Aspects of a Mathematical Theory of Data

John Cartmell

Jan 18, 2024

Preamble

www.researchgate.net/profile/John-Cartmell

- Preparation for a Mathematical Theory Of Data
- Concept Instance Algebras (*circa* 1973)
- Instances of Generalised Algebraic Theories

Terminology

- *modelling* for me is *theorizing*
- I speak of *instances of theories* rather then *models of theories*
- I speak of *data specifications* except when I forget and I call them *data models*
- the act of constructing data specifications is *data modelling*
- a *model of data* is a meta-theory (a meta-model) describing what constitutes a data specification. Most significantly there are
  - *relational* and
  - *nested relational* models of data
- the *mathematical theory of data* is a meta-theory of data that supports technology independent reasoning about data specifications in all their forms.

Why?

- There are gross inefficiencies in the methodologies and working practices used in a key activity in s/w systems development and maintenance namely in the creation and maintenance of specifications of the data stored in databases and represented in messages variously intra-communicated between components of systems and inter-communicated between systems.

Why?

- There are gross inefficiencies in the methodologies and working practices used in a key activity in s/w systems development and maintenance namely in the creation and maintenance of specifications of the data stored in databases and represented in messages variously intra-communicated between components of systems and inter-communicated between systems.
- These inefficiences have been established and endorsed by a theory which is grossly inadequate.

Why?

- There are gross inefficiencies in the methodologies and working practices used in a key activity in s/w systems development and maintenance namely in the creation and maintenance of specifications of the data stored in databases and represented in messages variously intra-communicated between components of systems and inter-communicated between systems.
- These inefficiences have been established and endorsed by a theory which is grossly inadequate.
- A new theory is required to expose and remedy the shortcomings.

Why?

- There are gross inefficiencies in the methodologies and working practices used in a key activity in s/w systems development and maintenance namely in the creation and maintenance of specifications of the data stored in databases and represented in messages variously intra-communicated between components of systems and inter-communicated between systems.
- These inefficiences have been established and endorsed by a theory which is grossly inadequate.
- A new theory is required to expose and remedy the shortcomings.
- The challenge is to positively impact best practice.

Mathematical Theory of Data

- is a meta-theory,
- it covers *principles* and *criteria* for goodness of data specifications,
- it reveals the significance of commutative diagrams and therefore category theory.
- The slogan on the tin is *Good Data Modelling is Good Theorising*.

Prior Theory — 1970 - 1979

- In 1970, E.F.Codd introduced the relational model of data and the idea of normal form.
- A year later he defines the term 'functional dependency' and uses it to define 'third normal form' (3NF).
- In 1977, Fagin defines the concept of a 'multivalued dependency' and uses it to define 'fourth normal form' (4NF).
- Two years on, Fagin defines 'projection-join normal form' which is also known as 'fifth normal form' (5NF).

The success of Codd's Relational Model of Data

- Codd's model of data has been very influential. Witness that by 2020 Oracle Corporation had grown from being founded in 1977 to having a 42% share of an estimated $30billion market for relational database technology.
- Codd in 1990 says that

  > *The relational model is solidly based on two parts of mathematics: first-order predicate logic and the theory of relations.*

- My opinion is that this has been to found data modelling on the wrong mathematics.
- Codd's mathematical basis and therefore his model do nothing to guide the programmer as navigator, to use Charles W Bachman's phrase,
- nor do they encourage thinking about navigation path equivalence, i.e. diagrams that commute ... even though thinking about diagrams that commute is essential to the goodness of data specifications.
- The right mathematical starting point for the theory of data is category theory.

Goodness Criteria

- From a mathematical perspective are not really normal forms!
- They are goodness criteria (GC) that articulate good engineering principles.
- I wish to show that we can
  - genericise relational database normal form criteria into abstract logical terms,
  - define goodness criteria that are generic i.e. can be applied to any data specifications not just to relational schema,
  - prove that the classic relational database normal form criteria (2NF, 3NF, BCNF, INC-NF, 4NF, 5NF) are consequences of these generic goodness criteria,
  - articulate principles from which the generic goodness criteria follow.

Goodness Criteria

- From a mathematical perspective are not really normal forms!
- They are goodness criteria (GC) that articulate good engineering principles.
- I wish to show that we can
  - genericise relational database normal form criteria into abstract logical terms,
  - define goodness criteria that are generic i.e. can be applied to any data specifications not just to relational schema,
  - prove that the classic relational database normal form criteria (2NF, 3NF, BCNF, INC-NF, 4NF, 5NF) are consequences of these generic goodness criteria,
  - articulate principles from which the generic goodness criteria follow.

Fundamental Principles $\Longrightarrow$ *Generic Goodness Criteria* $\Longrightarrow$ *Classic Normal Forms*

An abstract view

- A data specification is a presentation of a theory (of what is).
- There can be many different presentations of a single theory and these have different roles depending on their properties
  - some presentations are said to be *physical* — the choice of primitives in such a presentation is a choice of the individual elements to be represented in the data,
  - other presentations are said to be *logical* — these seek to describe the data by directly describing its internal relationships.
- Both the theory and its logical presentations express the overall information content of the data independently of the details of its representation.

Fundamental Principles at this Abstract Level

- Principle 1 – absence of redundancy in presentation.
- Principle 2 – the theory be the tightest possible fit to the facts.

- The two principles collectively
  - ensure absence of redundancy in data and in data management logic.

- Note: principle 2 expresses a kind of logical completeness.

Data Specifications

Two kinds of types in play

- the *definienda* – types all of whose instances are *particulars*
  - employee, department, student, account, product, order, shipment, delivery, flight, booking and so on,
  - molecular structure, atom, covalent bond, element, isotope, reaction, metabolite, mass trace, chromatogram, peak.

Data Specifications

Two kinds of types in play

- the *definienda* – types all of whose instances are *particulars*
    - employee, department, student, account, product, order, shipment, delivery, flight, booking and so on,
    - molecular structure, atom, covalent bond, element, isotope, reaction, metabolite, mass trace, chromatogram, peak.
- the *definiens* – types all of whose instances are *universals*
    - string, integer, float, boolean and so on.

Data Specifications

Two kinds of types in play

- the *definienda* – types all of whose instances are *particulars*
  - employee, department, student, account, product, order, shipment, delivery, flight, booking and so on,
  - molecular structure, atom, covalent bond, element, isotope, reaction, metabolite, mass trace, chromatogram, peak.
- the *definiens* – types all of whose instances are *universals*
  - string, integer, float, boolean and so on.
- I assume a fixed set V of universals and define data specifications and instances relative to V.

Data Specifications as Sketches

A data specification is a sketch of

- an RR.5 range category,
- with designated finite restriction products,
- designated monomorphisms with partial inverses,
- an object v representing the set V of universals.

Next I go through the background catagory theory that is involved in this definition.

Mono Sources

In a category $\mathbf{C}$, a *source* is a family of morphisms with common domain:

$$\begin{tikzcd} & b_1 \\ a \arrow[ur, "f_1"] \arrow[r, "f_2"] \arrow[ddr, "f_n"'] & b_2 \\ & \vdots \\ & b_n \end{tikzcd}$$

Such a source is said to be a *mono source* iff for all

$g, h : x \to a$ in $\mathbf{C}$ so that

$$\begin{tikzcd} & & b_1 \\ x \arrow[r, shift left, "g"] \arrow[r, shift right, "h"'] & a \arrow[ur, "f_1"] \arrow[r, "f_2"] \arrow[ddr, "f_n"'] & b_2 \\ & & \vdots \\ & & b_n \end{tikzcd}$$

in $\mathbf{C}$ then if

$g \circ f_i = h \circ f_i$, for each i, then $g = h$. OR, in presence of cartesian products,

$< f_1, ...f_n >$ is a monomorphim.

Category of Sets and Partial Functions

- There is a category **Par** of sets and partial functions.
- For a partial function $f : A \to B$ define its restriction idempotent to be the function $\bar{f} : A \to A$ is defined by

$$\bar{f}(a) = \begin{cases} a & \text{if } f \text{ defined at } a, \\ \mathit{undefined} & \text{otherwise.} \end{cases}$$

- This *bar* operator satisfies four algebraic identities R.1, R.2, R3, and R.4.
- Also for a partial $f : A \to B$ define its range idempotent to be the function $\hat{f} : B \to B$ is defined by

$$\hat{f}(b) = \begin{cases} b & \text{if there exists } a \in A \text{ such that } f(a) = b, \\ \mathit{undefined} & \text{otherwise.} \end{cases}$$

- This *hat* operator satisfies identities RR.1, ...RR.5.

Restriction Categories I(2002, Cockett and Lack)

A *restriction category* is a category along with an operator that maps every morphism f to an idempotent $\bar{f}$ on its domain satisfying

R.1 For $f : a \to b$ in $\mathbf{C}$

$$\bar{f} \circ f = f$$

.

R.2. If
$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$
in $\mathbf{C}$ then

$$\bar{g} \circ \bar{f} = \bar{f} \circ \bar{g}.$$

R.3. If
$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$
in $\mathbf{C}$ then

$$\overline{\bar{f} \circ g} = \bar{f} \circ \bar{g}$$

.

