Applied Compositional Thinking for Engineers

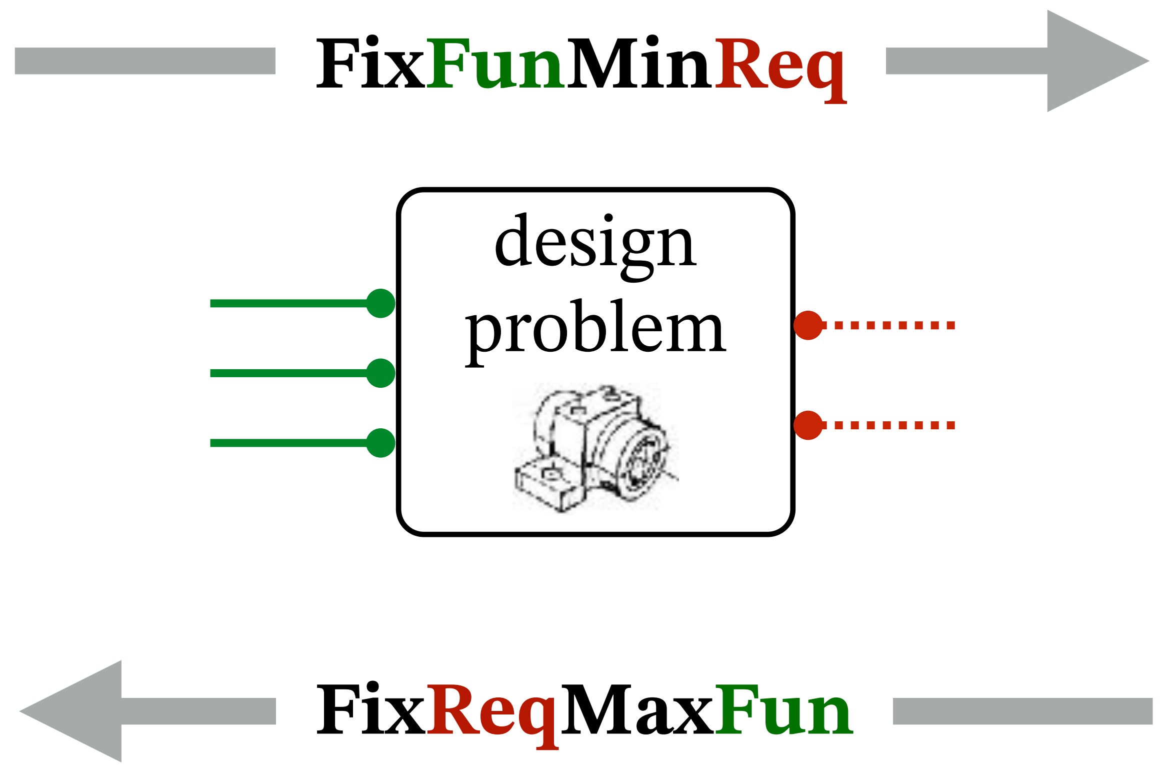
Session 12

Computation

Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided,
what are the **minimal resources** required?

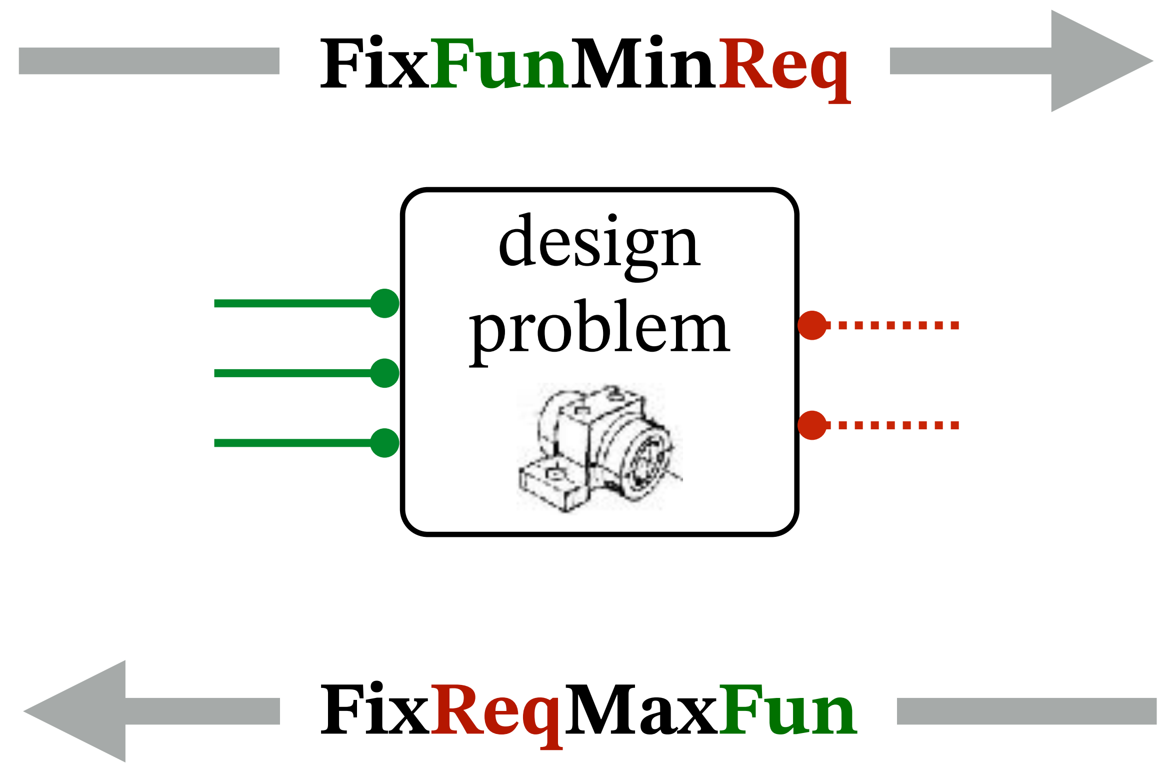


Given the resources that are available, what is
the **maximal functionality** that can be provided?

Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided, what are the **minimal resources** required?



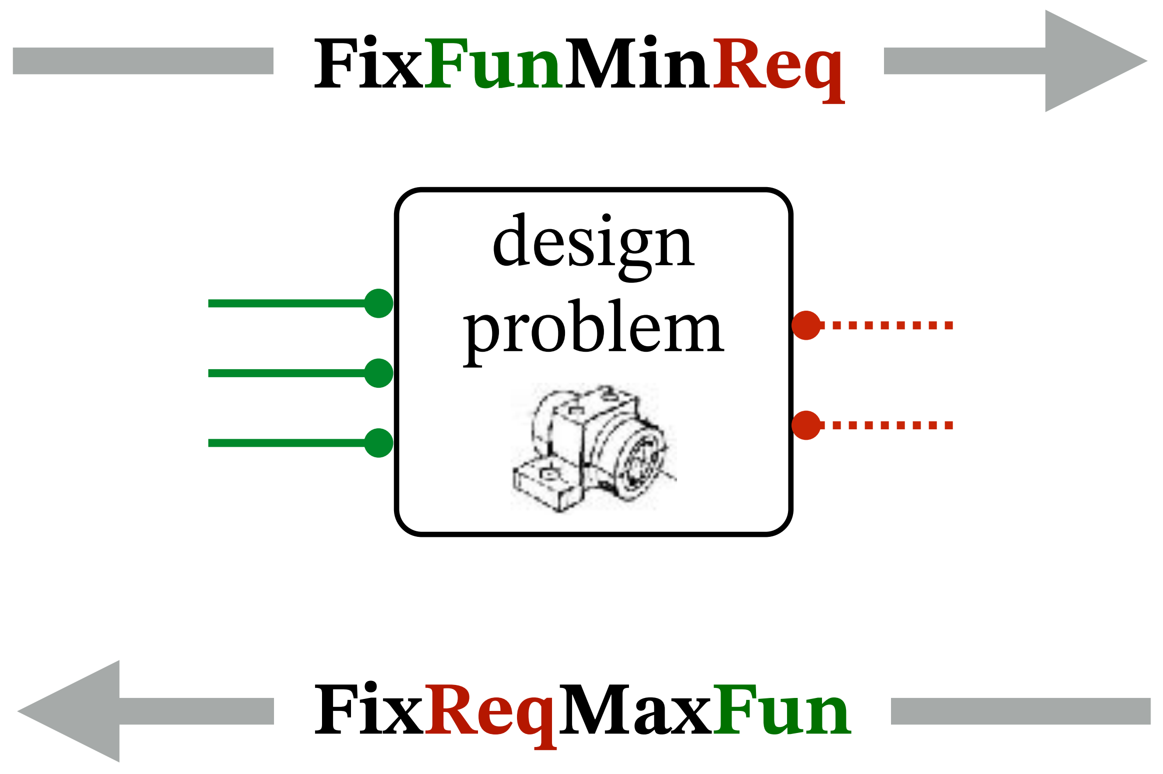
Given the resources that are available, what is the **maximal functionality** that can be provided?

- **The two problems are dual.**
- If you know how to solve these problems, you can also get the implementations with some book-keeping. We will forget about the implementations.

Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided, what are the **minimal resources** required?



Given the resources that are available, what is the **maximal functionality** that can be provided?

- Other variations of the problem, having constraints on both sides and mixed objectives, are formally equivalent:

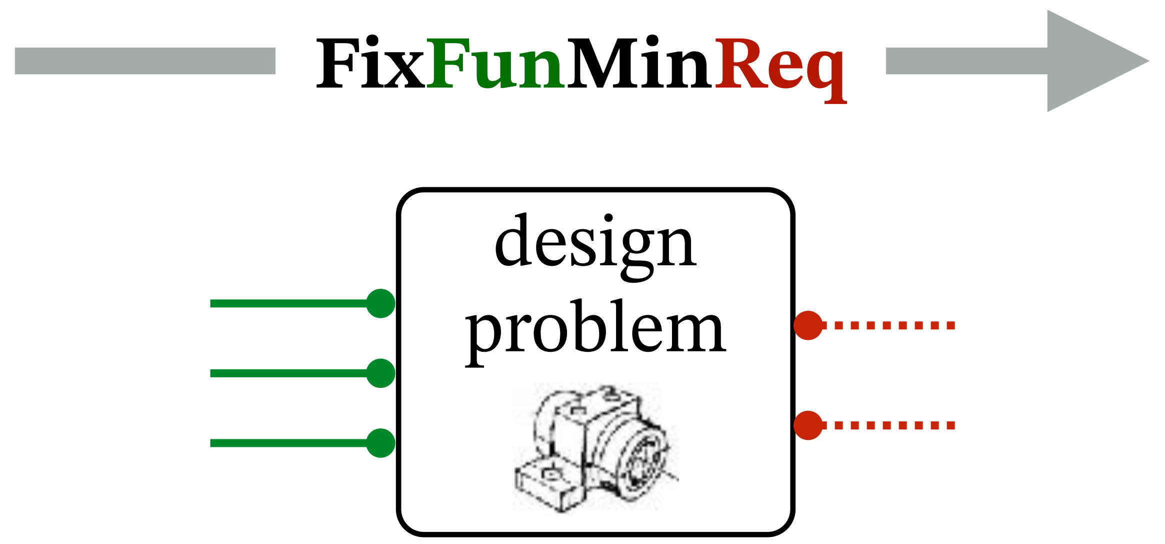
$$A \times B \nrightarrow C \times D$$

$$A \times B \times D^{\mathrm{op}} \nrightarrow C \quad \cong \quad A \times B \nrightarrow C \times D \quad \cong \quad B \nrightarrow C \times D \times A^{\mathrm{op}}$$

