Data Transformation by Calculation

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

GTTSE'07
2–7 July 2007
Braga

First lecture

Schedule: Monday July 2nd, 5pm-6pm

Learning outcomes:

- Identifying the **problem**
- Finding a **strategy** to face it

Motivation

- **Data** play an important **rôle** in our lifes (eg. medical records, bank details, CVs, ...)
- Information system **quality** is highly dependent upon consistency and reliability of data
- Data are **everywhere** in computing — statically (eg. machine states, databases) and dynamically (eg. messages, APIs, forms, etc)
- Data are what is left from the **past** (cf. historical archives)

However...

Motivation

- Data keep changing **format**
- No two people **think** data in the same way
- Data modeling is **technology** sensitive
- **Impedance mismatch** among data models
- Need for data **migration** software
- Data always put at **risk** — loss or damage

Motivation

Quoting Lämmel and Meijer (GTTSE'05):

- "Whatever programming **paradigm** for data processing we choose, data has the tendency to live on the other side or to eventually end up there. (...)
- This myriad of inter- and intra-paradigm data models calls for a good understanding of techniques for **mappings** between data **models**, actual **data**, and **operations** on data. (...)
- Given the fact that IT industry is fighting with various **impedance mismatches** and **data-model evolution** problems for decades, it seems to be safe to start a research career that specifically addresses these problems".

Our strategy in this tutorial:

Don't invent data mappings any more: calculate them!

Motivation

Quoting Lämmel and Meijer (GTTSE'05):

- "Whatever programming **paradigm** for data processing we choose, data has the tendency to live on the other side or to eventually end up there. (...)
- This myriad of inter- and intra-paradigm data models calls for a good understanding of techniques for **mappings** between data **models**, actual **data**, and **operations** on data. (...)
- Given the fact that IT industry is fighting with various **impedance mismatches** and **data-model evolution** problems for decades, it seems to be safe to start a research career that specifically addresses these problems".

Our strategy in this tutorial:

Don't invent data mappings any more: calculate them!

Motivation

Quoting Lämmel and Meijer (GTTSE'05):

- "Whatever programming **paradigm** for data processing we choose, data has the tendency to live on the other side or to eventually end up there. (...)
- This myriad of inter- and intra-paradigm data models calls for a good understanding of techniques for **mappings** between data **models**, actual **data**, and **operations** on data. (...)
- Given the fact that IT industry is fighting with various **impedance mismatches** and **data-model evolution** problems for decades, it seems to be safe to start a research career that specifically addresses these problems".

Our strategy in this tutorial:

Don't invent data mappings any more: calculate them!

Interacting with machines

Problems can arise anywhere at any time: even using a pocket calculator

$$
\begin{tikzcd}
\mathit{digits} \arrow[d, bend left, "\mathit{keyin}"] \\
\mathit{binary} \arrow[u, bend left, "\mathit{display}"]
\end{tikzcd}
$$

digits need to reach the machine binary so that it... calculates!

$$
\begin{tikzcd}
\mathit{digits} & & \mathit{digits} \arrow[d, bend left, "\mathit{keyin}"] \\
\mathit{binary} \arrow[u, bend left, "\mathit{display}"] & & \mathit{binary} \arrow[ll, bend left, "\sqrt{}"]
\end{tikzcd}
$$

Likely faults

- digit displayed not always the one whose key was pressed (**confusion**)
- nothing at all displayed (**loss**)
- required operation yields wrong output (**miscalculation**)

What about "**inside** the machine"?

- HCI is just a special case of **subcontracting** (a service)
- Subcontracting spreads over mutiple **layers**, different technologies
- Uncountable number of data mappings at work in **transactions** and layer inter-communication.

Likely faults

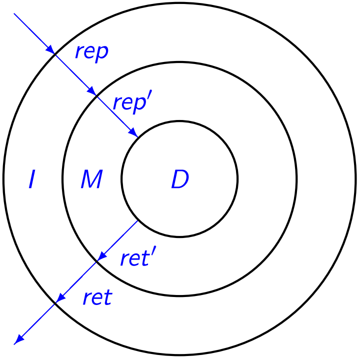
- digit displayed not always the one whose key was pressed (**confusion**)
- nothing at all displayed (**loss**)
- required operation yields wrong output (**miscalculation**)

What about "**inside** the machine"?

- HCI is just a special case of **subcontracting** (a service)
- Subcontracting spreads over mutiple **layers**, different technologies
- Uncountable number of data mappings at work in **transactions** and layer inter-communication.

Weaving data through I-M-D architecture

Layered-architectures rely on sub-contracting:

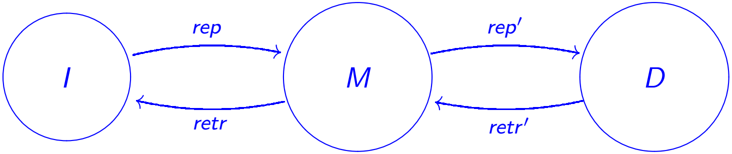


Legend:

I — interface
M — middleware
D — dataware
rep — represent
ret — retrieve

The same in different geometry

Separation principles (eg. Seheim model, client-server, etc) entail permanent data conversion across disparate technology layers:



Running example — genealogy website (I)

At **GUI** level, clients wish to see and browse their family trees:

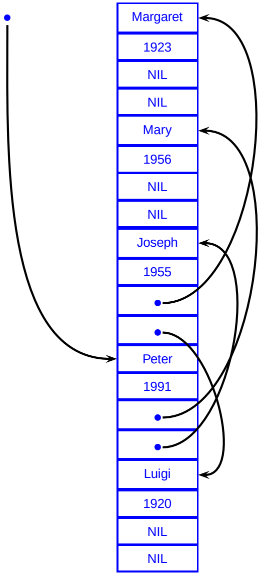
Margaret, b. 1923   Luigi, b. 1920   (1)

Mary, b. 1956   Joseph, b. 1955

Peter, b. 1991

Running example — genealogy website (M)

Trees become "more **concrete**" as they go down the layers of software architecture;



They convert to **pointer** structures (eg. in C++/C#) stored in dynamic heaps once reaching **middleware**.

Running example — genealogy website (D)

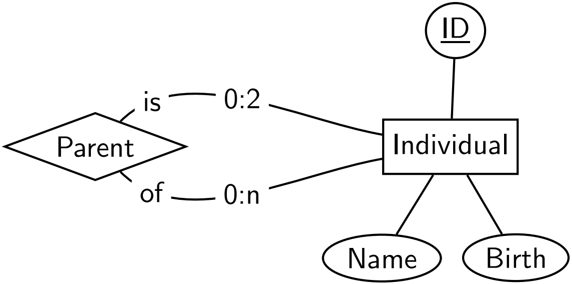
Finally channeled to dataware, heap structures are buried into **database** files as persistent data **records**:

ID	Name	Birth
1	Joseph	1955
2	Luigi	1920
3	Margaret	1923
4	Mary	1956
5	Peter	1991

ID	Ancestor	ID
5	Father	1
5	Mother	4
1	Father	2
1	Mother	3

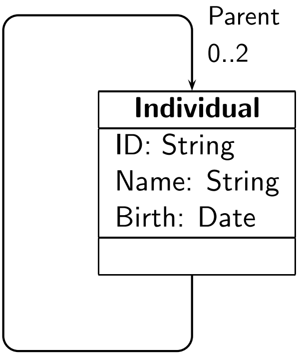
Too many paradigms

Data modeling notations, eg. **Entity-Relationship** (ER) diagrams



Too many paradigms

UML class diagrams



Too many paradigms

XML (version 1)

```
<!-- DTD for genealogical trees -->
<!ELEMENT tree (node+)>
<!ELEMENT node (name, birth, mother?, father?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birth (#PCDATA)>
<!ELEMENT mother EMPTY>
<!ELEMENT father EMPTY>
<!ATTLIST tree
  ident ID #REQUIRED>
<!ATTLIST mother
  refid IDREF #REQUIRED>
<!ATTLIST father
  refid IDREF #REQUIRED>
```

Too many paradigms

XML (version 2)

```
<!-- DTD for genealogical trees -->
<!ELEMENT tree (name, birth, tree?, tree?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birth (#PCDATA)>
```

Too many (programming) paradigms

Plain **SQL**

```
CREATE TABLE INDIVIDUAL (
  ID  NUMBER  (10) NOT NULL,
  Name   VARCHAR (80) NOT NULL,
  Birth  NUMBER  (8)  NOT NULL,
  CONSTRAINT INDIVIDUAL_pk PRIMARY KEY(ID)
);

CREATE TABLE ANCESTORS (
  ID        VARCHAR (8)  NOT NULL,
  Ancestor VARCHAR (8)  NOT NULL,
  PID       NUMBER  (10) NOT NULL,
  CONSTRAINT ANCESTORS_pk PRIMARY KEY (ID,Ancestor)
);
```

Too many (programming) paradigms

C/C++ etc

```
typedef struct Gen {
        char *name            /* name is a string */
        int   birth           /* birth year is a number */
        struct Gen *mother; /* genealogy of mother (if known) */
        struct Gen *father; /* genealogy of father (if known) */
        } ;
```

Haskell etc

```
    data PTree = Node {
            name   :: [ Char ],
            birth  :: Int      ,
            mother :: Maybe PTree,
            father :: Maybe PTree
            }
```

Questions

- Are all these data models "**equivalent**"?
- If so, in what sense?
- If not, how can they be ranked in terms of "**quality**"?
- How can we tell apart the **essence** of a data model from its **technology** wrapping?

Is there a notation unifying all the above?

Keep it simple

Let us write

$$c \ R \ a$$

to mean that

datum c (eg. byte) **represents** *datum a (eg. digit)*

and let the converse fact

$$a \ R^{\circ} \ c$$

mean

a is the datum represented by c

(**passive** voice).

