Logical Approach to Physical Data Independence and Query Compilation

Query Rewriting

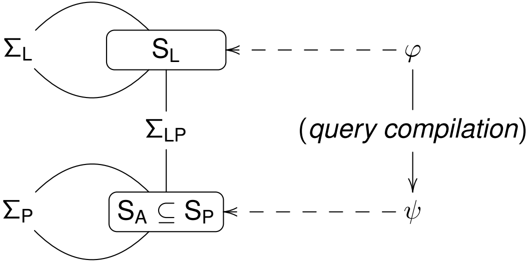
David Toman

D.R. Cheriton School of Computer Science

University of Waterloo

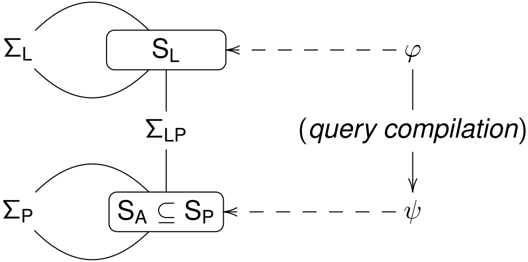
The Story So Far…

$$\Sigma = (\Sigma_L \cup \Sigma_{LP} \cup \Sigma_P)$$



The Story So Far. . .

$$\Sigma \;=\; (\Sigma_L \cup \Sigma_{LP} \cup \Sigma_P)$$



Features:

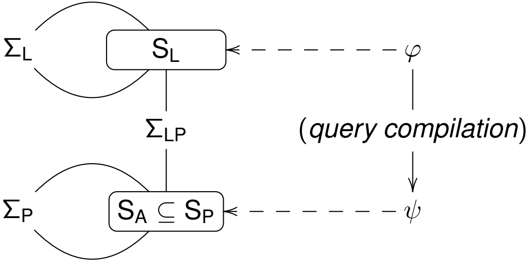
- Flexible *physical design*: constraints $\Sigma_P \cup \Sigma_{LP}$ and code for S_A

  $\Rightarrow$ main-memory operations, disk access, external sources of data, . . . ;
- Query plans are efficient

  $\Rightarrow$ all combination of *access paths* and simple operators;
  $\Rightarrow$ often comparable to hand-written programs.

The Story So Far. . .

$$\Sigma = (\Sigma_L \cup \Sigma_{LP} \cup \Sigma_P)$$



1. How do we find ψ such that $\psi \in \mathcal{L}(S_A)$ and $\Sigma \models \varphi \leftrightarrow \psi$?
2. How do we deal with non-logical issues (e.g., duplicates)?

Goal and Steps

1. Find ψ such that $\psi \in \mathcal{L}(\mathsf{S_A})$ and $\Sigma \models \varphi \leftrightarrow \psi$?
   - search for *optimal* ψ (according to a *cost model*)
   - in general many *candidates* (even for CQ: join-order optimization)

Goal and Steps

1. Find ψ such that $\psi \in \mathcal{L}(\mathsf{S_A})$ and $\Sigma \models \varphi \leftrightarrow \psi$?
   - search for *optimal* ψ (according to a *cost model*)
   - in general many *candidates* (even for CQ: join-order optimization)
2. How do we deal with non-logical issues?
   - elimination of unnecessary *duplicate elimination* operations
   - *cut* insertion (when one solution suffices)

QUERY REWRITING

Chase and Backchase

- Input: φ a *CQ*, Σ a set of *dependencies*, and $\mathsf{S_A}$.

  $\Rightarrow$ a *dependency* is a formula $\forall \bar{x}.\alpha \to \beta$ where α and β are CQs.

Chase and Backchase

- Input: φ a *CQ*, Σ a set of *dependencies*, and S_A.

  $\Rightarrow$ a *dependency* is a formula $\forall \bar{x}.\alpha \rightarrow \beta$ where α and β are CQs.

- Algorithm:
  1. chase φ with Σ producing a CQ $\mathbf{chase}_\Sigma(\varphi)$;
     - $\mathbf{chase}^0_\Sigma \quad = \varphi$
     - $\mathbf{chase}^{i+1}_\Sigma = \mathbf{chase}^i_\Sigma \wedge (\beta\theta)$ for $\forall \bar{x}.\alpha \rightarrow \beta \in \Sigma$ and $\theta : \alpha \mapsto \mathbf{chase}^i_\Sigma$;
     - $\mathbf{chase}_\Sigma \quad = \lim_{i\rightarrow\infty} \mathbf{chase}^i_\Sigma$.
  2. select $\psi \in \mathcal{L}(S_A)$ such that atoms(ψ) $\subseteq$ atoms($\mathbf{chase}_\Sigma(\varphi)$);
  3. chase ψ with Σ producing $\mathbf{chase}_\Sigma(\psi)$;
  4. test whether $\mathbf{chase}_\Sigma(\psi)$ implies φ

     $\Rightarrow$ essentially atoms(φ) $\subseteq$ atoms($\mathbf{chase}_\Sigma(\psi)$).

