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…

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

\[ \Sigma_L \]

\[ \Sigma_{LP} \]

\[ \Sigma_P \]

\[ S_L \]

\[ S_{LP} \]

\[ S_A \subseteq S_P \]

\[ \text{(query compilation)} \]

\[ \varphi \]

\[ \psi \]

How do we find \( \psi \) such that \( \psi \in L(S_A) \) and \( \Sigma |\psi = \phi \leftrightarrow \psi \)?

How do we deal with non-logical issues (e.g., duplicates)?
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 \( \psi \) 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

Find $\psi$ such that $\psi \in \mathcal{L}(S_A)$ and $\Sigma \models \varphi \leftrightarrow \psi$?

- search for optimal $\psi$ (according to a cost model)
- in general many candidates (even for CQ: join-order optimization)
Goal and Steps

1. Find $\psi$ such that $\psi \in \mathcal{L}(S_A)$ and $\Sigma \models \varphi \leftrightarrow \psi$?
   - search for optimal $\psi$ (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:** \( \varphi \) a CQ, \( \Sigma \) a set of *dependencies*, and \( S_A \).
  
  \[ \Rightarrow \text{a dependency is a formula } \forall \bar{x}. \alpha \rightarrow \beta \text{ where } \alpha \text{ and } \beta \text{ are CQs.} \]
Chase and Backchase

- Input: $\varphi$ a CQ, $\Sigma$ a set of dependencies, and $S_A$.
  $\Rightarrow$ a dependency is a formula $\forall \bar{x}. \alpha \rightarrow \beta$ where $\alpha$ and $\beta$ are CQs.

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

Problems:
- $\text{chase}_\Sigma(\varphi)$ may be infinite (non-termination);
- $\Rightarrow$ in theory restrict $\Sigma$ to constraints with terminating chase;
- $\Rightarrow$ in practice fair interleaving of the steps of the algorithm.

Query Rewriting
Chase and Backchase

- Input: $\varphi$ a CQ, $\Sigma$ a set of dependencies, and $S_A$.
  $\Rightarrow$ a dependency is a formula $\forall \bar{x}. \alpha \rightarrow \beta$ where $\alpha$ and $\beta$ are CQs.

- Algorithm:
  1. chase $\varphi$ with $\Sigma$ producing a CQ $\text{chase}_\Sigma(\varphi)$;
  2. select $\psi \in \mathcal{L}(S_A)$ such that $\text{atoms}(\psi) \subseteq \text{atoms}(\text{chase}_\Sigma(\varphi))$;
  3. chase $\psi$ with $\Sigma$ producing $\text{chase}_\Sigma(\psi)$;
  4. test whether $\text{chase}_\Sigma(\psi)$ implies $\varphi$  
     $\Rightarrow$ essentially $\text{atoms}(\varphi) \subseteq \text{atoms}(\text{chase}_\Sigma(\psi))$.

- Problems:
  - $\text{chase}_\Sigma(\varphi)$ may be infinite (non-termination);
    $\Rightarrow$ in theory restrict $\Sigma$ 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

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

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

... 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

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

\( \Rightarrow \) cannot be found by chase-backchase
INTERPOLATION
Definability and Interpolation

Definition (Beth Definability)

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

$\Rightarrow$ sometimes called **parametric definability** (due to $S_A$).
Definability and Interpolation

Definition (Beth Definability)
A formula $\varphi$ is **definable w.r.t.** $\Sigma$ and $S_A$ if $\varphi^{M_1} = \varphi^{M_2}$ for every pair $M_1, M_2$ of models of $\Sigma$ such that $R^{M_1} = R^{M_2}$ for all $R \in S_A$.

⇒ sometimes called *parametric definability* (due to $S_A$).

Theorem (Craig’57)
Let $\alpha$ and $\beta$ be FO formulæ such that $\models \alpha \rightarrow \beta$. Then there is a FO formula $\gamma \in L(\alpha) \cap L(\beta)$, called an interpolant, such that $\models \alpha \rightarrow \gamma$ and $\models \gamma \rightarrow \beta$. 
How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. $\Sigma$ 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. $\Sigma$ and $S_A$

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

**Definability Test:**

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

for $\Sigma^*$ and $\phi^*$ having all $R \not\in S_A$ replaced by $R^*$ (Beth).
How do we Use it?

