Logical Approach to Physical Data Independence and Query Compilation

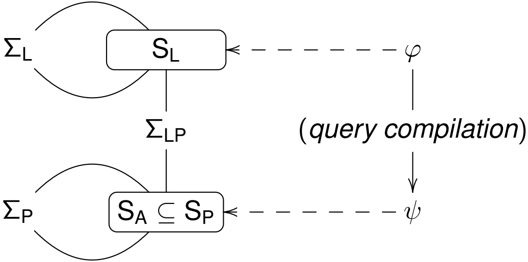
Updates

David Toman

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

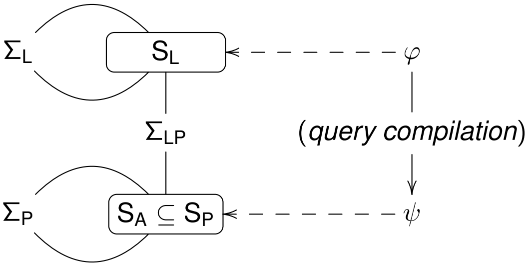
The Story So Far. . .

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



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;

  $\Rightarrow$ interpolation and postprocessing to find efficient ψ.

DATA UPDATE

What is an Update?

User Update: $R_i \in \mathrm{S_L}$

begin-transaction

$$R_1^n := \varphi_{R_1}^o;$$
$$R_2^n := \varphi_{R_2}^o;$$
$$\vdots$$
$$R_m^n := \varphi_{R_m}^o;$$

end-transaction

What is an Update?

User Update: $R_i \in \mathrm{S_L}$

$$
\begin{array}{l}
\textit{begin-transaction} \\
\quad \textit{insert } \varphi^{+}_{R_1} \textit{ into } R_1; \quad \textit{delete } \varphi^{-}_{R_1} \textit{ from } R_1; \\
\quad \textit{insert } \varphi^{+}_{R_2} \textit{ into } R_2; \quad \textit{delete } \varphi^{-}_{R_2} \textit{ from } R_2; \\
\qquad \vdots \\
\quad \textit{insert } \varphi^{+}_{R_m} \textit{ into } R_m; \quad \textit{delete } \varphi^{-}_{R_m} \textit{ from } R_m; \\
\textit{end-transaction}
\end{array}
$$

What is an Update?

User Update: $R_i \in \mathrm{S_L}$

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*.

What is an Update?

