Categorical structures for system modelling

José Luiz Fiadeiro

University of Leicester

Disclaimer

- I am not a mathematician by training but a practising computer scientist. This seminar presents (some) elements of category theory not as mathematical objects but as they can apply to computer science.
- This is not presented as a crash course in category theory, but as an introduction to (part of) what it offers for modelling software intensive systems. It should provide enough background and awareness to allow you to pick up a
- Full definitions, results and proofs
  "Categories for Software Engineers
- But other books exist... I tend to use "Categories for the Working Mathematician" by S. MacLane, Springer 1971.

It is a personal view and trajectory: "**my toolbox**"

Many of the "must haves" of CT will not be covered

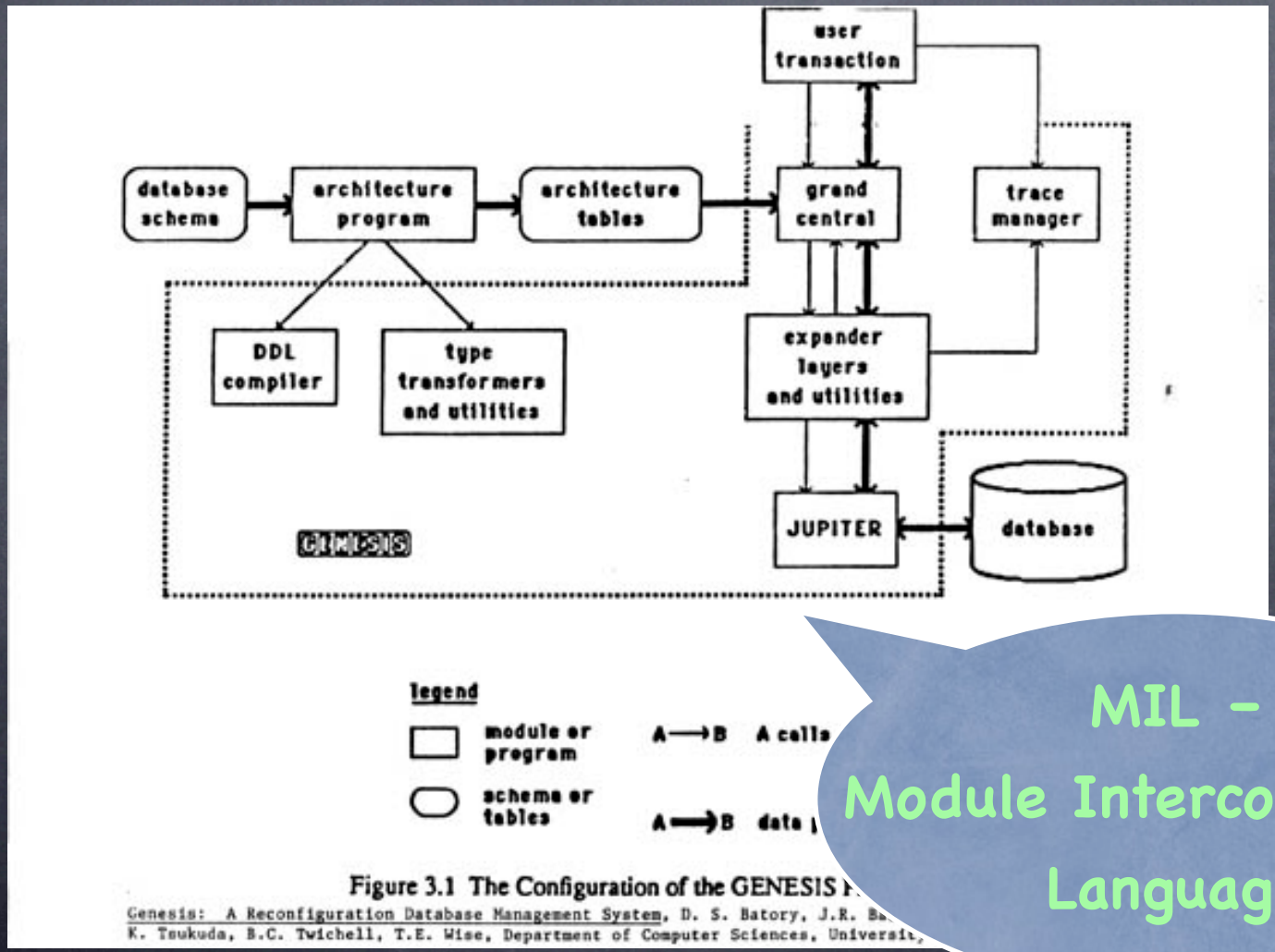
- **Modelling computations**
  - "...approach: CS as a discipline concerned with the ...ional phe... ...cus is on the ...ation...
  - ...cor...
  - ...other ... ... code, ...cations.
- **Modelling systems**
  - "Putting theories together to make specifications".
  - "Given a category of widgets, the operation of putting a system of widgets together to form a super-widget corresponds to taking a colimit of the diagram of widgets that shows how to interconnect them".

Run-time **architecture;**
Dynamic **reconfiguration**

CT as the mathematics of complex systems

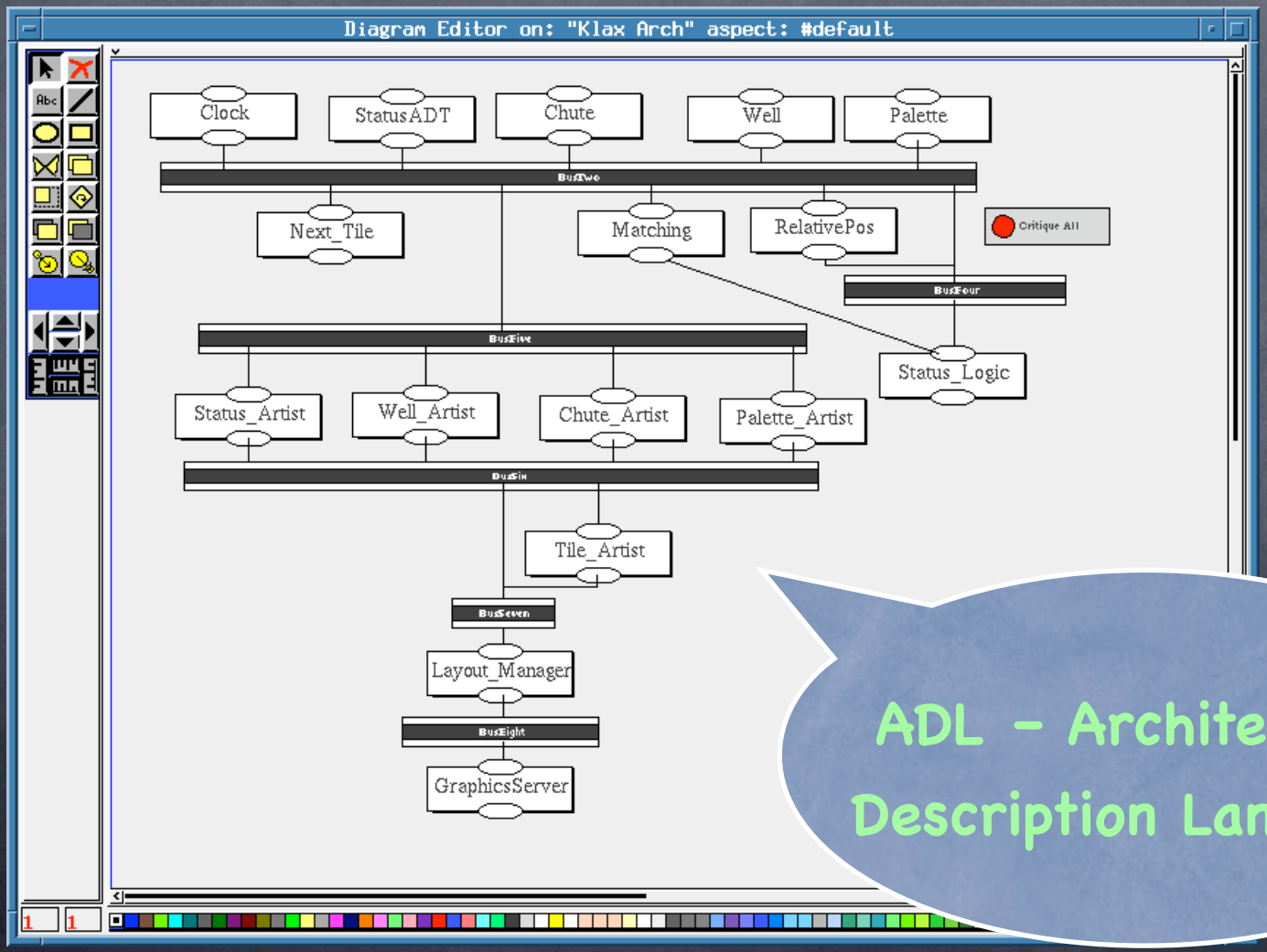
- **Typical areas of CS that we will subsume**
  - ADT (algebraic specifications)
    - Goguen and Burstall (Institutions)
    - "CASL"
  - Concurrency
    - Nielsen, Sassone, Winskel
    - Montanari, Corradini, ...
  - Parallel program design
    - Superposition (Katz, Chandy&Misra, Forman&Francez)
  - Software architecture
    - Components and connectors (Garlan; Kramer&Magee)

Typical structures in CS that we will address

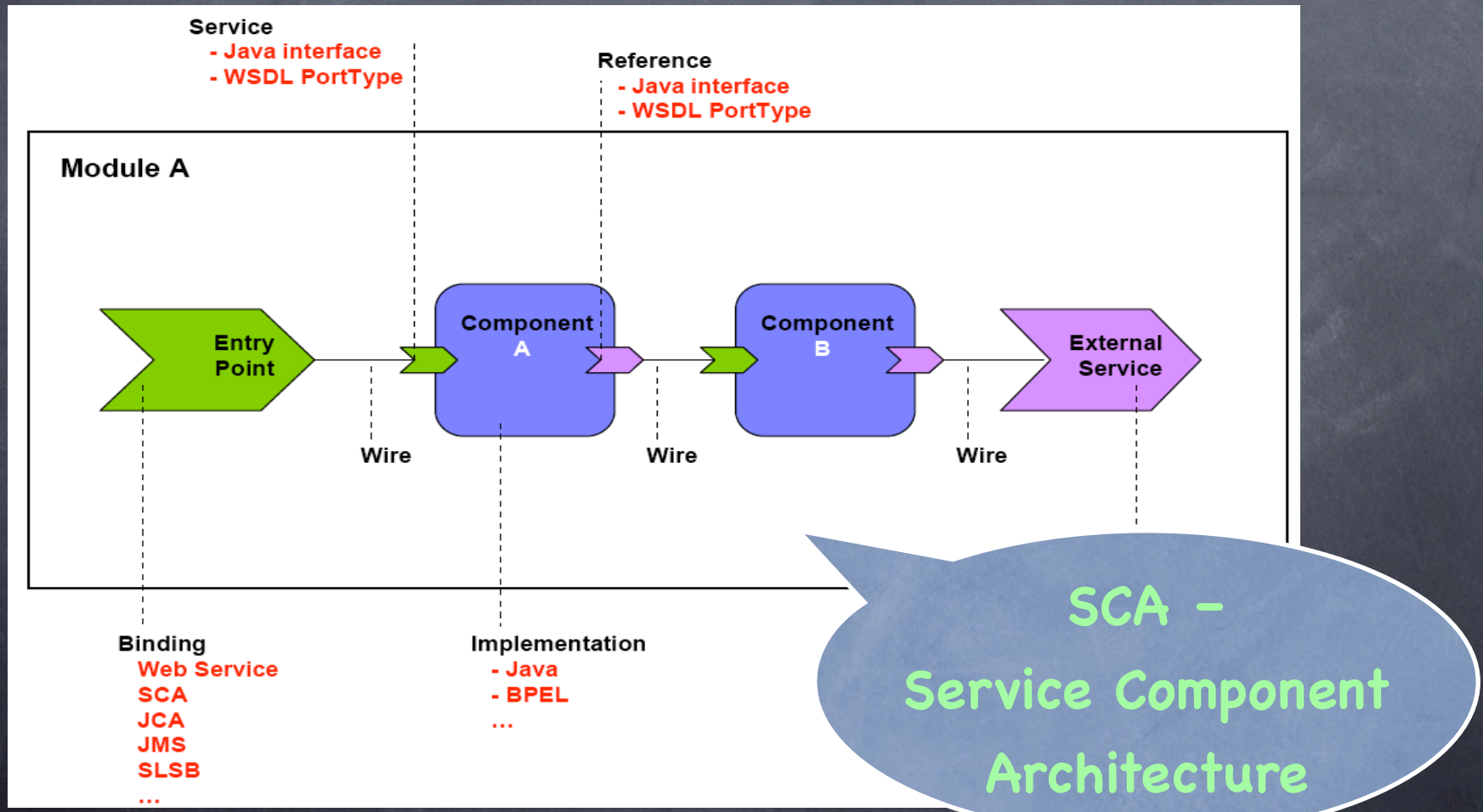


MIL – Module Interconnection Language

Typical structures in CS that we will address



Typical structures in CS that we will address

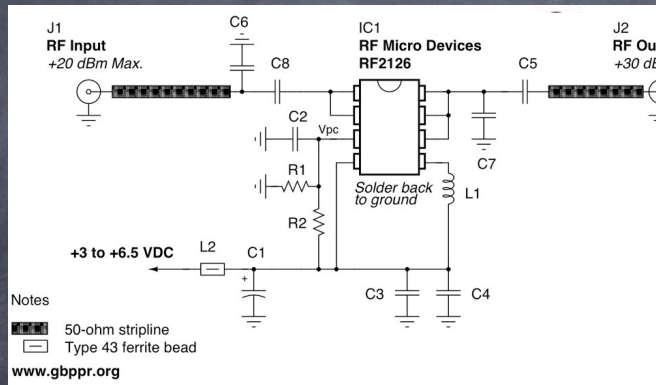


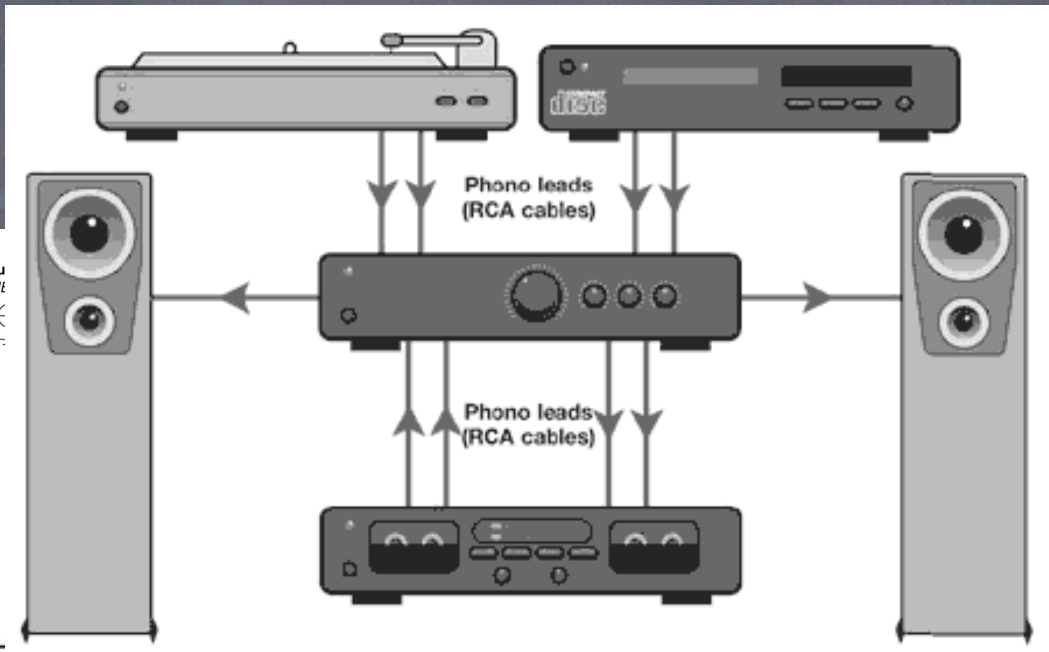
- **FAQs**
  - What is CT good for? When should I use it? Why should I bother learning CT?
- **An alternative toolbox to Set Theory**
  -

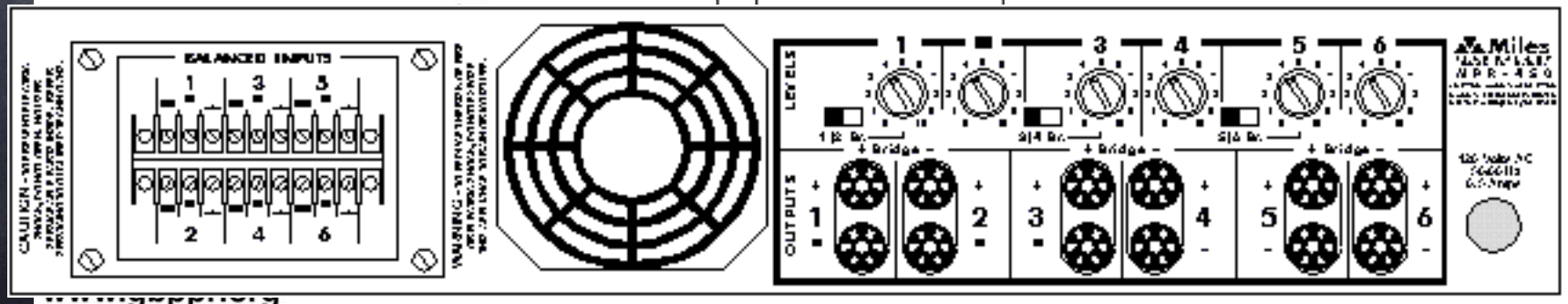
$\in$	$\rightarrow$
$=$	$\equiv$
Explicit	Implicit
Internal	External
White-box	Black-box

A power amplifier

In Set Theory







- **The "social metaphor"**
  - "The meaning of things lies not in the things themselves, but in our attitude towards them."
    Antoine de Saint-Exupéry
- **Sets in CT**
  - What is the social life of sets?
  - Who is the empty set?
  - Who are the singletons?
  - Where are the elements?
  - What is an injective function?

