Fundamentals of Physical Design
Query Processing: First-Order Queries

David Toman

D. R. Cheriton School of Computer Science
University of Waterloo

. . . the Story so Far:

1. Integrity constraints and *simple index declarations* capture a variety of logical-to-physical mappings and physical designs: *physical data independence*,
2. Most of *physical design* issues (including appropriate costs) can be captured in such a framework, and
3. *Conjunctive queries* can be compiled to low-level query plans.

$\Rightarrow$ complex queries: "select block at a time" approach.

Shortcomings

1. General first-order queries: not satisfactory
2. Complex schema: what to do with negations et al.?

   ⇒ additional "rewriting rules"? (when is it enough?)
3. Completeness (can we find a rewriting if one exists)?

   ⇒ conjunctive query over conjunctive materialized views

   ... may need *negation* in the plan.

Non-conjunctive Rewriting for CQ over CQ views

- Query:

$$Q(x, y) \equiv \exists z, v, u.R(z, x), R(z, v), R(v, u), R(u, y)$$

- Views (with "`index Vi () (x,y)`"):

$$\begin{aligned} V_1(x, y) &\equiv \exists z, v.R(z, x), R(z, v), R(v, y) \\ V_2(x, y) &\equiv \exists z.R(x, z), R(z, y) \\ V_3(x, y) &\equiv \exists z, v.R(x, z), R(z, v), R(v, y) \end{aligned}$$

- Rewriting (and a plan, assuming indices for the views):

$$\exists z.V_1(x, z) \land \forall v.(V_2(v, z) \to V_3(v, y))$$

... and there isn't an equivalent conjunctive query.

Non-conjunctive Rewriting for CQ over CQ views

- Query:

$$Q(x, y) \equiv \exists z, v, u.R(z, x), R(z, v), R(v, u), R(u, y)$$

- Views (with "`index Vi () (x,y)`"):

$$\begin{aligned} V_1(x, y) &\equiv \exists z, v.R(z, x), R(z, v), R(v, y) \\ V_2(x, y) &\equiv \exists z.R(x, z), R(z, y) \\ V_3(x, y) &\equiv \exists z, v.R(x, z), R(z, v), R(v, y) \end{aligned}$$

- Rewriting (and a plan, assuming indices for the views):

$$\exists z.V_1(x, z) \wedge \forall v.(V_2(v, z) \rightarrow V_3(v, y))$$

... and there isn't an equivalent *conjunctive query*.

Non-conjunctive Rewriting for CQ over CQ views

- Query:

$$Q(x, y) \equiv \exists z, v, u.R(z, x), R(z, v), R(v, u), R(u, y)$$

- Views (with "`index Vi () (x,y)`"):

$$\begin{aligned} V_1(x, y) &\equiv \exists z, v.R(z, x), R(z, v), R(v, y) \\ V_2(x, y) &\equiv \exists z.R(x, z), R(z, y) \\ V_3(x, y) &\equiv \exists z, v.R(x, z), R(z, v), R(v, y) \end{aligned}$$

- Rewriting (and a plan, assuming indices for the views):

$$\exists z.V_1(x, z) \wedge \forall v.(V_2(v, z) \rightarrow V_3(v, y))$$

... and there isn't an equivalent *conjunctive query*.

First-order (FO) Query Language

Syntax:

`Q ::= A v`	class access
`| v.Pf1 = u.Pf2`	equation
`| true`	singleton
`| from Q1,Q2`	natural join
`| elim v1,...,vk Q`	selection (distinct)
`| empty v1,...,vk`	empty set
`| Q1 union Q2`	union (union-compatible)
`| Q1 minus Q2`	set difference (union-compatible)

... essentially an alternative syntax for First-order Formulæ

... restricted to *domain-independent* queries.

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?
2. Can we *test* for this?
3. Does it mean we have a *rewriting* (a plan)?

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?