R.4. If $a \xrightarrow{f} b \xrightarrow{g} c$ in $\mathbf{C}$ then

$$f \circ \bar{g} = \overline{f \circ g} \circ f$$

.

Range Categories (2012, Cockett, Guo and Hofstra)

Range Categories (2012, Cockett, Guo and Hofstra)

- A *range category* is a restriction category with an additional operator as follows if $f : a \to b$ in $\mathbf{C}$ then $\hat{f} : b \to b$ satisfying

RR.1 For $f : a \to b$ *in* $\mathbf{C}$, $\overline{\hat{f}} = \hat{f}$.

RR.2 For $f : a \to b$ *in* $\mathbf{C}$, $f \circ \hat{f} = f$.

RR.3. If $a \xrightarrow{f} b \xrightarrow{g} c$ *in* $\mathbf{C}$ *then* $\widehat{f \circ \overline{g}} = \hat{f} \circ \overline{g}$.

RR.4. If $a \xrightarrow{f} b \xrightarrow{g} c$ *in* $\mathbf{C}$ *then* $(\widehat{hat(f) \circ g}) = \widehat{f \circ g}$.

Range Categories (2012, Cockett, Guo and Hofstra)

- A *range category* is a restriction category with an additional operator as follows if $f : a \to b$ in $\mathbf{C}$ then $\hat{f} : b \to b$ satisfying

  RR.1 For $f : a \to b$ *in* $\mathbf{C}$, $\overline{\hat{f}} = \hat{f}$.

  RR.2 For $f : a \to b$ *in* $\mathbf{C}$, $f \circ \hat{f} = f$.

  RR.3. If $a \xrightarrow{f} b \xrightarrow{g} c$ *in* $\mathbf{C}$ *then* $\widehat{f \circ \overline{g}} = \hat{f} \circ \overline{g}$.

  RR.4. If $a \xrightarrow{f} b \xrightarrow{g} c$ *in* $\mathbf{C}$ *then* $(\widehat{hat(f) \circ g}) = \widehat{f \circ g}$.

- A range category may additionally satisfy RR.5 if

$$\begin{tikzcd} a \arrow[r, "f"] & b \arrow[r, bend left, "g"] \arrow[r, bend right, "h"'] & c \end{tikzcd} \qquad \text{then } f \circ g = f \circ h \Rightarrow \hat{f} \circ g = \hat{f} \circ h.$$

Restriction Products

- The ususal cartesian product of sets in the category of sets and partial functions **Par** does not satisfy the ususal categorical cartesian product conditions.
- In 2006 "Restriction Categories III" Cockett and Lack define the appropriate notion of product.
- They define *restriction product* of a pair of objects in a restriction category.

Partial Ordering of $Hom(A, B)$

- In a restriction category we can define a partial ordering on each hom set Hom(a,b) by defining :

$$f \leq g \text{ iff } f = \bar{g} \circ f$$

,

- we can think of $f \leq g$ as meaning that if f is defined then g is defined and the two are equal,

Partial Ordering of $Hom(A, B)$

- In a restriction category we can define a partial ordering on each hom set Hom(a,b) by defining :

$$f \leq g \text{ iff } f = \bar{g} \circ f$$

,

- we can think of $f \leq g$ as meaning that if f is defined then g is defined and the two are equal,
- there are lots of data specifications having near commutative diagrams i.e. instances of relationships f, g and h satisfying

$$f \circ g \leq h$$

.

Partial Inverse of a Monomorphism

If $m : a \to b$ is a monomorphim in range category $\mathbf{C}$ then a map $m^{-1} : b \to a$

$$\begin{tikzcd} a \arrow[r, "m"] & b \arrow[l, bend left, "m^{-1}"] \end{tikzcd}$$

I will say m^{-1} is the *(partial) inverse* of m iff

$$m \circ m^{-1} = id_a \qquad \text{and} \qquad m^{-1} \circ m = \widehat{m}.$$

Definition – γ-structured category

I shall use the shorthand *γ-structured category* to mean a triple $\langle \mathbf{C}, M, v \rangle$ where

- $\mathbf{C}$ is a RR.5 range category with specified finite restriction products,
- M is a set of designated monomorphisms of $\mathbf{C}$ closed under composition and such that each $m \in M$ has a partial inverse m^{-1},
- a distinguished object v, such that every morphism $f : v \to x$ in $\mathbf{C}$ factors through m^{-1}, for some monomorphism m.

Note that it follows from this definition that a sketch for a *γ-structured category* has no need for edges with domain v.

Data Specification and Data Specification Instance

In this presentation,

- by *data specification* I shall mean a sketch for a γ-structured category such that the designated object v has no outgoing edges – neither edges $\underline{v \to v}$ nor edges $\underline{v \to \textit{non-}v}$.
- If S is a sketch for γ-structured category denote by $\mathbf{C}(S)$ the γ-structured category generated from S.

Data Specification and Data Specification Instance

In this presentation,

- by *data specification* I shall mean a sketch for a γ-structured category such that the designated object v has no outgoing edges – neither edges $\underline{v \to v}$ nor edges $\underline{v \to non\text{-}v}$.
- If S is a sketch for γ-structured category denote by $\mathbf{C}(S)$ the γ-structured category generated from S.
- Define an *instance* of a data specification S to be a range functor $F : \mathbf{C}(S) \to \mathbf{Par}$ that preserves the specified restriction products and maps the object v to the set V.

  Note that such an F will preserve designated monomorphisms and their inverses.

I will muddle up data specifications and sketches in these slides.
I will speak of $\mathbf{C}(S)$ as the theory category.

Next I want to give some examples to show how all this works in practice. The very next next example is of the ... relational model of data.

Relational Data

student

<u>sName</u>	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor

<u>pDept</u>*	<u>pId</u>	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department

<u>dName</u>	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

Relational Data

student

<u>sName</u>	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor

<u>pDept</u>*	<u>pId</u>	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department

<u>dName</u>	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

- students and departments are identified by name,
- professors are identified by combination of department and id,

Relational Data

student

<u>sName</u>	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor

<u>pDept</u>*	<u>pId</u>	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department

<u>dName</u>	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table,

Relational Data

student		
sName	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor		
pDept*	pId	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department	
dName	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table, $\boxed{\mathit{student}[\mathit{sDept}] \subseteq \mathit{department}[\mathit{dName}]}$,

Relational Data

student		
sName	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
…	…	…

professor		
pDept*	pId	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
…	…	…

department	
dName	dHd*
maths	#3
phil	#1
history	#5
physics	#1
…	…

- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table, $\boxed{student[sDept] \subseteq department[dName]}$,
- similarly, $\boxed{professor[pDept] \subseteq department[dName]}$,

Relational Data

student		
<u>sName</u>	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor		
<u>pDept</u>*	<u>pId</u>	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department	
<u>dName</u>	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table, $\boxed{student[sDept] \subseteq department[dName]}$,
- similarly, $\boxed{professor[pDept] \subseteq department[dName]}$,
- rows of the *student* table reference the *professor* table by virtue of two columns instancing values from the identifying columns of that table,

Relational Data

student		
sName	sDept*	sSv*
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor		
pDept*	pId	pName
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department	
dName	dHd*
maths	#3
phil	#1
history	#5
physics	#1
...	...

- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table, $\boxed{student[sDept] \subseteq department[dName]}$,
- similarly, $\boxed{professor[pDept] \subseteq department[dName]}$,
- rows of the *student* table reference the *professor* table by virtue of two columns instancing values from the identifying columns of that table, $\boxed{student[sDept, sSv] \subseteq professor[pDept, pId]}$,

Relational Data

student		
sName	**sDept***	**sSv***
gray	phil	#1
bohm	maths	#1
smith	maths	#2
doe	phil	#1
...	...	...

professor		
pDept*	**pId**	**pName**
maths	#1	scott
maths	#2	smith
maths	#3	gandy
phil	#1	smith
phil	#2	ayer
...	...	...

department	
dName	**dHd***
maths	#3
phil	#1
history	#5
physics	#1
...	...

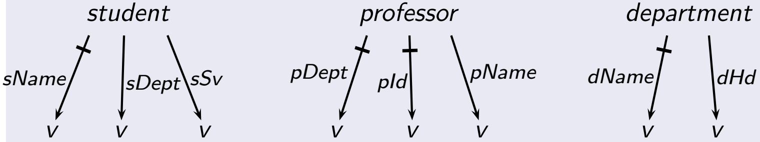
- rows of the *student* table reference the *department* table by virtue of a column that instances values from the identifying column that table, $\boxed{student[sDept] \subseteq department[dName]}$,
- similarly, $\boxed{professor[pDept] \subseteq department[dName]}$,
- rows of the *student* table reference the *professor* table by virtue of two columns instancing values from the identifying columns of that table, $\boxed{student[sDept, sSv] \subseteq professor[pDept, pId]}$,
- similarly, $\boxed{department[dName, dHd] \subseteq professor[pDept, pId]}$.