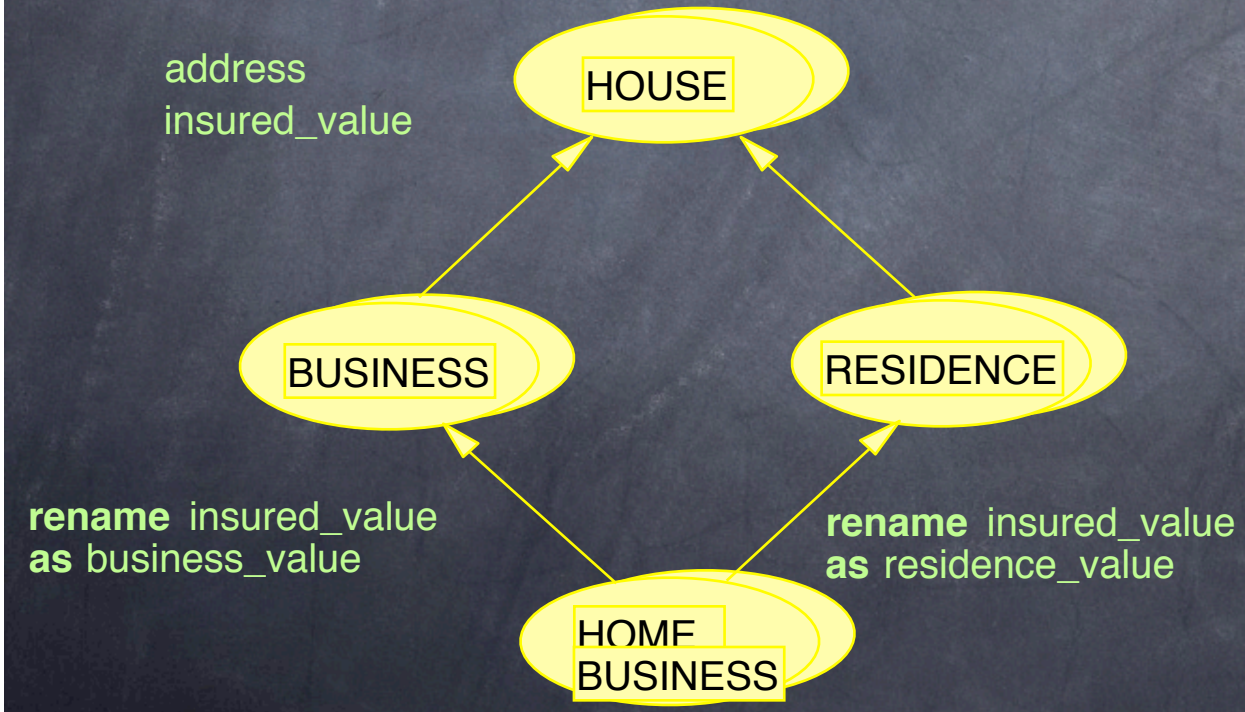
- **A Graph consists of**
  - a collection G_0 is of nodes
  - a collection G_1 of arrows
  - two maps src and trg from arrows to nodes (the source and target of the arrow).

We usually write $f:x \rightarrow y$ to indicate that $src(f)=x$ and $trg(f)=y$.

Categories work on a less immediate social relationship given by the paths of a graph.
$\langle f_1,\ldots,f_n\rangle : x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_{n+1} \in G_n$

Class inheritance hierarchies

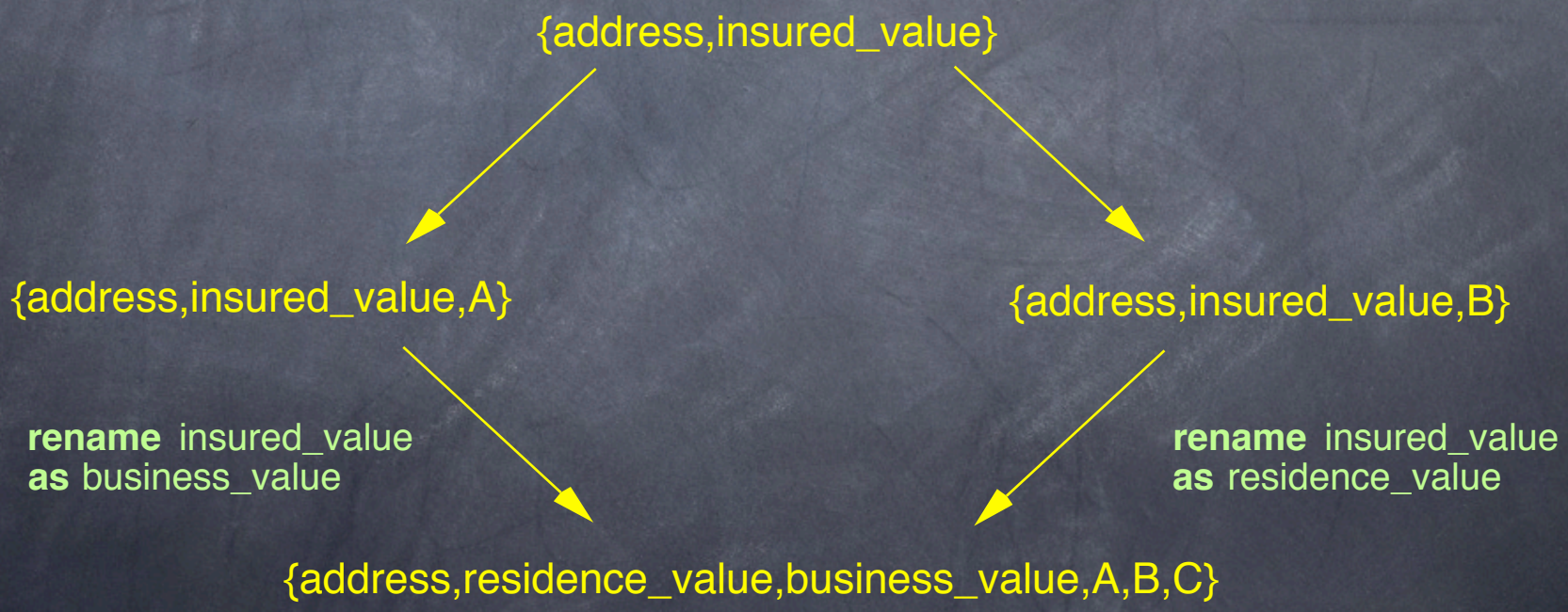
- These are graphs whose nodes are object classes and for which the existence of an arrow between two nodes (classes) means that the source class inherits from the target class



In class inheritance hierarchies, there exists at most one arrow between two nodes. However, arrows can carry more information.

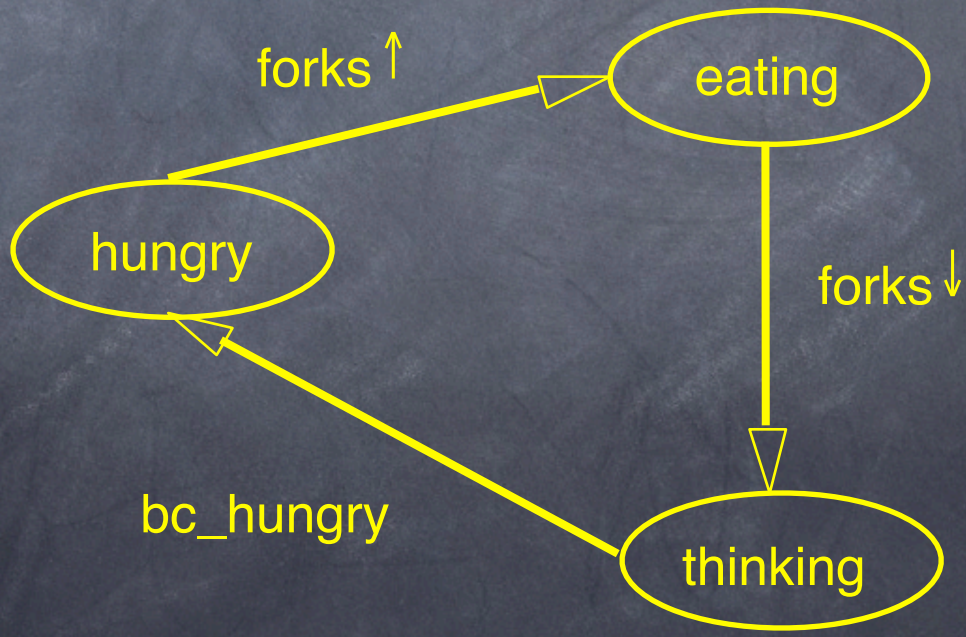
Class inheritance hierarchies

- Class inheritance graphs induce mappings between the sets of features of each class



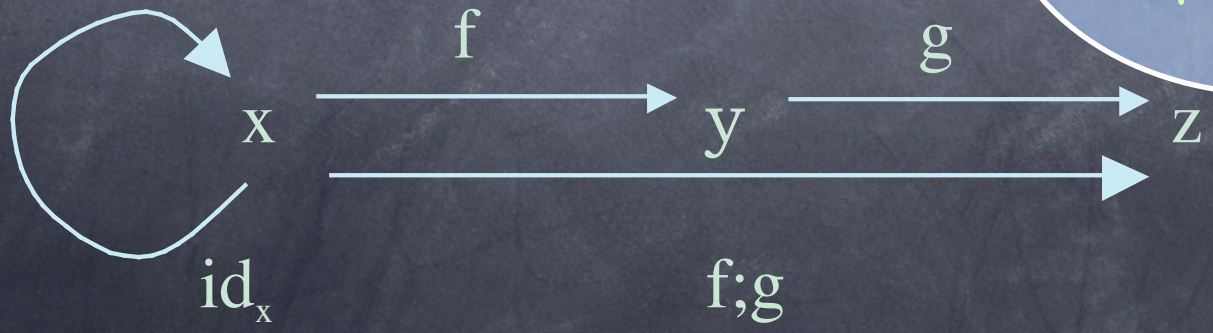
Transition systems

- Every transition system constitutes a graph whose nodes are the states and whose arrows are the transitions



- **A Category consists of**
  - a graph G
  - an associative composition law $;:G_2 \to G_1$
    $(f;g);h=f;(g;h)$.
  - an identity map $id:G_0 \to G_1$
    If $f:x \to y$, $id_x;f = f;id_y = f$.

many authors use functional composition ∘



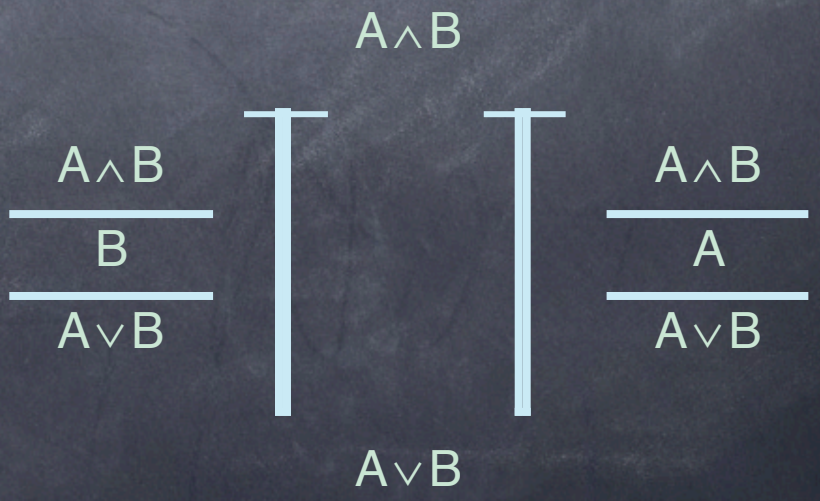
  - We denote by $hom_C(x,y)$ the morphisms from x to y.

Some mathematical structures

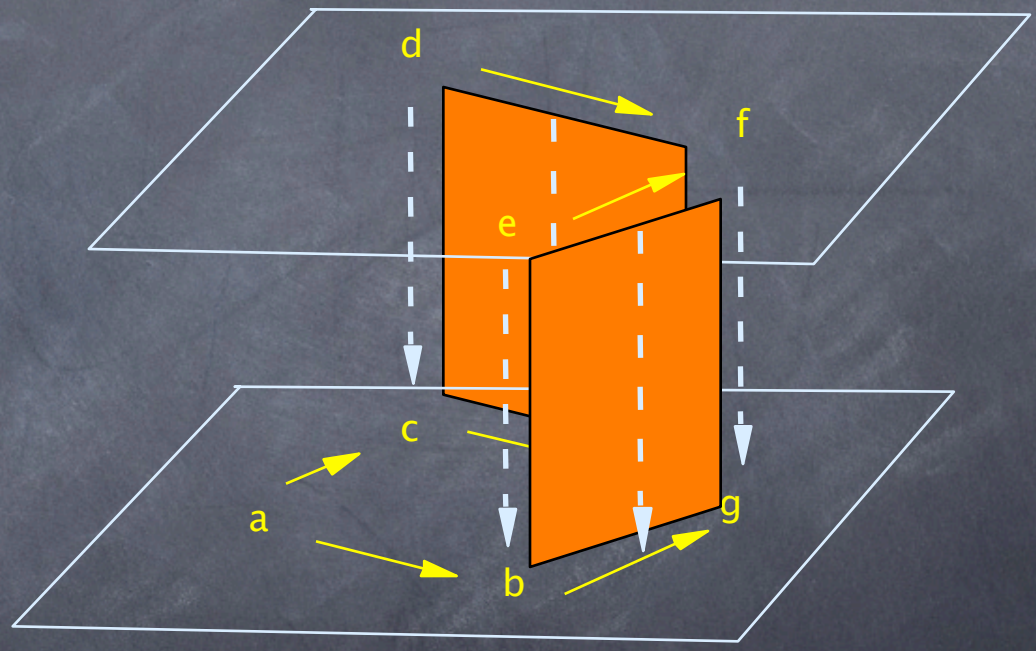
- Every pre-order $\langle S,\leq\rangle$ defines a category: the objects are the elements of S and there is an arrow between any two objects x and y iff $x\leq y$; reflexivity provides identities; transitivity ensures composition.
- Every monoid $\langle M,*,1\rangle$ defines a category: there is only one object, which we can denote by "•", and the morphisms $m:\bullet\rightarrow\bullet$ are the elements of M; 1 provides the identity and * is the composition law.
- Every set S determines a category by defining, for every element s of S an identity $\bullet_s:s\rightarrow s$.

- **SET**
  - sets and total functions
- **LOGI**
  - sentences and logical entailment $A \vdash B$

- **PROOFS**
  - sentences and proofs

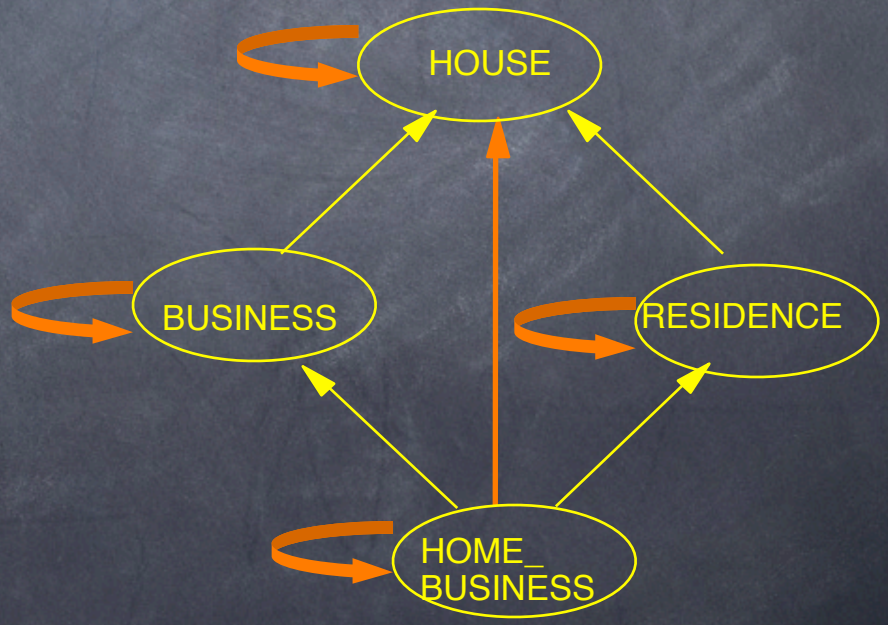


- **GRAPH**
  - a morphism $\varphi:G\to H$ is a pair of maps $\varphi_0:G_0\to H_0$, $\varphi_1:G_1\to H_1$ such that each arrow $f:x\to y$ is mapped to $\varphi_1(f):\varphi_0(x)\to\varphi_0(y)$.



Ancestor

- In Eiffel, given an inheritance graph G between classes, the category **ancestor**(G) is generated by completing the graph with the arrows that result from **reflexivity** (identities) and **transitivity** (compositions).



- **Category generated by a graph**
  - Every graph G generates a category **cat**(G) as follows:
    - **objects**: nodes
    - **arrows**: paths
    - **identities**: empty paths;
    - **composition**: path concatenation.
  - For instance, every transition system T generates the category Run(T) of its runs.

Diagram in a category

- Let C be a category and I a graph. A diagram in C with shape I is a morphism $\delta: I \to graph(C)$.
- A diagram $\delta: I \to C$ is said to commute iff, for every pair x,y of nodes and every pair of paths $w=u_1...u_m$, $w'=v_1...v_n$ from x to y in I, $\delta u_1;...; \delta u_m = \delta v_1;...;\delta v_n$

Adding structure

- The most typical way of building a new category consists of adding structure to the objects of a given category (or a subset thereof).
- The morphisms of the new category are the morphisms of the old category that "preserve" the additional structure.

