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!
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

digits need to reach the machine binary so that it... calculates!
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 multiple *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 multiple *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
Separation principles (eg. Seheim model, client-server, etc) entail permanent data conversion across disparate technology layers:

![Diagram showing the process of replication (rep) and retrieval (retr) between I, M, and D.]
Running example — genealogy website (I)

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

```
<table>
<thead>
<tr>
<th></th>
<th>Margaret, b. 1923</th>
<th>Luigi, b. 1920</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary, b. 1956</td>
<td>Joseph, b. 1955</td>
<td></td>
</tr>
<tr>
<td>Peter, b. 1991</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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.
Finally channeled to dataware, heap structures are buried into database files as persistent data records:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joseph</td>
<td>1955</td>
</tr>
<tr>
<td>2</td>
<td>Luigi</td>
<td>1920</td>
</tr>
<tr>
<td>3</td>
<td>Margaret</td>
<td>1923</td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>1956</td>
</tr>
<tr>
<td>5</td>
<td>Peter</td>
<td>1991</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Ancestor</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Father</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Mother</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>Father</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Mother</td>
<td>3</td>
</tr>
</tbody>
</table>
Too many paradigms

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

**UML** class diagrams

```
Individual
ID: String
Name: String
Birth: Date
```

Parent
0..2
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)**

```xml
<!-- DTD for genealogical trees -->
<!ELEMENT tree (name, birth, tree?, tree?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birth (#PCDATA)>
```
Too many (programming) paradigms

Plain SQL

```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

\[ c \quad R \quad a \]

to mean that

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

and let the converse fact

\[ a \quad R^\circ \quad c \]

mean

\( a \) **is the datum represented by** \( c \)

(passive voice).
Keep it simple

Let us write

\[ c \ R \ a \]

to mean that

\textit{datum} \( c \) (eg. byte) \textbf{represents} \textit{datum} \( a \) (eg. digit)

and let the converse fact

\[ a \ R^\circ \ c \]

mean

\( a \) \textit{is the datum represented by} \( c \)

(\textbf{passive} voice).
Definite article “the” instead of “a” in sentence

\[ a \text{ is the datum represented by } c \]

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

So \( R \) should be injective:

\[
\langle \forall c, a, a' :: c R a \land c R a' \Rightarrow a = a' \rangle
\] (2)
No confusion, please

Definite article “the” instead of “a” in sentence

\[ a \text{ is the datum represented by } c \]

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

So \( R \) should be injective:

\[
\langle \forall \ c, a, a' :: c \ R \ a \land c \ R \ a' \Rightarrow a = a' \rangle
\]  

(2)
No data loss, please

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

\[ \forall a :: (\exists c :: c R a) \]  \hspace{1cm} (3)

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

\[ R \rightarrow \]

\[ \hspace{1cm} (4) \]

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:

$$\langle \forall a, c :: c \ R \ a \Rightarrow a \ F \ c \rangle$$  \hspace{1cm} (5)

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

In a diagram:

$$A \leq C$$  \hspace{1cm} (6)

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

\[
\begin{array}{c}
A & \overset{R}{\rightarrow} & C \\
\downarrow{\leq} & & \downarrow{\leq} \\
B & \overset{R'}{\rightarrow} & D \\
\end{array}
\]

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 \((D)\):

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:

\[
\text{Proof of downwards simulation theorem for partial correctness (2)}
\]

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

\[
\begin{align*}
\rho[a'/a] \land x' &= x \rightsquigarrow (\varphi \rightsquigarrow \psi); (\rho[a'/a] \land x' = x) = \quad \text{(by (2))} \\
&= \beta \\
\forall x_0', a_0'. (\rho[a_0'/a] \land x_0' = x)[x', c'/x, c] &\to (\exists a. \rho \land \forall x_0. \varphi[x_0', a_0'/x, a] \to \psi) \\
&= \rho[a_0'/a] \land x_0' = x \rightsquigarrow \exists a. \rho \land \forall x_0. \varphi[x_0', a_0'/x, a] \to \psi \\
\end{align*}
\]

QED

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

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

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

\[ 3x + 9 = 48 \]

\[ \equiv \{ \text{“al-djabr” rule} \} \]

\[ 3x = 48 - 9 \]

\[ \equiv \{ \text{“al-hatt” rule} \} \]

\[ x = 16 - 3 \]
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

\[ 3x + 9 = 48 \]
\[ \equiv \begin{cases} \text{“al-djabr” rule} \end{cases} \]
\[ 3x = 48 - 9 \]
\[ \equiv \begin{cases} \text{“al-hatt” rule} \end{cases} \]
\[ x = 16 - 3 \]
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

\[ 3x + 9 = 48 \]

≡ \{ “al-djabr” rule \}

\[ 3x = 48 - 9 \]

≡ \{ “al-hatt” rule \}

\[ x = 16 - 3 \]
School maths example

The solution

\begin{align*}
  x &= 13 \\
  x + 3 &= 16 \\
  x + 6 &= 19
\end{align*}

Questions....

- “al-djabr” rule ?
- “al-hatt” rule ?

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

\[ x = 13 \]
\[ x + 3 = 16 \]
\[ x + 6 = 19 \]

Questions....

- “al-djabr” rule ?
- “al-hatt” rule ?

School maths example

The solution

\[ x = 13 \]
\[ x + 3 = 16 \]
\[ x + 6 = 19 \]

Questions....

- “al-djabr” rule ?
- “al-hatt” rule ?

