Logical Approach to Physical Data Independence and Query Compilation

Updates

David Toman

D.R. Cheriton School of Computer Science

University of Waterloo
The Story So Far…

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

Diagram:

- $\Sigma_L$ connected to $S_L$ with $\varphi$
- $\Sigma_{LP}$ connected to $S_A \subseteq S_P$ with $\psi$

- (query compilation)
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, \ldots;
- Query plans are efficient
  \Rightarrow all combination of access paths and simple operators;
  \Rightarrow interpolation and postprocessing to find efficient \( \psi \).
DATA UPDATE
What is an Update?

User Update: $R_i \in S_L$

\[
\text{begin-transaction} \\
R_1^n := \varphi^o_{R_1}; \\
R_2^n := \varphi^o_{R_2}; \\
\vdots \\
R_m^n := \varphi^o_{R_m}; \\
\text{end-transaction}
\]
What is an Update?

User Update: $R_i \in S_L$

\[
\begin{align*}
&\text{begin-transaction} \\
&\quad \text{insert } \varphi_{R_1}^+ \text{ into } R_1; \quad \text{delete } \varphi_{R_1}^- \text{ from } R_1; \\
&\quad \text{insert } \varphi_{R_2}^+ \text{ into } R_2; \quad \text{delete } \varphi_{R_2}^- \text{ from } R_2; \\
&\quad \vdots \\
&\quad \text{insert } \varphi_{R_m}^+ \text{ into } R_m; \quad \text{delete } \varphi_{R_m}^- \text{ from } R_m; \\
&\text{end-transaction}
\end{align*}
\]
What is an Update?

User Update: \( R_i \in S_L \)

\[
\text{begin-transaction} \\
\text{insert } \varphi^+_{R_1} \text{ into } R_1; \quad \text{delete } \varphi^-_{R_1} \text{ from } R_1; \\
\text{insert } \varphi^+_{R_2} \text{ into } R_2; \quad \text{delete } \varphi^-_{R_2} \text{ from } R_2; \\
\vdots \\
\text{insert } \varphi^+_{R_m} \text{ into } R_m; \quad \text{delete } \varphi^-_{R_m} \text{ from } R_m; \\
\text{end-transaction}
\]

\( \Rightarrow \) Assumption-1: transactions are consistency-preserving.

\( \Rightarrow \) Assumption-2: results of \( \varphi^+_{R_1} \) and \( \varphi^-_{R_1} \) are given explicitly as finite sets of tuples.
What is an Update?

User Update: $R_i \in S_L$

```
begin-transaction
    insert $\varphi^+_{R_1}$ into $R_1$; delete $\varphi^-_{R_1}$ from $R_1$;
    insert $\varphi^+_{R_2}$ into $R_2$; delete $\varphi^-_{R_2}$ from $R_2$;
    ...
    insert $\varphi^+_{R_m}$ into $R_m$; delete $\varphi^-_{R_m}$ from $R_m$;
end-transaction
```

$\Rightarrow$ Assumption-1: transactions are consistency-preserving.

$\Rightarrow$ Assumption-2: results of $\varphi^+_{R_1}$ and $\varphi^-_{R_1}$ are given explicitly as finite sets of tuples.

Observation:

Typical user transactions modify only few relations

$\Rightarrow$ transaction types (specify which updates are empty).
What is the Problem?

how do we get from *user update* to *physical update*?

⇒ we need to synthesize $\varphi^+_R, \varphi^-_R$ for each access path $R \in S_A$. 

⇒ how does this affect the synthesis of $\varphi^+_R, \varphi^-_R$ for other access paths?
What is the Problem?

how do we get from user update to physical update?
⇒ we need to synthesize $\varphi^+_R$, $\varphi^-_R$ for each access path $R \in S_A$.

... but what are the available access paths now?
What is the Problem?

