Categorical database modeling and lenses

Bob Rosebrugh
(with M. Johnson, RJ Wood)

Department of Mathematics and Computer Science
Mount Allison University

FMCS 2015 / 2015-06-05
Outline

Part 1:
- Categorical database models
- Database problems: nulls, (view) updates, integration
- The view update problem and universality

Part 2:
- The view update problem and asymmetric lenses
- Symmetric lenses and model synchronization
- Symmetric lenses via spans
Database systems

- To store/access persistent information (vs computation systems) and include:
  - storage management
  - query optimizers
  - journalling/backup and recovery
  - access control

- Modern systems are mainly *relational*
  - Codd defined relational model in 1970; implemented post 1980
  - Store *only* the relation (aka table) data structure
  - Well developed theory (but *not* our theory of relations...)
  - Relational design often uses Entity-Relationship-Attribute (ERA) model
To avoid redundancy, inconsistency

*Data models* prescribe database schemas with type *and* constraint information:

- Relational: table headings; (foreign) keys
- ERA: type graph; various constraint decorations
- Sketch Data Model (SkDM): types/constraints from a mixed sketch

Other data models: various extensions of ERA; functional; several other versions using sketches
(Some) categorical database studies (with syntax)

- Lellahi-Spyratos 1990–: Towards a Categorial Data Model Supporting Structured Objects and Inheritance (mixed sketches)
- Dampney-Johnson-Monro 1992–: An illustrated mathematical foundation for ERA (type cats)
- Rosebrugh-Wood 1992: Relational databases and indexed categories
- Baclawski et al. 1994: A categorical approach to database semantics
- Diskin-Cadish 1995–: Algebraic graph-based approach. . . (mixed sketches) . . .
- Piessens 1995–: Categorical data specifications (mixed sketches)
- Lippe-ter Hofstede 1996: A category theoretical approach to conceptual data modelling (type cats)
- Tuijn-Gyssens 1996: A categorical graph-oriented object data model (type graphs)
- Johnson-Rosebrugh-Dampney-Wood 1997–: the Sketch Data Model
- Pierce et al. 2006–: Lenses and view update translation . . .
- D. Spivak et al. 2009–: Simplicial databases ($\Delta$), FQL
Database states and queries

- **Database state** for a schema is the currently stored information
  - In the relational model this is a set of relations (tables)
  - ERA speaks of ‘entity sets’, ‘relationship sets’, aiming at relational implementation
  - Categorical states: model functors.
- **Queries** extract information from a database state, also define access control and more
  - Relational queries expressed in “relational algebra”, return relations
  - ERA has no query language
  - Sketches often have a theory
ERA data model

- ERA diagrams have a graph with nodes:
  - Entities: type or class described by their
  - Attributes: which are typed
  - Relationships: between entities

- and edges that:
  - Join entities in a relationship; indicate subtyping
  - Possibly decorated: indicating constraints on relationships
    eg many-one (partial function); one-one (partial iso)

- Map to relational schemas having tables from:
  - Entities: column headings from attributes
  - Relationships: column headings from their entities
ERA example

Approach is relationship from Navaids to Runways

Directed edge means Approach “many-one”: partial function Navaids to Runways

is_a means subtype

Attributes not shown
Relational model

- Data definition language creates tables and constraints
- Database schema: a set of relation (table) schemas + integrity
- Relation schema: a list of attribute names with types:
  the column headings for tables + keys
- A relational database (state) is a set of relations (tables)
- Elements of relation (rows of a table) called tuples
Relational example

- Approach database schema with four relations:
  - $Airport(\text{ident}, \text{tower}, \text{weather})$
  - $Runway(\text{length}, \text{heading}, \text{atAirport})$
  - $Navaid(\text{operational}, \text{frequency}, \text{nrstAirport})$
  - $Approach(\text{fafMin}, \text{type}, \text{ptMin}, \text{toRunway}, \text{faf})$

- Attributes have types

- Integrity:
  - primary key: $\text{ident}$ for $Airport$, . . .
  - foreign key: $\text{atAirport}$ for $Runway$
    must match an $Airport$ primary key, . . .
Database access - relational model

- Access to tables is by ‘relational operators’
- SQL the standard *query language*
  (also a data definition language – schemas)
- Some standard operators:
  - select some rows of a table (WHERE in SQL syntax)
  - project on some columns (SELECT in SQL)
  - product of tables (FROM in SQL)
  - JOIN: tables on common columns = pullback along projections
  - UNION: tables with common schema
- e.g. SELECT `Approach.type`, `Runway.heading`
  FROM `Approach`, `Runway`
  WHERE `Approach.type` = 'ILS' AND
    `Runway.atAirport` = 'Syracuse'
- outputs the *ILS* approaches at *Syracuse*
- a projection from an equalizer contained in a product
Sketch data model

- Database schema is syntax: a sketch
- Database state is semantics: a model
- Models are a category
- Schemas related by sketch morphism (giving model substitution)

An **EA-sketch** $\mathcal{E} = (G, D, L, C)$ is a finite limit, finite sum sketch with

- a specified empty-base cone in $L$ (vertex is called 1); domain 1 arrows called elements
- attributes are vertices of cocones of elements
- non-attributes called entities
- the graph of $G$ is finite
- An EA sketch is **keyed** if each entity $E$ has a specified monic arrow $k_E : E \rightarrowto A_E$ to an attribute $A_E$

$D$, $L$ and $C$ express constraints often *not expressible* in other data models
Example
Example

Arrows are foreign keys; Monos (incl. primary keys) are pullback constraints (not shown) ERA/Relational data models cannot even express that: Squares commute; bottom square is a pullback; \( \text{Navaid} \cong VOR + ILS \)
Database states and properties

- A database state $S$ for EA sketch $\mathbb{E}$: a model of $\mathbb{E}$ in a lextensive $S$ (usually $S = \text{set}_0$)

- Category of database states of $\mathbb{E}$ is $\text{mod}(\mathbb{E}, S)$ which has several properties:
  - Attribute values always the same (up to iso).
  - $\text{mod}(\mathbb{E}, S)$ has pullbacks – computed pointwise
  - Keyed $\mathbb{E}$ makes $\text{mod}(\mathbb{E}, S)$ ordered
Implementation: *EASIK 3.0*

Entity-Attribute Sketch Implementation Kit  
Third release January 2015

- Graphical interface for EA sketch design
- Automatic SQL code generation
- MySQL (open source) back end
- Has support for diagrams of SQL views (hence limited)
Implementation: EASIK 3.0
Associated theory and query language

- EA sketch $\mathbb{E}$ is FS (finite sum) in the sense of Barr/Wells
- The associated FS theory $Q(\mathbb{E})$ is lextensive (finite lims and disjoint universal sums)
- $Q(\mathbb{E})$ constructed by closing $\mathbb{E}$ under finite limits and disjoint universal sums.
- Basic fact:

  \[ \text{mod}(Q(\mathbb{E})) \overset{\sim}{\longrightarrow} \text{mod}(\mathbb{E}) \]

- $Q(\mathbb{E})$ is the query language for the data design.
  - It contains, for example:
    - $\text{Airport} \times \text{Navaid}$, $\text{VOR} + \text{ILS}$ ...
    - also selections (= equalizers), joins, ...
  - $Q(\mathbb{E})$ is (the initial) model of $\mathbb{E}$ in lextensive categories
Updates are models too