(...) 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**

\[ x - z \leq y \equiv x \leq y + z \]

**al-hatt**

\[ x \cdot z \leq y \equiv x \leq y \cdot z^{-1} \quad (z > 0) \]

**al-muqâbala**

Ex: \[ 4x^2 - 2x^2 = 2x + 6 - 3 \equiv 2x^2 = 2x + 3 \]
“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, ñ 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. ]
Letting “the symbols do the work” in the 16c

Deduction first

\[ Y \text{ tambien porque quien obra por Algebra va entendiendo la} \]
\[ \text{ razon de la obra que haze, hasta la yqualacion ser acabada.} \]
\[ (...) \text{ De suerte que, quien obra por Algebra, va haziendo} \]
\[ \text{ discursos demonstrativos.} \]

ie.

\[ \text{And also because one performing by Algebra is} \]
\[ \text{understanding the reason of the work one does, until the} \]
\[ \text{equality is finished. (...) So much so that, who works by} \]
\[ \text{Algebra is doing a demonstrative discourse.} \]

[ fol. 269r-269v ]
(...) De manera, que quien sabe por Algebra, sabe científicamente.

(...) in this way, who knows by Algebra knows **scientifically**
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.

\[ .40.\tilde{p}.2.ce.\ son\ yguales\ a\ .20.co \]

(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

\[
\begin{align*}
\nu(t) &= Ri(t) + \frac{1}{C} \int_0^t i(\tau)d\tau \\
\nu(t) &= V_0(u(t - a) - u(t - b)) \quad (b > a)
\end{align*}
\]
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:

![Oscilloscope Trace]

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

Given problem
\[ y'' + 4y' + 3y = 0 \]
\[ y(0) = 3 \]
\[ y'(0) = 1 \]

s-space

Subsidiary equation
\[ s^2 + 4sY + 3Y = 3s + 13 \]

Solution of given problem
\[ y(t) = -2e^{-3t} + 5e^{-t} \]

Solution of subs. equation
\[ Y = \frac{-2}{s+3} + \frac{5}{s+1} \]
Laplace-transformed RC-circuit model

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

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

whose \textit{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 . . .

\[ i(t) = \begin{cases} 
0 & \text{if } t < a \\
\left( \frac{V_0 e^{-\frac{a}{RC}}}{R} \right) e^{-\frac{t}{RC}} & \text{if } a < t < b \\
\left( \frac{V_0 e^{-\frac{a}{RC}}}{R} - \frac{V_0 e^{-\frac{b}{RC}}}{R} \right) e^{-\frac{t}{RC}} & \text{if } t > b 
\end{cases} \]
All we have seen applies to physics, mechanical eng., civil eng., electrical and electronic eng.

What about us? (software engineers)
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?

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

<table>
<thead>
<tr>
<th>[f(t)]</th>
<th>[\mathcal{L}(f)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>[\frac{1}{s}]</td>
</tr>
<tr>
<td>[t]</td>
<td>[\frac{1}{s^2}]</td>
</tr>
<tr>
<td>[t^n]</td>
<td>[\frac{n!}{s^{n+1}}]</td>
</tr>
<tr>
<td>[e^{at}]</td>
<td>[\frac{1}{s-a}]</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\text{PF } \phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \exists a :: b R a \land a S c \rangle$</td>
<td>$b(R \cdot S)c$</td>
</tr>
<tr>
<td>$\langle \forall a, b :: b R a \Rightarrow b S a \rangle$</td>
<td>$R \subseteq S$</td>
</tr>
<tr>
<td>$\langle \forall a :: a R a \rangle$</td>
<td>$id \subseteq R$</td>
</tr>
<tr>
<td>$\langle \forall x :: x R b \Rightarrow x S a \rangle$</td>
<td>$b(R \setminus S)a$</td>
</tr>
<tr>
<td>$\langle \forall c :: b R c \Rightarrow a S c \rangle$</td>
<td>$a(S / R)b$</td>
</tr>
<tr>
<td>$b R a \land c S a$</td>
<td>$(b, c)\langle R, S \rangle a$</td>
</tr>
<tr>
<td>$b R a \land d S c$</td>
<td>$(b, d)(R \times S)(a, c)$</td>
</tr>
<tr>
<td>$b R a \land b S a$</td>
<td>$b(R \cap S) a$</td>
</tr>
<tr>
<td>$b R a \lor b S a$</td>
<td>$b(R \cup S) a$</td>
</tr>
<tr>
<td>$(f \cdot b) R (g \cdot a)$</td>
<td>$b(f \circ R \cdot g)a$</td>
</tr>
<tr>
<td>True</td>
<td>$b \top a$</td>
</tr>
<tr>
<td>False</td>
<td>$b \bot a$</td>
</tr>
</tbody>
</table>

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 \( R \) — \( R^\circ \) 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:

<table>
<thead>
<tr>
<th>ker R</th>
<th>entire R</th>
<th>injective R</th>
</tr>
</thead>
<tbody>
<tr>
<td>img R</td>
<td>surjective R</td>
<td>simple R</td>
</tr>
</tbody>
</table>

where

\[
\ker R = R^\circ \cdot R \\
\img R = R \cdot R^\circ
\]
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

<table>
<thead>
<tr>
<th>Pointwise</th>
<th>Pointfree</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Left” Uniqueness</td>
<td>( \text{img } f \subseteq \text{id} )</td>
</tr>
<tr>
<td>( b f a \land b' f a \Rightarrow b = b' )</td>
<td>( f ) is simple</td>
</tr>
<tr>
<td>Leibniz principle</td>
<td>( \text{id} \subseteq \ker f )</td>
</tr>
<tr>
<td>( a = a' \Rightarrow f a = f a' )</td>
<td>( f ) is entire</td>
</tr>
</tbody>
</table>

• Back to useful “al-djabr” rules:

\[
\begin{align*}
(f \circ R \subseteq S & \equiv R \subseteq f \circ S) \\
R \cdot f^\circ & \subseteq S \equiv R \subseteq S \cdot f
\end{align*}
\]

• 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 $B \xleftarrow{R} A$ or $A \xrightarrow{R} B$ to indicate that $R$ is simple.

