

# Range Restriction for General Formulas

Stefan Brass

Martin-Luther-Universität Halle-Wittenberg  
Germany

# Motivation

- **Deductive databases:** seldom used in practice (yet), but important idea, great potential for future:
  - ◇ **Declarative programming:** very successful in SQL.
  - ◇ Real applications are not only SQL queries.
  - ◇ Program code around the queries is today written in Java, PHP, etc. (non-declarative).
  - ◇ In modern object-relational DBMS, SQL also calls program code written in PL/SQL or Java.
  - ◇ Interfaces not smooth, optimization difficult.
  - ◇ OODB or OO Framework: not declarative.

## Long-Term Goal

- Development of a deductive database system that
  - ◇ supports both: (some form of) Datalog, and SQL
  - ◇ permits declarative programming of typical database application tasks,
  - ◇ permits declarative programming of a significant percentage of its own code,
  - ◇ is reasonably efficient.
- SQL support needs general formulas in rule bodies. Lloyd-Topor-Transformation is not enough:  
e.g. semantics of duplicates, performance problems.

# Built-in Predicates

- Predicates in deductive databases are classified as
  - ◇ **EDB-predicates**: stored relations, defined extensionally by enumerating tuples (facts).
  - ◇ **IDB-predicates**: defined by rules, generalize views in classical databases (and used in query).
  - ◇ **Built-in predicates**: defined by program code in the DBMS.
- E.g.  $<$  is a built-in predicate: Infinite extension, can be called only with both arguments bound.

# Binding Patterns (1)

- A **binding pattern** specifies for each argument of a predicate whether the argument
  - ◇ must be “**bound**”, i.e. a given value (input), or
  - ◇ can be “**free**”, i.e. a variable (output).
- E.g.  $<$  supports only the binding pattern **bb**.
- A predicate can support several binding patterns, e.g.  $\text{sum}(X, Y, Z)$  meaning  $X + Y = Z$  supports **bbf**, **bfb**, **fbf** (and **bbb**: special case of other variants).
- EDB-predicates support **ff...f** (full table scan).

## Binding Patterns (2)

- Suppose that for each predicate  $p$ , a set  $bp(p) \neq \emptyset$  of allowed binding patterns are given.
- Interpretation  $\mathcal{I}$  is allowed iff for every predicate  $p$  and binding pattern  $\beta \in bp(p)$  the following holds:
  - ◇ Let  $n$  be the arity of  $p$  and  $1 \leq i_1 < \dots < i_k \leq n$  be the index positions with  $\beta(i_j) = \mathbf{b}$ .
  - ◇ Then for all values  $c_1, \dots, c_k$  from the domain of  $\mathcal{I}$ , the following set is finite and computable:

$$\{(d_1, \dots, d_n) \in \mathcal{I}[p] \mid d_{i_1} = c_1, \dots, d_{i_k} = c_k\}$$

# Task

- In Prolog: “instantiation fault” at runtime, when the binding restrictions are violated.
- For IDB-predicates, permitted binding patterns can be declared (mode declarations) or partially derived.
- The task now is to check the rules at compile-time, whether they are executable without violating binding restrictions for given arguments in the head.
- With automatic reordering of the body-literals.

When a predicate can be called with different input/output-args, the user cannot always write the body literals in an executable sequence.

## Example

- Reordering is not so simple for arbitrary formulas (complex tree-structures).

- Consider the condition

$$p(X, Y) \wedge (q(Y, Z) \wedge r(X))$$

and the binding restrictions:

- ◇  $p$  and  $q$  support only **bf** (1st arg. must be bound),
  - ◇  $r$  supports **f** (no restriction).
- The only possible evaluation sequence is

$$r(X), p(X, Y), q(Y, Z).$$

# Generalized Binding Pattern

- Binding patterns must be generalized from predicates to formulas: Arguments in predicates naturally correspond to free variables in formulas.
- A **generalized binding pattern (GBP)** for a formula  $\varphi$  is a pair of sets of variables (free in  $\varphi$ ), written

$$X_1, \dots, X_n \longrightarrow Y_1, \dots, Y_m.$$

- It means that when we have values  $d_i$  for the  $X_i$ , there can be only finitely many values for the  $Y_i$  and we can actually compute a finite upper bound (candidate values).

# Finiteness Dependencies

- Something very similar to generalized binding patterns has been used in the literature several times under the name “Finiteness Dependency”.

With several slightly different definitions of the semantics.

- E.g. it has been proven that the Armstrong Axioms, known for functional dependencies, are also sound and complete for finiteness dependencies.
  - ◊ If  $\mathcal{X} \subset \mathcal{Y}$ , then  $\mathcal{X} \longrightarrow \mathcal{Y}$ .
  - ◊ If  $\mathcal{X} \longrightarrow \mathcal{Y}$ , then  $\mathcal{X} \cup \mathcal{Z} \longrightarrow \mathcal{Y} \cup \mathcal{Z}$ .
  - ◊ If  $\mathcal{X} \longrightarrow \mathcal{Y}$  and  $\mathcal{Y} \longrightarrow \mathcal{Z}$ , then  $\mathcal{X} \longrightarrow \mathcal{Z}$ .

# Semantics: Valid GBP

- A GBP  $X_1, \dots, X_n \longrightarrow Y_1, \dots, Y_m$  for a formula  $\varphi$  is valid iff

- for every allowed interpretation  $\mathcal{I}$  and all values  $d_1, \dots, d_n$  from the domain of  $\mathcal{I}$  the set

$$\{(\mathcal{A}(Y_1), \dots, \mathcal{A}(Y_m)) \mid \langle \mathcal{I}, \mathcal{A} \rangle \models \varphi, \\ \mathcal{A}(X_1) = d_1, \dots, \mathcal{A}(X_n) = d_n\}$$

is finite and a finite superset of this set is effectively computable.