Chase and Backchase

- Input: φ a *CQ*, Σ a set of *dependencies*, and S_A.
  $\Rightarrow$ a *dependency* is a formula $\forall\bar{x}.\alpha \rightarrow \beta$ where α and β are CQs.
- Algorithm:
  1. chase φ with Σ producing a CQ $\mathbf{chase}_\Sigma(\varphi)$;
  2. select $\psi \in \mathcal{L}(S_A)$ such that atoms$(\psi) \subseteq$ atoms$(\mathbf{chase}_\Sigma(\varphi))$;
  3. chase ψ with Σ producing $\mathbf{chase}_\Sigma(\psi)$;
  4. test whether $\mathbf{chase}_\Sigma(\psi)$ implies φ
     $\Rightarrow$ essentially atoms$(\varphi) \subseteq$ atoms$(\mathbf{chase}_\Sigma(\psi))$.
- Problems:
  - $\mathbf{chase}_\Sigma(\varphi)$ may be infinite (non-termination);
    $\Rightarrow$ in theory restrict Σ to constraints w/terminating chase;
    $\Rightarrow$ in practice fair interleaving of the steps of the algorithm
  - it only works well for CQs.

Won't work in General

- Chase extensions
  - disjunctions in heads of dependencies: UCQ plans
  - denial dependencies: pruning of disjuncts in such UCQ

Won't work in General

- Chase extensions
  - disjunctions in heads of dependencies: UCQ plans
  - denial dependencies: pruning of disjuncts in such UCQ
- does the algorithm find a plan if one exists?

Won't work in General

- Chase extensions
  - disjunctions in heads of dependencies: UCQ plans
  - denial dependencies: pruning of disjuncts in such UCQ
- does the algorithm find a plan if one exists?

Example

- $\mathsf{S_L} = \{R/2\}$, $\mathsf{S_P} = \mathsf{S_A} = \{V_1/2/0, V_2/2/0, V_3/2/0\}$,
- $\Sigma = \{\ \forall x, y. V_1(x, y) \equiv \exists u, w.(R(u, x) \wedge R(u, w) \wedge R(w, y)),\ ,$
  $\forall x, y. V_2(x, y) \equiv \exists u, w.(R(x, u) \wedge R(u, w) \wedge R(w, y)),$
  $\forall x, y. V_3(x, y) \equiv \exists u.(R(x, u) \wedge R(u, y))\ \}$
- $\varphi = \exists u, v, w.(R(u, x) \wedge R(u, w) \wedge R(w, v) \wedge R(v, y))$,

Won't work in General

- Chase extensions
  - disjunctions in heads of dependencies: UCQ plans
  - denial dependencies: pruning of disjuncts in such UCQ
- does the algorithm find a plan if one exists?

Example

- $\mathsf{S}_\mathsf{L} = \{R/2\}$, $\mathsf{S}_\mathsf{P} = \mathsf{S}_\mathsf{A} = \{V_1/2/0, V_2/2/0, V_3/2/0\}$,
- $$\begin{aligned}\Sigma = \{\ & \forall x, y.V_1(x,y) \equiv \exists u, w.(R(u,x) \wedge R(u,w) \wedge R(w,y)), \quad ,\\ & \forall x, y.V_2(x,y) \equiv \exists u, w.(R(x,u) \wedge R(u,w) \wedge R(w,y)),\\ & \forall x, y.V_3(x,y) \equiv \exists u.(R(x,u) \wedge R(u,y)) \quad \}\end{aligned}$$
- $\varphi = \exists u, v, w.(R(u,x) \wedge R(u,w) \wedge R(w,v) \wedge R(v,y))$,
- $\psi = \exists u.(V_1(x,u) \wedge \forall w.(V_3(w,u) \to V_2(w,y)))$.

... but there is not a CQ rewriting.

Won't work in General

- Chase extensions
  - disjunctions in heads of dependencies: UCQ plans
  - denial dependencies: pruning of disjuncts in such UCQ

- does the algorithm find a plan if one exists?

