Logic Programs. Internal report, FORWISS, Munich, 1992.

S. Lee and J. Han [88]. Semantic Query Optimization in Recursive Databases. In Proc. 4th International Conference on Data Engineering, pp. 444-451, 1988.

J.W. Lloyd [87]. Foundations of Logic Programming. Springer, Berlin, 1987.

T.C. Przymusinski [88]. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker (ed.), Foundations of Deductive Databases and Logic Programming, pp. 193-216, California, 1988.

S.T. Shenoy and Z.M. Ozsoyoglu [87]. A System for Semantic Query Optimization. In Proc. ACM-SIGMOD, pp. 182-195, 1987.

S.T. Shenoy and Z.M. Ozsoyoglu [89]. Design and Implementation of a Semantic Query Optimizer. In IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 3, pp. 344-361, 1989.

M.D. Siegel [89]. Automatic Rule Derivation for Semantic Query Optimization. In L. Kerschberg (ed.), Expert Database Systems, pp. 669-698, 1989.

J.D. Ullman [89]. Principles of Database and Knowledge-Base Systems, Vol. 2, Computer Science Press, 1989.