The Bridge between Mathematical Models of Physics and Generic Simulations

Dániel Berényi - Wigner Research Centre for Physics
Gábor Lehel – Eötvös University

Logic, Relativity and Beyond
2015, Budapest

Early summary

Motivating Category Theory as a common language to use between Mathematics, Physics and Programming.

An example from Physics

Basis changes

- Covariant quantities (like linear functionals)
  Successive basis transformations, M and N act as:
  $$M \cdot N$$

- Contravariant quantities (vectors)
  Successive basis transformations, M and N act as:
  $$N^{-1} \cdot M^{-1}$$

An example from Programming

Apply an single valued function on a list

Like:

- multiply a list of numbers by 2
- Take the square root of each of a list of numbers

Programming Paradigms

Imperative Programming:

- Von Neumann systems: an abstract model of hardware
- Sequence of commands, r/w memory cells

```
A := [1, 2, 3, 4]
i := 0
while(i ≤ 3)
   A_i := A_i · 2
   i := i + 1
```

Programming Paradigms

Imperative Programming:

- Von Neumann systems: an abstract model of hardware
- Sequence of commands, r/w memory cells

```
A := [1, 2, 3, 4]
i := 0
while(i ≤ 3)
    Aᵢ:= Aᵢ · 2
    i := i + 1
```

Two problems:

- Low level, unnecessary details
  → Functional Programming
- Error prone
  → Type Theory

Programming Paradigms

Functional Programming
(solution to the low-levelness):

- Definitions of functions in terms of other functions
- Declarative (what-to-do instead of how-to-do)
- Functions first class citizens (and higher-order functions)

```
A = [1, 2, 3, 4]
B = map (·2) A
```

Type Theory

Typed programming languages
(solution to error proneness): **Terms** and **Types**

- Each term has a type,
- operations on terms may be restricted to certain types

$A : [\mathbb{Z}]$

$\text{map} : (\mathbb{Z} \to \mathbb{Z}) \times [\mathbb{Z}] \to [\mathbb{Z}]$

```
A = [1, 2, 3, 4]
B = map (·2) A
```

Generic Programming

Abstraction over types:

A : $[\mathbb{R}]$

map : $\forall a\,.\,(a \to a) \times [a] \to [a]$

```
A = [1.6, 2.5, 3.1, π]
B = map (√•) A
```

more general version:

map : $\forall a, b\,.\,(a \to b) \times F\ a \to F\ b$

Another example

Consider equivalence relations:

eq : $\mathbb{Z} \times \mathbb{Z} \rightarrow \{\text{True} \vee \text{False}\}$

or:

eq : $\mathrm{a} \times \mathrm{a} \rightarrow \{\text{True} \vee \text{False}\}$ for some a

What if, we'd like to modify this to be an equivalence over lists and use something like 'map' to compose a length function to 'eq'?

$\text{map}_2 : \forall a, b \,.\, (b \rightarrow a) \times F\ a \rightarrow F\ b$

Successive maps

Investigating the properties, we find that:

$$map\ f\ \circ\ map\ g\ \equiv map\ (f \circ\ g\ )$$
$$map_2\ f\ \circ\ map_2\ g\ \equiv map_2\ (g \circ f\ )$$

Successive maps

Investigating the properties, we find that:

$$map\ f \circ map\ g \equiv map\ (f \circ g)$$
$$map_2\ f \circ map_2\ g \equiv map_2\ (g \circ f)$$

We've already seen this earlier:

Successive basis transformations on linear functionals act similarly to map,
Successive basis transformations on vectors act similarly to map_2!

Is there anything deeper here?