- An update from $S$ to $S'$ is a delete, insert, modification or a composite of them
- Updates are (co)spans from $S$ to $S'$ in $\text{mod}(\mathbb{E}, S)$
- but

$$\text{spn}(\text{mod}(\mathbb{E}, S)) \sim \rightarrow \text{mod}(\mathbb{E}, \text{spn}(S))$$

- So one and the same EA sketch specifies all three of
  - the statics (models),
  - the queries $Q(\mathbb{E})$ and
  - the dynamics (updates)

for a data domain.
Incomplete information

- Missing/unknown information is common
- Not avoidable e.g. unknown/no phone number in address book
- SQL allows null - not a typed value, rather a mark/flag

SQL’s version:

- Entails a 3-valued “logic” e.g. A null implies [[A > B]] = unk
- Rather odd truth tables result e.g. unk AND unk =?
  unk AND false =?
- Quantifiers also need definitions

So in SQL some consequences are:

- \( P \ OR \ NOT \ P \) may not be true
- \( X = X \) may not be true
- Transitivity of equality fails

C. Date, others: allowing null and 3VL undermines integrity of the relational model
Incomplete information and SkDM

Three approaches to incomplete information in SkDM:
where ‘missing’ information only for some entity to attribute arrows $E \rightarrow A$

- add $1 \xrightarrow{null} A_f$ for specified $E \xrightarrow{f} A$
  adds null values and changes the sketch
- add $E \xleftarrow{i_f} E' \xrightarrow{f'} A$ for specified $E \xrightarrow{f} A$
  adds partial maps and changes the sketch
- take models in ‘lifted sets’ i.e. the lifts of discrete orders
  (changes meaning of model)

So SkDM

- provides three intrinsic solutions for incomplete information
- which can be Morita equivalent ($\cong$ model cats)
  (under fairly strong restrictions)
- and surprisingly, using lifted sets appears least suitable
  (at least for its effect on query language)
Database updates and views

Recall update

- changes database state(s)
- Examples: deletion, insertion, attribute modification

Either: modification of single state by delete or insert

or an update process: an endo $U$ of states, $S$

A view

- may limit access e.g. for security
- or present information to user class e.g. clerk
- or specify boundary for database integration
- View schema has derived types/constraints –

“Get” view states $V$ via the “view definition”

\[ G : S \rightarrow V \]
View update problem

When can an update to view state(s) either

▶ for single (view) state (e.g. formal insertion $a$):

$$\begin{align*}
S & \quad \downarrow \\
GS & \quad \xrightarrow{a} \quad V
\end{align*}$$

▶ for an update process (e.g. $U$):

$$\begin{align*}
S & \quad \downarrow \\
G & \quad \xrightarrow{U} \quad V \quad \rightarrow \quad V
\end{align*}$$
View update problem

When can an update to view state(s) either

- for **single (view) state** (e.g. insert \( a \)):

\[
S \rightarrow S' \\
\text{GS} \rightarrow^a V
\]

- for an **update process** (e.g. \( U \)):

\[
S \rightarrow S \\
\text{G} \downarrow \quad \downarrow \quad \text{G} \\
V \rightarrow V \\
\text{U} \quad \rightarrow
\]

propagate (or lift) correctly to full database update?
The **view update problem**: when can update to view state propagate correctly to underlying state?

- May be no solution
- May be many solutions, but no canonical
- Very restricted support in SQL
- Abstraction limits updatable views
- SkDM provides less abstract, universal updatability
Two views of updates and views

Updates (abstract and less):

- Bancilhon-Spyratos (1982, and others): An update is an endofunction on an *abstract set* of database states—an *abstract process* prescribing an updated database state

- SkDM: A single delete or insert update: a monic in the model category; general update is a (co)span

Views (abstract and less):

- 1980’s (B&S and several others): A view is a (surjective) function from an abstract set of database states to an abstract set of view states

- SkDM: A view compares the database syntax (sketch) to the view syntax (another sketch)
SkDM views

- View schema may have types/constraints from $E$, but also derived types/constraints—from $Q(E)$
- View (schema) for EA sketch $E$: an EA sketch $V$ and sketch morphism $V : V \rightarrow QE$.
- Obtain view state by substitution (model composition) from overlying state . . .
- Use $\text{mod}(E) \simeq \text{mod}(Q(E))$ and composition to define a “Get” functor:

$$V^* : \text{mod}(E) \rightarrow \text{mod}(V)$$

- so view state for view $V$ is model $V^*S$ for $V$
Views and database integration/interoperation

Views may describe a boundary:

\[ \mathbb{E} \xleftarrow{V} \mathbb{V} \xrightarrow{V'} \mathbb{E}' \]

so on models \( V^* : \text{mod}(\mathbb{E}) \to \text{mod}(\mathbb{V}) \leftarrow \text{mod}(\mathbb{E}') : (V')^* \)

Models for \( \mathbb{E}, \mathbb{E}' \) consistent if they agree at \( \mathbb{V} \)

Given an \( \mathbb{E} \) model, is there a consistent \( \mathbb{E}' \) model? (a lifting problem)

More generally, federated database from a colimit of sketches

A related, more studied issue ...
Abstract view update problem

Bancilhon and Spyrtos (1982, and others) studied the view update problem. For them:

- the Get is a surjective view definition abstract set mapping $G : S \rightarrow V$
- a view update is an endo-function (process) $U : V \rightarrow V$
- a translation $T_U$ of view update $U$ is a database update on $S$ lifting $UG$ through $G$

```
S ----> T_U ---> S
\( G \downarrow \)  \( U \)  \( G \downarrow \)
V ----> V
```

Will return to consider finding translations below…
View update problem - SkDM

- View states can be updated, so ...
- **View update problem**: when can update to a view state $V^*S$ propagate correctly to update of overlying ($\mathcal{E}$) state $S$?
- *or* if $T = V^*S$ updates to $T'$, is there $S$ to $S'$ update with $T' = V^*S'$?
- When is such $S'$ best possible? (which means?)
- Criteria on $V$, $V$ or $V^*$?
SkDM propagatability

Let \( V : \mathcal{V} \rightarrow Q \mathcal{E} \) a view schema and \( t : V^*S \rightarrow T' \) insert update (of view states)

- \( t \) propagatable if exists (insert) update in \( \text{mod}(\mathcal{E}) \), \( m : S \rightarrow S' \) such that:
  - \( V^*m = t \)
  - for \( \mathcal{E} \) state \( S'' \) and (insert) update \( m'' : S \rightarrow S'' \) with \( V^*m'' = t't \)
  - exists unique (insert) \( m' : S' \rightarrow S'' \) with \( V^*m' = t' \)
    (see next slides)

- If every insert update to \( T = V^*S \) propagatable, say that \( T \) is insert updatable.

- propagatable delete is dual.
SkDM propagatability

\[ V^* S \xrightarrow{t} T' \]
SkDM propagatability

\[ S \xrightarrow{m} S' \]

\[ V^* S \xrightarrow{t} T' \]
SkDM propagatability
SkDM propagatability
Criterion for updatability

- A view insert update $V^*S \rightarrow T'$ is propagatable precisely if it has an op-cartesian arrow.
- A view delete update $T' \rightarrow V^*S$ is propagatable when it has a cartesian arrow.
- So all delete (insert) updates are propagatable when $V^*$ is an (op-)fibration.
(Non)-Propagatable examples

- \( V : \mathbb{V} \rightarrow \mathbb{Q} \mathbb{E} \) is insert (delete) \textit{updatable at entity} \( w \in \mathbb{V} \) if all inserts (deletes) into (from) \( w \) are propagatable.