Referential Inclusion Dependencies as Range Identities

- Now think of each column as a function that maps rows of a table to values.
- Each inclusion dependency can be expressed as identity on the ranges of these functions.
- Each
$$a[f] \subseteq b[q]$$
can be represented as
$$\widehat{f} \leq \widehat{q} \text{ in } \mathbf{Par} \ ,$$
- Similarly
$$a[f_1, \ldots f_n] \subseteq b[q_1, \ldots q_n]$$
can be represented as
$$\langle \widehat{f_1, \ldots f_n} \rangle \leq \langle \widehat{q_1, \ldots q_n} \rangle \text{ in } \mathbf{Par} \ .$$

Sketch for γ-structured category(Relational)

Directed Graph



Mono-sources

Indicated by bars on the graph.

Identities

$$\widehat{sDept} \leq \widehat{dName}$$
$$\widehat{pDept} \leq \widehat{dName}$$
$$\widehat{\langle sDept, sSv \rangle} \leq \widehat{\langle pDept, pId \rangle}$$
$$\widehat{\langle dName, dHd \rangle} \leq \widehat{\langle pDept, pId \rangle}$$

Sketch for γ-structured category(Relational)

Directed Graph

$$
\begin{tikzcd}
& student \arrow[dl, tail, "sName"'] \arrow[d, "sDept"] \arrow[dr, "sSv"] & & & professor \arrow[dl, tail, "pDept"'] \arrow[d, tail, "pId"] \arrow[dr, "pName"] & & department \arrow[d, tail, "dName"'] \arrow[dr, "dHd"] & \\
V & V & V & V & V & V & V & V
\end{tikzcd}
$$

Mono-sources

Indicated by bars on the graph.

Identities

$$\widehat{sDept} \leq \widehat{dName} \quad \rightsquigarrow \quad sDept \circ \widehat{dName} = sDept$$

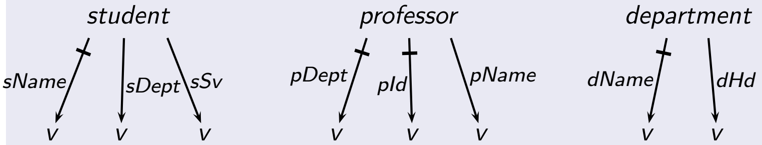
$$\widehat{pDept} \leq \widehat{dName}$$

$$\langle \widehat{sDept, sSv} \rangle \leq \langle \widehat{pDept, pId} \rangle$$

$$\langle \widehat{dName, dHd} \rangle \leq \langle \widehat{pDept, pId} \rangle$$

Sketch for γ-structured category(Relational)

Directed Graph



Mono-sources

Indicated by bars on the graph.

Identities

$$\widehat{sDept} \leq \widehat{dName} \qquad \leftrightsquigarrow \qquad sDept \circ \widehat{dName} = sDept$$

$$\widehat{pDept} \leq \widehat{dName} \qquad \leftrightsquigarrow \qquad pDept \circ \widehat{dName} = pDept$$

$$\langle \widehat{sDept, sSv} \rangle \leq \langle \widehat{pDept, pId} \rangle$$

$$\langle \widehat{dName, dHd} \rangle \leq \langle \widehat{pDept, pId} \rangle$$

Sketch for γ-structured category(Relational)

Directed Graph

$$
\begin{tikzcd}[column sep=small]
 & student \arrow[dl, "sName"'] \arrow[d, "sDept"] \arrow[dr, "sSv"] & & & professor \arrow[dl, "pDept"'] \arrow[d, "pId"'] \arrow[dr, "pName"] & & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & V & V & V & V & V & & V & V
\end{tikzcd}
$$

Mono-sources

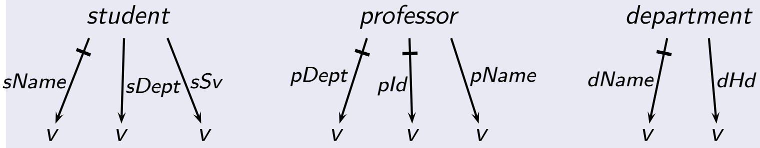
Indicated by bars on the graph.

Identities

$$
\begin{array}{rcl}
\widehat{sDept} \leq \widehat{dName} & \leftrightsquigarrow & sDept \circ \widehat{dName} = sDept \\
\widehat{pDept} \leq \widehat{dName} & \leftrightsquigarrow & pDept \circ \widehat{dName} = pDept \\
\langle \widehat{sDept, sSv} \rangle \leq \langle \widehat{pDept, pId} \rangle & \leftrightsquigarrow & \langle sDept, sSv \rangle \circ \langle \widehat{pDept, pId} \rangle = \langle sDept, sSv \rangle \\
\langle \widehat{dName, dHd} \rangle \leq \langle \widehat{pDept, pId} \rangle & &
\end{array}
$$

Sketch for γ-structured category(Relational)

Directed Graph



Mono-sources

Indicated by bars on the graph.

Identities

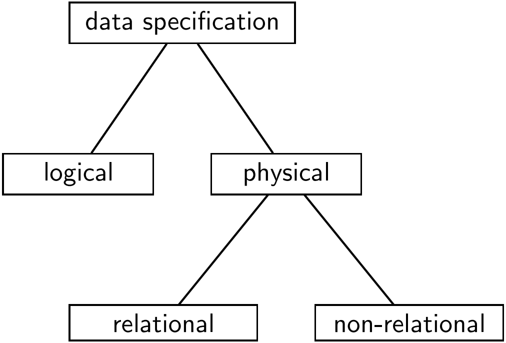
$$\widehat{sDept} \leq \widehat{dName} \quad \leftrightsquigarrow \quad sDept \circ \widehat{dName} = sDept$$

$$\widehat{pDept} \leq \widehat{dName} \quad \leftrightsquigarrow \quad pDept \circ \widehat{dName} = pDept$$

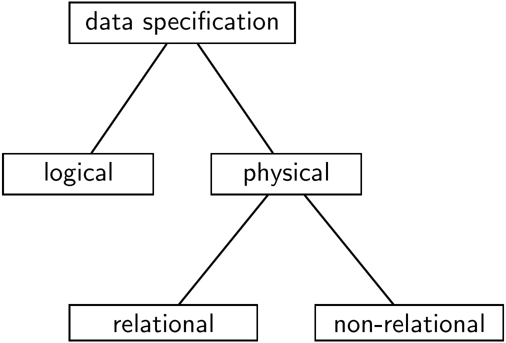
$$\widehat{\langle sDept, sSv \rangle} \leq \widehat{\langle pDept, pId \rangle} \quad \leftrightsquigarrow \quad \langle sDept, sSv \rangle \circ \widehat{\langle pDept, pId \rangle} = \langle sDept, sSv \rangle$$

$$\widehat{\langle dName, dHd \rangle} \leq \widehat{\langle pDept, pId \rangle} \quad \leftrightsquigarrow \quad \langle dName, dHd \rangle \circ \widehat{\langle pDept, pId \rangle} = \langle dName, d$$

Classifying Data Specifications

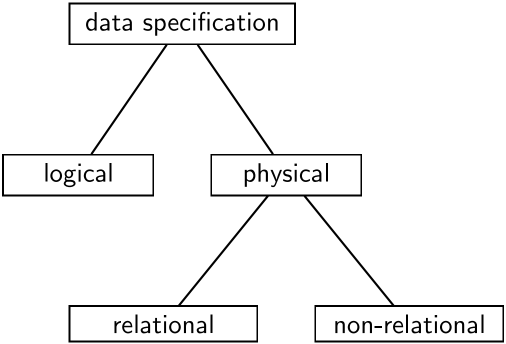


Classifying Data Specifications



- in *relational* sketches all edges are of the $\underline{\textit{non-}v \to v}$ type and each such represents a column of a table/relation,

Classifying Data Specifications



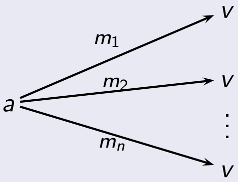
- in *relational* sketches all edges are of the <u>*non-v* $\rightarrow$ *v*</u> type and each such represents a column of a table/relation,
- other *physical sketches* (*non-relational*) in addition to the <u>*non-v* $\rightarrow$ *v*</u> type edges have edges of the <u>*non-v* $\rightarrow$ *non-v*</u> type and these represent structural containment,
- non-relational physical data specifications are also said to be *hierarchical*.

Characterisation of Relational Data Specifications

Definition

A data specification is *relational* iff

- all edges are of the <u>*non-v* $\rightarrow$ *v*</u> type,
- every non-v-node is the domain of at least one v-valued mono-source i.e. for every non-v-node a, for some $n \geq 1$, there exists a source



which is designated as a mono-source i.e. for which $\langle m_1, ... m_n \rangle$ is a designated monomorphism.

Construction – Transform a Relational Sketch to a Logical Sketch

Lemma

For any classic relational data specification there is an equivalent data specification (i.e. one with the same theory category) which is logical.

