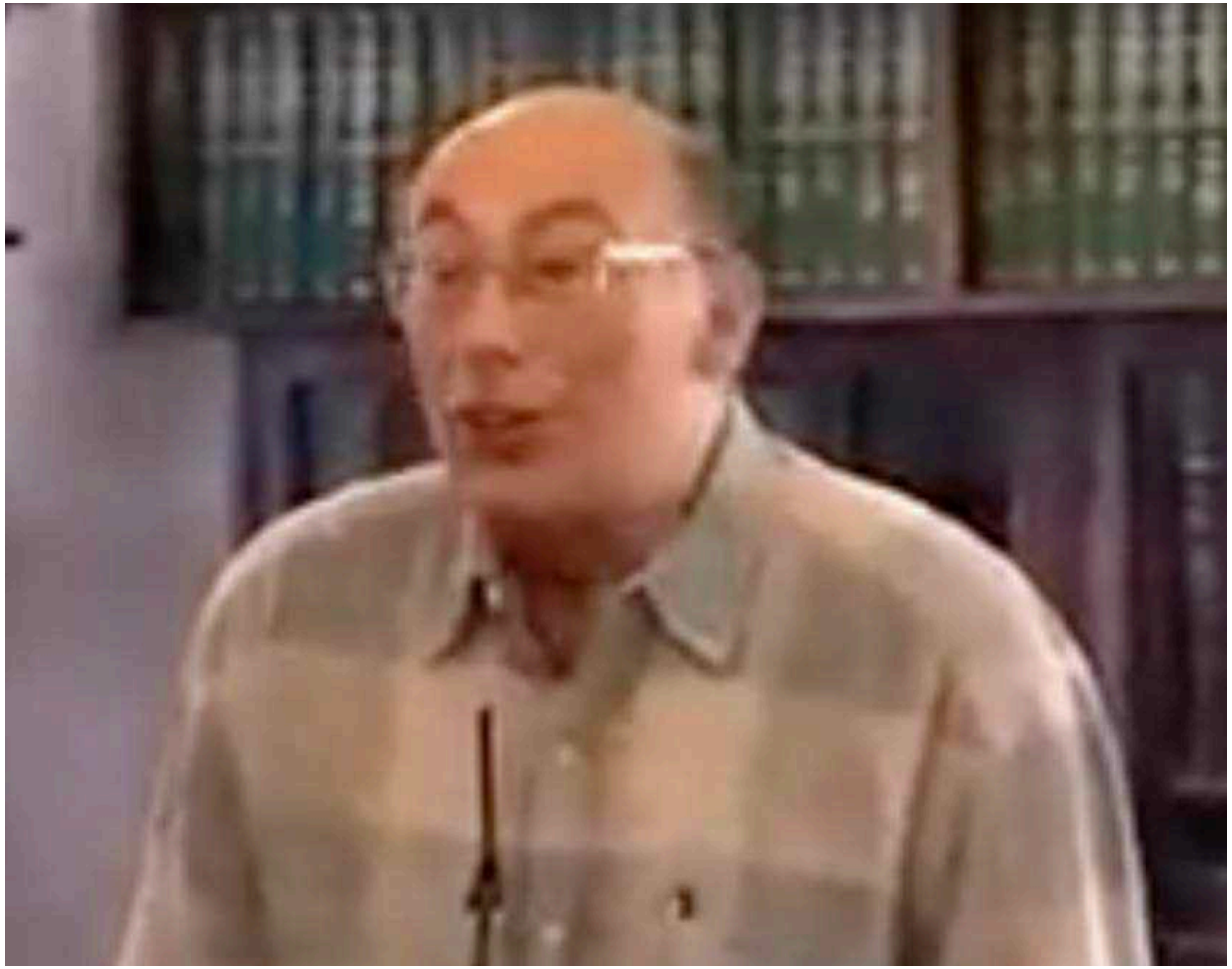
Peter Landin Semantics Seminar

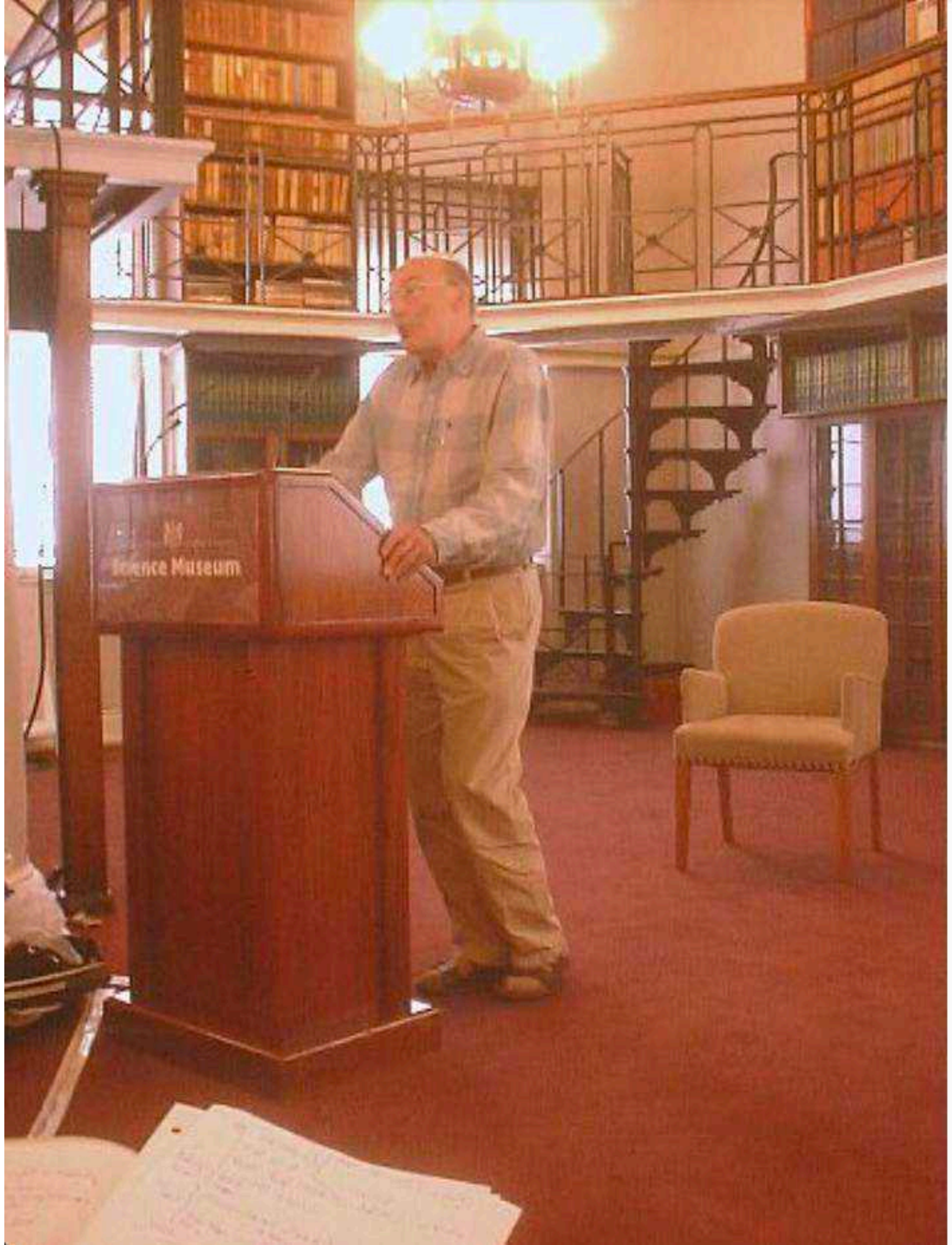
Edmund Robinson
Queen Mary University of London

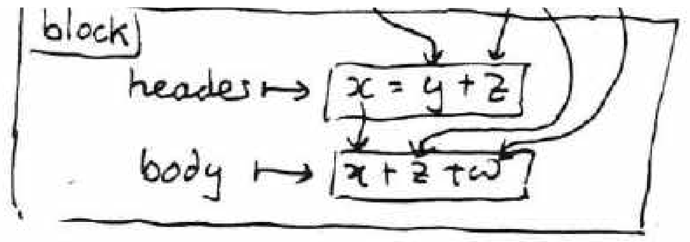
Logical Relations and Mathematical Foundations


Computer Science
Queen Mary
Peter Landin
Building
lime


Computer Science
Queen Mary
Peter Landin Building




Science Museum



[p]ointing to each demand for a name (aka, in Watt, "applied occurrence"), there is an arrow from the occurrence that supplies that demand, or else from outside the phrase.

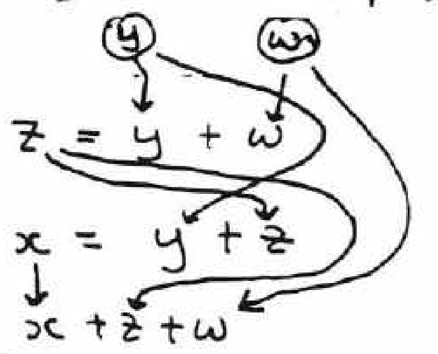
Example 80. Block Demand/Supply Analysis (omitting '+' because it's obvious):—

	set demanded	set supplied
header	$\{y, z\}$	$\{x\}$
body	$\{x, z, w\}$	$\{\}$
whole block	$\{y, z\} \cup \{x, z, w\}/\{x\} = \{y, z, w\}$	$\{\}$

For two (or more) declarations/definitions written one after another, it is natural to suppose that a later one may depend on (refer to, demand from, be affected by, be dependent on, be supplied by) the previous ones; and that their accumulated effect is supplied to the phrase that follows them. The notation of "supply/demand arrows", introduced in Fig 80 above, is a clumsy but picturesque, and precise, concrete syntax for indicating which occurrences are supplied from where. It will be used to explain current languages, and also to explain the four "plugging configurations" that were listed on p. 62, and have not yet been described.

Fig 81:

Example 81



Example 82 In Modula:—

```
VAR
  z: INTEGER := y+w;
  x: INTEGER := y+z;
BEGIN
  .. x+z+w...
END
```

Example 83 (or) if the types are right—

```
VAR
  z := y+w; x := y+z;
BEGIN ... x+z+w...END
```

Example 84 In C:—

```
{ int z = y+w, x = y+z;
  ... x+z+w... }
```

Example 85 In ML:—

```
let val z = y+w
in let val x = y+z
   in x+z+w
```

(or)

Example 86

```
let val z = y+w; val x = y+z
in x+z+w
```

In passing, note the various concrete syntaxes for header/body structure of a block. BUT, details of concrete syntax are BORING, trivial, IRRELEVANT, compared with questions about whether or not some intended mea[ning] ...

Ex 10.11

Assuming an implemented language in which the following phrase is a definition, give an experiment-by-context intended to settle whether it is a block or not (i.e. whether the first defined name in each case is hidden or supplied)

```
let (stopper: type) = [77..87]
in ( let (listt: type) = stop of stopper
                        | go_on of (number * listt))
```

Ex 10.12

Assuming an implemented language in which the following phrase is a definition, give an experiment-by-context intended to settle whether it is a block or not (i.e. whether the first defined name in each case is hidden or supplied)

```
let a = 11
in (let b = a + 66)
```

Ex 10.13

Assuming an implemented language in which the following phrase is a definition, give an experiment-by-context intended to settle whether it is a block or not (i.e. whether the first defined name in each case is hidden or supplied)

```
let f(x) = x + 11
in (let g(y) = f(y) + 77)
```

Ex 10.14

Assuming an implemented language in which the following phrase is a definition, give an experiment-by-context intended to settle whether it is a block or not (i.e. whether the first defined name in each case is hidden or supplied)

```
let pair = number * number
in (let treee = nil | (treee * pair * treee))
```

Performance

- Peter was very interested in describing what programs do.

Change in Semantics

- Move from proving programs “correct” in some absolute sense
- To providing tools to improve quality
  - and those tools have to fit in with the development chain

Formal models

- But you still have to produce a formal model

Mathematics

- The language we use when we want to do calculations about a system
- But the calculations are never about the actual system
- They are about models of the system

What happens if we use different models: do we get the same results?

Logical Relations

Two key messages

- Basic ideas are quite simple, and if you focus, then you can keep them like that.
- We can use them to justify (in fact derive) some standard notions of process equivalence.

Logical Relations

- Robert Milne: thesis - proving equivalence of implementations
- Mike Gordon: unpublished discussions
- Gordon Plotkin:

SCHOOL OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

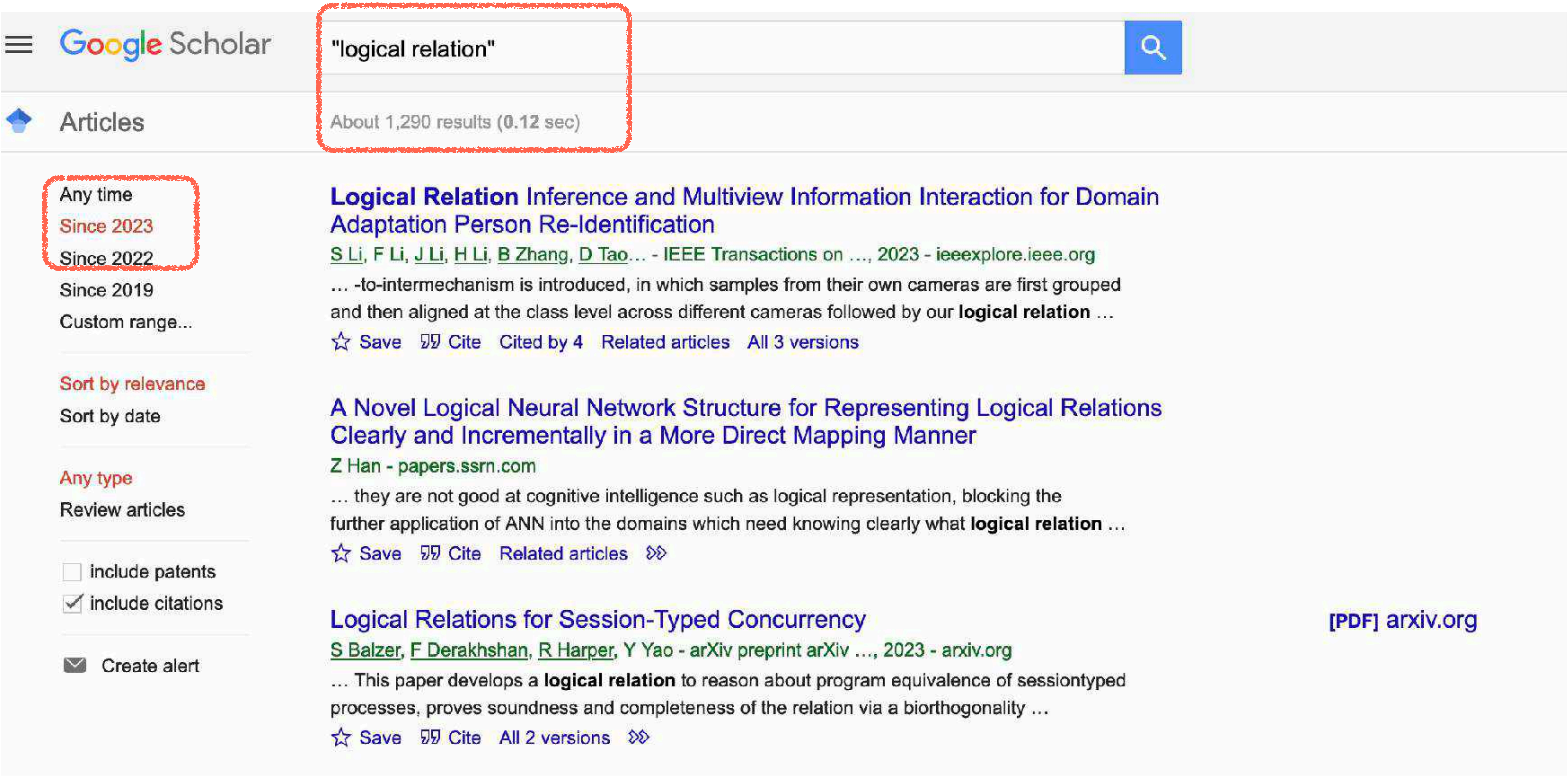
Memorandum: SAI-RM-4

Date:- October,1973

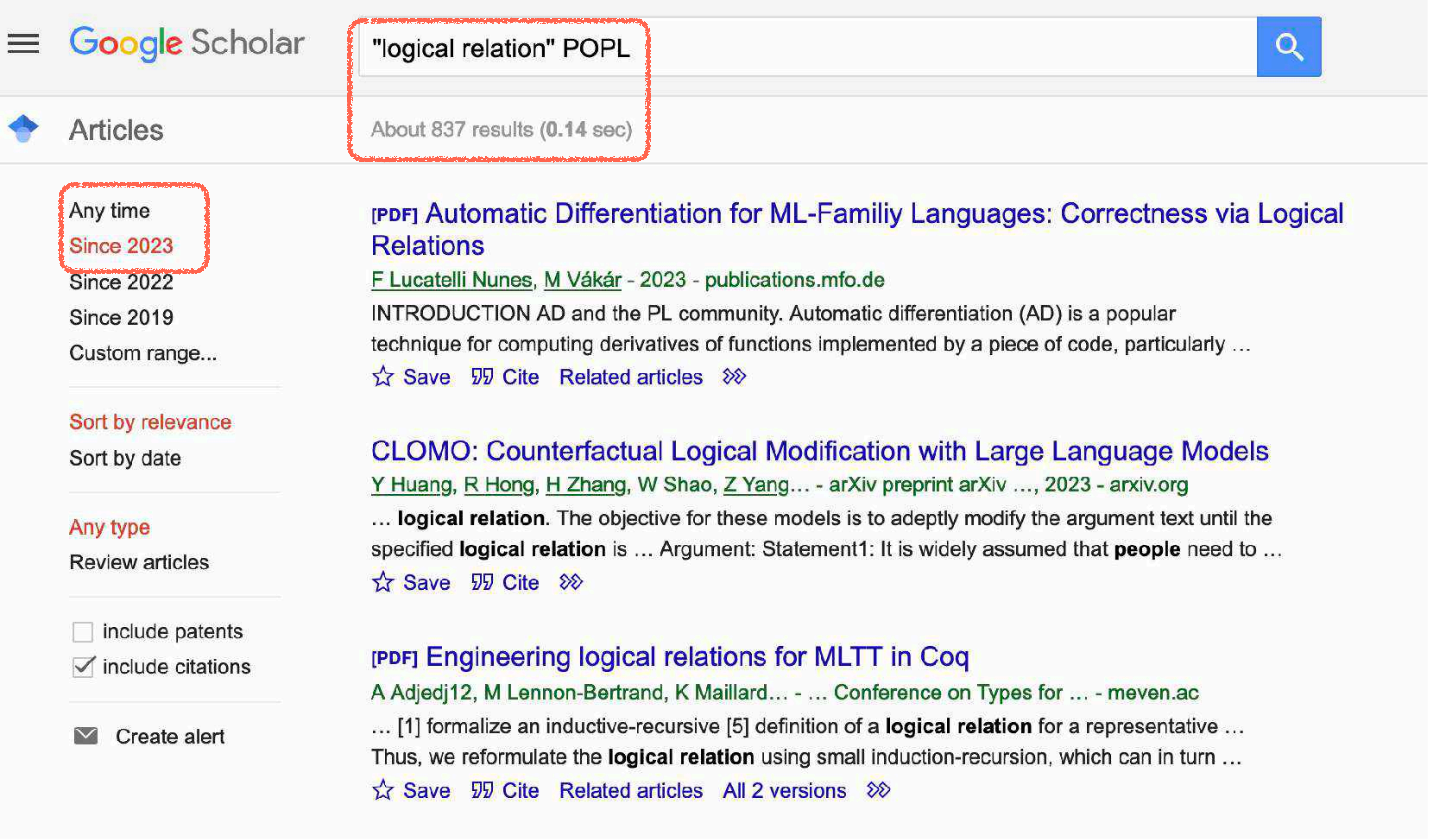
Subject: Lambda-definability and logical relations

Author: G.D. Plotkin

The main method will be to construct certain, so-called, logical relations which are satisfied by all (constant vectors of) λ-definable elements and yet are not satisfied by the lattice-theoretic entity under discussion. The definition of logical is derived from a corresponding one of M. Gordon for the typed λ-calculus. This in turn generalised the idea of an invariant functional [2]. R. Milne [3] has independently developed analogues of the logical relations for use in equivalence proofs about programming languages.