Automata

- Every object is a (deterministic) automaton consisting of of an input set X, a state set S, an output set Y, a transition function f: $X \times S \to S$, an initial state $s_0 \in S$ and an output function g: $S \to Y$.
- A morphism from $\langle X,S,Y,s_0,f,g\rangle$ to $\langle X',S',Y',s'_0,f',g'\rangle$ consists of three functions $\langle h:X\to X', i:S\to S', j:Y\to Y'\rangle$ such that

1. $i(s_0)=s'_0$.
2. $f;i=h\times i;f'$.
3. $g;j=i;g'$.

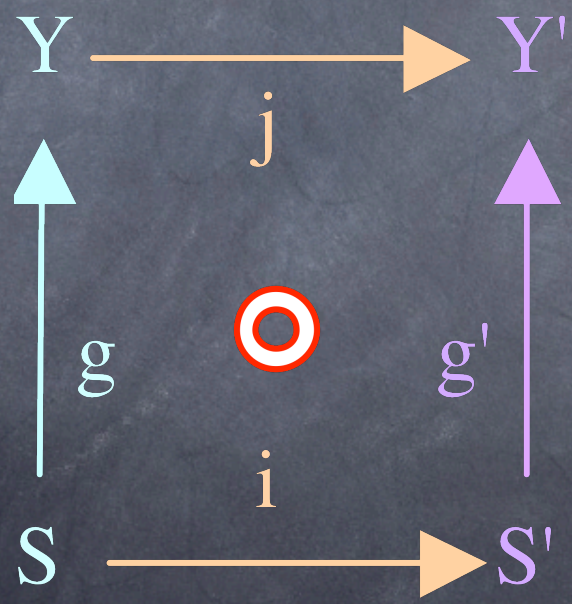
The initial states "agree"

The transition functions "agree"

The output functions "agree"

Expressing the properties with diagrams

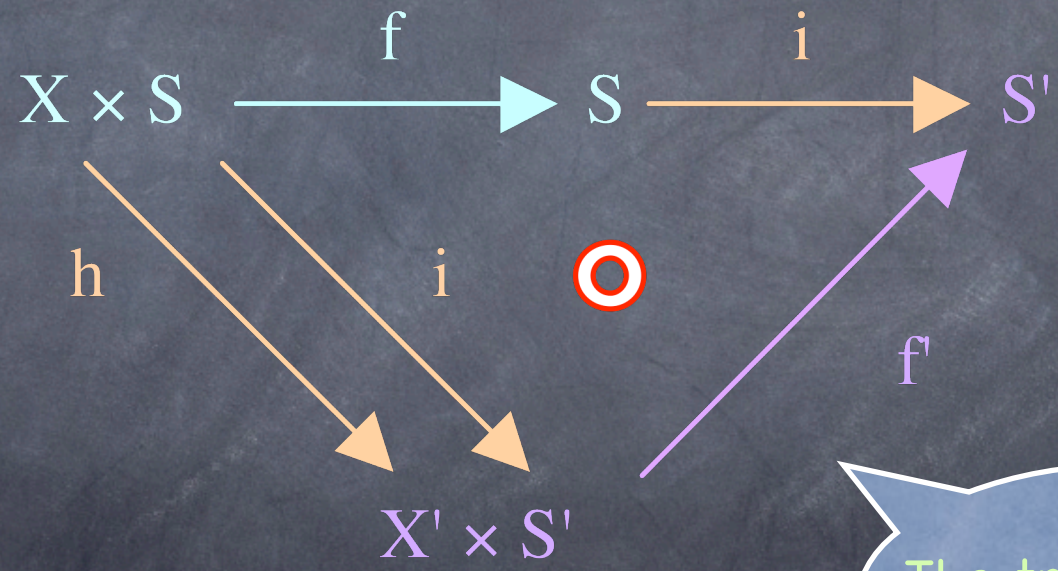
3. g;j=i;g'



The output functions "agree"

Expressing the properties with diagrams

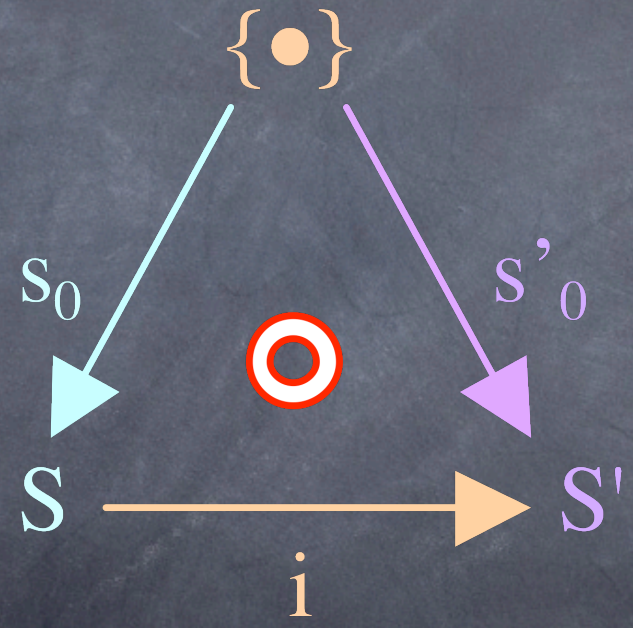
2. f;i=h×i;f'



The transition functions "agree"

Expressing the properties with diagrams

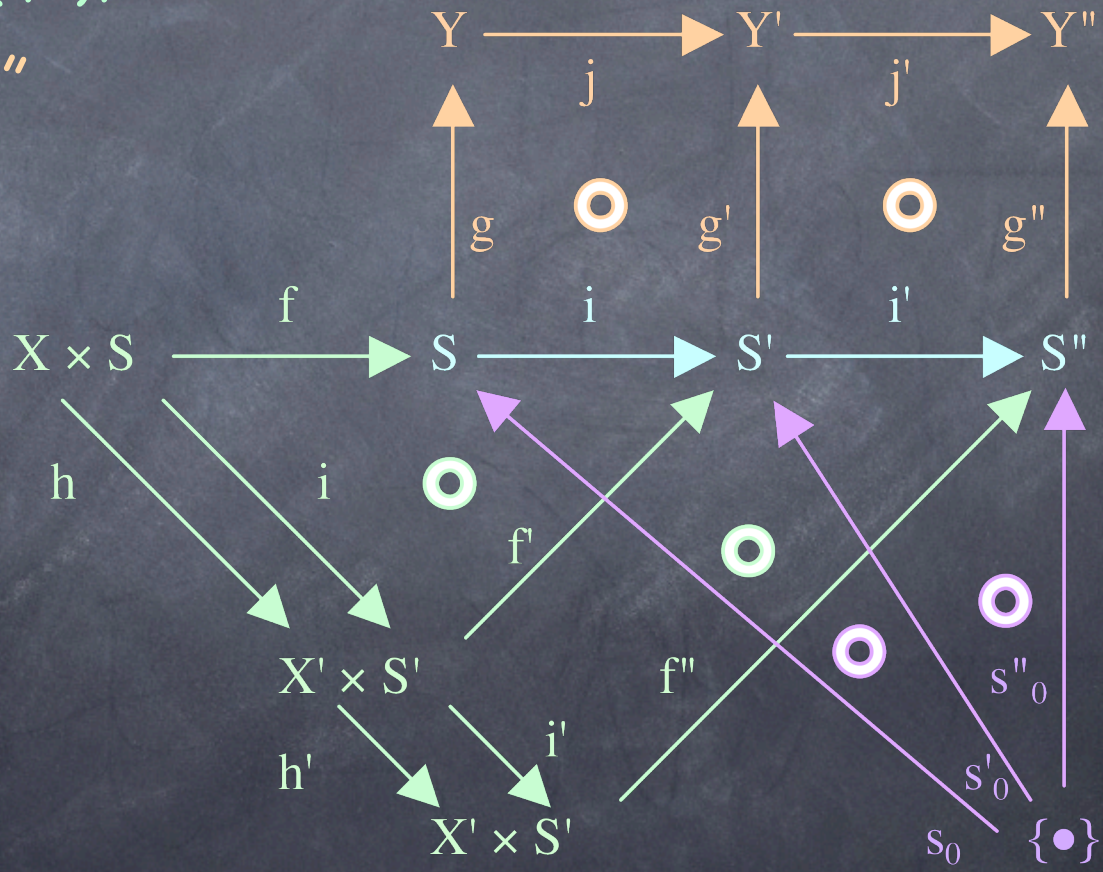
1. $i(s_0)=s'_0$



The initial states "agree"

Proving composition with diagrams

- f;(i;i')=(h;h)'x(i;i');f"
- g;(j;j')=(i;i');g"
- (i;i')(s0)=s"0



Pointed sets $SET_{\perp}$

- Its objects are the pairs $\langle A, \perp_A \rangle$, where A is a set and $\perp_A$ is an element of A call… d elemen…
- The mo… the total f…
- The con… pr… "inherited…

Pointed se… ng the alphabets o…

an event of the en…ironment… eve…

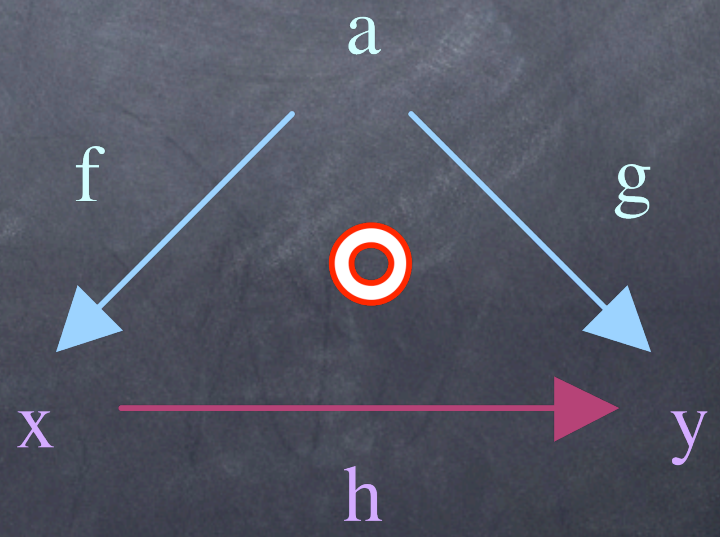
every morphi… identifie… in which … embe… compo… sou…

Each element represents an event whose occurrence may be witnessed during the … he … s.

The environment of the system is part of the environment of every component.

Comma Categories

- Given a category C and an object a:C we define the category of "objects under a" – a↓C
  - objects are the pairs <f,x>, where f:a→x is a morphism in C.
  - the morphisms between f:a→x and g:a→y are all the morphisms h:x→y such that f;h=g.



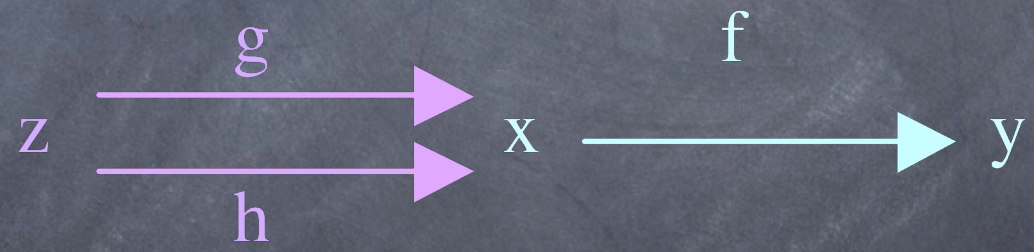
Isomorphisms

- A morphism $f:x \to y$ is an isomorphism iff there is a morphism $g:y \to x$ such that: $f;g=id_x$ and $g;f=id_y$.
- In these conditions, x and y are said to be isomorphic.

What are the isomorphisms in the categories identified so far?

Mono/Epimorphisms

- A morphism f:x→y is a monomorphism (or a mono or monic) iff, for every pair of morphisms g,h:z→x, g;f=h;f implies g=h.
  - monos do not interfere with interactions at the source



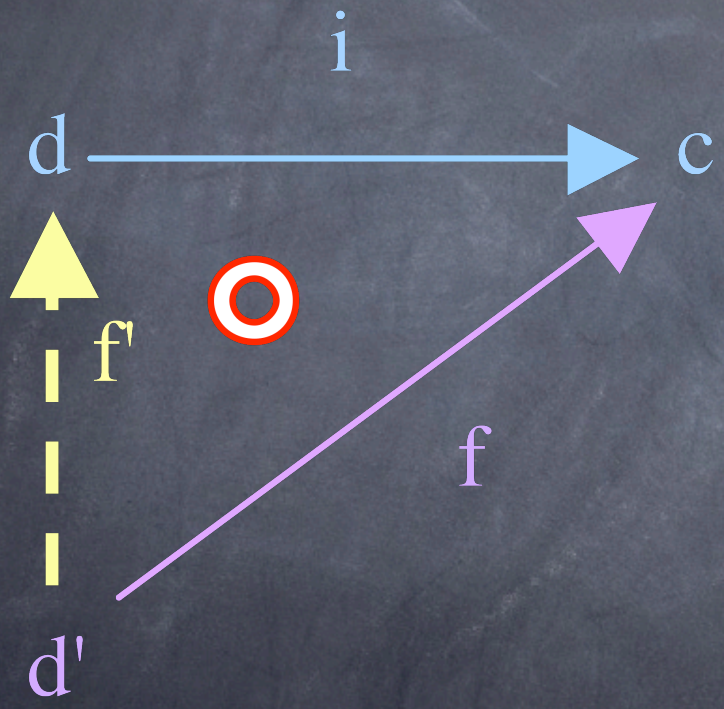
- A morphism f:x→y is an epimorphism (or an epi or epic) iff, for every pair of morphisms g,h:y→z, f;g=f;h implies g=h.

Subcategories

- A category D is a subcategory of a category C iff
  - $D_0 \subseteq G_0$
  - $hom_D(x,y) \subseteq hom_C(x,y)$ for all objects x and y of D
- A subcategory D of C is full iff $hom_D(x,y)=hom_C(x,y)$ for all objects x and y of D
  - This means that the structure of the objects is preserved (same social life but among a smaller population)
  - SET is a subcategory of the category PAR of sets and partial functions, but not a full one.
  - The category fSET of finite sets is a full subcategory of SET.

- **REACH⊆AUTOM**
  - Reachable automata give rise to a full subcategory **REACH** of **AUTOM**.
    - their states can be reached through the transition function from the initial state and some input sequence
  - Every automaton $A=\langle X,S,Y,s_0,f,g\rangle$ is related to a "canonical" reachable automaton $R=\langle X,S_R,Y,s_0,f_R,g_R\rangle$ through a morphism $c:R\to A$ such that, given any reachable automaton R' and simulation $h:R'\to A$, there is a unique morphism of reachable automata $h':R'\to R$ such that $h=h';c$. This means that R is the reachable automaton that is "closest" to A.

(Co)Reflections



Let D be a subcategory of a category C.

- A D-coreflection for c:C is i:d→c for d:D such that, for any f:d'→c where d':D, there is a unique f':d'→d in D such that f=f';i

- **(Co)Reflective Subcategories**
  - We say that D is a coreflective subcategory of C iff every C-object admits a D-coreflection.
  - Co-reflections are essentially "unique": if $i:d \to c$ and $j:e \to c$ are both D-coreflections for c, then there is a D-isomorphism $f:e \to d$ such that $j=f;i$.
  - If $i:d \to c$ is a D-coreflection for c, and $f:e \to d$ is a D-isomorphism, then $f;i:e \to c$ is also a D-coreflection for c.
  - Let D be a coreflective subcategory of a category C. Then, D is a full subcategory of C iff, for every D-object d, id_d is a D-coreflection for d.

- **Initial objects in a category**
  - An object x is initial iff, for every object y, there is a unique morphism from x to y.
  - Any two initial objects are isomorphic
  - We usually denote "the" initial object, if one exists, by 0. The unique morphism from 0 into an object a is denoted by 0_a:0→a

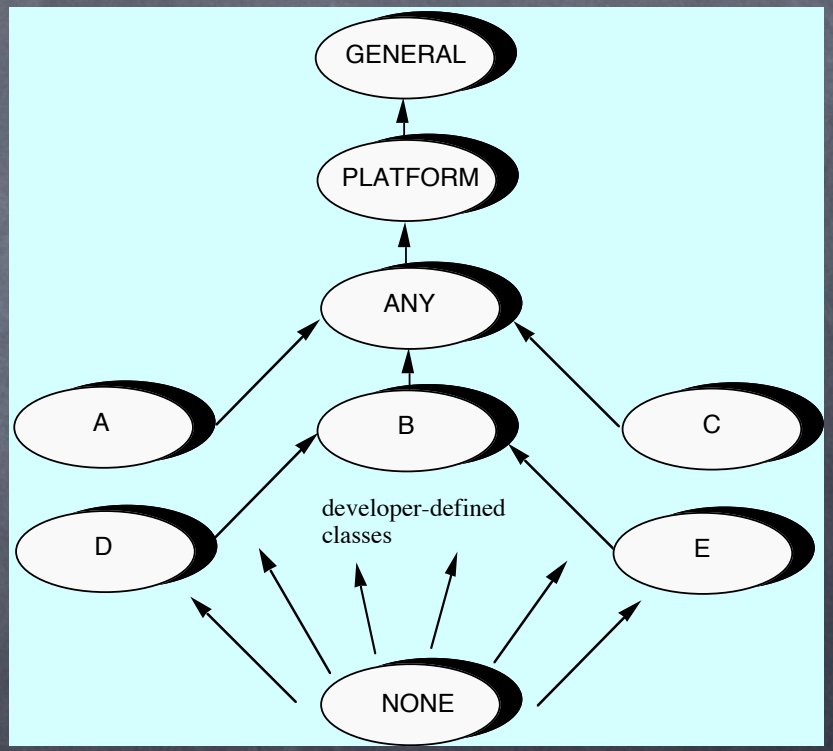
Which are the initial objects of the categories identified so far?