Definition (Implicit Definability)

A query Q is *implicitly definable in A_js* if $Q(M_1) = Q(M_2)$
for all $M_1 \models \mathcal{T}$ and $M_2 \models \mathcal{T}$ two databases such that
`(select v.`V_j `(`A_j `v))`(M_1) = `(select v.`V_j `(`A_j `v))`(M_2).

2. Can we *test* for this?

3. Does it mean we have a *rewriting* (a plan)?

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?

Definition (Implicit Definability)

A query Q is *implicitly definable in A_js* if $Q(M_1) = Q(M_2)$ for all $M_1 \models \mathcal{T}$ and $M_2 \models \mathcal{T}$ two databases such that
`(select v.`V_j `(`A_j `v))`(M_1) = `(select v.`V_j `(`A_j `v))`(M_2).

2. Can we *test* for this?

3. Does it mean we have a *rewriting* (a plan)?

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?

Definition (Implicit Definability)

A query Q is *implicitly definable in A_js* if $Q(M_1) = Q(M_2)$ for all $M_1 \models \mathcal{T}$ and $M_2 \models \mathcal{T}$ two databases such that
`(select v.`V_j` (`A_j` v))`(M_1) = `(select v.`V_j` (`A_j` v))`(M_2).

2. Can we *test* for this? YES:

$$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^*, \text{ where } (.)^* \text{ renames all symbols but } A_j \& V_j.$$

3. Does it mean we have a *rewriting* (a plan)?

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?
   Implicit Definability: Q is *determined by the data only*
2. Can we *test* for this? YES:
   $\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^*$, where $(.)^*$ renames all symbols but $A_j \& V_j$.
3. Does it mean we have a *rewriting* (a plan)?

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it meant for a query to be *defined* by the *data*?
   Implicit Definability: Q is *determined by the data only*
2. Can we *test* for this? YES:
   $$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^*,\ \text{where } (.)^* \text{ renames all symbols but } A_j \& V_j.$$
3. Does it mean we have a *rewriting* (a plan)? YES:

Theorem (Beth, 1953)

Q implicitly definable in $A_1, \ldots, A_k$. Then Q is explicitly definable, i.e., $Q \equiv \zeta$ for some $\zeta \in FO_{|[A_1,\ldots,A_k,V_1,\ldots,V_j]}$.

Beth Definability

(1) $\mathcal{T}$ a schema (theory), (2) Q a (FO) query, and
(3) $A_1, \ldots, A_k$ data: indices with $BP(A_j) = (\{\}, V_j)$.

1. What does it mean for a query to be *defined* by the *data*?
   Implicit Definability: Q is *determined by the data only*
2. Can we *test* for this? YES:
   $$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^*,$$ where $(.)^*$ renames all symbols but A_j & V_j.
3. Does it mean we have a *rewriting* (a plan)? YES:

Theorem (Beth, 1953)

Q implicitly definable in $A_1, \ldots, A_k$. Then Q is explicitly definable, i.e., $Q \equiv \zeta$ for some $\zeta \in FO_{|[A_1,\ldots,A_k,V_1,\ldots,V_j]}$.

... this unfortunately fails in *finite models*

Craig Interpolation

Theorem (Craig, 1957)

For $\varphi \rightarrow \psi$ a valid FO formula there is a FO formula $\zeta \in \mathcal{L}(\varphi) \cap \mathcal{L}(\psi)$, an interpolant, such that $\varphi \rightarrow \zeta$ and $\zeta \rightarrow \psi$ are valid.

How do we use this? Convert

$$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^* \quad \text{to} \quad \models ((\bigwedge \mathcal{T}) \wedge Q) \rightarrow ((\bigwedge \mathcal{T}^*) \rightarrow Q^*)$$

and then extract an interpolant ζ from $\models ((\bigwedge \mathcal{T}) \wedge Q) \rightarrow ((\bigwedge \mathcal{T}^*) \rightarrow Q^*)$

... note that ζ contains only the "required" symbols.