Keep it simple

Let us write

$$c \; R \; a$$

to mean that

datum c *(eg. byte)* **represents** *datum* a *(eg. digit)*

and let the converse fact

$$a \; R^{\circ} \; c$$

mean

a *is the datum represented by* c

(**passive** voice).

No confusion, please

Definite article "*the*" instead of "*a*" in sentence

a is the datum represented by c

already a symptom of the **no confusion** principle: we want c to represent **only one** datum of interest.

So R should be **injective**:

$$\langle \forall\ c, a, a' \ :: \ c\ R\ a \wedge c\ R\ a' \ \Rightarrow \ a = a' \rangle \tag{2}$$

No confusion, please

Definite article "*the*" instead of "*a*" in sentence

a is the datum represented by c

already a symptom of the **no confusion** principle: we want c to represent **only one** datum of interest.

So R should be **injective**:

$$\langle \forall\ c, a, a' \ :: \ c\ R\ a \wedge c\ R\ a' \ \Rightarrow \ a = a' \rangle \tag{2}$$

No data loss, please

No loss principle: no data are lost in the representation process,

$$\langle \forall\ a\ ::\ \langle \exists\ c\ ::\ c\ R\ a \rangle \rangle \tag{3}$$

ie. every datum a is representable — R is **totally** defined. In a diagram:

$$\begin{tikzcd} A \arrow[r, bend left, "R"] & C \arrow[l, bend left, "R^\circ"] \end{tikzcd} \tag{4}$$

for R injective and totally defined

Freeing the retrieve relation

Useful (in general) to give some freedom to the retrieve relation, say F, provided that it **connects with** the chosen representation:

$$\langle \forall\ a, c\ ::\ c\ R\ a\ \Rightarrow\ a\ F\ c \rangle \tag{5}$$

(= "if c represents a then a can be retrieved from c).

In a diagram:

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "F"] \end{tikzcd} \tag{6}$$

(Meaning of $\leq$ to be explained soon.)

Mapping scenarios

Diagram

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "F"] \end{tikzcd}$$

already captures some of the ingredients of Lämmel and Meijer's **mapping scenarios**:

- the **type**-*level mapping* of a source data model (A) to a target data model (C);
- two maps — "**map forward**" (R) and "**map backward**" (F) — between source / target data;
- the **transcription level** mapping of source operations into target operations — see next slide

Mapping scenarios

Diagram

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "F"] \end{tikzcd}$$

already captures some of the ingredients of Lämmel and Meijer's **mapping scenarios**:

- the **type**-*level mapping* of a source data model (A) to a target data model (C);
- two maps — "**map forward**" (R) and "**map backward**" (F) — between source / target data;
- the **transcription level** mapping of source operations into target operations — see next slide

Transcription level

Source (eg. CRUD) **operations** mapped to target operations — put two $\leq$-diagrams together:

$$\begin{tikzcd}
A \arrow[rr, bend left, "R"] \arrow[rr, phantom, "\leq"] \arrow[d, "O"'] & & C \arrow[ll, bend left, "F"] \arrow[d, "P"] \\
B \arrow[rr, bend left, "R'"] \arrow[rr, phantom, "\leq"] & & D \arrow[ll, bend left, "F'"]
\end{tikzcd} \tag{7}$$

The (safe) transcription of O into P can be formally stated by ensuring that the picture is a commutative diagram. (Details soon.)

Chaining

In general, it will make sense to chain two or more mapping scenarios, eg. between interface (I) and middleware (M), and between middleware and dataware (D):

$$
\begin{tikzcd}
I \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & M \arrow[l, bend left, "F"] \arrow[r, bend left, "R'"] \arrow[r, phantom, "\leq"] & D \arrow[l, bend left, "F'"]
\end{tikzcd}
$$

However, how can we be sure that mapping scenarios *compose* with each other?

Data refinement

- All questions so far are addressed in the well studied discipline of **data refinement**
- However, data refinement not "sexy enough" — too complex, too many symbols:

Proof of downwards simulation theorem for partial correctness (2)

3. Case $\beta \rightsquigarrow (\varphi \rightsquigarrow \psi) \,;\, \beta$**:**

$$\underbrace{\rho[a'/a] \wedge x' = x}_{=\beta} \rightsquigarrow (\varphi \rightsquigarrow \psi) \,;\, (\underbrace{\rho[a'/a] \wedge x' = x}_{=\beta}) = \quad \text{(by (2))}$$

$$\underbrace{\forall x_0', a_0'.(\rho[a_0'/a] \wedge x_0' = x)[x', c'/x, c] \rightarrow (\exists a.\rho \wedge \forall x_0.\varphi[x_0', a_0'/x, a] \rightarrow \psi)}_{= \rho[a_0'/a] \wedge x_0' = x \rightsquigarrow \exists a.\rho \wedge \forall x_0.\varphi[x_0', a_0'/x, a] \rightarrow \psi}$$

QED

I.e., $S \subseteq \beta \rightsquigarrow (\varphi \rightsquigarrow \psi) \,;\, \beta$ iff

$$\models \left\{\rho[a_0'/a] \wedge x_0' = x\right\} S \left\{\exists a.\rho \wedge \forall x_0.\varphi[x_0', a_0'/x, a] \rightarrow \psi\right\}$$

Can't we do better?

Interlude

Problem-solving strategy

Recall the *universal problem solving* strategy which one is taught at school:

- **understand** your problem
- build a mathematical **model** of it
- **reason** in such a model
- upgrade your model, if necessary
- **calculate** a final solution and implement it.

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{array}{cl} & 3x + 9 = 48 \\ \equiv & \{ \text{ "al-djabr" rule } \} \\ & 3x = 48 - 9 \\ \equiv & \{ \text{ "al-hatt" rule } \} \\ & x = 16 - 3 \end{array}$$

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{array}{ll} & 3x + 9 = 48 \\ \equiv & \{ \text{ “al-djabr” rule } \} \\ & 3x = 48 - 9 \\ \equiv & \{ \text{ “al-hatt” rule } \} \\ & x = 16 - 3 \end{array}$$

School maths example

The problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

The model

$$x + (x + 3) + (x + 6) = 48$$

The calculation

$$\begin{array}{ll} & 3x + 9 = 48 \\ \equiv & \qquad \{ \text{ “al-djabr” rule } \} \\ & 3x = 48 - 9 \\ \equiv & \qquad \{ \text{ “al-hatt” rule } \} \\ & x = 16 - 3 \end{array}$$

School maths example

The solution

$$\begin{aligned} x &= 13 \\ x+3 &= 16 \\ x+6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Have a look at Pedro Nunes (1502-1578) *Libro de Algebra en Arithmetica y Geometria* (dated 1567) ...

School maths example

The solution

$$\begin{aligned} x &= 13 \\ x + 3 &= 16 \\ x + 6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Have a look at Pedro Nunes (1502-1578) *Libro de Algebra en Arithmetica y Geometria* (dated 1567) . . .

School maths example

The solution

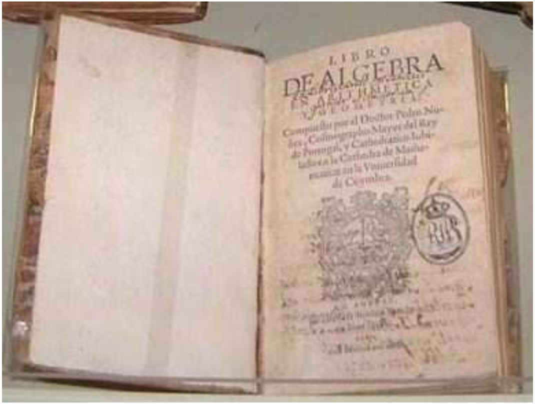
$$\begin{aligned} x &= 13 \\ x+3 &= 16 \\ x+6 &= 19 \end{aligned}$$

Questions....

- "al-djabr" rule ?
- "al-hatt" rule ?

Have a look at Pedro Nunes (1502-1578) *Libro de Algebra en Arithmetica y Geometria* (dated 1567) ...

Libro de Algebra en Arithmetica y Geometria (1567)



(...) the inventor of this art was a Moorish mathematician, whose name was Gebre, & in some libraries there is a small arabic treaty which contains chapters that we use
(fol. a ij r)

Reference to *On the calculus of al-gabr and al-muqâbala* by Abû Al-Huwârizmî, a famous 9c Persian mathematician.

Calculus of al-gabr, al-hatt and al-muqâbala

al-djabr

$$x - z \leq y \quad \equiv \quad x \leq y + z$$

al-hatt

$$x * z \leq y \quad \equiv \quad x \leq y * z^{-1} \qquad\qquad (z > 0)$$

al-muqâbala

Ex: $$4x^2 - 2x^2 = 2x + 6 - 3 \quad \equiv \quad 2x^2 = 2x + 3$$

“Algebra (...) is thing causing admiration”

> (...) Principalmente que vemos algumas vezes, no poder vn gran Mathematico resoluer vna question por medios Geometricos, y resolverla por Algebra, siendo la misma Algebra sacada de la Geometria, q̃ es cosa de admiraciõ.

ie.

> *(...) Mainly because we see often a great Mathematician unable to resolve a question by Geometrical means, and solve it by Algebra, being that same Algebra taken from Geometry, which is thing causing admiration.*

[*in* Nunes' *Libro de Algebra*, fols. 270–270v.]

Deduction first

> *Y tambien porque quien obra por Algebra va entendiendo la razon de la obra que haze, hasta la yqualacion ser acabada. (...) De suerte que, quien obra por Algebra, va haziendo discursos demonstrativos.*

ie.

> *And also because one performing by Algebra is understanding the reason of the work one does, until the equality is finished. (...) So much so that, who works by Algebra is doing a demonstrative discourse.*

[fol. 269r-269v]

Verdict

(...) De manera, que quien sabe por Algebra, sabe <u>scientificamente</u>.

(...) in this way, who knows by Algebra knows <u>scientifically</u>)

Trend for notation economy

Well-known throughout the history of maths — a kind of "natural language **implosion**" — particularly visible in the syncopated phase (16c), eg.