- **Terminal objects in a category**
  - An object x is terminal iff, for every object y, there is a unique morphism from y to x.
  - Any two terminal objects are isomorphic
  - We usually denote "the" terminal object, if one exists, by 1. The unique morphism from an object a into 1 is denoted by 1_a:a→1

Which are the terminal objects of the categories identified so far?

Eiffel's Inheritance Structure

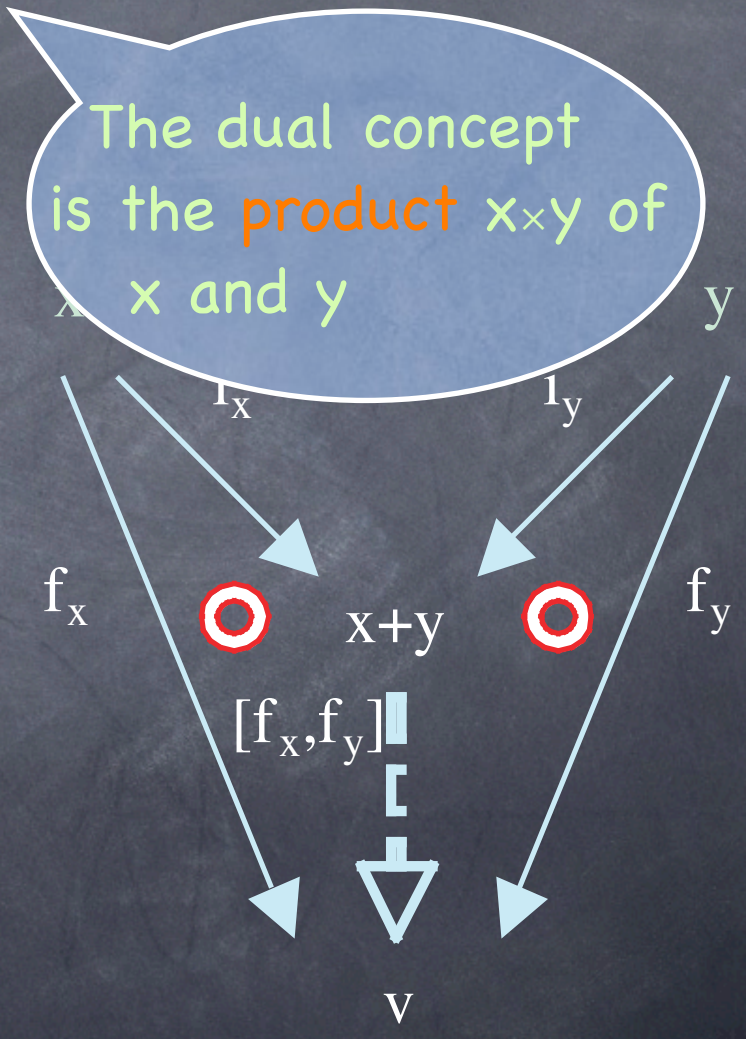
- Eiffel has an initial object – the class NONE that inherits from every other class.
- And also a terminal object – the class GENERAL from which every other class inherits.



- **Dual of a category C**
  - C^{op} has the same objects and arrows as C.
  - The arrows of C^{op} go in the reverse direction: if $f:A \rightarrow B$ in C then $f:B \rightarrow A$ in C^{op}.
  - Arrow composition is in the reverse direction: f;g in C^{op} is g;f in C.
- **Duality principle**
  - Every concept has a dual, which is the concept that is obtained by reversing the direction of the arrows.
  - Every result in category theory has a dual that holds for the dual concepts.

Sum in a category C

- An object z is a sum (or coproduct) of x and y with injections i_x:x→z and i_y:y→z iff for any object v and pair of morphisms f_x:x→v, f_y:y→v there is a unique morphism k:z→v, usually denoted $[f_x,f_y]$ such that $i_x;k=f_x$ and $i_y;k=f_y$.
- If a sum of x and y exists, it is unique up to isomorphism and is denoted by x+y.



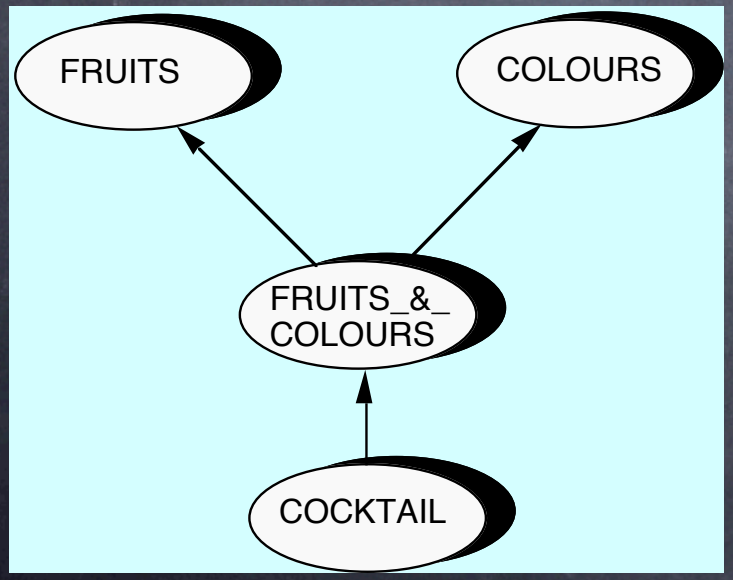
- **Sum in PROOF**
  - Sums correspond to disjunctions.
  - The categorical definition of sum actually provides, through the morphisms the usual rules of natural deduction for OR introduction and elimination.

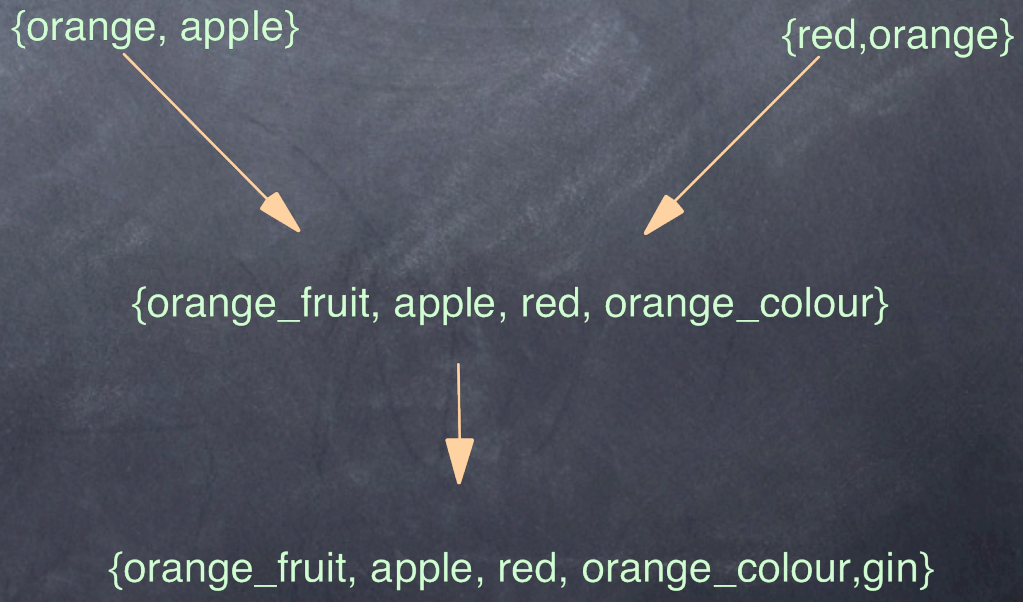
$$i_A : A \to A \vee B \qquad i_B : B \to A \vee B$$

$$\frac{f_A : A \to C,\ f_B : B \to C}{[f_A, f_B] : A \vee B \to C}$$

Eiffel's Inheritance Structure

- Eiffel has a notion of inheritance without name clashes that corresponds to a product.
- The corresponding construction on sets of features corresponds to a sum.

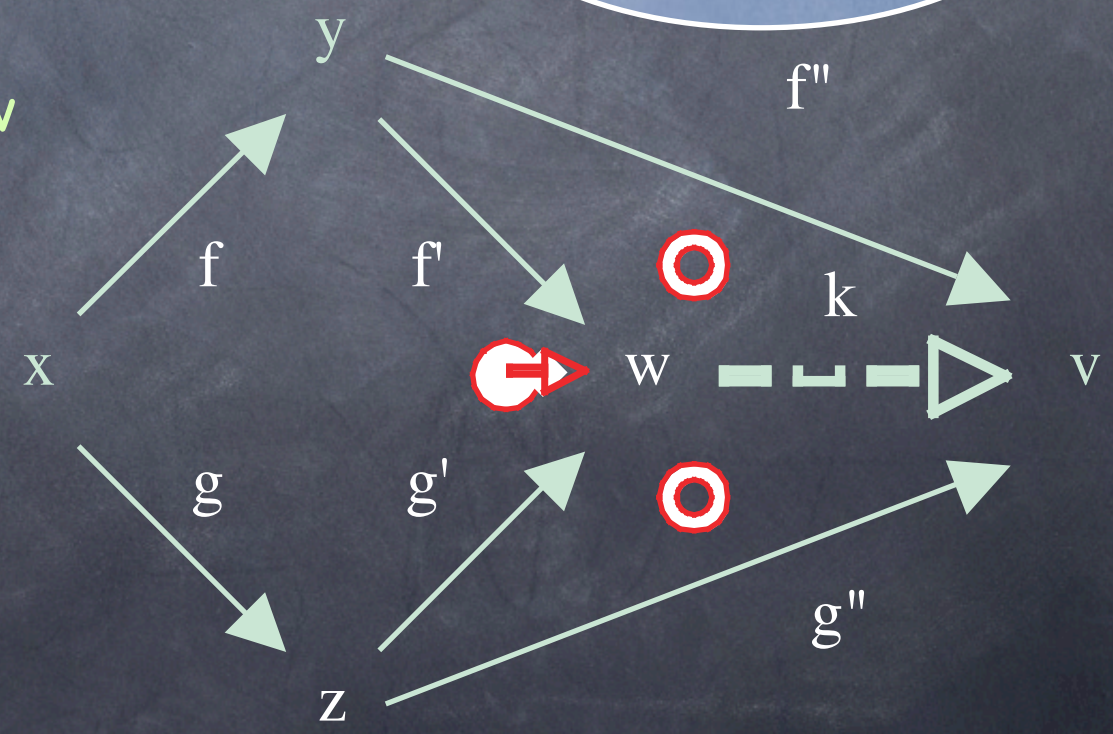




Pushout in a category C

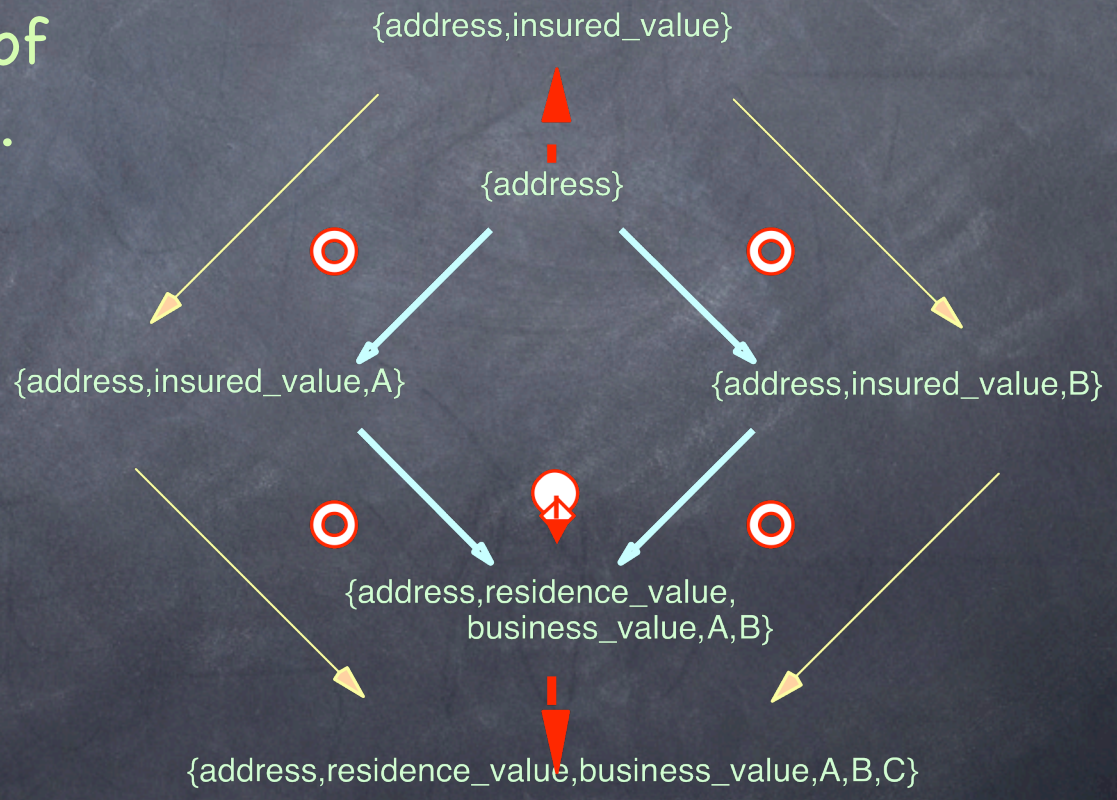
- A pushout of f:x→y and g:x→z co
  morphisms f':y→w and g':z→w su
  (1) f;f'=g;g' and
  (2) for any other
  f'':y→v and g'':z→v
  st f;f''=g;g'', there
  is a unique
  k:w→v st
  f';k=f'' and
  g';k=g''.

The dual concept is the pullback of f and g



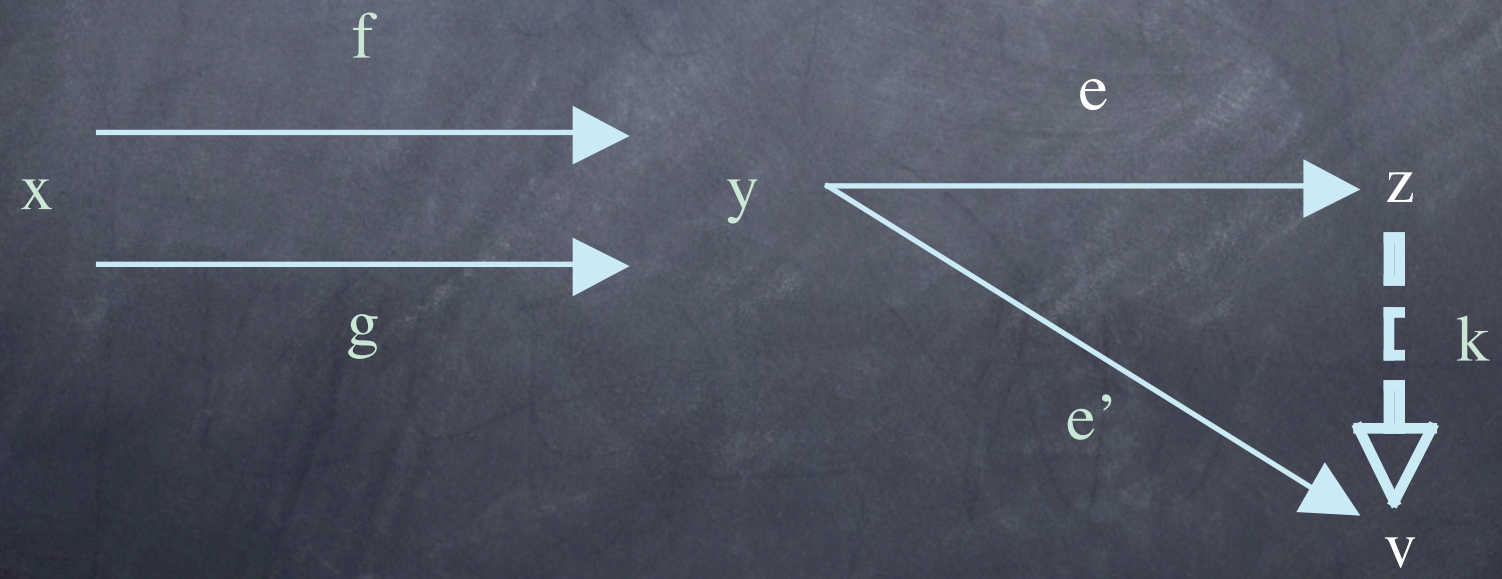
Eiffel's Interitance Structure

- Pushouts allows us to join (merge) features during multiple inheritance, as in the construction of HOME_BUSINESS.



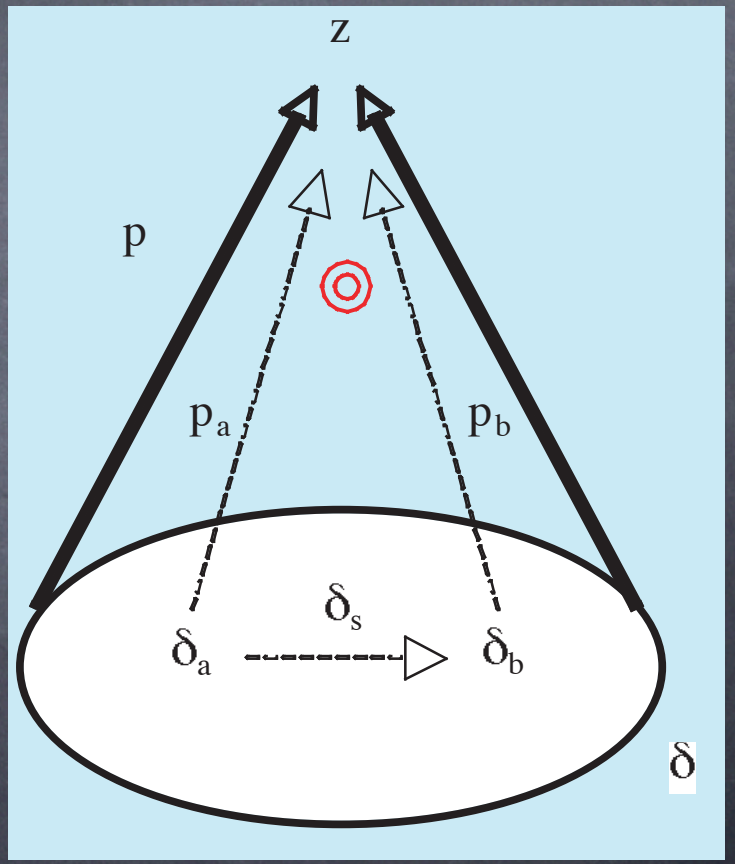
Coequaliser in a category C

- A coequaliser of f:x→y and g:x→y consists of a morphism e:y→z such that (1) f;e=g;e and (2) for any other e':y→v st f;e'=g;e', there is a unique k:z→v st e;k=e'.