- An insert or delete \textit{at} \( w \) changes the database state’s value only at \( w \) — the values in the model of other entities and attributes remain unchanged.

Assume \( \mathbb{E} \) has no (co)cones except 1 and attributes. Assume \( V \) ‘injective’.

- If \( Vw \) is not initial node of a commutative diagram in \( \mathbb{E} \) and arrows out of \( Vw \) (in \( \mathbb{E} \)) are in image of \( V \), then \( V \) insert updatable at \( w \).

- If arrows into \( Vw \) are in image of \( V \), then \( V \) is delete updatable at \( w \).
Suppose that $V = \{ w \}$

- if $E$ has $f : Vw \rightarrow a$ where $a$ non-trivial attribute, then $V$ is not insert updatable at $w$.

- suppose $E$ has $f : b \rightarrow a$ and $A$ has element $a : 1 \rightarrow a$. Let $Vw$ the pullback of $f$ along $a$, then $V$ is insert updatable at $w$.

- if $E$ has discrete entities $a$ and $b$ and $Vw$ the sum of $a$ and $b$, $V$ is not insert updatable.

- if $E$ has discrete entities $a$ and $b$ and $Vw$ the product of $a$ and $b$. Then $V$ is not insert updatable.

- Suppose $V = \{ a_0 \rightarrow w \}$, $E$ has discrete entities $a$ and $b$, $Va_0 = a$ and $Vw$ the sum of $a$ and $b$. Then $V$ is insert updatable.
Sketch cofibrations

- In some cases we can guarantee full updatability, extending results of Street:

- sketch morphism \( \mathcal{V} : \mathcal{V} \to \mathcal{E} \) is a sketch embedding if:
  - graph morphism \( \mathcal{V} \) is injective on objects and edges
  - \( \mathcal{E} \) edge between nodes in \( \mathcal{V} \) image is in \( \mathcal{V} \) image (full)
  - \( \mathcal{E} \) comm diagram in \( \mathcal{V} \) image is from \( \mathcal{V} \) comm diagram
  - \( \mathcal{E} \) (co)cone in \( \mathcal{V} \) image is image of \( \mathcal{V} \) (co)cone
Sketch right cofibration: sketch embedding $V : \mathcal{V} \to \mathcal{E}$ such that
- $\mathcal{E}$ (co)cone with base or vertex node in $V$ image lies entirely in $V$ image
- no edge in $\mathcal{E}$ from node in image of $V$ to node not in the image of $V$

Sketch left cofibration: sketch embedding $V : \mathcal{V} \to \mathcal{E}$ such that
- $\mathcal{E}$ (co)cone with base or vertex node in $V$ image lies entirely in $V$ image
- no edge in $\mathcal{E}$ from node not in image of $V$ to node in image of $V$

Theorem
$V$ a sketch left (respectively right) cofibration then $V^* : \text{mod}(\mathcal{E}) \to \text{mod}(\mathcal{V})$ a left (respectively right) fibration.
Abstract view updates (again)

Bancilhon and Spyropoulos (and others) studied the view update problem. They consider that

- database states are a set $S$
- view states are a set $V$ – codomain of
- surjective view definition mapping $f : S \rightarrow V$
- view update is an endo-function $u : V \rightarrow V$

They consider a

- set $U$ of view updates:
- assumed complete: a monoid of endos
Translation

- **translation** $T_u$ of view update $u$ is database update: endo-function on $S$ such that $f(T_u(s)) = u(f(s))$

  \[ s \xrightarrow{T_u} T_u(s) \quad f(s) \xrightarrow{u} u(f(s)) \]

  and $T_u(s) = s$ if $u(s) = s$

- a **translator** $T$ for complete set of updates $U$ is translations \{ $T_u \mid u \in U$ \}
Updatability and complements

- Bancilhon and Spyropoulos: a translator $T$ for complete updates $U$ implies exists “constant complement” view $g : S \rightarrow C$:
  - $\langle f, g \rangle : S \rightarrow V \times C$ a bijection ($C$ complement of $V$)
  - $g(T_u(s)) = g(s)$ for $T_u \in T, s \in S$
    i.e. any $T_u$ is “constant” on $C$
- also showed a converse—constant complement view gives translator
SkDM and complements

- For views \( \mathcal{V} \xrightarrow{V} \mathcal{Q}\mathcal{E} \) and \( \mathcal{C} \xrightarrow{C} \mathcal{Q}\mathcal{E} \), say \( \mathcal{C} \) a complement of \( \mathcal{V} \) if

\[
\text{mod}(\mathcal{E}) \xrightarrow{\langle V^*, C^* \rangle} \text{mod}(\mathcal{V}) \times \text{mod}(\mathcal{C})
\]

is full, faithful and one-one on objects.

- We don’t need ess’ly surjective

- (Insert) update \( \alpha : V^* S \rightarrow T \) in \( \text{mod}(\mathcal{V}) \) has \( \mathcal{C} \)-constant update if exists \( \hat{\alpha} \) in \( \text{mod}(\mathcal{E}) \) with \( \alpha = V^* \hat{\alpha} \) and \( C^*(\hat{\alpha}) \) an iso.

A single update result:

**Theorem**

\( \mathcal{V} \xrightarrow{V} \mathcal{Q}(\mathcal{E}) \) a view, \( \mathcal{C} \xrightarrow{C} \mathcal{Q}\mathcal{E} \) a complement and \( \alpha : V^* S \rightarrow T \) an insertion in \( \text{mod}(\mathcal{V}) \). \( \alpha \) propagatable if it has a \( \mathcal{C} \)-constant update. Similar result for deletes.
An example

- Persons with name, department, project assignments
- No commutative diagrams

![Diagram]

- Attributes: $K_A$, Name, $K_D$ and $K_P$
- Entities Asst, Person, Dept and Proj
- Arrows $k_A$, $n$, $k_D$ and $k_P$ are keys
An example view

- View $V$ specified by inclusion of sketch with graph: below.

- Composites $np$ and $k_Pq$ not edges in $E$ but are in $Q(E)$.
View update may be propagatable with complement, but no ‘constant complement’

complement C for view V of the assignments database (example above) with graph:

\[
\begin{array}{c}
\text{Person} \\
\downarrow n \\
\text{Name} \\
\text{Dept} \\
\downarrow k_D \\
K_D \\
\text{Proj} \\
\downarrow k_P \\
K_P \\
\end{array}
\]

insertion of an assignment with new project value in V model is propagatable, but
cannot have a C-constant update (value at the entity Proj must change)
Pointed translation

Viewing updates as processes:

- **pointed view update** is $\langle U, u \rangle$ where

$$
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow U \\
\text{mod}(\mathbb{V}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow u \\
\text{mod}(\mathbb{V}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow 1 \\
\text{mod}(\mathbb{V}) \\
\end{array}