User Update: $R_i \in \mathrm{S_L}$

```
begin-transaction
   insert φ⁺_{R1} into R1;    delete φ⁻_{R1} from R1;
   insert φ⁺_{R2} into R2;    delete φ⁻_{R2} from R2;
        ⋮
   insert φ⁺_{Rm} into Rm;    delete φ⁻_{Rm} from Rm;
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?

1. how do we get from *user update* to *physical update*?
   $\Rightarrow$ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in \mathsf{S_A}$.

What is the Problem?

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

   $\Rightarrow$ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in \mathsf{S_A}$.

   ... but what are the *available access path*s now?

What is the Problem?

1. how do we get from *user update* to *physical update*?
   $\Rightarrow$ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in \mathsf{S_A}$.
   ... but what are the *available access path*s now?

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

University of Waterloo

What is the Problem?

1. how do we get from *user update* to *physical update*?
   $\Rightarrow$ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in \mathsf{S_A}$.
   ... but what are the *available access path*s now?

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

3. access paths are modified *one-at-time*: what should the ordering be?
   $\Rightarrow$ how does this affect the synthesis of φ^+, φ^- for other access paths?

UPDATING DATA VIA INTERPOLATION

Updating Access Paths

IDEA: “Update Schema” Σ^U

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

Updating Access Paths

IDEA: "Update Schema" Σ^U

1. make two copies of the schema Σ:
   Σ^o (old: before update) and Σ^n (new: after update);
2. add constraints describing the effect(s) of the *update*:

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

Updating Access Paths

IDEA: "Update Schema" Σ^U

1. make two copies of the schema Σ:
   Σ^o (old: before update) and Σ^n (new: after update);
2. add constraints describing the effect(s) of the *update*:

$$\begin{aligned}\Sigma^{\pm} = \{ & \forall x_1,\ldots,x_k.R^{+}(x_1,\ldots,x_k) \equiv R^n(x_1,\ldots,x_k) \wedge \neg R^o(x_1,\ldots,x_k), \\ & \forall x_1,\ldots,x_k.R^{-}(x_1,\ldots,x_k) \equiv R^o(x_1,\ldots,x_k) \wedge \neg R^n(x_1,\ldots,x_k) \\ & \mid R/k \in \mathsf{S_L} \cup \mathsf{S_P} \}.\end{aligned}$$

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

Updating Access Paths

IDEA: "Update Schema" Σ^U

1. make two copies of the schema Σ:
   Σ^o (old: before update) and Σ^n (new: after update);
2. add constraints describing the effect(s) of the *update*:

$$\begin{aligned}\Sigma^\pm = \{ & \forall x_1,\ldots,x_k.R^+(x_1,\ldots,x_k) \equiv R^n(x_1,\ldots,x_k) \wedge \neg R^o(x_1,\ldots,x_k), \\ & \forall x_1,\ldots,x_k.R^-(x_1,\ldots,x_k) \equiv R^o(x_1,\ldots,x_k) \wedge \neg R^n(x_1,\ldots,x_k) \\ & \mid R/k \in \mathsf{S_L} \cup \mathsf{S_P} \}.\end{aligned}$$

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

$$\begin{aligned}\mathsf{S_L} &= \{R^n \mid R \in \mathsf{S_A}\} \cup \{R^+, R^- \mid R \in \mathsf{S_A}\} \cup \mathsf{S_L^o} \cup \mathsf{S_L^n}; \\ \mathsf{S_A} &= \{R^o \mid R \in \mathsf{S_A}\} \cup \{R^+, R^- \mid R/ \in \mathsf{S_L}\}.\end{aligned}$$

Updating Access Paths

IDEA: "Update Schema" Σ^U

1. make two copies of the schema Σ:
   Σ^o (old: before update) and Σ^n (new: after update);
2. add constraints describing the effect(s) of the *update*:
   $$\Sigma^{\pm} = \{\forall x_1,\ldots,x_k.R^{+}(x_1,\ldots,x_k) \equiv R^n(x_1,\ldots,x_k) \wedge \neg R^o(x_1,\ldots,x_k),\ \forall x_1,\ldots,x_k.R^{-}(x_1,\ldots,x_k) \equiv R^o(x_1,\ldots,x_k) \wedge \neg R^n(x_1,\ldots,x_k) \mid R/k \in \mathsf{S_L} \cup \mathsf{S_P}\}.$$

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

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

Update code = interpolants for $R^+(\bar{x})$ and $R^-(\bar{x})$ under Σ^U (for each $R \in \mathsf{S_A}$);
... still needs *code* that inserts/removes tuples into/from access paths.

Example

Relational Design: `emp` records _hold_ dept-id (instead of pointer)

- User transaction of the form:

$$\{\texttt{employee}^+(123, \mathit{Bob}, \$50k), \texttt{works}^+(123, 345)\}$$

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

$$\texttt{emp}^+(x_1, x_2, x_3, x_4) := \texttt{employee}^+(x_1, x_2, x_3) \wedge \texttt{works}^+(x_1, x_4)$$

Example

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

- User transaction of the form:

$$\{\texttt{employee}^{+}(123, \mathit{Bob}, \$50k), \texttt{works}^{+}(123, 345)\}$$

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

$$\texttt{emp}^{+}(x_1, x_2, x_3, x_4) := \texttt{employee}^{+}(x_1, x_2, x_3) \land \texttt{works}^{+}(x_1, x_4)$$

… doesn't *quite* work with our *physical design with pointers* (why?)

Where do the "invented" Values come from?

Problem

- User transaction of the form:

$$\{\texttt{employee}^+(123, \mathit{Bob}, \$50k), \texttt{works}^+(123, 345)\}$$

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

Where do the “invented” Values come from?

Problem

- User transaction of the form:

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

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

$$\forall x.y,z.(\texttt{employee}(x,y,z) \rightarrow \exists w.(\texttt{empfile}(w) \wedge \texttt{emp-num}(w,x)))$$
$$\forall x,y,z,w.((\texttt{employee}(x,y,z) \wedge \texttt{emp-num}(w,x)) \rightarrow \texttt{emp-name}(w,y))$$
$$\forall x,y,z,w,u.((\texttt{employee}(x,y,z) \wedge \texttt{emp-num}(w,x) \wedge \texttt{works}(x,u) \wedge \texttt{dept-num}(v,u)) \rightarrow \texttt{emp-dept}(w,v)).$$

Where do the "invented" Values come from?

Problem

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

$$\forall x.y,z.(\texttt{employee}(x,y,z) \to \exists w.(\texttt{empfile}(w) \wedge \texttt{emp-num}(w,x)))$$
$$\forall x,y,z,w.((\texttt{employee}(x,y,z) \wedge \texttt{emp-num}(w,x)) \to \texttt{emp-name}(w,y))$$
$$\forall x,y,z,w,u.((\texttt{employee}(x,y,z) \wedge \texttt{emp-num}(w,x) \wedge \texttt{works}(x,u) \wedge \texttt{dept-num}(v,u)) \to \texttt{emp-dept}(w,v)).$$

1. for $\texttt{empfile}^+(x)$:
$$\exists y,z,t,u,v.\texttt{employee}^+(y,z,t) \wedge \texttt{works}^+(y,u) \wedge \texttt{deptcomp}(u,v) \wedge \texttt{empcomp}(y,z,t,v,x)$$
2. for $\texttt{emp-num}$, etc.: no-op.

Constant Complement Access Path

CC for emp records: empcomp/5/4