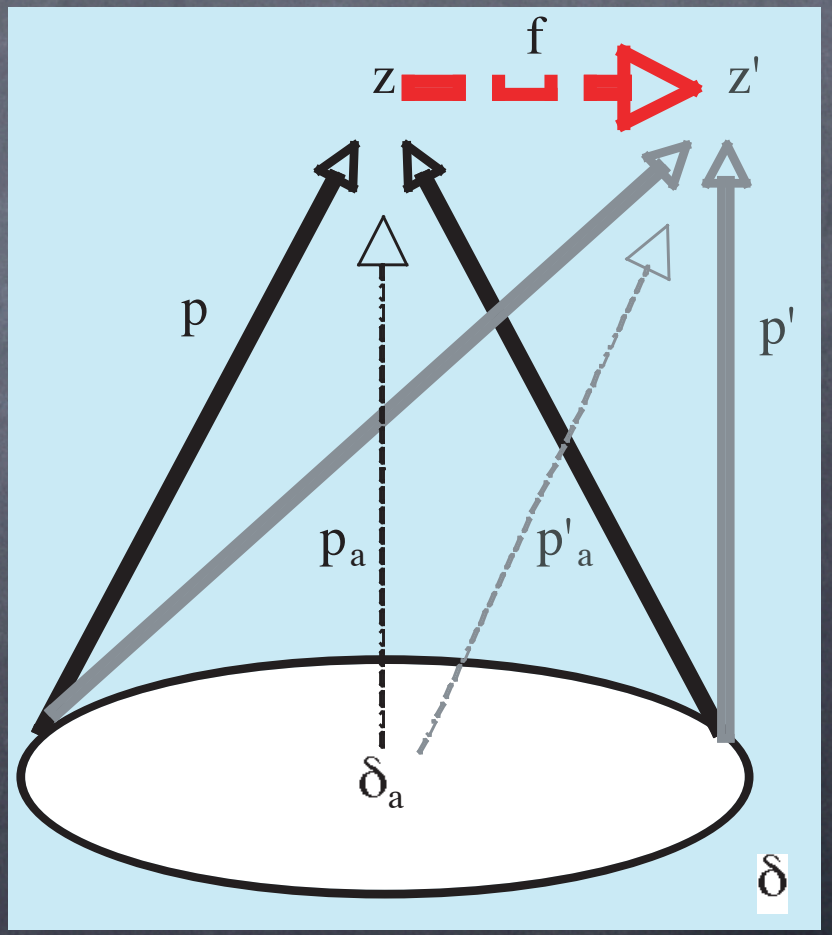
Co-cone in a category C

- Let $\delta: I \to C$ be a diagram. A cocone with base δ is an object z (the vertex) together with a family $\{p_a: \delta_a \to z\}_{a \in I_0}$ of morphisms (edges), usually denoted by $p: \delta \to z$.
- A cocone p with base δ vertex z is said to be commutative iff for every arrow $s: a \to b$ of I, $\delta_s; p_b = p_a$.



Colimit of a diagram

- Let $\delta: I \to C$ be a diagram. A colimit of δ is a commutative cocone $p: \delta \to z$ such that, for every other commutative cocone $p': \delta \to z'$ there is a unique morphism $f: z \to z'$ such that $p_a ; f = p'_a$.



- **Cocomplete category**
  - A category is (finitely) cocomplete if all (finite) diagrams have colimits.
- **Generating finite colimits**
  - A category is finitely cocomplete iff it has initial objects and pushouts of all pairs of morphisms with common source.

- **PROC**
  - The set of trajectories over a pointed set $A_\perp=\langle A,\perp_A\rangle$ is $tra(A_\perp)=\{\lambda:\omega\rightarrow A\}$
  - Every morphism $f:A_\perp\rightarrow A'_\perp$ extends to $tra(A_\perp)$ by pointwise application $tra(f)(\lambda)=\lambda;f:\omega\rightarrow A'$.
  - A process is a pair $\langle A_\perp,\Lambda\subseteq tra(A_\perp)\rangle$
  - A process morphism $f:\langle A_\perp,\Lambda\rangle\rightarrow\langle A'_\perp,\Lambda'\rangle$ is a morphism of alphabets $f:A_\perp\rightarrow A'_\perp$ such that $tra(f)(\Lambda)\subseteq\Lambda'$
  - PROC is complete and cocomplete.

- **PROC**
  - The terminal process is $\langle \perp_\emptyset, tra(\perp_\emptyset) \rangle$
  - The initial process is $\langle \perp_\emptyset, \emptyset \rangle$
  - A pullback of $g_1:\langle A_{1\perp}, \Lambda_1 \rangle \rightarrow \langle A_\perp, \Lambda \rangle$, $g_2:\langle A_{2\perp}, \Lambda_2 \rangle \rightarrow \langle A_\perp, \Lambda \rangle$ is $g'_1:\langle A'_\perp, \Lambda' \rangle \rightarrow \langle A_{1\perp}, \Lambda_1 \rangle$, $g'_2:\langle A'_\perp, \Lambda' \rangle \rightarrow \langle A_{2\perp}, \Lambda_2 \rangle$ where $g'_1:A'_\perp \rightarrow A_{1\perp}$, $g'_2:A'_\perp \rightarrow A_{2\perp}$ is a pullback of g_1 and g_2 and $\Lambda' = g'_1{}^{-1}(\Lambda_1) \cap g'_2{}^{-1}(\Lambda_2)$.

This corresponds to parallel composition in trace-based semantics.

PROC - examples

process VM **is**
alphabet co, ca, ci
behaviour

$$\Lambda ::= \perp^{\omega} \mid \perp^{*}co\perp^{\omega} \mid (\perp^{*}co\perp^{*}\{ca,ci\})\Lambda$$

process RVM **is**
alphabet co, ca, ci, co|to
behaviour

$$\Lambda ::= \perp^{\omega} \mid \perp^{*}co\perp^{\omega} \mid (\perp^{*}co\perp^{*}ca)\Lambda$$
$$\mid \perp^{*}co|to\perp^{*} \mid (\perp^{*}co|to\perp^{*}ci)\Lambda$$

- **Functors**
  - A functor $\varphi:C \to D$ between two categories is a graph homomorphism between their underlying graphs that preserves the composition law and the identities, i.e. $\varphi(f;g)=\varphi(f);\varphi(g)$ and $\varphi(id_x)=id_{\varphi(x)}$
  - Functors compose in the obvious way and admit the obvious identity, leading to the category CAT of categories and functors.
- **Process alphabets**
  - $alph:PROC \to SET_\perp$ mapping processes to their alphabets is a (forgetful) functor.

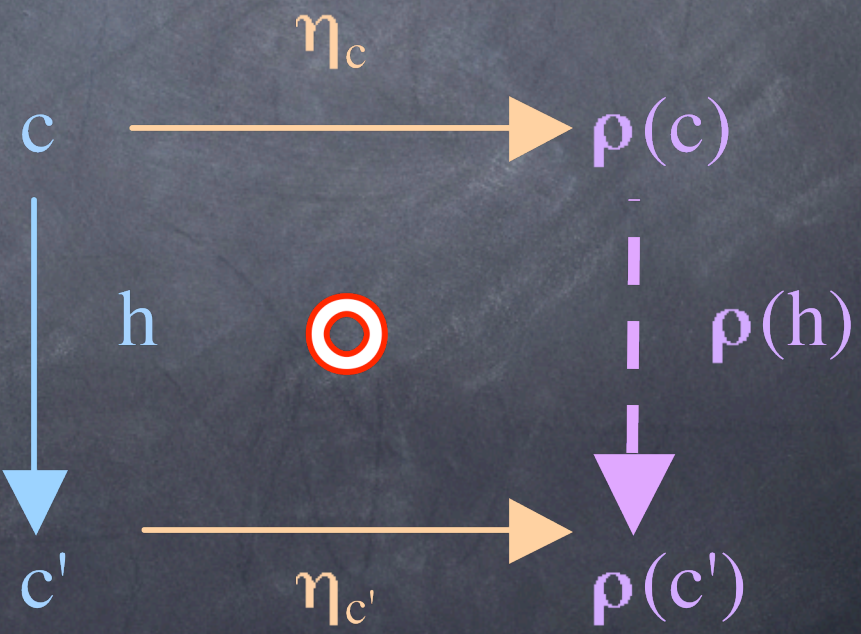
Classes of functors

- A functor is an isomorphism iff it admits an inverse.
- A functor is an embedding iff it is injective.
- A functor is faithful iff all the restrictions $\varphi:hom_c(x,y)\rightarrow hom_c(\varphi(x),\varphi(y))$ are injective.
- A functor is full iff all the restrictions $\varphi:hom_c(x,y)\rightarrow hom_c(\varphi(x),\varphi(y))$ are surjective.

alph:PROC$\rightarrow$SET$_\perp$ is faithful but not full.

Reflector

- Let D be a reflective subcategory of C. We define a functor $\rho: C \to D$ by mapping each object $c:C$ to the target of its reflection $\eta_c: c \to d = \rho(c)$ and every morphism $h: c \to c'$ to the unique morphism $\rho(h): \rho(c) \to \rho(c')$ such that $\eta_c ; \rho(h) = h ; \eta_{c'}$
- This functor is called a reflector for the inclusion of D in C

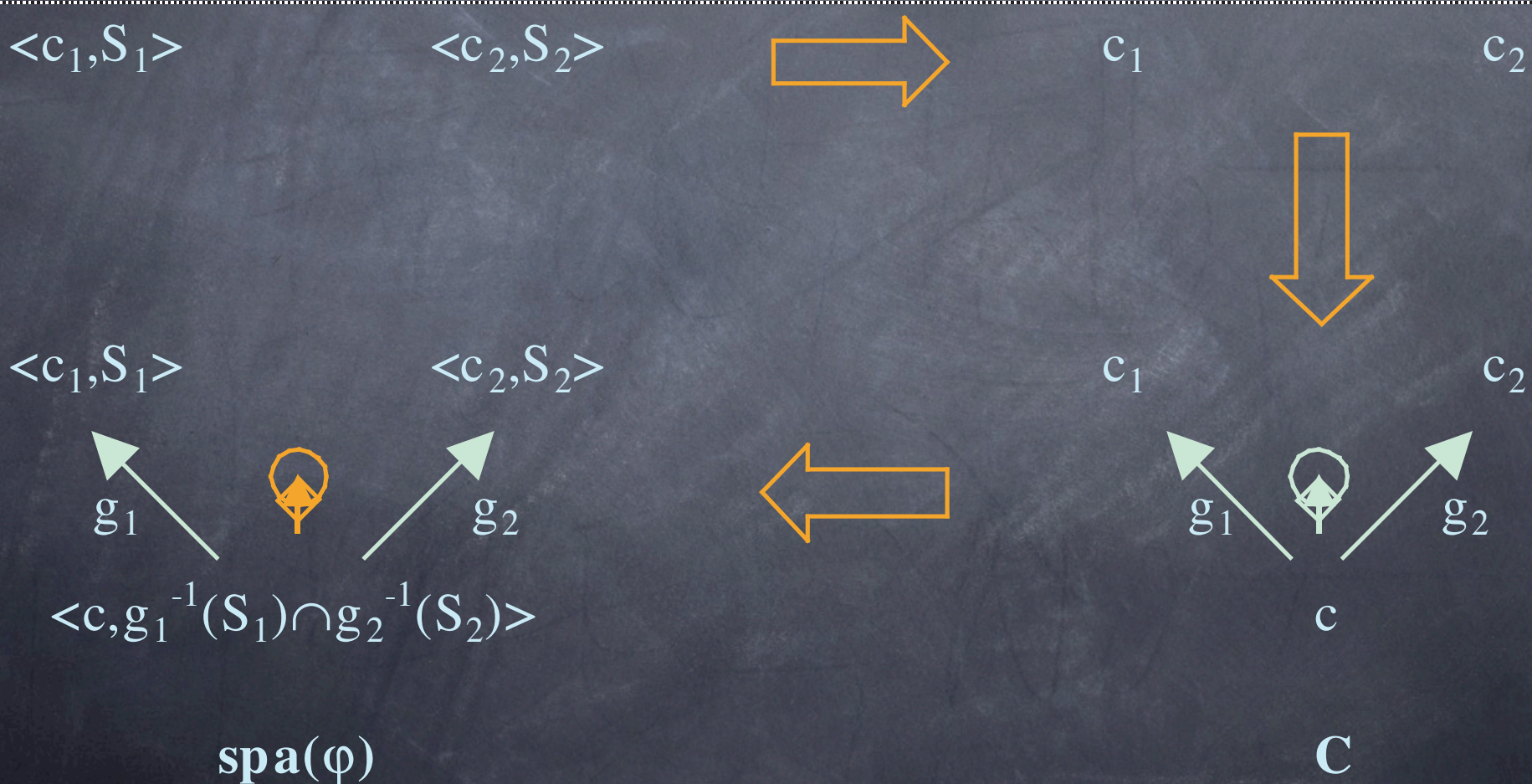


- **Example**
  - tra($A_\perp$)=$\{\lambda:\omega\to A\}$ defines a functor tra:$SET_\perp\to SET$.

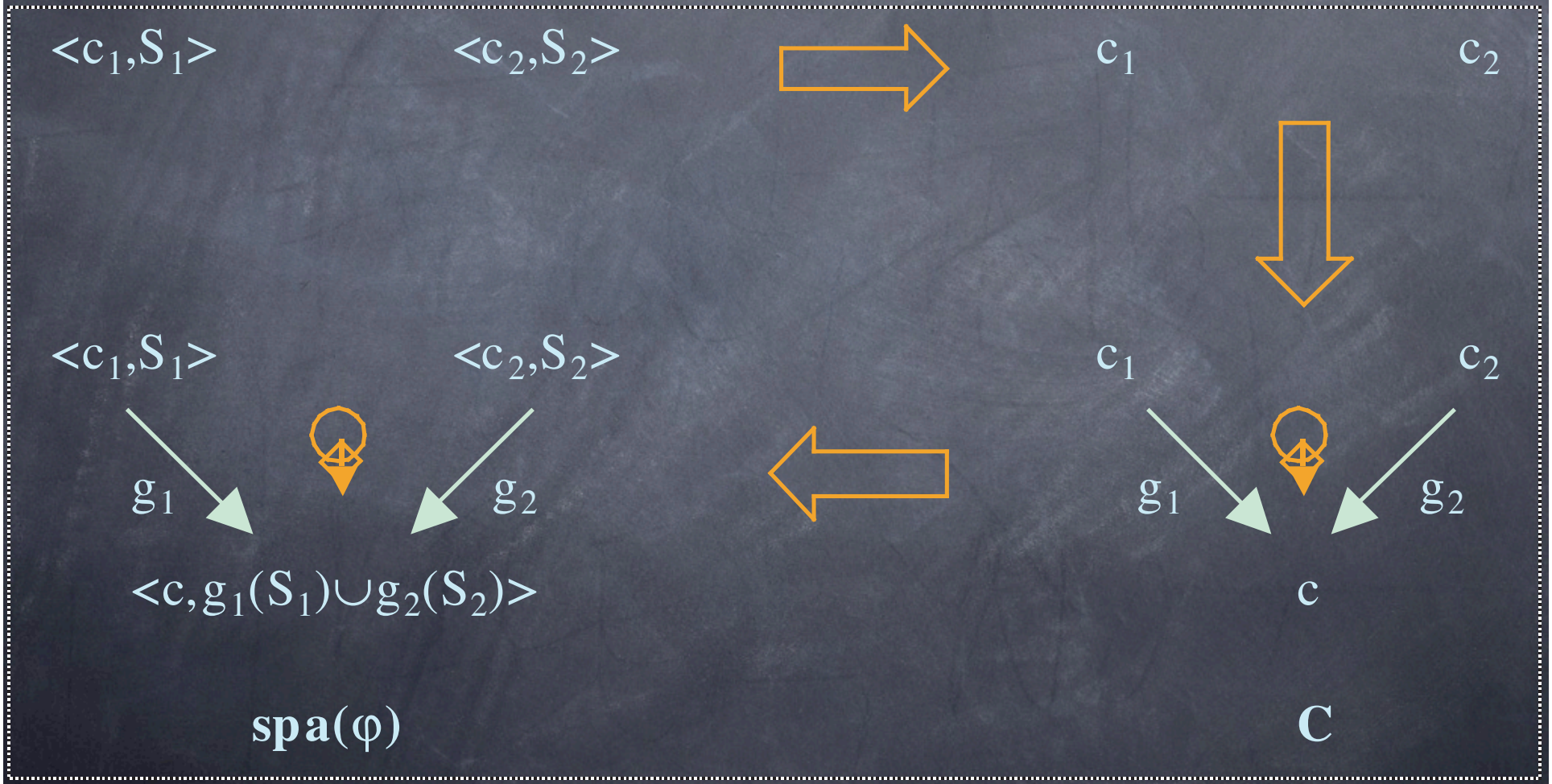
- **Spa-categories**
  - Given a functor φ:C→SET, we define spa(φ) whose objects are pairs <c,S⊆φ(c)> and whose morphisms f:<c,S>→<d,T> are the morphisms f:c→d such that tra(f)(S)⊆T.
  - PROC is spa(tra).