We shall be using (simple) relations to model **both** algorithms and data.
“Al-djabr” rules for simple $M$:

\begin{align*}
(M \cdot R) \subseteq T & \equiv (\delta M) \cdot R \subseteq M^\circ \cdot T \quad (8) \\
R \cdot (M^\circ) \subseteq T & \equiv R \cdot \delta M \subseteq T \cdot M \quad (9)
\end{align*}

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

$$\rho R = \text{img} R \cap id$$
Predicate PF-transformed

- **Binary** predicates:
  \[ R = \llbracket b \rrbracket \equiv (y \ R \ x \equiv b(y, x)) \]

- **Unary** predicates become fragments of \textit{id} (coreflexives):
  \[ R = \llbracket p \rrbracket \equiv (y \ R \ x \equiv (p \ x) \land x = y) \]

  eg. (in the natural numbers)

  \[ \llbracket 1 \leq x \leq 4 \rrbracket = \]

![Graph showing a line with points (1,1), (2,2), (3,3), (4,4) and the set {1,2,3,4} on the x-axis. The line represents the coreflexive for the set.]
Boolean algebra of coreflexives

\[ [p \land q] = [p] \cdot [q] \] \hspace{1cm} (10)
\[ [p \lor q] = [p] \cup [q] \] \hspace{1cm} (11)
\[ \neg p = \text{id} - [p] \] \hspace{1cm} (12)
\[ \text{false} = \bot \] \hspace{1cm} (13)
\[ \text{true} = \text{id} \] \hspace{1cm} (14)

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:

\[
\{ a \mapsto S a \mid a \in \text{dom } S \}
\]

where \( \text{dom } S \) is the set-theoretic version of \( \delta S \).

- Useful PF patterns:

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

    \[
    \{ g \ a \mapsto f(S \ a) \mid a \in \text{dom } S \}
    \]

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

    \[
    \{ a \mapsto S a \mid a \in \text{dom } S \land \phi a \land \psi(S a) \}
    \]
Simple relation expressive power

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

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

where $\text{dom } S$ is the set-theoretic version of $\delta S$.

- Useful PF **patterns**:

  - **projection** — $f \cdot S \cdot g^\circ$ ($g$ injective):
    $$\{ g a \mapsto f(S a) \mid a \in \text{dom } S \}$$

  - **selection** — $\Psi \cdot S \cdot \Phi$ ($\Psi, \Phi$ coreflexives):
    $$\{ a \mapsto S a \mid a \in \text{dom } S \land \phi a \land \psi(S a) \}$$
All (data structures) in one (PF notation)

Products
Database records — eg. 5 Peter 1991 — C/C++

structs etc are products:

\[
\begin{align*}
A & \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B \\
\langle R, S \rangle & \quad C
\end{align*}
\]

where

\[
\begin{align*}
\psi & \quad PF \psi \\
(a, b) \langle R, S \rangle c & \quad (b, d)(R \times S)(a, c)
\end{align*}
\]

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. \[\begin{array}{ccc} 5 & \text{Peter} & 1991 \end{array}\] — C/C++

structs etc are products:

\[
\begin{array}{ccc}
A & \overset{\pi_1}{\longleftarrow} & A \times B & \overset{\pi_2}{\longrightarrow} & B \\
\downarrow & & \downarrow & & \downarrow \\
R & \langle R, S \rangle & C & & S
\end{array}
\]

(15)

where

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

(16)

Clearly: \[R \times S = \langle R \cdot \pi_1, S \cdot \pi_2 \rangle\]
Sums

Example (Haskell):

\[
data X = \text{Boo Bool} \mid \text{Err String}
\]

PF-transforms to

\[
\begin{array}{c}
\text{Bool} \xrightarrow{i_1} \text{Bool} + \text{String} \xleftarrow{i_2} \text{String}
\end{array}
\]

where

\[
[R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)
\]

cf.

Dually:

\[
R + S = [i_1 \cdot R, i_2 \cdot S]
\]
Sums

Example (Haskell):

\[
data X = \text{Boo} \Bool \mid \text{Err} \String
\]

PF-transforms to

\[
\begin{array}{c}
\text{Bool} \\
\text{Boo}
\end{array} 
\begin{array}{c}
i_1 \\
i_2
\end{array} 
\begin{array}{c}
\text{Bool} + \text{String} \\
X
\end{array} 
\begin{array}{c}
\text{String} \\
\text{Err}
\end{array}
\]

(17)

where

\[
[R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)
\]

cf.