.40.p̃.2.ce. son yguales a .20.co

(P. Nunes, Coimbra, 1567) for nowadays $40 + 2x^2 = 20x$, or

B 3 in A quad - D plano in A + A cubo æquatur Z solido

(F. Viète, Paris, 1591) for nowadays $3BA^2 - DA + A^3 = Z$

Later on (18c, 19c, ...)

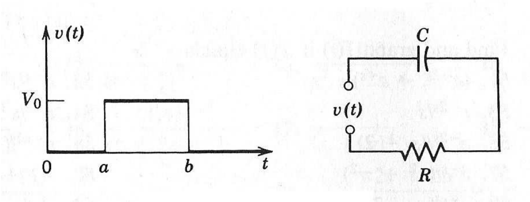
More demanding problems to be modelled/solved, eg. electrical circuits:

From a simple law ...

$V = R \times I$ by Georg Ohm (1789-1854) ...

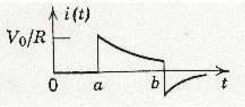
...to non-linear RC-circuits

$$v(t) = Ri(t) + \frac{1}{C}\int_0^t i(\tau)d\tau$$
$$v(t) = V_0(u(t-a) - u(t-b)) \qquad (b > a)$$



Calculate $i(t)$

The following $i(t)$ can be observed on an oscilloscope:



Can you explain it?

Is 16c maths still enough for the required calculations?

No. Need for the **differential/integral** calculus.

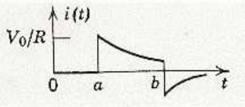
But there is more:

For the underlying maths to scale up

Need for an *integral transform*, eg. the Laplace transform.

Calculate $i(t)$

The following $i(t)$ can be observed on an oscilloscope:



Can you explain it?

Is 16c maths still enough for the required calculations?

No. Need for the **differential/integral** calculus.

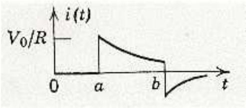
But there is more:

For the underlying maths to scale up

Need for an *integral transform*, eg. the Laplace transform.

Calculate $i(t)$

The following $i(t)$ can be observed on an oscilloscope:



Can you explain it?

Is 16c maths still enough for the required calculations?

No. Need for the **differential/integral** calculus.

But there is more:

For the underlying maths to scale up

Need for an *integral transform*, eg. the Laplace transform.

Laplace transform

t-space

s-space

Given problem

$$y'' + 4y' + 3y = 0$$
$$y(0) = 3$$
$$y'(0) = 1$$

Subsidiary equation

$$s^2 + 4sY + 3Y = 3s + 13$$

Solution of subs. equation

$$Y = \frac{-2}{s+3} + \frac{5}{s+1}$$

Solution of given problem

$$y(t) = -2e^{-3t} + 5e^{-t}$$

Laplace-transformed RC-circuit model

$\mathcal{L}$(t-space RC model) is

$$RI(s) + \frac{I(s)}{sC} \;=\; \frac{V_0}{s}(e^{-as} - e^{-bs})$$

whose *algebraic* solution for $I(s)$ is

$$I(s) \;=\; \frac{\frac{V_0}{R}}{s + \frac{1}{RC}}(e^{-as} - e^{-bs})$$

Now, the converse transformation:

$$\mathcal{L}^{-1}\left(\frac{\frac{V_0}{R}}{s + \frac{1}{RC}}\right) \;=\; \frac{V_0}{R}e^{-\frac{t}{RC}}$$

Analytical solution

After some algebraic manipulation we will obtain an analytical answer ...

$$i(t) \;=\; \begin{cases} 0 \quad\quad if \quad t < a \\ (\frac{V_0 e^{-\frac{a}{RC}}}{R})e^{-\frac{t}{RC}} \quad\quad if \quad a < t < b \\ (\frac{V_0 e^{-\frac{a}{RC}}}{R} - \frac{V_0 e^{-\frac{b}{RC}}}{R})e^{-\frac{t}{RC}} \quad\quad if \quad t > b \end{cases}$$

Question

All we have seen applies to physics, mechanical eng., civil eng., electrical and electronic eng.

What about us? (software engineers)

Question

All we have seen applies to physics, mechanical eng., civil eng., electrical and electronic eng.

What about us? (software engineers)

Need for a transform

Integration? Quantification?

$(\mathcal{L}\ f)s = \int_0^\infty e^{-st} f(t) dt$

$f(t)$	$\mathcal{L}(f)$
1	$\frac{1}{s}$
t	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^{n+1}}$
e^{at}	$\frac{1}{s-a}$
etc	

A parallel:

$$\langle \int \ x \ : \ 0 \leq x \leq 10 : \ x^2 - x \rangle$$

$$\langle \forall \ x \ : \ 0 \leq x \leq 10 : \ x^2 \geq x \rangle$$

An "s-space analog" for logical quantification

The pointfree (PF) transform

ϕ	$PF\ \phi$
$\langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle$	$b(R \cdot S)c$
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \forall\ x\ ::\ x\ R\ b \Rightarrow x\ S\ a \rangle$	$b(R \setminus S)a$
$\langle \forall\ c\ ::\ b\ R\ c \Rightarrow a\ S\ c \rangle$	$a(S\ /\ R)b$
$b\ R\ a \wedge c\ S\ a$	$(b, c)\langle R, S \rangle a$
$b\ R\ a \wedge d\ S\ c$	$(b, d)(R \times S)(a, c)$
$b\ R\ a \wedge b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ \ b)\ R\ (g\ \ a)$	$b(f^\circ \cdot R \cdot g)a$
$\textsc{True}$	$b\ \top\ a$
$\textsc{False}$	$b\ \bot\ a$

What are R, S, id ?

End of interlude

A transform for logic and set-theory

An old idea

$$PF(\text{sets, predicates}) = \text{binary relations}$$

Calculus of binary relations

- 1860 - introduced by De Morgan, embryonic
- 1941 - Tarski's school, cf. *A Formalization of Set Theory without Variables*
- 1980's - coreflexive models of sets (Freyd and Scedrov, Eindhoven school)

Unifying approach

Everything is a (binary) relation

A transform for logic and set-theory

An old idea

$$PF(\text{sets, predicates}) \;=\; \text{binary relations}$$

Calculus of binary relations

- 1860 - introduced by De Morgan, embryonic
- 1941 - Tarski's school, cf. *A Formalization of Set Theory without Variables*
- 1980's - coreflexive models of sets (Freyd and Scedrov, Eindhoven school)

Unifying approach

Everything is a (binary) relation

Binary Relations

Arrow notation

Arrow $A \xrightarrow{R} B$ denotes a binary relation to B (target) from A (source).

Identity of composition

id such that $R \cdot id = id \cdot R = R$

Converse

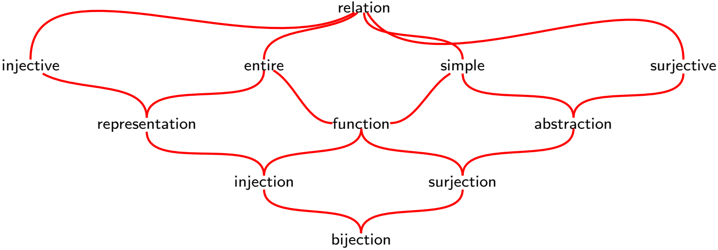
Converse of R — R° such that $a(R^{\circ})b$ iff $b\ R\ a$.

Ordering

"$R \subseteq S$ — the "R is at most S" — the obvious $R \subseteq S$ **ordering**.

Binary relation taxonomy

The whole picture:



where

	Reflexive ($\supseteq id$)	*Coreflexive* ($\subseteq id$)
ker R	**entire** R	**injective** R
img R	**surjective** R	**simple** R

$$\ker R = R^\circ \cdot R$$
$$\operatorname{img} R = R \cdot R^\circ$$

Second lecture

Schedule: Tuesday July 3rd, 11h30am-12h30m

Learning outcomes:

- PF-transform essentials
- PF-transform at work: describing data models and data impedance mismatch

Functions in one slide

- A function f is a relation such that $b\ f\ a \equiv b = f\ a$ and

Pointwise	Pointfree	
"Left" Uniqueness		
$b\ f\ a \wedge b'\ f\ a \quad \Rightarrow \quad b = b'$	$\mathrm{img}\ f \quad \subseteq \quad id$	(f is simple)
Leibniz principle		
$a = a' \quad \Rightarrow \quad f\ a = f\ a'$	$id \quad \subseteq \quad \ker f$	(f is entire)

- Back to useful "al-djabr" rules:

$$f \cdot R \subseteq S \equiv R \subseteq f^{\circ} \cdot S$$

$$R \cdot f^{\circ} \subseteq S \equiv R \subseteq S \cdot f$$

- Equality:

$$f \subseteq g \equiv f = g \equiv f \supseteq g$$

Simple relations

Simple relations are **everywhere** in computing:

- As computations: **partial functions** are simple relations
- As data: (finite) simple relations model **functional dependencies**, object identity, etc
- We will draw harpoon arrows $B \overset{R}{\leftharpoondown} A$ or $A \overset{R}{\rightharpoondown} B$ to indicate that R is simple.

We shall be using (simple) relations to model **both** algorithms and data.

Simple relations in one slide

"Al-djabr" rules for simple M:

$$M \cdot R \subseteq T \equiv (\delta M) \cdot R \subseteq M^\circ \cdot T \tag{8}$$

$$R \cdot M^\circ \subseteq T \equiv R \cdot \delta M \subseteq T \cdot M \tag{9}$$

where

$$\delta R = \ker R \cap id$$

the **domain** of R is the coreflexive part of $\ker R$.