1. how do we get from user update to physical update?
   ⇒ we need to synthesize $\varphi^+_R, \varphi^-_R$ for each access path $R \in S_A$.
   ... but what are the available access paths now?

2. $\varphi^+, \varphi^-$ may need invented values (RIDs): where do they come from?
   ⇒ constant complement idea (kind of an oracle).
What is the Problem?

1. how do we get from *user update* to *physical update*?
   ⇒ we need to synthesize $\varphi_R^+, \varphi_R^-$ for each access path $R \in S_A$.
   ... but what are the *available access paths* now?

2. $\varphi^+, \varphi^-$ may need *invented values* (RIDs): where do they come from?
   ⇒ *constant complement* idea (kind of an *oracle*).

3. access paths are modified *one-at-time*: what should the ordering be?
   ⇒ how does this affect the synthesis of $\varphi^+, \varphi^-$ for other access paths?
UPDATING DATA VIA INTERPOLATION
Updating Access Paths

**IDEA: “Update Schema” $\Sigma^U$**

1. make two copies of the schema $\Sigma$:
   - $\Sigma^o$ (old: before update) and $\Sigma^n$ (new: after update);
Updating Access Paths

**IDEA: “Update Schema”** $\Sigma^U$

1. make two copies of the schema $\Sigma$:
   $\Sigma^o$ (old: before update) and $\Sigma^n$ (new: after update);

2. add constraints describing the effect(s) of the *update*:

   $$\Sigma^\pm = \{ \forall x_1, \ldots, x_k. R^+(x_1, \ldots, x_k) \equiv R^n(x_1, \ldots, x_k) \land \neg R^o(x_1, \ldots, x_k),$$
   $$\forall x_1, \ldots, x_k. R^-(x_1, \ldots, x_k) \equiv R^o(x_1, \ldots, x_k) \land \neg R^n(x_1, \ldots, x_k) \mid R/k \in S_L \cup S_P \}. $$
Updating Access Paths

**IDEA: “Update Schema” $\Sigma^U$**

1. make two copies of the schema $\Sigma$: $\Sigma^o$ (old: before update) and $\Sigma^n$ (new: after update);

2. add constraints describing the effect(s) of the update:

$$
\Sigma^\pm = \{ \forall x_1, \ldots, x_k. R^+(x_1, \ldots, x_k) \equiv R^n(x_1, \ldots, x_k) \land \neg R^o(x_1, \ldots, x_k), \\
\forall x_1, \ldots, x_k. R^-(x_1, \ldots, x_k) \equiv R^o(x_1, \ldots, x_k) \land \neg R^n(x_1, \ldots, x_k) \\
| R/k \in S_L \cup S_P \}.
$$

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in $\Sigma^U$??
Updating Access Paths

IDEA: “Update Schema” $\Sigma^U$

1. make two copies of the schema $\Sigma$:
   $\Sigma^o$ (old: before update) and $\Sigma^n$ (new: after update);

2. add constraints describing the effect(s) of the update:
   $\Sigma^\pm = \{ \forall x_1, \ldots, x_k. R^+(x_1, \ldots, x_k) \equiv R^n(x_1, \ldots, x_k) \land \neg R^o(x_1, \ldots, x_k),$
   $\forall x_1, \ldots, x_k. R^-(x_1, \ldots, x_k) \equiv R^o(x_1, \ldots, x_k) \land \neg R^n(x_1, \ldots, x_k) \mid R/k \in S_L \cup S_P \}.$

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in $\Sigma^U$??