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:

\[
\text{Maybe } A \overset{\text{def}}{=} A + 1
\]  

(18)

(where 1 is the singleton type inhabited by \texttt{NIL}

• Grammars:

<table>
<thead>
<tr>
<th>BNF notation</th>
<th>Polynomial notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha \mid \beta )</td>
<td>( \alpha + \beta )</td>
</tr>
<tr>
<td>( \alpha \beta )</td>
<td>( \alpha \times \beta )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>1</td>
</tr>
</tbody>
</table>

(19)
Grammars and inductive data models

For instance,

\[ X \rightarrow \epsilon | a \ A \ X \]

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

\[ X = 1 + A \times X \]  \hspace{1cm} (20)

cf.

```c
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 \( PTree \)

\[
data \text{PTree} = \text{Node} \{ \text{name} :: \left[ \text{Char} \right], \text{birth} :: \text{Int}, \text{mother} :: \text{Maybe PTree}, \text{father} :: \text{Maybe PTree} \} \]
\]

becomes

\[
PTree \cong \text{Ind} \times (PTree + 1) \times (PTree + 1) \tag{21}
\]

where \( \text{Ind} = \text{Name} \times \text{Birth} \) packages the information relative to the name and birth year, ie.

\[
PTree \cong G(\text{Ind}, PTree) \tag{22}
\]

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

\[
G(X, Y) \overset{\text{def}}{=} X \times (Y + 1) \times (Y + 1)
\]

( \( X \) refers to the parametric information and \( Y \) to the inductive part.)
Entity-Relationship diagrams

PF-transform of

$$\text{Books} \triangleq ISBN \rightarrow \text{Title} \times (5 \rightarrow \text{Author}) \times \text{Publisher}$$

$$\text{Borrowers} \triangleq \text{PID} \rightarrow \text{Name} \times \text{Address} \times \text{Phone}$$

$$\text{Reserved} \triangleq ISBN \times \text{PID} \rightarrow \text{Date}$$
Business rules

Example
“(…) Only existing books can be borrowed by known borrowers”

Pointwise

\[ \phi(M, N, R) \overset{\text{def}}{=} \langle \forall i, p, d :: d R (i, p) \Rightarrow \langle \exists x :: x M i \rangle \wedge \langle \exists y :: y M p \rangle \rangle \]

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

\[ \phi(M, N, R) \overset{\text{def}}{=} R \preceq M \cdot \pi_1 \land R \preceq N \cdot \pi_2 \]

cf. diagram

whose geometrical similarity with the original is striking, recall:
Data impedance mismatch expressed in the PF-style

\[
\begin{array}{ccc}
A & \leq & B \\
\downarrow R & & \downarrow F \\
\end{array}
\]

where

- \( \ker R = \text{id} \) (representation) and \( \text{img} F = \text{id} \) (abstraction)
- connection between \((R, F)\)
  \[\langle \forall a, b :: b R a \Rightarrow a F b \rangle\]

shrinks to

\[ R^\circ \subseteq F \] (23)

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

\[ R \subseteq F^\circ \] (24)

(\( F^\circ \) largest representation one can connect to retrieve relation \( F \)).
\[ \leq \text{ is a preorder} \]

- \( \leq \text{ is reflexive:} \) Between a datatype and itself

\[
\begin{array}{c}
A \quad \leq \quad A \\
\downarrow \quad \quad \quad \downarrow \quad \quad \quad \quad \quad \downarrow \\
\text{id} \quad \quad \quad \text{id} \quad \quad \quad \quad \quad \quad \text{id}
\end{array}
\]

there is *no impedance* at all

- \( \leq \text{ is transitive:} \)

\[
\begin{array}{c}
A \quad \leq \quad B \quad \leq \quad C \\
\downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
R \quad \quad \quad S \\
\downarrow \quad \quad \quad \downarrow \\
F \quad \quad \quad G
\end{array}
\Rightarrow
\begin{array}{c}
A \quad \leq \quad C \\
\downarrow \quad \quad \quad \downarrow \\
S \cdot R \quad \quad \quad F \cdot G
\end{array}
\]

that is, data impedances compose.
One slide long calculations

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

\[
S \cdot R \subseteq (F \cdot G)^\circ
\]
\[
\equiv \quad \{ \text{converses: } (R \cdot S)^\circ = S^\circ \cdot R^\circ \ \}
\]
\[
S \cdot R \subseteq G^\circ \cdot F^\circ
\]
\[
\leftarrow \quad \{ \text{monotonicity} \ \}
\]
\[
S \subseteq G^\circ \land R \subseteq F^\circ
\]
\[
\equiv \quad \{ \text{since } S, G \text{ and } R, F \text{ are assumed connected} \ \}
\]
\[
\text{TRUE}
\]
Right-invertibility

That $\leq$-rules entail \emph{right-invertibility}

$$F \cdot R = id$$ (25)

is again a one slide long calculation:

$$F \cdot R = id$$

$$\equiv \quad \{ \text{equality of relations} \}$$

$$F \cdot R \subseteq id \land id \subseteq F \cdot R$$

$$\equiv \quad \{ \text{img } F = id \text{ and ker } R = id \}$$

$$F \cdot R \subseteq F \cdot F^\circ \land R^\circ \cdot R \subseteq F \cdot R$$

$$\equiv \quad \{ \text{converses} \}$$

$$F \cdot R \subseteq F \cdot F^\circ \land R^\circ \cdot R \subseteq R^\circ \cdot F^\circ$$

$$\iff \quad \{ (F \cdot) \text{ and } (R^\circ \cdot) \text{ are monotone (cf. GCs)} \}$$

$$R \subseteq F^\circ \land R \subseteq F^\circ$$
Functions only

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

\[
A \overset{r}{\leftrightharpoons} C \quad \equiv \quad f \cdot r = \text{id}
