Information Systems and the Physical World

Nick Rossiter
nick.rossiter1@btinternet.com
http://computing.unn.ac.uk/staff/cgnr1

Mike Heather
Dimitris Sisiaridis

CEIS
Northumbria University

ANPA 31 (9-13 August 2010)
Cambridge
Outline

• Formal representations of real world
  • Based on information systems
  • Look at underlying assumptions
    - How questionable are they?
  • Consider maths in terms of underlying physics
    - Increases our confidence

• Review formal structures
  • Locally Cartesian closed category (LCCC)
    • Underlying data structures
  • Cartesian monad
    • Unification of categorial structures and manipulation
Formal Representation

• Based very much on
  • Cartesian closed category (CCC)
    - Connectivity (exponential)
    - Product (prerequisite for relationships)
    - Initial object (unique starting point)
    - Terminal object (unique finishing point)
  • Fits in with philosophy
    - Everything is connected
    - Everything is related
    - Everything is limited
LCCC

• In practice we use a variant of Cartesian closed categories
  • Locally Cartesian closed category (LCCC)
    – Product is replaced by a relationship
  • Product is all possible pairs
    – e.g. account number X borrower name (A X B)
  • Relationship is those pairs that satisfy a particular context
    – e.g. account number X borrower name in the context of cash owed (A X₂ B)
• In category theory this is a pullback (with adjointness properties)
Pullback

A \times_C B \rightarrow A \rightarrow C

C is A + B + C
∃ is an equaliser: $\exists = \iota_l \circ \pi_l = \iota_r \circ \pi_r$
Pullback

Adjointness requirements $\exists \vdash \Delta$ and $\Delta \vdash \forall$
Working Assumption

• The Pullback has underpinned much of our work on information systems
• But is this justified?
• Information systems are open ended.
• We cannot prove all our instances of data are pullbacks.
• But we can try to relate pullbacks to accepted practice in software engineering.
Software Engineering Principles

- Information system data design
  - Normalisation Commonly to 3NF (third normal form)
- Process design
  - High coherence
  - Low coupling
  - Transaction
- How do these concepts relate to LCCC?
- LCCC have been popular in theoretical computing science
  - But little attempt to handle design issues
Normalisation Outline

• A relation comprises a collection of attributes
  • e.g. delivered (customer_id, customer_name, customer_address, item_code, driver_id, driver_name)

• Decide on those that provide uniqueness and make these the key
  • customer_id, item_code

• The others become non-key
  • customer_name, customer_address, driver_id, driver_name

• Requires knowledge of how things are done physically
Normalisation Stages

• Then check validity against 3 forms of increasing severity:
  
  • 1NF: for relation R each non-key attribute is functionally dependent on the key
  
  • 2NF: R is in 1NF and each non-key attribute is fully functionally dependent on the key (not dependent on any component of key)
  
  • 3NF: R is in 2NF and no non-key attribute is functionally dependent on another non-key attribute

• Maths in set theory is convoluted – students find it challenging. e.g. Ullman, J D, Principles of Database and Knowledge-base Systems (1988).

• Some category theory work has tried to directly represent set approach in categories – categorification e.g. Johnson, M, & Rosebrugh, R, Sketch Data Models, Relational Scheme and Data Specifications, Electronic Notes in Theoretical Computer Science 61 51-63 (2002).
1NF

- A relation is in 1NF if there is a functional dependency from the key to each non-key attribute.

- So expectation is:

  ```
  customer_id, item_code → customer_name
  ```

  ```
  customer_address
  driver_id
  driver_name
  ```

  If add something unrelated such as football_club then not in 1NF: need everything to be connected
All attributes must be related; adding stand-alone attributes means it's not even CCC
1NF is insufficient

- Everything is connected
- But may not be connected optimally
  - May be other arrows
    - From key component to non-key as a functional dependency
    - From non-key to non-key as a functional dependency
- Tests for these arrows are done in 2NF and 3NF respectively
- Potential presence of these unwanted arrows means that the diagram is not yet a LCCC
Introducing arrow to invalidate 2NF

$fd_1 : A \rightarrow D$; $I_1 : A \rightarrow A + B + C + D$;

adding $fd_1$ means that component of key determines non-key
Example of failing 2NF relation

Functional dependencies below are from component of key to non-key

```
customer_id       customer_name
               customer_address
```

Vast duplication of customer data each time something is delivered
Diagram does not commute. D+C obtained by following top path does not equal that obtained by following bottom path.
Solution

• Take $A \to D$ arrow out of pullback diagram
• Insert $A \to D$ dependency within category $A$, giving $A$ more internal structure
• $A$ (or $B$) can be an object or a pullback category with identity functor for reference purposes
• Alternative: possibly paste an additional pullback onto previous structure.
LCCC view of 2NF - Pullback

Category A contains dependency \( fd_1 : A \rightarrow D \)
Introducing arrow to invalidate 3NF

A \times_{C+F} B

\text{key}

\text{functional dependency}

\text{two non-key attributes}

\text{fd}_2 : C \rightarrow F;
adding \text{fd}_2 \text{ means that one non-key determines another non-key}
Example of failing 3NF relation