Craig Interpolation

Theorem (Craig, 1957)

For $\varphi \to \psi$ a valid FO formula there is a FO formula $\zeta \in \mathcal{L}(\varphi) \cap \mathcal{L}(\psi)$, an interpolant, such that $\varphi \to \zeta$ and $\zeta \to \psi$ are valid.

How do we use this? Convert

$$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^* \quad \text{to} \quad \models ((\bigwedge \mathcal{T}) \wedge Q) \to ((\bigwedge \mathcal{T}^*) \to Q^*)$$

and then extract an interpolant ζ from $\vdash ((\bigwedge \mathcal{T}) \wedge Q) \to ((\bigwedge \mathcal{T}^*) \to Q^*)$

… note that ζ contains only the "required" symbols.

Craig Interpolation

Theorem (Craig, 1957)

For $\varphi \to \psi$ a valid FO formula there is a FO formula $\zeta \in \mathcal{L}(\varphi) \cap \mathcal{L}(\psi)$, an interpolant, such that $\varphi \to \zeta$ and $\zeta \to \psi$ are valid.

How do we use this? Convert

$$\mathcal{T} \cup \mathcal{T}^* \models Q \leftrightarrow Q^* \quad \text{to} \quad \models ((\bigwedge \mathcal{T}) \wedge Q) \to ((\bigwedge \mathcal{T}^*) \to Q^*)$$

and then extract an interpolant ζ from $\vdash ((\bigwedge \mathcal{T}) \wedge Q) \to ((\bigwedge \mathcal{T}^*) \to Q^*)$

... note that ζ contains only the "required" symbols.

How to Get Interpolants? Biased Tableaux

Prove $\varphi \rightarrow \psi$ by *refuting* $\{L(\varphi), R(\neg\psi)\}$ using tableaux rules.

Closed Branches

$$S \cup \{L(\varphi), L(\neg\varphi)\} \xrightarrow{int} \bot \qquad S \cup \{L(\varphi), R(\neg\varphi)\} \xrightarrow{int} \varphi$$

$$S \cup \{R(\varphi), R(\neg\varphi)\} \xrightarrow{int} \top \qquad S \cup \{R(\varphi), L(\neg\varphi)\} \xrightarrow{int} \neg\varphi$$

Propositional Rules (only the “interesting” ones)

$$\frac{S \cup \{L(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} \zeta}{S \cup \{L(\varphi_1), L(\varphi_2)\} \xrightarrow{int} \zeta} \qquad \frac{S \cup \{L(\neg(\varphi_1 \wedge \varphi_2))\} \xrightarrow{int} \zeta_1 \vee \zeta_2}{S \cup \{L(\neg\varphi_1)\} \xrightarrow{int} \zeta_1 \qquad S \cup \{L(\neg\varphi_2)\} \xrightarrow{int} \zeta_2}$$

$$\frac{S \cup \{R(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} \zeta}{S \cup \{R(\varphi_1), R(\varphi_2)\} \xrightarrow{int} \zeta} \qquad \frac{S \cup \{R(\neg(\varphi_1 \wedge \varphi_2))\} \xrightarrow{int} \zeta_1 \wedge \zeta_2}{S \cup \{R(\neg\varphi_1)\} \xrightarrow{int} \zeta_1 \qquad S \cup \{R(\neg\varphi_2)\} \xrightarrow{int} \zeta_2}$$

Quantifiers&Equality Rules . . . similar

How to Get Interpolants? Biased Tableaux

Prove $\varphi \to \psi$ by *refuting* $\{L(\varphi), R(\neg\psi)\}$ using tableaux rules.