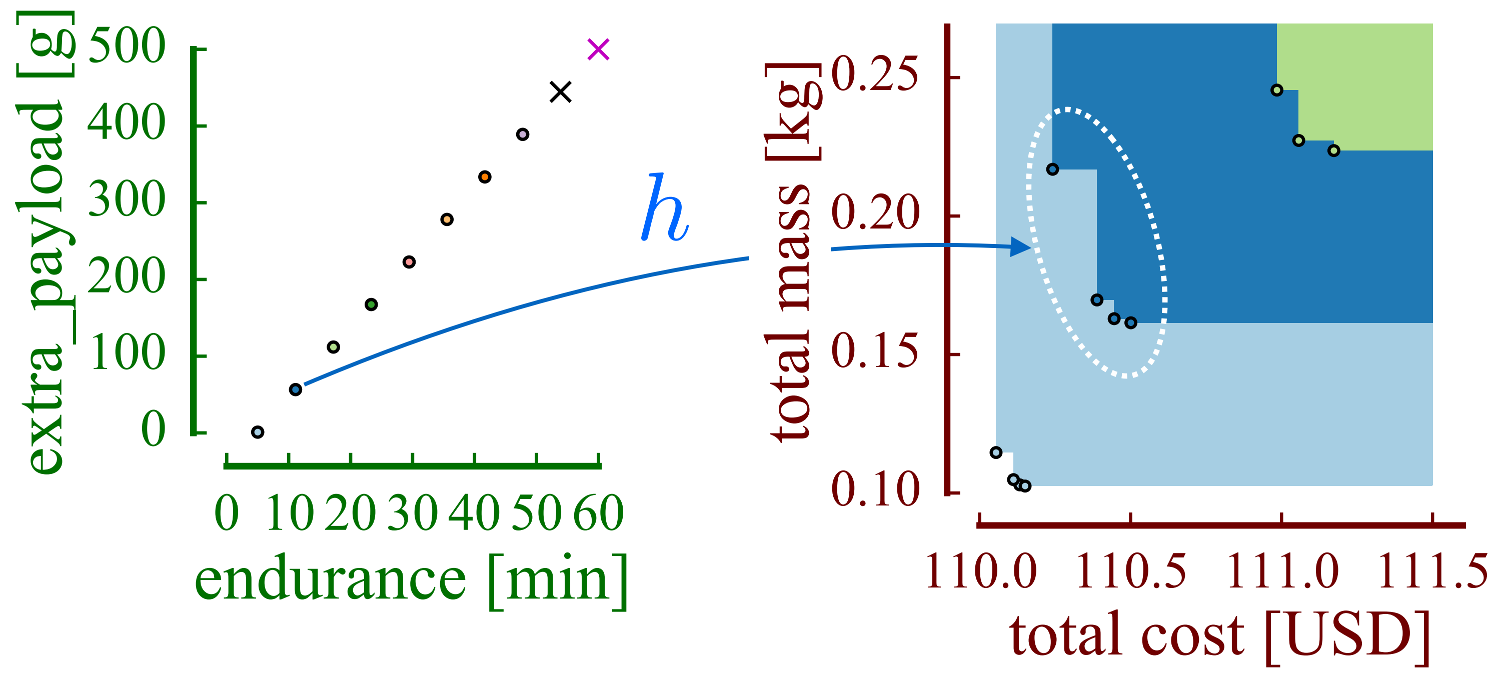
Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided,
what are the **minimal resources** required?



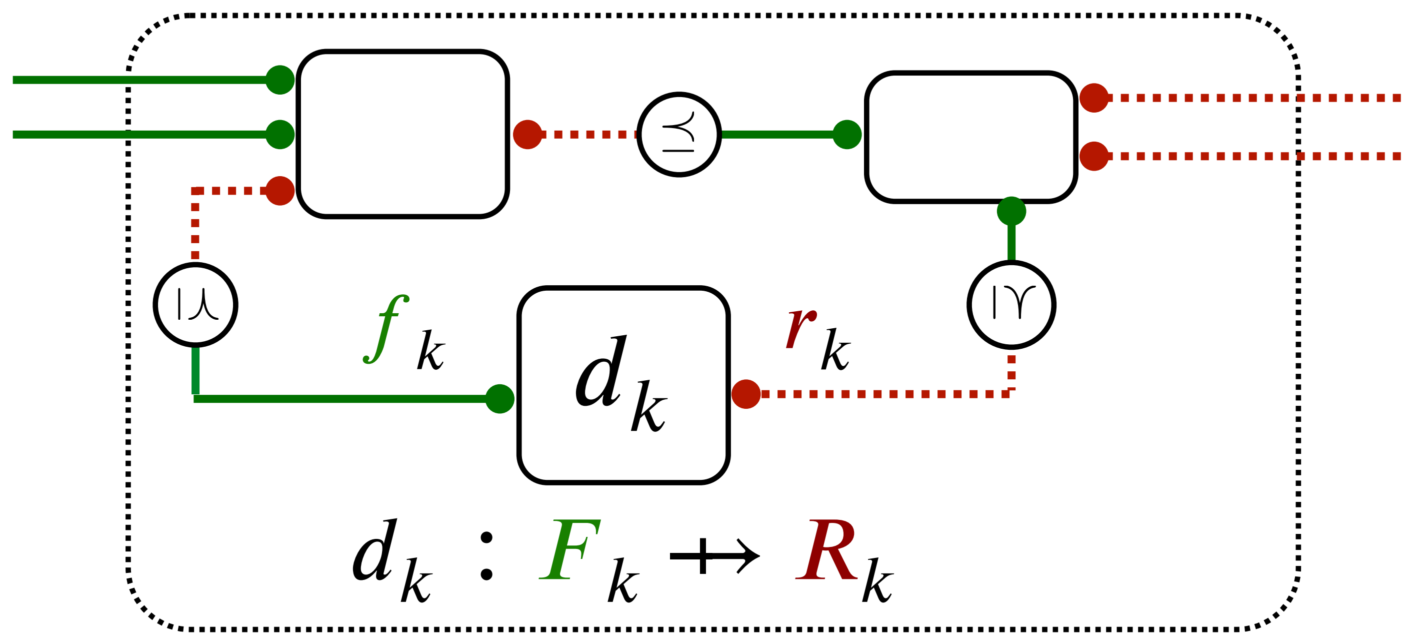
- We are looking for:
  - A map from functionality to upper sets of feasible resources; $h : F \to \mathbf{U}R$
  - A map from functionality to antichains of minimal resources. $h : F \to \mathcal{A}R$



Optimization semantics

- This is the semantics of **FixFunMinReq** as a **family of optimization problems.**

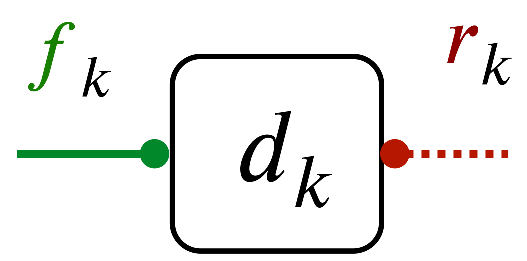
chosen by user $\overline{f}$

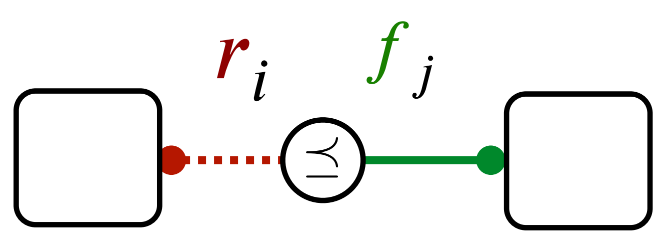


$\overline{r}$ **to minimize**

variables $r_k \in \langle R_k, \preceq_{R_k} \rangle$ $f_k \in \langle F_k, \preceq_{F_k} \rangle$

constraints ***for each node:*** ***for each edge:***





$d_k(f^*_k, r_k) = \top$ $r_i \preceq f_j$

objective $\underset{\preceq}{\mathrm{Min}}\, \overline{r}$

! not convex
! not differentiable
! not continuous
! not even defined on continuous spaces

How can category theory help?

- For engineering, having only a categorical model of the domain is of limited utility.
  How does it help solving a **real problem**™ ?

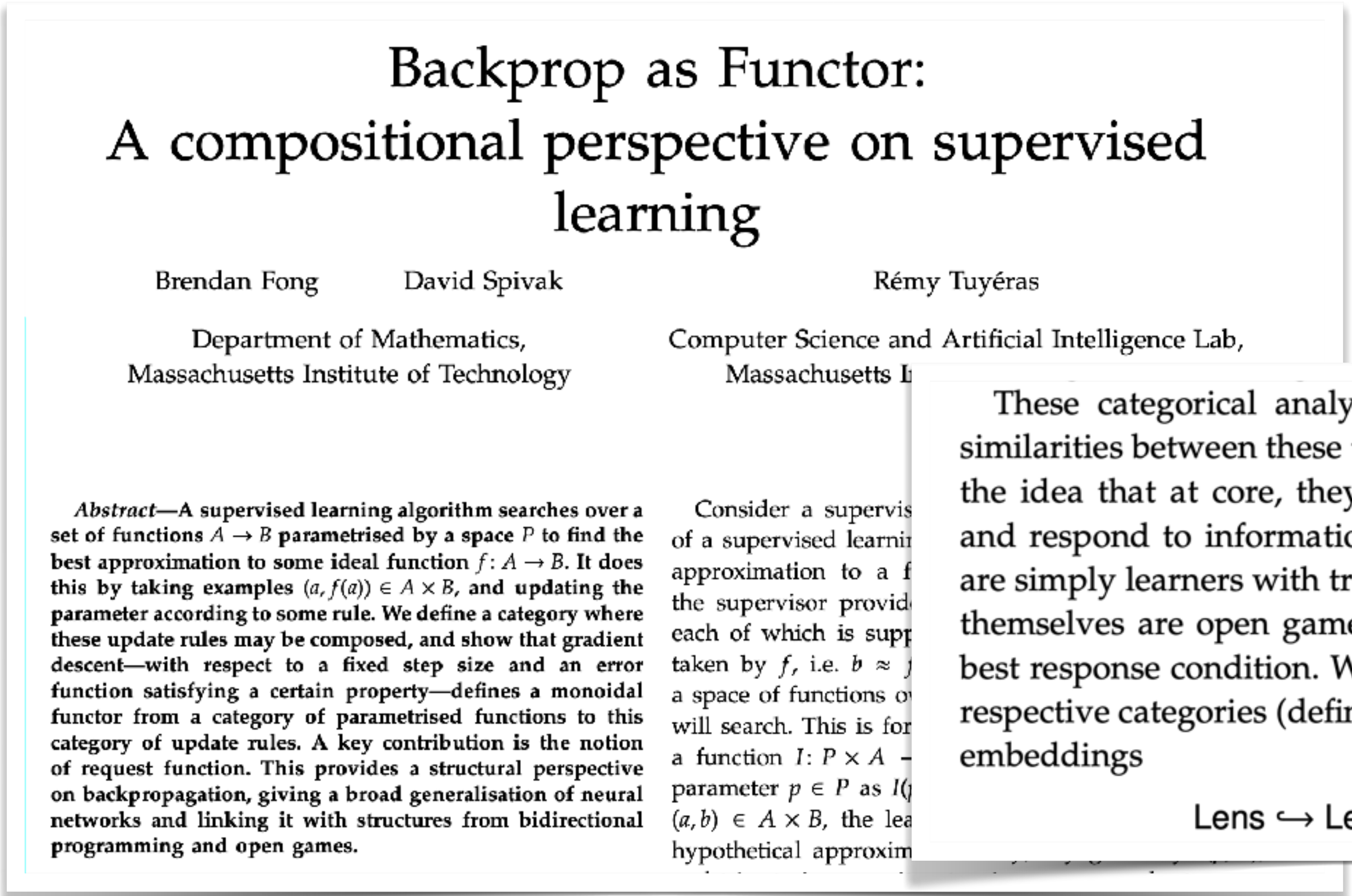
descriptive *vs* **actionable**

How can category theory help?

- For engineering, having only a categorical model of the domain is of limited utility.
  How does it help solving a **real problem**™ ?

descriptive *vs* **actionable**

- **Possible risk**: engineer **reading descriptive papers expecting actionable information**, gets disappointed, dismisses category theory as abstract nonsense.