Dually, we define the **range** of R as

$$\rho R = \operatorname{img} R \cap id$$

Predicates PF-transformed

- **Binary** predicates :

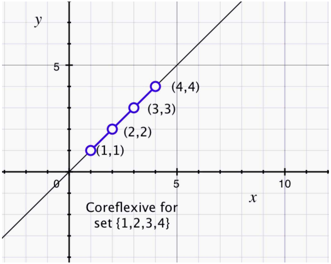
$$R = [\![b]\!] \;\equiv\; (y \; R \; x \equiv b(y,x))$$

- **Unary** predicates become fragments of *id* (coreflexives) :

$$R = [\![p]\!] \;\equiv\; (y \; R \; x \equiv (p \; x) \wedge x = y)$$

eg. (in the natural numbers)

$$[\![1 \leq x \leq 4]\!] \quad =$$



Boolean algebra of coreflexives

$$
\begin{aligned}
[\![p \wedge q]\!] &= [\![p]\!] \cdot [\![q]\!] && (10)\\
[\![p \vee q]\!] &= [\![p]\!] \cup [\![q]\!] && (11)\\
[\![\neg p]\!] &= id - [\![p]\!] && (12)\\
[\![false]\!] &= \bot && (13)\\
[\![true]\!] &= id && (14)
\end{aligned}
$$

Note the very useful fact that **conjunction** of coreflexives is **composition**

Simple relation expressive power

- **Comprehension** notation borrowed from VDM to denote a (finite) simple relation S at pointwise level:

$$\{a \mapsto S\ a \mid a \in dom\ S\}$$

where $dom\ S$ is the set-theoretic version of $\delta\, S$.

- Useful PF **patterns**:

  - **projection** — $f \cdot S \cdot g^{\circ}$ (g injective):

$$\{g\ a \mapsto f(S\ a) \mid a \in dom\ S\}$$

  - **selection** — $\Psi \cdot S \cdot \Phi$ (Ψ, Φ coreflexives):

$$\{a \mapsto S\ a \mid a \in dom\ S\ \wedge\ \phi\ a\ \wedge\ \psi(S\ a)\}$$

Simple relation expressive power

- **Comprehension** notation borrowed from VDM to denote a (finite) simple relation S at pointwise level:

$$\{a \mapsto S\ a \mid a \in dom\ S\}$$

where $dom\ S$ is the set-theoretic version of $\delta\, S$.

- Useful PF **patterns**:

  - **projection** — $f \cdot S \cdot g^{\circ}$ (g injective):

$$\{g\ a \mapsto f(S\ a) \mid a \in dom\ S\}$$

  - **selection** — $\Psi \cdot S \cdot \Phi$ (Ψ, Φ coreflexives):

$$\{a \mapsto S\ a \mid a \in dom\ S\ \wedge\ \phi\ a\ \wedge\ \psi(S\ a)\}$$

All (data structures) **in one** (PF notation)

Products

Database records — eg. $\boxed{5 \mid \textit{Peter} \mid 1991}$ — C/C++ `structs` etc are products:

$$\begin{tikzcd} A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\ & C \arrow[ul, "R"] \arrow[u, "\langle R,S\rangle"] \arrow[ur, "S"'] & \end{tikzcd} \tag{15}$$

where

$$\begin{array}{c|c} \psi & PF\ \psi \\ \hline\hline a\ R\ c \wedge b\ S\ c & (a,b)\langle R,S\rangle c \\ b\ R\ a \wedge d\ S\ c & (b,d)(R \times S)(a,c) \end{array} \tag{16}$$

Clearly: $R \times S = \langle R \cdot \pi_1, S \cdot \pi_2 \rangle$

All (data structures) **in one** (PF notation)

Products

Database records — eg.

5	*Peter*	1991

— C/C++ `structs` etc are products:

$$
\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
 & C \arrow[ul, "R"] \arrow[u, "\langle R,S \rangle"] \arrow[ur, "S"'] &
\end{tikzcd}
\tag{15}
$$

where

$$
\begin{array}{c|c}
\psi & PF\ \psi \\
\hline\hline
a\ R\ c \wedge b\ S\ c & (a, b)\langle R, S\rangle c \\
b\ R\ a \wedge d\ S\ c & (b, d)(R \times S)(a, c)
\end{array}
\tag{16}
$$

Clearly: $R \times S = \langle R \cdot \pi_1, S \cdot \pi_2 \rangle$

Sums

Example (Haskell):

```
data X = Boo Bool | Err String
```

PF-transforms to

$$\begin{tikzcd} Bool \arrow[r, "i_1"] \arrow[dr, "Boo"'] & Bool + String \arrow[d, "{[Boo\ ,Err]}"] & String \arrow[l, "i_2"'] \arrow[dl, "Err"] \\ & X & \end{tikzcd} \qquad (17)$$

where

$$[R\ ,S] \ = \ (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \quad \text{cf.} \quad \begin{tikzcd} A \arrow[r, "i_1"] \arrow[dr, "R"'] & A + B \arrow[d, "{[R\ ,S]}"] & B \arrow[l, "i_2"'] \arrow[dl, "S"] \\ & C & \end{tikzcd}$$

Dually: $R + S = [i_1 \cdot R\ , i_2 \cdot S]$

Sums

Example (Haskell):

```
data X = Boo Bool | Err String
```

PF-transforms to

$$
\begin{tikzcd}
Bool \arrow[r, "i_1"] \arrow[dr, "Boo"'] & Bool + String \arrow[d, "{[Boo\ , Err]}"] & String \arrow[l, "i_2"'] \arrow[dl, "Err"] \\
& X &
\end{tikzcd}
\tag{17}
$$

where

$$[R\ , S] \ = \ (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \quad \text{cf.} \quad
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "R"'] & A + B \arrow[d, "{[R\ , S]}"] & B \arrow[l, "i_2"'] \arrow[dl, "S"] \\
& C &
\end{tikzcd}$$

Dually: $R + S = [i_1 \cdot R\ , i_2 \cdot S]$

Polynomial types and grammars

- With sums and products one can build **polynomials**, "pointers" included:

$$Maybe\ A \stackrel{\mathrm{def}}{=} A + 1 \tag{18}$$

(where 1 is the singleton type inhabited by $\textsc{Nil}$):

- Grammars:

$$\begin{array}{ccc} \hline \textsc{Bnf notation} & & \textsc{Polynomial notation} \\ \hline \alpha \mid \beta & \mapsto & \alpha + \beta \\ \alpha\beta & \mapsto & \alpha \times \beta \\ \epsilon & \mapsto & 1 \\ a & \mapsto & 1 \\ \hline \end{array} \tag{19}$$

Grammars and inductive data models

For instance,

$$X \rightarrow \epsilon \mid a\ A\ X$$

(where X, A are non-terminals and a is terminal) leads **equation**

$$X = 1 + A \times X \tag{20}$$

cf.

```
typedef struct x {
  A data;
  struct x *next;
} Node;

typedef Node *X;
```

since $1 + A \times X$ is an instance of the "pointer to struct" pattern.

PF-transformed $PTree$

```
data PTree = Node { name :: [ Char ], birth ::
Int , mother :: Maybe PTree, father :: Maybe
PTree }
```

becomes

$$PTree \cong Ind \times (PTree + 1) \times (PTree + 1) \tag{21}$$

where $Ind = Name \times Birth$ packages the information relative to the name and birth year, ie.

$$PTree \cong \mathsf{G}(Ind, PTree) \tag{22}$$

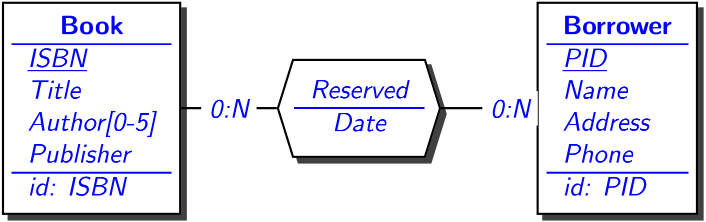
where G captures the particular pattern of recursion chosen to model family trees

$$\mathsf{G}(X, Y) \stackrel{\mathrm{def}}{=} X \times (Y + 1) \times (Y + 1)$$

(X refers to the parametric information and Y to the inductive part.)

Entity-Relationship diagrams

PF-transform of



is

$$
\begin{aligned}
Db &\stackrel{\mathrm{def}}{=} Books \times Borrowers \times Reserved \\
Books &\stackrel{\mathrm{def}}{=} ISBN \rightharpoonup Title \times (5 \rightharpoonup Author) \times Publisher \\
Borrowers &\stackrel{\mathrm{def}}{=} PID \rightharpoonup Name \times Address \times Phone \\
Reserved &\stackrel{\mathrm{def}}{=} ISBN \times PID \rightharpoonup Date
\end{aligned}
$$

Business rules

Example

"(...) Only existing books can be borrowed by known borrowers"

Pointwise

$$\phi(M, N, R) \stackrel{\mathrm{def}}{=}$$
$$\langle \forall\ i, p, d\ ::\ d\ R\ (i, p) \Rightarrow \langle \exists\ x\ ::\ x\ M\ i \rangle \wedge \langle \exists\ y\ ::\ y\ M\ p \rangle \rangle$$

where i, p, d range over $ISBN$, PID and $Date$, respectively,

PF-transform

We first order relations by how defined they are,

$$R \preceq S \ \equiv\ \delta\, R \subseteq \delta\, S$$

Then...