Closed Branches

$$S \cup \{L(\varphi), L(\neg\varphi)\} \xrightarrow{int} \bot \qquad S \cup \{L(\varphi), R(\neg\varphi)\} \xrightarrow{int} \varphi$$

$$S \cup \{R(\varphi), R(\neg\varphi)\} \xrightarrow{int} \top \qquad S \cup \{R(\varphi), L(\neg\varphi)\} \xrightarrow{int} \neg\varphi$$

Propositional Rules (only the "interesting" ones)

$$\frac{S \cup \{L(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} \zeta}{S \cup \{L(\varphi_1), L(\varphi_2)\} \xrightarrow{int} \zeta} \qquad \frac{S \cup \{L(\neg(\varphi_1 \wedge \varphi_2))\} \xrightarrow{int} \zeta_1 \vee \zeta_2}{S \cup \{L(\neg\varphi_1)\} \xrightarrow{int} \zeta_1 \quad S \cup \{L(\neg\varphi_2)\} \xrightarrow{int} \zeta_2}$$

$$\frac{S \cup \{R(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} \zeta}{S \cup \{R(\varphi_1), R(\varphi_2)\} \xrightarrow{int} \zeta} \qquad \frac{S \cup \{R(\neg(\varphi_1 \wedge \varphi_2))\} \xrightarrow{int} \zeta_1 \wedge \zeta_2}{S \cup \{R(\neg\varphi_1)\} \xrightarrow{int} \zeta_1 \quad S \cup \{R(\neg\varphi_2)\} \xrightarrow{int} \zeta_2}$$

Quantifiers&Equality Rules . . . similar

. . . and then extract the interpolant ζ s.t. $\varphi \to \zeta \to \psi$.

Example: Tableaux for $\{x \mid \exists y, z.E(x, y, z)\}$

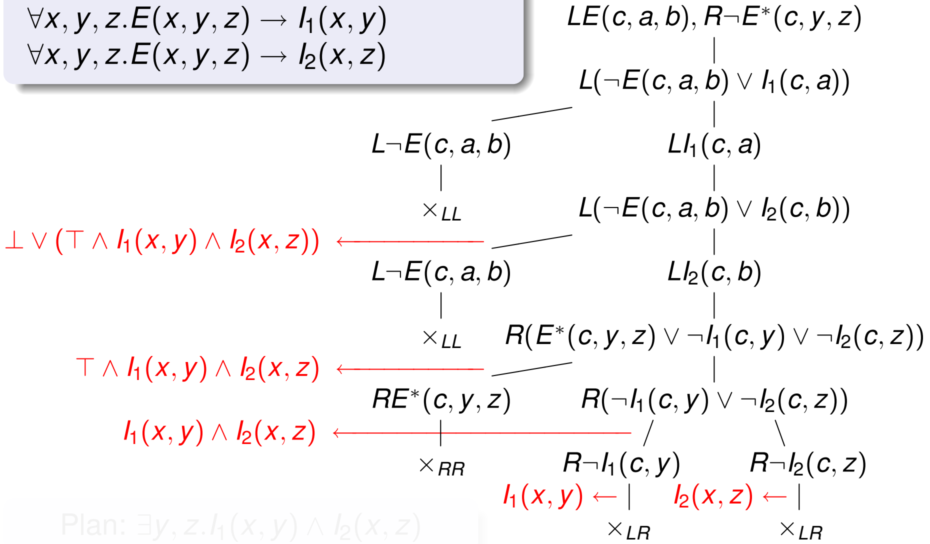
$$\forall x, y, z.I_1(x, y) \wedge I_2(x, z) \rightarrow E(x, y, z)$$
$$\forall x, y, z.E(x, y, z) \rightarrow I_1(x, y)$$
$$\forall x, y, z.E(x, y, z) \rightarrow I_2(x, z)$$

- $LE(c, a, b), R\neg E^*(c, y, z)$
  - $L(\neg E(c, a, b) \vee I_1(c, a))$
    - $L\neg E(c, a, b)$
      - $\times_{LL}$
    - $LI_1(c, a)$
      - $L(\neg E(c, a, b) \vee I_2(c, b))$
        - $L\neg E(c, a, b)$
          - $\times_{LL}$
        - $LI_2(c, b)$
          - $R(E^*(c, y, z) \vee \neg I_1(c, y) \vee \neg I_2(c, z))$
            - $RE^*(c, y, z)$
              - $\times_{RR}$
            - $R(\neg I_1(c, y) \vee \neg I_2(c, z))$
              - $R\neg I_1(c, y)$
                - $\times_{LR}$
              - $R\neg I_2(c, z)$
                - $\times_{LR}$