Backprop as Functor:
A compositional perspective on supervised learning

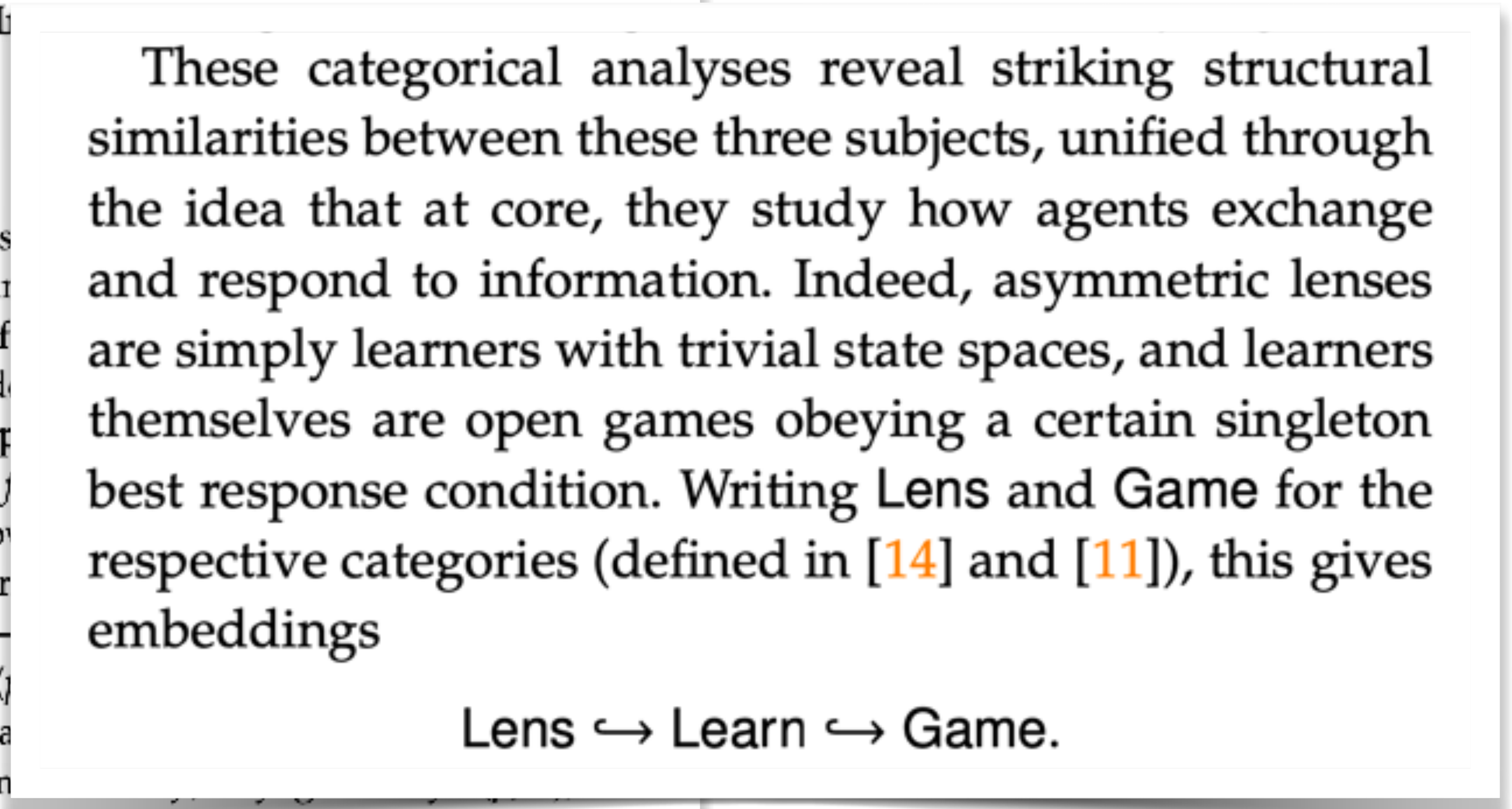
Brendan Fong David Spivak Rémy Tuyéras

Department of Mathematics, Massachusetts Institute of Technology

Computer Science and Artificial Intelligence Lab, Massachusetts I

Abstract—A supervised learning algorithm searches over a set of functions $A \to B$ parametrised by a space P to find the best approximation to some ideal function $f\colon A \to B$. It does this by taking examples $(a, f(a)) \in A \times B$, and updating the parameter according to some rule. We define a category where these update rules may be composed, and show that gradient descent—with respect to a fixed step size and an error function satisfying a certain property—defines a monoidal functor from a category of parametrised functions to this category of update rules. A key contribution is the notion of request function. This provides a structural perspective on backpropagation, giving a broad generalisation of neural networks and linking it with structures from bidirectional programming and open games.

Consider a supervis
of a supervised learni
approximation to a f
the supervisor provid
each of which is supp
taken by f, i.e. $b \approx f$
a space of functions o
will search. This is for
a function $I\colon P \times A$ –
parameter $p \in P$ as $I(p$
$(a, b) \in A \times B$, the lea
hypothetical approxim



These categorical analyses reveal striking structural similarities between these three subjects, unified through the idea that at core, they study how agents exchange and respond to information. Indeed, asymmetric lenses are simply learners with trivial state spaces, and learners themselves are open games obeying a certain singleton best response condition. Writing Lens and Game for the respective categories (defined in [14] and [11]), this gives embeddings

$$\mathsf{Lens} \hookrightarrow \mathsf{Learn} \hookrightarrow \mathsf{Game}.$$

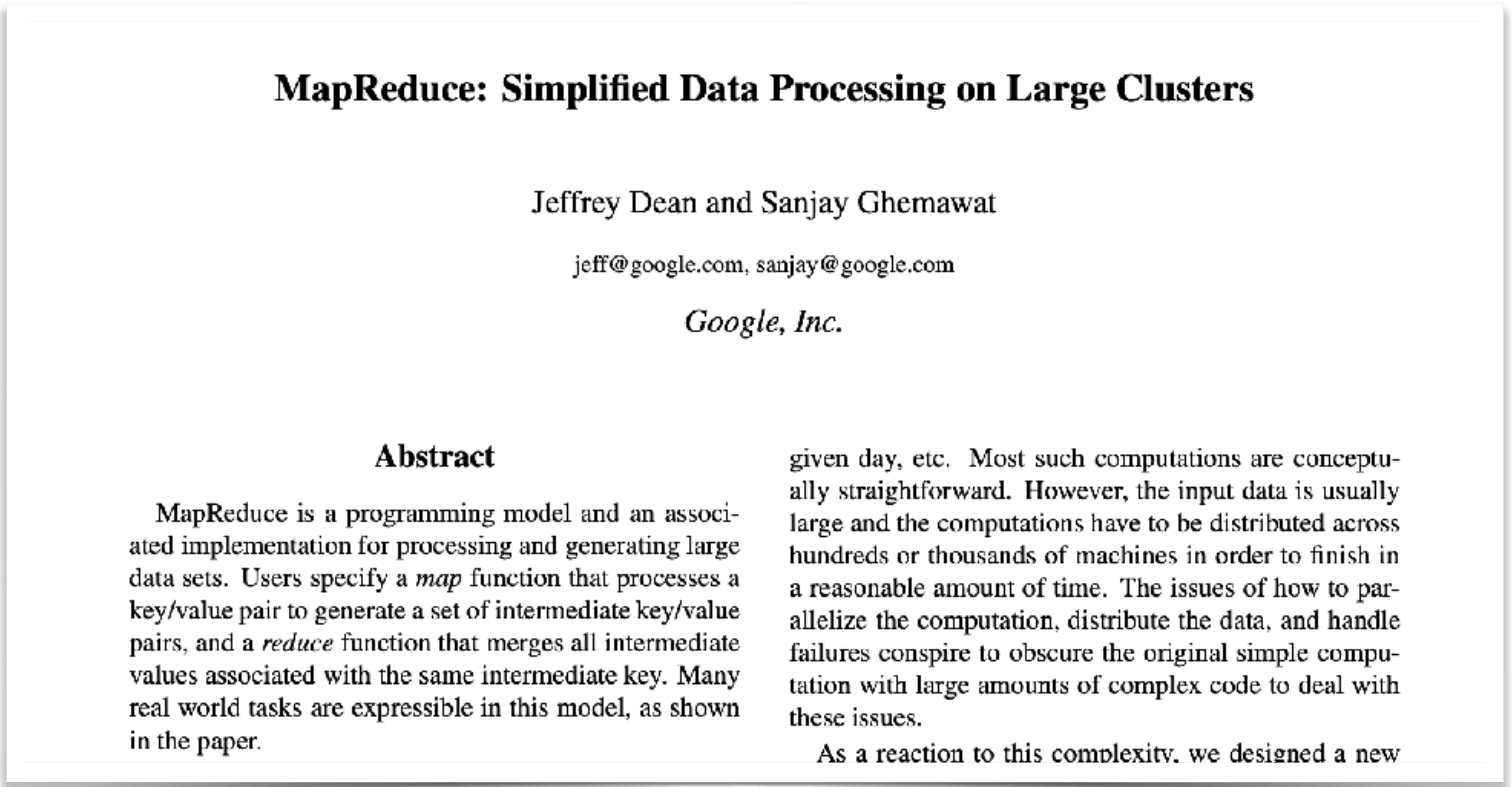
How can category theory help?

- For engineering, having only a categorical model of the domain is of limited utility.
  How does it help solving a **real problem**™ ?

descriptive *vs* **actionable**

- **Possible risk**: engineer **reading descriptive papers expecting actionable information**, gets disappointed, dismisses category theory as abstract nonsense.

- Perhaps, it is about **workflow systematization**.



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new

How can category theory help?

- For engineering, having only a categorical model of the domain is of limited utility.
  How does it help solving a **real problem**™ ?

descriptive *vs* **actionable**

- **Possible risk**: engineer **reading descriptive papers expecting actionable information**, gets disappointed, dismisses category theory as abstract nonsense.

- Perhaps, it is about **workflow systematization**.

- **My own experience:** CT helps greatly to define and implement solutions for "compositional domains" like co-design, computation graphs, etc.
  - Both **my papers and my code were much shorter**!

Looking for patterns

- We distinguish among:
  - **Models:** the data of the problem. Modeled as a category.
  - **Problem:** the type of question that we want to ask.
  - **Question:** an *instance* of the problem; to which we need to find an **Answer.**

	Models	**Problem**	**Questions**	**Answers**
Path planning	Cost-labeled graphs	minimum cost	a pair of nodes	a path in the graph + certificate of optimality; or a proof of infeasibility.
Co-design	co-design diagram	**FixFunMinReq**	functionality	upper set of resources

Looking for patterns

- We distinguish among:
  - **Models:** the data of the problem. Modeled as a category.
  - **Problem:** the type of question that we want to ask.
  - **Question:** an *instance* of the problem; to which we need to find an **Answer.**

$$\begin{tikzcd} M \arrow[r, "P"] & (Q \arrow[r] & A) \end{tikzcd}$$

	Models	Problem	Questions	Answers
	M	P	Q	A
Path planning	Cost-labeled graphs	minimum cost	a pair of nodes	a path in the graph + certificate of optimality; or a proof of infeasibility.
Co-design	co-design diagram	**FixFunMinReq**	functionality	upper set of resources

Looking for compositionality

- **Can we find a compositional structure**?
  - Models are morphisms in a category;
  - “Solvers” are morphisms in another category;
  - **Functoriality of *P***: if two models compose,
    you can find the solver by composing the solvers.

Models **Problem** **Questions** **Answers**

$$M \xrightarrow{\;P\;} \left(Q \longrightarrow A\right)$$

$$\mathbf{DP}(F;R) \xrightarrow{\mathbf{FixFunMinReq}} \left(F \xrightarrow[\mathbf{Pos}]{} \mathrm{U}R\right)$$

$$\mathbf{DP}(F;R) \xrightarrow{\mathbf{FixFunMinReq}} \left(F \xrightarrow[\mathbf{UPos}]{} R\right)$$

$$\mathbf{DP} \xrightarrow{\mathbf{FixFunMinReq}} \mathbf{UPos}$$

Kleisli category for the “upper set monad” U.