\]  

(26)

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

\[
\begin{align*}
\text{equality of functions} \\
\text{“al-djabr” (shunting)}
\end{align*}
\]
Functions only

\[ r \subseteq f^\circ \]

\[ \Rightarrow \{ \text{composition is monotonic} \} \]

\[ f \cdot r \subseteq f \cdot f^\circ \wedge r^\circ \cdot r \subseteq r^\circ \cdot f^\circ \]

\[ \equiv \{ f \cdot r = id \; ; \text{converses} \} \]

\[ id \subseteq f \cdot f^\circ \wedge r^\circ \cdot r \subseteq id \]

\[ \equiv \{ \text{definitions} \} \]

\[ f \text{ surjective } \wedge r \text{ injective} \]

Equivalence: \( \Rightarrow \) (above) \( + \) \( \Leftarrow \) (which of holds in general)
Well-known surjections and injections

From cancellation-laws

\[ \pi_1 \cdot \langle f, g \rangle = f, \quad \pi_2 \cdot \langle f, g \rangle = g \]

\[ [g, f] \cdot i_1 = g, \quad [g, f] \cdot i_2 = f \]

we get some basic impedance mismatches captured by \( \leq \)-rules:
Pointers and references

Pointers

\[ i_1, A \leq A + 1 \]

\[ [\text{id}, F] \]

References ("references cheaper to move around than referents")

\[ G A \leq (\mathcal{N} \rightarrow A) \times G \mathcal{N} \]

\[ R, Dref \]

(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

\[
\begin{align*}
A & \xrightarrow{r} C \\
A & \xleftarrow{f} C
\end{align*}
\]

hold. This equivales

\[
\begin{align*}
\begin{aligned}
r & \subseteq f^\circ \land f & \subseteq r^\circ \\
\equiv & \quad \{ \text{converses ; equality of relations} \} \\
r^\circ & = f
\end{aligned}
\end{align*}
\]

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

\[
\begin{align*}
f \text{ is a bijection} & \equiv f^\circ \text{ is a function}
\end{align*}
\]
Isomorphic data types

In a diagram:

\[ A \cong C \]

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

Example: function \( \text{swap} \) \( \overset{\text{def}}{=} \langle \pi_2, \pi_1 \rangle \) witnesses

\[ A \times B \cong B \times A \]

(eg. change order of entries in structs; swap order of columns in a spreadsheet, etc.)
When the converse of a function is a function

\[ \text{swap}^\circ \]

\[ = \left\{ \langle R, S \rangle = \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \right\} \]

\[ (\pi_1^\circ \cdot \pi_2 \cap \pi_2^\circ \cdot \pi_1)^\circ \]

\[ = \left\{ \text{converses} \right\} \]

\[ \pi_2^\circ \cdot \pi_1 \cap \pi_1^\circ \cdot \pi_2 \]

\[ = \left\{ \text{back to splits} \right\} \]

\[ \text{swap} \]

So \textit{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,

\[ A + (B + C) \cong (A + B) + C \]  \hspace{1cm} (33)

by calculating the function \( r \) which is the converse of \( f \).

\[ ([id + i_1, i_2 \cdot i_2])^\circ \]

\[ = \{ \text{expand} \ [R, S] \} \]

\[ = \{ \text{converses} \} \]

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]
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,

\[ A + (B + C) \cong (A + B) + C \]  \hspace{1cm} (33)

by calculating the function \( r \) which is the converse of \( f \).

\[
\begin{align*}
\text{id} + i_1 \cdot i_2 & \circ \circ \\
\text{expand } [R, S] & \quad \text{converses} \\
\end{align*}
\]
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,

\[ A + (B + C) \cong (A + B) + C \]  \hspace{1cm} (33)

by calculating the function \( r \) which is the converse of \( f \).

\[
\begin{align*}
&[id + i_1 , i_2 \cdot i_2]^\circ \\
= & \begin{cases}
\text{expand } [R , S] \\
\end{cases} \\
= & \begin{cases}
\text{converses} \\
\end{cases} \\
= & i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ
\end{align*}
\]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \quad (\text{from last slide}) \]

\[ = \{ \text{expand } R + S \} \]

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

\[ = \{ \text{expand } [R, S] \} \]

\[ i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]

\[ = \{ \text{distribution ; associativity} \} \]

\[ i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \]

\[ = \{ \text{wrap up (function!)} \} \]

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

\[ = \{ \text{spruce it} \} \]

\[ [i_1 \cdot i_1, i_2 + id] \]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^o) \cup i_2 \cdot i_2^o \cdot i_2^o \quad \text{(from last slide)} \]
\[ = \quad \{ \text{expand } R + S \} \]
\[ i_1 \cdot [i_1, i_2 \cdot i_1^o] \cup i_2 \cdot i_2^o \cdot i_2^o \]
\[ = \quad \{ \text{expand } [R, S] \} \]
\[ i_1 \cdot (i_1 \cdot i_1^o \cup i_2 \cdot i_1^o \cdot i_2^o) \cup i_2 \cdot i_2^o \cdot i_2^o \]
\[ = \quad \{ \text{distribution ; associativity} \} \]
\[ i_1 \cdot i_1 \cdot i_1^o \cup (i_1 \cdot i_2 \cdot i_1^o \cup i_2 \cdot i_2^o) \cdot i_2^o \]
\[ = \quad \{ \text{wrap up (function!)} \} \]
\[ [i_1 \cdot i_1, [i_1 \cdot i_2, i_2]] \]
\[ = \quad \{ \text{spruce it} \} \]
\[ [i_1 \cdot i_1, i_2 + id] \]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]  
(from last slide)

\[ = \{ \text{expand } R + S \} \]

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