Business rules

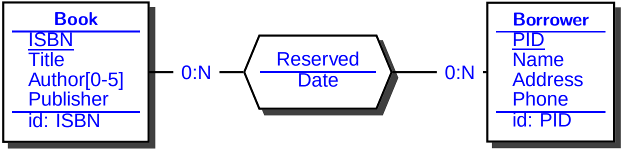
Rule

$$\phi(M, N, R) \stackrel{\mathrm{def}}{=} R \preceq M \cdot \pi_1 \ \wedge \ R \preceq N \cdot \pi_2$$

cf. diagram

$$\begin{tikzcd}
ISBN \arrow[d, "M"'] & ISBN \times PID \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "R"'] & PID \arrow[d, "N"] \\
Title \times (5 \rightharpoonup Author) \times Publisher & Date & Name \times Address \times Phone
\end{tikzcd}$$

whose geometrical similarity with the original is striking, recall:



Data impedance mismatch expressed in the PF-style

$$\begin{tikzcd} A \arrow[rr, bend left, "R"] \arrow[rr, phantom, "\leq"] & & B \arrow[ll, bend left, "F"] \end{tikzcd}$$ where

- $\mathsf{ker}\ R = id$ (representation) and $\mathsf{img}\ F = id$ (abstraction)
- connection between (R, F)

$$\langle \forall\ a, b\ ::\ b\ R\ a\ \Rightarrow\ a\ F\ b \rangle$$

shrinks to

$$R^\circ \subseteq F \tag{23}$$

($=R^\circ$ is the *least* retrieve relation associated with R) equivalent to

$$R \subseteq F^\circ \tag{24}$$

($=F^\circ$ largest representation one can connect to retrieve relation F).

$\leq$ is a preorder

- $\leq$ **is reflexive**: Between a datatype and itself

$$\begin{tikzcd} A \arrow[r, bend left, "id"] \arrow[r, phantom, "\leq"] & A \arrow[l, bend left, "id"] \end{tikzcd}$$

  there is *no impedance* at all

- $\leq$ **is transitive**:

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "F"] \end{tikzcd} \wedge \begin{tikzcd} B \arrow[r, bend left, "S"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "G"] \end{tikzcd} \Rightarrow \begin{tikzcd} A \arrow[r, bend left, "S \cdot R"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "F \cdot G"] \end{tikzcd}$$

  that is, data impedances compose.

One slide long calculations

$(F \cdot G, S \cdot R)$ are connected:

$$
\begin{array}{ll}
 & S \cdot R \subseteq (F \cdot G)^\circ \\
\equiv & \quad \{ \text{ converses: } (R \cdot S)^\circ = S^\circ \cdot R^\circ \ \} \\
 & S \cdot R \subseteq G^\circ \cdot F^\circ \\
\Leftarrow & \quad \{ \text{ monotonicity } \} \\
 & S \subseteq G^\circ \wedge R \subseteq F^\circ \\
\equiv & \quad \{ \text{ since } S, G \text{ and } R, F \text{ are assumed connected } \} \\
 & \textsc{True}
\end{array}
$$

Right-invertibility

That $\leq$-rules entail *right-invertibility*

$$F \cdot R \;=\; id \tag{25}$$

is again a one slide long calculation:

$$
\begin{array}{ll}
& F \cdot R = id \\
\equiv & \qquad \{ \text{ equality of relations } \} \\
& F \cdot R \subseteq id \;\wedge\; id \subseteq F \cdot R \\
\equiv & \qquad \{ \text{ img } F = id \text{ and ker } R = id \; \} \\
& F \cdot R \subseteq F \cdot F^{\circ} \;\wedge\; R^{\circ} \cdot R \subseteq F \cdot R \\
\equiv & \qquad \{ \text{ converses } \} \\
& F \cdot R \subseteq F \cdot F^{\circ} \;\wedge\; R^{\circ} \cdot R \subseteq R^{\circ} \cdot F^{\circ} \\
\Leftarrow & \qquad \{ \; (F\cdot) \text{ and } (R^{\circ}\cdot) \text{ are monotone (cf. GCs) } \} \\
& R \subseteq F^{\circ} \;\wedge\; R \subseteq F^{\circ}
\end{array}
$$

Functions only

Right-invertibility happens to be *equivalent* to connectivity wherever both abstraction and representation are **functions**, say f, r:

$$
\begin{tikzcd}
A \arrow[rr, bend left, "r"] \arrow[rr, phantom, "\leq"] & & C \arrow[ll, bend left, "f"]
\end{tikzcd}
\qquad \equiv \qquad f \cdot r = id \tag{26}
$$

That $f \cdot r = id$ equivales $r \subseteq f^{\circ}$ and entails f surjective and r injective is again a short calculation:

$$
\begin{array}{ll}
 & f \cdot r = id \\
\equiv & \quad \{ \text{ equality of functions } \} \\
 & f \cdot r \subseteq id \\
\equiv & \quad \{ \text{ ``al-djabr'' (shunting) } \} \\
 & r \subseteq f^{\circ}
\end{array}
$$

Functions only

$$
\begin{array}{ll}
 & r \subseteq f^{\circ} \\
\Rightarrow & \quad \{ \text{ composition is monotonic } \} \\
 & f \cdot r \subseteq f \cdot f^{\circ} \quad \wedge \quad r^{\circ} \cdot r \subseteq r^{\circ} \cdot f^{\circ} \\
\equiv & \quad \{ \ f \cdot r = id \ ; \text{ converses } \} \\
 & id \subseteq f \cdot f^{\circ} \quad \wedge \quad r^{\circ} \cdot r \subseteq id \\
\equiv & \quad \{ \text{ definitions } \} \\
 & f \text{ surjective} \wedge r \text{ injective}
\end{array}
$$

Equivalence: $\Rightarrow$ (above) + $\Leftarrow$ (which of holds in general)

Well-known surjections and injections

From cancellation-laws

$$\pi_1 \cdot \langle f, g \rangle = f \ , \ \pi_2 \cdot \langle f, g \rangle = g$$
$$[g \ , f] \cdot i_1 = g \ , \ [g \ , f] \cdot i_2 = f$$

we get some basic impedance mismatches captured by $\leq$-rules:

$$\begin{tikzcd} A \arrow[rr, bend left, "\langle id, g \rangle"] \arrow[rr, phantom, "\leq"] && A \times B \arrow[ll, bend left, "\pi_1"] \end{tikzcd} \qquad \begin{tikzcd} B \arrow[rr, bend left, "\langle f, id \rangle"] \arrow[rr, phantom, "\leq"] && A \times B \arrow[ll, bend left, "\pi_2"] \end{tikzcd}$$

$$\begin{tikzcd} A \arrow[rr, bend left, "i_1"] \arrow[rr, phantom, "\leq"] && A + B \arrow[ll, bend left, "{[id \ , f]}"] \end{tikzcd} \qquad \begin{tikzcd} B \arrow[rr, bend left, "i_2"] \arrow[rr, phantom, "\leq"] && A + B \arrow[ll, bend left, "{[g \ , id]}"] \end{tikzcd}$$

Pointers and references

Pointers

$$
\begin{tikzcd}
A \arrow[rr, bend left, "i_1"] \arrow[rr, phantom, "\leq"] & & A + 1 \arrow[ll, bend left, "{[id\ ,F]}"]
\end{tikzcd}
$$

References (“**references** cheaper to move around than **referents**”)

$$
\begin{tikzcd}
\mathsf{G}\,A \arrow[rr, bend left, "R"] \arrow[rr, phantom, "\leq"] & & (\boldsymbol{N} \rightharpoonup A) \times \mathsf{G}\,\boldsymbol{N} \arrow[ll, bend left, "Dref"]
\end{tikzcd}
\tag{27}
$$

cf. containers, shapes etc — details to be given later on.

Isomorphic data types

A quite special case of (r, f) pair is one such that both

$$
\begin{tikzcd}
A \arrow[rr, bend left, "r"] \arrow[rr, phantom, "\leq"] & & C \arrow[ll, bend left, "f"]
\end{tikzcd}
\quad \wedge \quad
\begin{tikzcd}
A \arrow[rr, bend left, "f"] \arrow[rr, phantom, "\leq"] & & C \arrow[ll, bend left, "r"]
\end{tikzcd}
\tag{28}
$$

hold. This equivales

$$
\begin{array}{cl}
& r \subseteq f^\circ \wedge f \subseteq r^\circ \\
\equiv & \qquad \{ \text{ converses ; equality of relations } \} \\
& r^\circ = f
\end{array}
\tag{29}
$$

So r (a function) is the converse of another function f. This means that both are bijections (isomorphisms) since

$$
f \text{ is a bijection} \equiv f^\circ \text{ is a function} \tag{30}
$$

Isomorphic data types

In a diagram:

$$
\begin{tikzcd}
A \arrow[rr, bend left, "r=f^{\circ}"] \arrow[rr, phantom, "\cong"] & & C \arrow[ll, bend left, "f=r^{\circ}"]
\end{tikzcd}
\tag{31}
$$

Isomorphism $A \cong C$ corresponds to *minimal* impedance mismatch between types A and C — although the **format** of data changes, data conversion **in both ways** is wholly **recoverable**.

Example: function *swap* $\stackrel{\mathrm{def}}{=} \langle \pi_2, \pi_1 \rangle$ witnesses

$$
\begin{tikzcd}
A \times B \arrow[rr, bend left, "swap"] \arrow[rr, phantom, "\cong"] & & B \times A \arrow[ll, bend left, "swap"]
\end{tikzcd}
\tag{32}
$$

(eg. change order of entries in `struct`s; swap order of columns in a spreadsheet, etc.)

When the converse of a function is a function

$$
\begin{array}{ll}
 & swap^{\circ} \\
= & \quad \{\ \langle R, S\rangle \ = \ \pi_1^{\circ} \cdot R \cap \pi_2^{\circ} \cdot S\ \} \\
 & (\pi_1^{\circ} \cdot \pi_2 \cap \pi_2^{\circ} \cdot \pi_1)^{\circ} \\
= & \quad \{\ \text{converses}\ \} \\
 & \pi_2^{\circ} \cdot \pi_1 \cap \pi_1^{\circ} \cdot \pi_2 \\
= & \quad \{\ \text{back to splits}\ \} \\
 & swap
\end{array}
$$

So *swap* is its own inverse and therefore a bijection.