** technical assumptions?*

FixFunMinReq

Looking for compositionality

- **Can we find a compositional structure**?
  - Models are morphisms in a category;
  - “Solvers” are morphisms in another category;
  - **Functoriality of *P***: if two models compose,
    you can find the solver by composing the solvers.

$$\begin{tikzcd}
\textbf{Models} & \textbf{Problem} & \textbf{Questions} & \textbf{Answers} \\
\boldsymbol{M} \arrow[rr, "\boldsymbol{P}"] & & \big(\boldsymbol{Q} \arrow[r] & \boldsymbol{A}\big)
\end{tikzcd}$$

- Note: **Functoriality is very strong.**

$$\mathsf{Compile} : \text{syntactic units} \to \text{IR units}$$

$$\mathsf{Compile}(f_1 ⨾ f_2) = \mathsf{Compile}(f_1) ⨾ \mathsf{Compile}(f_2)$$

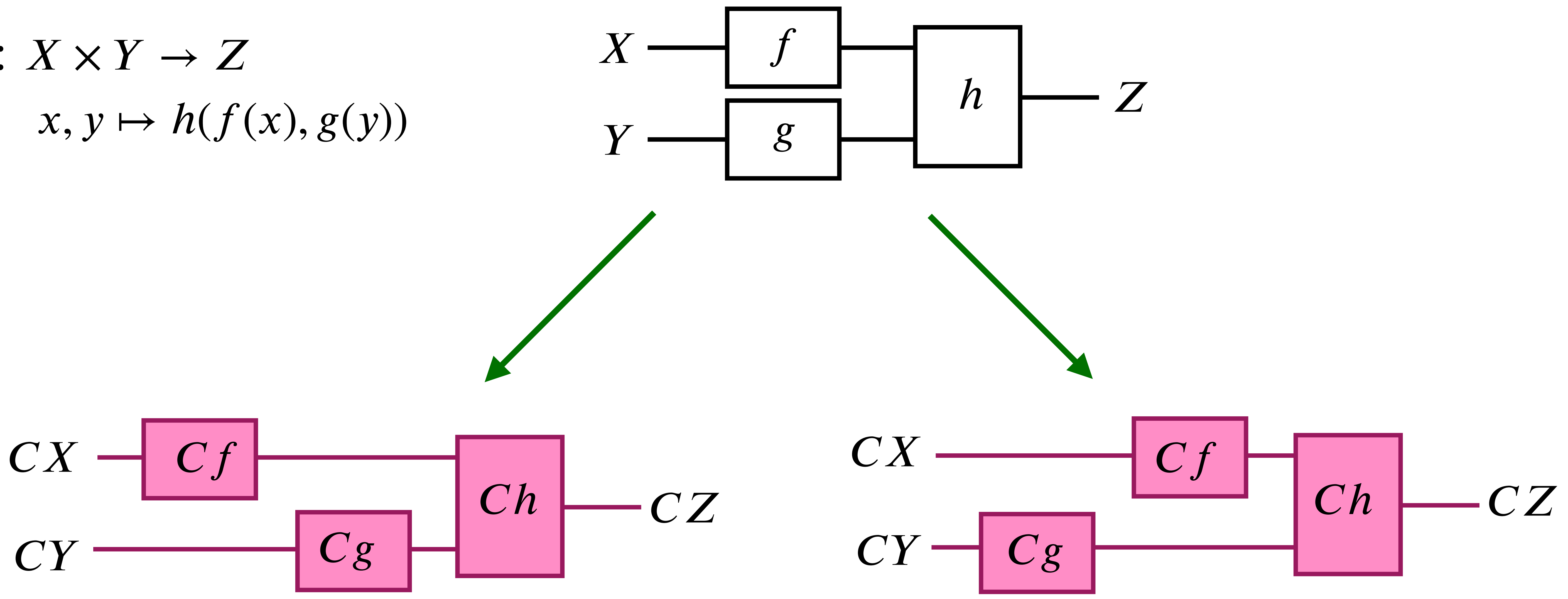
$$\mathsf{Compile}(f_1 ⨾ f_2) = \alpha(\mathsf{Compile}(f_1), \mathsf{Compile}(f_2))$$

Monoidal functoriality is very strong

- Translating from **Types** to **SerialPrograms**, a category of **serialized computation**.
  - Note: whether the compiler has any freedom of choice here depends on the semantics of your programming language.

$$(f \otimes g) \mathbin{⨟} h$$

$$\begin{aligned} \alpha : X \times Y &\to Z \\ x, y &\mapsto h(f(x), g(y)) \end{aligned}$$



$$(Cf \boxtimes \mathrm{Id}_{CY}) \mathbin{⨟} (\mathrm{Id}_{CX} \boxtimes Cg) \mathbin{⨟} Ch$$

$$(\mathrm{Id}_{CX} \boxtimes Cg) \mathbin{⨟} (Cf \boxtimes \mathrm{Id}_{CY}) \mathbin{⨟} Ch$$

SerialPrograms

"PreMonoidal" category allowing monoidal composition only with identities.

$$\frac{f : A \to B}{\mathrm{Id}_U \boxtimes f \boxtimes \mathrm{Id}_V : U \times A \times V \to U \times B \times V}$$

Enrichment for modeling performance

- If we use functors, each model is mapped to 1 solver, the only “right” one.
- Rather, there are typically **many solutions for each problem.**
- Solutions often have a **notion of “quality”** over which they can be ranked.
- **Profunctors / enriched categories** appear naturally in this context.

$$\begin{tikzcd}
\textbf{Models} & \textbf{Problem} & \textbf{Questions} & & \textbf{Answers} \\
M \arrow[rr, "\shortmid" marking, "P"] & & (Q \arrow[rr, "\shortmid" marking] & & A)
\end{tikzcd}$$

Enrichment for modeling performance

- If we use functors, each model is mapped to 1 solver, the only “right” one.
- Rather, there are typically **many solutions for each problem.**
- Solutions often have a **notion of “quality”** over which they can be ranked.
- **Profunctors / enriched categories** appear naturally in this context.

Models **Problem** **Questions** **Answers**

$$\begin{tikzcd} \boldsymbol{M} \arrow[r, "\boldsymbol{P}", "|" marking] & \big(\boldsymbol{Q} \arrow[r, "|" marking] & \boldsymbol{A}\big) \end{tikzcd}$$

- Still missing:
  - What would be a **computable** (finite) **representation** of the problem?
  - When do we start talking about **computational resources?** **“Procedures” *vs* functions.**

We cannot even represent all posets.

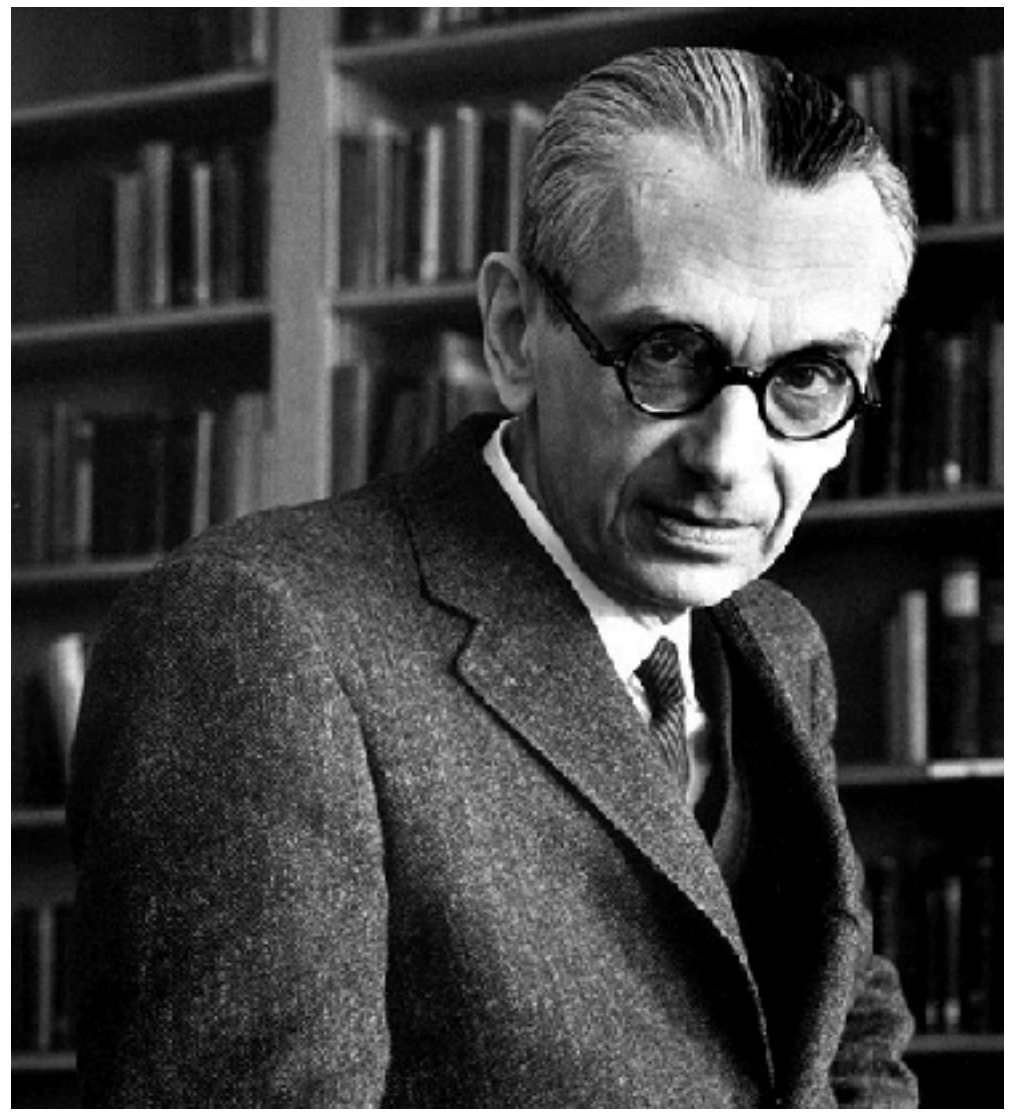
$$\begin{tikzcd} \mathbf{DP}(F; R) \arrow[r, "\mathbf{FixFunMinReq}"] & \big(F \arrow[r, "\mathbf{Pos}"'] & \mathrm{U}R \big) \end{tikzcd}$$

Multiple “procedures” “implement” the same function.

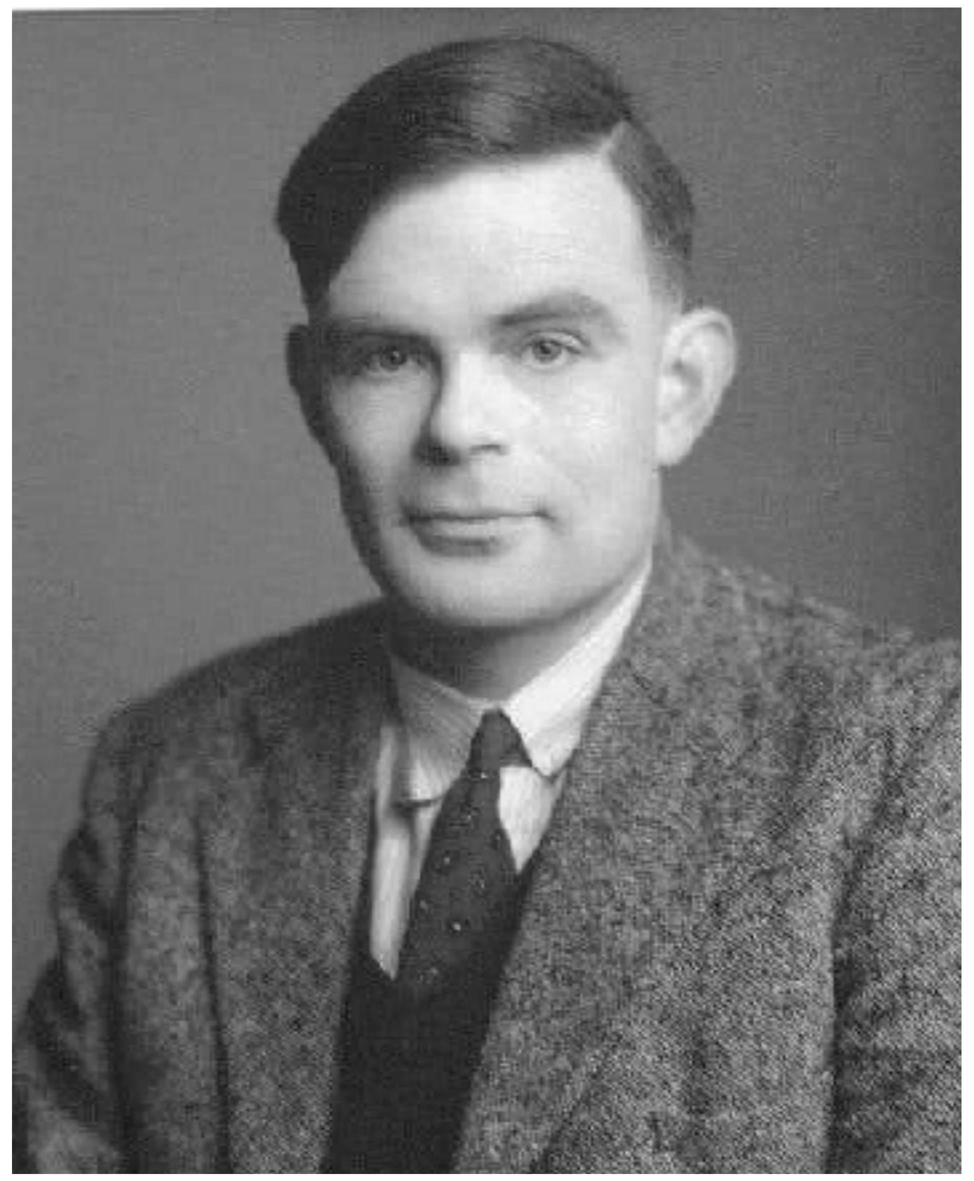
- Coming up: remarks about making mathematical problems computable.