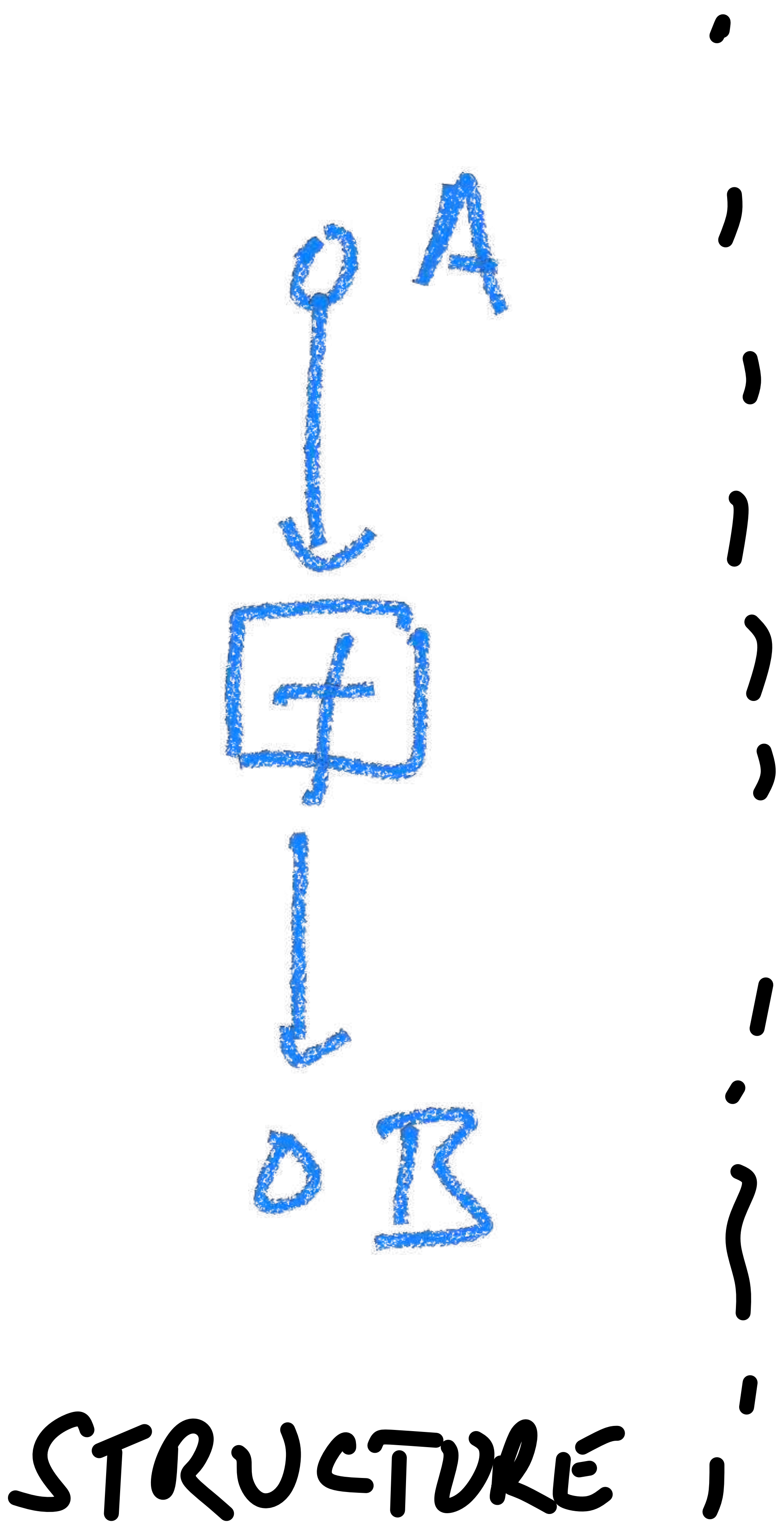


Google Scholar
"logical relation"
Articles
About 1,290 results (0.12 sec)
Any time
Since 2023
Since 2022
Since 2019
Custom range...
Sort by relevance
Sort by date
Any type
Review articles
include patents
include citations
Create alert
Logical Relation Inference and Multiview Information Interaction for Domain Adaptation Person Re-Identification
S Li, F Li, J Li, H Li, B Zhang, D Tao… - IEEE Transactions on …, 2023 - ieeexplore.ieee.org
… -to-intermechanism is introduced, in which samples from their own cameras are first grouped and then aligned at the class level across different cameras followed by our logical relation …
Save Cite Cited by 4 Related articles All 3 versions
A Novel Logical Neural Network Structure for Representing Logical Relations Clearly and Incrementally in a More Direct Mapping Manner
Z Han - papers.ssrn.com
… they are not good at cognitive intelligence such as logical representation, blocking the further application of ANN into the domains which need knowing clearly what logical relation …
Save Cite Related articles
Logical Relations for Session-Typed Concurrency
[PDF] arxiv.org
S Balzer, F Derakhshan, R Harper, Y Yao - arXiv preprint arXiv …, 2023 - arxiv.org
… This paper develops a logical relation to reason about program equivalence of sessiontyped processes, proves soundness and completeness of the relation via a biorthogonality …
Save Cite All 2 versions


Google Scholar
"logical relation" POPL
Articles
About 837 results (0.14 sec)
Any time
Since 2023
Since 2022
Since 2019
Custom range...
Sort by relevance
Sort by date
Any type
Review articles
include patents
include citations
Create alert
[PDF] Automatic Differentiation for ML-Familiy Languages: Correctness via Logical Relations
F Lucatelli Nunes, M Vákár - 2023 - publications.mfo.de
INTRODUCTION AD and the PL community. Automatic differentiation (AD) is a popular technique for computing derivatives of functions implemented by a piece of code, particularly …
Save Cite Related articles
CLOMO: Counterfactual Logical Modification with Large Language Models
Y Huang, R Hong, H Zhang, W Shao, Z Yang… - arXiv preprint arXiv …, 2023 - arxiv.org
… logical relation. The objective for these models is to adeptly modify the argument text until the specified logical relation is … Argument: Statement1: It is widely assumed that people need to …
Save Cite
[PDF] Engineering logical relations for MLTT in Coq
A Adjedj12, M Lennon-Bertrand, K Maillard… - … Conference on Types for … - meven.ac
… [1] formalize an inductive-recursive [5] definition of a logical relation for a representative … Thus, we reformulate the logical relation using small induction-recursion, which can in turn …
Save Cite Related articles All 2 versions

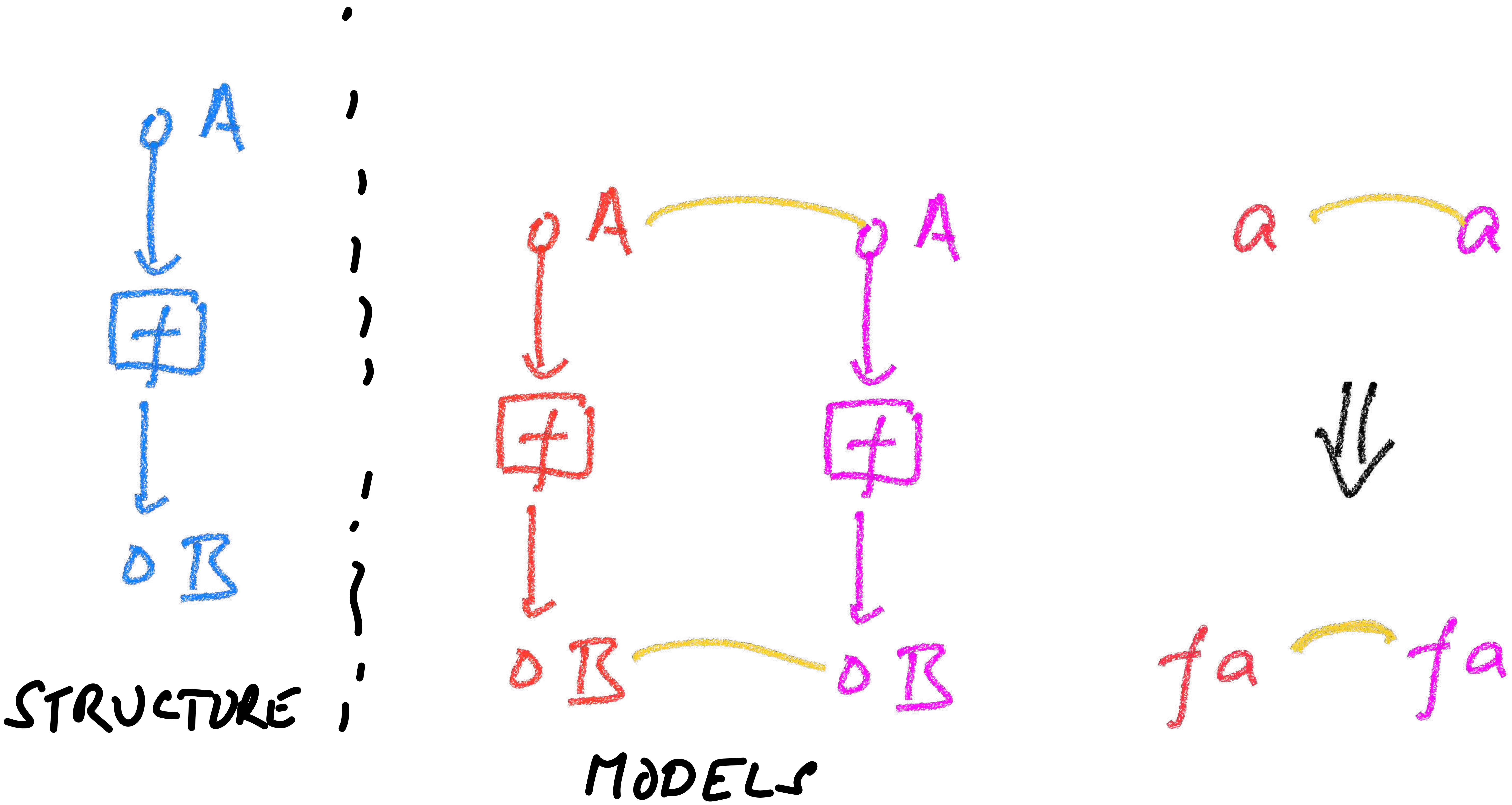
Basic types and operations

A Simple View

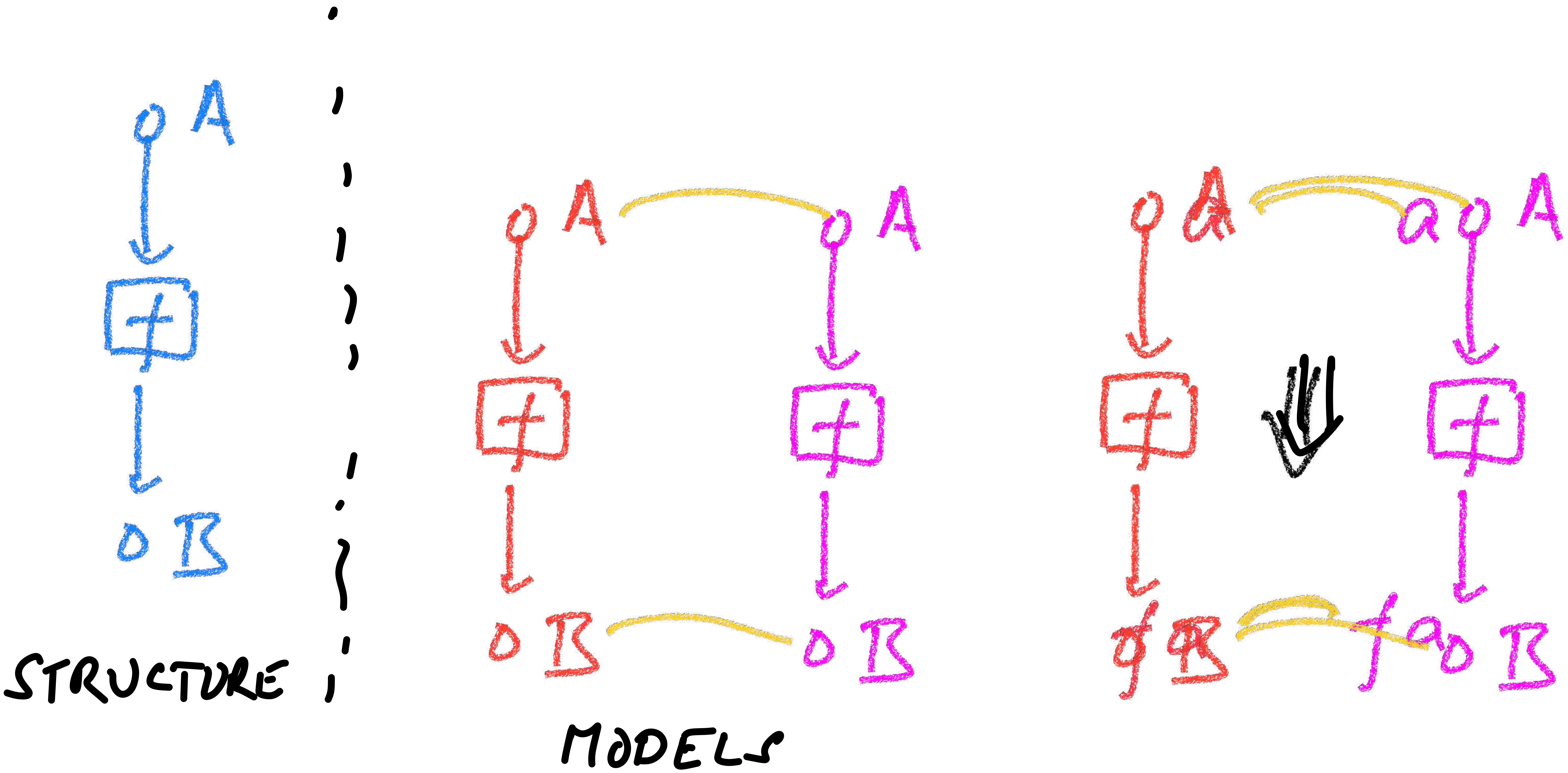


- Our structure comprises:
- Some basic entities (objects, A, B,…)
- And operations between them.

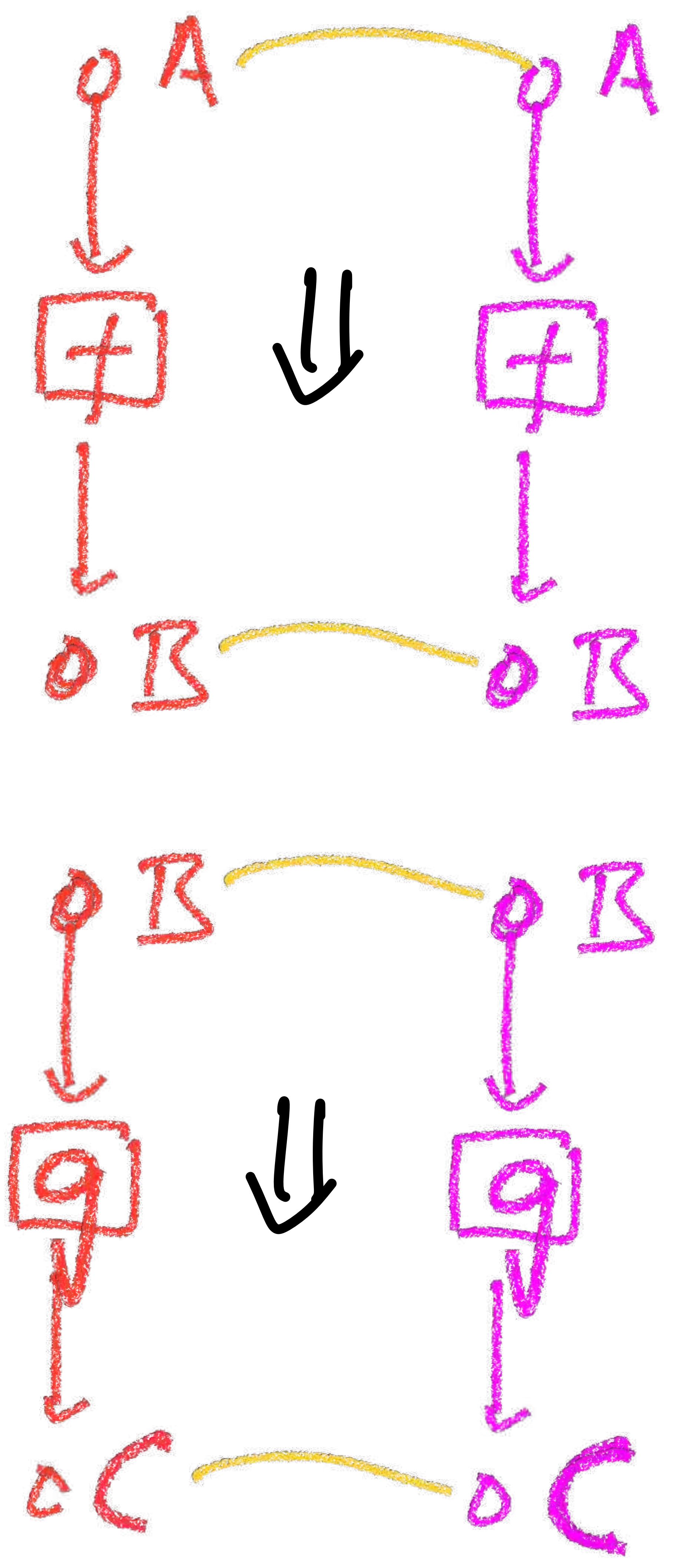
A Simple View



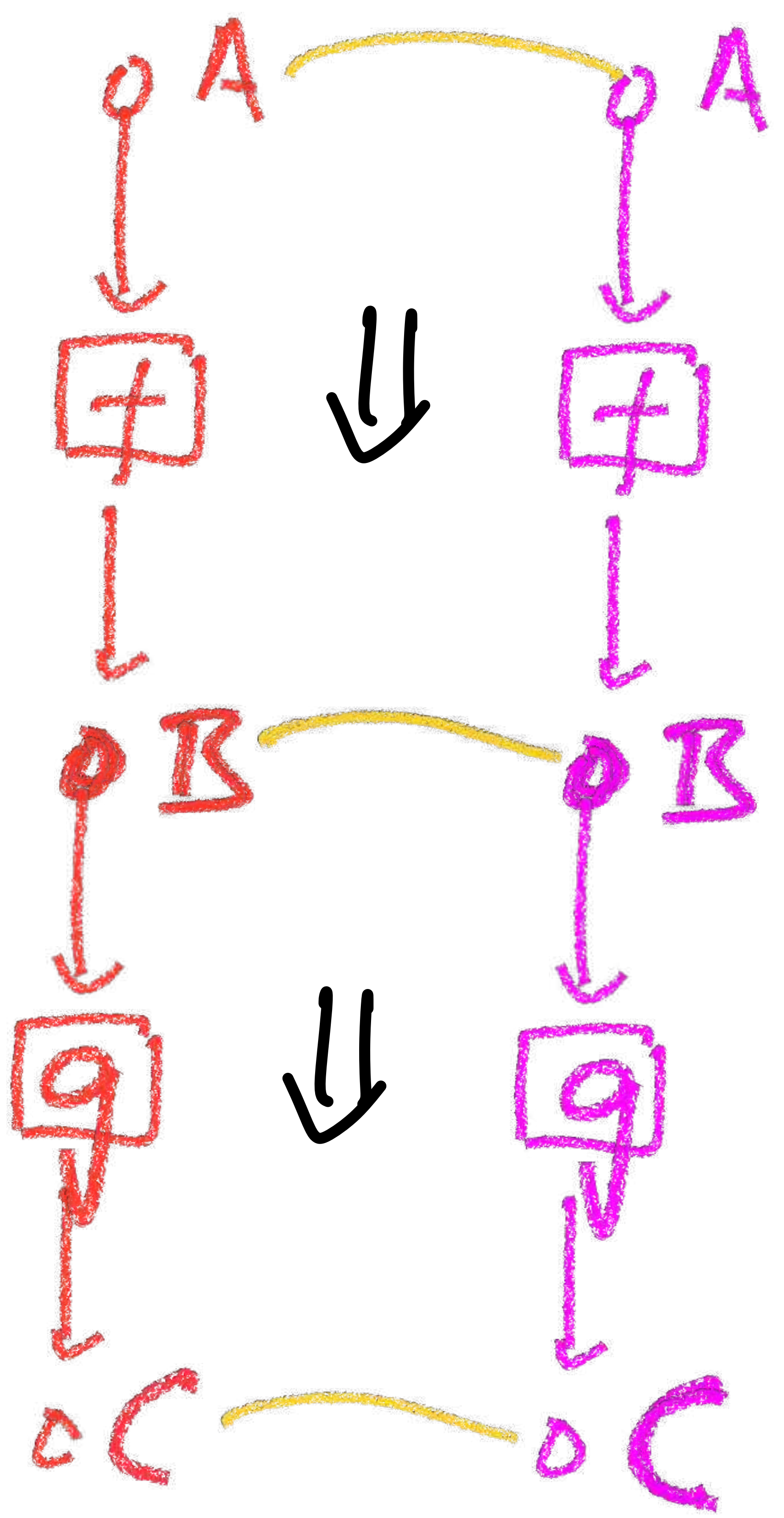
A Simple View



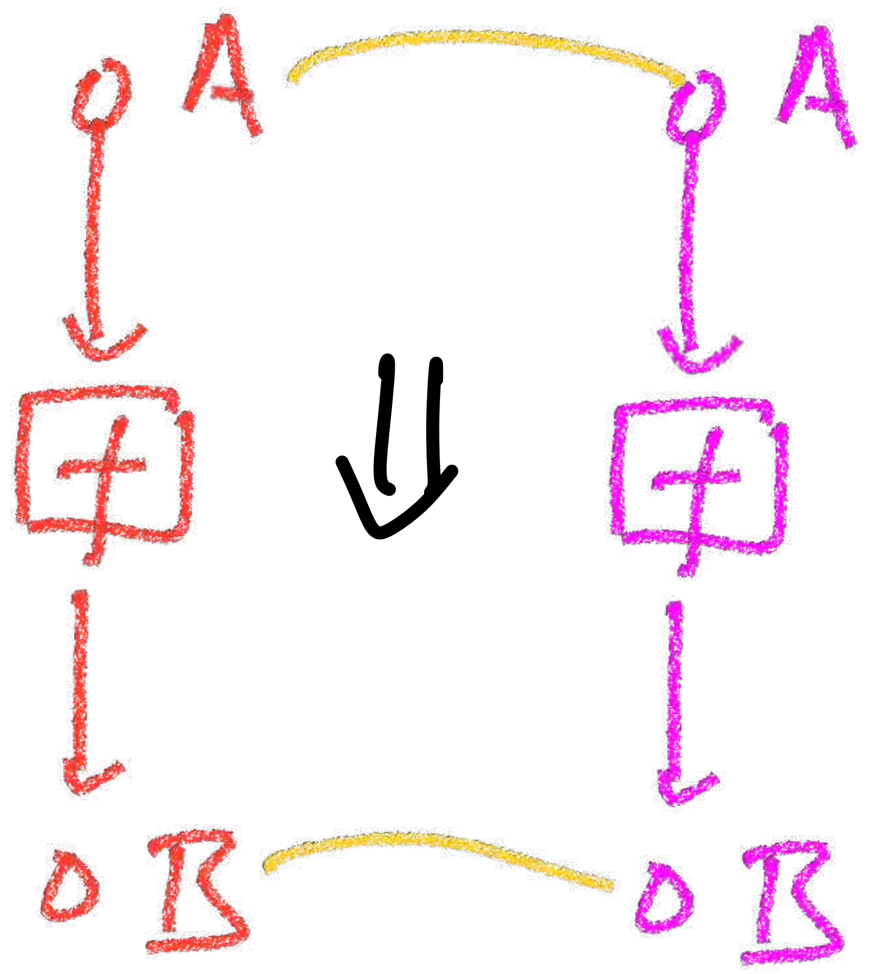
Putting things together: operations compose