Example

- $\mathsf{S_L} = \{R/2\}$, $\mathsf{S_P} = \mathsf{S_A} = \{V_1/2/0, V_2/2/0, V_3/2/0\}$,
- $$\Sigma = \{ \begin{array}{l} \forall x, y. V_1(x, y) \equiv \exists u, w.(R(u, x) \wedge R(u, w) \wedge R(w, y)), \\ \forall x, y. V_2(x, y) \equiv \exists u, w.(R(x, u) \wedge R(u, w) \wedge R(w, y)), \\ \forall x, y. V_3(x, y) \equiv \exists u.(R(x, u) \wedge R(u, y)) \end{array} \}$$
- $\varphi = \exists u, v, w.(R(u, x) \wedge R(u, w) \wedge R(w, v) \wedge R(v, y))$,
- $\psi = \exists u.(V_1(x, u) \wedge \forall w.(V_3(w, u) \rightarrow V_2(w, y)))$.

. . . but there is not a CQ rewriting.

$\Rightarrow$ cannot be found by chase-backchase

INTERPOLATION

Definability and Interpolation

Definition (Beth Definability)

A formula φ is *definable w.r.t. Σ and $\mathsf{S_A}$* if $\varphi^{M_1} = \varphi^{M_2}$
for every pair M_1, M_2 of models of Σ such that $R^{M_1} = R^{M_2}$ for all $R \in \mathsf{S_A}$.

$\Rightarrow$ sometimes called *parametric definability* (due to $\mathsf{S_A}$).

Definability and Interpolation

Definition (Beth Definability)

A formula φ is *definable w.r.t. Σ and $\mathsf{S_A}$* if $\varphi^{M_1} = \varphi^{M_2}$ for every pair M_1, M_2 of models of Σ such that $R^{M_1} = R^{M_2}$ for all $R \in \mathsf{S_A}$.

$\Rightarrow$ sometimes called *parametric definability* (due to $\mathsf{S_A}$).

Theorem (Craig'57)

Let α and β be FO formulæ such that $\models \alpha \to \beta$. Then there is a FO formula $\gamma \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$, called an interpolant, such that $\models \alpha \to \gamma$ and $\models \gamma \to \beta$.

How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. Σ and S_A

$\Rightarrow$ provides users with an illusion of a *single model*

How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. Σ and $\mathsf{S_A}$

$\Rightarrow$ provides users with an illusion of a *single model*

Definability Test:

φ is definable w.r.t. Σ and $\mathsf{S_A}$ if and only if $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

for Σ^* and φ^* having all $R \notin \mathsf{S_A}$ replaced by R^* (Beth).

How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. Σ and S_A

$\Rightarrow$ provides users with an illusion of a *single model*

Definability Test:

φ is definable w.r.t. Σ and S_A if and only if $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

for Σ^* and φ^* having all $R \notin S_A$ replaced by R^* (Beth).

Interpolant Existence:

If φ is definable w.r.t. Σ and S_A then

there is a FO $\psi \in \mathcal{L}(S_A)$ such that $\Sigma \models \varphi \leftrightarrow \psi$ (Craig).

How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. Σ and $\mathsf{S_A}$

$\Rightarrow$ provides users with an illusion of a *single model*

Definability Test:

φ is definable w.r.t. Σ and $\mathsf{S_A}$ if and only if $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

for Σ^* and φ^* having all $R \notin \mathsf{S_A}$ replaced by R^* (Beth).

Interpolant Existence:

If φ is definable w.r.t. Σ and $\mathsf{S_A}$ then

there is a FO $\psi \in \mathcal{L}(\mathsf{S_A})$ such that $\Sigma \models \varphi \leftrightarrow \psi$ (Craig).

NOTE: this does NOT account for *binding patterns*.

Derivation

INPUT: finite Σ and φ.; output: ψ

$$
\begin{array}{ll}
\Sigma \cup \Sigma^* \models \varphi \to \varphi^* & \Rightarrow \\
\models (\bigwedge \Sigma) \to ((\bigwedge \Sigma^*) \to (\varphi \to \varphi^*)) & \Rightarrow \\
\models (\bigwedge \Sigma) \to (\varphi \to ((\bigwedge \Sigma^*) \to \varphi^*)) & \Rightarrow \\
\models ((\bigwedge \Sigma) \wedge \varphi) \to ((\bigwedge \Sigma^*) \to \varphi^*)) & \Rightarrow \\
\models ((\bigwedge \Sigma) \wedge \varphi) \to \psi \text{ and } \models \psi \to ((\bigwedge \Sigma^*) \to \varphi^*) & \Rightarrow \\
\models (\bigwedge \Sigma) \to (\varphi \to \psi) \text{ and } \models (\bigwedge \Sigma^*) \to (\psi \to \varphi^*) & \Rightarrow \\
\Sigma \models \varphi \to \psi \text{ and } \Sigma^* \models \psi \to \varphi^* & \Rightarrow \\
\Sigma \cup \Sigma^* \models \varphi \to \psi \text{ and } \Sigma \cup \Sigma^* \models \psi \to \varphi^* &
\end{array}
$$