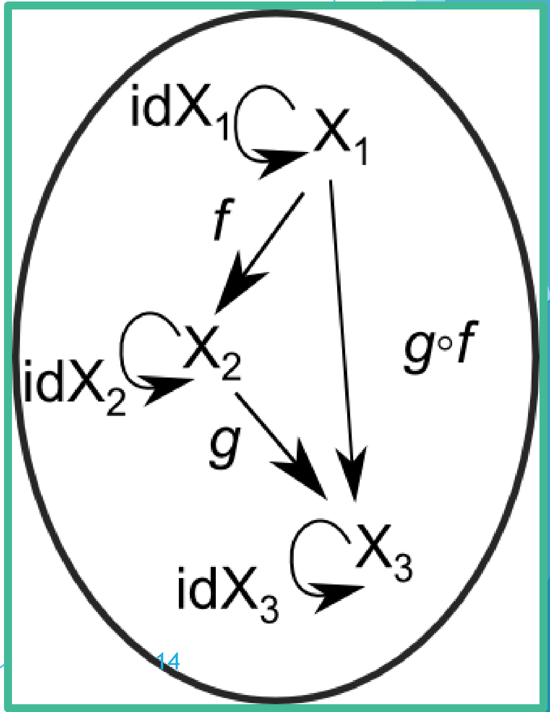
Category Theory

In a nutshell…

Category Theory

A category consists of:

- A collection of **Objects**, dented by capital letters: X
- A collection of **Morphisms**, that map between objects: $X \to Y$
- the binary operation of **Morphism Composition**



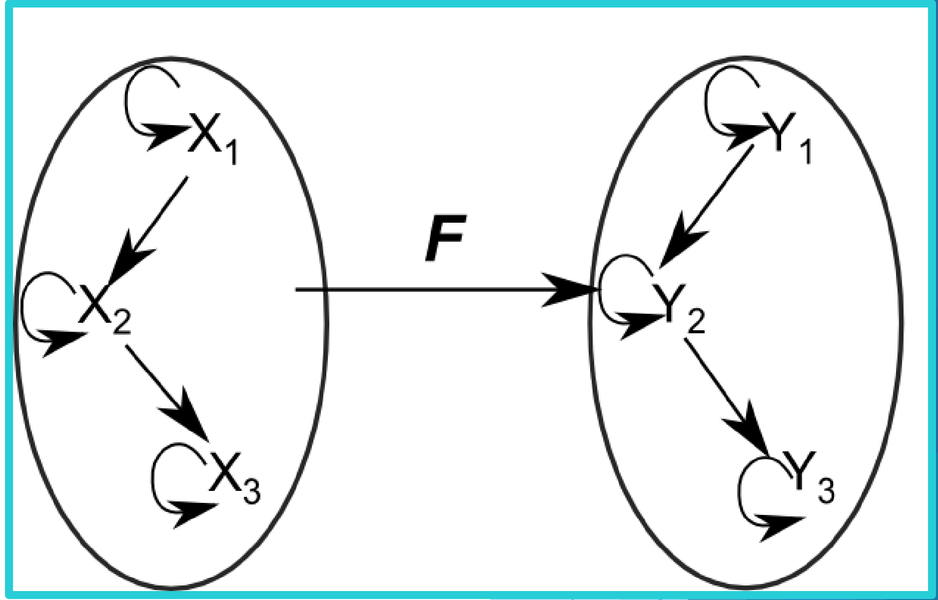
Required properties:

- **Associativity** of Morphism composition
- Existence of **Identity morphisms** for all objects

Category Theory - Functors

Functor from $C \rightarrow D$:

- $X \in C \rightarrow F(X) \in D$
- $f: X \rightarrow Y \in C \rightarrow F(f): F(X) \rightarrow F(Y) \in D$



Such that:

- $F(id_X) \rightarrow id_{F(X)}\ \forall X \in C$
- $F(g \circ f) = F(g) \circ F(f)\ \forall f, g \in C$

Category Theory - Functors

Example: Lists and Natural numbers

- binary operations:
  - On lists: list concatenation:
    [a, b, c] + [d, e] = [a, b, c, d, e]
  - On naturals:
    addition

- Functor: Length of List

Category Theory - Functors

Some functors reverse the direction of morphisms:

Covariant Functors, F: $C \to D$:

- $f: X \to Y \ \in C \to F(f): F(\mathbf{X}) \to F(\mathbf{Y}) \in \ D$
- $F(g \circ f) = F(\mathbf{g}) \circ F(\mathbf{f}) \ \forall f, g \in C$