# Example (1)

- Consider again

$$p(X, Y) \wedge (q(Y, Z) \wedge r(X))$$

with the binding restrictions  $p:bf$ ,  $q:bf$ ,  $r:f$ .

- We now compute bottom-up sets of valid binding patterns  $gbp^+(\varphi)$  for every subformula  $\varphi$ .

There is also a function  $gbp^-$  which works in negated context, with true and false inverted. Note that  $gbp^+(\varphi)$  are not all valid GBPs. E.g. for  $X = 5 \wedge \neg(X = 5)$ , the GBP  $\longrightarrow X$  is valid, but not in  $gbp^+$ .

- For the subformula  $r(X)$ , the GBP  $\longrightarrow X$  is valid: Because the interpretation is allowed with  $\mathbf{f} \in bp(\mathbf{r})$ , there can be only finitely many values for  $X$ .

## Example (2)

- For the subformula  $q(Y, Z)$ , we get  $Y \longrightarrow Z$  from the binding pattern  $\mathbf{bf}$  for  $q$ .
- Now conjunction simply takes the union of the binding patterns computed for the subformulas, plus a closure operation for computing derived GBPs.
- E.g.  $\longrightarrow X$  is still valid for  $q(Y, Z) \wedge r(X)$ : Without having a value for  $Y$ , we cannot determine the exact  $X$ , for which the formula is satisfied, but we know a finite superset of candidate values.

## Example (3)

- Of course,  $p(X, Y)$  with  $p:bf$  gives  $X \longrightarrow Y$ .

- The closure operation derives from

$$\longrightarrow X \quad \text{and} \quad X \longrightarrow Y \quad \text{and} \quad Y \longrightarrow Z$$

the GBP  $\longrightarrow X, Y, Z$ .

Using transitivity and augmentation.

- Thus, we can compute a candidate set of  $(X, Y, Z)$  values, for which the formula might be true.

It is certainly false for all other values.

- A second step checks these variable assignments, whether the formula is indeed true.

## Example (4)

- If we know values for all variables, we can check whether an atomic formula is true or false.

It follows from the definition of “allowed interpretation” that the containment of a tuple in the extension of a predicate can be decided.

- Also formulas composed by propositional connectives  $\wedge$ ,  $\vee$ ,  $\neg$  are no problem.
- The interesting case are subformulas with quantifiers  $\forall$ ,  $\exists$ , because then we need a finite set of values for the quantified variable, which must be tried.

# Range-Restricted Rule

- A rule  $A \leftarrow \varphi$  is **range restricted** for a binding pattern  $\beta$  for  $A$  iff  $\mathcal{X} \longrightarrow \mathcal{Y} \in gbp^+(\varphi)$  where
  - ◇  $\mathcal{X} := input(A, \beta)$  (variables occurring in bound arguments in the head)
  - ◇  $\mathcal{Y} := free(p(t_1, \dots, t_n) \leftarrow \varphi)$  (all variables in the rule except quantified ones).
- for every subformula  $\exists Z: \varphi_0$  occurring in positive (unnegated) context:
 
$$(free(\varphi_0) - \{Z\}) \longrightarrow Z \in gbp^+(\varphi_0)$$
- (similar conditions for  $\forall$ , negated context)

## Example: Quantifier (1)

- Consider the formula

$$p(X, Y) \wedge \exists Z: (q(Y, Z) \wedge r(X))$$

- The quantified subformula produces candidate values for  $X$  according to the GBP  $\longrightarrow X$ .

Note that it is not possible to check whether the existential quantifier is indeed satisfied without having a value for  $Y$ . Nevertheless, we know that the existential condition is false if  $X$  has a value outside the finite set of values. This is all that is needed at the moment.

- Then  $p(X, Y)$  gives candidate values for  $(X, Y)$ .
- In the second phase, the subformulas are tested whether they are indeed true (see next slide).

## Example: Quantifier (2)

- The requirement for the existentially quantified formula is that given values for  $x$  and  $y$ , it must be possible to determine a finite set of candidate values for  $z$ , i.e.  $x, y \longrightarrow z$ .

- In the example

$$p(x, y) \wedge \exists z: (q(y, z) \wedge r(x))$$

this is possible because of the subformula  $q(y, z)$  with the binding requirement  $q:bf$ .

- Given a finite set of possible values for  $z$ , each can be tried.

## A Possible Extension

- According to the above definition, the rule

$$p(X) \leftarrow q(X) \vee r(X, Y).$$

is not range-restricted, because  $Y$  is not (always) bound to a finite set when the rule body is true.

- The user could write an explicit quantifier:

$$p(X) \leftarrow q(X) \vee \exists Y: r(X, Y).$$

- But one could define range-restriction in a way that makes the original formula permissible.
- Having to check only a finite set of values is not the same as being true for only a finite set of values!

# Conclusion

- Deductive databases basically use an iteration of the  $T_P$ -operator to compute the minimal model.
- While the iteration might not terminate, at least each single application of the  $T_P$ -operator must be effectively computable.
- This is not trivial for general formulas and built-in predicates with different binding restrictions.
- The methods used in this paper can be used as compile-time check and as basis for evaluation at runtime (with many optimization possibilities).