- A translation of $\langle U, u \rangle$ is $\langle L_U, l_u \rangle$ with $L_U$ endo functor on $\text{mod}(\mathbb{E})$ and $UV^* = V^*L_U$, $1 \xrightarrow{l_u} L_U$ natural and $uV^* = V^*l_u : V^* \longrightarrow V^*L_U$

$$
\begin{array}{c}
\text{mod}(\mathbb{E}) \\
\downarrow L_U \\
\text{mod}(\mathbb{E}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{E}) \\
\downarrow V^* \\
\text{mod}(\mathbb{E}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{E}) \\
\downarrow 1 \\
\text{mod}(\mathbb{E}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{E}) \\
\downarrow 1 \\
\text{mod}(\mathbb{E}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow U \\
\text{mod}(\mathbb{V}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow V^* \\
\text{mod}(\mathbb{V}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow 1 \\
\text{mod}(\mathbb{V}) \\
\end{array}
\quad
\begin{array}{c}
\text{mod}(\mathbb{V}) \\
\downarrow 1 \\
\text{mod}(\mathbb{V}) \\
\end{array}$
Universal translation

Translation $\langle L_U, l_u \rangle$ is universal when, for another translation $\langle L'_U, l'_u \rangle$ there is a unique $k : L_U \rightarrow L'_U$ such that $l'_u = kl_u$ and $V^*k$ is the identity on the identity functor on mod($\mathbb{V}$).

Theorem

Let $\mathbb{V} \xrightarrow{V} Q(\mathbb{E})$ a view and $\langle U, u \rangle$ a pointed view update. If $V^*$ is an opfibration, then there is a universal translation $\langle L_U, l_u \rangle$ of $\langle U, u \rangle$.

End of Part 1...
Recall... Part 1:
- Categorical database models
- Database problems: nulls, (view) updates, integration
- The view update problem and universality

Now, Part 2:
- The view update problem and asymmetric lenses
- Symmetric lenses and model synchronization
- Symmetric lenses and spans
View update problem

When can an update to view state(s) either

- for single (view) state (e.g. formal insertion $a$):

$$GS \xrightarrow{a} V$$

- for an update process (e.g. $U$):

$$V \xrightarrow{U} V$$
View update problem

When can an update to view state(s) either

- for single (view) state (e.g. insert $a$):

$$\begin{array}{c}
S - - - - - S' \\
\downarrow \quad \downarrow \\
GS \xrightarrow{a} V
\end{array}$$

- for an update process (e.g. $U$):

$$\begin{array}{c}
S - - - - - S \\
\downarrow \quad \downarrow \\
V \xrightarrow{U} V
\end{array}$$

propagate (or lift) correctly to full database update?
Asymmetric Lenses

(B. Pierce et al, 2005)
Consider a full database state $s$ and view state $G(s)$
When $G(s)$ updated to $v$, say, want strategy to find
updated full database state $s' = T_U s$ (over $v$):

$Idea$: provide a process $P : V \times S \rightarrow S$ called “Put” so that
$P(v, s)$ is the translated state $s'$ after $G(s)$ updated to $v$
Some equations should follow...

This structure, called a lens, provides translations

Also arose in considering “abstract models of storage”
(where there is a similar update problem)
Asymmetric Lenses

Let $\mathbf{C}$ be a category with finite limits

**Definition**

An asymmetric lens in $\mathbf{C}$ is $L = (S, V, G, P)$ [or just $(G, P)$] where

- $S$ and $V$ are $\mathbf{C}$ objects (... database states/view states)
- $S \xrightarrow{G} V$ aka ‘Get’ and $V \times S \xrightarrow{P} S$ aka ‘Put’

is called well-behaved (wb) if satisfying:

- PutGet: Get of Put is projection: $GP = \pi_0$ (or $GP(v, s) = v$)
- GetPut: Put for non-update is trivial $P\langle G, 1_S \rangle = 1_S$

and very well-behaved (vwb) if also satisfying:

PutPut: repeated Puts depend only on the last:

$P(1_V \times P) = P\pi_{0,2}$ (or $P(v', P(v, s)) = P(v', s)$)
the equations diagrammatically

\[ V \times S \xrightarrow{P} S \xleftarrow{\text{PutGet}} V \]
\[ S \xrightarrow{\langle G,1 \rangle} V \times S \]
\[ V \times V \times S \xrightarrow{1 \times P} V \times S \]
\[ V \times S \xrightarrow{P} V \]

So \( \Delta \Sigma G \xrightarrow{P} G \) is in \( \mathbf{C}/V \) where

\[ \Sigma \]
\[ \mathbf{C} \]
\[ \perp \]
\[ \Delta \]
\[ \mathbf{C}/V \]
And moreover . . .

**Proposition (JRW)**

A (vwb) lens has $P$ an algebra structure on $G$ in $\mathbf{C}/\mathbf{V}$ for the monad $\Delta \Sigma$ on $\mathbf{C}/\mathbf{V}$.

For vwb lenses:

- $\mathbf{C} = \textbf{set}$, $L = (G, P)$ recovers B&S results:
  
  $S \cong V \times C$, $G$ the projection, $C$ ‘complement’ of $V$, the translation: $T_U(s) := P(UGs, s)$

- $\mathbf{C} = \textbf{ord}$, recovers results of S. Hegner (2004)

- $\mathbf{C} = \textbf{cat}$: $G$ a projection and hence fibration and opfibration
Lenses compose

We can compose lenses:
if $L = (S, V, G, P)$ and $M = (V, W, H, Q)$ are lenses in $\mathbf{C}$
then $ML = (S, W, HG, R)$ is a lens, with the Put $R$ defined:

$W \times S \xrightarrow{1_W \times \langle G, 1_S \rangle} W \times V \times S \xrightarrow{\langle Q, 1_S \rangle} V \times S \xrightarrow{P} S$

Composites of $wb$, resp $vwb$, lenses are $wb$, resp $vwb$

There are identity on objects (ioo), \textit{non-full}
functors between categories of asymmetric lenses in $\mathbf{C}$

$\text{ALens}_v(\mathbf{C}) \longrightarrow \text{ALens}_w(\mathbf{C}) \longrightarrow \text{ALens}(\mathbf{C})$
Lenses preserved

Suppose $F : \mathbf{C} \to \mathbf{D}$ is a finite product preserving functor. For $L = (G, P)$ an asymmetric lens in $\mathbf{C}$, respectively: a well-behaved lens, very well-behaved lens $FL = (FG, FP)$ is an asymmetric lens in $\mathbf{D}$, respectively: a well-behaved lens, very well-behaved lens.