- **Dr. Turing, please forgive some poetic license!**

Wenn das Mathematik ist, bin ich ein Strauß.



It’s cool, mate!



Imprecision is a sign of a weak mind.

From math to implementation

Mathematical phase

Prove the **problem is well posed** and that **a solution exists.**

Constructive phase

Define a **constructive method** to find the solution.

Algorithmic phase

Find an **effective** method for a specific **model of computation**.

Implementation

Implement on a specific machine, with limited resources.

From math to implementation

Mathematical phase

Prove the **problem is well posed** and that **a solution exists.**

For any vector v, $\exists n \colon \langle n, v \rangle = \|v\|$.

Constructive phase

Define a **constructive method** to find the solution.

$$n = \frac{v}{\|v\|}$$

Algorithmic phase

Find an **effective** method for a specific **model of computation**.

```
$M \leftarrow v_1^2 + v_2^2 + v_3^2$
$m \leftarrow \text{Newton}\left(a \mapsto 1/\sqrt{a}, M\right)$
return $\langle m v_1, m v_2, m v_3 \rangle$
```

Implementation

Implement on a specific machine, with limited resources.

Carmack's *Fast inverse square root*

```
float InvSqrt(float x){
    float xhalf = 0.5f * x;
    int i = *(int*)&x;            // store floating-point bits in integer
    i = 0x5f3759df - (i >> 1);    // initial guess for Newton's method
    x = *(float*)&i;              // convert new bits into float
    x = x*(1.5f - xhalf*x*x);     // One round of Newton's method
    return x;
}
```

From math to implementation

Mathematical phase

Prove the **problem is well posed** and that **a solution exists.**

Constructive phase

Define a **constructive method** to find the solution.

Algorithmic phase

Find an **effective** method for a specific **model of computation**.

Implementation

Implement on a specific machine, with limited resources.

“Secure one-way function:” whoever has the resources to find a collision would rather use garden-hose cryptanalysis.

Philosophical perspectives

⊢ Same thing if you are a **constructivist** ⊣

⊢ Same thing if you are a **finitist** ⊣

⊢ Same thing if you are an **ultra-finitist** ⊣

From math to implementation

Mathematical phase

Prove the **problem is well posed** and that **a solution exists.**

Constructive phase

Define a **constructive method** to find the solution.

Algorithmic phase

Find an **effective** method for a specific **model of computation**.

Implementation

Implement on a specific machine, with limited resources.

Philosophical perspectives

├── Same thing if you are a **constructivist** ──┤

├──────── Same thing if you are a **finitist** ────────┤

├──────── Same thing if you are an **ultra-finitist** ────────┤

Engineering syncretism:
I will believe in any philosophy or pantheon of deities,
if it helps me getting things done with less stress.

Solving co-design problems

Mathematical phase

Prove the **problem is well posed** and that **a solution exists.**

$$\begin{tikzcd}\mathbf{DP} \arrow[d, "\mathbf{FixFunMinReq}"] \\ \mathbf{UPos}\end{tikzcd}$$

$h : F \to \mathbf{U}R$

Constructive phase

Define a **constructive method** to find the solution.

$$\begin{tikzcd}\mathbf{DP}_{\text{comp}} \arrow[d, "\mathbf{FixFunMinReq}"] \\ \mathbf{UPos}_{\text{comp}}\end{tikzcd}$$

$h : F \to \mathbf{U}R$

1) Allow only posets P such that $\mathbf{U}P$ is a *direct complete partial order*.
2) Allow only DPs such that
   $h : F \to \mathbf{U}R$
   is a *Scott-Continuous* map.

Algorithmic phase

Find an **effective** method for a specific **model of computation**.

$$\begin{tikzcd}\mathbf{DP}_{\text{discrete}} \arrow[d, "\mathbf{FixFunMinReq}"] \\ \mathbf{UPos}_{\text{discrete}}\end{tikzcd}$$

$h : F \to \mathcal{A}R$

Allow only upper sets that can be represented as finite anti-chains.

$$S = \uparrow \operatorname{Min} S$$

Implementation

Implement on a specific machine, with limited resources.

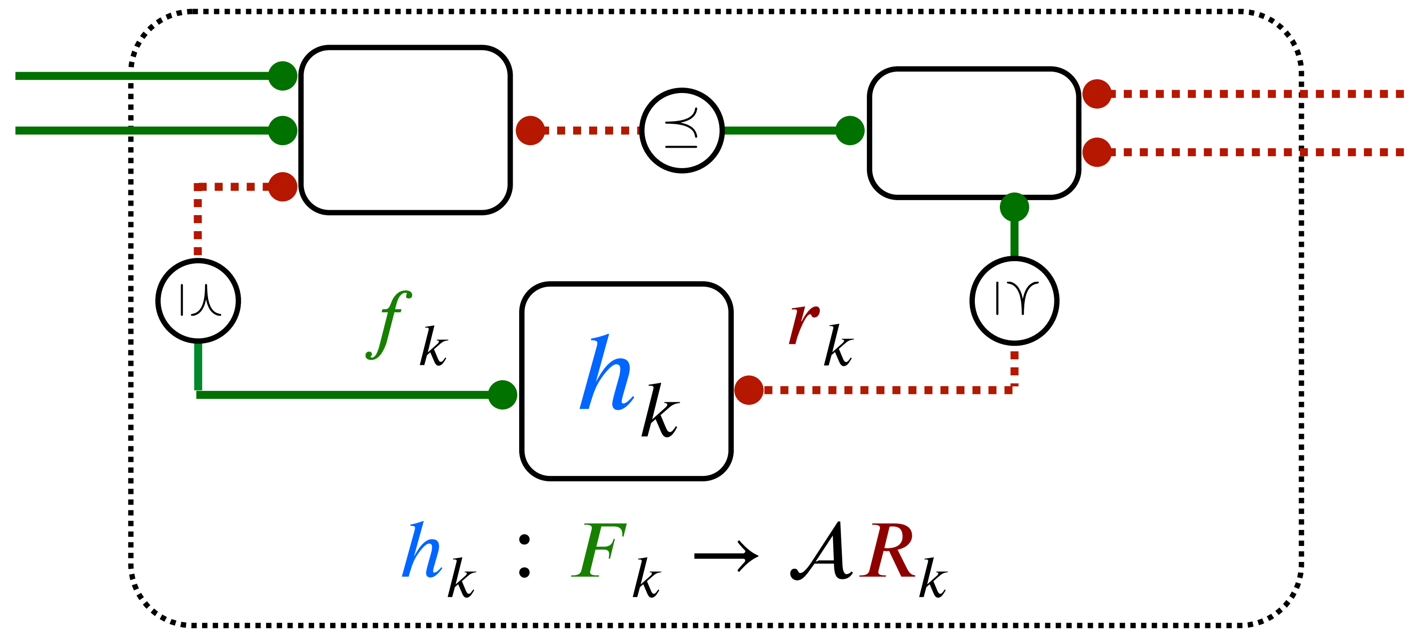
(In this context, DCPO and Scott Continuity are like compactness and Cauchy sequences for analysis: they ensure that some type of sequences will converge somewhere.)

sufficient condition:

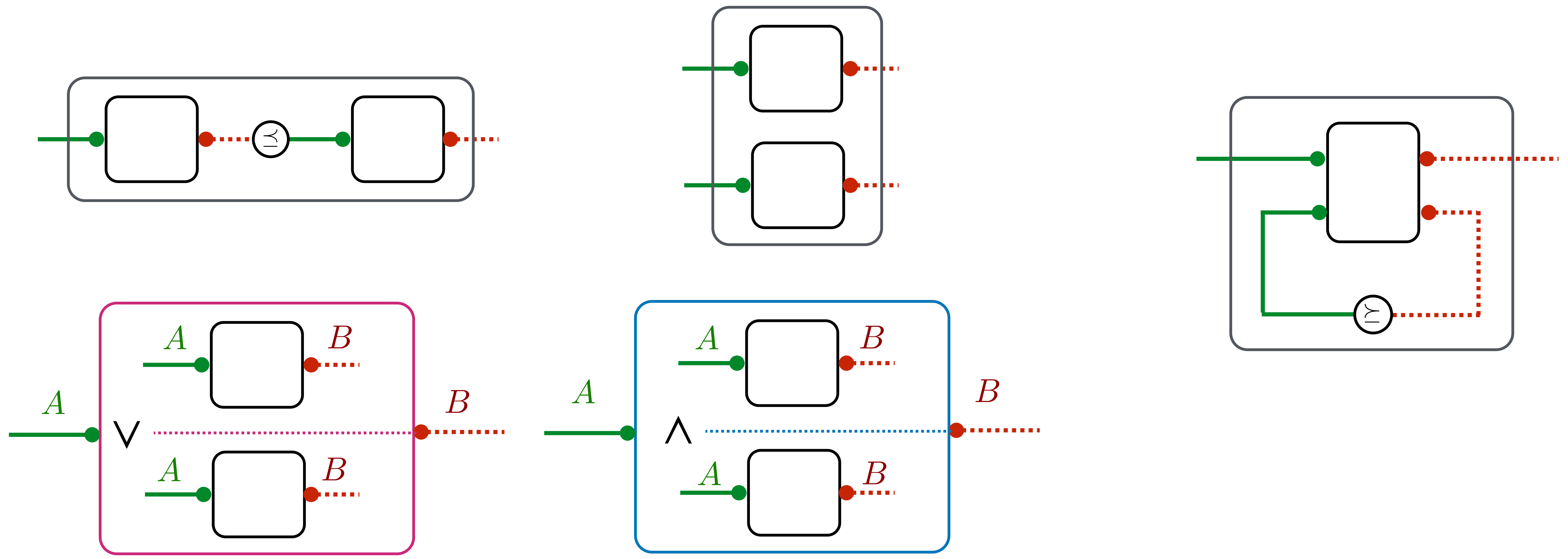
All posets are finite.

Solution formulas

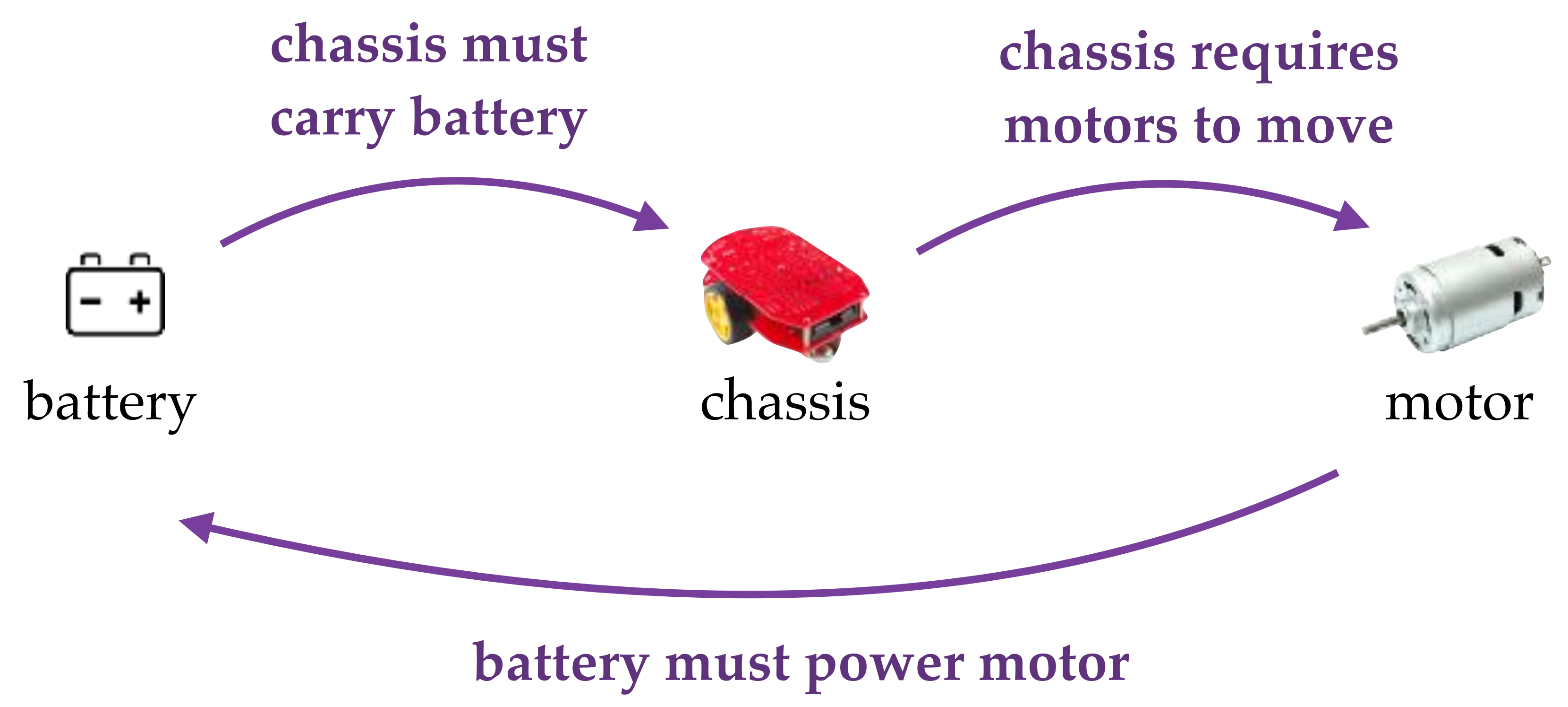
- Suppose we are given the function $h_k : F_k \to \mathcal{A}R_k$ for all nodes in the co-design graph.