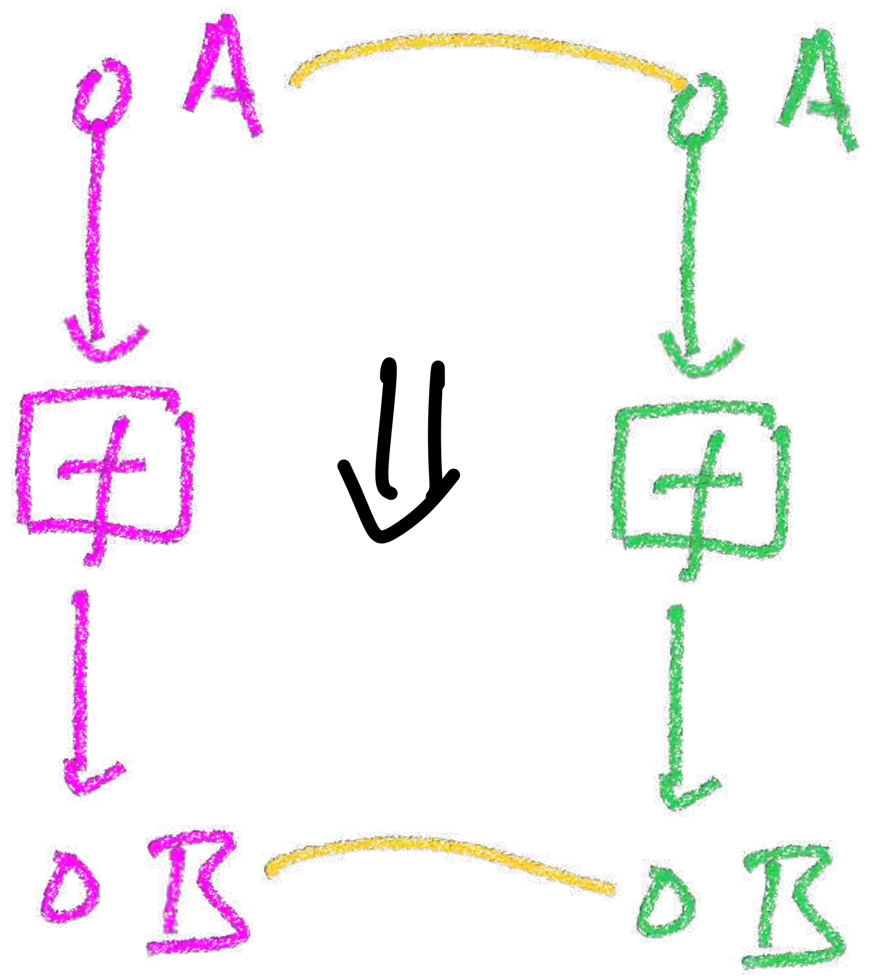


Putting things together: operations compose

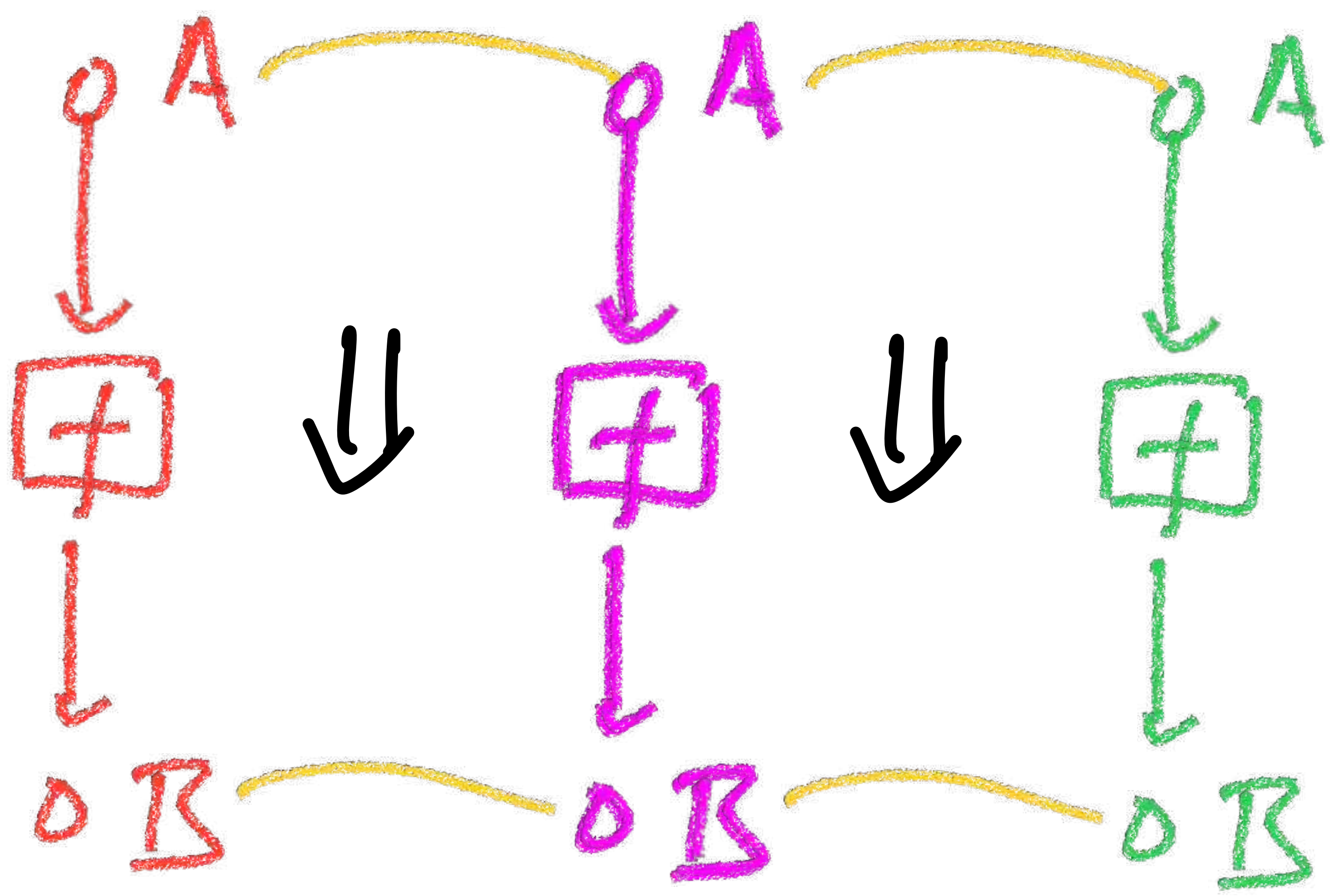


Putting things together: relations compose

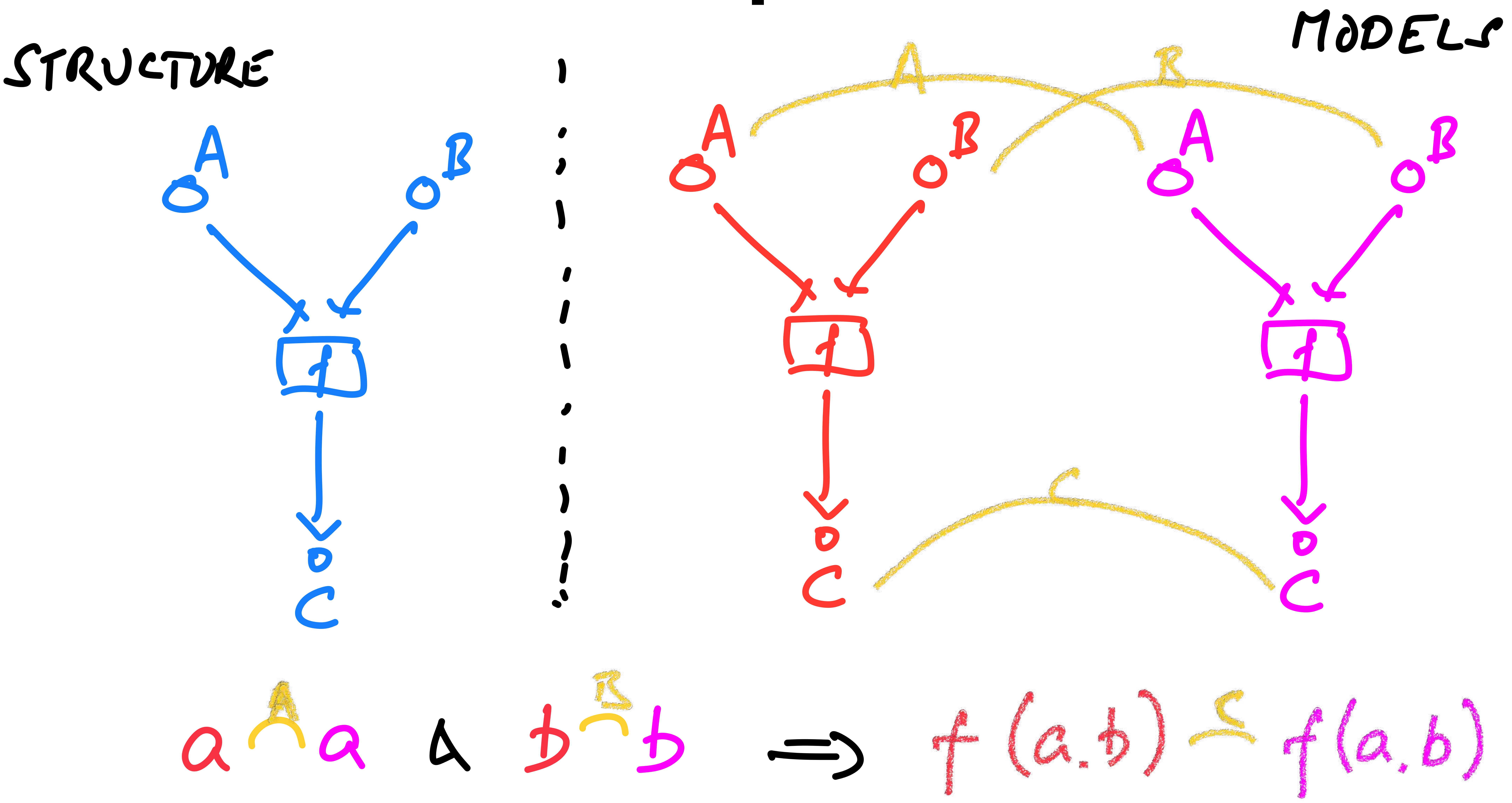




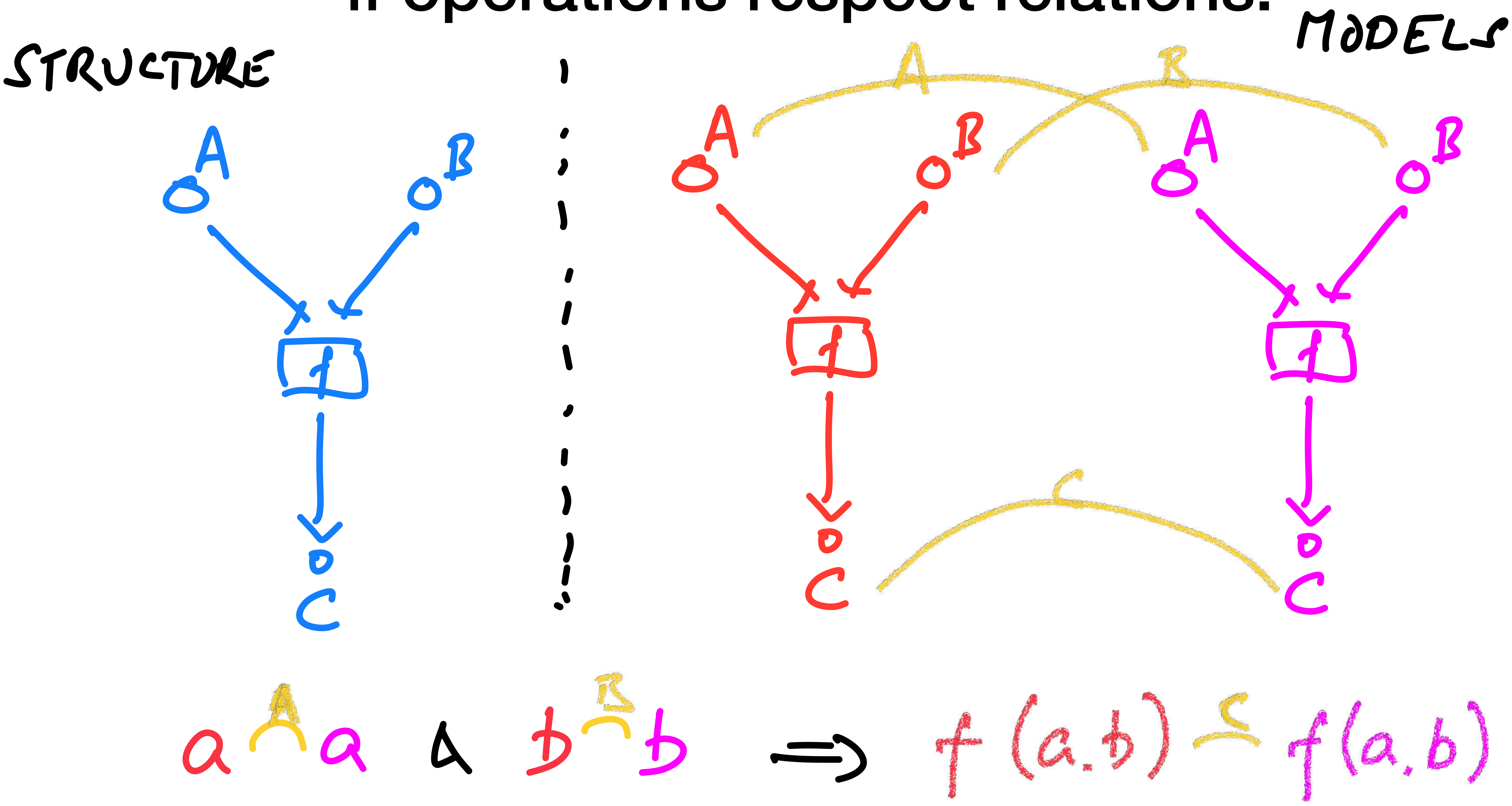
Putting things together: relations compose



A Simple View



A system of relations is “Logical” if operations respect relations.



A system of relations between models
is “logical”
if
the operations respect the relations.

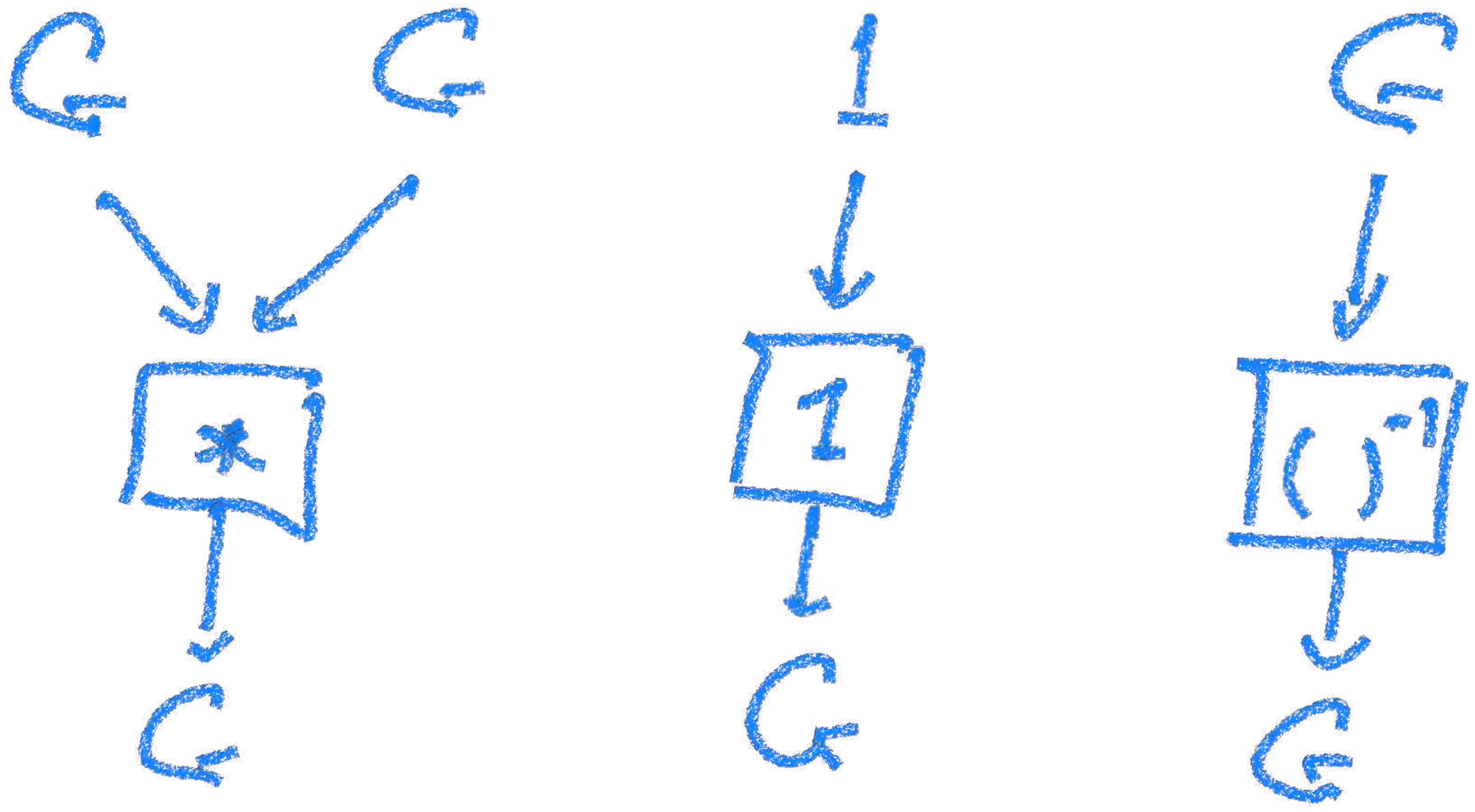
Algebra

- Objects - giving basic sorts
- Operations - between objects
- Equations - between operations
- Usual interpretation:
  - Object = Set
  - Operation = Function

Example: Groups

- One basic objects/sort: the group carrier

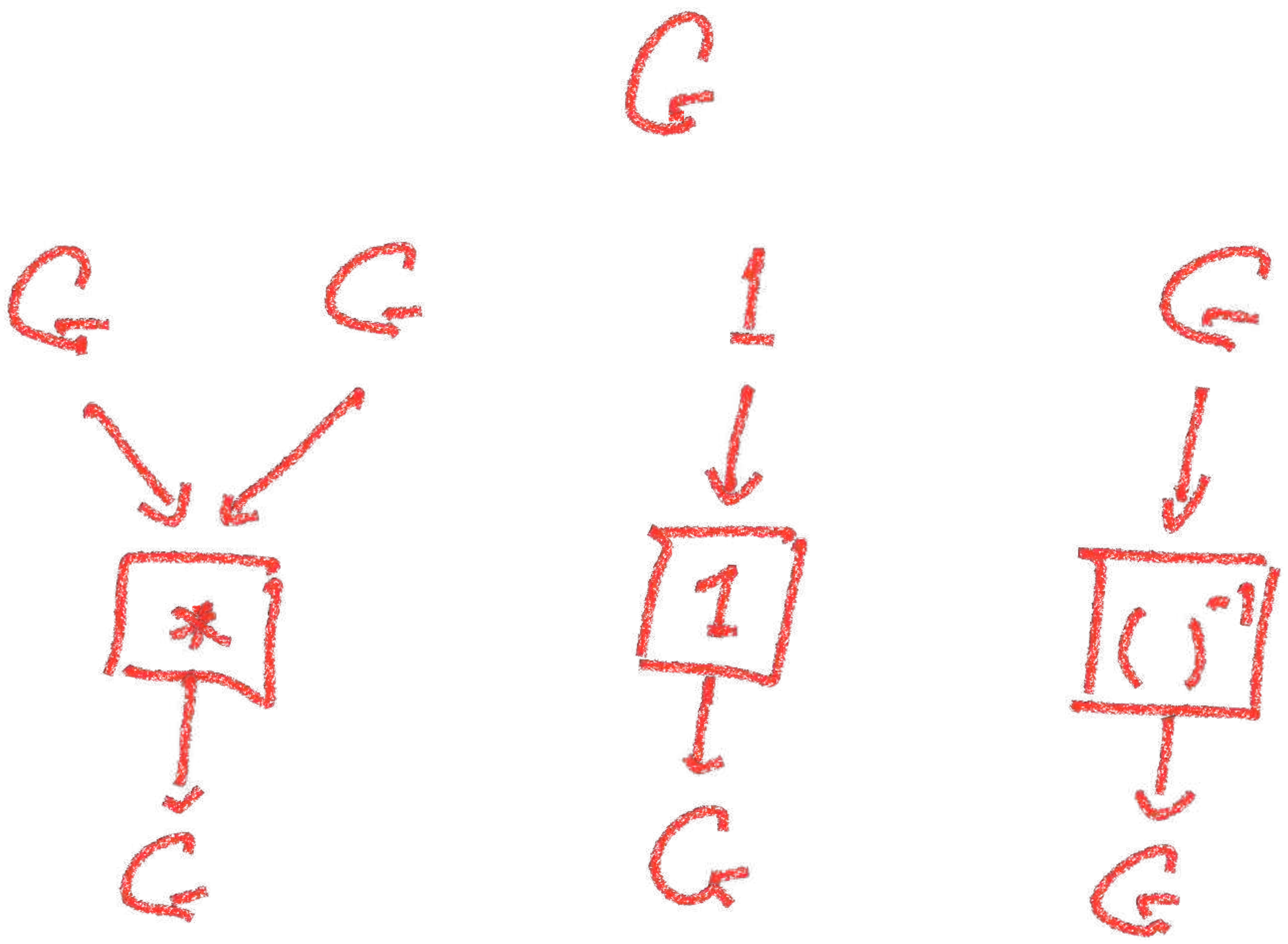
- Three operations:

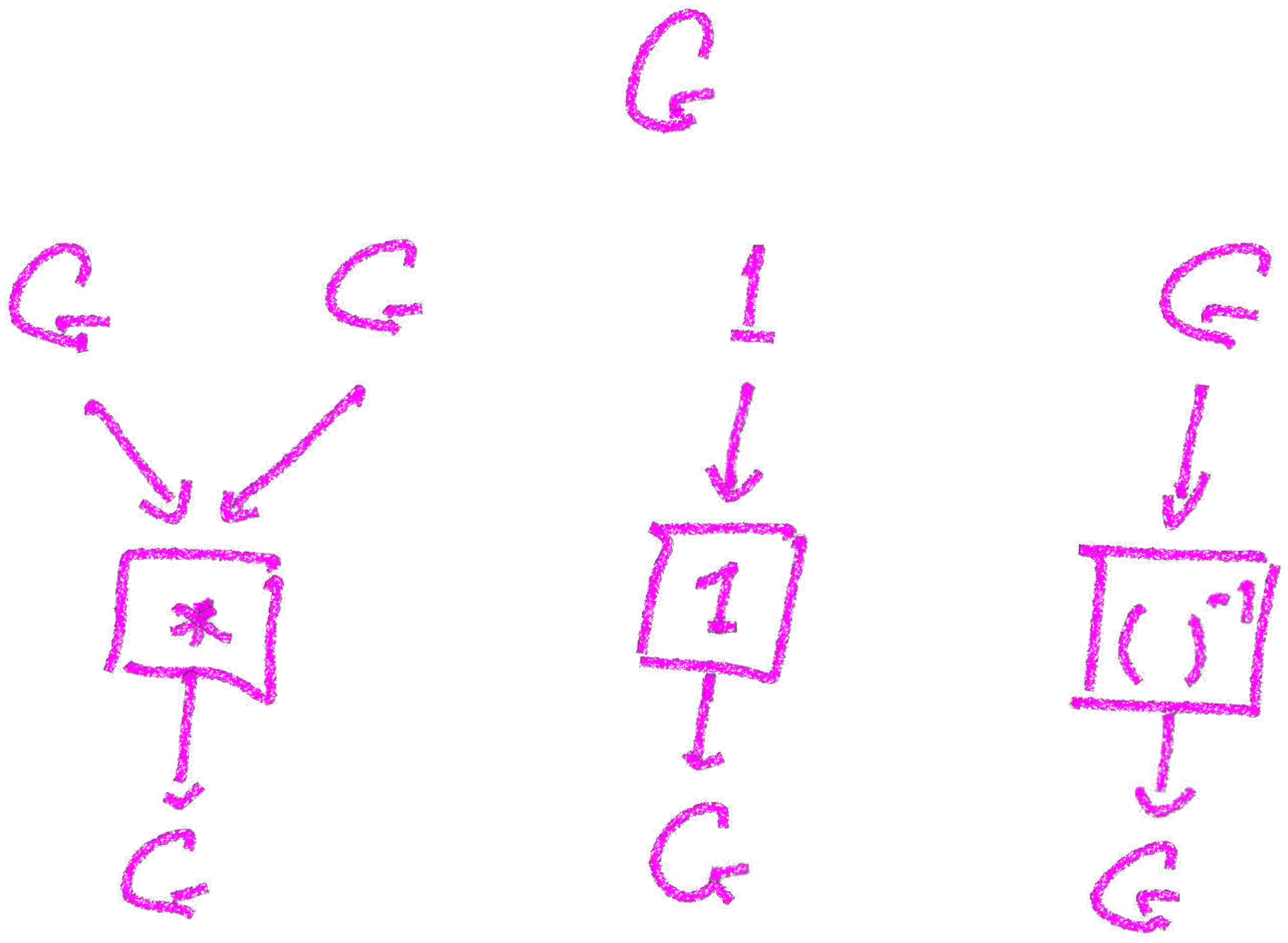


- Plus equations: associativity, identity, inverse

Models: Groups

- Actual groups:



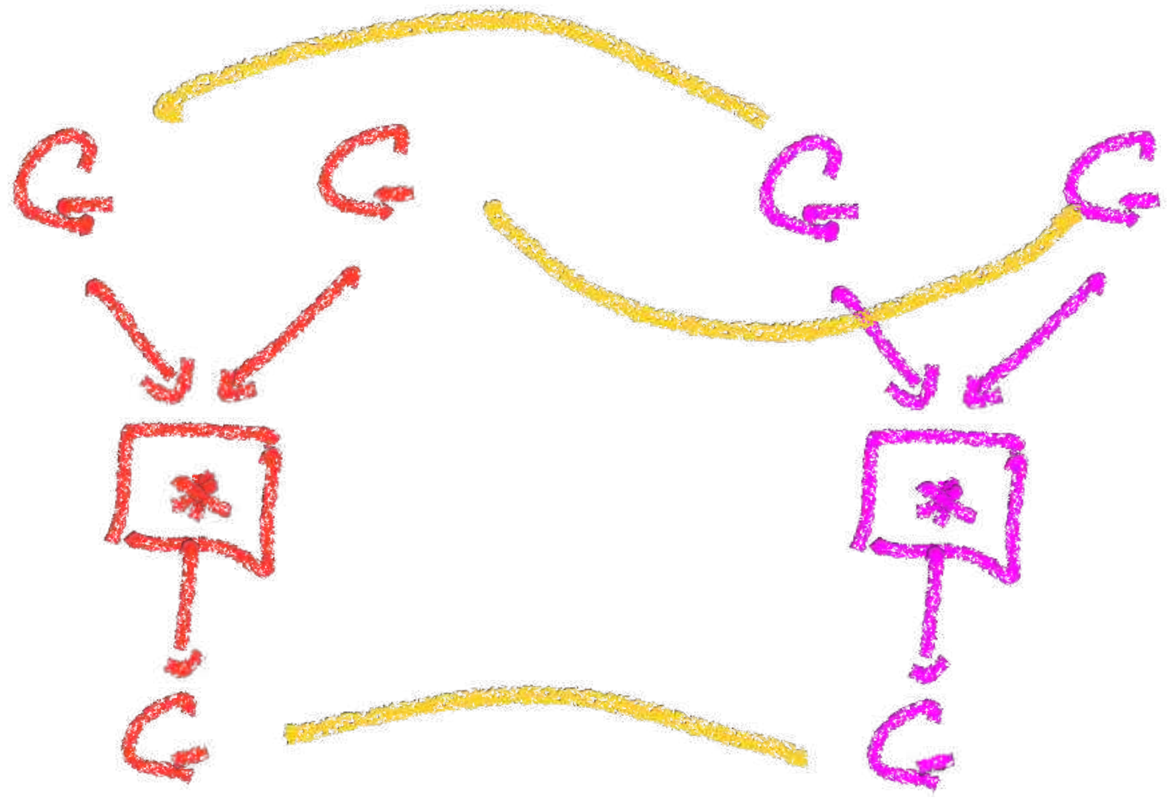


Definition. *A group G is a system of elements which is closed under a single-valued binary operation which is associative, and relative to which G contains an element satisfying the identity law, and with each element another element (called its inverse) satisfying the inverse law.*

Corollary. *A group has only one identity element, and only one inverse a^{-1} for each element a.*

Birkhoff and Mac Lane: A Survey of Modern Algebra

Models



- Models are actual groups: $G \quad G$
- A relation $G \frown G$ is logical iff
- $g \frown g \;\wedge\; g' \frown g' \;\Rightarrow\; g * g' \frown g * g'$
- $1 \frown 1$
- $g \frown g \;\Rightarrow\; g^{-1} \frown g^{-1}$

Models

- If f is a function G -> G and G and G are groups, then
  - Graph(f) is a logical relation for the group operations
  - iff f is a group homomorphism.

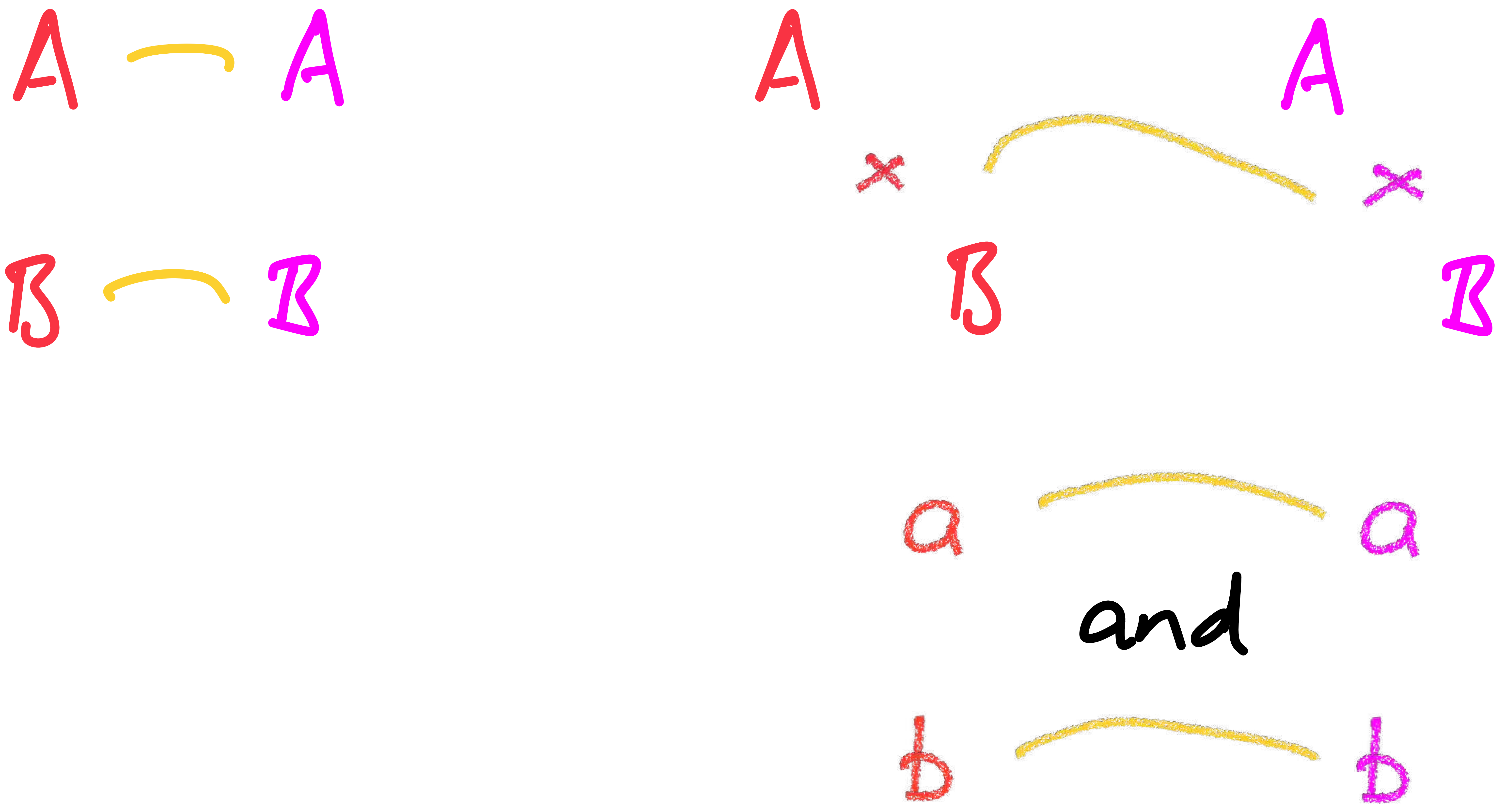
- This result holds for arbitrary algebraic theories.
- Logical relations generalise, and encapsulate a standard algebraic concept.

First-order types and a bit of category theory

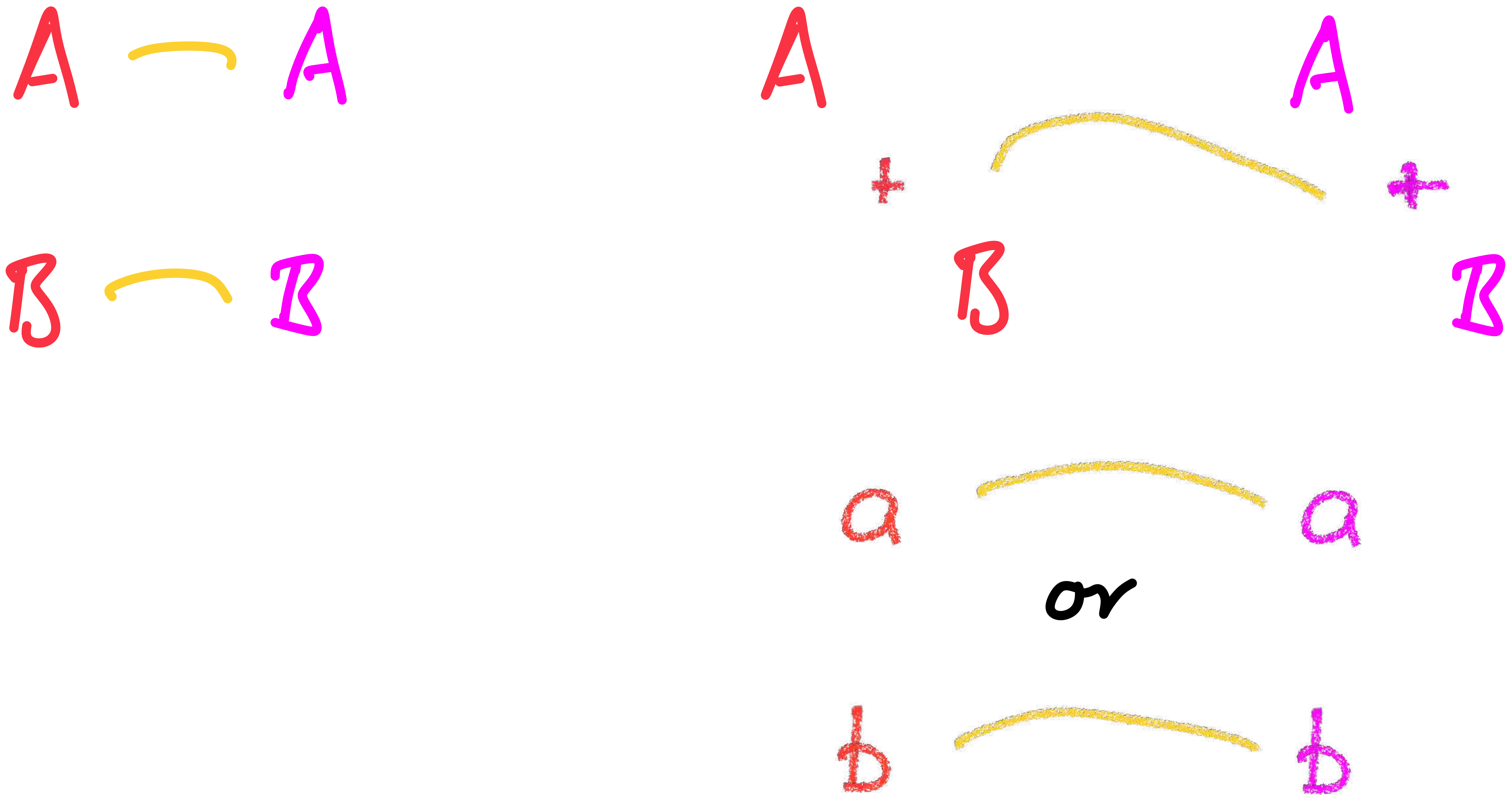
We want to use more than just the basic objects

- First-order types:
  - Products and sums
- If we have a product of types in our structure, then we want to generate a relation between the corresponding products in our two different models.

Products of relations

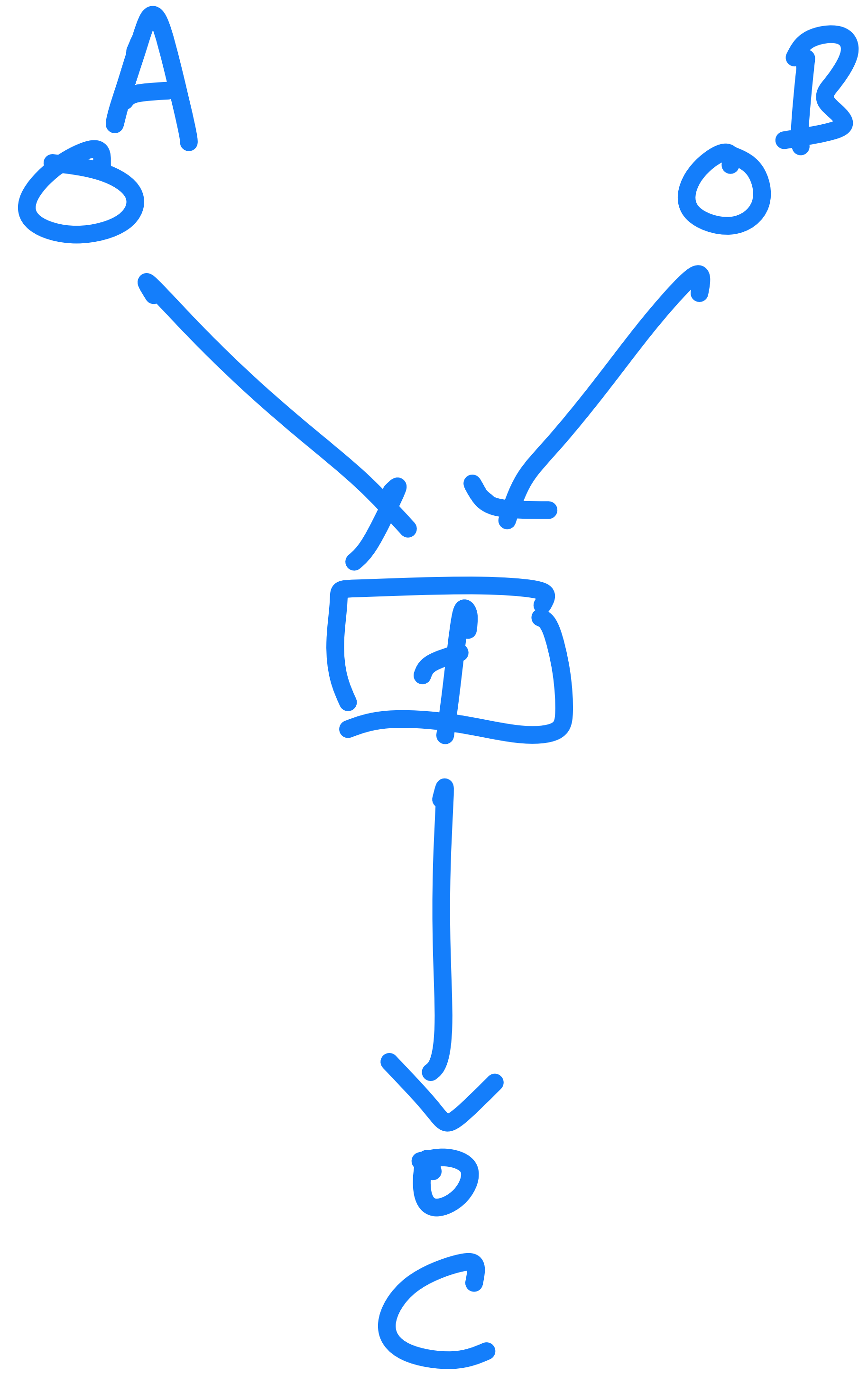


Sums of relations



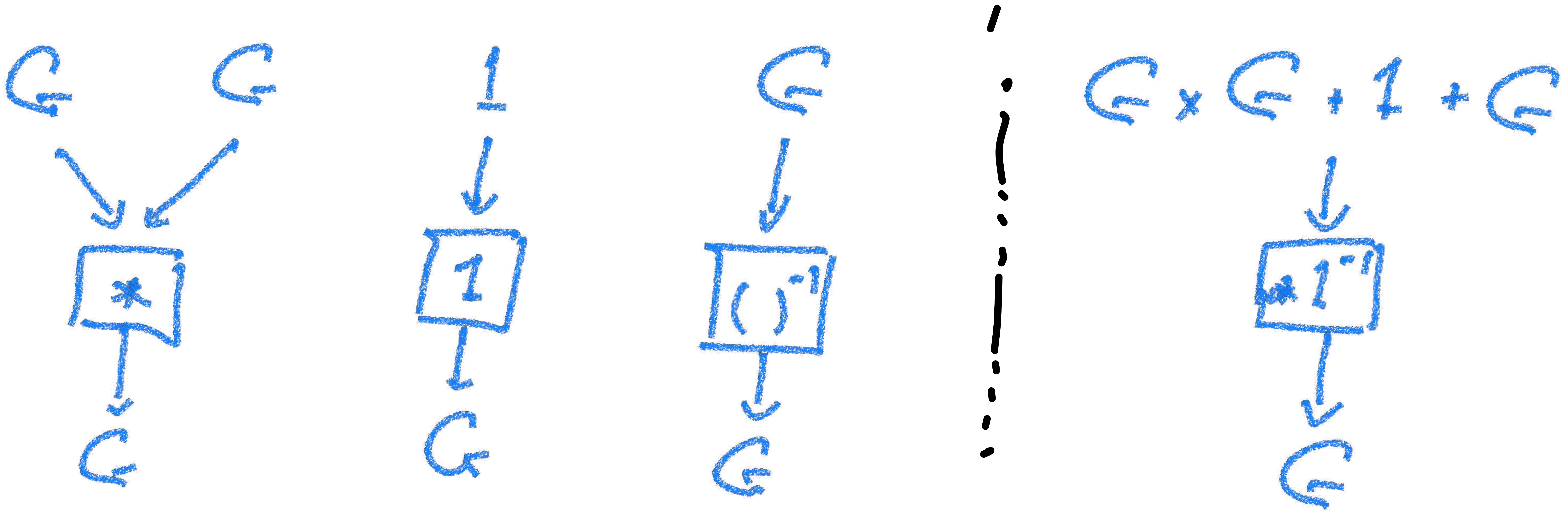
n-ary operations

- A n-ary operation is equivalent to a unary operation on the product of the inputs.



amalgamating operations

- Having two operations is equivalent to a unary operation on the sum of the inputs.
  - Example: groups



Algebra and Co-algebra

- Classically, both deal with one-sorted theories, ie one basic type
  - algebra says that elements of that type can be combined into others by applying operations
  - co-algebra says that elements of that type can be decomposed into the result of applying such operations to other elements of the type.