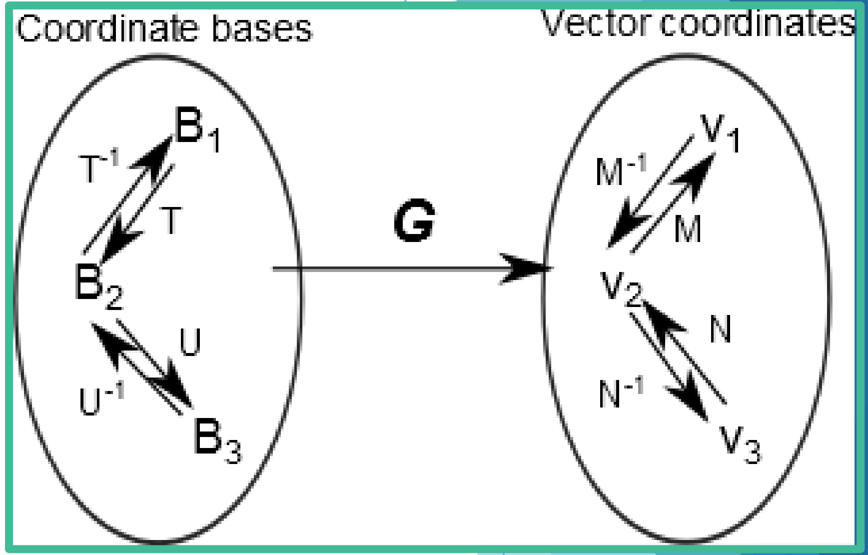
Contravariant Functors, G: $C \to D$:

- $f: Y \to X \ \in C \to G(f): G(\mathbf{Y}) \to G(\mathbf{X}) \in \ D$
- $G(g \circ f) = G(\mathbf{f}) \circ G(\mathbf{g}) \ \forall f, g \in C$

Functors in Physics

See the online paper for details!

Example from Physics:
Consider the following category $\boldsymbol{ABS}$:
Objects: Bases of a Vector space,
Morphisms: Basis changes



and an other category $\boldsymbol{REPR}$ that consists of coordinate representations of $\boldsymbol{ABS}$.

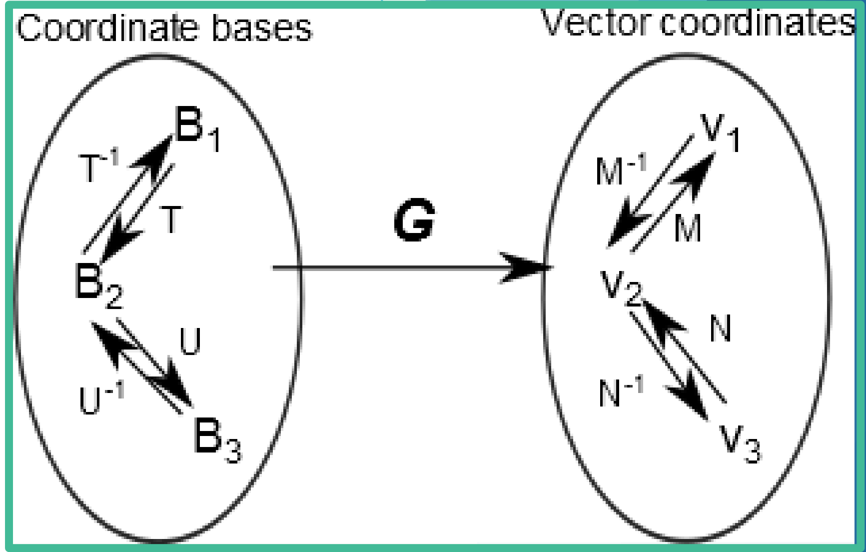
For a fixed vector v, the Functor $F_v : \boldsymbol{ABS} \rightarrow \boldsymbol{REPR}$ is contravariant, since the basis transformation matrices act in the reverse order.

For a fixed linear functional ϕ, the Functor G_ϕ: $\boldsymbol{ABS} \rightarrow \boldsymbol{REPR}$ is covariant, since the basis transformation matrices act in normal order.