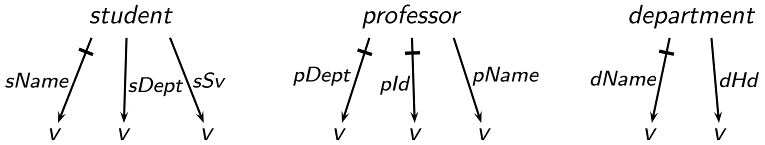
Proof.

In outline: We construct a series of equivalent sketches by eliminating each inclusion dependency in turn. When all eliminated the resulting sketch is the required logical sketch. Eliminate the inclusion dependency $a[f_1, \ldots f_n] \subseteq b[m_1, \ldots m_n]$ as follows:

- remove the inclusion dependency,
- replace by an edge $f : a \to b$,
- remove those f_i that are edges and rewrite any occurrence of such f_i in the remaining inclusion dependencies by $f \circ m_i$,
- for those f_i that are not edges add a path equivalence (i.e. a commuting diagram) $f \circ m_i = f_i$.

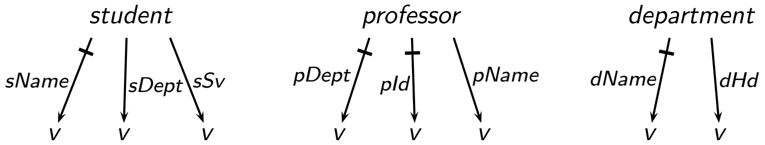
□

Example — Transform Relational Sketch to Logical Sketch



- $student[sDept] \subseteq department[dName]$
- $professor[pDept] \subseteq department[dName]$
- $student[sDept, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

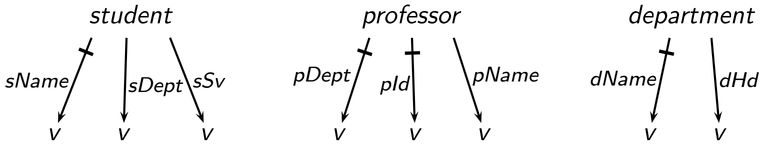
Example — Transform Relational Sketch to Logical Sketch



- $student[sDept] \subseteq department[dName]$
- $professor[pDept] \subseteq department[dName]$
- $student[sDept, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 1. Eliminate $student[sDept] \subseteq department[dName]$

Example — Transform Relational Sketch to Logical Sketch

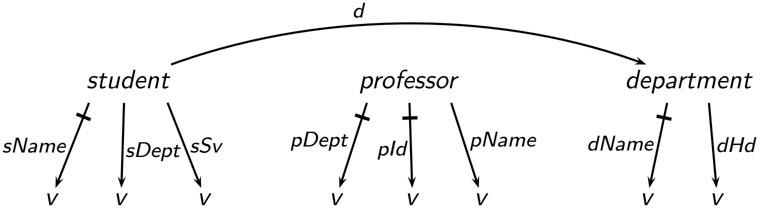


- $student[sDept] \subseteq department[dName]$
- $professor[pDept] \subseteq department[dName]$
- $student[sDept, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 1. Eliminate $student[sDept] \subseteq department[dName]$

Remove *sDept* and replace by an edge $d : student \to department$. Rewrite appearances of *sDept* in the sketch by $d \circ dName$.

Example — Transform Relational Sketch to Logical Sketch

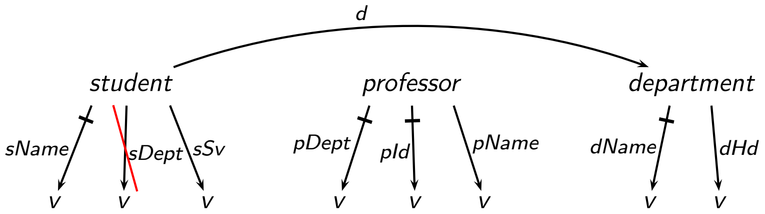


- $student[sDept] \subseteq department[dName]$
- $professor[pDept] \subseteq department[dName]$
- $student[sDept, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 1. Eliminate $student[sDept] \subseteq department[dName]$

Remove *sDept* and replace by an edge $d : student \to department$. Rewrite appearances of *sDept* in the sketch by $d \circ dName$.

Example — Transform Relational Sketch to Logical Sketch

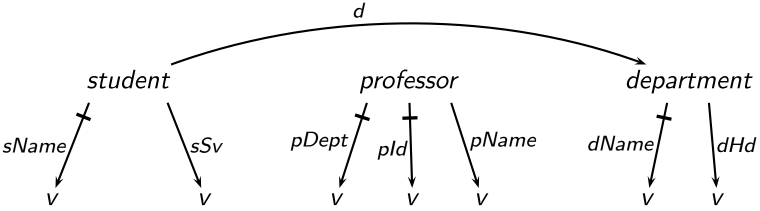


- ~~$student[sDept] \subseteq department[dName]$~~
- $professor[pDept] \subseteq department[dName]$
- $student[sDept, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 1. Eliminate $student[sDept] \subseteq department[dName]$

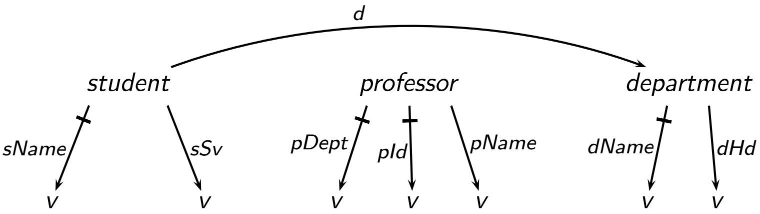
Remove *sDept* and replace by an edge $d : student \to department$. Rewrite appearances of *sDept* in the sketch by $d \circ dName$.

Example — Transform Relational Sketch to Logical Sketch



- $professor[pDept] \subseteq department[dName]$
- $student[d \circ dName, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

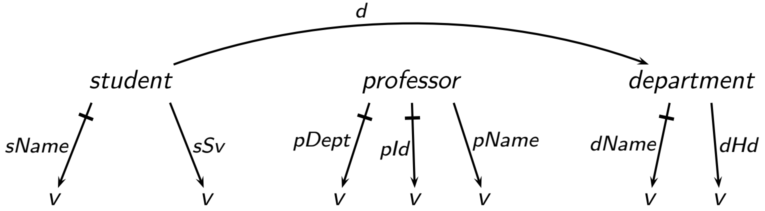
Example — Transform Relational Sketch to Logical Sketch



- $professor[pDept] \subseteq department[dName]$
- $student[d \circ dName, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 2. Eliminate $professor[pDept] \subseteq department[dName]$

Example — Transform Relational Sketch to Logical Sketch

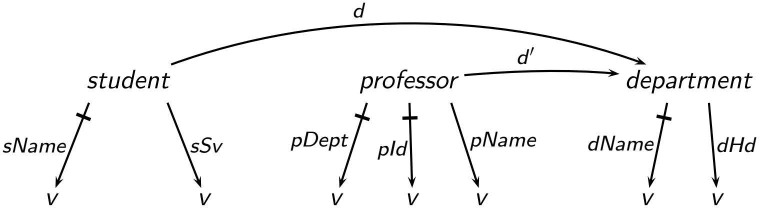


- $professor[pDept] \subseteq department[dName]$
- $student[d \circ dName, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 2. Eliminate $professor[pDept] \subseteq department[dName]$

Remove *pDept* and replace ny an edge $d' : professor \to department$.
Rewrite appearances of *pDept* in the sketch by $d' \circ dName$.

Example — Transform Relational Sketch to Logical Sketch

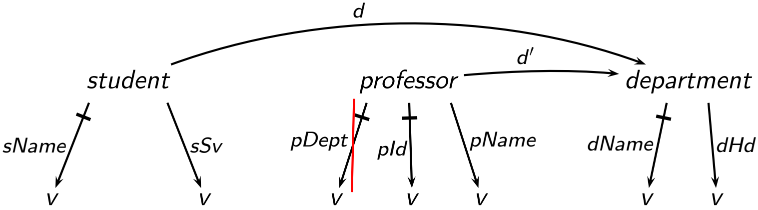


- $professor[pDept] \subseteq department[dName]$
- $student[d \circ dName, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 2. Eliminate $professor[pDept] \subseteq department[dName]$

Remove $pDept$ and replace ny an edge $d' : professor \to department$.
Rewrite appearances of $pDept$ in the sketch by $d' \circ dName$.

Example — Transform Relational Sketch to Logical Sketch



- ~~$professor[pDept] \subseteq department[dName]$~~
- $student[d \circ dName, sSv] \subseteq professor[pDept, pId]$
- $department[dName, dHd] \subseteq professor[pDept, pId]$

Step 2. Eliminate $professor[pDept] \subseteq department[dName]$

Remove $pDept$ and replace ny an edge $d' : professor \to department$.
Rewrite appearances of $pDept$ in the sketch by $d' \circ dName$.