Moreover, $F$ preserves lens composition and we denote:

$$F : \text{ALens}(\mathbf{C}) \to \text{ALens}(\mathbf{D})$$

respectively from $\text{ALens}_w(\mathbf{C})$ and $\text{ALens}_v(\mathbf{C})$. 
Lenses and pullbacks

Proposition

For an asymmetric lens \( L = (G, P) \) and \( H : V' \to V \) in \( C \) pulling back \( G \) along \( H \) in \( C \)

\[
\begin{array}{c}
S \\
\downarrow G \\
V \\
\downarrow H \\
V' \\
\end{array}
\quad \begin{array}{c}
T \\
\downarrow G' \\
S \\
\downarrow H' \\
V' \\
\end{array}
\]

gives the Get for asymmetric lens \( L' = (G', P') \) with \( P' = \langle P(H \times H'), \pi_0 \rangle \)

Similarly for well-behaved and very well-behaved lenses

But: \( \text{ALens}(C), \text{ALens}_w(C), \text{ALens}_v(C) \) may not have pullbacks.
Less abstract lenses

For a Get (view functor) in \textbf{cat} denoted \( G : S \to V \) we prefer that (insert) view updates needing lifts should be morphisms \( GS \to V \), objects of \((G, 1_V)\)
(They were simply pairs \((V, S)\) in \(V \times S\) for the lenses above)

Thus, the \textbf{domain} of a less abstract Put \( P \) for \( G \) should be objects \((S, GS \to V)\) in \((G, 1_V)\)
Sorry about changing the order of arguments...
Values of \( P \) should be morphisms of \( G ?? \)
But universality allows them to be objects and \( P : (G, 1_V) \to S \)

First some notation...
A monad

Right comma projection $R(-)$ is functor part of a monad

$$R : \text{cat}/V \rightarrow \text{cat}/V$$

with unit component $G \xrightarrow{\eta_G} RG$ defined by

$$\eta_G = (1_V, G, 1_G) : S \rightarrow (G, 1_V)$$
defined universally.
A monad

and multiplication \( RRG \xrightarrow{\mu_G} RG \) defined by:

\[
\begin{array}{c}
\mu_G = (L_G 1_v \cdot L_{RG} 1_v, RRG, \beta(\alpha L_{RG} 1_v)) : (RG, 1_v) \to (G, 1_v)
\end{array}
\]

For later: left comma projection \( L(-) \) is functor part of a monad

\[
L : \text{cat}/V \to \text{cat}/V
\]

with \( LG : (1_v, G) \to V \)
And an iterate of a $P$

For $G : S \rightarrow V$ consider a $P : (G, 1_V) \rightarrow S$ satisfying $GP = RG$, so that

$GPL_{RG} 1_V = RG \cdot L_{RG} 1_V \xrightarrow{\beta} RRG$, define: $(P, 1_V)$ by
c-Lenses

Again, for a view in \textbf{cat}, $G : S \rightarrow V$
the “Put” for (insert) view updates $GS \rightarrow V$ should be
a process $P : (G, 1_V) \rightarrow S$, and we define:

**Definition**
A c-lens in \textbf{cat} is $L = (S, V, G, P)$, or just $(G, P)$
satisfying

- c-PutGet: $GP = RG$
- c-GetPut: $P\eta_G = 1_S$
- c-PutPut: $P\mu_G = P(P, 1_V)$

Compare with the original (vwb) lens equations... Could model
delete updates $V \rightarrow GS$, then “Put” s.b.
$P : (1_V, G) \rightarrow S$ using $LG$ in the PutGet equation...
c-Lenses are opfibrations

or diagrammatically:

Recalling that an algebra structure for the monad

\[ \text{cat} / V \xrightarrow{R} \text{cat} / V \]

is a split opfibration:

**Proposition (JRW)**

*For a c-lens \( L = (S, V, G, P) \) in \( \text{cat} \), \( P \) is an algebra structure for \( R \) so \( G \) is a split opfibration.*

65
**c-Lenses compose**

Opfibrations compose, so if $G : S \to V$ and $G' : V \to W$ are c-lenses, so is $G'G : S \to W$

Subcategory of $\text{cat}$ with arrows c-lenses is denoted $\text{ACLens}$. Asymmetric lens in $\text{cat}$ is a c-lens, so $\text{ALens}_v(\text{cat})$ is a subcategory.

Further, opfibrations pull back (along any functor) and a cospan of c-lenses gives span of c-lenses. Our interest in spans actually motivated by cospan of views $G, H$:

![cospan diagram]

giving a span of views $G', H'$ (of c-lenses if $G, H$ are)

Recall integration (for interoperation) of databases via views
The other side...

c-lenses are about inserts, what about deletes?
As noted, can use \( L \), but general updates are spans
Moreover, BX (lens) theorists don’t much like PutPut
But really not worried about composing inserts, or deletes
Rather concerned about “Mixed Put-Put”

Our resolution from a distributive law

\[
LR \xrightarrow{\lambda} RL
\]

For \( G : S \to V \) the functor \( LRG : (1_V, RG) \to V \)
domain objects of the form \( GS \xrightarrow{a} V \xleftarrow{b} V' \),
cospans \((a, b)\) from \( GS \) to \( V' \) and \( LR(G)(a, b) = V' \).

Now \( \lambda_G(GS \xrightarrow{a} V \xleftarrow{b} V') \) is the pullback of the cospan \((a, b)\).
The other side...

An RL algebra is R- and L-algebras $RG \xrightarrow{r} G$ and $LG \xrightarrow{l} G$ satisfying:
for any object $S$ in $S$ and any pullback (in $V$):

\[
\begin{array}{ccc}
V & \xrightarrow{i} & GS \xrightarrow{k} W \\
\downarrow{j} & & \downarrow{m} \\
V' & \xrightarrow{m} & W
\end{array}
\]

it is the case that $r(I(i, S), j) \cong I(m, r(S, k))$

Thus, if we can update both inserts ($R$ transitions) and deletes ($L$ transitions) \textit{and} those updates satisfy the condition, then we can update spans – arbitrary compositions of $R$ and $L$ transitions.
Another categorical version of lenses

Motivated by similar considerations Z. Diskin and co-authors called updates or morphisms deltas, made the set of deltas the domain of Put – now returning a delta (morphism) – with axioms similar to c-lenses

An (asymmetric) delta lens (d-lens) in \textbf{cat} is \( L = (S, V, G, P) \) where \( G : S \rightarrow V \) is a functor and \( P : \left| (G, 1_V) \right| \rightarrow |S^2| \) is a function and the data satisfy:

(i) d-PutInc: the domain of \( P(S, \alpha : GS \rightarrow V) \) is \( S \)

(ii) d-PutId: \( P(S, 1_{GS} : GS \rightarrow GS) = 1_S \)

(iii) d-PutGet: \( GP(S, \alpha : GS \rightarrow V) = \alpha \)

(iv) d-PutPut:

\[
P(S, \beta \alpha : GS \rightarrow V \rightarrow V') = P(S', \beta : GS' \rightarrow V')P(S, \alpha : GS \rightarrow V)
\]

where \( S' \) is the codomain of \( P(S, \alpha : GS \rightarrow V) \)
ADLens

Proposition
If \( L = (S, V, G, P) \) and \( M = (V, W, H, Q) \) are \( d \)-lenses then
then \( ML = (S, W, HG, R) \) is a \( d \)-lens, with \( R \) as
\[
|(HG, 1_W)| \xrightarrow{Q} |(G, 1_V)| \xrightarrow{P} |S|^2
\]
Identity functor is Get for a \( d \)-lens and unitary for composition.
Denote the resulting category ADLens

Proposition
If \( L = (S, V, G, P) \) is a \( d \)-lens and \( F : V' \rightarrow V \) is a functor then
\( G' \) in the pullback (in \text{cat}) is the Get of a \( d \)-lens
c-Lenses and d-Lenses

For $G : \mathbf{S} \to \mathbf{V}$, denote $G_0 = |\mathbf{S}| \to \mathbf{S} \xrightarrow{G} \mathbf{V}$ and $R_0 G : (G_0, 1_\mathbf{V}) \to \mathbf{V}$

Semi-monad $(R_0, \mu^0)$ on $\text{cat/\mathbf{V}}$ similar to $R$, and transformation $\eta^0$ to $R_0$ (from functor sending $G$ to $G_0$)

Proposition

If $L = (\mathbf{S}, \mathbf{V}, G, P)$ is a d-lens then $(G, P_0)$ is an $(R_0, \mu^0)$ algebra satisfying $P_0 \eta^0 G = P_0 \eta_{G_0} = I_\mathbf{S}$, and conversely.