Exercise 12, page 169

The calculation just above was too simple. To recognize the power of rule *"when the converse of a function is a function"* prove the associative property of sum,

$$\begin{tikzcd} A + (B + C) \arrow[rr, bend left, "r"] \arrow[rr, phantom, "\cong"] & & (A + B) + C \arrow[ll, bend left, "f = [id + i_1\ ,\, i_2 \cdot i_2]"] \end{tikzcd} \tag{33}$$

by calculating *the function* r which is the converse of f.

$$
\begin{array}{ll}
 & [id + i_1\ ,\, i_2 \cdot i_2]^\circ \\
= & \quad \{\ \text{expand } [R\ ,\, S]\ \} \\
 & ((id + i_1) \cdot i_1^\circ \cup i_2 \cdot i_2 \cdot i_2^\circ)^\circ \\
= & \quad \{\ \text{converses}\ \} \\
 & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ
\end{array}
$$

Exercise 12, page 169

The calculation just above was too simple. To recognize the power of rule *"when the converse of a function is a function"* prove the associative property of sum,

$$\begin{tikzcd} A+(B+C) \arrow[rr, bend left, "r"] \arrow[rr, phantom, "\cong"] & & (A+B)+C \arrow[ll, bend left, "f=[id+i_1\ ,i_2\cdot i_2]"] \end{tikzcd} \tag{33}$$

by calculating *the function* r which is the converse of f.

$$\begin{array}{cl} & [id + i_1\ , i_2 \cdot i_2]^\circ \\ = & \qquad \{\ \text{expand } [R\ , S]\ \} \\ & ((id + i_1) \cdot i_1^\circ \cup i_2 \cdot i_2 \cdot i_2^\circ)^\circ \\ = & \qquad \{\ \text{converses}\ \} \\ & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \end{array}$$

Exercise 12, page 169

The calculation just above was too simple. To recognize the power of rule *"when the converse of a function is a function"* prove the associative property of sum,

$$
\begin{tikzcd}
A + (B + C) \arrow[rr, bend left, "r"] \arrow[rr, phantom, "\cong"] & & (A + B) + C \arrow[ll, bend left, "f = [id + i_1\ , i_2 \cdot i_2]"]
\end{tikzcd}
\tag{33}
$$

by calculating *the function* r which is the converse of f.

$$
\begin{array}{ll}
& [id + i_1\ , i_2 \cdot i_2]^\circ \\
= & \{ \text{ expand } [R\ , S] \ \} \\
& ((id + i_1) \cdot i_1^\circ \cup i_2 \cdot i_2 \cdot i_2^\circ)^\circ \\
= & \{ \text{ converses } \} \\
& i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ
\end{array}
$$

Exercise 12, page 169

$$i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)}$$

$= \quad \{$ expand $R + S$ $\}$

$$i_1 \cdot [i_1 \ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ$$

$= \quad \{$ expand $[R \ , S]$ $\}$

$$i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ$$

$= \quad \{$ distribution ; associativity $\}$

$$i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ$$

$= \quad \{$ wrap up (function!) $\}$

$$[i_1 \cdot i_1 \ , [i_1 \cdot i_2 \ , i_2]]$$

$= \quad \{$ spruce it $\}$

$$[i_1 \cdot i_1 \ , i_2 + id]$$

Exercise 12, page 169

$$
\begin{array}{ll}
 & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)} \\
= & \qquad \{ \ \text{expand } R + S \ \} \\
 & i_1 \cdot [i_1 \ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \qquad \{ \ \text{expand } [R \ , S] \ \} \\
 & i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \qquad \{ \ \text{distribution ; associativity} \ \} \\
 & i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \\
= & \qquad \{ \ \text{wrap up (function!)} \ \} \\
 & [i_1 \cdot i_1 \ , [i_1 \cdot i_2 \ , i_2]] \\
= & \qquad \{ \ \text{spruce it} \ \} \\
 & [i_1 \cdot i_1 \ , i_2 + id]
\end{array}
$$

Exercise 12, page 169

$$
\begin{array}{ll}
 & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)} \\
= & \quad \{ \text{ expand } R + S \ \} \\
 & i_1 \cdot [i_1 \ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ expand } [R \ , S] \ \} \\
 & i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ distribution ; associativity } \} \\
 & i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \\
= & \quad \{ \text{ wrap up (function!) } \} \\
 & [i_1 \cdot i_1 \ , [i_1 \cdot i_2 \ , i_2]] \\
= & \quad \{ \text{ spruce it } \} \\
 & [i_1 \cdot i_1 \ , i_2 + id]
\end{array}
$$

Exercise 12, page 169

$$
\begin{array}{ll}
 & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)} \\
= & \qquad \{ \text{ expand } R + S \ \} \\
 & i_1 \cdot [i_1 \ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \qquad \{ \text{ expand } [R \ , S] \ \} \\
 & i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \qquad \{ \text{ distribution ; associativity } \} \\
 & i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \\
= & \qquad \{ \text{ wrap up (function!) } \} \\
 & [i_1 \cdot i_1 \ , [i_1 \cdot i_2 \ , i_2]] \\
= & \qquad \{ \text{ spruce it } \} \\
 & [i_1 \cdot i_1 \ , i_2 + id]
\end{array}
$$

Exercise 12, page 169

$$
\begin{array}{ll}
& i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)} \\
= & \quad \{ \text{ expand } R + S \ \} \\
& i_1 \cdot [i_1\ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ expand } [R\ , S] \ \} \\
& i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ distribution ; associativity } \} \\
& i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \\
= & \quad \{ \text{ wrap up (function!) } \} \\
& [i_1 \cdot i_1\ , [i_1 \cdot i_2\ , i_2]] \\
= & \quad \{ \text{ spruce it } \} \\
& [i_1 \cdot i_1\ , i_2 + id]
\end{array}
$$

Exercise 12, page 169

$$
\begin{array}{ll}
& i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \qquad \text{(from last slide)} \\
= & \quad \{ \text{ expand } R + S \ \} \\
& i_1 \cdot [i_1 \ , i_2 \cdot i_1^\circ] \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ expand } [R \ , S] \ \} \\
& i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \\
= & \quad \{ \text{ distribution ; associativity } \} \\
& i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \\
= & \quad \{ \text{ wrap up (function!) } \} \\
& [i_1 \cdot i_1 \ , [i_1 \cdot i_2 \ , i_2]] \\
= & \quad \{ \text{ spruce it } \} \\
& [i_1 \cdot i_1 \ , i_2 + id]
\end{array}
$$

Exercise 13, page 170

The following are known isomorphisms involving sums and products:

$$
\begin{aligned}
A \times (B \times C) &\cong (A \times B) \times C && (34)\\
A &\cong A \times 1 && (35)\\
A &\cong 1 \times A && (36)\\
A + B &\cong B + A && (37)\\
C \times (A + B) &\cong C \times A + C \times B && (38)
\end{aligned}
$$

Guess the relevant isomorphism pairs.

More elaborate isomorphisms

Let us introduce variables in isomorphism pair (r, f):

$$
\begin{array}{ll}
 & r^{\circ} = f \\
\equiv & \qquad \{ \text{ introduce variables } \} \\
 & \langle \forall\ a, c \ :: \ c\ (r^{\circ})\ a \equiv c\ f\ \ a \rangle \\
\equiv & \qquad \{ \ b(f^{\circ} \cdot R \cdot g)a \equiv (f\ \ b)R(g\ \ a) \ \} \\
 & \langle \forall\ a, c \ :: \ r\ c = a \ \equiv \ c = f\ \ a \rangle
\end{array}
$$

You've seen this pattern already at school, recall eg.

$$
\langle \forall\ a, c \ :: \ b + c = a \ \equiv \ c = a - b \rangle \tag{39}
$$

Let us see a few data transformations which share this pattern.

Transposes

Every relation can be safely converted into a *the corresponding* set-valued function:

$$\langle \forall\ R, k\ ::\ k = \Lambda R\ \equiv\ R =\ \in \cdot\, k \rangle \tag{40}$$

With more variables (omitting outer $\forall$):

$$k = \Lambda R\ \equiv\ \langle \forall\ b, a\ ::\ b\ R\ a \equiv b \in (k\ a) \rangle$$

Diagram:

$$\begin{tikzcd}
(\mathcal{P}B)^A \arrow[rr, bend left, "(\in\cdot)"] \arrow[rr, phantom, "\cong"] & & A \to B \arrow[ll, bend left, "\Lambda"]
\end{tikzcd}$$

Transposes

Simple relations "are *Maybe* functions":

$$k = tot\ S \ \equiv\ S = i_1^\circ \cdot k$$

With more variables (omitting outer $\forall$):

$$k = tot\ S \ \equiv\ \langle \forall\ b, a\ ::\ b\ S\ a \equiv (i_1 b) = k\ a \rangle$$

Diagram:

$$\begin{tikzcd} (B+1)^A \arrow[rr, bend left, "untot=(i_1^\circ \cdot)"] \arrow[rr, phantom, "\cong"] & & A \rightharpoonup B \arrow[ll, bend left, "tot"] \end{tikzcd}$$

(Handles impedance mismatch between **pointer** data models and **relational** models.)

Relational currying

Isomorphism

$$
\begin{tikzcd}
(C \to A)^B \arrow[r, bend left, "\overline{(\_)}^{\circ}"] \arrow[r, phantom, "\cong"] & B \times C \to A \arrow[l, bend left, "\overline{(\_)}"]
\end{tikzcd}
\tag{41}
$$

and associated universal property,

$$
k = \overline{R} \ \equiv \ \langle \forall\ a, b, c\ ::\ a\ (k\ b)\ c \equiv a\ R\ (b, c) \rangle \tag{42}
$$

express a kind of *selection/projection* mechanism: given some b_0, $\overline{R}\ b_0$ selects the "sub-relation" of R of all pairs (a, c) related to b_0.

Functional currying

Isomorphism (in case $R := f$)

$$\begin{tikzcd} (A^C)^B \arrow[rr, bend left, "\overline{(\_)}^\circ"] \arrow[rr, phantom, "\cong"] & & A^{B \times C} \arrow[ll, bend left, "\overline{(\_)}"] \end{tikzcd} \qquad (43)$$

Associated universal property

$$k = \overline{f} \;\equiv\; \langle \forall\ b, c\ ::\ (k\ b)\ c = f\ (b, c) \rangle \qquad (44)$$

simpler because f is a function. (The usual notation for $\overline{f}$ is $curry\ f$, and $uncurry = curry^\circ$.)