- $S_L = \{ R^n \mid R \in S_A \} \cup \{ R^+, R^- \mid R \in S_A \} \cup S^o_L \cup S^n_L$;
- $S_A = \{ R^o \mid R \in S_A \} \cup \{ R^+, R^- \mid R/ \in S_L \}.$
Updating Access Paths

IDEA: “Update Schema” $\Sigma^U$

1. make two copies of the schema $\Sigma$:
   $\Sigma^o$ (old: before update) and $\Sigma^n$ (new: after update);

2. add constraints describing the effect(s) of the update:
   \[
   \Sigma^\pm = \{ \forall x_1, \ldots, x_k. R^+(x_1, \ldots, x_k) \equiv R^n(x_1, \ldots, x_k) \land \neg R^o(x_1, \ldots, x_k), \\
   \forall x_1, \ldots, x_k. R^-(x_1, \ldots, x_k) \equiv R^o(x_1, \ldots, x_k) \land \neg R^n(x_1, \ldots, x_k) \\
   \mid R/k \in S_L \cup S_P \}. \]

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in $\Sigma^U$?

$S_L = \{ R^n \mid R \in S_A \} \cup \{ R^+, R^- \mid R \in S_A \} \cup S^o_L \cup S^n_L$;

$S_A = \{ R^o \mid R \in S_A \} \cup \{ R^+, R^- \mid R/k \in S_L \}$.

Update code = interpolants for $R^+(\bar{x})$ and $R^-(\bar{x})$ under $\Sigma^U$ (for each $R \in S_A$);

... still needs code that inserts/removes tuples into/from access paths.
Example

**Relational Design:** \texttt{emp} records hold dept-id (instead of pointer)

- User transaction of the form:

  \begin{align*}
  \{ \text{employee}^+(123, \text{Bob}, $50k), \text{works}^+(123, 345) \}
  \end{align*}

- Update code (for \texttt{emp}):

  \begin{align*}
  \text{emp}^+(x_1, x_2, x_3, x_4) := \text{employee}^+(x_1, x_2, x_3) \land \text{works}^+(x_1, x_4)
  \end{align*}
Example

Relational Design: \texttt{emp} records \textit{hold} dept-id (instead of pointer)

- User transaction of the form:

\[
\{\text{employee}^+(123, \textit{Bob}, \$50k), \text{works}^+(123, 345)\}
\]

- Update code (for \texttt{emp}^+):

\[
\text{emp}^+(x_1, x_2, x_3, x_4) := \text{employee}^+(x_1, x_2, x_3) \land \text{works}^+(x_1, x_4)
\]

\ldots doesn’t \textit{quite} work with our \textit{physical design with pointers} (why?)
Problem

- User transaction of the form:
  \[
  \{\text{employee}^+(123, Bob, $50k), \text{works}^+(123, 345)\}
  \]

- What should \text{empfile}^+(w)'s update code look like?
Where do the “invented” Values come from?

Problem

- User transaction of the form:

\[
\{ \text{employee}^{+}(123, Bob, $50k), \text{works}^{+}(123, 345) \}
\]

- What should \( \text{empfile}^{+}(w) \)'s update code look like?

\[
\forall x.y.z. (\text{employee}(x, y, z) \rightarrow \exists w. (\text{empfile}(w) \land \text{emp-num}(w, x)))
\]

\[
\forall x.y.z.w. ((\text{employee}(x, y, z) \land \text{emp-num}(w, x)) \rightarrow \text{emp-name}(w, y))
\]

\[
\forall x.y.z.w.u. ((\text{employee}(x, y, z) \land \text{emp-num}(w, x) \land \text{works}(x, u)
\land \text{dept-num}(v, u)) \rightarrow \text{emp-dept}(w, v)).
\]
Problem

What should empfile\textsuperscript{+}(w)’s update code look like?

∀x, y, z.(employee(x, y, z) → ∃w.(empfile(w) \land emp-num(w, x)))
∀x, y, z, w.((employee(x, y, z) \land emp-num(w, x)) → emp-name(w, y))
∀x, y, z, w, u.((employee(x, y, z) \land emp-num(w, x) \land works(x, u)
\land dept-num(v, u)) → emp-dept(w, v)).