- **If C is (co)complete so is spa(φ)**
  - The terminal object is $\langle 1, \varphi(1) \rangle$
  - The initial object is $\langle 0, \emptyset \rangle$

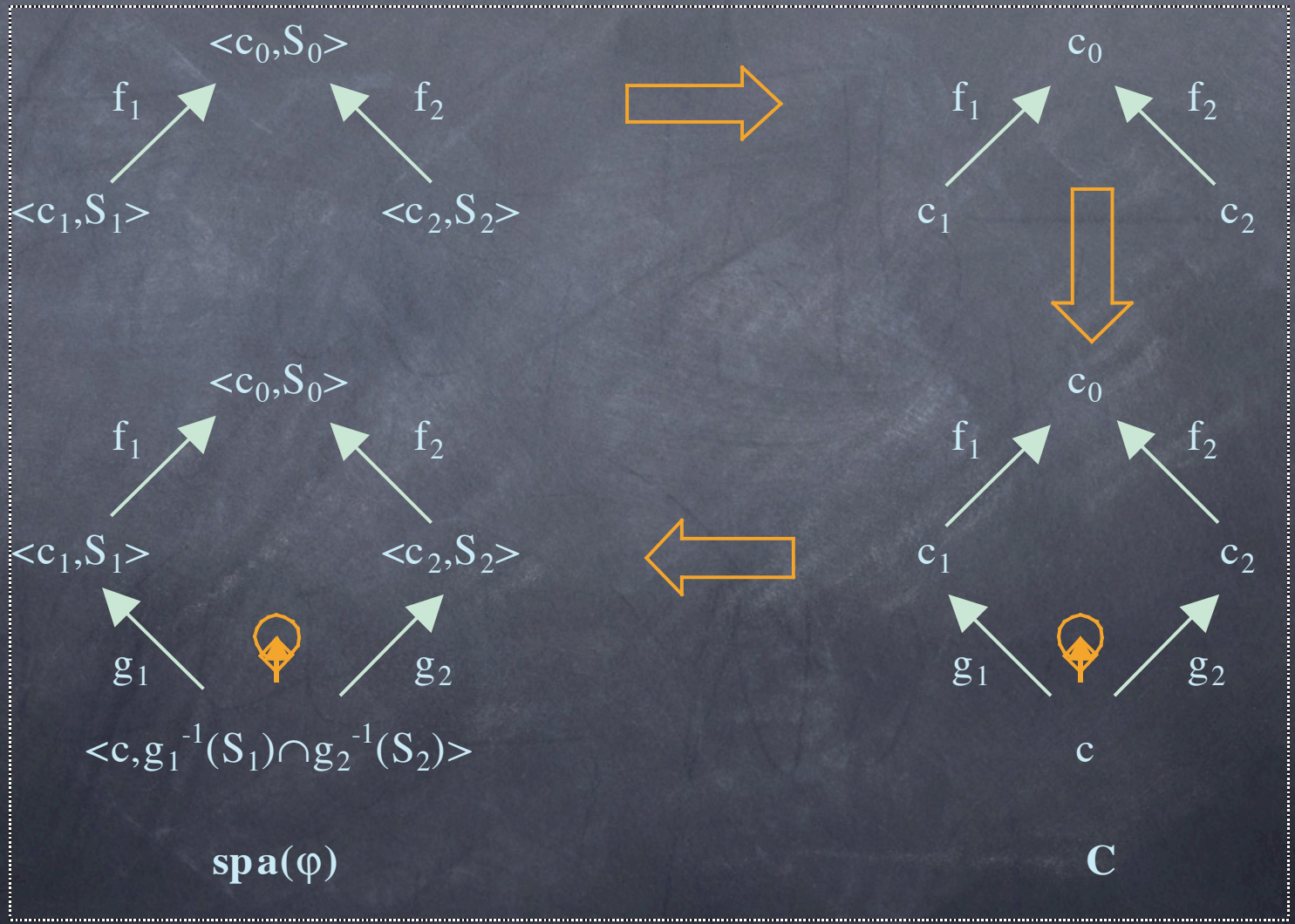
The products



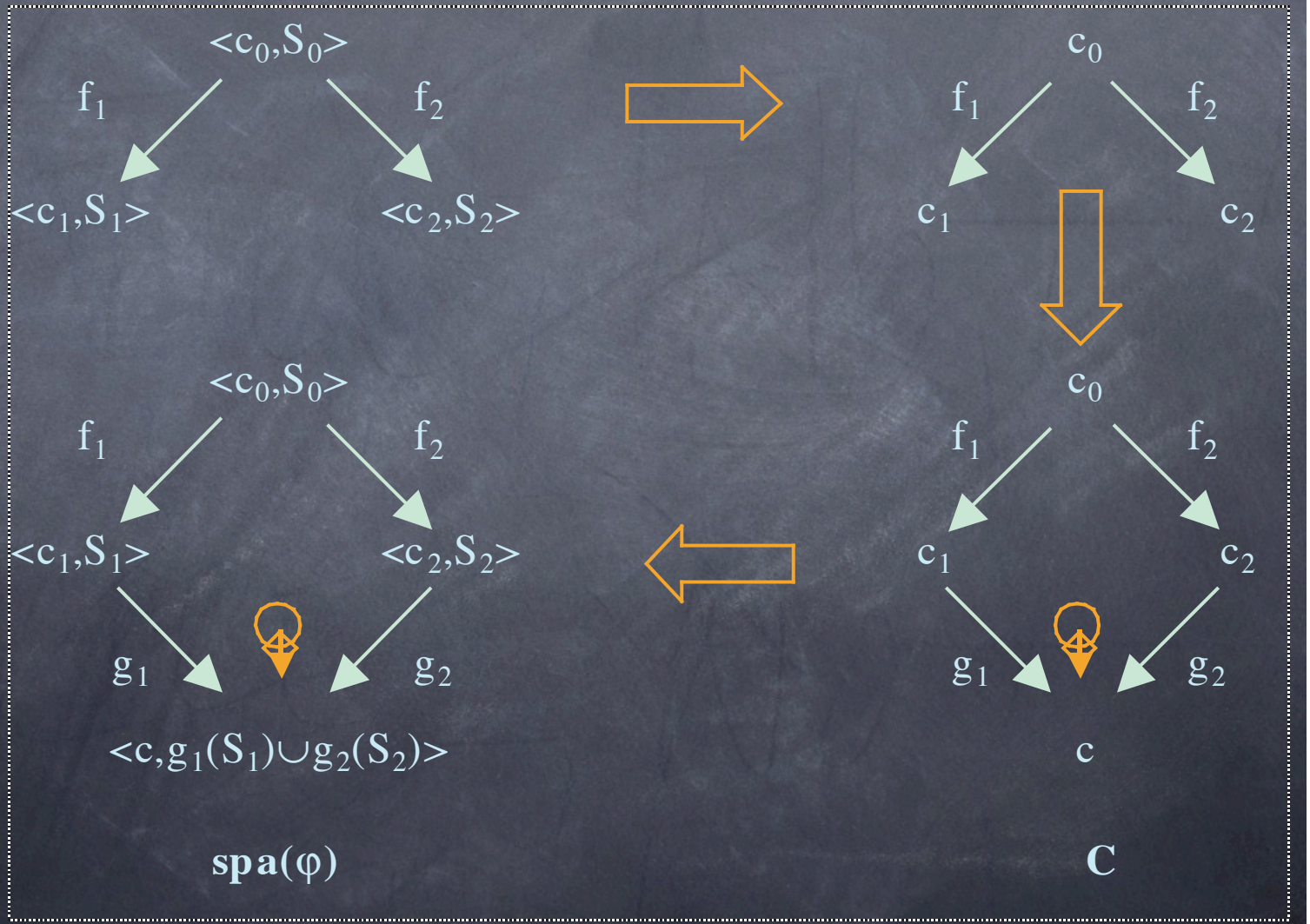
The sums



The pullbacks



The pushouts



The forgetful functor

- ν:spa(φ)$\rightarrow$C "forgets" the SET-component
- A functor ν:A$\rightarrow$B lifts limits (uniquely) iff for any diagram δ in A and limit μ':b$\rightarrow\delta;\nu$ in B there is a (unique) cone μ:a$\rightarrow\delta$ that is a limit of δ and $\mu'=\nu(\mu)$.
- ν:spa(φ)$\rightarrow$C lifts limits uniquely:
  Any limit μ:c$\rightarrow\delta;\nu$ with $\delta_i=\langle c_i,S_i\rangle$ is lifted uniquely to μ:$\langle c,S\rangle\rightarrow\delta$ where $S=\cap_i\varphi(\mu_i)^{-1}(S_i)$
- ν:spa(φ)$\rightarrow$C lifts colimits uniquely:
  Any colimit $\mu:\delta;\nu\rightarrow$c with $\delta_i=\langle c_i,S_i\rangle$ is lifted uniquely to $\mu:\delta\rightarrow\langle c,S\rangle$ where $S=\cup_i\varphi(\mu_i)(S_i)$

- **Temporal specification**
  - Linear temporal language PROP(Σ) over a set
    $\phi ::= \mathbf{beg} \mid a \in \Sigma \mid \neg\phi \mid \phi_1 \supset \phi_2 \mid \phi_1 \,\mathbf{U}\, \phi_2$
  - translation defined by $f:\Sigma \to \Sigma'$
    $\underline{f}(\phi) ::= \mathbf{beg} \mid \underline{f}(a) \mid \neg\underline{f}(\phi) \mid \underline{f}(\phi_1) \supset \underline{f}(\phi_2) \mid \underline{f}(\phi_1) \,\mathbf{U}\, \underline{f}(\phi_2)$

Semantics of PROP(Σ) over $(2^{\Sigma})^{\omega}$

$\lambda \models^i a$ iff $a \in \lambda(i)$

$\lambda \models^i \mathbf{beg}$ iff $i=0$

$\lambda \models^i \neg\phi$ iff it is not the case that $\lambda \models^i \phi$

$\lambda \models^i \phi_1 \supset \phi_2$ iff $\lambda \models^i \phi_1$ implies $\lambda \models^i \phi_2$,

$\lambda \models^i \phi_1 \mathbf{U} \phi_2$ iff, for some $j>i$, $\lambda \models^i \phi_2$ and, for every $i<k<j$, $\lambda \models^k \phi_1$

$\lambda \models \phi$ iff $\lambda \models^i \phi$ for every i

$\Phi \models \phi$ iff, for every λ, $\lambda \models \Phi$ implies $\lambda \models \phi$

THEO

objects: $\langle\Sigma,\Phi\rangle$ such that Φ is closed - $c(\Phi)=\Phi$

arrows: $f:\langle\Sigma,\Phi\rangle\rightarrow\langle\Sigma',\Phi'\rangle$ is $f:\Sigma\rightarrow\Sigma'$ s.t. $PROP(f)(\Phi)\subseteq\Phi'$

PRES

objects: theory presentations $\langle\Sigma,\Phi\rangle$

arrows: $f:\langle\Sigma,\Phi\rangle\rightarrow\langle\Sigma',\Phi'\rangle$ is $f:\Sigma\rightarrow\Sigma'$ s.t. $f(\Phi)\subseteq c(\Phi')$

where $c(\Phi)=\{\phi: \Phi\models\phi\}$

Example

Specification vending machine **is**

signature coin, cake, cigar

axioms

beg $\supset \neg cake \wedge \neg cigar \wedge (coin \vee (\neg cake \wedge \neg cigar) \mathbf{W} coin)$

$coin \supset (\neg coin) \mathbf{W} (cake \vee cigar)$

$(cake \vee cigar) \supset (\neg cake \wedge \neg cigar) \mathbf{W} coin$

$cake \supset \neg cigar$

Example

Specification regulated vending machine **is**

signature coin, cake, cigar, token

axioms

beg $\supset \neg cake \wedge \neg cigar \wedge \neg token \wedge$
$(coin \vee (\neg cake \wedge \neg cigar) \mathbf{W} coin)$

$coin \supset (\neg coin) \mathbf{W} (cake \vee cigar)$

$coin \supset (\neg cigar) \mathbf{W} token$

$(cake \vee cigar) \supset (\neg cake \wedge \neg cigar) \mathbf{W} coin$

$cake \supset \neg cigar$

$token \supset \neg cigar$

The forgetful functors

- sign:THEO→SET and sign:PRES→SET forget the axioms/theorems and retain the signature
- Both functors lift limits:
  Any limit $\mu:\Sigma\rightarrow\delta;\nu$ with $\delta_i=\langle\Sigma_i,\Phi_i\rangle$ is lifted to $\mu:\langle\Sigma,\Phi\rangle\rightarrow\delta$ where $\Phi=\cap_i\mu_i^{-1}(\Phi_i)$ for THEO and
  $\Phi=\cap_i\mu_i^{-1}(c(\Phi_i))$ for PRES
- Both functors lift colimits:
  Any colimit $\mu:\delta;\nu\rightarrow\Sigma$ with $\delta_i=\langle\Sigma_i,\Phi_i\rangle$ is lifted to $\mu:\delta\rightarrow\langle\Sigma,\Phi\rangle$ where $\Phi=c(\cup_i\mu_i(\Phi_i))$
  (For PRES this is isomorphic to $\Phi=\cup_i\mu_i(\Phi_i)$)

Example

Specification regulator **is**

signature trigger, ted, tor

axioms

beg $\supset \neg$tor

trigger $\supset$ ($\neg$ted)**W**tor

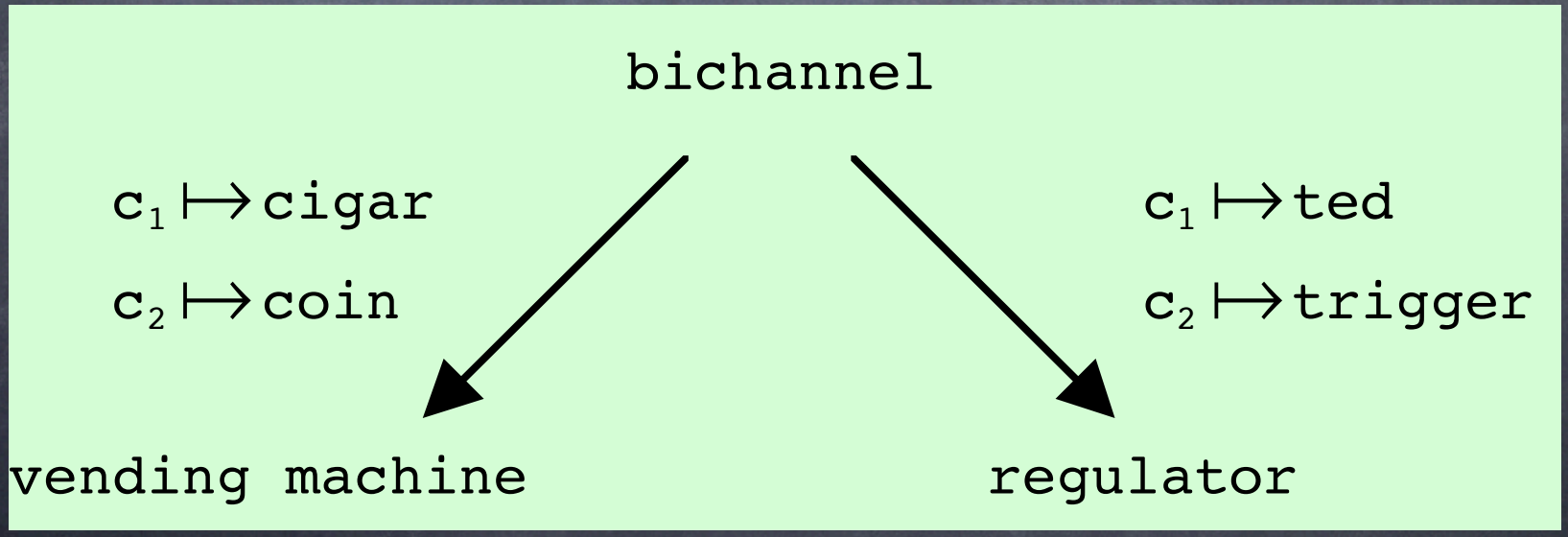
tor $\supset \neg$ted

Specification bichannel **is**

signature c_1, c_2

Pushout in PRES

- The specification of the regulated vending machine can be obtained (up to isomorphism) as a pushout of the following diagram:



- **Fibres**
  - Consider a functor $\nu: D \to C$.
    The fibre of $c:C$ is the subcategory of D that consists of all the objects d that are mapped to c, i.e. such that $\nu(d)=c$, together with the morphisms $f:d_1 \to d_2$ such that $id_c=\nu(f)$.
- **Examples**
  - All the theories for a given signature;
  - All the processes over a given alphabet.