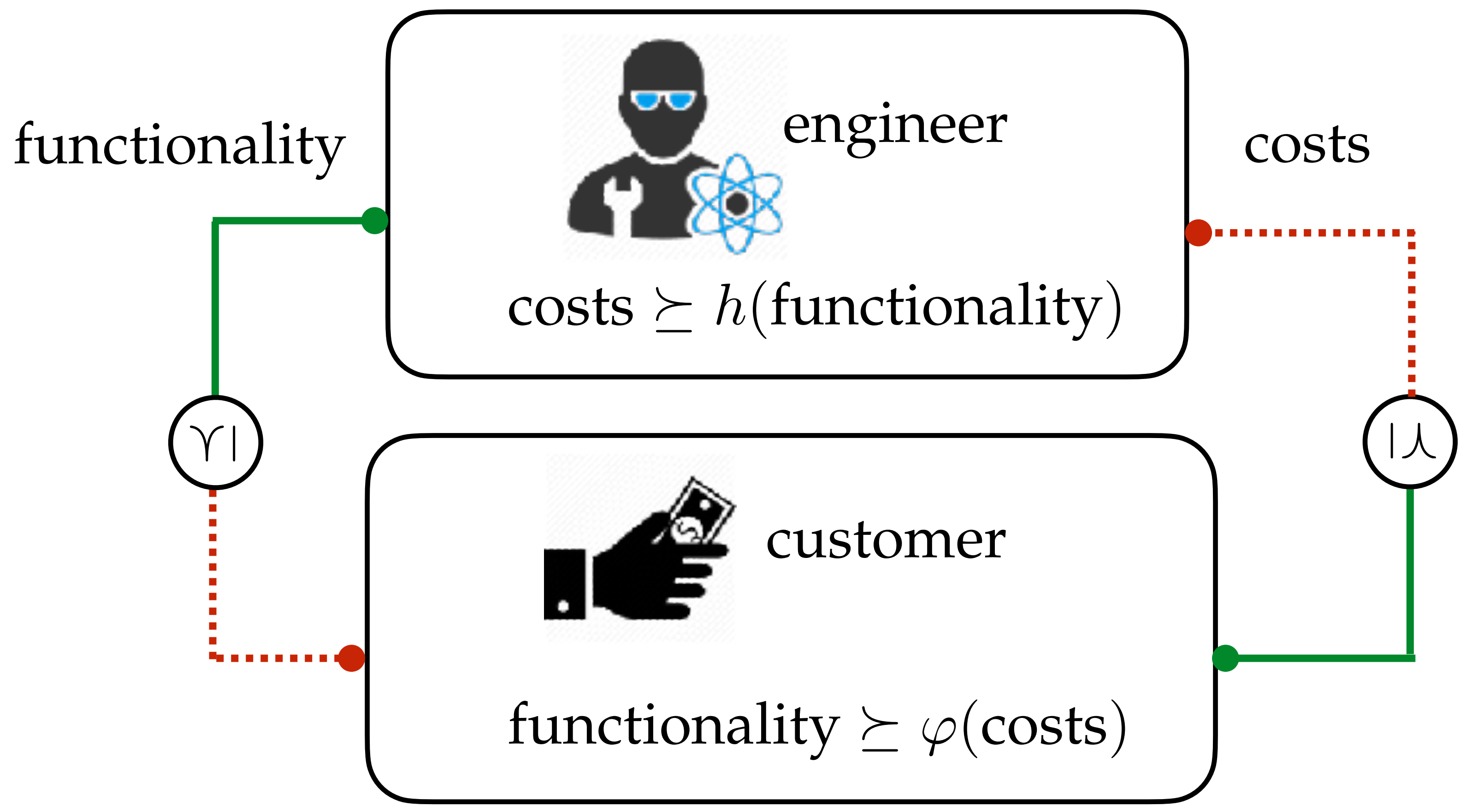


- Can we find the map $h : F \to \mathcal{A}R$ for the entire diagram?
- By induction, we just need to work out the the composition formulas for all operations we have defined.



What about loops?

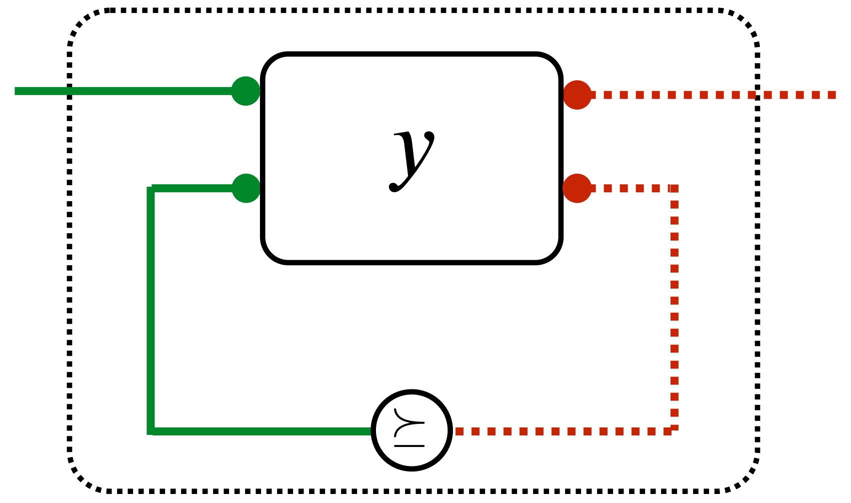



functionality
engineer
costs
costs $\succeq h(\text{functionality})$
customer
functionality $\succeq \varphi(\text{costs})$

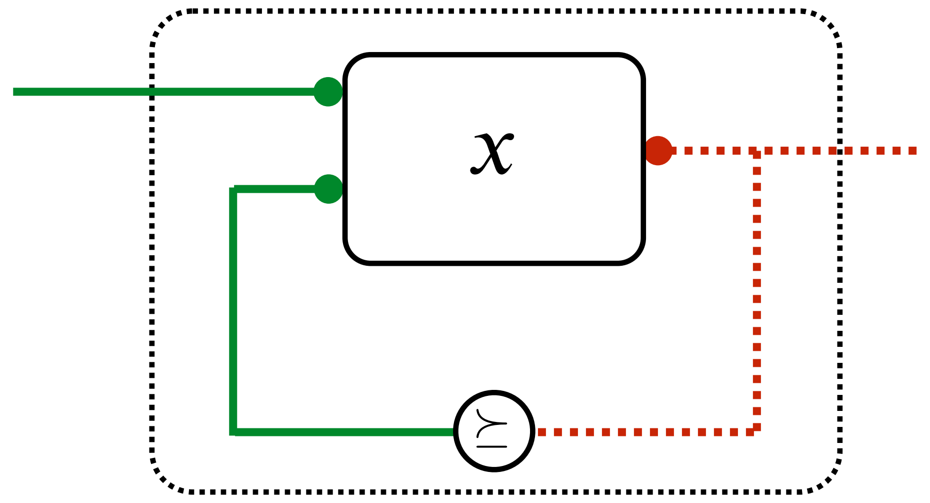
Trace *vs* Conway

- It is convenient to use the **Conway operator** rather than the Trace operator.
  - These are equivalent, in the sense that I can define Trace from Conway and vice-versa.

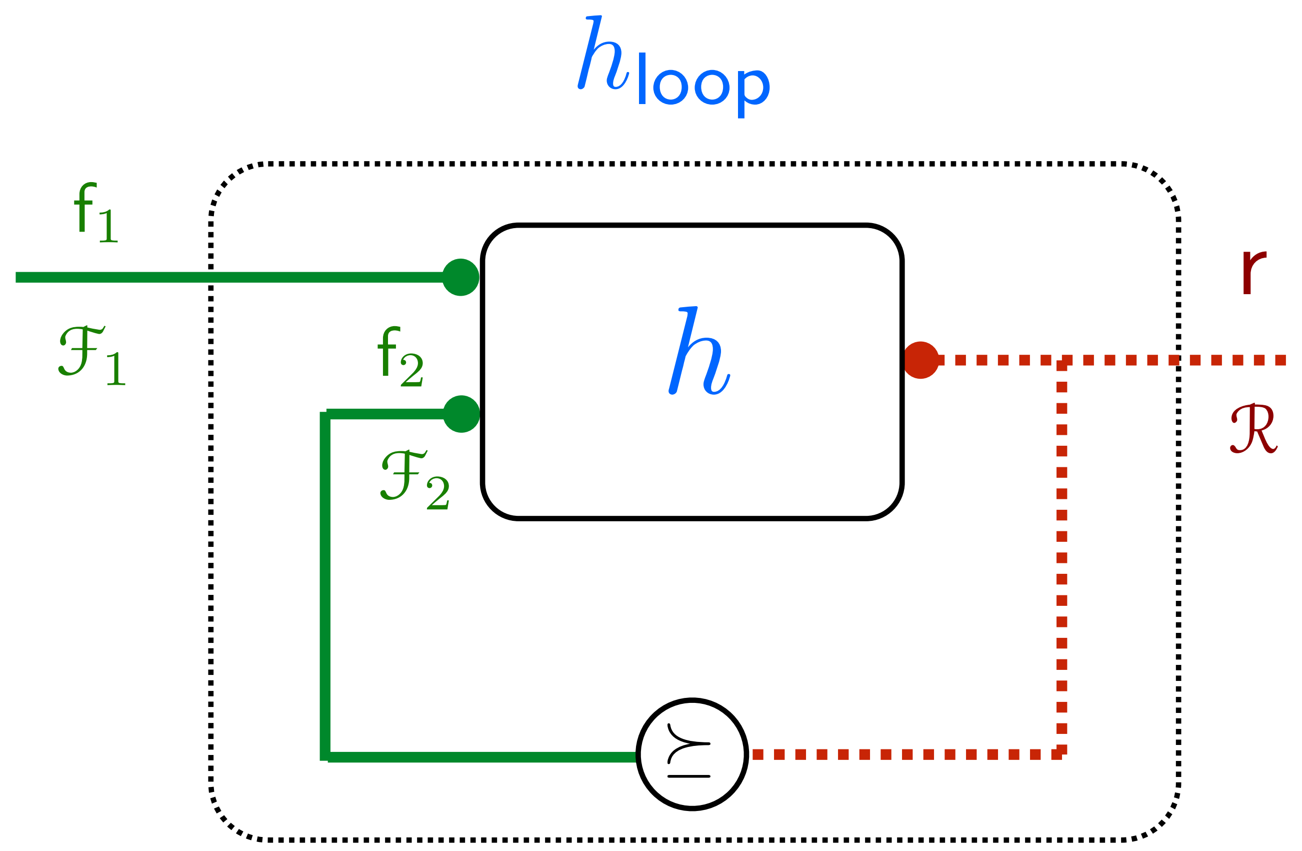
Trace(y)

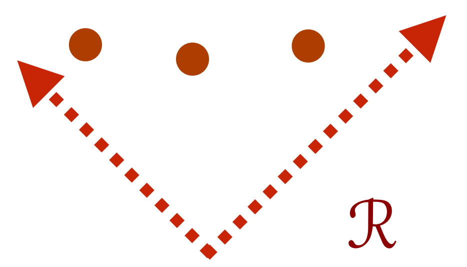


Conway(x)