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
 & student \arrow[dl, mapsto, "sName"'] \arrow[dr, "sSv"] \arrow[rrrrr, bend left, "d"] & & professor \arrow[d, mapsto, "pId"'] \arrow[dr, "pName"] \arrow[rrr, mapsto, "d'"] & & & department \arrow[dl, mapsto, "dName"'] \arrow[dr, "dHd"] & \\
V & & V & V & V & V & & V
\end{tikzcd}
$$

- $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Example — Transform Relational Sketch to Logical Sketch

$$\begin{tikzcd}
& student \arrow[dl, "sName"'] \arrow[dr, "sSv"] \arrow[rrrrrr, bend left, "d"] & & & professor \arrow[dl, "pId"'] \arrow[dr, "pName"] \arrow[rrr, "d'"] & & & department \arrow[dl, "dName"'] \arrow[dr, "dHd"] & \\
V & & V & V & & V & V & & V
\end{tikzcd}$$

- $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 3. Eliminate
$student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
 & student \arrow[dl, "sName"'] \arrow[dr, "sSv"] \arrow[rrrr, bend left, "d"] & & professor \arrow[d, "pId"'] \arrow[dr, "pName"] \arrow[rr, "d'"] & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & & V & V & V & V & V
\end{tikzcd}
$$

- $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 3. Eliminate $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$

Remove sSv and replace by an edge $s : student \to professor$. Rewrite appearances of sSv in the sketch by $s \circ pId$. Add commutative diagram

$$
\begin{tikzcd}
\text{student} \arrow[rr, "s"] \arrow[dr, "d \circ dName"'] & & \text{professor} \arrow[dl, "d' \circ dName"] \\
 & \text{v} &
\end{tikzcd}
\text{ simplify to }
\begin{tikzcd}
\text{student} \arrow[rr, "s"] \arrow[dr, "d"'] & & \text{professor} \arrow[dl, "d'"] \\
 & \text{department} &
\end{tikzcd}
.
$$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
student \arrow[d, "sName"'] \arrow[dr, "sSv"] \arrow[rr, "s"] \arrow[rrrr, bend left, "d"] & & professor \arrow[d, "pId"'] \arrow[dr, "pName"] \arrow[rr, "d'"] & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & V & V & V & V & V
\end{tikzcd}
$$

- $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 3. Eliminate $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$

Remove sSv and replace by an edge $s : student \to professor$. Rewrite appearances of sSv in the sketch by $s \circ pId$. Add commutative diagram

$$
\begin{tikzcd}
\text{student} \arrow[r, "s"] \arrow[dr, "d \circ dName"'] & \text{professor} \arrow[d, "d' \circ dName"] \\
& \text{V}
\end{tikzcd}
\quad \text{simplify to} \quad
\begin{tikzcd}
\text{student} \arrow[r, "s"] \arrow[dr, "d"'] & \text{professor} \arrow[d, "d'"] \\
& \text{department}
\end{tikzcd}
.
$$

Example — Transform Relational Sketch to Logical Sketch

$$\begin{tikzcd}
 & student \arrow[dl, "sName"'] \arrow[d, "sSv"] \arrow[rr, "s"] \arrow[rrrr, bend left, "d"] & & professor \arrow[d, "pId"'] \arrow[dr, "pName"] \arrow[rr, "d'"] & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
v & v & & v & v & v & v
\end{tikzcd}$$

- ~~$student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$~~
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 3. Eliminate $student[d \circ dName, sSv] \subseteq professor[d \circ dName, pId]$

Remove sSv and replace by an edge $s : student \to professor$. Rewrite appearances of sSv in the sketch by $s \circ pId$. Add commutative diagram

$$\begin{tikzcd}
\text{student} \arrow[r, "s"] \arrow[dr, "d \circ dName"'] & \text{professor} \arrow[d, "d' \circ dName"] \\
 & \text{v}
\end{tikzcd}
\text{ simplify to }
\begin{tikzcd}
\text{student} \arrow[r, "s"] \arrow[dr, "d"'] & \text{professor} \arrow[d, "d'"] \\
 & \text{department}
\end{tikzcd}.$$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
& student \arrow[rr, "s"] \arrow[rrrr, bend left, "d"] \arrow[dl, "sName"'] & & professor \arrow[rr, "d'"] \arrow[d, "pId"'] \arrow[dr, "pName"] & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & & & V & V & V & V
\end{tikzcd}
$$

- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
& student \arrow[rrr, "s"] \arrow[dl, "sName"'] \arrow[rrrrr, bend left, "d"] & & & professor \arrow[rrr, "d'"] \arrow[d, "pId"'] \arrow[dr, "pName"] & & & department \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & & & & V & V & & V & V
\end{tikzcd}
$$

- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 4. Eliminate this final inclusion dependency.

Example — Transform Relational Sketch to Logical Sketch

$$\begin{tikzcd}
 & student \arrow[rr, "s"] \arrow[rrrrr, bend left, "d"] \arrow[dl, "sName"'] & & professor \arrow[rrr, shift left, "d'"] \arrow[d, "pId"'] \arrow[dr, "pName"] & & & department \arrow[lll, shift left, "h"] \arrow[d, "dName"'] \arrow[dr, "dHd"] & \\
V & & & V & V & & V & V
\end{tikzcd}$$

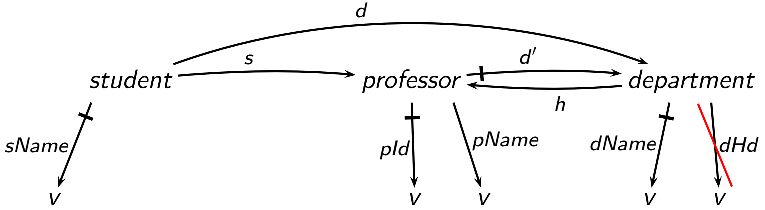
- $department[dName, dHd] \subseteq professor[d' \circ dName, pId]$

Step 4. Eliminate this final inclusion dependency.

Replace dHd by an edge $h : department \to professor$, add commutivity of

$$\begin{tikzcd}
\text{department} \arrow[rr, "h"] \arrow[dr, "dName"'] & & \text{professor} \arrow[dl, "d' \circ dName"] \\
 & \text{V} &
\end{tikzcd}
\quad \text{simplfy to} \quad
\begin{tikzcd}
\text{department} \arrow[rr, "h"] \arrow[dr, "id"'] & & \text{professor} \arrow[dl, "d'"] \\
 & \text{department} &
\end{tikzcd} .$$

Example — Transform Relational Sketch to Logical Sketch



- $\cancel{department[dName, dHd] \subseteq professor[d' \circ dName, pId]}$

Step 4. Eliminate this final inclusion dependency.

Replace *dHd* by an edge $h : department \to professor$, add commutivity of

$$\begin{tikzcd} \text{department} \arrow[r, "h"] \arrow[dr, "dName"'] & \text{professor} \arrow[d, "d' \circ dName"] \\ & v \end{tikzcd} \quad \text{simplfy to} \quad \begin{tikzcd} \text{department} \arrow[r, "h"] \arrow[dr, "id"'] & \text{professor} \arrow[d, "d'"] \\ & \text{department} \end{tikzcd} .$$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}
student \arrow[rr, "s"] \arrow[rrrr, bend left, "d"] \arrow[d, "sName"'] & & professor \arrow[rr, "d'"] \arrow[d, "pId"'] \arrow[dr, "pName"] & & department \arrow[d, "dName"'] \\
V & & V & V & V
\end{tikzcd}
$$

Example — Transform Relational Sketch to Logical Sketch

$$
\begin{tikzcd}[column sep=large]
 & \mathit{student} \arrow[rr, "s"] \arrow[rrrr, bend left, "d"] \arrow[dl, "\mathit{sName}"'] & & \mathit{professor} \arrow[rr, "d'"] \arrow[d, "\mathit{pId}"'] \arrow[dr, "\mathit{pName}"] & & \mathit{department} \arrow[d, "\mathit{dName}"'] \\
V & & & V & V & V
\end{tikzcd}
$$

subject to commutivity of

$$
\begin{tikzcd}
 & \text{department} & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] & & \text{professor} \arrow[ul, "d'"']
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
 & \text{department} \arrow[dr, "h"] & \\
\text{professor} \arrow[ur, "d'"] \arrow[rr, "id"'] & & \text{professor}
\end{tikzcd}
$$

Resulting Logical Data Specification

$$
\begin{tikzcd}
 & \text{department} \arrow[dr, bend left, "h"] \arrow[drr, bend left, |->, "dName"] & & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] \arrow[rrr, bend right=40, |->, "sName"'] & & \text{professor} \arrow[ul, bend left, |->, "d'"] \arrow[r, bend left, |->, "pId"] \arrow[r, bend right, "pName"'] & v
\end{tikzcd}
$$

subject to commutivity of

$$
\begin{tikzcd}
 & \text{department} & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] & & \text{professor} \arrow[ul, "d'"']
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
 & \text{department} \arrow[dr, "h"] & \\
\text{professor} \arrow[ur, "d'"] \arrow[rr, "id"'] & & \text{professor}
\end{tikzcd}
$$

Characterisation of Logical Data Specification

Definition

A data specification S is *logical* iff

- there does not exist an edge e of the sketch S for which there is a decomposition in the theory category $\mathbf{C}(S)$ i.e. such that for some morphisms f_1 and f_2 distinct from e, $e = f_1 \circ f_2$.