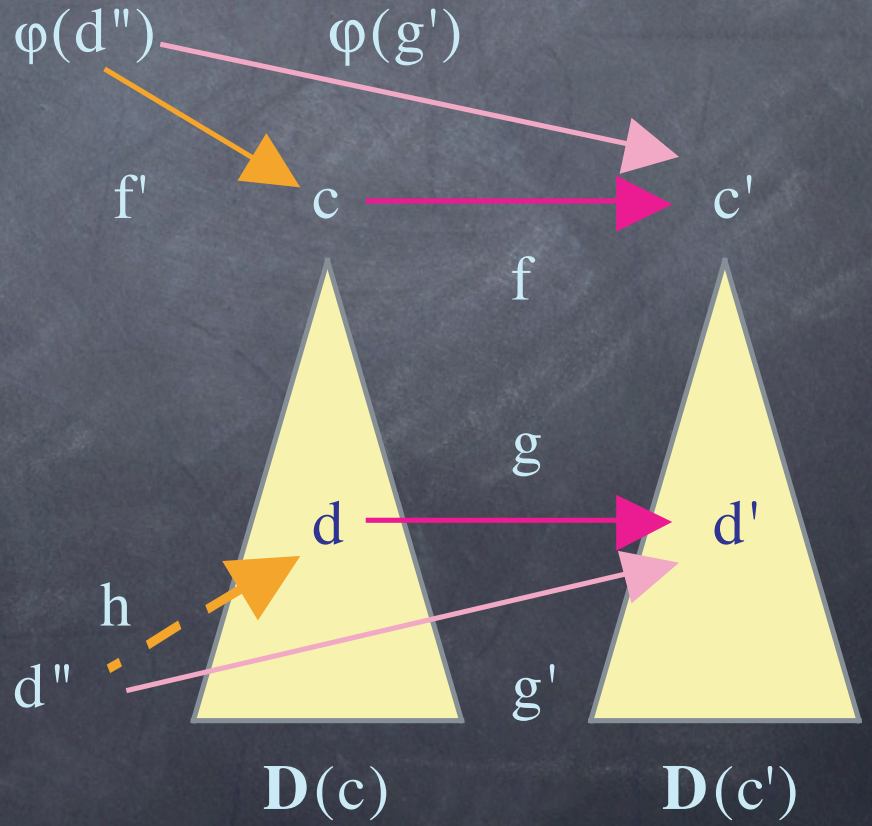
Cartesian morphisms

- Consider a functor $\varphi: D \to C$, $f: c \to c'$, and $d': D(c')$. A Cartesian morphism for f and d' is $g: d \to d'$ st:

$\varphi(g) = f$.

For every $g'': d'' \to d'$ and $f': \varphi(d'') \to c$ such that $\varphi(g'') = f'; f$, there is a unique $h: d'' \to d$ st $\varphi(h) = f'$ and $g'' = h; g$.

We use $\varphi^{-1}(d')$ for d.

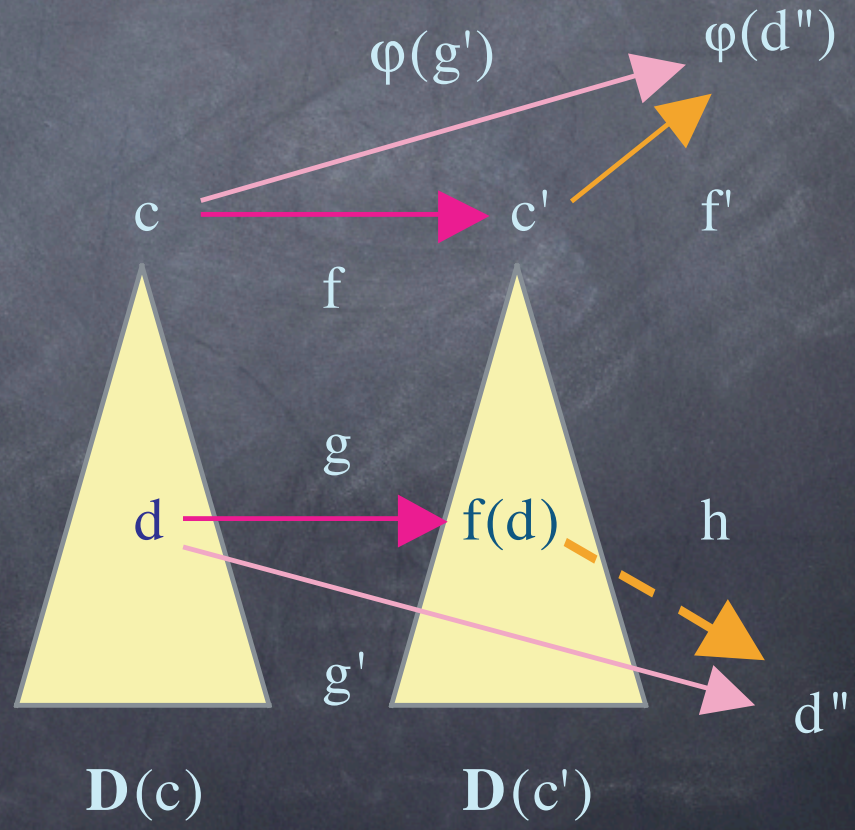


(Co)Cartesian morphisms

- Consider a functor $\varphi: D \to C$, $f: c \to c'$, and $d: D(c)$. A co-Cartesian morphism for f and d is $g: d \to d'$ st:

$\varphi(g) = f$.

For every $g': d \to d''$ and $f': c' \to \varphi(d'')$ such that $\varphi(g') = f;f'$, there is a unique $h: d' \to d''$ st $\varphi(h) = f'$ and $g' = g;h$.

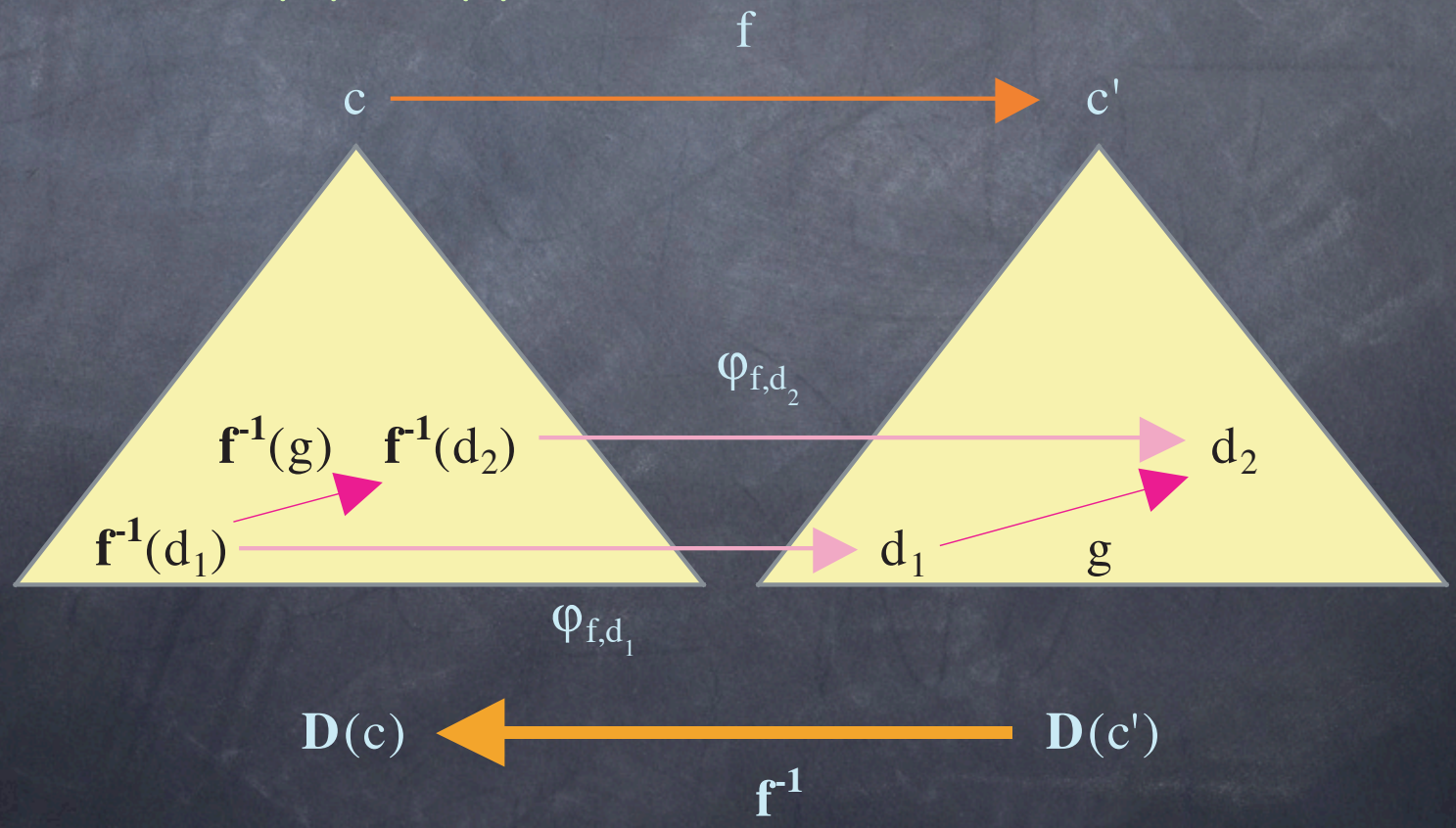


- **Examples**
  - Given $f:\Sigma \rightarrow \Sigma'$,
    $\langle \Sigma', c(f(\Phi)) \rangle$ is co-cartesian for $\langle \Sigma, \Phi \rangle$ in THEO
    $\langle \Sigma', f(\Phi) \rangle$ is co-cartesian for $\langle \Sigma, \Phi \rangle$ in PRES
    and so is $\langle \Sigma', c(f(\Phi)) \rangle$: there is no uniqueness!
    $\langle \Sigma, f^{-1}(\Phi') \rangle$ is cartesian for $\langle \Sigma', \Phi' \rangle$ in THEO
    $\langle \Sigma, f^{-1}(c(\Phi')) \rangle$ is cartesian for $\langle \Sigma', \Phi' \rangle$ in PRES
  - In spa(φ), given $f:c \rightarrow c'$,
    $\langle c', \varphi(f)(S) \rangle$ is co-cartesian for $\langle c, S \rangle$
    $\langle c, \varphi(f)^{-1}(S') \rangle$ is cartesian for $\langle c', S' \rangle$

- **(Co)Fibrations**
  - A fibration is a functor $\varphi: D \to C$ such that there is a Cartesian morphism for every $f: c \to c'$ and $d': D(c')$.
  - A cofibration is a functor $\varphi: D \to C$ such that there is a coCartesian morphism for every $f: c \to c'$ and $d: D(c)$.
- **Cleavages**
  - A choice of a Cartesian morphism for every $f: c \to c'$ and $d': D(c')$ is called a cleavage. A fribration equipped with a cleavage is said to be cloven.
  - Co-cleavages are defined by duality.

Induced functors

- A cloven fibration defines, for every $f:c \to c'$, a functor $f^{-1}:D(c') \to D(c)$

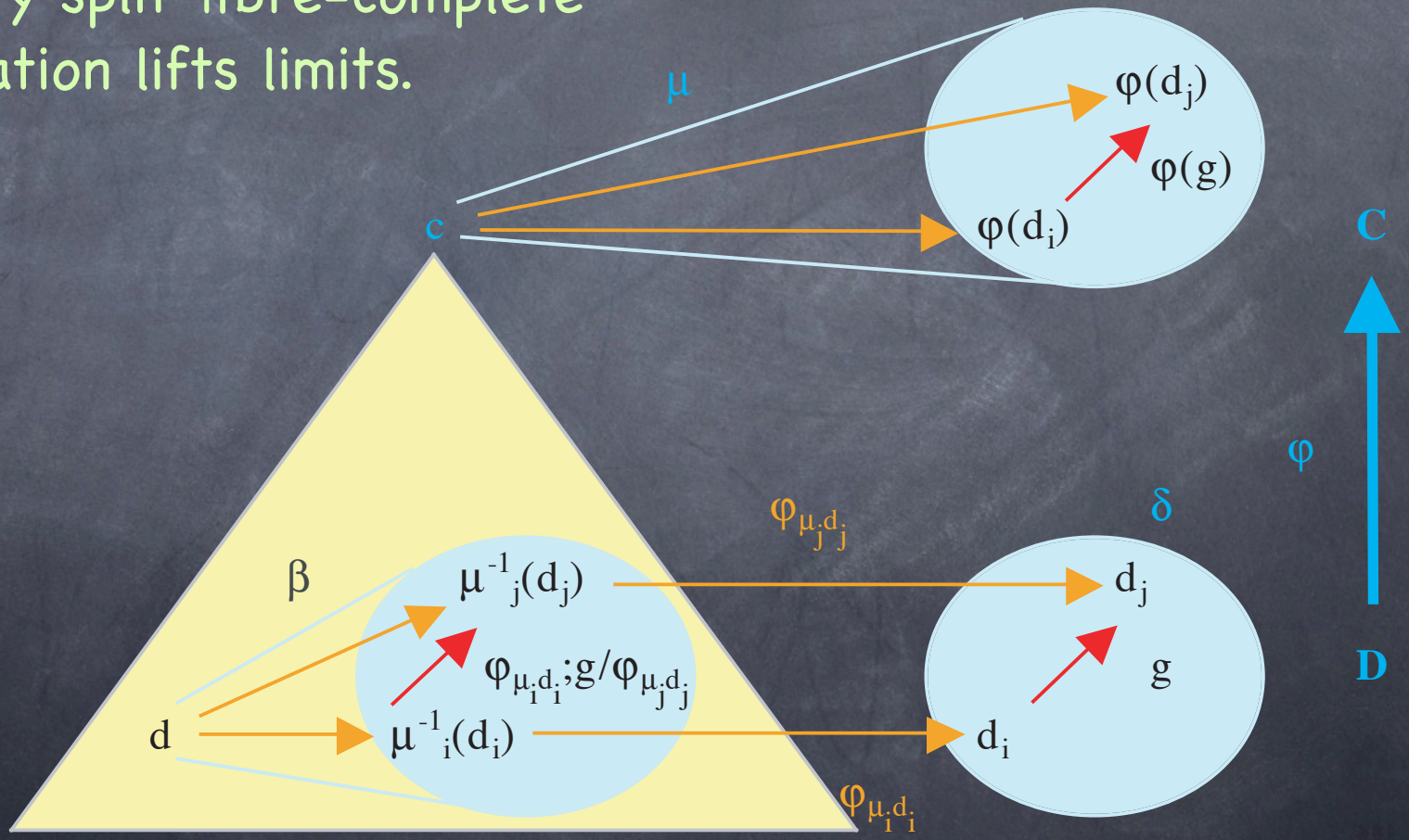


- **Fibre-completeness**
  - A cloven fibration is said to be fibre-complete if, for every C-object c, its fibre D(c) is complete and, for every morphism $f:c \to c'$, the functor $f^{-1}:D(c') \to D(c)$ preserves limits.
  - A functor $\nu:A \to B$ preserves limits iff for any limit $p:\delta \to a$ in A the cocone $\nu(p):\delta;\phi \to \phi(a)$ is a limit of $\delta;\phi$.

- **Split**
  - A cloven fibration is said to be split if, for every C-object c, id_c^{-1} is $id_{D(c)}$ and, for every decomposition $f=f_1;f_2$, f^{-1} is the composition $f^{-1}{}_2;f^{-1}{}_1$.
  - Every split fibration $\varphi:D\rightarrow C$ defines the functor $ind(\varphi):C^{op}\rightarrow CAT$ that maps every object c to its fibre D(c) and every morphism $f:c\rightarrow c'$ to the functor $f^{-1}:D(c')\rightarrow D(c)$.

Lifting limits

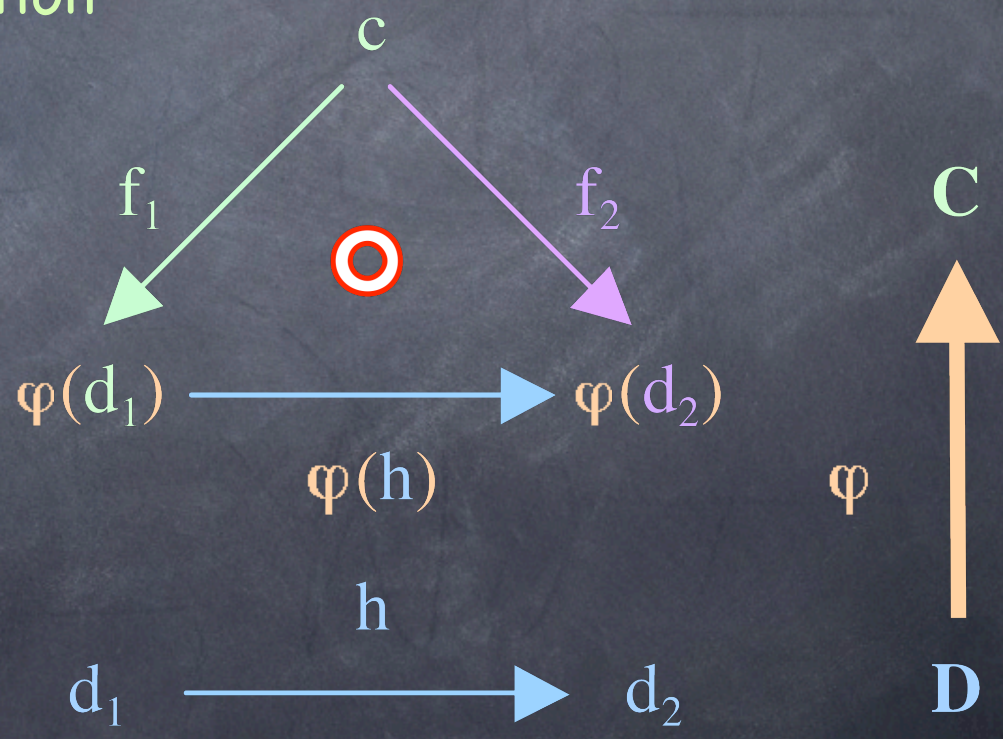
- Every split fibre-complete fibration lifts limits.



- **Concrete**
  - A concrete category is a faithful functor $\varphi:D\rightarrow C$.
  - The fibres of a concrete category are pre-orders. Those of a cloven fibration are partial orders.
  - In a concrete fibre-complete split fibration, any limit $\mu:c\rightarrow\delta;\varphi$ is lifted to $\mu:\cap_i\mu_i^{-1}(\delta_i)\rightarrow\delta$.
  - THEO and PRES are concrete fibre-(co)complete split (co)fibrations.
  - Every spa-category is a concrete fibre-(co)complete split (co)fibration.