Solution for Conway form



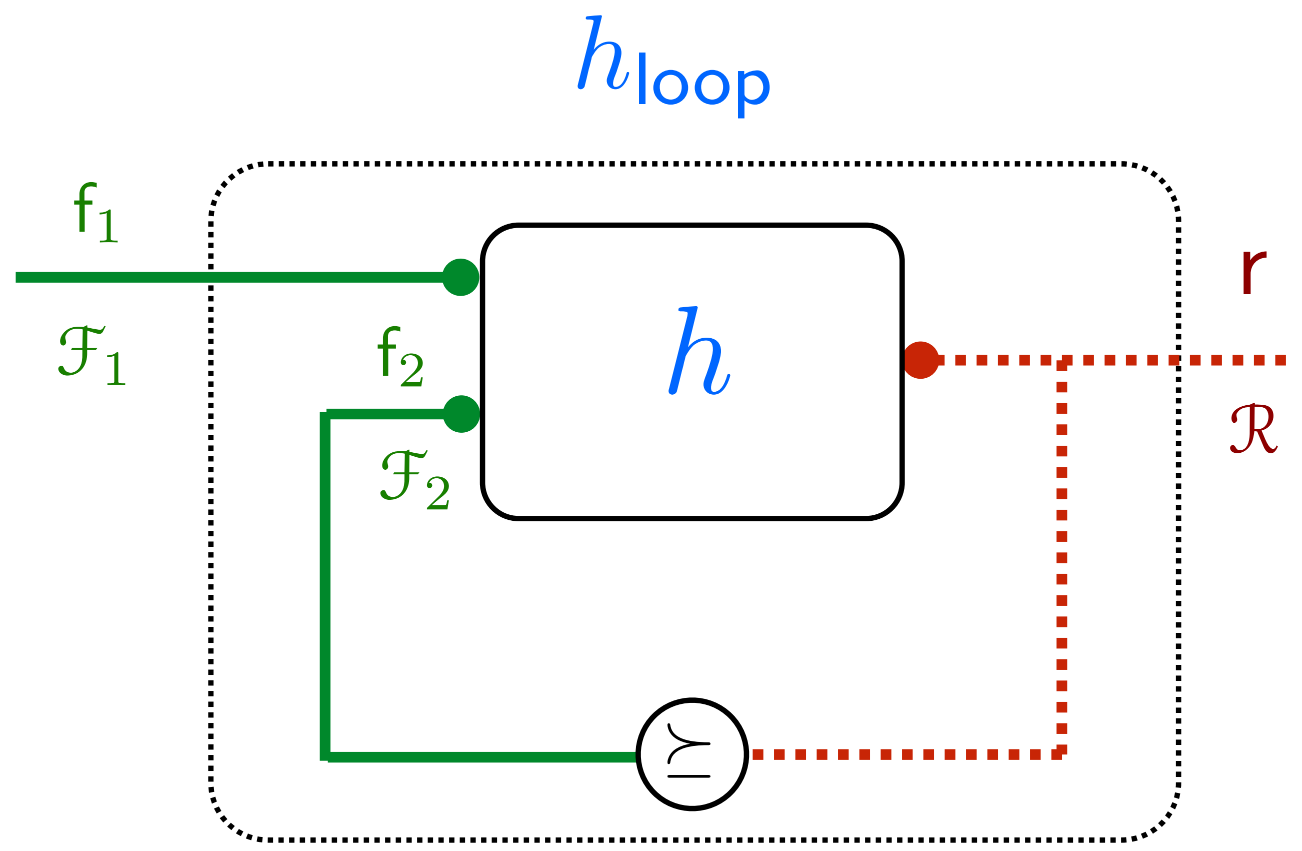


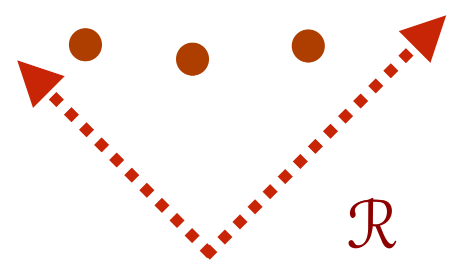
Theorem. The **set of minimal feasible resources** can be obtained as the least fixed point of a monotone function in the space of anti-chains.

$$\begin{aligned} h_{\text{loop}} : \mathcal{F}_1 &\to \text{antichains}(\mathcal{R}) \\ f_1 &\mapsto \overbrace{\text{least-fixed-point}(\Phi_{f_1})} \end{aligned}$$

$$\begin{aligned} \Phi_{f_1} : \text{antichains}(\mathcal{R}) &\to \text{antichains}(\mathcal{R}) \\ S &\mapsto \operatorname*{Min}_{\preceq_{\mathcal{R}}} \bigcup_{r \in S} h(f_1, r) \cap \uparrow r \end{aligned}$$

Solution for Conway form



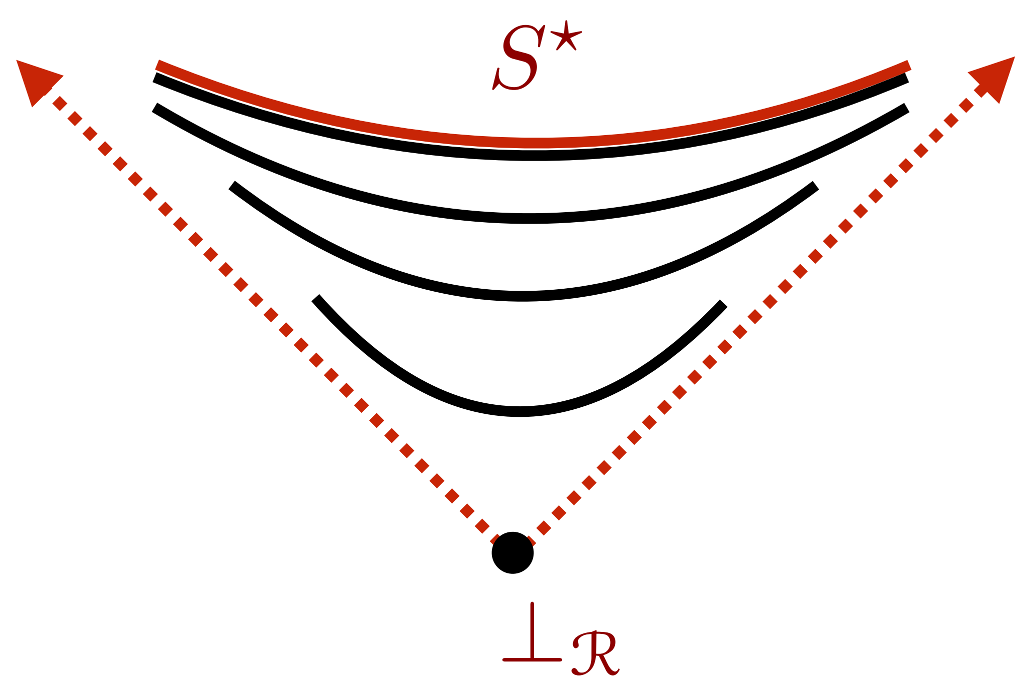


Corollary. The set of minimal solutions can be found using Kleene's algorithm.

$$S \subset \text{antichains}(\mathcal{R})$$

$$S_0 = \{\bot_{\mathcal{R}}\}$$

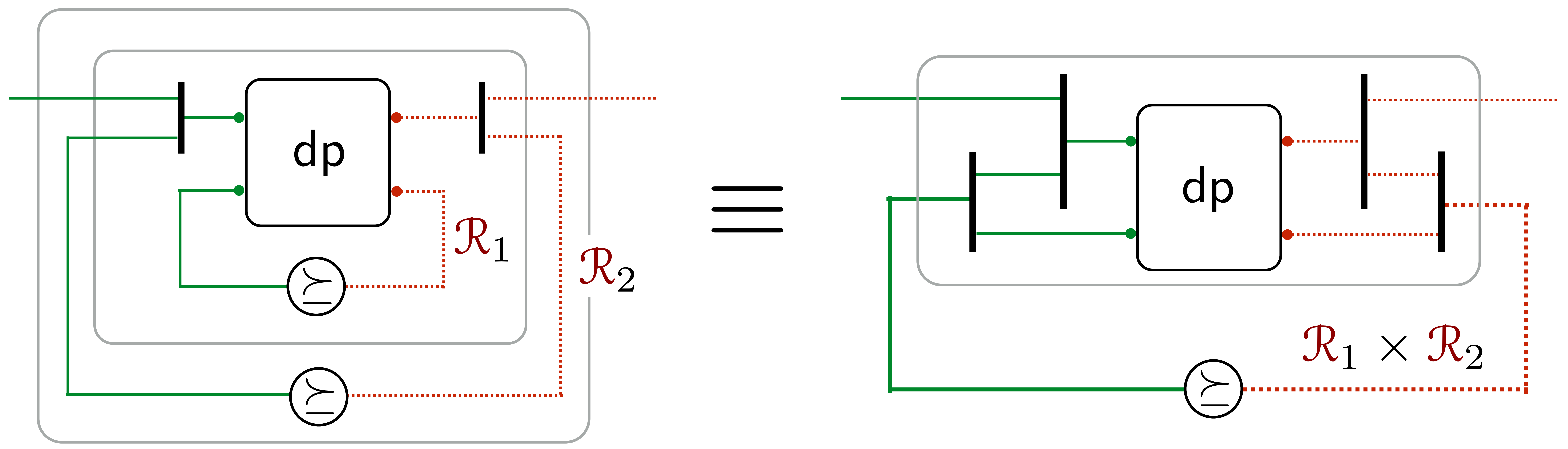
$$S_{k+1} = \Phi_{\mathsf{f}_1}(S_k)$$



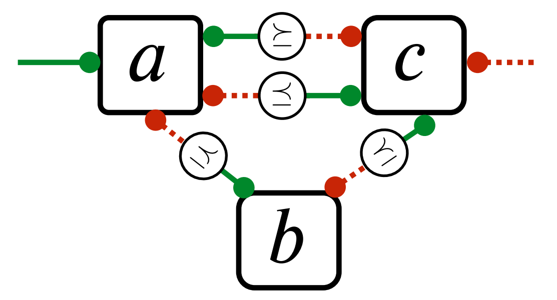
If the iteration diverges, it is a certificate of infeasibility.

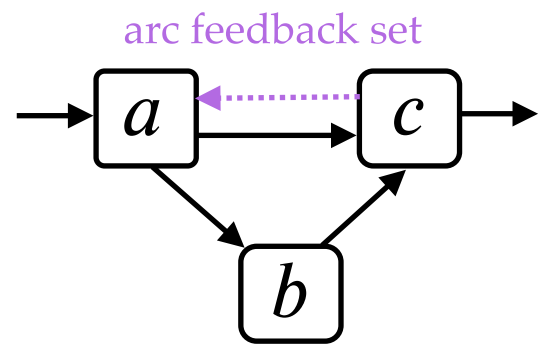
What about multiple loops?

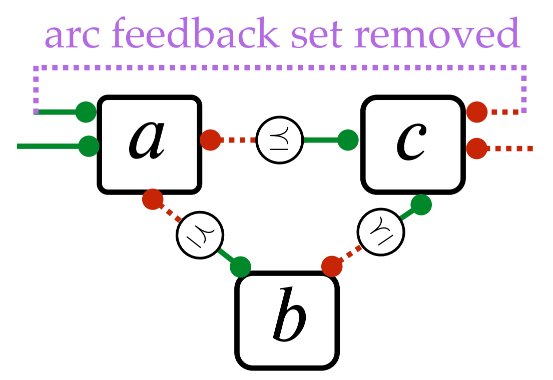
- Generally speaking, a fixed point iteration will **converge at the ω-th step**, where ω is the first infinite ordinal - a countable number of steps.
- But: if we close 2 loops, we need to compute a fixed point of a fixed point: this will take ω^2 steps.
- The properties of trace allows us to only reduce to 1 loop.