Best Practice – Structured Systems Analysis and Design

Current Best Practice

$$\text{logical data specification} \xrightarrow{\textit{Chen's 1976 transform (automated)}} \text{relational data specification} \xrightarrow{\textit{normalise a la Codd (manual)}} \text{relational data specification}$$

Chen's Transformation 1976

Construction

From a logical data specification construct a relational data specification .

Chen's 1976 Method Replace $f : a \to b$ in the sketch by edges $f_1, ...f_n$ where $m_1, ...m_n$ is a v-valued mono-source with domain b and add inclusion dependency $a[f_1, ...f_n] \subseteq b[m_1, ...m_n]$.

Chen's Transformation 1976

Construction

From a logical data specification construct a relational data specification .

Chen's 1976 Method Replace $f : a \to b$ in the sketch by edges $f_1, \ldots f_n$ where $m_1, \ldots m_n$ is a v-valued mono-source with domain b and add inclusion dependency $a[f_1, \ldots f_n] \subseteq b[m_1, \ldots m_n]$.

Problem with this method

- Doesn't take account of commutative diagrams,
- therefore resulting relational specification
  - doesn't have equivalent theory category,
  - often is not be in normal form.
- This weakness negatively impacts how data specifications are written and maintained.

Chen's Transformation 1976 made diagram aware

Construction

with the same theory category

From a logical data specification construct a relational data specification .

Chen's 1976 Method Replace $f : a \to b$ in the sketch by edges $f_1, \ldots f_n$ where $m_1, \ldots m_n$ is a v-valued mono-source with domain b and add inclusion dependency $a[f_1, \ldots f_n] \subseteq b[m_1, \ldots m_n]$.

Problem with this method

- Doesn't take account of commutative diagrams,
- therefore resulting relational specification
  - doesn't have equivalent theory category,
  - often is not be in normal form.
- This weakness negatively impacts how data specifications are written and maintained.

Mission

- Theoretically justify an improved algorithm, i.e. one that takes account of commutative diagrams, and thereby change how data specifications are managed and databases are programmed.

Revised Best Practice

$$\begin{array}{l}\text{logical}\\\text{data}\\\text{specification}\end{array} \xrightarrow{\begin{array}{l}\textit{diagram}\\\textit{aware}\\\textit{transformation}\\\textit{(automated)}\end{array}} \begin{array}{l}\text{relational}\\\text{data}\\\text{specification}\end{array}$$

Such that

- If appropriate goodness criteria met by the logical specification then the relational specification meets the classic relational goodness criteria.

Impact

- No manual normalisation process.
- No source code required to describe the physical level.

Nested Relational Data – Same information as before.

department					
		student		professor	
<u>name</u>	hd	<u>name</u>	svr	<u>no</u>	name
maths	#3	bohm	#1	#1	scott
		smith	#2	#2	smith
				#3	gandy
phil	#1	gray	#1	#1	smith
		doe	#1	#2	ayer
history	#5	...		...	
physics	#1	...		...	

$$student[.., svr] \subseteq professor[.., pId]$$

$$department[identity, hd] \subseteq professor[.., pId]$$

Nested Relational Data – Same information as before.

<table>
<tr><th colspan="6">department</th></tr>
<tr><th rowspan="2">name</th><th rowspan="2">hd</th><th colspan="2">student</th><th colspan="2">professor</th></tr>
<tr><th>name</th><th>svr</th><th>no</th><th>name</th></tr>
<tr><td rowspan="3">maths</td><td rowspan="3">#3</td><td>bohm</td><td>#1</td><td>#1</td><td>scott</td></tr>
<tr><td>smith</td><td>#2</td><td>#2</td><td>smith</td></tr>
<tr><td></td><td></td><td>#3</td><td>gandy</td></tr>
<tr><td rowspan="2">phil</td><td rowspan="2">#1</td><td>gray</td><td>#1</td><td>#1</td><td>smith</td></tr>
<tr><td>doe</td><td>#1</td><td>#2</td><td>ayer</td></tr>
<tr><td>history</td><td>#5</td><td colspan="2">...</td><td colspan="2">...</td></tr>
<tr><td>physics</td><td>#1</td><td colspan="2">...</td><td colspan="2">...</td></tr>
</table>

what we see here – a combination of

- structural containment
- relational referencing.

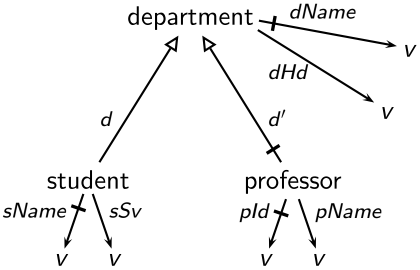
Nested Relational Data – Same information as before.

department					
name	hd	student		professor	
		name	svr	no	name
maths	#3	bohm	#1	#1	scott
		smith	#2	#2	smith
				#3	gandy
phil	#1	gray	#1	#1	smith
		doe	#1	#2	ayer
history	#5	…		…	
physics	#1	…		…	

this is all there is

- the sole mechanisms for representing internal relationships in data are
  - structural containment
  - relational referencing.
- ∴ all data can be viewed abstractly as nested relational,

Hierarchical Data Specification Sketch



$$student[d, svr] \subseteq professor[d', no]$$
$$department[identity, hd] \subseteq professor[d', no]$$

Characterisation of Physical Data Specification

Definition

A data specification is *physical* iff

- every non-v-node is the domain of at most one edge of the <u>*non-v* $\rightarrow$ *non-v*</u> type.

In a physical data specification every node and every edge has physical significance in the database or message structure.

- Nodes other than v in a physical data specification represent entity types (ER-notation) or tables (relational) or structs (IDL) or similar.
- Edges of the <u>*non-v* $\rightarrow$ *non-v*</u> type represent those relationships in the data that are physically represented by *structural containment*.
- Remaining edges (i.e. those of the <u>*non-v* $\rightarrow$ *v*</u> type) represent attributes (ER) or columns of tables (relational) or scalar fields within structs (IDL) or such like.

Subtle annotation of the logical sketch.

$$
\begin{tikzcd}
& \text{department} \arrow[drr, bend left, "\mathit{dName}"] \arrow[dr, bend left, "h"] & & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] \arrow[rrr, bend right, "\mathit{sName}"'] & & \text{professor} \arrow[ul, bend left, "d'"] \arrow[r, bend left, "\mathit{pId}"] \arrow[r, bend right, "\mathit{pName}"'] & v
\end{tikzcd}
$$

subject to commutivity of

$$
\begin{tikzcd}
& \text{department} & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] & & \text{professor} \arrow[ul, "d'"']
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
& \text{department} \arrow[dr, "h"] & \\
\text{professor} \arrow[ur, "d'"] \arrow[rr, "id"'] & & \text{professor}
\end{tikzcd}
$$

Subtle annotation of the logical sketch.

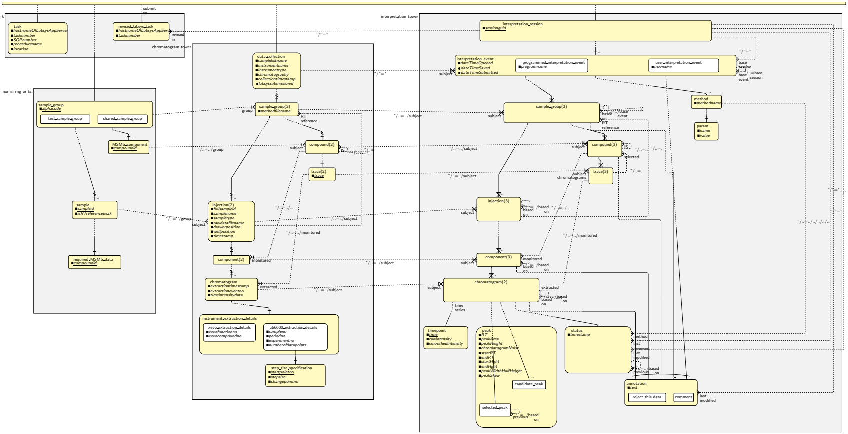
$$
\begin{tikzcd}
& \text{department} \arrow[dr, bend left, "h"] \arrow[drr, bend left=40, "dName"] & & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] \arrow[rrr, bend right=40, "sName"'] & & \text{professor} \arrow[ul, bend left, "d'"] \arrow[r, bend left, "pId"] \arrow[r, bend right, "pName"'] & v
\end{tikzcd}
$$

subject to commutivity of

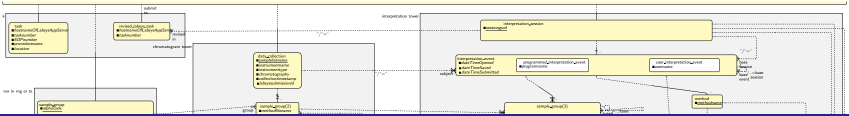
$$
\begin{tikzcd}
& \text{department} & \\
\text{student} \arrow[ur, "d"] \arrow[rr, "s"'] & & \text{professor} \arrow[ul, "d'"']
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
& \text{department} \arrow[dr, "h"] & \\
\text{professor} \arrow[ur, "d'"] \arrow[rr, "id"'] & & \text{professor}
\end{tikzcd}
$$