Functors of Physics in Haskell

Covariant class – fmap:

```
class Functor G where
      fmap :: (B1 → B2) → (G B1 → G B2)
```

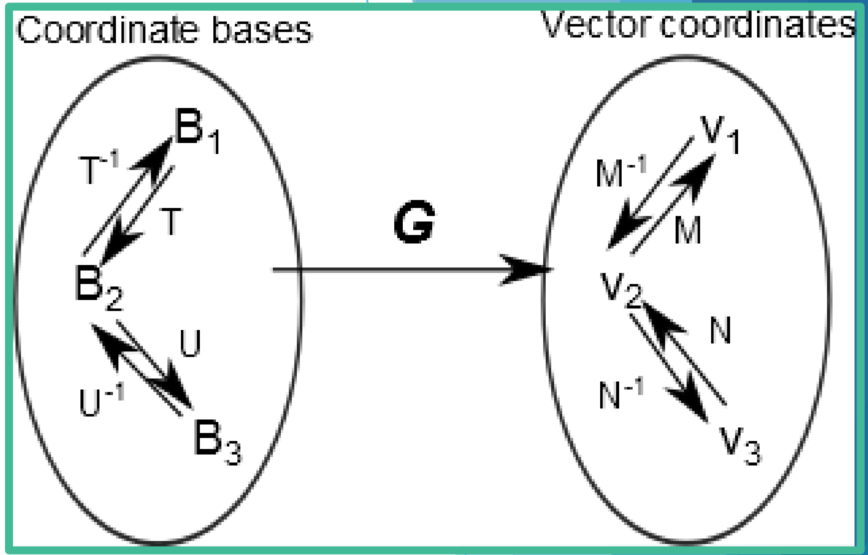


In words:
fmap can take an abstract basis change and create the coordinate representation of it

Functors of Physics in Haskell

Covariant class – fmap:

```
class Functor G where
      fmap :: (B1 → B2) → (G B1 → G B2)
```



Contravariant class – contramap:

```
class Contravariant F where
      contramap :: (B2 → B1) → (F B1 → F B2)
```

contramap takes the inverse of the abstract coordinate transform!

Functors In Generic Programming

Series of abstractions in a generic linear algebra library:

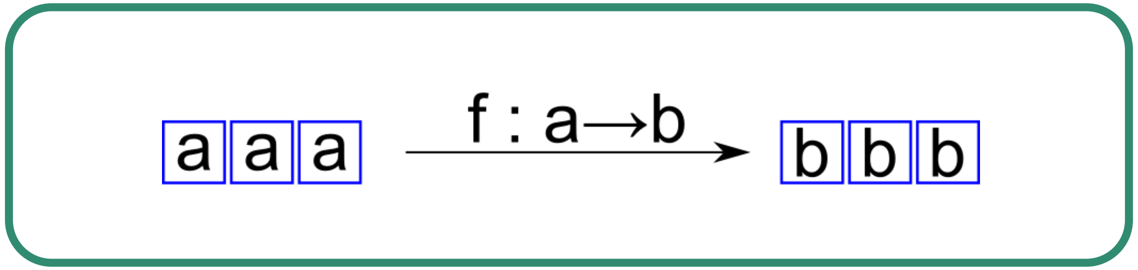
- Vector of doubles – Scalar Multiplication function
- Vector of doubles – Generic Unary operation
- Vector of any type – Generic Unary operation

See the online paper for an example in C++!

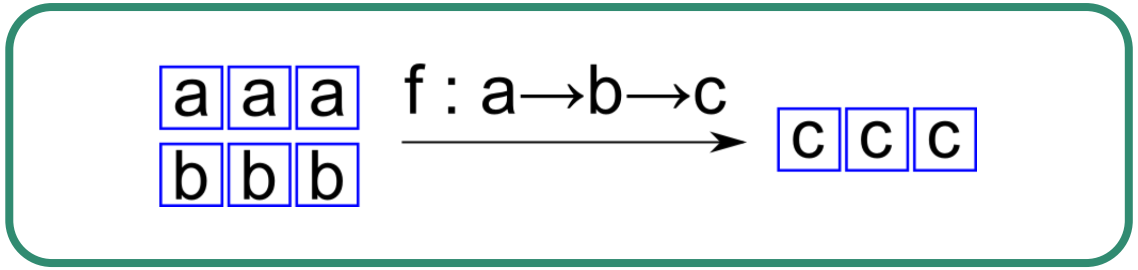
This leads to the implementation of fmap and
the application of the implicit “concept” of Functor over the Vector of any type

ZipWith

fmap on containers can be viewed:



Easily generalized to n-ary functions, called zipWith:

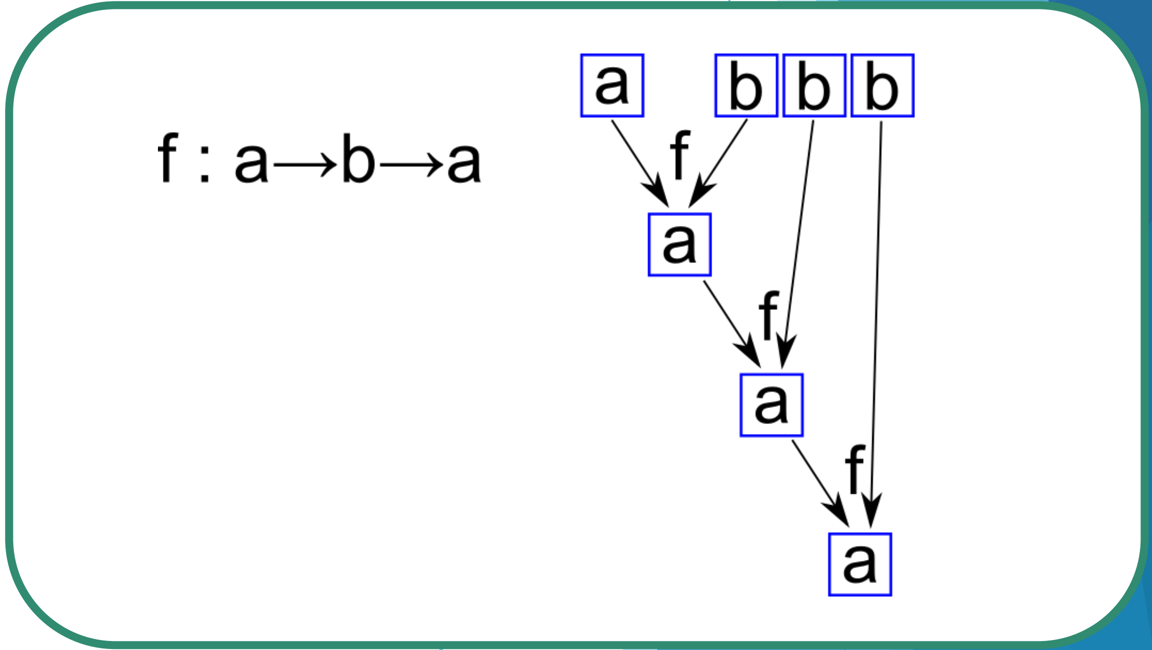


Fold

Another important concept is the **Foldable**, whose method is fold (from the left):

```
foldl (a -> b -> a) -> a -> Foldable b -> a
```

On container like structures it is like:



Example: The case of a linear Algebra library

If **fmap**, **zipwith** and **fold** are available, we can express everything that people usually want from a linear algebra library.

Scalar multiplication:

```
sclmul v x = fmap (*x) v
```

The dot product for example:

```
dot u v = foldl (+) 0 (zipWith (*) u v)
```

The dyadic product for example:

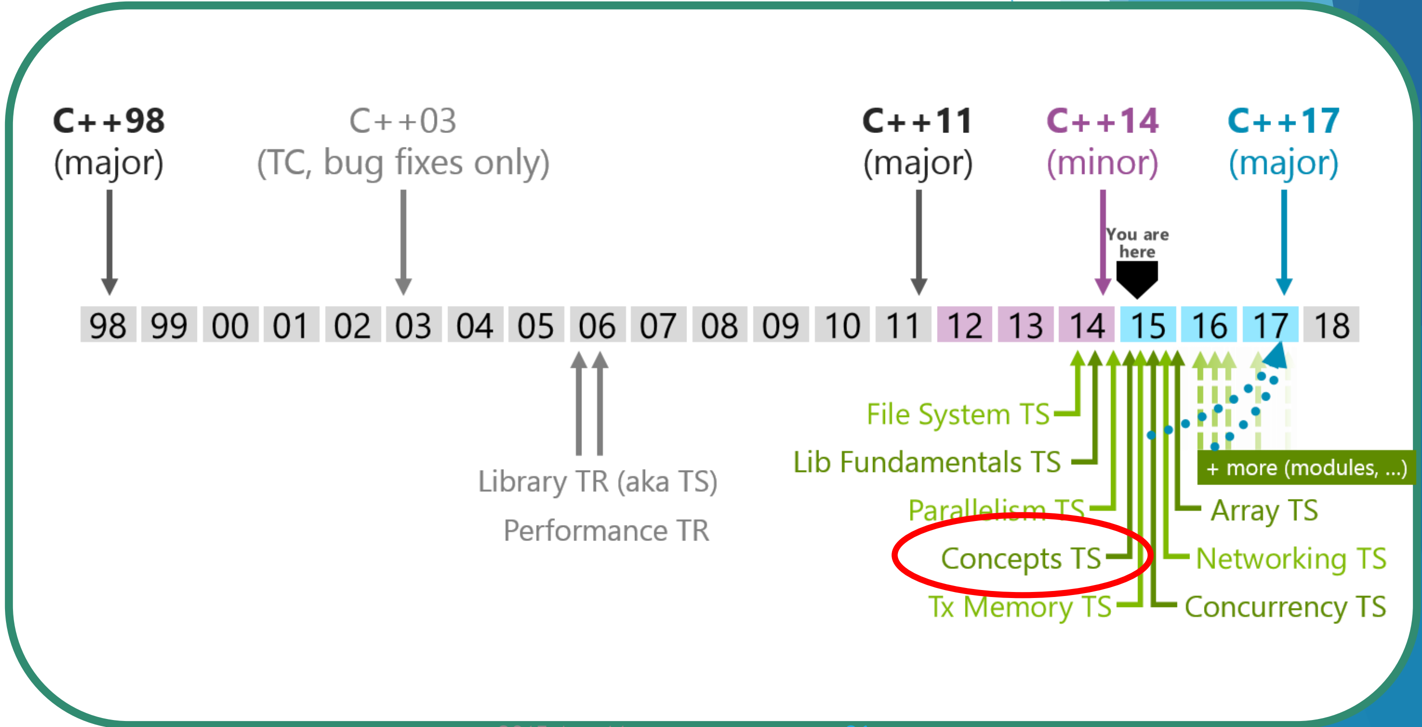
```
dyadic u v = fmap (sclmul v) u
```

Outlook

Future tendencies

Until very recently these programming concepts were just seen as toys of research in academic programming languages

However, recent directions in the evolution of mainstream programming languages (like C++) shows a drastic shift towards functional and generic programming!



Future tendencies

Todays physics simulations and other HPC solutions have to parallelize calculations in order to utilize hardware.

When combining generic programming with automatic parallelization, abstractions like the presented ones from Category Theory are ubiquitous!

Future tendencies

If programmers, physicists, mathematicians would agree to speak a common language,

Category Theory,

they could be more effective in their own fields and their collaborative efforts.

Correspondences

Curry-Howard Correspondence:

Logic	Programming
Proposition	Type
Proof	Program
Disjunction	Sum type (tagged union)
Conjunction	Product type (struct, tuple)
Implication	Function
Invalidity	Uninhabited type (bottom type)

Correspondences

More correspondences (John C. Baez [arxiv:0903.0340]):
Category Th., Logic, Topology, Physics, Computation

Category Theory	Physics	Topology	Logic	Programming
Object	Hilbert space	Manifold	Proposition	Type
Morphism	Operator	Cobordism	Proof	Program
Tensor Product	Hilbert space of joint system	Disjoint union of manifolds	Conjunction	Product type (struct, tuple)
Internal Homomorphism	Hilbert space of anti-X and Y	Disjoint union of orientation-reversed X and Y	Implication	Function

Further outlook

- Homotopy Type Theory:
  http://homotopytypetheory.org/
- Urs Schreiber - Differential cohomology in a cohesive infinity-topos:
  arxiv:1310.7930

Thank you for your attention!

Work supported in part by
OTKA grant: NK 106119

Type Theory

Origins: The need to avoid paradoxes in formal logic

- Example: predicate cannot refer to its self

Ingredients: **Terms** and **Types**

Each term has a type, operations may be restricted to certain types

Contrast to Set Theory:

- Constructive (No Law of Excluded middle)
- can be run as a program