\[ = \{ \text{expand } [R, S] \} \]

\[ i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]

\[ = \{ \text{distribution ; associativity} \} \]

\[ i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \]

\[ = \{ \text{wrap up (function!)} \} \]

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

\[ = \{ \text{spruce it} \} \]

\[ [i_1 \cdot i_1, i_2 + id] \]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \quad \text{(from last slide)} \]

\[ = \begin{cases} \text{expand } R + S \end{cases} \]

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

\[ = \begin{cases} \text{expand } [R, S] \end{cases} \]

\[ i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]

\[ = \begin{cases} \text{distribution; associativity} \end{cases} \]

\[ i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \]

\[ = \begin{cases} \text{wrap up (function!)} \end{cases} \]

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

\[ = \begin{cases} \text{spruce it} \end{cases} \]

\[ [i_1 \cdot i_1, i_2 + id] \]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \quad \text{(from last slide)} \]

\[ = \quad \{ \text{expand } R + S \ \} \]

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

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

\[ i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]

\[ = \quad \{ \text{distribution ; associativity } \ \} \]

\[ i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \]

\[ = \quad \{ \text{wrap up (function!)} \ \} \]

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

\[ = \quad \{ \text{spruce it } \ \} \]

\[ [i_1 \cdot i_1 , i_2 + id] \]
Exercise 12, page 169

\[ i_1 \cdot (id + i_1^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \quad \text{(from last slide)} \]

\[ = \{ \text{expand } R + S \} \]

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

\[ = \{ \text{expand } [R , S] \} \]

\[ i_1 \cdot (i_1 \cdot i_1^\circ \cup i_2 \cdot i_1^\circ \cdot i_2^\circ) \cup i_2 \cdot i_2^\circ \cdot i_2^\circ \]

\[ = \{ \text{distribution ; associativity} \} \]

\[ i_1 \cdot i_1 \cdot i_1^\circ \cup (i_1 \cdot i_2 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ) \cdot i_2^\circ \]

\[ = \{ \text{wrap up (function!)} \} \]

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

\[ = \{ \text{spruce it} \} \]

\[ [i_1 \cdot i_1 , i_2 + id] \]
The following are known isomorphisms involving sums and products:

\[
A \times (B \times C) \cong (A \times B) \times C \tag{34}
\]

\[
A \cong A \times 1 \tag{35}
\]

\[
A \cong 1 \times A \tag{36}
\]

\[
A + B \cong B + A \tag{37}
\]

\[
C \times (A + B) \cong C \times A + C \times B \tag{38}
\]

Guess the relevant isomorphism pairs.
More elaborate isomorphisms

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

\[
r^\circ = f
\]

\[
\equiv \quad \{ \text{introduce variables} \}
\]

\[
\langle \forall \ a, c :: c \ (r^\circ) \ a \equiv c \ f \ a \rangle
\]

\[
\equiv \quad \{ \ b(f^\circ \cdot R \cdot g)a \equiv (f \ b)R(g \ a) \ \}
\]

\[
\langle \forall \ a, c :: r \ c = a \equiv c = f \ a \rangle
\]

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

\[
\langle \forall \ a, c :: b + c = a \equiv c = a - b \rangle \quad (39)
\]

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:

\[
\langle \forall R, k :: k = \Lambda R \equiv R = \in \cdot k \rangle
\]

(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:

\[
\begin{array}{c}
(PB)^A \\
\cong \\
\Lambda \\
\Rightarrow \\
\cong \\
A \rightarrow B
\end{array}
\]
Transposes

Simple relations “are *Maybe functions*”:

\[ k = \text{tot } S \equiv S = i_1^\circ \cdot k \]

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

\[ k = \text{tot } S \equiv \langle \forall b, a :: b \ S \ a \equiv (i_1 b) = k a \rangle \]

Diagram:

\[
\begin{array}{c}
(B + 1)^A \cong A \rightarrow B \\
\end{array}
\]

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

Isomorphism

\[
(C \to A)^B \cong B \times C \to A
\]  \hspace{1cm} (41)

and associated universal property,

\[
k = \overline{R} \equiv \forall a, b, c :: a (k b) c \equiv a R (b, c)
\]  \hspace{1cm} (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 $R := f$)

\[
\begin{array}{c}
(A^C)^B \\ \cong \\
\vdots \\
A^{B \times C}
\end{array}
\]

Associated universal property

\[
k = \bar{f} \equiv \langle \forall b, c :: (k b) c = f (b, c) \rangle
\]

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

\[
RDBT \overset{\text{def}}{=} \prod_{i=1}^{n} \left( \prod_{j=1}^{n_i} K_j \rightarrow \prod_{k=1}^{m_i} D_k \right)
\]  

(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

\[
RDBT \overset{\text{def}}{=} \prod_{i=1}^{n} (\prod_{j=1}^{n_i} K_j \rightarrow \prod_{k=1}^{m_i} D_k)
\]

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:

\[
(B + C) \rightarrow A \cong (B \rightarrow A) \times (C \rightarrow A)
\]  

(46)

Universal property:

\[
T = [R, S] \equiv T \cdot i_1 = R \quad \land \quad T \cdot i_2 = S
\]