$d \in department, x \in student(d) \vdash s(x) \in professor(d)$
and $d \in department \vdash h(d) \in professor(d)$

Example – LCMSMS Data

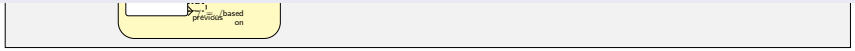


Example – LCMSMS Data

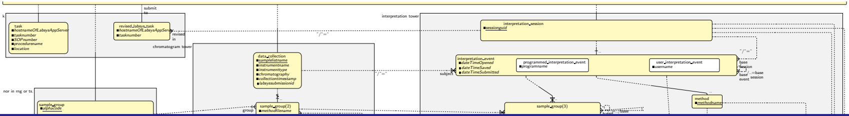


This example has

- 33 relationships implemented by structural containment,
- 26 relationships implemented by relational referencing (inclusion depedencies),

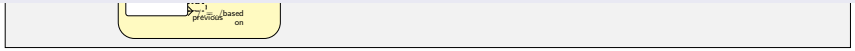


Example – LCMSMS Data

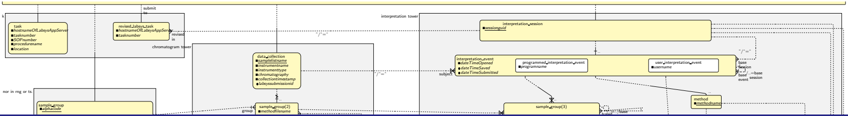


This example has

- 33 relationships implemented by structural containment,
- 26 relationships implemented by relational referencing (inclusion depedencies),
- 16 non-trivial commutative diagrams,



Example – LCMSMS Data



This example has

- 33 relationships implemented by structural containment,
- 26 relationships implemented by relational referencing (inclusion depedencies),
- 16 non-trivial commutative diagrams,
- 6 pullback diagrams.

Generated into code in XML, ECMA Javascript and Python.

Data Specification Instances and Requirements

- A *data specification* is a sketch S for a γ-structured category $\mathbf{C}(S)$.
- An *instance* of a data specification S is a structure preserving functor $D : \mathbf{C}(S) \to \mathbf{Par}$.
- A *requirement* for a data specification S is a set of such instances i.e. is a set R_C of structure preserving functors where for each $D \in R_C$, $D : \mathbf{C}(S) \to \mathbf{Par}$.

Fundamental Principles of Data Specification

If S is a sketch for γ-structured category $\mathbf{C}$ and if S is considered as a data specification with requirement $\mathbf{R_C}$

- **Principle 1 :** No redundancy. The sketch S ought to be a minimum sketch for structured category $\mathbf{C}$ i.e. there should be no subsketch of S which generates $\mathbf{C}$.
- **Principle 2:** $\mathbf{C}$ ought to be *maximally constrained* to $\mathbf{R_C}$. When defined, this will be the most fundamental way of saying that $\mathbf{C}$ is a tightest fit to the facts $\mathbf{R_C}$.

Representational Completeness — Goodness Critera

Another way of approaching tightest fit:

- That which is in the requirement and can be represented in the theory should be represented in the theory.

Representational Completeness — Goodness Critera

Another way of approaching tightest fit:

- That which is in the requirement and can be represented in the theory should be represented in the theory.
- To make precise we can give definitions of *representational completeness* wrt $\mathbf{R_C}$

Goodness Criteria 2A.	equational completeness,
Goodness Criteria 2B.	functional completeness,
Goodness Criteria 2C.	referential completeness,

and others beside.

Representational Completeness — Goodness Critera

Another way of approaching tightest fit:

- That which is in the requirement and can be represented in the theory should be represented in the theory.
- To make precise we can give definitions of *representational completeness* wrt $\mathbf{R_C}$

Goodness Criteria 2A.	equational completeness,
Goodness Criteria 2B.	functional completeness,
Goodness Criteria 2C.	referential completeness,

and others beside.

- In these definitions that $\mathbf{C}$ is x complete wrt $\mathbf{R_C}$ will mean exactly that the set of instances $\mathbf{R_C}$ are jointly reflective of x.

Equational Completeness — Goodness Criteria 2A

Equational Completeness — Goodness Criteria 2A

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances, then say that $\mathbf{C}$ is *equationally complete* with respect to the requirement $\mathbf{R_C}$ iff all path equivalences with respect to R_C are represented in $\mathbf{C}$ i.e. iff for all diagrams

$a \underset{g}{\overset{f}{\rightrightarrows}} b$ in $\mathbf{C}$, if in all instances $D \in \mathbf{R_C}$, $D(f) = D(g)$, then $f = g$.

In other words,

- loosely speaking ... if $f = g$ in all data instances then $f = g$,
- or...the set of functors $\mathbf{R_C}$ is jointly faithful.

Equational Completeness — Goodness Criteria 2A

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances, then say that $\mathbf{C}$ is *equationally complete* with respect to the requirement $\mathbf{R_C}$ iff all path equivalences with respect to R_C are represented in $\mathbf{C}$ i.e. iff for all diagrams

$$\begin{tikzcd} a \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & b \end{tikzcd}$$

in $\mathbf{C}$, if in all instances $D \in \mathbf{R_C}$, $D(f) = D(g)$, then $f = g$.

In other words,

- loosely speaking ... if $f = g$ in all data instances then $f = g$,
- or...the set of functors $\mathbf{R_C}$ is jointly faithful.

Goodness Criteria 2A: If S is a sketch for γ-structured category $\mathbf{C}$ considered as a data specification with requirement $\mathbf{R_C}$ then $\mathbf{C}$ ought to be equationally complete with respect to R_C.

Functional Dependencies — Goodness Criteria 2B

To describe Goodness Criteria 2B I first need to

- Define what we mean by *functional dependency* – abstracted and simplified from definition given by Codd 1971.
- Define what we mean by a functional dependency being *represented* – inspired by language found in Zaniolo 1982.
- State as the criteria that all functional dependencies ought to be represented – the spirit of Zaniolo's paper.

Definition of Functional Dependency

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances and if

$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$

in $\mathbf{C}$ then there is a *functional dependency* of g on f with respect to $\mathbf{R_C}$ iff there is a family of functions $H_D)_{D \in \mathbf{R_C}}$ such that in each instance D, H_D is a partial function $H_D : D(b) \to D(c)$, such that both

$$\overline{H_D} = \widehat{D(f)}$$

and

$$D(f) \circ H_D = D(g).$$

- H_D will be the unique such partial function (this follows from RR.5),

Definition of Functional Dependency

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances and if

$$\begin{tikzcd}[row sep=small] & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$

in $\mathbf{C}$ then there is a *functional dependency* of g on f with respect to $\mathbf{R_C}$ iff there is a family of functions $H_D)_{D \in \mathbf{R_C}}$ such that in each instance D, H_D is a partial function $H_D : D(b) \to D(c)$, such that both

$$\overline{H_D} = \widehat{D(f)}$$

and

$$D(f) \circ H_D = D(g).$$

- H_D will be the unique such partial function (this follows from RR.5),
- If H is such a functional dependency then we say that $f \xrightarrow{H} g$ in $\mathbf{C}$ with respect to $\mathbf{R_C}$.

Functional Dependencies — Goodness Criteria 2B

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances, if

$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$

in $\mathbf{C}$ and if there is a functional dependency $f \xrightarrow{H} g$ then say that the functional dependency H is *represented* in $\mathbf{C}$ iff there exists a morphism $h : b \to c$ in $\mathbf{C}$ such that $D(h) = H_D$.

Functional Dependencies — Goodness Criteria 2B

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances, if

$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$

in $\mathbf{C}$ and if there is a functional dependency $f \xrightarrow{H} g$ then say that the functional dependency H is *represented* in $\mathbf{C}$ iff there exists a morphism $h : b \to c$ in $\mathbf{C}$ such that $D(h) = H_D$.

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ a set of instances then $\mathbf{C}$ is said to be *functionally complete* with respect to $\mathbf{R_C}$ iff every functional dependency present in $\mathbf{R_C}$ is represented in $\mathbf{C}$. Loosely speaking ... whenever g factors through f in every data instance then g should factor through f.

Functional Dependencies — Goodness Criteria 2B

Definition

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances, if

$$\begin{tikzcd} & b \\ a \arrow[ur, "f"] \arrow[dr, "g"'] & \\ & c \end{tikzcd}$$

in $\mathbf{C}$ and if there is a functional dependency $f \xrightarrow{H} g$ then say that the functional dependency H is *represented* in $\mathbf{C}$ iff there exists a morphism $h : b \to c$ in $\mathbf{C}$ such that $D(h) = H_D$.