Plan: $\exists y, z.I_1(x, y) \wedge I_2(x, z)$

Example: Tableaux for $\{x \mid \exists y, z.E(x, y, z)\}$

$\forall x, y, z.I_1(x, y) \wedge I_2(x, z) \rightarrow E(x, y, z)$
$\forall x, y, z.E(x, y, z) \rightarrow I_1(x, y)$
$\forall x, y, z.E(x, y, z) \rightarrow I_2(x, z)$



Example: Tableaux for $\{x \mid \exists y, z.E(x, y, z)\}$

$$\forall x, y, z.I_1(x, y) \wedge I_2(x, z) \rightarrow E(x, y, z)$$
$$\forall x, y, z.E(x, y, z) \rightarrow I_1(x, y)$$
$$\forall x, y, z.E(x, y, z) \rightarrow I_2(x, z)$$

- $LE(c, a, b), R\neg E^*(c, y, z)$
  - $L(\neg E(c, a, b) \vee I_1(c, a))$
    - $L\neg E(c, a, b)$
      - $\times_{LL}$
    - $LI_1(c, a)$
      - $L(\neg E(c, a, b) \vee I_2(c, b))$
        - $L\neg E(c, a, b)$
          - $\times_{LL}$
        - $LI_2(c, b)$
          - $R(E^*(c, y, z) \vee \neg I_1(c, y) \vee \neg I_2(c, z))$
            - $RE^*(c, y, z)$
              - $\times_{RR}$
            - $R(\neg I_1(c, y) \vee \neg I_2(c, z))$
              - $R\neg I_1(c, y)$
                - $\times_{LR}$
              - $R\neg I_2(c, z)$
                - $\times_{LR}$

Plan: $\exists y, z.I_1(x, y) \wedge I_2(x, z)$

Example2: CQ Views (via Resolution&Vampire)