IDEA:

Only allow queries that are Beth definable w.r.t. $\Sigma$ and $S_A$

$\implies$ provides users with an illusion of a single model

Definability Test:

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

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

Interpolant Existence:

If $\varphi$ is definable w.r.t. $\Sigma$ 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. \( \Sigma \) and \( S_A \)

\[ \Rightarrow \text{provides users with an illusion of a single model} \]

**Definability Test:**

\[ \varphi \text{ is definable w.r.t. } \Sigma \text{ and } S_A \text{ if and only if } \Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^* \]

for \( \Sigma^* \) and \( \varphi^* \) having all \( R \notin S_A \) replaced by \( R^* \) (Beth).

**Interpolant Existence:**

If \( \varphi \) is definable w.r.t. \( \Sigma \) and \( S_A \) then

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

**NOTE:** this does NOT account for *binding patterns*.
INPUT: finite $\Sigma$ and $\varphi$.; output: $\psi$

$\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

$\models (\forall \Sigma) \rightarrow (\forall \Sigma^*) \rightarrow (\varphi \rightarrow \varphi^*)$

$\models \varphi \rightarrow \psi$ and $\models \varphi^* \rightarrow \varphi^*$

$\models (\forall \Sigma) \rightarrow (\varphi \rightarrow \psi)$ and $\models (\forall \Sigma^*) \rightarrow (\psi \rightarrow \varphi^*)$

$\models \varphi \rightarrow \psi$ and $\Sigma \cup \Sigma^* \models \psi \rightarrow \varphi^*$

$\models (\forall \Sigma) \rightarrow (\varphi \rightarrow \psi)$ and $\Sigma \cup \Sigma^* \models \psi \rightarrow \varphi^*$
Constructive Interpolation via Tableau

**IDEA:**

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

**IDEA:**
We try to prove $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$ 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 $\vdash \alpha \rightarrow \beta$ 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 $S \overset{\text{int}}{\rightarrow} \psi$ is $(\land S^L) \rightarrow \psi$ and $\psi \rightarrow (\neg \land S^R)$
where $S^L$ and $S^R$ are subsets of $S$ derived from adornments of formulas.
Interpolant Extraction (by example)

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

- tableau rules (sample):
  
  LR clash $\begin{array}{c} S \cup \{R^L, \neg R^R\} \xrightarrow{\text{int}} R \end{array}$, $R \in S_A$ because
  
  $(\land S^L \land R^L) \rightarrow R$ and $R \rightarrow (R^R \lor \neg \land S^R)$
Interpolant Extraction (by example)

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

- tableau rules (sample):
  - LR clash
    \[
    \begin{array}{c}
    S \cup \{ R^L, \neg R^R \} \xrightarrow{\text{int}} R, \ R \in S_A \text{ because} \\
    (\land S^L \land R^L) \rightarrow R \text{ and } R \rightarrow (R^R \lor \neg \land S^R)
    \end{array}
    \]
  - L-conjunction
    \[
    \begin{array}{c}
    S \cup \{ \alpha^L, \beta^L \} \xrightarrow{\text{int}} \delta \\
    S \cup \{ (\alpha \land \beta)^L \} \xrightarrow{\text{int}} \delta
    \end{array}
    \]
    because
    \[
    (\land S^L \land \alpha^L \land \beta^L) \rightarrow \delta \text{ implies } (\land S^L \land (\alpha \land \beta)^L) \rightarrow \delta.
    \]
Interpolant Extraction (by example)

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

- tableau rules (sample):
  - LR clash $S \cup \{R^L, \neg R^R\} \xrightarrow{\text{int}} R$, $R \in S_A$ because
    $$(\land S^L \land R^L) \to R \text{ and } R \to (R^R \lor \neg \land S^R)$$
  - L-conjunction
    $$(\land S^L \land \alpha^L \land \beta^L) \to \delta \text{ implies } (\land S^L \land (\alpha \land \beta)^L) \to \delta.$$  
  - R-Disjunction
    $$\land S^L \to \delta_\alpha, \delta_\alpha \to (\alpha^R \lor \neg \land S^R) \text{ and } \land S^L \to \delta_\beta, \delta_\beta \to (\beta^R \lor \neg \land S^R)$$
    $$\text{implies } (\land S^L) \to \delta_\alpha \land \delta_\beta, \delta_\alpha \land \delta_\beta \to (\alpha \lor \beta)^R \lor \neg \land S^R.$$
Plan enumeration:

