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.
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. sum(X, Y, Z) meaning X + Y = Z supports bbf, bfb, fbb (and bbb: special case of other variants).

EDB-predicates support ff…f (full table scan).
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 $1 \leq i_1 < \cdots < i_k \leq n$ be the index positions with $\beta(i_j) = b$.
- Then for all values $c_1, \ldots, c_k$ from the domain of $\mathcal{I}$, the following set is finite and computable:

$$\{(d_1, \ldots, d_n) \in \mathcal{I}[p] \mid d_{i_1} = c_1, \ldots, d_{i_k} = c_k\}$$
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
  \[ p(X, Y) \land (q(Y, Z) \land r(X)) \]
  and the binding restrictions:
  - \( p \) and \( q \) support only \( \text{bf} \) (1st arg. must be bound),
  - \( r \) supports \( \text{f} \) (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 $X_1, \ldots, X_n \rightarrow Y_1, \ldots, Y_m$.

- 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 $X \subseteq Y$, then $X \rightarrow Y$.
  - If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$.
  - If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
Semantics: Valid GBP

- A GBP $X_1, \ldots, X_n \to Y_1, \ldots, Y_m$ for a formula $\varphi$ is valid iff
- for every allowed interpretation $\mathcal{I}$ and all values $d_1, \ldots, d_n$ from the domain of $\mathcal{I}$ the set

\[ \{(A(Y_1), \ldots, A(Y_m)) \mid \langle \mathcal{I}, A \rangle \models \varphi, \quad A(X_1) = d_1, \ldots, A(X_n) = d_n \} \]

is finite and a finite superset of this set is effectively computable.
Example (1)

- Consider again
  \[ p(X, Y) \land (q(Y, Z) \land r(X)) \]
  with the binding restrictions \textit{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 \( X = 5 \land \neg(X = 5) \), the GBP \( \rightarrow X \) is valid, but not in \( gbp^+ \).

- For the subformula \( r(X) \), the GBP \( \rightarrow X \) is valid: Because the interpretation is allowed with \( f \in bp(r) \), there can be only finitely many values for \( x \).
Example (2)

• For the subformula $q(Y, Z)$, we get $Y \rightarrow Z$ from the binding pattern $bf$ 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. $\rightarrow X$ is still valid for $q(Y, Z) \land r(X)$: 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 $p:bfg$ gives $X \rightarrow Y$.
- The closure operation derives from $\rightarrow X$ and $X \rightarrow Y$ and $Y \rightarrow Z$ the GBP $\rightarrow X, Y, Z$.
  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.
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 $\land, \lor, \neg$ 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.
A rule $A \leftarrow \varphi$ is range restricted for a binding pattern $\beta$ for $A$ iff $\mathcal{X} \rightarrow \mathcal{Y} \in gbp^+(\varphi)$ where

- $\mathcal{X} := \text{input}(A, \beta)$ (variables occurring in bound arguments in the head)
- $\mathcal{Y} := \text{free}(p(t_1, \ldots, t_n) \leftarrow \varphi)$ (all variables in the rule except quantified ones).

For every subformula $\exists Z : \varphi_0$ occurring in positive (unnegated) context:

$$(\text{free}(\varphi_0) - \{Z\}) \rightarrow Z \in gbp^+(\varphi_0)$$

(Similar conditions for $\forall$, negated context)
Example: Quantifier (1)

- Consider the formula
  
  \[ p(X, Y) \land \exists Z: (q(Y, Z) \land r(X)) \]

- The quantified subformula produces candidate values for \( X \) according to the GBP \( \rightarrow X \).

  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. $X, Y \rightarrow Z$.

- In the example

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

  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
  \[ p(X) \leftarrow q(X) \lor r(X, Y). \]
  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:
  \[ p(X) \leftarrow q(X) \lor \exists Y: r(X, Y). \]
- 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).