Multiplication and co-multiplication

Category Theory

- In the categorical account of algebra, terms are packaged up into a functor.
- TA = terms built from algebra operations and constants that are elements of A
- Algebra: $TA \longrightarrow A$
- Coalgebra: $A \longrightarrow TA$

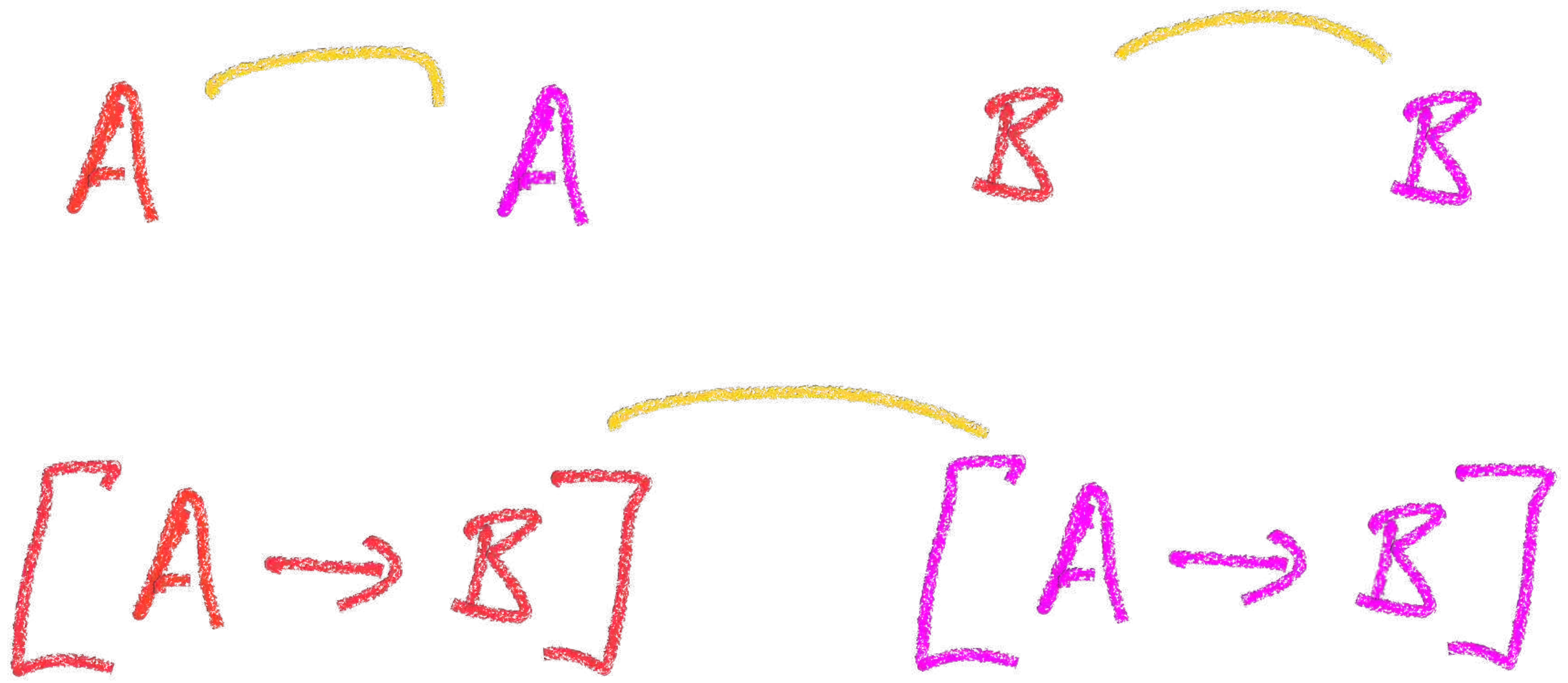
Compositionality

- At this level everything is fine.
- Operations compose.
- We can use type-theoretic operations we expect (projection, tupling, injection, case).
- Logical relations compose.

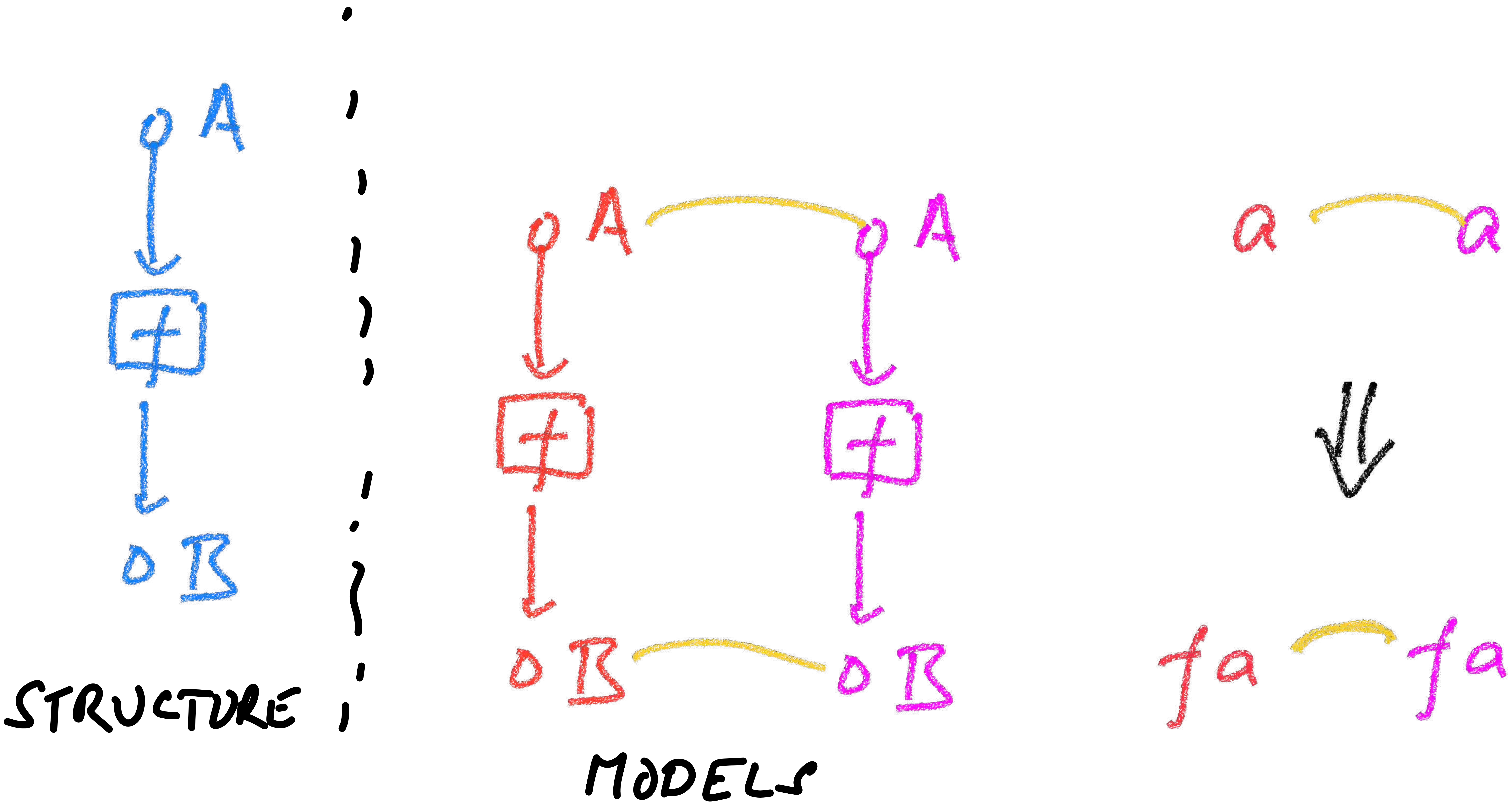
Higher-order types: Functions

Exponentiation of relations

- Given pairs of types
- And relations
- Can we get a relation

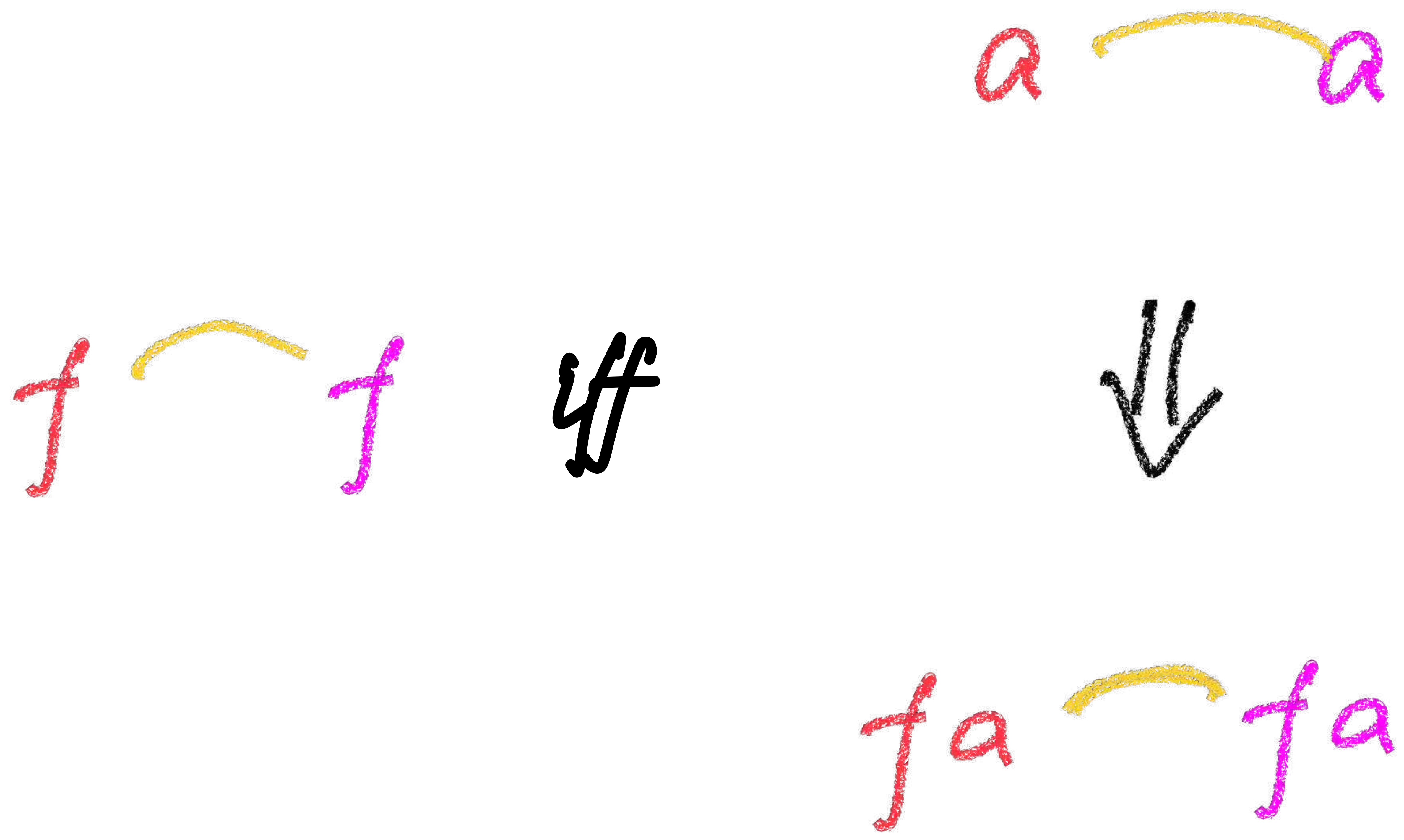


A Simple View

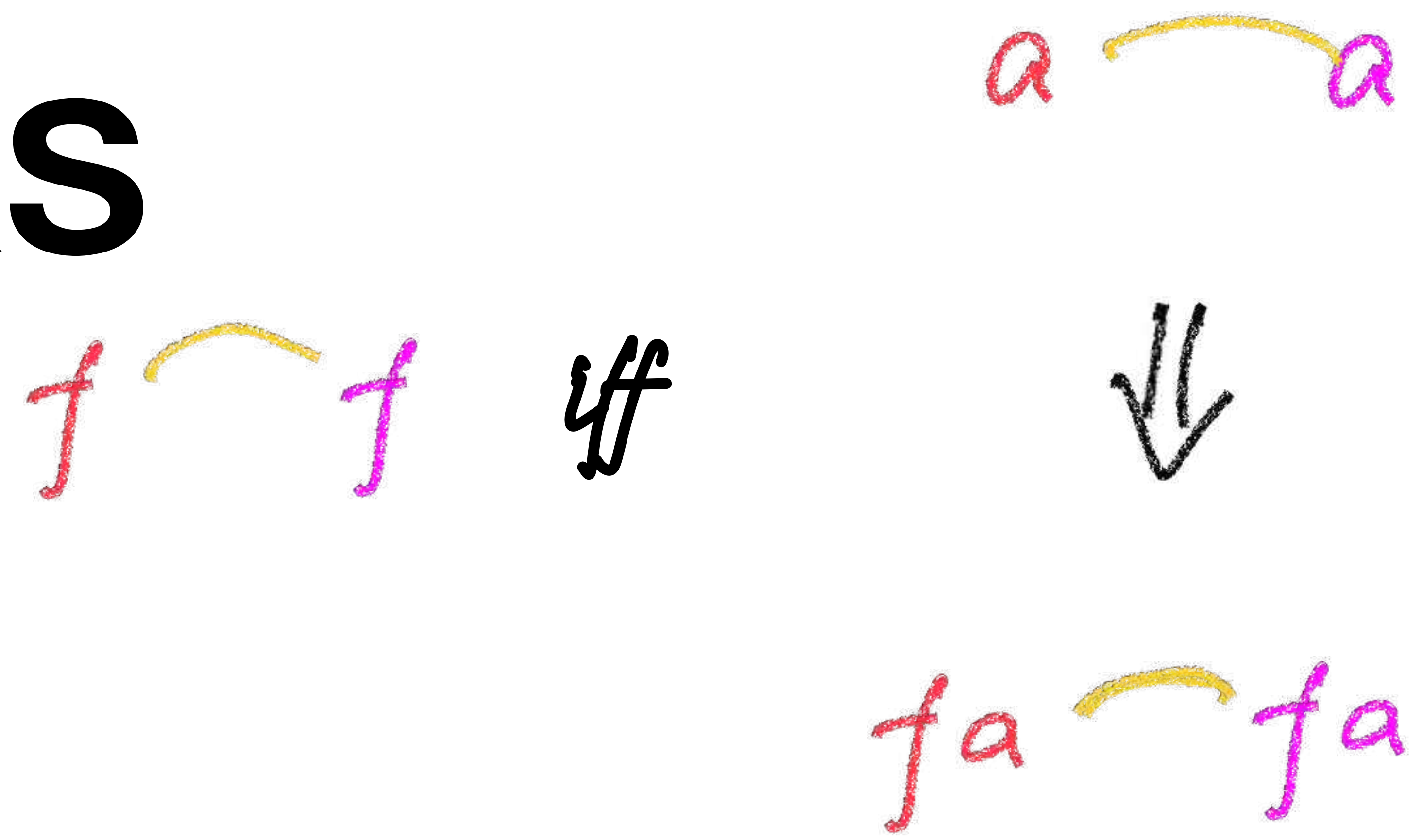


Exponentiation of relations

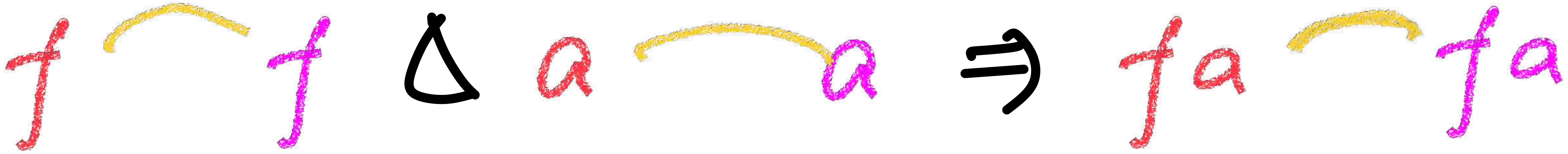
- Given pairs of types
- And relations
- Can we get a relation
- Ans: yes



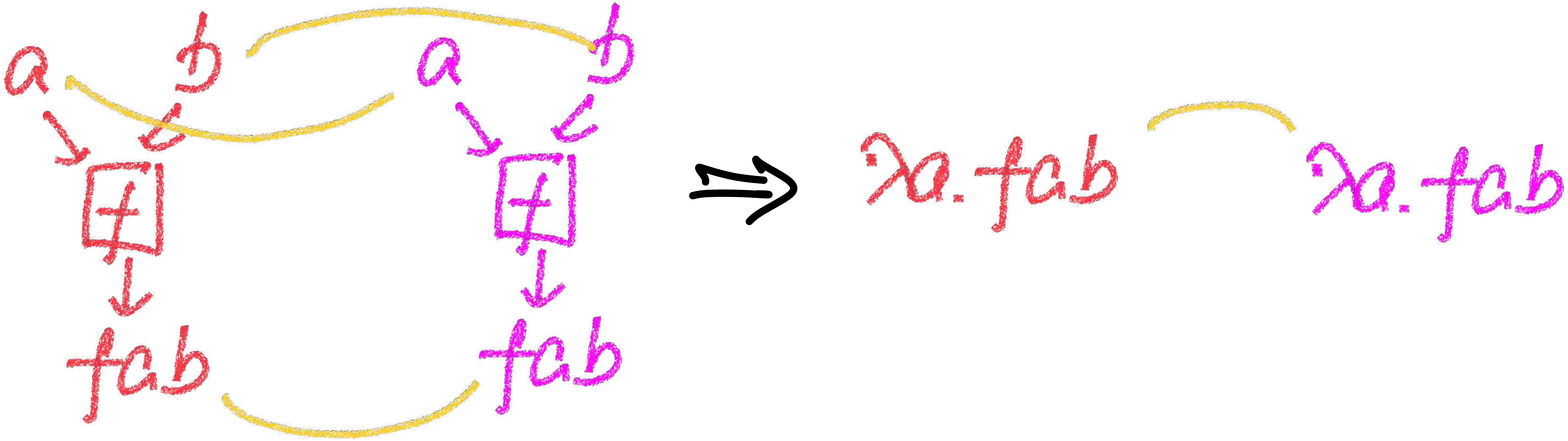
How this works



- Application is OK:



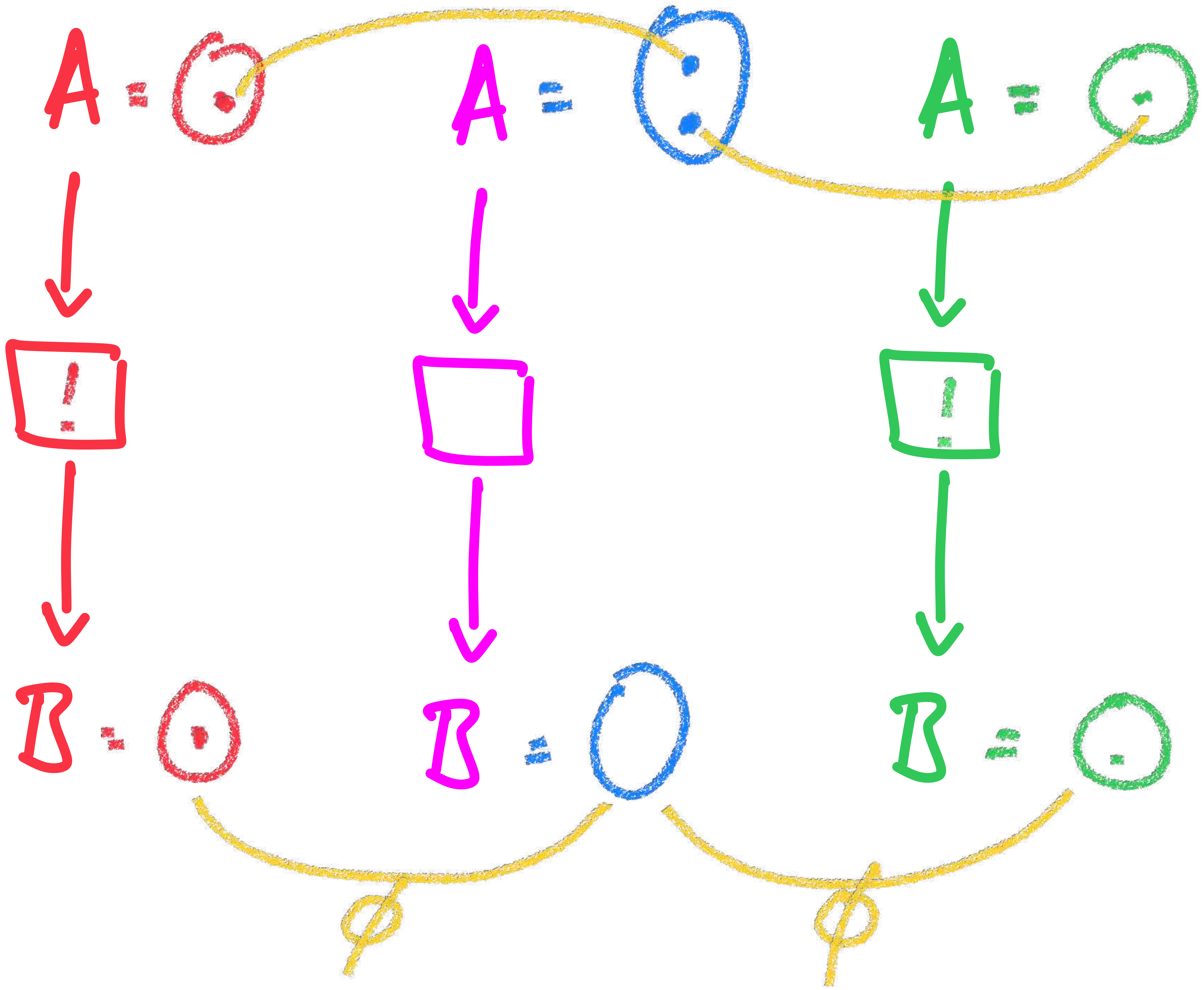
- As is lambda abstraction



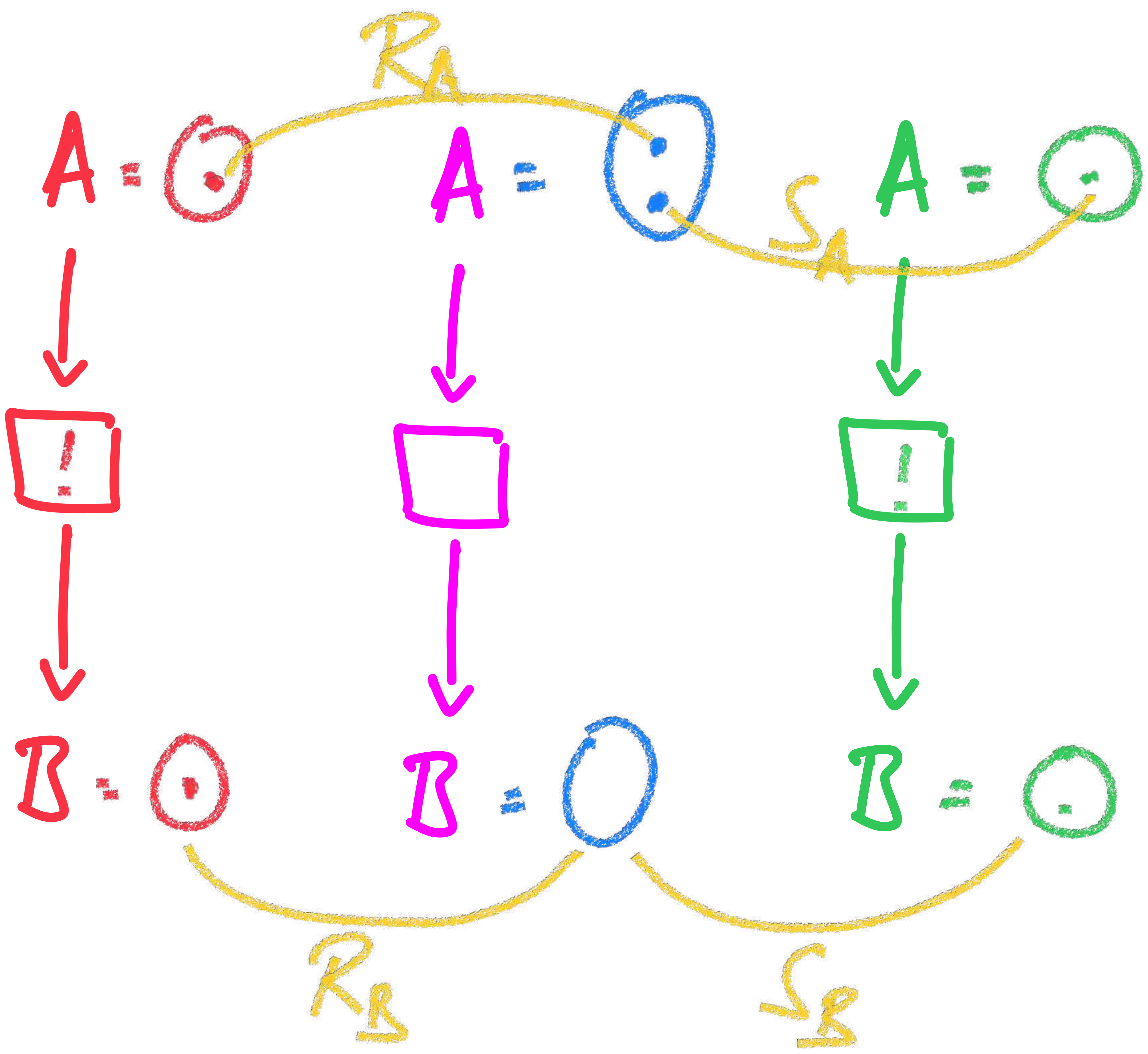
How it breaks down: failure of composition

- Other constructors preserve order on relations, $\to$ does not.
- Compositionality of relations fails

How it breaks down: failure of composition



How it breaks down: failure of composition



$$[R_A \to R_B][S_A \to S_B] \neq [R_A S_A \to R_B S_B]$$

But we do get $\leq$

So far all very concrete:
Sets are the go to mathematical structure for building things

But logic is the go to tool for reasoning about them.

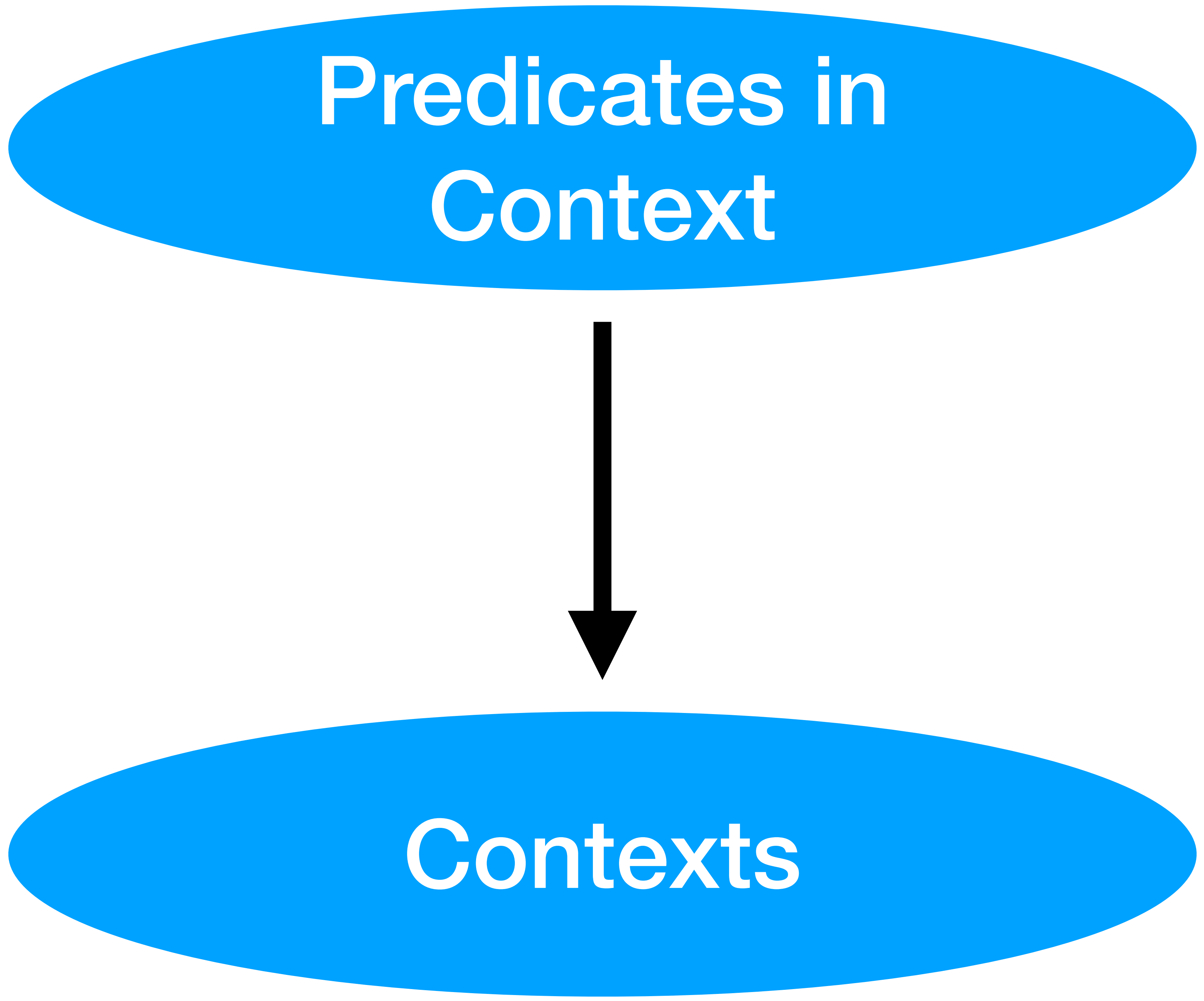
Get rid of the sets: a logic-based approach

Look at the proofs

- Proofs all use logic and basic type theory, not really set theory
- We need a predicate logic, with sorts and predicates over sorts.
- Start with a unary version.

Logic as a type theory

- Types: sort (=context) + predicate defined in that context.
- Terms: have two components -
  - substitution
  - entailment



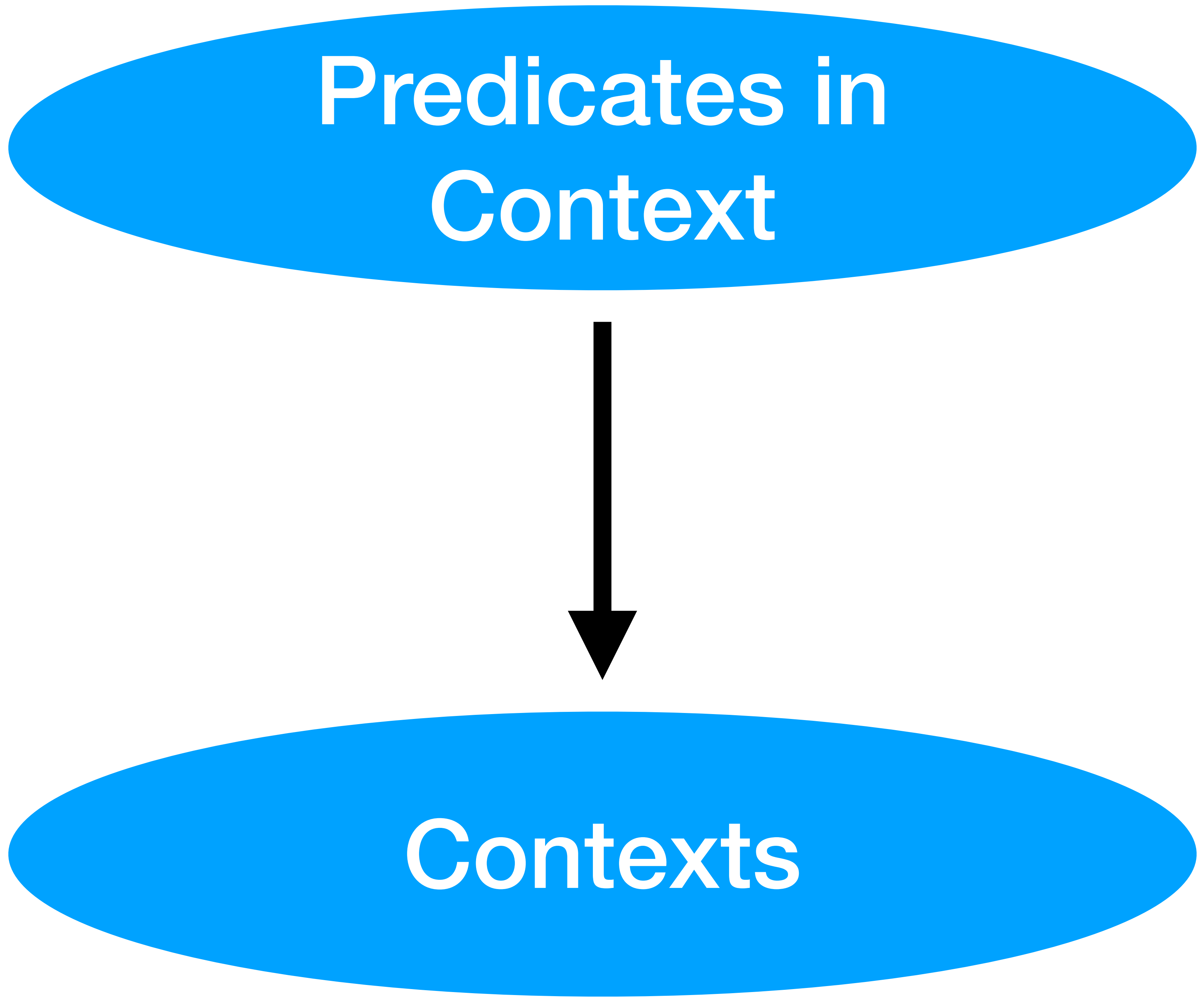
Logic as a type theory

- Functions between contexts generalise terms (they are substitutions).

$$x : A \xrightarrow{y := e(x)} y : B$$

- Predicates have a context (their free variables).

$$x : A \vdash P(x)$$



Logic as a type theory

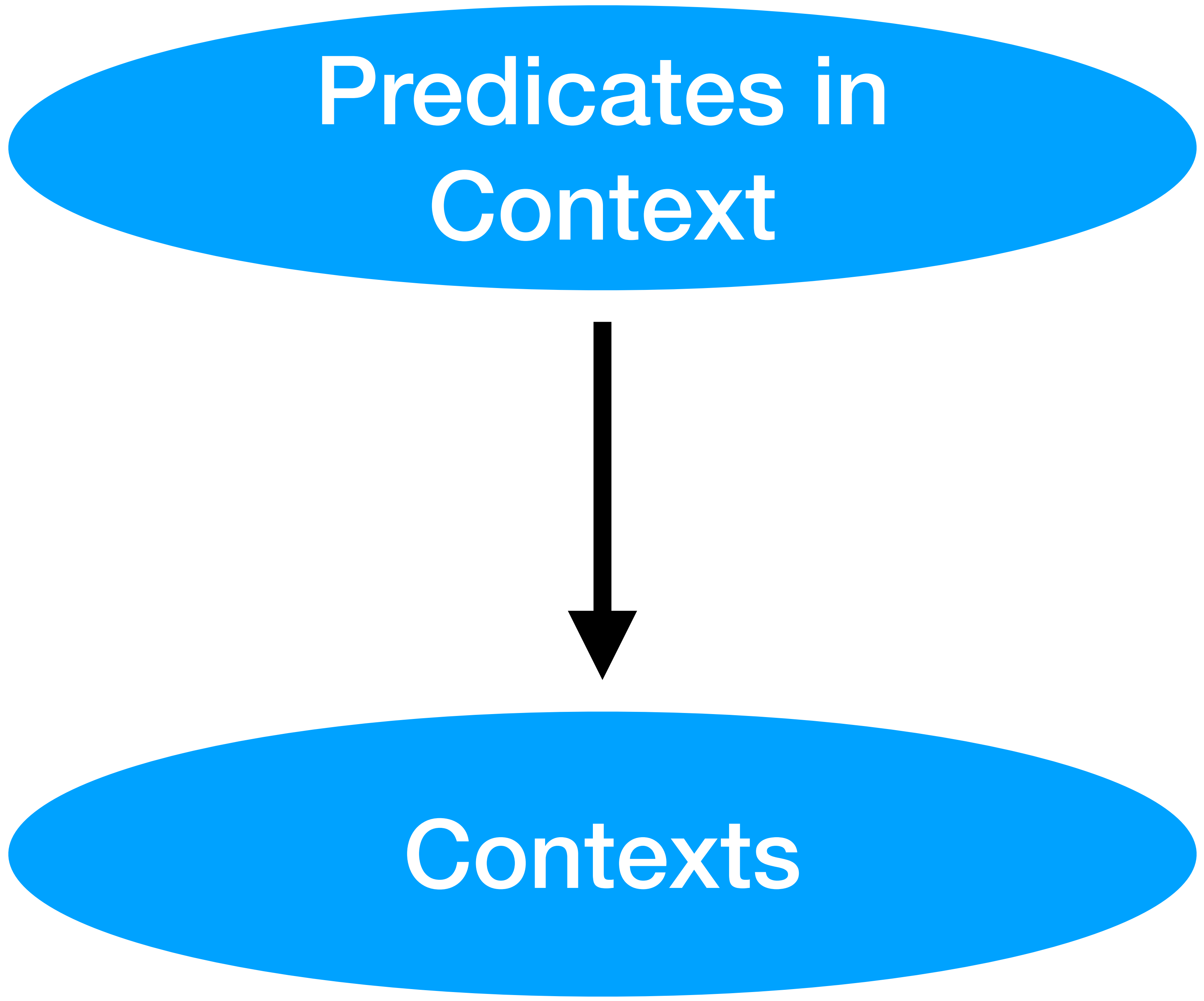
- Predicates have a context (their free variables).

$$x : A \vdash P(x)$$

- Functions between predicates are a substitution and an entailment.

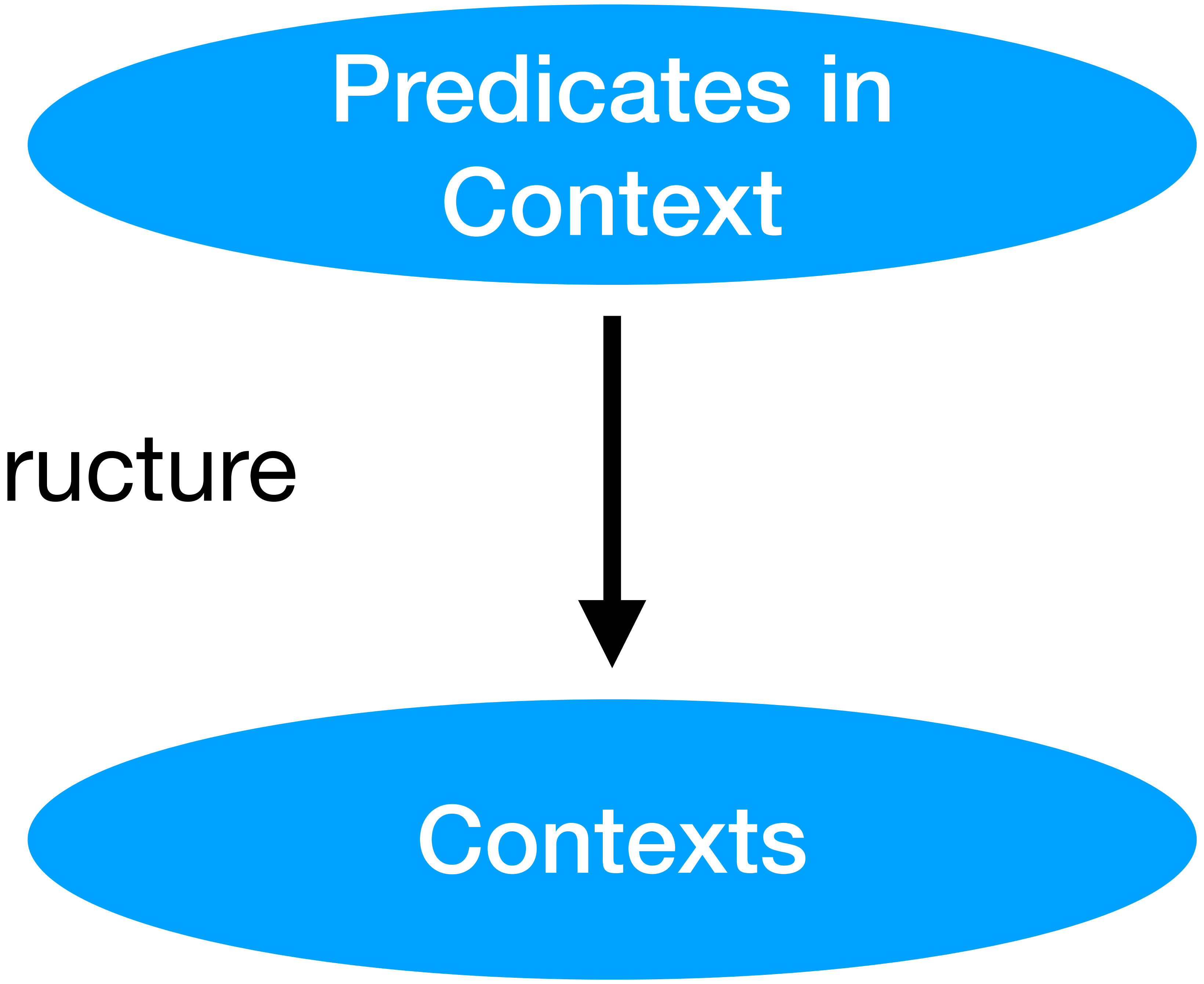
$$x : A \vdash P(x) \underset{y := e(x)}{\rightsquigarrow} y : B \vdash Q(y)$$