(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,

$$
A \rightarrow (B + C) \cong (A \rightarrow B) \times (A \rightarrow C)
$$  \hspace{1cm} (48)

where

$$
M \triangleright N \overset{\text{def}}{=} i_1 \cdot M \cup i_2 \cdot N
$$  \hspace{1cm} (49)

$$
\triangle_+ M \overset{\text{def}}{=} \langle i_1^\circ \cdot M, i_2^\circ \cdot M \rangle
$$  \hspace{1cm} (50)
Getting rid of multivalued attributes

Recall that

\[ Books \overset{\text{def}}{=} ISBN \rightarrow Title \times (5 \rightarrow Author) \times Publisher \]

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

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

\[ A \rightarrow (D \times (B \rightarrow C)) \]
New $\leq$-rules from old

\[ A \rightarrow (D \times (B \rightarrow C)) \]

<table>
<thead>
<tr>
<th>$\Rightarrow$</th>
<th>{ \textit{Maybe transpose} }</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Rightarrow$</td>
<td>$(D \times (B \rightarrow C) + 1)^A$</td>
</tr>
<tr>
<td>$\Leftrightarrow$</td>
<td>{ \textit{exercise 10} }</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$((D + 1) \times (B \rightarrow C))^A$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>{ \textit{splitting:} $(B \times C)^A \cong B^A \times C^A$ }</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$(D + 1)^A \times (B \rightarrow C)^A$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>{ \textit{Maybe transpose and relational (un)currying} }</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$(A \rightarrow D) \times (A \times B \rightarrow C)$</td>
</tr>
</tbody>
</table>
Getting rid of multivalued attributes

In summary:

\[
A \to (D \times (B \to C)) \quad \trianglelefteq \quad (A \to D) \times (A \times B \to C) \tag{51}
\]

Illustration:

\[
Books = ISBN \to (Title \times (5 \to Author) \times Publisher)
\]

\[
\cong_1 \quad \{ \quad r_1 = id \to \langle \langle \pi_1, \pi_3 \rangle, \pi_2 \rangle, \quad f_1 = id \to \langle \pi_1 \cdot \pi_1, \pi_2, \pi_2 \cdot \pi_1 \rangle \quad \}
\]

\[
ISBN \to (Title \times Publisher) \times (5 \to Author)
\]

\[
\trianglelefteq_2 \quad \{ \quad r_2 = \triangle_n, \quad f_2 = \bigtriangleup_n, \quad \text{cf. (??)} \quad \}
\]

\[
(ISBN \to Title \times Publisher) \times (ISBN \times 5 \to Author)
\]

\[
= Books_2
\]
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:

\[
\begin{align*}
A & \\
A_1 & \\
A_2 & \\
A_3[0:N] & \\
id: A_1 & \\
\end{align*}
\Leftrightarrow
\begin{align*}
A' & \\
A_1 & \\
A_2 & \\
id: A_1 & \\
rA & \\
0:N & \\
1:N & \\
\end{align*}
\begin{align*}
EA_3 & \\
K_3 & \\
A_3 & \\
id: K_3 & \\
\end{align*}
\]

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

Starting point

\[A = A_1 \rightarrow A_2 \times P A_3\]
Checking model transformations

\[ A_1 \rightarrow A_2 \times \mathcal{P}A_3 \]

\[ \leq_1 \{ \text{create references to } A_3 \} \]

\[ (K_3 \rightarrow A_3) \times (A_1 \rightarrow A_2 \times \mathcal{P}K_3) \]

\[ \equiv_2 \{ \mathcal{P}A \cong A \rightarrow 1 \} \]

\[ (K_3 \rightarrow A_3) \times (A_1 \rightarrow A_2 \times (K_3 \rightarrow 1)) \]

\[ \leq_3 \{ \text{unnest } (\rightarrow) \} \]

\[ (K_3 \rightarrow A_3) \times ((A_1 \rightarrow A_2) \times (A_1 \times K_3 \rightarrow 1)) \]

\[ \equiv_4 \{ \text{introduce ternary product} \} \]

\[ (A_1 \rightarrow A_2) \times (\mathcal{P}(A_1 \times K_3)) \times (K_3 \rightarrow A_3) \]

\[ \overbrace{A'} \quad \overbrace{rA} \quad \overbrace{EA_3} \]
Calculating model transformations

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

\[ A_1 \Rightarrow A_2 \times \mathcal{P}A_3 \leq (A_1 \Rightarrow A_2) \times (\mathcal{P}(A_1 \times K_3)) \times (K_3 \Rightarrow A_3) \]

thanks to the **accuracy** of PF-reasoning.
On the impedance of recursive data models

Recall that our starting model for family trees is recursive:

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

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

\[
PTree \cong Ind \times (PTree + 1) \times (PTree + 1)
\]

In general

\[
\mu G \cong G \mu G
\]
Getting rid of $\mu$'s

\[
\begin{array}{c}
\mu G \\ \leq \\
Unf
\end{array} \xrightarrow{R} (K \xrightarrow{G K} \times K) \quad \text{“heap”}
\]

(52)

where $K$ ia as a data type of “heap addresses” and $K \xrightarrow{G K}$ 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

$$r \ (\text{Node} \ n \ b \ m \ f) = \text{let} \ x = \text{fmap} \ r \ m$$
$$y = \text{fmap} \ r \ f$$
$$\text{in} \ \text{merge} \ (n,b) \ x \ y$$

where \text{merge} performs \textit{separated union} of heaps

$$\text{merge} \ a \ \text{Nothing} \ \text{Nothing} =$$
$$\text{Heap} \ ([1 \mapsto (a, \text{Nothing, Nothing})]) 1$$
$$\text{merge} \ a \ (\text{Just} \ x) \ (\text{Just} \ y) =$$
$$\text{Heap} \ ([1 \mapsto (a, \text{Just} \ k1, \text{Just} \ k2)] ++ h1 ++ h2) 1$$
$$\text{where} \ (\text{Heap} \ h1 \ k1) = \text{bmap id even}_\ x$$
$$(\text{Heap} \ h2 \ k2) = \text{bmap id odd}_\ y$$

....
....
Data “heapification”

Source