$$
\begin{array}{ll}
\Sigma \cup \Sigma^* \models \varphi^* \to \varphi & \Rightarrow \\
\vdots & \Rightarrow \\
\Sigma \cup \Sigma^* \models \varphi^* \to \psi \text{ and } \Sigma \cup \Sigma^* \models \psi \to \varphi &
\end{array}
$$

Constructive Interpolation via Tableau

IDEA:

We try to *prove* $\Sigma \cup \Sigma^* \models \varphi \to \varphi^*$ producing a proof (in a form of closed tableau) from which *extract the interpolant*.

Constructive Interpolation via Tableau

IDEA:

We try to *prove* $\Sigma \cup \Sigma^* \models \varphi \to \varphi^*$ producing a proof (in a form of closed tableau) from which *extract the interpolant*.

(Biased) Analytic Tableau

A *refutation* proof system for FOL:

- instead of $\vdash \alpha \to \beta$ we show $S = \{\alpha^L, \neg\beta^R\}$ is inconsistent
  formulæ in S are *adorned* by L and R (needed for interpolant extraction);
- we use *inference rules* to generate *successors* of S in a proof tree;
- a proof is complete if all leaves contain a clash, a pair $\delta, \neg\delta$
  otherwise the tableau saturates an we can extract a counterexample.

Interpolant Extraction (by example)

- an *invariant* for interpolation $S \xrightarrow{int} \psi$ is $(\bigwedge S^L) \to \psi$ and $\psi \to (\neg \bigwedge S^R)$ where S^L and S^R are subsets of S derived from adornments of formulas.

Interpolant Extraction (by example)

- an *invariant* for interpolation $S \xrightarrow{int} \psi$ is $(\bigwedge S^L) \to \psi$ and $\psi \to (\neg \bigwedge S^R)$ where S^L and S^R are subsets of S derived from adornments of formulas.
- tableau rules (sample):
  - LR clash $\boxed{S \cup \{R^L, \neg R^R\} \xrightarrow{int} R}$, $R \in \mathsf{S}_\mathsf{A}$ because $(\bigwedge S^L \wedge R^L) \to R$ and $R \to (R^R \vee \neg \bigwedge S^R)$

Interpolant Extraction (by example)

- an *invariant* for interpolation $S \xrightarrow{int} \psi$ is $(\bigwedge S^L) \rightarrow \psi$ and $\psi \rightarrow (\neg \bigwedge S^R)$ where S^L and S^R are subsets of S derived from adornments of formulas.
- tableau rules (sample):
  - LR clash $\boxed{S \cup \{R^L, \neg R^R\} \xrightarrow{int} R}$, $R \in \mathsf{S_A}$ because $(\bigwedge S^L \wedge R^L) \rightarrow R$ and $R \rightarrow (R^R \vee \neg \bigwedge S^R)$
  - L-conjunction $\boxed{\dfrac{S \cup \{\alpha^L, \beta^L\} \xrightarrow{int} \delta}{S \cup \{(\alpha \wedge \beta)^L\} \xrightarrow{int} \delta}}$ because $(\bigwedge S^L \wedge \alpha^L \wedge \beta^L) \rightarrow \delta$ implies $(\bigwedge S^L \wedge (\alpha \wedge \beta)^L) \rightarrow \delta$.

Interpolant Extraction (by example)