1. for empfile\textsuperscript{+}(x):
   ∃y, z, t, u, v.employee\textsuperscript{+}(y, z, t) \land works\textsuperscript{+}(y, u)
   \land deptcomp(u, v) \land empcomp(y, z, t, v, x)

2. for emp-num, etc.: no-op.
Constant Complement Access Path

CC for \texttt{emp} records: \texttt{empcomp/5/4}

function \texttt{empcomp-first}

\texttt{if} an \texttt{emp} record \texttt{r} with \texttt{r->num} = \texttt{x}_1
\texttt{exists} at address \texttt{x}_5 \texttt{return} true

\texttt{x}_5 := \texttt{new emp}
\texttt{x}_5->\texttt{num} := \texttt{x}_1
\texttt{x}_5->\texttt{name} := \texttt{x}_2
\texttt{x}_5->\texttt{sal} := \texttt{x}_3
\texttt{x}_5->\texttt{dept} := \texttt{x}_4
\texttt{return} true

function \texttt{empcomp-next}
\texttt{return} false

Observation(s):
"fills" all fields of a new record
⇒ \texttt{emp-id}, etc., no-ops;
needs to check for existence of all \texttt{emp} records (not just in \texttt{empfile}!)

Updating Data via Interpolation

Updates 10 / 12
Constant Complement Access Path

CC for **emp records**: **empcomp/5/4**

```plaintext
function empcomp-first
    if an emp record r with r->num = x_1
        exists at address x_5 return true
    x_5 := new emp
    x_5->num := x_1
    x_5->name := x_2
    x_5->sal := x_3
    x_5->dept := x_4
    return true

function empcomp-next
    return false
```

Observation(s):
- “fills” all fields of a new record ⇒ emp-id, etc., no-ops;
- needs to check for existence of **all emp records** (not just in empfile!)
Schematic Cycles and Update Sequencing

Still a problem:

For \( \text{empfile}^+(x) \):

\[
\exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \\
\land \text{deptcomp}(u, v) \land \text{empcomp}(y, z, t, v, x),
\]
Schematic Cycles and Update Sequencing

Still a problem:

for \text{empfile}^+(x):

\[ \exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \land \text{deptcomp}(u, v) \land \text{empcomp}(y, z, t, v, x), \]

... what should happen if department \( u \) doesn't exist?
Schematic Cycles and Update Sequencing

Still a problem:

for empfile^+(x):

\[ \exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \]
\[ \land \text{deptcomp}(u, v) \land \text{empcomp}(y, z, t, v, x), \]

... what should happen if department \( u \) doesn’t exist?
... ok, if it is a new department, who manages it?
Schematic Cycles and Update Sequencing

Still a problem:

for $\text{empfile}^+(x)$:

$\exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \land \text{deptcomp}(u, v) \land \text{empcomp}(y, z, t, v, x)$,

\[ \text{... what should happen if department } u \text{ doesn't exist?} \]
\[ \text{... ok, if it is a new department, who manages it?} \]

**IDEA**

*Constant Complement* can ONLY be used with the AP that *stores* the records

$\Rightarrow$ modify $S_A$ as required when compiling AP update code.
Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that stores the records.

⇒ modify $S_A$ as required when compiling AP update code.

Attempt #1: force CC to use all attributes of the AP

Modify $\text{deptcomp}(y, x)$ to $\text{deptcomp}(y, n, m, x)/4/3$

⇒ i.e., force it to take complete dept records (same as empcomp).

for $\text{empfile}^+(x)$:

$\exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u)\ \\ \land \text{department}^+(u, n, m)\ \\ \land \text{deptcomp}(u, n, x, v) \land \text{empcomp}(y, z, t, v, x)$
Schematic Cycles and Update Sequencing

**IDEA**

*Constant Complement* can ONLY be used with the AP that stores the records
⇒ modify $S_A$ as required when compiling AP update code.