```
function empcomp-first
   if an emp record r with r->num = x1
         exists at address x5 return true
   x5  := new emp
   x5->num  := x1
   x5->name := x2
   x5->sal  := x3
   x5->dept := x4
   return true

function empcomp-next
   return false
```

Constant Complement Access Path

CC for emp records: empcomp/5/4

```
function empcomp-first
   if an emp record r with r->num = x1
        exists at address x5 return true
   x5  := new emp
   x5->num   := x1
   x5->name  := x2
   x5->sal   := x3
   x5->dept  := x4
   return true

function empcomp-next
   return false
```

Observation(s):

- "fills" all fields of a new record $\Rightarrow$ `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:

1. for $\texttt{empfile}^{+}(x)$:

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

Schematic Cycles and Update Sequencing

Still a problem:

1. for $\texttt{empfile}^+(x)$:

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

... what should happen if department u doesn't exist?

Schematic Cycles and Update Sequencing

Still a problem:

1. for $\texttt{empfile}^+(x)$:

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

. . . 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:

1. for $\texttt{empfile}^{+}(x)$:

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

... what should happen if department u doesn't exist?
... ok, if it is a new department, who manages it?

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
$\Rightarrow$ modify S_A as required when compiling AP update code.

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

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

1. for $\texttt{empfile}^{+}(x)$:

$$\exists y, z, t, u, v.\texttt{employee}^{+}(y, z, t) \wedge \texttt{works}^{+}(y, u) \wedge \texttt{department}^{+}(u, n, m) \wedge \texttt{deptcomp}(u, n, x, v) \wedge \texttt{empcomp}(y, z, t, v, x)$$

Schematic Cycles and Update Sequencing

IDEA

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

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

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

1. for $\texttt{empfile}^{+}(x)$:

$$\exists y, z, t, u, v.\texttt{employee}^{+}(y, z, t) \wedge \texttt{works}^{+}(y, u) \wedge \texttt{department}^{+}(u, n, m) \wedge \texttt{deptcomp}(u, n, x, v) \wedge \texttt{empcomp}(y, z, t, v, x)$$

How do you insert the first employee and department??
OOPS: not definable (because of *binding patterns*)

Schematic Cycles and Update Sequencing

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.

Attempt #2: stage updates via reification of attributes

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

University of Waterloo

Schematic Cycles and Update Sequencing

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.

Attempt #2: stage updates via reification of attributes

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

Schematic Cycles and Update Sequencing

IDEA

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

Attempt #2: stage updates via reification of attributes

1. for $\texttt{deptfile}^+(x)$:

   $\exists y, z, t.\texttt{department}^+(z, y, t) \wedge \texttt{deptcomp}(z, x)$,

2. for $\texttt{empfile}^+(x)$:

   $\exists y, z, t, u, v.\texttt{employee}^+(y, z, t) \wedge \texttt{works}^+(y, u)$
   $\wedge\, \texttt{deptfile}(v) \wedge \texttt{dept-num}(v, u) \wedge \texttt{empcomp}(y, z, t, v, x)$,

3. for $\texttt{dept-manager}^+(x, y)$:

   $\exists z, t, u.\, \texttt{department}^+(z, t, u) \wedge \texttt{deptfile}(x) \wedge \texttt{dept-num}(x, z)$
   $\wedge\, (\exists z, t, v, w.\texttt{employee}^+(u, z, t) \wedge \texttt{works}^+(u, v)$
   $\wedge\, \texttt{empfile}(y) \wedge \texttt{emp-id}(u, y)\,)$

University of Waterloo

Schematic Cycles and Update Sequencing

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.

Attempt #2: stage updates via reification of attributes

1. for $\texttt{deptfile}^+(x)$:

   $\exists y, z, t.\texttt{department}^+(z, y, t) \wedge \texttt{deptcomp}(z, x)$,

2. for $\texttt{empfile}^+(x)$:

   $\exists y, z, t, u, v.\texttt{employee}^+(y, z, t) \wedge \texttt{works}^+(y, u)$
   $\wedge \texttt{deptfile}(v) \wedge \texttt{dept-num}(v, u) \wedge \texttt{empcomp}(y, z, t, v, x)$,

3. for $\texttt{dept-manager}^+(x, y)$:

   $\exists z, t, u. \texttt{department}^+(z, t, u) \wedge \texttt{deptfile}(x) \wedge \texttt{dept-num}(x, z)$
   $\wedge (\exists z, t, v, w.\texttt{employee}^+(u, z, t) \wedge \texttt{works}^+(u, v)$
   $\wedge \texttt{empfile}(y) \wedge \texttt{emp-id}(u, y))$

4. for $\texttt{dept-name}^+(x, y)$:

   $\exists z, t.\texttt{department}^+(z, y, t) \wedge \texttt{deptfile}(x) \wedge \texttt{dept-num}(x, z)$

Summary

- code for update of an access path
  1. synthesized queries ψ^+, ψ^- 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 ψ^+, ψ^- 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)