- an *invariant* for interpolation $S \xrightarrow{int} \psi$ is $(\bigwedge S^L) \to \psi$ and $\psi \to (\neg \bigwedge S^R)$ where S^L and S^R are subsets of S derived from adornments of formulas.
- tableau rules (sample):
  - LR clash $\boxed{S \cup \{R^L, \neg R^R\} \xrightarrow{int} R}$, $R \in \mathsf{S_A}$ because
    $(\bigwedge S^L \wedge R^L) \to R$ and $R \to (R^R \vee \neg \bigwedge S^R)$
  - L-conjunction $\boxed{\dfrac{S \cup \{\alpha^L, \beta^L\} \xrightarrow{int} \delta}{S \cup \{(\alpha \wedge \beta)^L\} \xrightarrow{int} \delta}}$ because
    $(\bigwedge S^L \wedge \alpha^L \wedge \beta^L) \to \delta$ implies $(\bigwedge S^L \wedge (\alpha \wedge \beta)^L) \to \delta$.
  - R-Disjunction $\boxed{\dfrac{S \cup \{\alpha^R\} \xrightarrow{int} \delta_\alpha \text{ and } S \cup \{\beta^R\} \xrightarrow{int} \delta_\beta}{S \cup \{(\alpha \vee \beta)^R\} \xrightarrow{int} \delta_\alpha \wedge \delta_\beta}}$ because
    $\bigwedge S^L \to \delta_\alpha, \delta_\alpha \to (\alpha^R \vee \neg \bigwedge S^R)$ and $\bigwedge S^L \to \delta_\beta, \delta_\beta \to (\beta^R \vee \neg \bigwedge S^R)$
    implies $(\bigwedge S^L) \to \delta_\alpha \wedge \delta_\beta, \delta_\alpha \wedge \delta_\beta \to (\alpha \vee \beta)^R \vee \neg \bigwedge S^R$.
  - etc. (see [Fitting] for details)

Implementation “details”

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

Implementation "details"

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO)

Implementation “details”

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO, why?)

Implementation “details”

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO)

   $\Rightarrow$ do we get “enough”? (NO)

Implementation "details"

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO)

   $\Rightarrow$ do we get "enough"? (NO, needs tableau modifications)

Implementation “details”

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO)

   $\Rightarrow$ do we get “enough”? (NO)

2. Is backtracking of the tableau proofs feasible approach? (NO)

   $\Rightarrow$ in Σ we separate

   - “logical” (lots, complex) and
   - “physical” (few, simple) constraints

   … limits backtracking during plan search to physical constraints;

Implementation "details"

1. Plan enumeration:

   $\Rightarrow$ enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)

   $\Rightarrow$ do we want to enumerate all equivalent rewritings? (NO)

   $\Rightarrow$ do we get "enough"? (NO)

2. Is backtracking of the tableau proofs feasible approach? (NO)

   $\Rightarrow$ in Σ we separate

   - "logical" (lots, complex) and
   - "physical" (few, simple) constraints

   ... limits backtracking during plan search to physical constraints;

3. Still needs to check for satisfaction of *binding patterns*.

POST-PROCESSING

Duplicate Elimination Elimination

In general $\exists x.\psi$ has to *eliminate duplicates* in the result (expensive)

$\Rightarrow$ we want to detect when duplicate elimination can be safely omitted.

Duplicate Elimination Elimination

In general $\exists x.\psi$ has to *eliminate duplicates* in the result (expensive)

$\Rightarrow$ we want to detect when duplicate elimination can be safely omitted.

IDEA:

Separate the projection operation ($\exists \bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Duplicate Elimination Elimination

In general $\exists x.\psi$ has to *eliminate duplicates* in the result (expensive)

$\Rightarrow$ we want to detect when duplicate elimination can be safely omitted.

IDEA:

Separate the projection operation ($\exists \bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$\begin{aligned}
Q[\{R(x_1,\ldots,x_k)\}] &\leftrightarrow Q[R(x_1,\ldots,x_k)] \\
Q[\{Q_1 \wedge Q_2\}] &\leftrightarrow Q[\{Q_1\} \wedge \{Q_2\}] \\
Q[\{\neg Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg\{Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{Q_1 \vee Q_2\}] &\leftrightarrow Q[\{Q_1\} \vee \{Q_2\}] \quad \text{if } \Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \rightarrow \bot \\
Q[\{\exists x.Q_1\}] &\leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if} \\
&\quad \Sigma \cup \{Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x]\} \models y_1 \approx y_2
\end{aligned}$$

where y_1 and y_2 are fresh variable names not occurring in Q, Q_1, and Q_2.

Summary

1. interpolation provides a powerful tool for query optimization, but
   - efficiency of reasoning is an issue (single proof is not sufficient)
   - generating enough candidate plans (at odds with structural proofs)
   - but needs to avoid useless plans (e.g., co-joining tautologies, etc.)

Summary

1. interpolation provides a powerful tool for query optimization, but
   - efficiency of reasoning is an issue (single proof is not sufficient)
   - generating enough candidate plans (at odds with structural proofs)
   - but needs to avoid useless plans (e.g., co-joining tautologies, etc.)
2. postprocessing needed to deal with non-FO features
   - duplicate semantics (hard to even define query equivalence!)
   - cuts (see textbook for details)