Third lecture

Schedule: Thursday July 5th, 11h30am-12h30m

Learning outcomes:

- PF-transform at work:
  - new $\leq$-rules from old
  - calculating **implementations** from abstract models
  - dealing with **recursive** data model impedance mismatch
- PF-transform in the lab: the **2LT** package
- Topics for research

Calculating database schemes from abstract models

- *Generic type* of a relational database

$$RDBT \stackrel{\text{def}}{=} \prod_{i=1}^{n} \left(\prod_{j=1}^{n_i} K_j \rightharpoonup \prod_{k=1}^{m_i} D_k \right) \tag{45}$$

  only admits products and simple relations
- $db \in RDBT$ is a collection of n relational **tables** (index $i = 1, n$) each of which maps tuples of **keys** (index j) to **tuples** of *data of interest* (index k).
- What about datatype **sums**, **multivalued** types, **inductive** types etc?

Some impedance **mismatch** to be expected!

Calculating database schemes from abstract models

- *Generic type* of a relational database

$$RDBT \stackrel{\mathrm{def}}{=} \prod_{i=1}^{n} \left(\prod_{j=1}^{n_i} K_j \rightharpoonup \prod_{k=1}^{m_i} D_k \right) \tag{45}$$

  only admits products and simple relations

- $db \in RDBT$ is a collection of n relational **tables** (index $i = 1, n$) each of which maps tuples of **keys** (index j) to **tuples** of *data of interest* (index k).
- What about datatype **sums**, **multivalued** types, **inductive** types etc?

Some impedance **mismatch** to be expected!

Getting rid of sums

Diagram:

$$
\begin{tikzcd}
(B + C) \to A \arrow[rr, bend left, "{[\_ , \_]^{\circ}}"] \arrow[rr, phantom, "\cong"] & & (B \to A) \times (C \to A) \arrow[ll, bend left, "{[\_ , \_]}"]
\end{tikzcd}
\tag{46}
$$

Universal property:

$$T = [R\ , S] \equiv T \cdot i_1 = R \ \wedge \ T \cdot i_2 = S \tag{47}$$

Pragmatics: when applied from left to right, this rule helps in removing sums from data models: relations with input sums decompose into pairs of relations

Getting rid of sums

What about sums at the output? Another sum-elimination rule is applicable to such situations,

$$
\begin{tikzcd}
A \to (B + C) \arrow[rr, bend left, "\triangle_+"] \arrow[rr, phantom, "\cong"] & & (A \to B) \times (A \to C) \arrow[ll, bend left, "\overset{+}{\bowtie}"]
\end{tikzcd}
\tag{48}
$$

where

$$
M \overset{+}{\bowtie} N \stackrel{\mathrm{def}}{=} i_1 \cdot M \cup i_2 \cdot N \tag{49}
$$

$$
\triangle_+ M \stackrel{\mathrm{def}}{=} \langle i_1^\circ \cdot M, i_2^\circ \cdot M \rangle \tag{50}
$$

Getting rid of multivalued attributes

Recall that

$$Books \stackrel{\mathrm{def}}{=} ISBN \rightharpoonup Title \times (5 \rightharpoonup Author) \times Publisher$$

has a multivalued type (up to 5 authors). How do we remove $(\rightharpoonup)$-*nesting*?

In the next slide we calculate a rule which gets rid of pattern

$$A \rightharpoonup (D \times (B \rightharpoonup C))$$

New $\leq$-rules from old

$$
\begin{array}{ll}
 & A \rightharpoonup (D \times (B \rightharpoonup C)) \\
\cong & \qquad \{ \ \textit{Maybe}\ \text{transpose} \ \} \\
 & (D \times (B \rightharpoonup C) + 1)^A \\
\leq & \qquad \{ \ \text{exercise 10} \ \} \\
 & ((D + 1) \times (B \rightharpoonup C))^A \\
\cong & \qquad \{ \ \text{splitting: } (B \times C)^A \ \cong \ B^A \times C^A \ \} \\
 & (D + 1)^A \times (B \rightharpoonup C)^A \\
\cong & \qquad \{ \ \textit{Maybe}\ \text{transpose and relational (un)currying} \ \} \\
 & (A \rightharpoonup D) \times (A \times B \rightharpoonup C)
\end{array}
$$

Getting rid of multivalued attributes

In summary:

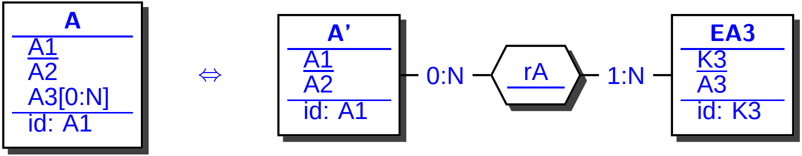
$$
\begin{tikzcd}
A \rightharpoonup (D \times (B \rightharpoonup C)) \arrow[r, bend left, "\triangle_n"] \arrow[r, phantom, "\leq"] & (A \rightharpoonup D) \times (A \times B \rightharpoonup C) \arrow[l, bend left, "\bowtie_n"]
\end{tikzcd}
\quad (51)
$$

Illustration:

$$
\begin{array}{cl}
 & Books \;=\; ISBN \rightharpoonup (Title \times (5 \rightharpoonup Author) \times Publisher) \\
\cong_1 & \qquad \{\; r_1 = id \rightharpoonup \langle\langle\pi_1, \pi_3\rangle, \pi_2\rangle \;,\; f_1 = id \rightharpoonup \langle\pi_1 \cdot \pi_1, \pi_2, \pi_2 \cdot \pi_1\rangle \;\} \\
 & ISBN \rightharpoonup (Title \times Publisher) \times (5 \rightharpoonup Author) \\
\leq_2 & \qquad \{\; r_2 = \triangle_n \;,\; f_2 = \bowtie_n, \text{cf. } (\mathbf{??}) \;\} \\
 & (ISBN \rightharpoonup Title \times Publisher) \times (ISBN \times 5 \rightharpoonup Author) \\
= & Books_2
\end{array}
$$

Checking model transformations

Entity-Relationship PF-semantics makes it possible to check **model transformation rules** available from the literature, eg. Rule 12.2 of J.-L. Hainaut (GTTSE'05) catalogue:



(This converts a $0 : N$ attribute into an entity.)

Starting point

$$\mathbf{A} = A_1 \rightharpoonup A_2 \times \mathcal{P} A_3$$

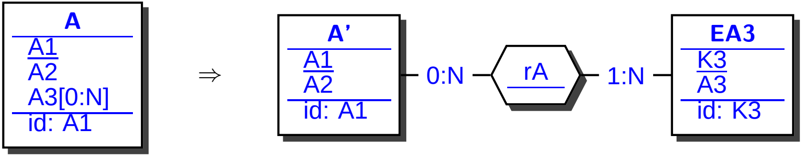
Checking model transformations

$$
\begin{array}{ll}
& A_1 \rightharpoonup A_2 \times \mathcal{P} A_3 \\
\leq_1 & \qquad \{ \text{ create references to } A_3 \ \} \\
& (K_3 \rightharpoonup A_3) \times (A_1 \rightharpoonup A_2 \times \mathcal{P} K_3) \\
\cong_2 & \qquad \{ \ \mathcal{P} A \ \cong \ A \rightharpoonup 1 \ \} \\
& (K_3 \rightharpoonup A_3) \times (A_1 \rightharpoonup A_2 \times (K_3 \rightharpoonup 1)) \\
\leq_3 & \qquad \{ \text{ unnest } (\rightharpoonup) \ \} \\
& (K_3 \rightharpoonup A_3) \times ((A_1 \rightharpoonup A_2) \times (A_1 \times K_3 \rightharpoonup 1)) \\
\cong_4 & \qquad \{ \text{ introduce ternary product } \} \\
& \underbrace{(A_1 \rightharpoonup A_2)}_{A'} \times \underbrace{(\mathcal{P}(A_1 \times K_3))}_{rA} \times \underbrace{(K_3 \rightharpoonup A_3)}_{EA3}
\end{array}
$$

Calculating model transformations

Our conclusion is that — strictly speaking — the law is uni-directional, not an equivalence:

$$A_1 \rightharpoonup A_2 \times \mathcal{P}A_3 \;\leq\; \underbrace{(A_1 \rightharpoonup A_2)}_{A'} \times \underbrace{(\mathcal{P}(A_1 \times K_3))}_{rA} \times \underbrace{(K_3 \rightharpoonup A_3)}_{EA3}$$



thanks to the **accuracy** of PF-reasoning.

On the impedance of recursive data models

Recall that our starting model for family trees is recursive:

```
data PTree = Node {
        name   :: String ,
        birth  :: Int     ,
        mother :: Maybe PTree,
        father :: Maybe PTree
        }
```

that is (for *Ind* abbreviating *name* and *birth*)

$$PTree \cong \underbrace{Ind \times (PTree + 1) \times (PTree + 1)}_{\mathsf{G}\ PTree}$$

In general

$$\mu\mathsf{G} \cong \mathsf{G}\,\mu\mathsf{G}$$

Getting rid of μ's

$$\begin{tikzcd} \mu\mathsf{G} \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & \underbrace{(K \rightharpoonup \mathsf{G}\,K)}_{\text{“heap”}} \times K \arrow[l, bend left, "Unf"] \end{tikzcd} \tag{52}$$

where K ia as a data type of *“heap addresses”* and $K \rightharpoonup \mathsf{G}\,K$ a datatype of G-structured *heaps*.