1. $Q(sk_0, sk_1)$ [Input]
2. $\neg Q'(sk_0, sk_1)$ [Input]
3. $V_2(X_0, X_1) \vee \neg R'(X_0, X_3) \vee \neg R'(X_3, X_1)$ [Input]
4. $R(sk_3(X_0, X_1), X_1) \vee \neg V_2(X_0, X_1)$ [Input]
5. $R(X_0, sk_3(X_0, X_1)) \vee \neg V_2(X_0, X_1)$ [Input]
6. $R'(sk_4(X_0, X_1), sk_5(X_0, X_1)) \vee \neg V_1(X_0, X_1)$ [Input]
7. $R'(sk_5(X_0, X_1), X_1) \vee \neg V_1(X_0, X_1)$ [Input]
8. $R'(sk_4(X_0, X_1), X_0) \vee \neg V_1(X_0, X_1)$ [Input]
9. $R'(sk_6(X_0, X_1), sk_7(X_0, X_1)) \vee \neg V_3(X_0, X_1)$ [Input]
10. $R'(sk_7(X_0, X_1), X_1) \vee \neg V_3(X_0, X_1)$ [Input]
11. $R'(X_0, sk_6(X_0, X_1)) \vee \neg V_3(X_0, X_1)$ [Input]
12. $V_3(X_0, X_1) \vee \neg R(X_0, X_4) \vee \neg R(X_5, X_1) \vee \neg R(X_4, X_5)$ [Input]
13. $V_1(X_0, X_1) \vee \neg R(X_4, X_0) \vee \neg R(X_5, X_1) \vee \neg R(X_4, X_5)$ [Input]
14. $Q'(X_0, X_1) \vee \neg R'(X_2, X_3) \vee \neg R'(X_2, X_0) \vee \neg R'(X_4, X_1) \vee \neg R'(X_3, X_4)$ [Input]
15. $R(sk_{13}(X_0, X_1), sk_{14}(X_0, X_1)) \vee \neg Q(X_0, X_1)$ [Input]
16. $R(sk_1 4(X_0, X_1), X_1) \vee \neg Q(X_0, X_1)$ [Input]
17. $R(sk_1 2(X_0, X_1), X_0) \vee \neg Q(X_0, X_1)$ [Input]
18. $R(sk_{12}(X_0, X_1), sk_{13}(X_0, X_1)) \vee \neg Q(X_0, X_1)$ [Input]
19. $R(sk_{13}(sk_0, sk_1), sk_{14}(sk_0, sk_1))$ [Res:1,15]
20. $R(sk_{12}(sk_0, sk_1), sk_0)$ [Res:1,17]
21. $R(sk_{12}(sk_0, sk_1), sk_{13}(sk_0, sk_1))$ [Res:1,18]
22. $\neg R(sk_{12}(sk_0, sk_1), X_1) \vee \neg R(sk_{13}(sk_0, sk_1), X_2) \vee V_1(X_1, X_2)$ [Res:13,21]
23. $\neg R(sk_{13}(sk_0, sk_1), X_1) \vee V_1(sk_0, X_1)$ [Res:20,22]
24. $V_1(sk_0, sk_{14}(sk_0, sk_1))$ [Res:19,23]
25. $\neg R'(X_0, X_1) \vee \neg R'(X_0, sk_0) \vee \neg R'(X_2, sk_1) \vee \neg R'(X_1, X_2)$ [Res:2,14]
26. $\neg R'(sk_7(X_1, X_2), sk_1) \vee \neg R'(X_3, sk_6(X_1, X_2)) \vee \neg V_3(X_1, X_2) \vee \neg R'(X_3, sk_0)$ [Res:9,25]
27. $\neg R'(X_1, sk_6(X_2, sk_1)) \vee \neg V_3(X_2, sk_1) \vee \neg R'(X_1, sk_0)$ [Res:10,26]
28. $\neg V_3(X_1, sk_1) \vee \neg R'(X_1, sk_0)$ [Res:11,27]
29. $\neg R'(X_1, sk_5(X_2, X_3)) \vee V_2(X_1, X_3) \vee \neg V_1(X_2, X_3)$ [Res:3,7]
30. $V_2(sk_4(X_1, X_2), X_2) \vee \neg V_1(X_1, X_2)$ [Res:6,29]
31. $R(sk_1 4(sk_0, sk_1), sk_1)$ [Res:1,16]
32. $\neg R(X_1, sk_3(X_2, X_3)) \vee \neg R(X_3, X_4) \vee \neg V_2(X_2, X_3) \vee V_3(X_1, X_4)$ [Res:4,12]
33. $\neg R(X_1, X_2) \vee \neg V_2(X_3, X_1) \vee V_3(X_3, X_2)$ [Res:5,32]
34. $\neg V_2(X_1, sk_{14}(sk_0, sk_1)) \vee V_3(X_1, sk_1)$ [Res:31,33]
35. $V_3(sk_4(X_1, sk_{14}(sk_0, sk_1)), sk_1) \vee \neg V_1(X_1, sk_{14}(sk_0, sk_1))$ [Res:30,34]
36. $\neg R'(sk_4(X_1, sk_{14}(sk_0, sk_1)), sk_0) \vee \neg V_1(X_1, sk_{14}(sk_0, sk_1))$ [Res:28,35]
37. $\square$ [Res:8,24,36 (w/forward subsumption)]

Domain Independence

Question

Given a schema $\mathcal{T}$, set of indices $\mathcal{I}$, and a (range restricted) query Q such that a rewriting ζ for Q over $\mathcal{I}$ exists.

is ζ domain independent (DI)?

- Not in general;
- To guarantee DI we define a *restriction* on constraints in $\mathcal{T}$:

Definition (Domain Independent Constraint)

A constraint is *domain independent* if its truth depends only on the interpretation of logical parameters (and not on the domain).

Theorem

Domain Independence

Question

Given a schema $\mathcal{T}$, set of indices $\mathcal{I}$, and a (range restricted) query Q such that a rewriting ζ for Q over $\mathcal{I}$ exists.

is ζ domain independent (DI)?