**Attempt #1: force CC to use all attributes of the AP**

Modify $\text{deptcomp}(y, x)$ to $\text{deptcomp}(y, n, m, x)/4/3$
⇒ i.e., force it to take complete $\text{dept}$ records (same as $\text{empcomp}$).

for $\text{empfile}^+(x)$:

\[
\exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \\
\land \text{department}^+(u, n, m) \\
\land \text{deptcomp}(u, n, x, v) \land \text{empcomp}(y, z, t, v, x)
\]

How do you insert the first employee and department??
OOPS: not definable (because of *binding patterns*)
Constant Complement can ONLY be used with the AP that stores the records

⇒ modify $S_A$ as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

1. for deptfile$^+(x)$:
   $$\exists y, z, t.\text{department}^+(z, y, t) \land \text{deptcomp}(z, x),$$
IDEA

*Constant Complement* can ONLY be used with the AP that *stores* the records
⇒ modify $S_A$ as required when compiling AP update code.

**Attempt #2: stage updates via reification of attributes**

1. **for deptfile$^+(x)$:**
   \[ \exists y, z, t. \text{department}^+(z, y, t) \land \text{deptcomp}(z, x), \]

2. **for empfile$^+(x)$:**
   \[ \exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \]
   \[ \land \text{deptfile}(v) \land \text{dept-num}(v, u) \land \text{empcomp}(y, z, t, v, x), \]
Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that stores the records
⇒ modify $S_A$ as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

1. for deptfile$^+(x)$:
   \[ \exists y, z, t. \text{department}^+(z, y, t) \land \text{deptcomp}(z, x), \]

2. for empfile$^+(x)$:
   \[ \exists y, z, t, u, v. \text{employee}^+(y, z, t) \land \text{works}^+(y, u) \]
   \[ \land \text{deptfile}(v) \land \text{dept-num}(v, u) \land \text{empcomp}(y, z, t, v, x), \]

3. for dept-manager$^+(x, y)$:
   \[ \exists z, t, u. \text{department}^+(z, t, u) \land \text{deptfile}(x) \land \text{dept-num}(x, z) \]
   \[ \land (\exists z, t, v, w. \text{employee}^+(u, z, t) \land \text{works}^+(u, v) \]
   \[ \land \text{empfile}(y) \land \text{emp-id}(u, y) ) \]
Schematic Cycles and Update Sequencing

**IDEA**

*Constant Complement* can ONLY be used with the AP that *stores* the records

⇒ modify $S_A$ as required when compiling AP update code.

**Attempt #2: stage updates via reification of attributes**

1. for $deptfile^+(x)$:
   \[
   \exists y, z, t. department^+(z, y, t) \land deptcomp(z, x),
   \]

2. for $empfile^+(x)$:
   \[
   \exists y, z, t, u, v. employee^+(y, z, t) \land works^+(y, u) \\
   \land deptfile(v) \land dept-num(v, u) \land empcomp(y, z, t, v, x),
   \]

3. for $dept-manager^+(x, y)$:
   \[
   \exists z, t, u. department^+(z, t, u) \land deptfile(x) \land dept-num(x, z) \\
   \land (\exists z, t, v, w. employee^+(u, z, t) \land works^+(u, v) \\
   \land empcomp(y, z, t, v, x) )
   \]

4. for $dept-name^+(x, y)$:
   \[
   \exists z, t. department^+(z, y, t) \land deptfile(x) \land dept-num(x, z)
   \]
Summary

- code for update of an access path
  1. synthesized queries $\psi^+, \psi^-$ over update schema,
  2. code for primitive inserts/deletes,
  3. code for constant complement access paths (for “invented values”);
- schematic cycles must be broken via reification;
Summary

- code for update of an access path
  1. synthesized queries $\psi^+, \psi^-$ over *update schema*,
  2. code for primitive *inserts/deletes*,
  3. code for *constant complement* access paths (for “invented values”);

- schematic cycles must be broken via reification;

- not entirely satisfactory (e.g., no in-place update)