Corollary

A c-lens is a d-lens; composition is compatible.

Though not every d-lens is a c-lens
Categories of asymmetric lenses

In summary:

\[ \text{ALens}_v(\text{set}) \rightarrow \text{ALens}_v(\text{cat}) \rightarrow \text{ACLens} \rightarrow \text{ADLens} \]

\[ \text{ALens}_v(\text{ord}) \]

All admit the \( Sp(U) \) construction which follows...
The $Sp(U)$ Construction

$C$ with finite limits; $U : A \rightarrow C$ ioo functor reflecting isos
(We are thinking $ALens \rightarrow C$)
Assume an operation $P$ on $C$ cospans

$$B \overset{g}{\rightarrow} C \overset{U(r)}{\leftarrow} D$$

giving arrows $P(g, r)$ in $A$ such that
1) there is in $C$ a pullback:

$$\begin{array}{ccc}
B & \overset{g}{\rightarrow} & C \\
\downarrow{g} & & \downarrow{U(r)} \\
C & \overset{U(r)}{\leftarrow} & D \\
\end{array}$$

with $t' = U(r')$ where $r' = P(g, r)$

And...
The $Sp(U)$ Construction

2) If also $g = U(v)$ then for $v' = P(G(r), v)$ the square commutes (in $A$):

![Diagram]

Next, given $U$ and operation $P$, define category $Sp(U)$:
Objects of $A$ (or $C$)
Arrows $\equiv_U$ equiv classes of spans in $A$ where
The $Sp(U)$ Construction

$\equiv_U$ generated by span morphisms in $A$

\[ \begin{array}{ccc}
A & \overset{u}{\leftarrow} & C & \overset{v}{\rightarrow} & B \\
\downarrow & & \downarrow t & & \downarrow \\
D & \underset{u'}{\leftarrow} & C & \underset{v'}{\rightarrow} & A
\end{array} \]

with $u = u't$ and $v = v't$ and $G(t)$ split epi.

$Sp(U)$ composition by span composition in $C$

**Proposition**

*With the data just defined, $Sp(U)$ is a category.*
Proposition

Let $G : S \rightarrow V \leftarrow W : H$ be a cospan of functors and $(G, P)$ a $d$-lens. Then, in the pullback square in $\text{cat}$:

\[
\begin{array}{ccc}
S & \rightarrow & T \\
\downarrow & & \downarrow \\
V & \leftarrow & W
\end{array}
\]

the functor $G'$ together with $P' : |(G', 1_W)| \rightarrow |T^2|$ defined by $P'((S, W), \beta : G'(S, W) \rightarrow W') = (P(S, H(\beta)), \beta) : (S, W) \rightarrow (S', W')$ define a d-lens from $T$ to $W$.

So $Sp(U)$ applies...
Symmetric lenses

(Hoffman, Pierce and Wagner, 2011)

Idea: Describe **re-synchronization** for model classes (of states) \(X, Y\) having synchronization (“complement”) information from \(C\).

Given states \(x, y\) synchronized by a complement \(c\) and an (updated) state \(x'\) of \(X\), determine re-synchronizing complement \(c'\) from \((x', c)\) and an updated \(y'\) of \(Y\) (and vice versa)

So an arrow \(r: X \times C \rightarrow Y \times C\) and vice versa.

\[
\begin{array}{c}
X \times C \\
\downarrow r \\
Y \times C \\
\downarrow l \\
X \times C
\end{array}
\]

Now \((x', c', y')\) is (re)synchronized. Think \(y'\) as a **Put** of \((x', y)\)... Some equations are expected because:

if \(l\) applied to \((y', c')\) then the result should be \((x', c')\)
Example (from H,P,W)

The data in states \(x, y\) might initially be the following

\[
\begin{align*}
x &: \\
\text{Schubert} & 1797-1828 & \text{Schubert} & \text{Austria} \\
\text{Schumann} & 1810-1856 & \text{Schumann} & \text{Germany}
\end{align*}
\]

with initial complement, “hidden data” (a \(C\) state):

\[
\begin{align*}
\text{c} &: \\
1797-1828 & \text{Austria} \\
1810-1856 & \text{Germany}
\end{align*}
\]

An edit to \(x\) gives new \(X\) state \(x'\):

\[
\begin{align*}
x' &: \\
\text{Schubert} & 1797-1828 \\
\text{Schumann} & 1810-1856 \\
\text{Monteverdi} & 1567-1643
\end{align*}
\]

then applying \(r(x', c)\) results in new \(C\) and \(Y\) states:

\[
\begin{align*}
\text{c'} &: \\
1797-1828 & \text{Austria} \\
1810-1856 & \text{Germany} \\
1567-1643 & \text{?country}
\end{align*}
\]

\[
\begin{align*}
\text{y'} &: \\
\text{Schubert} & \text{Austria} \\
\text{Schumann} & \text{Germany} \\
\text{Monteverdi} & \text{?country}
\end{align*}
\]
Symmetric lenses

Let $\mathbf{C}$ be a category with finite limits.

For objects $X$, $Y$ in $\mathbf{C}$, an rl lens from $X$ to $Y$ is denoted $L = (X, Y, C, r, l)$ with $C$ an object of “complements” and morphisms

$$r : X \times C \to Y \times C \quad \text{and} \quad l : Y \times C \to X \times C$$

satisfying the equations:

$$\pi_X lr = \pi_X : X \times C \to X \quad \pi_C lr = \pi_C r : X \times C \to C \quad \text{(PutRL)}$$
$$\pi_Y rl = \pi_Y : Y \times C \to Y \quad \pi_C rl = \pi_C l : Y \times C \to C \quad \text{(PutLR)}$$

HPW require an element $m : 1 \to C$ where $m$ is for “missing” (called pc-symmetric below)
Symmetric lenses decompose

Remark
For an RL lens \( L = (X, Y, C, r, l) \) in \( C \), the equations \( rlr = r \) and \( lrl = l \) hold.

Suppose that \( L = (X, Y, C, r, l) \) is an rl lens in \( C \).
Let \( e : S_L \rightarrow X \times Y \times C \) be an equalizer of \( r_{\pi_0,2} \) and \( \pi_{1,2} \).
If \( C = \text{set} \),
\[ S_L = \{ (x, y, c) | r(x, c) = (y, c) \} = \{ (x, y, c) | l(y, c) = (x, c) \} \]
Elements of \( S_L \) are the “synchronized triples”
Symmetric lenses decompose

For $L, S_L$ as above:

**Proposition**

There is a span

$$L_I : X \leftarrow S_L \rightarrow Y : L_r$$

in $\text{ALens}_w$ from $X$ to $Y$ with $\text{Gets}$ defined by $g_I = \pi_X e, g_r = \pi_Y e$.

The $\text{Put.}$ $p_I$ for $L_I$ ($p_r$ similar) is defined by

$$X \times S_L \xrightarrow{1_X \times 1_C} X \times X \times Y \times C \xrightarrow{\pi_{0,3}} X \times C \xrightarrow{1_X \times r} S_L$$

(The set formula for $p_I$ is $p_I(x', (x, y, c)) = (x', r(x', c))$.)