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ a set of instances then $\mathbf{C}$ is said to be *functionally complete* with respect to $\mathbf{R_C}$ iff every functional dependency present in $\mathbf{R_C}$ is represented in $\mathbf{C}$. Loosely speaking ... whenever g factors through f in every data instance then g should factor through f.

Goodness Criteria 2B: If S is a sketch for γ-structured category $\mathbf{C}$ considered as a data specification with requirement $\mathbf{R_C}$ then $\mathbf{C}$ ought to be functionally complete with respect to $\mathbf{R_C}$.

Definition of Inclusion Dependencies

If $\mathbf{C}$ is a γ-structured category and $\mathbf{R_C}$ is a set of instances and if

$$\begin{tikzcd} & b_i & \\ a \arrow[ur, "f_i"] & & c \arrow[ul, "q_i"'] \end{tikzcd}$$

in $\mathbf{C}$, for i, $1 \leq i \leq n$, then an *inclusion dependency* J, written $a[f_1, ...f_n] \overset{J}{\subseteq} c[q_1, ..q_n]$, is a family of functions $J_D)_{D \in \mathbf{R_C}}$ such that each instance $D \in \mathbf{R_C}$, $J_D : D(a) \rightarrow D(c)$ is a partial function (so that we have this diagram in **Par**

$$\begin{tikzcd} & D(b_i) & \\ D(a) \arrow[ur, "D(f_i)"] \arrow[rr, "J_D"'] & & D(c) \arrow[ul, "D(q_i)"'] \end{tikzcd}$$

) and J_D satisfies

$$\overline{J_D} = \overline{\langle D(f_1), ...D(f_n) \rangle} \tag{1}$$

and, for each i, $1 \leq i \leq n$,

$$J_D \circ D(q_i) = \overline{(}J_D) \circ D(f_i) \tag{2}$$

or, equivalent to (2) in the presence of (1):

$$J_D \circ \langle D(q_1), ...D(q_n) \rangle = \langle D(f_1), ...D(f_n) \rangle \tag{3}$$

- If each J_D is the unique such function then the inclusion dependency is said to be referential.

Referential Completeness and Goodness Criteria 2C

Definition

If **C** is a category and $\mathbf{R_C}$ is a set of instances and if

$$\begin{tikzcd} & b_1 & \\ a \arrow[ur, "f_1"] \arrow[dr, "f_n"'] & & c \arrow[ul, "q_1"'] \arrow[dl, "q_n"] \\ & b_n & \end{tikzcd}$$

in **C** and if $a[f_1, ...f_n] \overset{J}{\subseteq} c[q_1, ..q_n]$ is a referential inclusion dependency with respect to $\mathbf{R_C}$ then say that the inclusion dependency J is *represented* in **C** iff there exists a morphism $j : a \to c$ in **C** such that in each instance $D \in \mathbf{R_C}$, $D(j) = J_D$.

If **C** is a category and $\mathbf{R_C}$ a set of instances then **C** is *referentially complete* with respect to $\mathbf{R_C}$ iff all referential inclusion dependencies present in $\mathbf{R_C}$ are represented in **C**.

Goodness Criteria 2C: If S is a sketch for γ-structured category **C** considered as a data specification with requirement $\mathbf{R_C}$ then **C** ought to be referentially complete with respect to $\mathbf{R_C}$.

BCNF in the abstract (based on Zaniolo 1982 Definition 2)

If S is a simple relational sketch for a γ-structured category $\mathbf{C}$ and S is considered as a data specification with requirement $\mathbf{R_C}$, then it ought to be the case that if

$$\begin{tikzcd}[row sep=small] & b_1 \\ & b_2 \\ a \arrow[ur, "x_1"] \arrow[r, "x_2"'] \arrow[dr, "x_n"'] \arrow[ddr, "y"'] & \vdots \\ & b_n \\ & c \end{tikzcd}$$

are edges of S and if

$\{x_1, \ldots x_n\} \to y$ is a non-trivial functional dependency between these edges

BCNF in the abstract (based on Zaniolo 1982 Definition 2)

If S is a simple relational sketch for a γ-structured category $\mathbf{C}$ and S is considered as a data specification with requirement $\mathbf{R_C}$, then it ought to be the case that if

$$\begin{tikzcd}
 & b_1 \\
 & b_2 \\
a \arrow[ur, "x_1"] \arrow[urr, "x_2"] \arrow[dr, "x_n"] \arrow[ddr, "y"] & \vdots \\
 & b_n \\
 & c
\end{tikzcd}$$

are edges of S and if $\{x_1, \ldots x_n\} \to y$ is a non-trivial functional dependency between these edges then

$$\begin{tikzcd}
 & b_1 \\
a \arrow[ur, "x_1"] \arrow[r, "x_2"] \arrow[ddr, "x_n"] & b_2 \\
 & \vdots \\
 & b_n
\end{tikzcd}$$

is a designated mono-source.

Deriving the classic normal form criteria

Lemma

(i) *For a simple relational data specification S with requirement $\mathbf{R_C}$, if S meets the minimality condition (principle 1) and $\mathbf{C}(S)$ meets the goodness condition 2B then S meets the condtions of Codd's third normal form.*

(ii) *In addition to (i), if for each designated mono-source $< m_1, \ldots m_n >$ of the associated logical sketch, each m_i is an edge then the data specification S meets the conditions of Boyce-Codd normal form (BCNF).*

(iii) *In addition to (i), if we follow principle 1 and* **do not introduce limits into a sketch needlessly** *then the data specification S meets the fourth and fifth normal form criteria of Fagin.*

Deriving the classic normal form criteria

Lemma

(i) *For a simple relational data specification S with requirement $\mathbf{R_C}$, if S meets the minimality condition (principle 1) and $\mathbf{C}(S)$ meets the goodness condition 2B then S meets the condtions of Codd's third normal form.*

(ii) *In addition to (i), if for each designated mono-source $< m_1, ...m_n >$ of the associated logical sketch, each m_i is an edge then the data specification S meets the conditions of Boyce-Codd normal form (BCNF).*

(iii) *In addition to (i), if we follow principle 1 and* **do not introduce limits into a sketch needlessly** *then the data specification S meets the fourth and fifth normal form criteria of Fagin.*

Significance

We have defined criteria which are generic in the sense that they apply to any kind of data specification. They genericise the classic relational normal form criteria.

Definition: **C** *maximally constrained* to $\mathbf{R_C}$

- Question – is there a **C'** that extends **C** and that will do a better job.
- Is there a **C'** and an $I : \mathbf{C} \to \mathbf{C'}$ such that all instances in the requirement $\mathbf{R_C}$ uniquely factor though I

$$
\begin{tikzcd}
C' \arrow[dr, "D'"] & \\
 & \mathbf{Par} \\
C \arrow[uu, "I"] \arrow[ur, "D"] \arrow[ur, bend right, "F"'] &
\end{tikzcd}
$$

and at least one other instance F of **C** does not factor through I.

Definition: $\mathbf{C}$ *maximally constrained* to $\mathbf{R_C}$

- Question – is there a **C'** that extends **C** and that will do a better job.
- Is there a **C'** and an $I : \mathbf{C} \to \mathbf{C'}$ such that all instances in the requirement $\mathbf{R_C}$ uniquely factor though I

$$\begin{tikzcd} C' \arrow[rr, "D'"] & & \mathbf{Par} \\ C \arrow[u, "I"] \arrow[urr, "D"] \arrow[urr, bend right, "F"'] & & \end{tikzcd}$$

and at least one other instance F of **C** does not factor through I.

If there is no such $I : \mathbf{C} \to \mathbf{C'}$ then we shall say that **C** is *maximally constrained* with respect to $\mathbf{R_C}$.

...meaning that structured category **C** is the tightest possible fit to facts i.e. to the requirement $R_{\mathbf{C}}$.

Principles imply specific criteria

Principles imply specific criteria

- We would like to show (the grand plan) that sketch S meets Principle 1 (minimality of the sketch) and if $\mathbf{C}(S)$ meets Principle 2 (that it should be maximally constrained) then it also meets specific representational completeness criteria 2A, 2B, 2C and so on.

Principles imply specific criteria

- We would like to show (the grand plan) that sketch S meets Principle 1 (minimality of the sketch) and if $\mathbf{C}(S)$ meets Principle 2 (that it should be maximally constrained) then it also meets specific representational completeness criteria 2A, 2B, 2C and so on.
- If we can get to this then we have fundamental principles which are both generic across all kinds of data specifications and which imply the specific representation completeness criteria which in turn imply the classic relational normal forms.

Revised Best Practice

$$\text{logical data specification} \xrightarrow{\textit{diagram aware transformation (automated)}} \text{relational data specification}$$

Such that

- If appropriate goodness criteria met by the logical specification then the relational specification meets the classic relational goodness criteria.

Impact

- No manual normalisation process.
- No source code required to describe the physical level.