$$x : A \vdash P(x) \to Q(e(x))$$

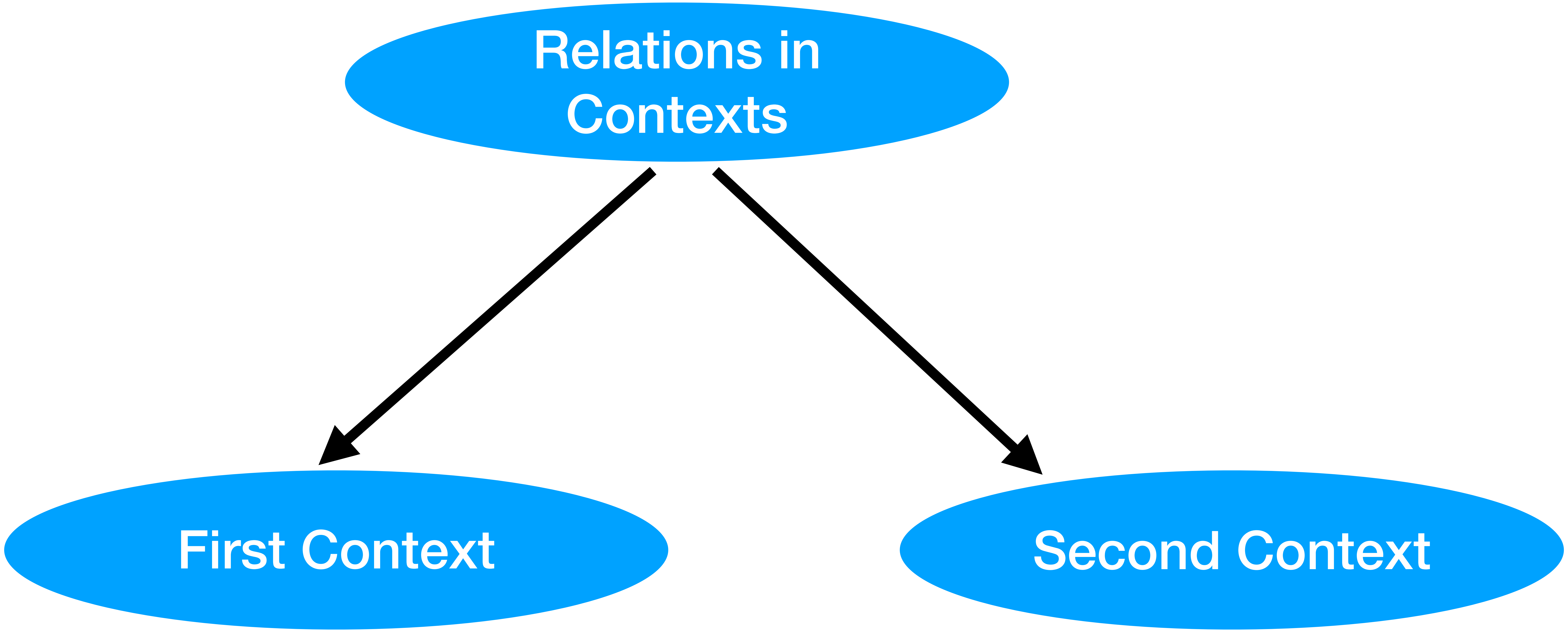


Logic as a type theory

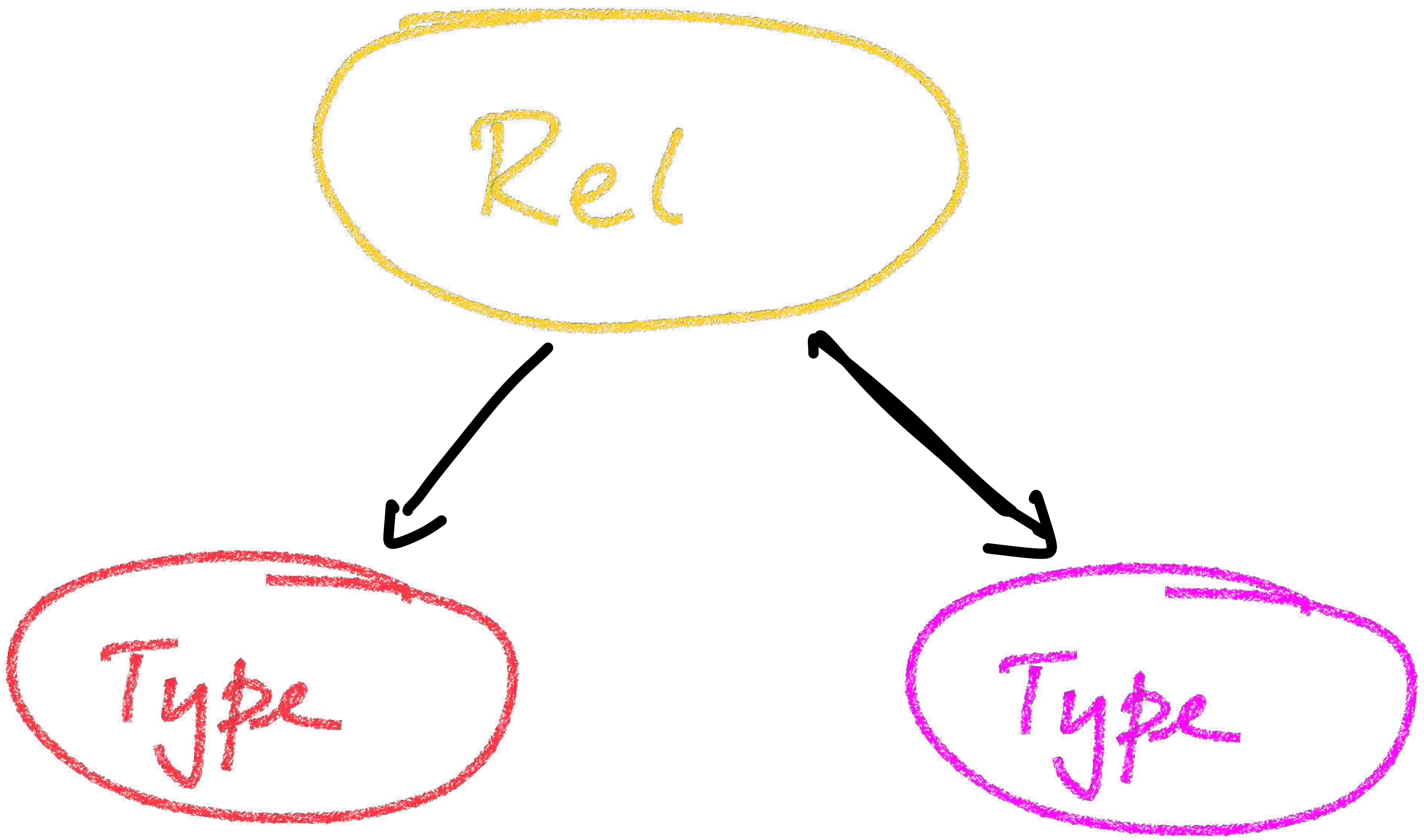
- Model of type theory
    - Homomorphism of structure
- Model of type theory



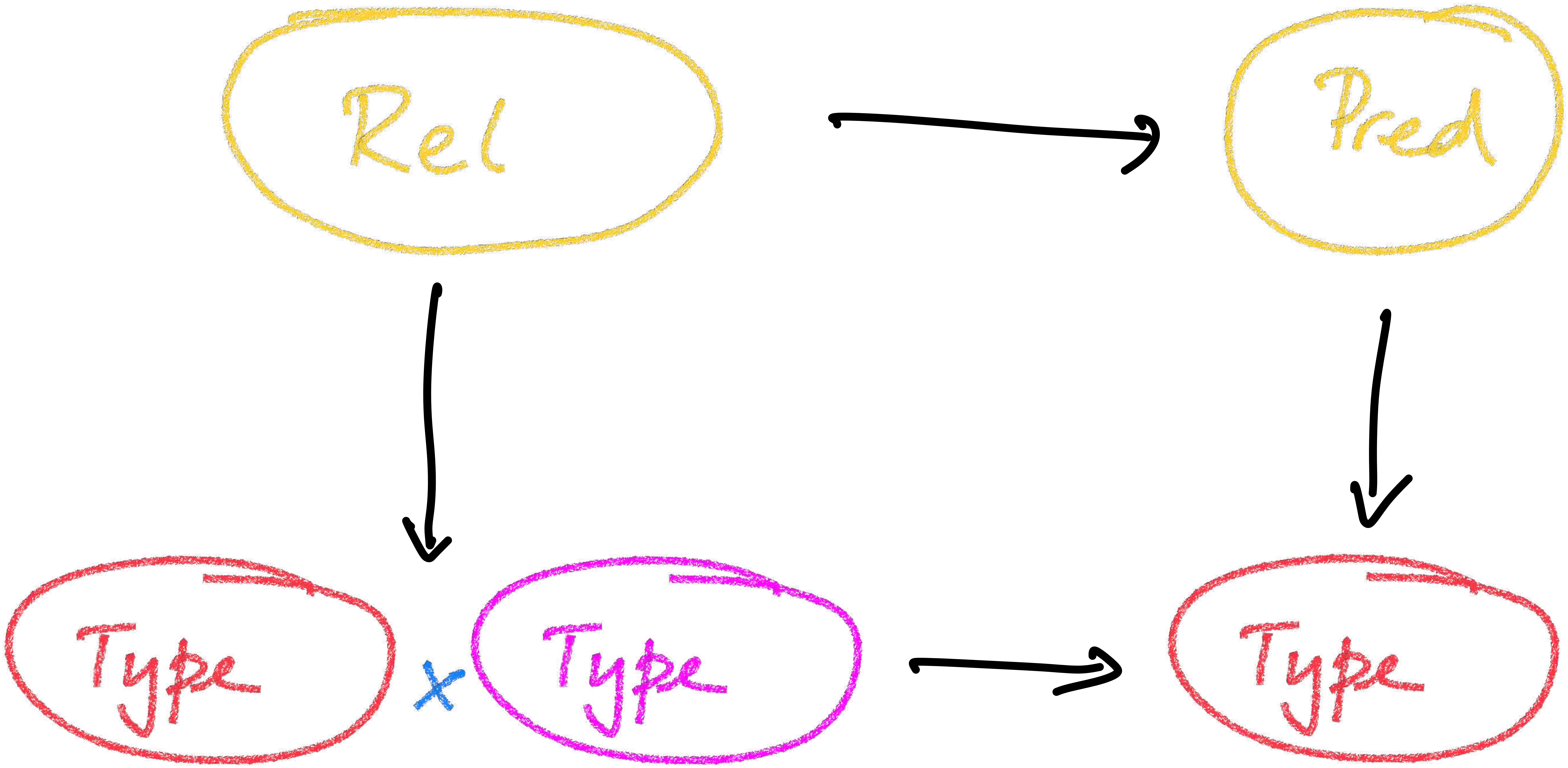
Can also do this for binary predicates (relations)



Core idea



Core idea



Sets again

- This story works semantically for sets and relations.
- Ditch the idea that a relation is just the set of elements.
- A relation also has to know what sets it is a relation between.

Why should we care?

- Ans: not everything is a set, not every construction uses set-theoretic constructions.
- Example: Kripke logical relations, step-indexed logical relations
- Idea: work in a world where everything is, say, Kripke. Kripke gives good interpretation of logic. Binary predicates give logical relations.
- Key to understanding lots of complicated papers is that they are just talking about this simple picture in the context of a complicated world.

Using structure to derive congruences

Example: State transition systems

Different forms of bisimulation can be derived from different ways of modelling systems

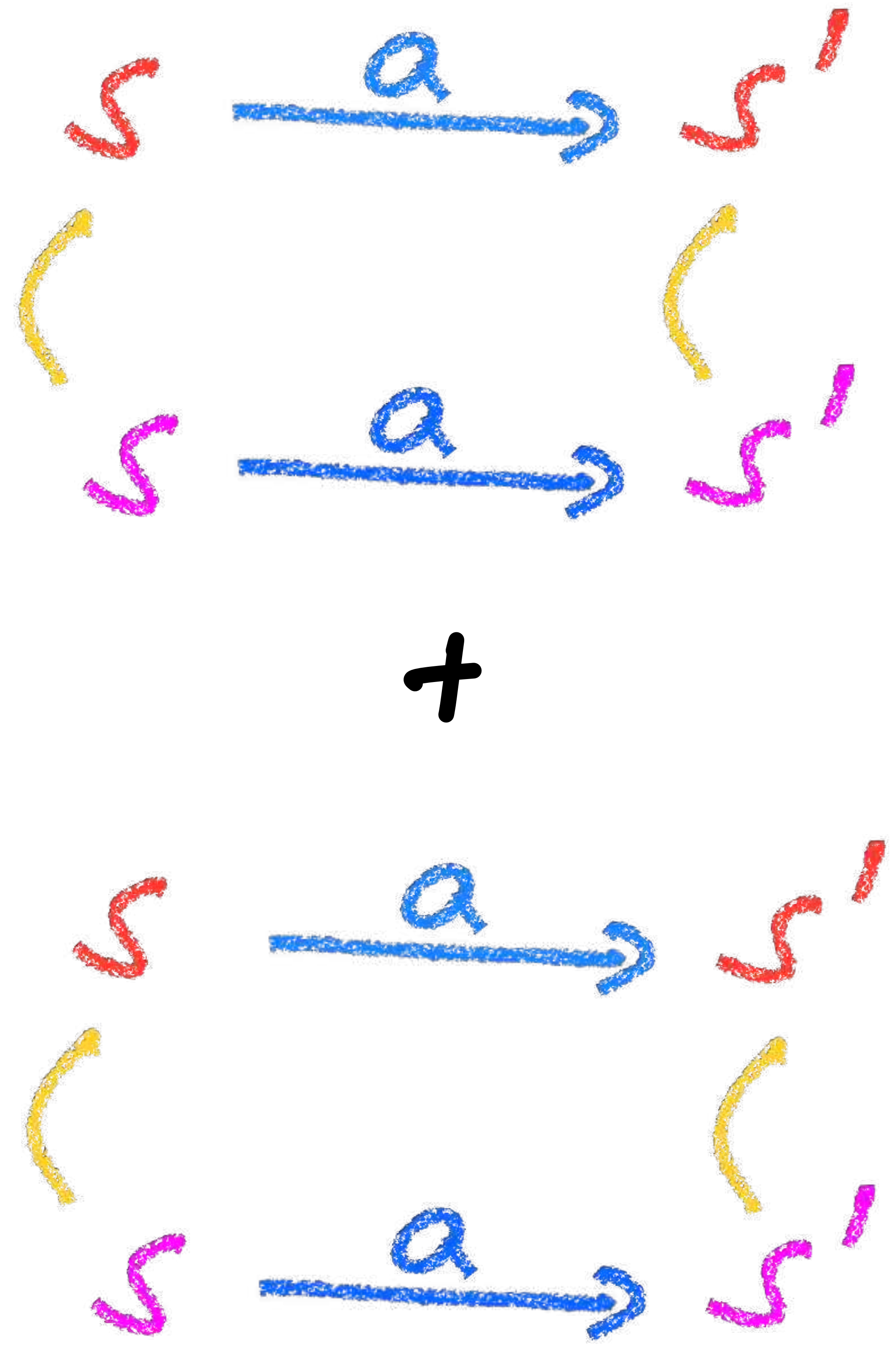
Labelled non-deterministic state transition system

- A set of States S
- a labelled transition relation $s \xrightarrow{a} s'$

Labelled non-deterministic state transition system: bisimulation (Park-Milner)

- Two systems

- Relation between them is a bisimulation if



Formalising

$$\alpha : A \times S \longrightarrow P(S)$$

$$\alpha(a, s) = \{ s' \mid s \xrightarrow{a} s' \}$$

Given $\alpha : A \times S \longrightarrow P(S)$

$\alpha : A \times S \longrightarrow P(S)$

for what relation $S \frown S$ is $\alpha \frown \alpha$?

Formalising

Given $\alpha : A \times S \longrightarrow P(S)$

$$\alpha : A \times S \longrightarrow P(S)$$

for what relations $S \frown S$ is $\alpha \frown \alpha$?

- Does not depend on exact way model is structured. e.g. $A \longrightarrow [S \to P(S)]$
- Does depend on how we extend $P(S)$ to relations.

Power-set as a type constructor: possibility 1

- Interpret the powerset of S as functions S -> Bool: P(S) = S -> Bool
- really strong, relational version of the contravariant power-set functor.

Power-set as a type constructor: possibility 2

- P(S) covariant power set functor,
- is the “free complete sup-semi-lattice on S”
  - algebraic theory
  - have V_x for any set X.
  - equations between the V_x
  - (Proper class of operations and proper class of equations, but up to equality only a set of operations for each set).

Extension to Rel

- What is the free complete sup-semilattice in Rel?
- Given R a relation between A and B, we need P(R) defined to be a relation between P(A) and P(B)
- U P(R) V iff
  - there is an S subset of R such that pi_o S = U and pi_1 S = V
  - iff forall u in U there is a v in V such that uRv, and for all v in V there is a u in U such that uRv.

Extension to Rel

- What is the free complete sup-semilattice in Rel?
- Given R a relation between A and B, we need P(R) defined to be a relation between P(A) and P(B)
- U P(R) V iff
  - there is an S subset of R such that pi_o S = U and pi_1 S = V
  - iff forall u in U there is a v in V such that uRv, and for all v in V there is a u in U such that uRv.

Extension to Rel

- U P(R) V
  - iff there is an S subset of R such that pi_o S = U and pi_1 S = V
  - iff forall u in U there is a v in V such that uRv, and for all v in V there is a u in U such that uRv.

Strong bisimulation

Given $S \overset{R}{\frown} S$ and $\alpha : A \times S \to P(S)$

$\alpha : A \times S \to P(S)$

then $\alpha \frown \alpha$ iff when $s \frown s$ then

$$\alpha(a, s) \frown \alpha(a, s)$$

i.e. when $s' \in \alpha(a, s)$ $(s \xrightarrow{a} s')$

then there is $s' \frown s' \in \alpha(a, s)$ $(s \xrightarrow{a} s')$

etc.

i.e. iff $\overset{R}{\frown}$ is a <u>strong bisimulation</u>.

Other forms of bisimulation

Other forms of bisimulation

- weak bisimulation
- branching bisimulation
- semi-branching bisimulation
- probabilistic bisimulation

Basic strategy

- There are other ways of modelling state transition systems.
- For weak bisimulation we are interested in systems that have silent internal computations.
- For branching bisimulation we have silent internal computations, but also synchronisation points.
- For probabilistic bisimulation we need models of stochastic processes.

State transition systems as monoid HM

- Our model only deals with single transitions.

$$A \times S \longrightarrow P(S)$$

or

$$A \longrightarrow [S \longrightarrow P(S)]$$

- We could ask it to account for sequences of transitions.

$$A^* \longrightarrow [S \longrightarrow P(S)]$$

- A monoid homomorphism

Weak bisimulation (Milner)

- Processes have silent tau actions, representing internal computation.

Definition 20. *(Milner (1989)) Let S be a labelled transition system for $A = L + \{\tau\}$, and $v \in L^*$, then*

$$s \overset{v}{\Rightarrow} s' \text{ iff there is a } w \in A^* = (L + \{\tau\})^* \text{ such that } v = \hat{w} \text{ and } s \overset{w}{\rightarrow} s'.$$

We can type $\Rightarrow$ *as* $\Rightarrow\ : [L^* \rightarrow [S \rightarrow \mathscr{P}\, S]]$, *and we refer to it as the* system derived from $\rightarrow$.

$$s \overset{a_1 \dots a_n}{\Longrightarrow} s' \quad \text{iff} \quad s \overset{\tau^*}{\longrightarrow} \overset{a_1}{\longrightarrow} \overset{\tau^*}{\longrightarrow} \overset{a_2}{\longrightarrow} \dots \overset{a_n}{\longrightarrow} \overset{\tau^*}{\longrightarrow} s'$$

Weak bisimulation (Milner)

- Processes have silent tau actions, representing internal computation.

Definition 20. *(Milner (1989)) Let S be a labelled transition system for $A = L + \{\tau\}$, and $v \in L^*$, then*

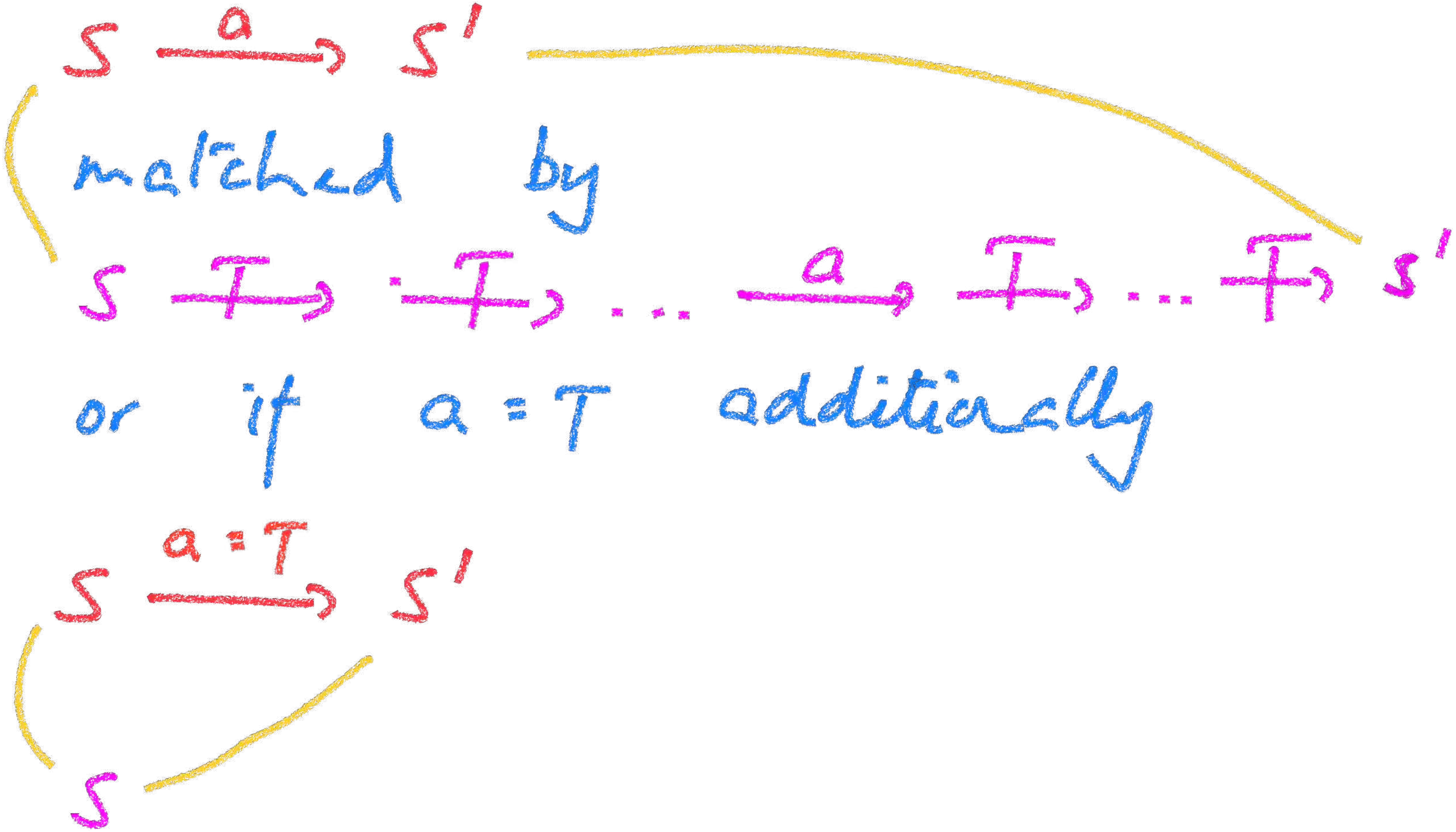
$$s \overset{v}{\Rightarrow} s' \text{ iff there is a } w \in A^* = (L + \{\tau\})^* \text{ such that } v = \hat{w} \text{ and } s \overset{w}{\rightarrow} s'.$$

We can type $\Rightarrow$ *as* $\Rightarrow : [L^* \to [S \to \mathscr{P}\, S]]$, *and we refer to it as the* system derived from $\to$.