Functional dependencies below are from non-key to non-key

driver_id \rightarrow driver_name

Vast duplication of driver data each time something is delivered
Terminal object should be $A+B+C+F$ (typed as a disjoint sum); May not even be a category (depends on how constructed)
Solution

- Take \( C \rightarrow F \) arrow out of pullback diagram
- Develop new pullback to represent relationship between \( C \) and \( F \)
- Paste new pullback onto existing structure.
3NF and LCCC

- **3NF (non-stepping stone via 1NF and 2NF)**
  - A relation is in 3NF if each non-key attribute is dependent on the key, the whole key and nothing but the key

- **LCCC**
  - A relation is in 3NF if a valid pullback can be constructed from its functional dependencies
LCCC view of 3NF – Single Pullback Diagram

No other arrows permitted

functional dependency

key component

non-key

key component
LCCC view of 3NF – Pasted Pullback Diagram

Complex pullback diagrams can be pasted together as below
Format of squares as below must be respected
No other arrows allowed

\[(A \times_F C) \times_C B \rightarrow A \times_F C \rightarrow A \]
\[B \rightarrow C \rightarrow F\]
Higher Normal Forms

• In database theory go up to Boyce-Codd, 4NF and 5NF.
  • But 3NF is industry standard

• 5NF is Project-Join Normal Form
  • Define relations so that projection of attributes followed by joining together again returns starting point

• Already provided by LCCC in the adjointness between the X side and the + side.
LCCC for 5NF

Existential

Pullback functor (f*)

Universal (limit)

Adjointness $\exists \dashv \Delta$ and $\Delta \dashv \forall$ between functors mapping between $X$ and $+$ (project-join)
Interesting Points

- So assumption that LCCC is a satisfactory basis for information system representation is justified by its close correspondence to data normalisation at industry standard (and beyond)
- Data normalisation has a sounder basis in LCCC than in set theory
  - Conceptual bases conform naturally
    - Arrows naturally handled with categories
  - All normal forms up to 5NF are handled in a single diagram
  - LCCC provide a springboard for further data semantics
## Class Model Constraints as LCCC Types

<table>
<thead>
<tr>
<th>Arrow</th>
<th>Epic (surjective)</th>
<th>Membership class</th>
<th>Monic (injective)</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_l$</td>
<td>Y</td>
<td>A mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>A optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi_l^<em>$ (</em> is inverse)</td>
<td>Y</td>
<td></td>
<td>Each A onto 1 relation instances</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td></td>
<td>Each A onto N relation instances</td>
<td></td>
</tr>
<tr>
<td>$\pi_r$</td>
<td>Y</td>
<td>B mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>B optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi_r^*$</td>
<td>Y</td>
<td></td>
<td>Each B onto 1 relation instances</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td></td>
<td>Each B onto N relation instances</td>
<td></td>
</tr>
</tbody>
</table>
Software Engineering – Process

• Principles include
  • High cohesion
    • Everything is connected
      • Cartesian closed category
  • Low coupling
    • Entrance is always through official interface
      • Initial object in Cartesian closed category
    • Exit is always through official closure point
      • Terminal object in Cartesian closed category

• So less formal than with structures but some properties of CCC
Software Engineering – Transaction

• Transaction is standard way of defining a process
  • Principles of ACID
    • Atomicity, Consistency, Isolation, Durability
  • Logical technique for controlling the physical world e.g. banking transaction

• Requires three cycles of adjointness between initial and target state
  • First two for atomicity, consistency and isolation
  • Third for durability

• Process as a World Transaction, same authors as this paper, 36pp ANPA(2006).
Transaction ~ Monad/Comonad

- In category theory, transaction is effectively represented by a monad/comonad pairing

a) Associative law for monad $<T, \eta, \mu>$; b) Associative law for comonad $<S, \varepsilon, \delta>$
Monad/Comonad

• Functionality
  – Monad (looking back over 3 cycles)
    • $\mu : T^2 \rightarrow T$ (multiplication)
  – Comonad (looking forward over 3 cycles)
    • $\delta : S \rightarrow S^2$ (comultiplication)

• Objects of monad/comonad
  – Adjoint pair of functors between initial and target state
  – Initial and target state are LCCC (pullbacks)
Cartesian Monad

• If underlying categories are pullbacks
AND T preserves pullbacks
AND μ and η are Cartesian
Then the monad is a Cartesian monad

• That is, the underlying structures and the manipulation language are unified into a single categorial concept
  • The relational model (with sets) elevated to a categorial representation much closer to the physical world
Summary

• LCCC are indeed justified as the choice of category for representing information systems
  • Data structures as pullback
    • Data normalisation
      • to 3NF industry standard and beyond to 5NF
    • Typing of class model constraints
      • Membership class
      • Cardinality
  • Manipulation as Cartesian monad/comonad on pullback
    • Transaction
      • Unification with data structures
Advantages of LCCC over Sets

- 3NF is achieved directly through the pullback construction
  - Not through an optional design process of normalisation, unenforced in relational database systems
- Class model constraints are typed in the arrows of the pullback
  - Not labelled as in the Entity-Relationship model
- Manipulation by transactions is unified
  - Not with impedance mismatch of relational systems