Denote the span $(L_I, L_r)$ by $A(L)$

Recalling $U_w : \text{ALens}_w \rightarrow \mathbf{C}$, define $\text{SLens}_w = \text{Sp}(U_w)$
Symmetric lenses compose

For rl lenses \( L_1 = (X, Y, C_1, r_1, l_1) \) and \( L_2 = (X, Y, C_2, r_2, l_2) \):

\( L_1 \sim L_2 \) if there exists a well-behaved asymmetric lens \( L = (C_1, C_2, t, p) \) with \( t \) a split epi and respecting \( L_1, L_2 \) operations, which means:

\[
\begin{align*}
    r_2(X \times t) &= (Y \times t)r_1 \\
    l_2(Y \times t) &= (X \times t)l_1
\end{align*}
\]

and

\[
\begin{align*}
    r_1(X \times p) &= (Y \times p)(r_2 \times C_1) \\
    l_1(Y \times p) &= (X \times p)(l_2 \times C_1).
\end{align*}
\]

\( \sim \) generates equivalence relation on rl lenses \( X \) to \( Y \) denoted \( \equiv_{rl} \)

\( \equiv_{rl} \) class of \( L \) denoted \( [L]_{rl} \).
**Symmetric lenses compose**

\[ L = (X, Y, C, r, l), \quad M = (Y, Z, C', r', l') \text{ rl lenses} \]

Their *rl-composite lens* is \( ML = (X, Z, C'', r'', l'', m'') \)

where \( C'' = C \times C' \) and

\[ r'' = \langle \pi_{0,2}, \pi_1 \rangle (r' \times 1_C) \langle \pi_{0,2}, \pi_1 \rangle (r \times 1_{C'}) \quad (l'' \text{ similar}) \]

**Proposition**

For rl lenses \( L_1, L_2 \) from \( X \) to \( Y \) and \( M_1, M_2 \) from \( Y \) to \( Z \) in \( C \), if \( L_1 \equiv_{rl} L_2 \) and \( M_1 \equiv_{rl} M_2 \) then \( M_1 L_1 \equiv_{rl} M_2 L_2 \).

**RLLens** has objects of \( C \); arrows \( X \) to \( Y \) are \( \equiv_{rl} \) classes

**Proposition**

There is an identity on objects functor

\[
A : \text{RLLens} \longrightarrow \text{SLens}_w
\]

*defined by* \( A([L]_{rl}) = [A(L)]_{U_w} \).
Symmetric lenses from asymmetric

Going the other way... From span of wb asymmetric lenses $L = (S, X, G_X, P_X)$, $M = (S, Y, G_Y, P_Y)$, construct rl lens $S(L, M) = (X, Y, S, r, l)$ where (in set)

$$r(x', (x, y, c)) = (G_Y P_X(x', (x, y, c)), P_X(x', (x, y, c))) \ (l \text{ similar})$$

Proposition

Denote $AS(L, M)$ by $L_l : X \leftarrow S_L \rightarrow Y : L_r$. There is iso span morphism $g : S \rightarrow S_L$, so $AS(L, M) \equiv_{U_w} (L, M)$,
Categories of symmetric lenses

Proposition
If \( L : X \leftarrow S \rightarrow Y : M, L' : X \leftarrow S' \rightarrow Y : M' \) are \( \equiv_{U_w} \) equivalent spans of well behaved asymmetric lenses then
\[ S(L, M) \equiv_{rl} S(L', M') \] and \( S([(L, M)]_{U_w}) = [S(L, M)]_{rl} \) defines functor \( S : SLens_w \rightarrow RLLens \).

Theorem
\( SLens_w \) is a retraction of \( RLLens \) via \( A \) and \( S \).
Hofmann, Pierce and Wagner introduced an equivalence relation we denote $\equiv_{pc}$ on their pc-symmetric lenses from $X$ to $Y$

$\equiv_{pc}$ allows well-defined composition of pc-symmetric lenses giving $\text{pcLens}$

Starting from rl lenses, suitably adding points so that $\equiv_{Uw}$ can be compared, we can show that $\equiv_{pc}$ is in fact coarser than $\equiv_{Uw}$
Symmetric delta lenses (Diskin et al. 2011/12)

For symmetric version of d-lens, again use morphisms for updates:

Let $\textbf{X}$ and $\textbf{Y}$ be small categories.

Given a delta or update $x : X \rightarrow X'$ in $\textbf{X}$ from state $X$ where $X$ synchronized with $Y$ by “correspondence” $r : X \leftrightarrow Y$, symmetric d-lens should deliver an update $y : Y \rightarrow Y'$ in $\textbf{B}$ and, as for rl-lenses, a re-synchronization $r' : X' \leftrightarrow Y'$:

\[
\begin{array}{c}
X & \xleftarrow{r} & Y \\
\downarrow x & & \downarrow y \\
X' & \xleftarrow{r'} & Y'
\end{array}
\]
Symmetric delta lenses

A symmetric delta lens (fb-lens) from \( X \) to \( Y \) is \( L = (\delta_X, \delta_Y, f, b) \) with a span of sets

\[
\delta_X : |X| \leftarrow R_{XY} \rightarrow |Y| : \delta_Y
\]

(elements of \( R_{XY} \) called corrs are denoted \( R : X \leftrightarrow Y \)) and forward and backward propagation operations

\[
f : Arr(X) \times |X| R_{XY} \rightarrow Arr(Y) \times |Y| R_{XY}
\]

\[
b : Arr(X) \times |X| R_{XY} \leftarrow Arr(Y) \times |Y| R_{XY}
\]
satisfying obvious equations, so that
Symmetric delta lenses

display instances of propagation operations as:

\[
\begin{array}{ccc}
X & \xleftarrow{R} & Y \\
\downarrow x & & \downarrow y \\
X' & \xleftarrow{R'} & Y'
\end{array}
\quad
\begin{array}{ccc}
X & \xleftarrow{R} & Y \\
\downarrow x & & \downarrow y \\
X' & \xleftarrow{R'} & Y'
\end{array}
\]

where \( f(x, R) = (y, R') \) and \( b(y, R) = (x, R') \)...

we have propagation respects identities: \( R : X \leftrightarrow Y \) implies
\( f(\text{id}_X, R) = (\text{id}_Y, R) \) and \( b(\text{id}_Y, R) = (\text{id}_X, R) \)
and composition in \( X \) and \( Y \):
\( f(x'x, R) = f(x', \pi_1(f(x, R))) \), similarly for \( b \).
Composite symmetric delta lenses

For fb-lenses
\[ L = (\delta^R_X, \delta^R_Y, f^R, b^R), \quad L' = (\delta^S_Y, \delta^S_C, f^S, b^S) \] and \( T_{XZ} \) the pullback in

\[ \begin{array}{c}
\delta_1 \\
\delta^R_Y \\
\delta^S_Y
\end{array} \xrightarrow{T_{XZ}} \begin{array}{c}
\delta_2 \\
| \mathcal{Y} | \\
\delta^S_C
\end{array} \]

The composite fb-lens \( L'L = (\delta_X, \delta_Z, f, b) \) has
\[ \delta_X = \delta^R_X \delta_1, \quad \delta_Z = \delta^S_Z \delta_2 \] and
\[ f(x, (R, S)) = (z, (R', S')) \] where \( f^R(x, R) = (y, R'), f^S(y, S) = (z, S') \)