Definition 21. *If S and T are two labelled transition systems for $A = L + \{\tau\}$, then a relation $R \subseteq S \times T$ is a* weak bisimulation *iff for all $a \in A = L + \{\tau\}$, whenever sRt*

- *for all $s \overset{a}{\rightarrow} s'$, there is t' such that $t \overset{a}{\Rightarrow} t'$ and $s'Rt'$*
- *and for all $t \overset{a}{\rightarrow} t'$, there is s' such that $s \overset{a}{\Rightarrow} s'$ and $s'Rt'$.*

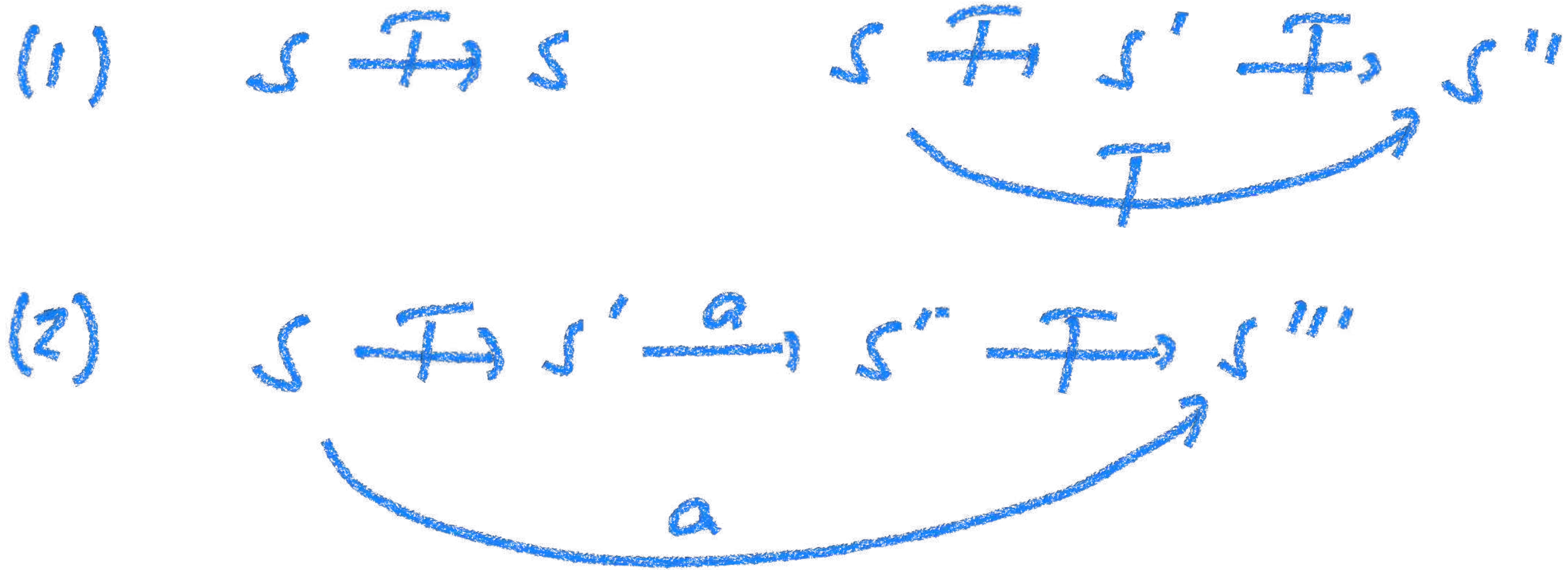
Weak Bisimulation



Weak bisimulation (1): saturation

Definition 26. *Let* $F : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}\, S]$ *be a transition system with internal action. We say that F is* saturated *if*

(1) $\mathrm{id} \leq F(\tau)$ *and* $F(\tau).F(\tau) \leq F(\tau)$ *and*
(2) *for all* $a \in L$, $F(\tau).F(a).F(\tau) \leq F(a)$



Weak bisimulation (1): saturation

Definition 26. *Let* $F : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}\, S]$ *be a transition system with internal action. We say that* F *is* saturated *if*

(1) $\mathrm{id} \leq F(\tau)$ *and* $F(\tau).F(\tau) \leq F(\tau)$ *and*
(2) *for all* $a \in L$, $F(\tau).F(a).F(\tau) \leq F(a)$

Proposition 27. *Suppose* $F : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}\, S]$ *and* $G : (L + \{\tau\}) \longrightarrow [T \to \mathscr{P}\, T]$ *are saturated transition systems with internal actions, then* $R \subseteq S \times T$ *is a weak bisimulation between the systems if and only if it is a strong bisimulation between them.*

Weak bisimulation (1): saturation

ORIGINAL MODEL $\alpha : (A + \{\tau\}) \times S \longrightarrow P(S)$

SATURATE IT $\overline{\alpha} : (A + \{\tau\}) \times S \longrightarrow P(S)$

- SAME STATE SPACE

- CHECK AN FOR STRONG BISIMULATION

Weak bisimulation (1): saturation

Original model : visible actions + τ actions

generates new model :

- sequences of visible actions

$$\alpha : A^* \longrightarrow [S \to P(S)]$$

But $\alpha(\varepsilon) \neq \lambda s.\{s\}$

So not a <u>monoid</u> HM

Just respects concatenation (semi-group).

Weak Bisimulation

- Model is a semi-group HM
- constructed from an original
- A relation between two such models is logical iff it is a weak bisimulation between the original models.

Weak bisimulation (2): lax HM

Definition 31. *A* lax transition system *on an alphabet L (not including an internal action τ) is a function* $F : L^* \longrightarrow [S \to \mathscr{P}\, S]$ *such that:*

(1) $\mathrm{id} \leq F(\varepsilon)$ *(reflexivity)*
(2) $F(vw) = F(v).F(w)$ *(composition)*

Definition 32. *Let* $F : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}\, S]$ *be a transition system with internal action, then its* laxification $\hat{F} : L^* \longrightarrow [S \to \mathscr{P}\, S]$ *is the lax transition system defined by:*

(1) $\hat{F}(\varepsilon) = F(\tau)^*$
(2) $\hat{F}(a) = F(\tau)^*.F(a).F(\tau)^*$*, for any* $a \in L$.
(3) $\hat{F}(vw) = \hat{F}(v).\hat{F}(w)$.

Lemma 35. *Suppose* $F : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}\, S]$ *and* $G : (L + \{\tau\}) \longrightarrow [T \to \mathscr{P}\, T]$ *are transition systems with internal actions, and* $R \subseteq S \times T$. *Then the following are equivalent:*

(1) R is a weak bisimulation between F and G
(2) $(\hat{F}, \hat{G}) \in [\mathrm{Id}_{L^*} \to [R \to \mathscr{P}\, R]]$
(3) R is the state space of a lax transition system in Rel *whose first projection is* $\hat{F}$ *and whose second is* $\hat{G}$.

Weak bisimulation (2): lax HM

Original model : visible actions + τ actions

generates new model :

- sequences of visible actions

  $\alpha : A^* \longrightarrow [S \to P(S)]$

But $\alpha(\varepsilon) \neq \lambda s. \{s\}$

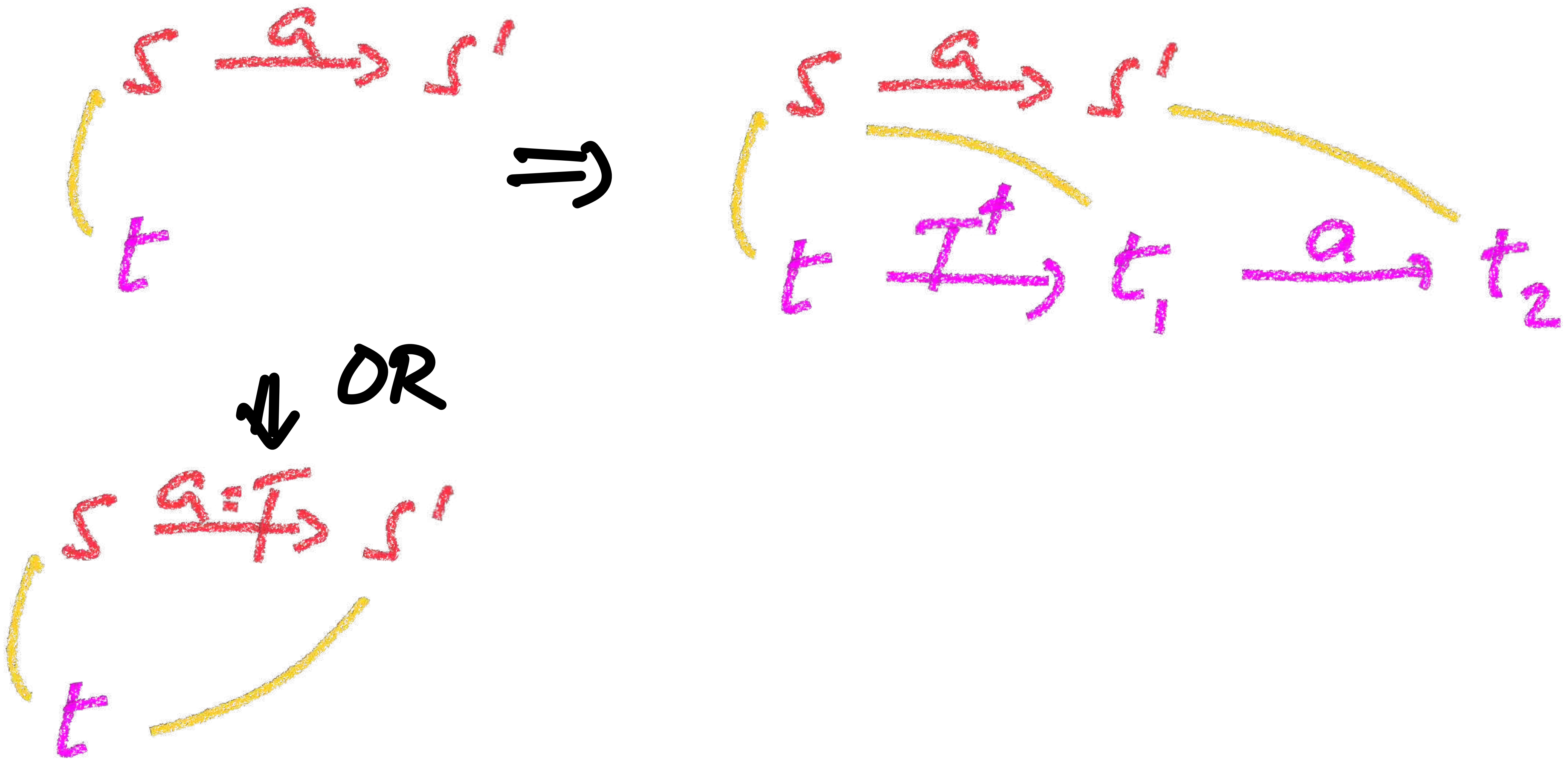
So not a monoid HM

Just respects concatenation (semi-group).

Branching bisimulation

Definition 36. *A relation* $R \subseteq S \times T$ *is called a* branching bisimulation *if and only if whenever* sRt*:*

- $s \xrightarrow{a} s'$ *implies* $\big((\exists t_1, t_2 \in T.\, t \xrightarrow{\tau^*} t_1 \xrightarrow{a} t_2 \wedge sRt_1 \wedge s'Rt_2)$ *or* $(a = \tau \wedge s'Rt)\big)$,
- $t \xrightarrow{a} t'$ *implies* $\big((\exists s_1, s_2 \in S.\, s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2 \wedge s_1Rt \wedge s_2Rt')$ *or* $(a = \tau \wedge sRt')\big)$.



Branching bisimulation

Definition 36. *A relation* $R \subseteq S \times T$ *is called a* branching bisimulation *if and only if whenever* sRt*:*

- $s \xrightarrow{a} s'$ *implies* $\big((\exists t_1, t_2 \in T.\, t \xrightarrow{\tau^*} t_1 \xrightarrow{a} t_2 \wedge sRt_1 \wedge s'Rt_2)$ *or* $(a = \tau \wedge s'Rt)\big)$,
- $t \xrightarrow{a} t'$ *implies* $\big((\exists s_1, s_2 \in S.\, s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2 \wedge s_1Rt \wedge s_2Rt')$ *or* $(a = \tau \wedge sRt')\big)$.

$$\overline{F}^b : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}(S \times S)]$$

$$\overline{F}^b as = \{(s_1, s_2) \in S \times S \mid (s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2) \text{ or } (a = \tau \text{ and } s = s_1 = s_2)\}.$$

Branching bisimulation

Definition 36. *A relation* $R \subseteq S \times T$ *is called a* branching bisimulation *if and only if whenever* sRt*:*

- $s \xrightarrow{a} s'$ *implies* $\big((\exists t_1, t_2 \in T.\, t \xrightarrow{\tau^*} t_1 \xrightarrow{a} t_2 \wedge sRt_1 \wedge s'Rt_2)$ *or* $(a = \tau \wedge s'Rt)\big)$,
- $t \xrightarrow{a} t'$ *implies* $\big((\exists s_1, s_2 \in S.\, s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2 \wedge s_1Rt \wedge s_2Rt')$ *or* $(a = \tau \wedge sRt')\big)$.

$$\overline{F}^b : (L + \{\tau\}) \longrightarrow [S \to \mathscr{P}(S \times S)]$$

$$\overline{F}^b as = \{(s_1, s_2) \in S \times S \mid (s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2) \text{ or } (a = \tau \text{ and } s = s_1 = s_2)\}.$$

Theorem 39. Let $R \subseteq S \times T$. Then R is a branching bisimulation if and only if $(\overline{F}^b, \overline{G}^b) \in [\mathrm{Id}_{L+\{\tau\}} \to [R \to \mathscr{P}(R \times R)]]$.

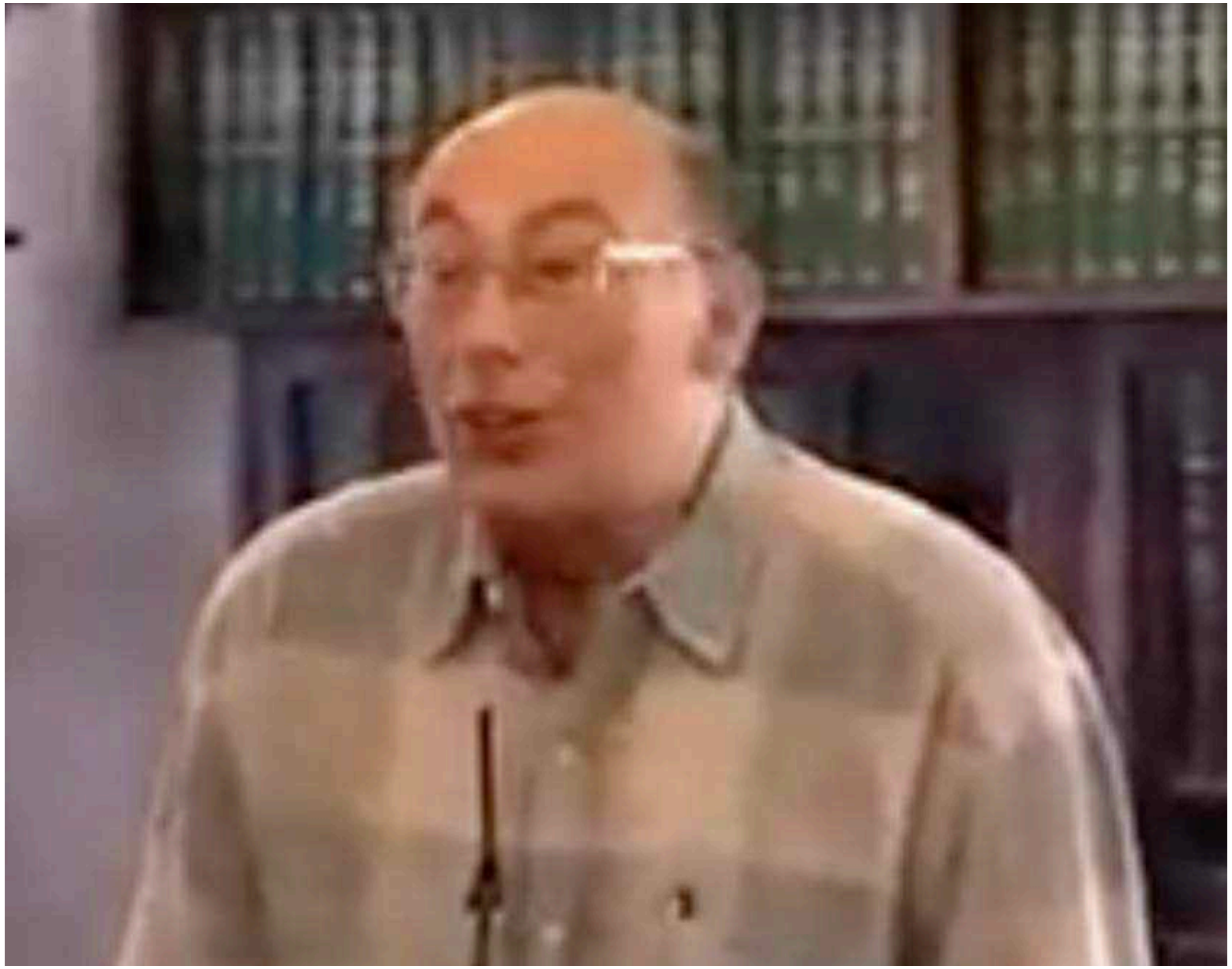
Probabilistic Bisimulation

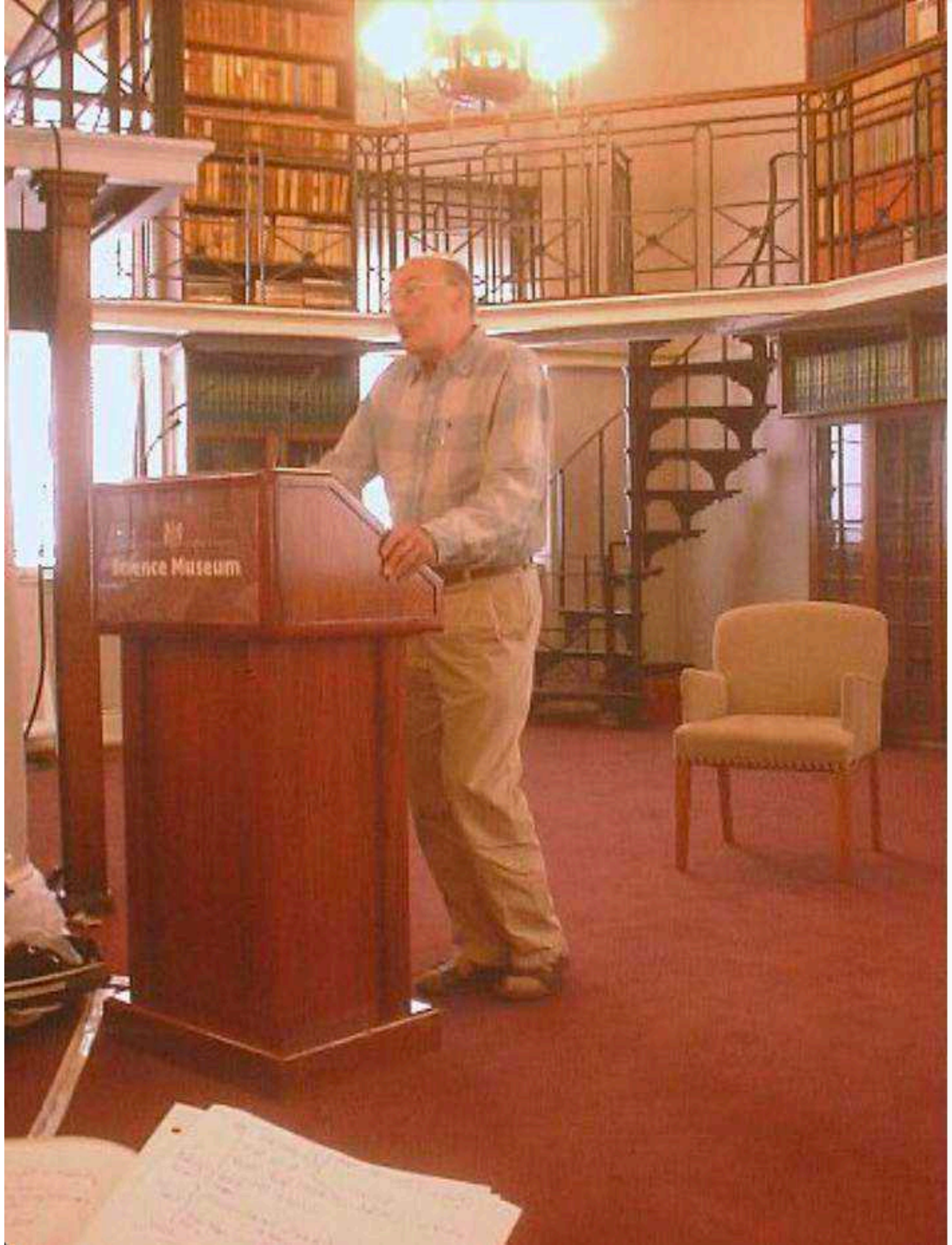
- Need to model stochastic processes not just state transition.

- Idea (Lawvere, Giry) process is given by a form of “Markov kernel”: an operator that relates a probability space on the domain to a measure space on the codomain and gives the probability of a transition function taking a value in a given measurable set.

- Notion of bisimulation arising from logical relations is strong probabilistic bisimulation.

- Have to work harder to get close to Pi-bisimulation.




Science Museum


Peter Landin Teaching Rooms
No smoking


Francis Bancroft Building
Francis Bancroft Building


Peter Landin Teaching Rooms
No smoking
CCT

References

Plotkin, Gordon. *Lambda-definability and logical relations*. Edinburgh University, 1973.

Milne, Robert. "The Formal Semantics of Computer Languages and their Implementations." PhD diss., University of Cambridge, 1974.

Hermida, Claudio, Uday S. Reddy, and Edmund P. Robinson. "Logical relations and parametricity–a Reynolds programme for category theory and programming languages." *Electronic Notes in Theoretical Computer Science* 303 (2014): 149-180.

Hermida, Claudio, Uday S. Reddy, and Edmund P. Robinson. "Deriving Logical Relations from Interpretations of Predicate Logic." *Electronic Notes in Theoretical Computer Science* 347 (2019): 241-259.

Hermida, Claudio, Uday Reddy, Edmund Robinson, and Alessio Santamaria. "Bisimulation as a logical relation." *Mathematical Structures in Computer Science* 32, no. 4 (2022): 442-471.