- Not in general; consider

$$\mathcal{T} = \{\forall x.P(x) \vee R(x), \neg\exists x.P(x) \wedge R(x)\}$$

Then, for $\{x \mid P(x)\}$, the rewriting over $\{R\}$ is $\{x \mid \neg R(x)\}$.

- To guarantee DI we define a *restriction* on constraints in $\mathcal{T}$:

Definition (Domain Independent Constraint)

A constraint is *domain independent* if its truth depends only on the interpretation of logical parameters (and not on the domain).

Theorem

Domain Independence

Question

Given a schema $\mathcal{T}$, set of indices $\mathcal{I}$, and a (range restricted) query Q such that a rewriting ζ for Q over $\mathcal{I}$ exists.

is ζ domain independent (DI)?

- Not in general;
- To guarantee DI we define a *restriction* on constraints in $\mathcal{T}$:

Definition (Domain Independent Constraint)

A constraint is *domain independent* if its truth depends only on the interpretation of logical parameters (and not on the domain).

... all usual database constraints are domain independent

Theorem

Let $\mathcal{T}$ be a schema, $\mathcal{I}$ a set of indices, and Q a range restricted query for which a rewriting ζ exists. Then ζ is domain independent

Domain Independence

Question

Given a schema $\mathcal{T}$, set of indices $\mathcal{I}$, and a (range restricted) query Q such that a rewriting ζ for Q over $\mathcal{I}$ exists.

is ζ domain independent (DI)?

- Not in general;
- To guarantee DI we define a *restriction* on constraints in $\mathcal{T}$:

Definition (Domain Independent Constraint)

A constraint is *domain independent* if its truth depends only on the interpretation of logical parameters (and not on the domain).

Theorem

Let $\mathcal{T}$ be a schema, $\mathcal{I}$ a set of indices, and Q a range restricted query for which a rewriting ζ exists. Then ζ is domain independent.

Summary

Beth Definability and Interpolation provide
a starting point for *query optimization* in first-order logic.

Features:

- ⇒ handles full first order logic (constraints and queries)
- ⇒ builds on decades of research in *theorem proving*

Drawbacks:

- ⇒ handles full first order logic (constraints and queries)
  ... no decidability (of plan existence) in general
- ⇒ expensive&incomplete reasoning

Open Issues:

- ⇒ how to handle “database extras”
- ⇒ how to get “optimal plans”

Summary

Beth Definability and Interpolation provide
a starting point for *query optimization* in first-order logic.

Features:

⇒ handles full first order logic (constraints and queries)

⇒ builds on decades of research in *theorem proving*

Drawbacks:

⇒ handles full first order logic (constraints and queries)
. . . no decidability (of plan existence) in general

⇒ expensive&incomplete reasoning

Open Issues:

⇒ how to handle “database extras”

⇒ how to get “optimal plans”

Summary

Beth Definability and Interpolation provide
a starting point for *query optimization* in first-order logic.

Features:

$\Rightarrow$ handles full first order logic (constraints and queries)

$\Rightarrow$ builds on decades of research in *theorem proving*

Drawbacks:

$\Rightarrow$ handles full first order logic (constraints and queries)
... no decidability (of plan existence) in general

$\Rightarrow$ expensive&incomplete reasoning

Open Issues:

$\Rightarrow$ how to handle "database extras"

$\Rightarrow$ how to get "optimal plans"

Summary

Beth Definability and Interpolation provide
a starting point for *query optimization* in first-order logic.

Features:

$\Rightarrow$ handles full first order logic (constraints and queries)

$\Rightarrow$ builds on decades of research in *theorem proving*

Drawbacks:

$\Rightarrow$ handles full first order logic (constraints and queries)
. . . no decidability (of plan existence) in general

$\Rightarrow$ expensive&incomplete reasoning

Open Issues:

$\Rightarrow$ how to handle "database extras"

$\Rightarrow$ how to get "optimal plans"