```
t= Node {name = "Peter", birth = 1991,
         mother = Just (Node {name = "Mary", birth = 1956,
                              mother = Nothing,
                              father = Just (Node {name = "Jules",
                                                   birth = 1917,
                                                   ...... }})
         ...... }}
```

“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))]
```
Abstractions are “unfolds”

Abstraction is a (partial!) unfold:

\[
    f \text{(Heap } h \text{ } k) = \text{ let } \text{Just} \ (a,x,y) = \text{lookup} \ k \ h \\
    \text{in } \text{Node} \ (\text{fst } a)(\text{snd } a) \\
    \text{(fmap } (f \ . \ \text{Heap } h) \ x) \\
    \text{(fmap } (f \ . \ \text{Heap } h) \ y)
\]

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

Thanks to the \( \leq \)-rule

\[
f(r \ t) = t \text{ always holds.}
\]
Abstractions are “unfolds”

Abstraction is a (partial!) unfold:

\[
\begin{align*}
f (\text{Heap } h \ k) &= \text{let } \text{Just } (a,x,y) = \text{lookup } k \ h \\
& \quad \text{in } \text{Node } (\text{fst } a)(\text{snd } a) \\
& \quad (\text{fmap } (f \ . \ \text{Heap } h) x) \\
& \quad (\text{fmap } (f \ . \ \text{Heap } h) y)
\end{align*}
\]

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

Thanks to the \(\leq\)-rule

\[f(r \ t) = t \text{ always holds.}\]
Boiling recursion down to SQL

\[ PTree \]

\[ \mu_G \]

\[ \leq_2 \]

\[ \leq_3 \]

\[ \leq_4 \]

\[ \leq_5 \]

\[ \leq_6 \]
Boiling recursion down to SQL

\[( (K \rightarrow \text{Ind}) \times (K \times 2 \rightarrow K)) \times K \]
\[\cong_7 \{ \ r_7 = \text{flatl}, f_7 = \text{flatl}^\circ \ \} \]
\[(K \rightarrow \text{Ind}) \times (K \times 2 \rightarrow K) \times K \]
\[=_{8} \{ \text{since Ind} = \text{Name} \times \text{Birth} \ \} \]
\[(K \rightarrow \text{Name} \times \text{Birth}) \times (K \times 2 \rightarrow K) \times K \]

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\text{-calculus is structural: given parametric type } G,\

\[ \begin{aligned}
A & \leq B & \Rightarrow & \quad G A & \leq G B \\
\end{aligned} \]

\[ \begin{aligned}
\xymatrix{
A \ar[r]^{R} & B \\
& F & & \ar[r]_{G F} & G B \\
& G R & & \ar[r]_{G R} & G A \\
A \ar@/_/[rru] & & & & B \ar@/^/[llu] \\
} \]

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

\[ A \leq C \land B \leq D \Rightarrow A + B \leq C + D \]
The 2LT engine is based on *strategic* term rewriting.
2LT demos: {XML, VDM} ↔ 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 ≤-rules

Not only connectivity of

Not only connectivity of

\[ \pi_1^\circ \rightarrow T \times S \xrightarrow{\text{putback}} \]

\[ \leq \]

\[ \text{get} \rightarrow S \]

\[ T \xrightarrow{\text{get}} \leq \]

\[ \pi_1 \rightarrow S \]

cf.

\[ \text{putback} \cdot \pi_1^\circ \subseteq \text{get}^\circ \]

\[ \equiv \quad \{ \text{“al-djabr” twice (functions)} \} \]

\[ \text{get} \cdot \text{putback} \subseteq \pi_1 \]

\[ \equiv \quad \{ \text{equality of functions} \} \]

\[ \text{get} \cdot \text{putback} = \pi_1 \]

\[ \equiv \quad \{ \text{add variables: acceptability} \} \]

\[ \text{get}(\text{putback}(v, s)) = v \]
Lenses relate to $\leq$-rules

... but also connectivity of:

$$\langle \text{get}, \text{id} \rangle \rightarrow \leq \rightarrow T \times S$$

\[ \text{putback} \]

cf.

$$\langle \text{get}, \text{id} \rangle \subseteq \text{put}^\circ$$

$$\equiv \{ \text{“al-djabr”} \}$$

$$\text{putback} \cdot \langle \text{get}, \text{id} \rangle \subseteq \text{id}$$

$$\equiv \{ \text{add variables: stability} \}$$

$$\text{putback}(\text{get } s, s) = s$$
View-update context

\[ \mathcal{L}u \overset{\text{def}}{=} \text{putback} \cdot (u \times \text{id}) \cdot \langle \text{get}, \text{id} \rangle \]

Note that \textbf{stability} is nothing but lens \( \mathcal{L} \) preserving the identity update:

\[ \mathcal{L}\text{id} = \text{id} \]

Seeking compositionality:

\[ \mathcal{L}(u_1 \theta u_2) = (\mathcal{L}u_1) \phi (\mathcal{L}u_2) \Leftarrow \ldots \]
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!)