Representations are "folds"

A typical representation R is the function r which builds the heap for a tree by joining (separated) heaps for the subtrees, for instance

```
r (Node n b m f) = let x = fmap r m
                       y = fmap r f
                   in merge (n,b) x y
```

where `merge` performs **separated union** of heaps

```
merge a Nothing Nothing  =
      Heap ([ 1 |-> (a, Nothing, Nothing) ]) 1
merge a (Just x) (Just y) =
      Heap ([ 1 |-> (a, Just k1, Just k2) ] ++ h1 ++ h2) 1
            where (Heap h1 k1) = bmap id even_ x
                  (Heap h2 k2) = bmap id odd_ y
....
....
```

Data "heapification"

Source

```
t= Node {name = "Peter", birth = 1991,
         mother = Just (Node {name = "Mary", birth = 1956,
                                 mother = Nothing,
                                 father = Just (Node {name = "J
                                                       birth = 1
         ...... }}}
```

"heapifies" into:

```
r t = Heap [(1,(("Peter",1991),Just 2,Just 3)),
            (2,(("Mary",1956),Nothing,Just 6)),
            (6,(("Jules",1917),Nothing,Nothing)),
            (3,(("Joseph",1955),Just 5,Just 7)),
            (5,(("Margaret",1923),Nothing,Nothing)),
            (7,(("Luigi",1920),Nothing,Nothing))]
      1
```

Abstractions are “unfolds”

Abstraction is a (partial!) unfold:

```
f (Heap h k) = let Just (a,x,y) = lookup k h
               in  Node (fst a)(snd a)
                        (fmap (f . Heap h) x)
                        (fmap (f . Heap h) y)
```

(can be “totalized” via the `Maybe` transpose yielding a monadic unfold)

Thanks to the $\leq$-rule

$f(r\ t) = t$ always holds.

Abstractions are "unfolds"

Abstraction is a (partial!) unfold:

```
f (Heap h k) = let Just (a,x,y) = lookup k h
               in  Node (fst a)(snd a)
                        (fmap (f . Heap h) x)
                        (fmap (f . Heap h) y)
```

(can be "totalized" via the `Maybe` transpose yielding a monadic unfold)

Thanks to the $\leq$-rule

$f(r\ t) = t$ always holds.

Boiling recursion down to SQL

$$
\begin{array}{ll}
& PTree \\
\cong_1 & \quad \{\ r_1 = out\ ,\ f_1 = in,\ \text{for}\ \mathsf{G}\, K \stackrel{\mathrm{def}}{=} Ind \times (K+1) \times (K+1)\ \} \\
& \mu \mathsf{G} \\
\leq_2 & \quad \{\ R_2 = Unf^{\circ},\ F_2 = Unf\ \} \\
& (K \rightharpoonup Ind \times (K+1) \times (K+1)) \times K \\
\cong_3 & \quad \{\ r_3 = (id \rightharpoonup flatr^{\circ}) \times id\ ,\ f_3 = (id \rightharpoonup flatr) \times id\ \} \\
& (K \rightharpoonup Ind \times ((K+1) \times (K+1))) \times K \\
\cong_4 & \quad \{\ r_4 = (id \rightharpoonup id \times p2p) \times id\ ,\ f_4 = (id \rightharpoonup id \times p2p^{\circ}) \times id\ \} \\
& (K \rightharpoonup Ind \times (K+1)^2) \times K \\
\cong_5 & \quad \{\ r_5 = (id \rightharpoonup id \times tot^{\circ}) \times id\ ,\ f_5 = (id \rightharpoonup id \times tot) \times id\ \} \\
& (K \rightharpoonup Ind \times (2 \rightharpoonup K)) \times K \\
\leq_6 & \quad \{\ r_6 = \triangle_n\ ,\ f_6 = \bowtie_n\ \}
\end{array}
$$

Boiling recursion down to SQL

$$
\begin{array}{ll}
& ((K \rightharpoonup Ind) \times (K \times 2 \rightharpoonup K)) \times K \\
\cong_7 & \quad \{\ r_7 = \mathit{flatl}\ ,\ f_7 = \mathit{flatl}^\circ\ \} \\
& (K \rightharpoonup Ind) \times (K \times 2 \rightharpoonup K) \times K \\
=_8 & \quad \{\ \text{since } Ind = Name \times Birth\ \} \\
& (K \rightharpoonup Name \times Birth) \times (K \times 2 \rightharpoonup K) \times K
\end{array}
$$

In summary:

- Step 2 moves from the functional (inductive) to the pointer-based representation.
- Step 5 starts the move from **pointer**-based to **relational**-based representation: pointers "become" primary/foreign keys.
- Steps 7 and 8 deliver the final **RDBT** structure. (Third factor K gives access to the root of the original tree.)

Last but not least

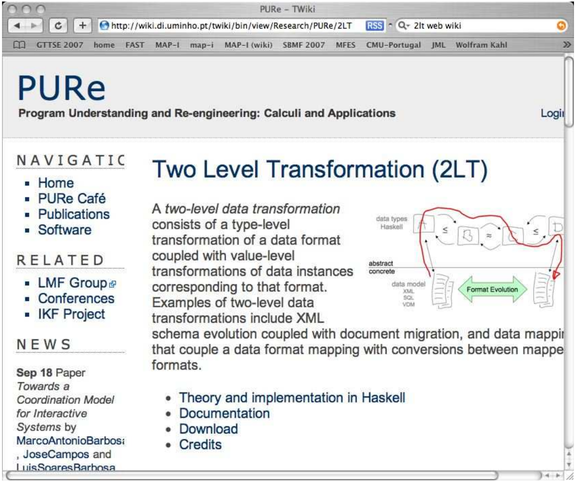
$\leq$-calculus is structural: given parametric type G,

$$
\begin{tikzcd}
A \arrow[rr, bend left, "R"] \arrow[rr, phantom, "\leq"] & & B \arrow[ll, bend left, "F"]
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
\mathsf{G}\,A \arrow[rr, bend left, "\mathsf{G}\,R"] \arrow[rr, phantom, "\leq"] & & \mathsf{G}\,B \arrow[ll, bend left, "\mathsf{G}\,F"]
\end{tikzcd}
$$

- Easy PF-proof (see notes)
- Also valid for *n* parameter types, eg.

$$A \leq C \wedge B \leq D \;\Rightarrow\; A + B \leq C + D$$

Tools: **2LT** (@ UMinho Haskell Libraries)



The 2LT engine is based on *strategic* term rewriting.

2LT demos: {XML,VDM} $\leftrightarrow$ SQL

Demo 1:

- Bridging XML and SQL:
  - **PTree** example replayed by 2LT

Demo 2:

- Generating SQL from VDM data models:
  - **Project:** development of a **repository** of courses on formal methods in Europe
  - **Client:** Formal Methods Europe (**FME**) association
  - **Method:** formal model in **VDM++** lead to a prototype webservice (using **CSK VDMTools**); database model automatically calculated by 2LT, including data migration.

Conclusions

Summary

- Data model impedance mismatch can be **calculated**
- PF-transform makes calculations agile and elegant
- $e = m + c$ approach to software engineering

Topics in the notes not covered in the lectures

- Operation transcription (more technical but great fun)
- Concrete invariant calculation

Still a lot of work to do: see next slides

Research topic: Lenses relate to $\leq$-rules

Not only connectivity of

$$\begin{tikzcd} & T \times S \arrow[dr, bend left, "putback"] & \\ T \arrow[ur, bend left, "\pi_1^\circ"] \arrow[rr, phantom, "\leq"] & & S \arrow[ll, bend left, "get"] \end{tikzcd}$$

cf.

$$
\begin{array}{ll}
& putback \cdot \pi_1^\circ \subseteq get^\circ \\
\equiv & \quad \{ \text{ “al-djabr” twice (functions) } \} \\
& get \cdot putback \subseteq \pi_1 \\
\equiv & \quad \{ \text{ equality of functions } \} \\
& get \cdot putback = \pi_1 \\
\equiv & \quad \{ \text{ add variables: } \textbf{acceptability} \ \} \\
& get(putback(v, s)) = v
\end{array}
$$

Lenses relate to $\leq$-rules

... but also connectivity of:

$$\begin{tikzcd} S \arrow[rr, bend left, "\langle get, id \rangle"] \arrow[rr, phantom, "\leq"] & & T \times S \arrow[ll, bend left, "putback"] \end{tikzcd}$$

cf.

$$\begin{array}{cl} & \langle get, id \rangle \subseteq put^{\circ} \\ \equiv & \qquad \{ \text{ “al-djabr” } \} \\ & putback \cdot \langle get, id \rangle \subseteq id \\ \equiv & \qquad \{ \text{ add variables: } \textbf{stability} \} \\ & putback(get\ s, s) = s \end{array}$$

View-update context

$$
\begin{tikzcd}
T \arrow[d, "u"'] \\
T
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\langle get, id \rangle"] \arrow[d, "\mathcal{L}u"'] & T \times S \arrow[d, "u \times id"] \\
S \arrow[dr, "get"'] & T \times S \arrow[l, "putback"'] \arrow[d, "\pi_1"] \\
& T
\end{tikzcd}
$$

$$\mathcal{L}u \stackrel{\text{def}}{=} putback \cdot (u \times id) \cdot \langle get, id \rangle$$

Note that **stability** is nothing but lens $\mathcal{L}$ preserving the identity update:

$$\mathcal{L}id = id$$

Seeking compositionality:

$$\mathcal{L}(u_1 \, \theta \, u_2) = (\mathcal{L}u_1) \, \phi \, (\mathcal{L}u_2) \;\Leftarrow\; \ldots\ldots$$

Other research topics and applications

Heapification: Law given can be generalized to **mutually** recursive datatypes

Separation logic: Law given has a clear connection to shared-mutable data representation and thus with **separation logic**.

Concrete invariants: $\leq$-rules should be able to take data type **invariants** into account

Mapping scenarios for the UML: A calculational theory of **UML** mapping scenarios could be developed from eg. Kevin Lano's catalogue

2LT: Tool can be of help in industrial applications (about 70% of **data-warehousing** projects fail because of faulty data migrations!)