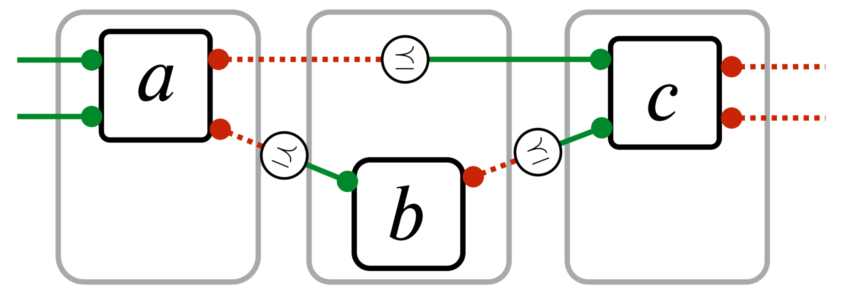


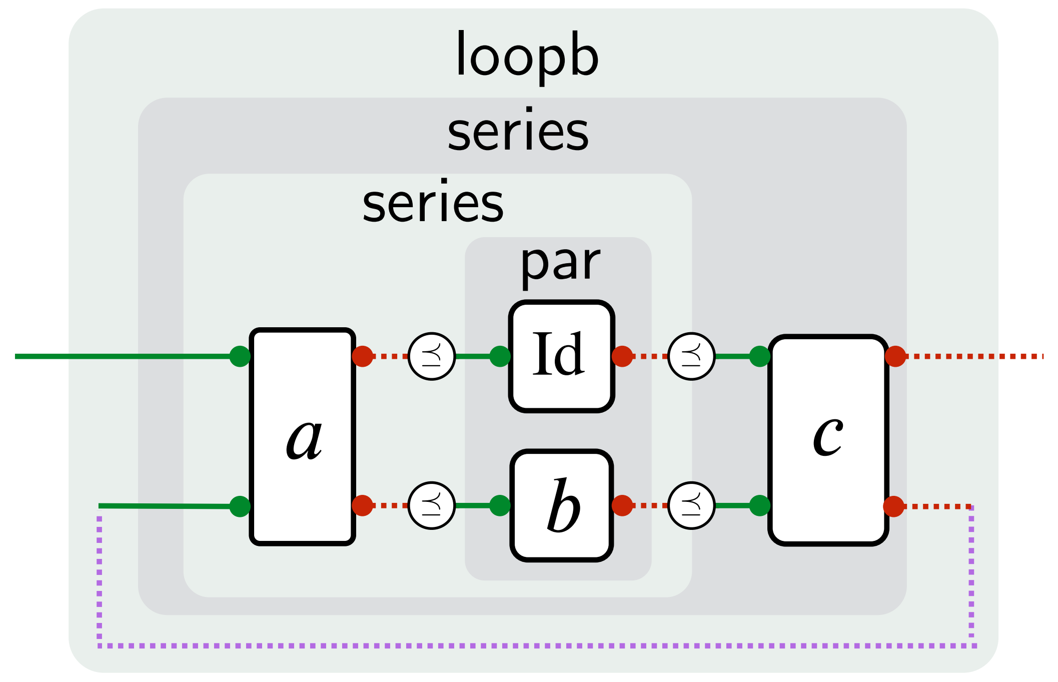
Reducing to a normal form







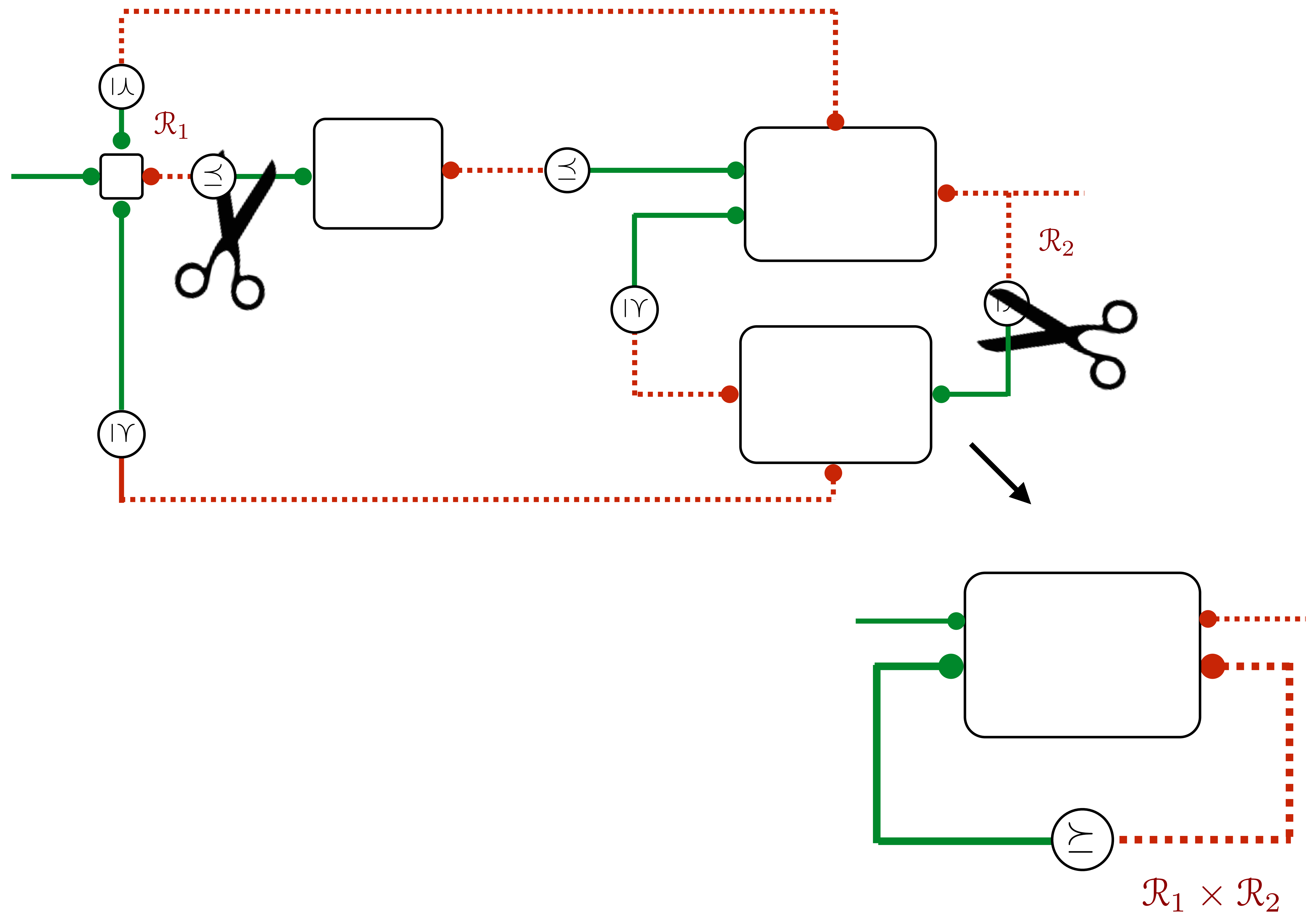




$$\begin{tikzcd}[column sep=small]
 & & \text{loopb} \arrow[d, dash] & \\
 & & \text{series} \arrow[dl, dash] \arrow[dr, dash] & \\
 & \text{series} \arrow[dl, dash] \arrow[dr, dash] & & c \\
a & & \text{par} \arrow[dl, dash] \arrow[dr, dash] & \\
 & b & & \text{Id}
\end{tikzcd}$$

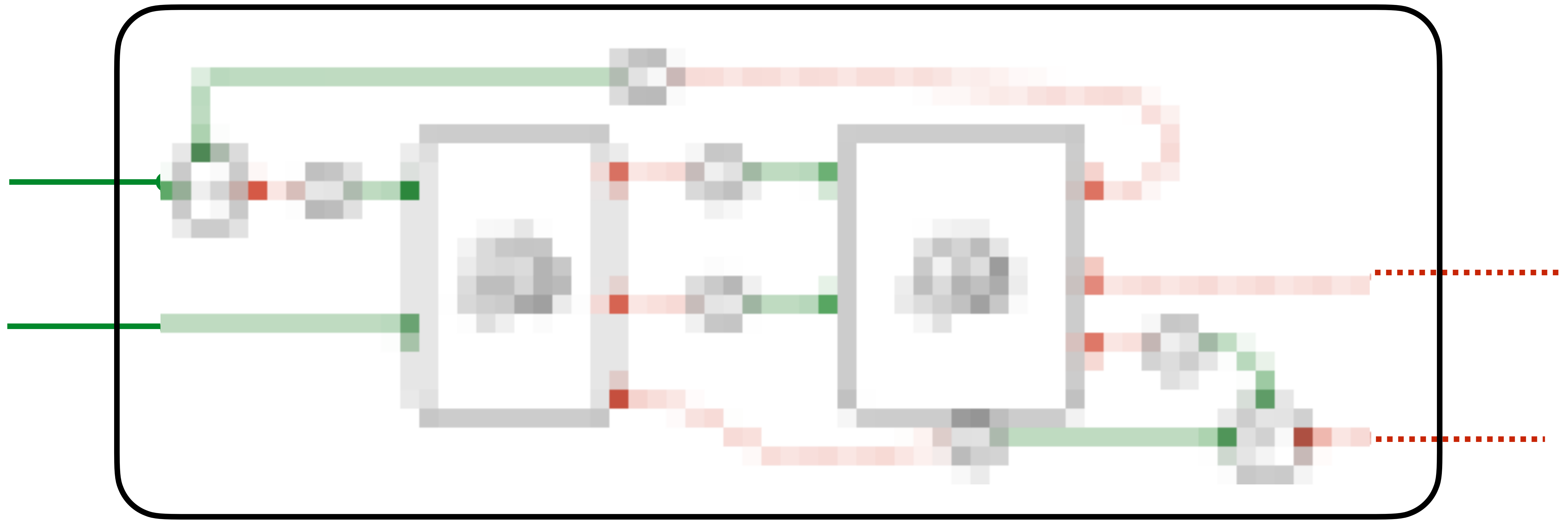
Complexity

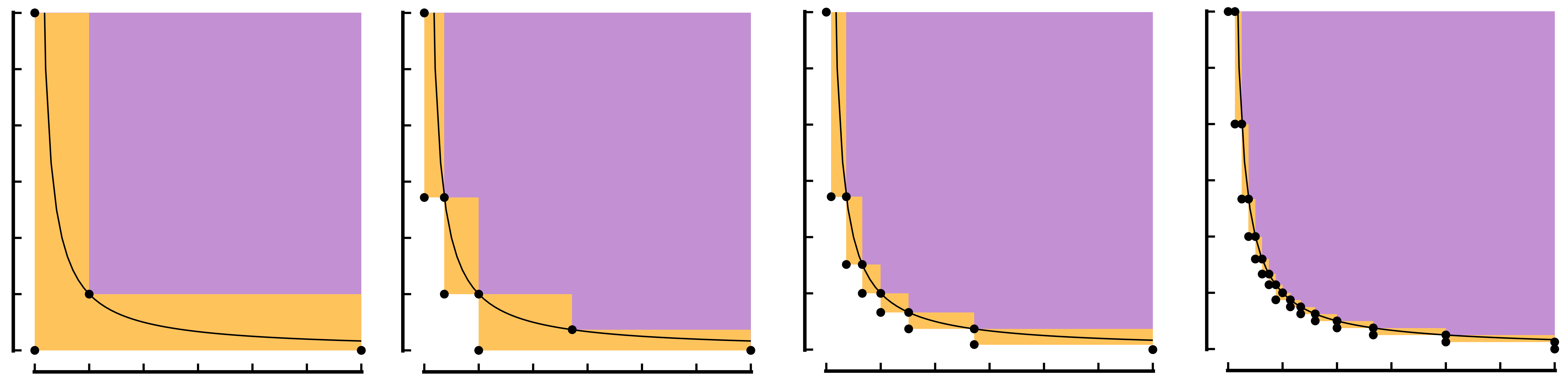
- The complexity of solving the problem depends on the "thickness" of the "minimal feedback arc set" cut to create the normal form.
  - *<u>Not combinatorial</u>* in the size of the implementations!



Bounded approximations

- We can obtain bounded approximation with converging sequences considering **a category of DP intervals** (twisted arrow category).





Conclusions

- A **useful direction for applied category theory** is looking at **modeling problems**, rather than just modeling the structure of the domain.

$$\begin{tikzcd} \textbf{Models} & \textbf{Problem} & \textbf{Questions} & \textbf{Answers} \\ M \arrow[rr, "P", "|" marking] & & (Q \arrow[r, "|" marking] & A) \end{tikzcd}$$

- It seems that **many synthesis problems have a compositional structure**: models and *solvers* are categories, linked by a functor-like ***P*** arrow.
- **Enriched categories** may help modeling **performance levels** and **resources usage.**
- **Monads, operads, etc.** and other more advanced topics that we never mentioned start to shine at this level of abstraction.