⇒ enumeration of proofs \sim enumeration all equivalent rewritings? (NO)
Plan enumeration:

⇒ enumeration of proofs ~ enumeration all equivalent rewritings? (NO)
⇒ do we want to enumerate all equivalent rewritings? (NO)
Plan enumeration:

⇒ enumeration of proofs ~ enumeration all equivalent rewritings? (NO)
⇒ do we want to enumerate all equivalent rewritings? (NO, why?)
Plan enumeration:

⇒ enumeration of proofs ∼ enumeration all equivalent rewritings? (NO)
⇒ do we want to enumerate all equivalent rewritings? (NO)
⇒ do we get “enough”? (NO)
Plan enumeration:

⇒ enumeration of proofs \sim enumeration all equivalent rewritings? (NO)
⇒ do we want to enumerate all equivalent rewritings? (NO)
⇒ do we get “enough”? (NO, needs tableau modifications)
Plan enumeration:

⇒ enumeration of proofs \( \sim \) enumeration all equivalent rewritings? (NO)
⇒ do we want to enumerate all equivalent rewritings? (NO)
⇒ do we get “enough”? (NO)

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

⇒ in \( \Sigma \) we separate

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

... limits backtracking during plan search to physical constraints;
Plan enumeration:

- enumeration of proofs $\sim$ enumeration all equivalent rewritings? (NO)
- do we want to enumerate all equivalent rewritings? (NO)
- do we get “enough”? (NO)

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

- in $\Sigma$ we separate
  - “logical” (lots, complex) and
  - “physical” (few, simple) constraints

$\ldots$ limits backtracking during plan search to physical constraints;

Still needs to check for satisfaction of binding patterns.
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 ($\exists \bar{x}. \psi$) to a duplicate preserving projection ($\exists$) and an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

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

$Q[\{R(x_1, \ldots, x_k)\}] \leftrightarrow Q[R(x_1, \ldots, x_k)]$

$Q[\{Q_1 \land Q_2\}] \leftrightarrow Q[\{Q_1\} \land \{Q_2\}]$

$Q[\{\neg Q_1\}] \leftrightarrow Q[\neg Q_1]$

$Q[\{Q_1 \lor Q_2\}] \leftrightarrow Q[\{Q_1\} \lor \{Q_2\}]$

if $\Sigma \cup \{Q\} | = Q_1 \land Q_2 \rightarrow \bot$

$Q[\{\exists x. Q_1\}] \leftrightarrow Q[\exists x. \{Q_1\}]$

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

Post-processing

Query Rewriting 15 / 1
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 ($\exists \bar{x}.\cdot$) to
- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

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

$$Q[\{R(x_1, \ldots, x_k)\}] \leftrightarrow Q[R(x_1, \ldots, x_k)]$$
$$Q[\{Q_1 \land Q_2\}] \leftrightarrow Q[\{Q_1\} \land \{Q_2\}]$$
$$Q[\{\neg Q_1\}] \leftrightarrow Q[\neg Q_1]$$
$$Q[\neg\{Q_1\}] \leftrightarrow Q[\neg Q_1]$$
$$Q[\{Q_1 \lor Q_2\}] \leftrightarrow Q[\{Q_1\} \lor \{Q_2\}] \quad \text{if } \sum \cup \{Q[]\} \models Q_1 \land Q_2 \rightarrow \bot$$
$$Q[\{\exists x. Q_1\}] \leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if } \sum \cup \{Q[]\} \land (Q_1)[y_1/x] \land (Q_1)[y_2/x] \models y_1 \approx y_2$$

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

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.).

Post-processing needed to deal with non-FO features:

- Duplicate semantics (hard to even define query equivalence!).
- Cuts (see textbook for details).
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.)

postprocessing needed to deal with non-FO features
- duplicate semantics (hard to even define query equivalence!)
- cuts (see textbook for details)