\( b \) similarly
Back and forth

Let

\[ L = (G_L, P_L), \quad K = (G_K, P_K) \]

a span of d-lenses with

\[ G_L : V \leftarrow S \rightarrow W : G_K \]

Construction: \[ M_{L,K} = (\delta_V, \delta_W, f, b) \]

an fb-lens:

- the corrs are \( R_{V,W} = |S| \)
- \( \delta_V S = G_L S \) and \( \delta_W S = G_K S \)
- for \( v : V \rightarrow V', \ S : V \leftrightarrow W \) set \( f(v, S) = (w, S') \)
  where \( w = G_K(P_L(S, v)) \), \( S' \) codomain \( P_L(S, v) \);
- \( b \) is defined analogously.
Back and forth

Let $M = (\delta_V, \delta_W, f, b)$ an fb-lens $V$ to $W$, corrs $R_V, W$

Construction: span of d-lenses $L_M : V \leftarrow S \rightarrow W : K_M$

First the head category $S$:

- objects are $R_V, W$, the corrs of $M$
- morphisms from $R$ to $R'$,

$$\{(v, w) | d_0 v = \delta_V(R), d_1 v = \delta_V(R'), d_0 w = \delta_W(R), d_1 v = \delta_W(R')\}$$

i.e. $(v, w)$ a formal square:

\[
\begin{array}{ccc}
V & \overset{R}{\leftarrow} & W \\
\downarrow v & & \downarrow w \\
V' & \overset{R'}{\leftarrow} & W'
\end{array}
\]

- Composition inherited from $V$ and $W$:

$$(v', w')(v, w) = (v'v, w'w) \text{ if } (v, w) \in S(R, R'), (v', w') \in S(R', R'')$$
Back and forth

$L_M$ is defined as $(G_L, P_L)$ where

1. $G_L : S \rightarrow V$ has $G_L(R) = \delta_V(R)$, $G_L(v, w) = v$
2. $P_L : \left| (G_L, 1_V) \right| \rightarrow \left| S^2 \right|$ has $P_L(R, v) = (v, w)$

where $f(v, R) = (w, R')$

$K_M = (G_K, P_K)$ is similar.

The two constructions are closely related.
One composite is actually the identity:

**Proposition**

*For any fb-lens $M$, we have*

\[ M = M_{L_M, K_M} \]
Two equivalence relations (1)

First: An equivalence relation on spans of d-lenses \(X\) to \(Y\):

\[
\begin{align*}
X & \xleftarrow{(G_L,P_L)} S \xrightarrow{(G_R,P_R)} Y \quad \text{and} \\
X & \xleftarrow{(G'_L,P'_L)} S' \xrightarrow{(G'_R,P'_R)} Y
\end{align*}
\]

Say functor \(\Phi : S \rightarrow S'\) satisfies conditions (E) if:

1. \(G'_L \Phi = G_L\) and \(G'_R \Phi = G_R\),
2. \(\Phi\) surjective on objects and
3. whenever \(\Phi S = S'\) we have both
   \[
   P'_L(S', G'_L S' \xrightarrow{\alpha} X) = \Phi P_L(S, G_L S \xrightarrow{\alpha} X) \quad \text{and} \\
P'_R(S', G'_R S' \xrightarrow{\beta} Y) = \Phi P_R(S, G_R S \xrightarrow{\beta} Y).
   \]

Thus

1. says \(\Phi\) a cell between \(X\) to \(Y\) spans
2. expresses a compatibility with Puts.
3. But, \(\Phi\) need not be a lens
Two equivalence relations (1)

Definition
≡_{Sp} the equiv rel’n on spans of d-lenses X to Y generated by Φ satisfying (E).

Lemma
(1) Composite of d-lens span cells satisfying (E) satisfies (E).
(2) Suppose S, S', S'' heads of d-lens spans X to Y and Φ, Φ' satisfy(E) in the cat pullback:

```
  Ψ   T   Ψ'
S ← S' ← S''
  Φ   Φ'
```

Then T the head of d-lens span X to Y; Ψ and Ψ' satisfy (E).

Corollary
Zig-zag of span cells satisfying (E) from span of cells satisfying (E). Reducing proof of ≡_{Sp} to a single span
Second: An equivalence relation on fb-lenses from $X$ to $Y$.

Definition

$L = (\delta_X, \delta_Y, f, b)$ and $L' = (\delta'_X, \delta'_Y, f', b')$ fb-lenses, corrs $R_{XY}, R'_{XY}$. Define $L \equiv_{fb} L'$ iff exists relation $\sigma \subseteq R_{XY} \times R'_{XY}$ with:

1. $\sigma$ compatible w $\delta$’s: $R \sigma R'$ implies $\delta_X R = \delta'_X R'$ and for $\delta_Y$
2. $\sigma$ projections surjective.
3. $R \sigma R'$ then $f(x, R) = (y, S)$, $f'(x, R') = (y', S')$ implies $y = y'$, $S \sigma S'$
4. corresponding condition for $b$

Condition 3. says one $X$ update from related corrs delivers same $Y$ update and related corrs.
Two equivalence relations (2)

Remark
\[ \equiv_{\text{fb}} \text{ generated by zig-zags of certain surjections, } \]
\[ \text{but any such zig-zag reducible to single span of such} \]

Proposition
Suppose \( M \equiv_{\text{fb}} M' \) are fb-lenses \( X \) to \( Y \) equivalent by generating
surjection. Then \((L_M, K_M) \equiv_{\text{Sp}} (L_{M'}, K_{M'})\) as spans of d-lenses

Proposition
Suppose \((L, K) \equiv_{\text{Sp}} (L', K')\) equivalent d-lenses spans by
generating \( \Phi \). Then \( M_{L,K} \equiv_{\text{fb}} M'_{L',K'}\) as fb-lenses.
Two equivalence relations (2)

Cutting to the chase...

Both $\equiv_{Sp}$ and $\equiv_{fb}$ are compatible with composites
Their equivalence classes are arrows of categories denoted $SpDLens$ and $fbDLens$
Define functors

$$S : SpDLens \rightarrow fbDLens$$
$$A : fbDLens \rightarrow SpDLens$$

with actions (up to equivalence):

$$L, K \mapsto M_{L,K} \quad M \mapsto L_M, K_M$$

Proposition

*The functors $A$ and $S$ are an isomorphism of categories.*

Proof from $M = M_{L_M,K_M}$ noted above and showing

$$L, K \equiv_{Sp} L_{M_{L,K}}, K_{M_{L,K}}$$

So once again, symmetric lenses arise from spans of asymmetric lenses
Some further points/questions

- RLLens is a non-full subcategory of fbDLens
- What corresponds to SpCLens?
- What corresponds to the inclusion of “Edit lenses”?
- Properties of fbDLens?
- Should a two-dimensional structure be considered?
Conclusion

- Sketch Data Model expressive syntax, semantics
- View update problem gets universal solution
- Asymmetric lenses provide solutions to the view update problem in several contexts
- Symmetric lenses describe model synchronization processes also in various contexts
- Symmetric lenses arise via spans of asymmetric lenses and often arise from cospans

- Some urls:
  - www.mta.ca/~rrosebru
  - www.comp.mq.edu.au/~mike/
Thanks!