Structured morphisms

- Given a functor $\varphi:D \to C$, a φ-structured morphism is a C-morphism of the form $f:c \to \varphi(d)$.
- We generalise the notion of comma-category to structured morphisms, i.e. given an object c:C we define $c\downarrow\varphi$ whose objects are the structured morphisms whose domain is c.

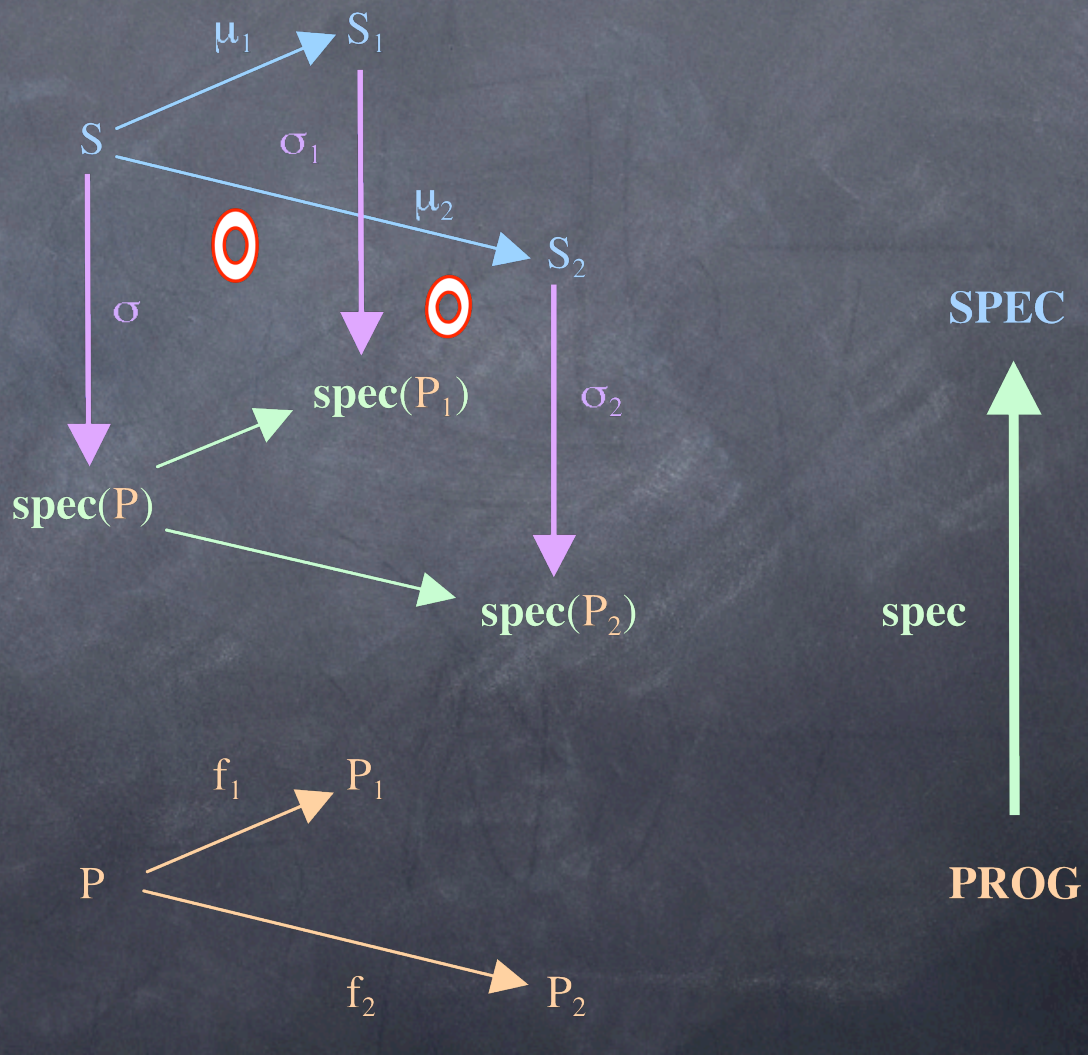


Programs and specifications

- Given a category PROG of programs and a category SPEC of program specifications, we can think of a functor spec:PROG→SPEC that maps programs to the strongest specification that they satisfy.
- A realisation of a specification S can be identified by a structured morphism σ:S→spec(P), which expresses how P satisfies/refines S.

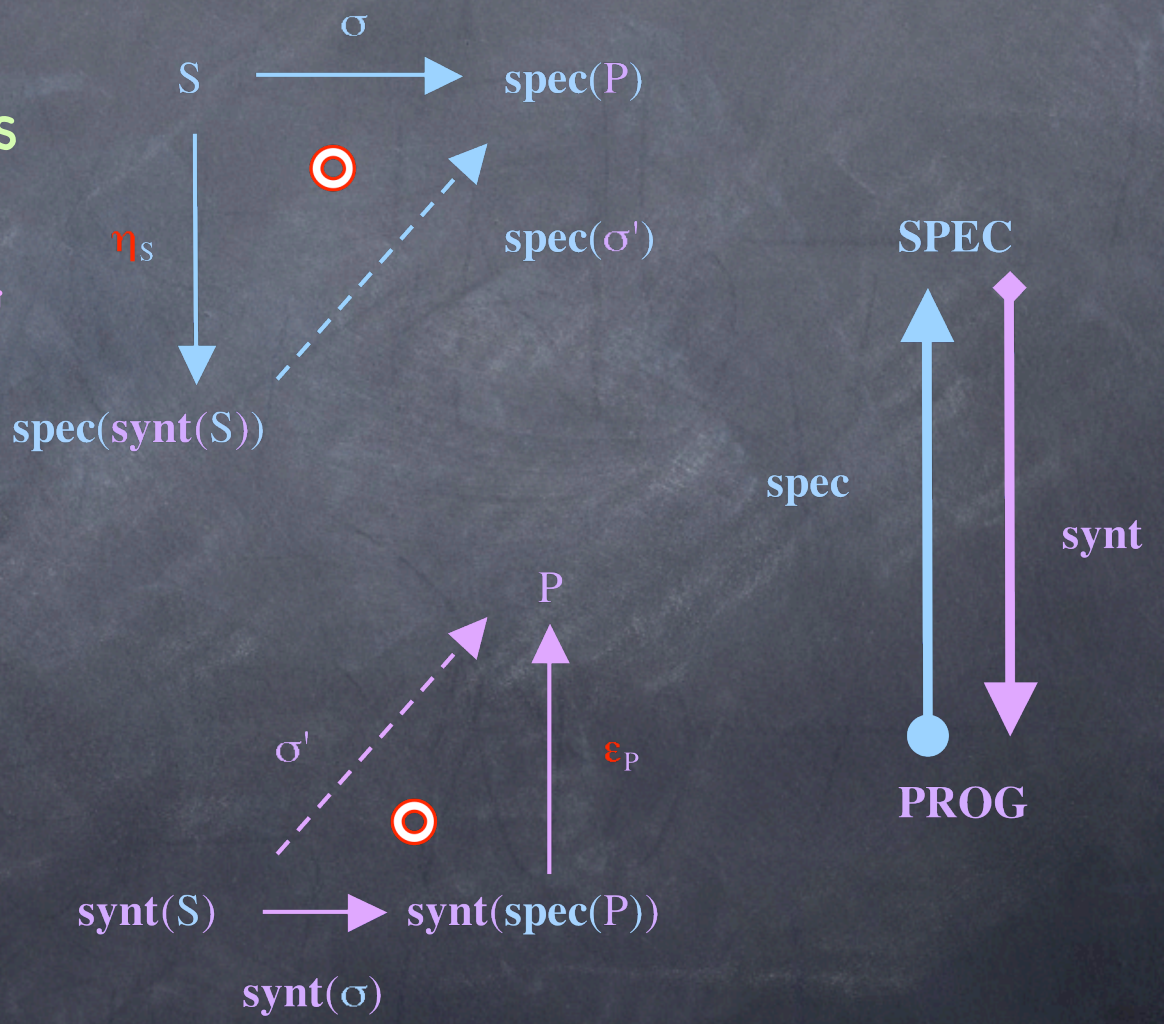
Realisations of complex systems

- This notion can be extended to configurations of complex systems expressed as diagrams.
- The fact that spec if a functor ensures compositionality.



Synthesising specifications

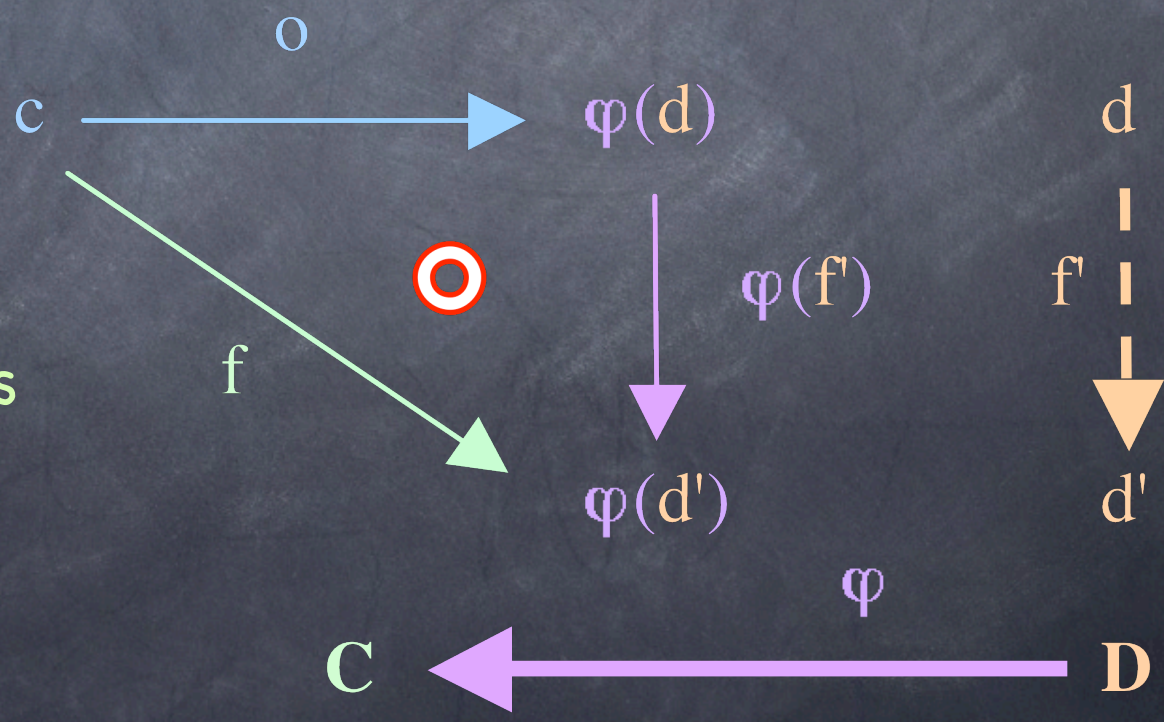
- What properties should a synthesis functor synt:SPEC→PROG have?
- η_S and ϵ_P are not necessarily isomorphisms!



Reflections

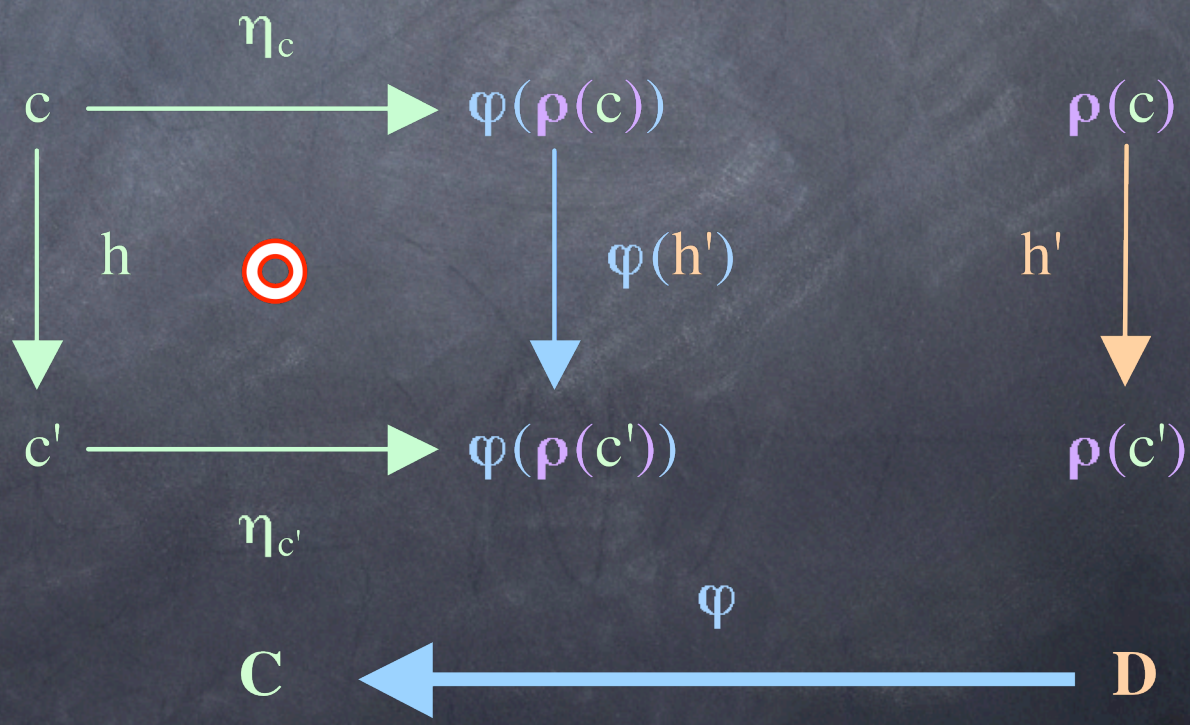
Let $\varphi: D \to C$ be a functor.

- A φ-reflection for $c:C$ is a structured morphism $o: c \to \varphi(d)$ such that, for any $f: c \to \varphi(d')$ there is a unique $f': d \to d'$ in D st $f = o; \varphi(f')$.
- φ is reflective iff every $c:C$ has a φ-reflection.



Reflectors

Let $\varphi:D\to C$ be a reflective functor. We define a functor $\rho:C\to D$ by mapping every $c:C$ to the target of its φ-reflection $\eta_c:c\to\varphi(\rho(c))$, and every $h:c\to c'$ to the unique $h':\rho(c)\to\rho(c')$ that makes the diagram commute.



- **Properties**

  Let $\varphi: D \to C$ be a reflective functor.

  - Its reflector $\rho: C \to D$ is co-reflective and φ is the co-reflector of ρ.
  - φ preserves limits.
  - φ is faithful iff each ρ-co-reflection $\epsilon_d: \rho(\varphi(d)) \to d$ is an epi.

Adjunctions

- An adjunction from a category C to a category D consists of a reflective functor φ:D→C together with is reflector ρ:C→D.
- φ is called the right adjoint (or adjoint or the "forgetful" functor).
- ρ is called the left adjoint (or co-adjoint or the "free" functor).
- Other more "standard" characterisations exist...

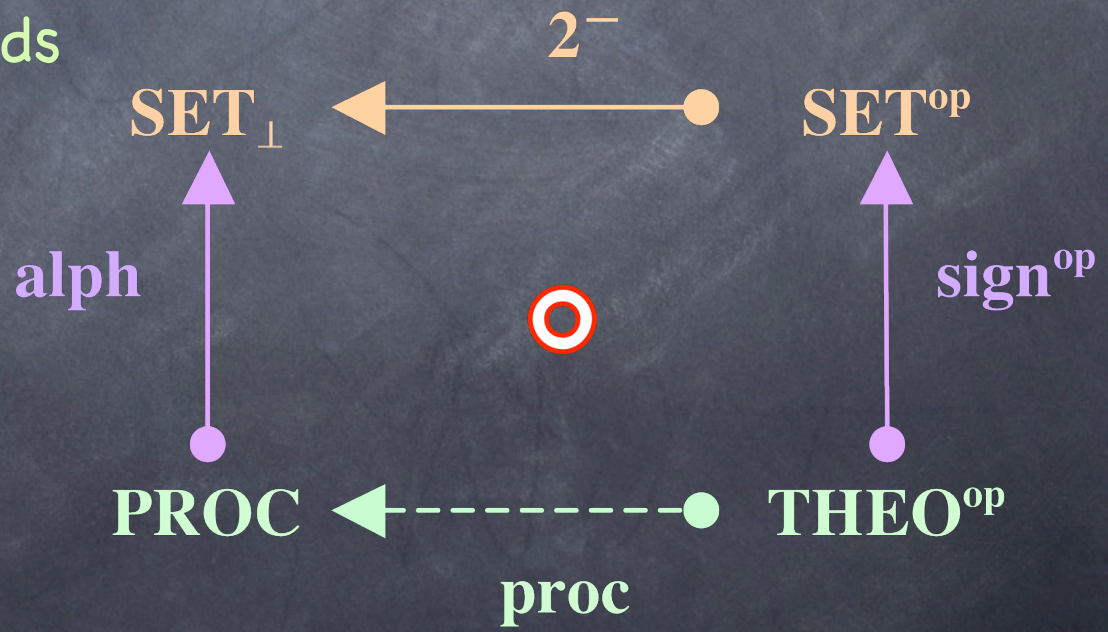
- **Examples**
  - Every reflective subcategory defines an adjunction in which the inclusion is the right adjoint.
  - Given a functor φ:C→SET, the forgetful functor spa(φ)→C has a left adjoint that maps c:C to <c,$\emptyset$> and a right adjoint that maps it to <c,φ(c)>.
  - The forgetful functor THEO→SET has a left adjoint that maps Σ:SET to <Σ,c($\emptyset$)> and a right adjoint that maps it to <Σ,PROP(Σ)>.

Examples

- The contravariant powerset functor $2^{-}:SET^{op}\rightarrow SET_{\perp}$

  is reflective and its left adjoint computes powersets of proper elements and inverse images.

- This functor extends to an adjunction between $THEO^{op}$ and PROC.

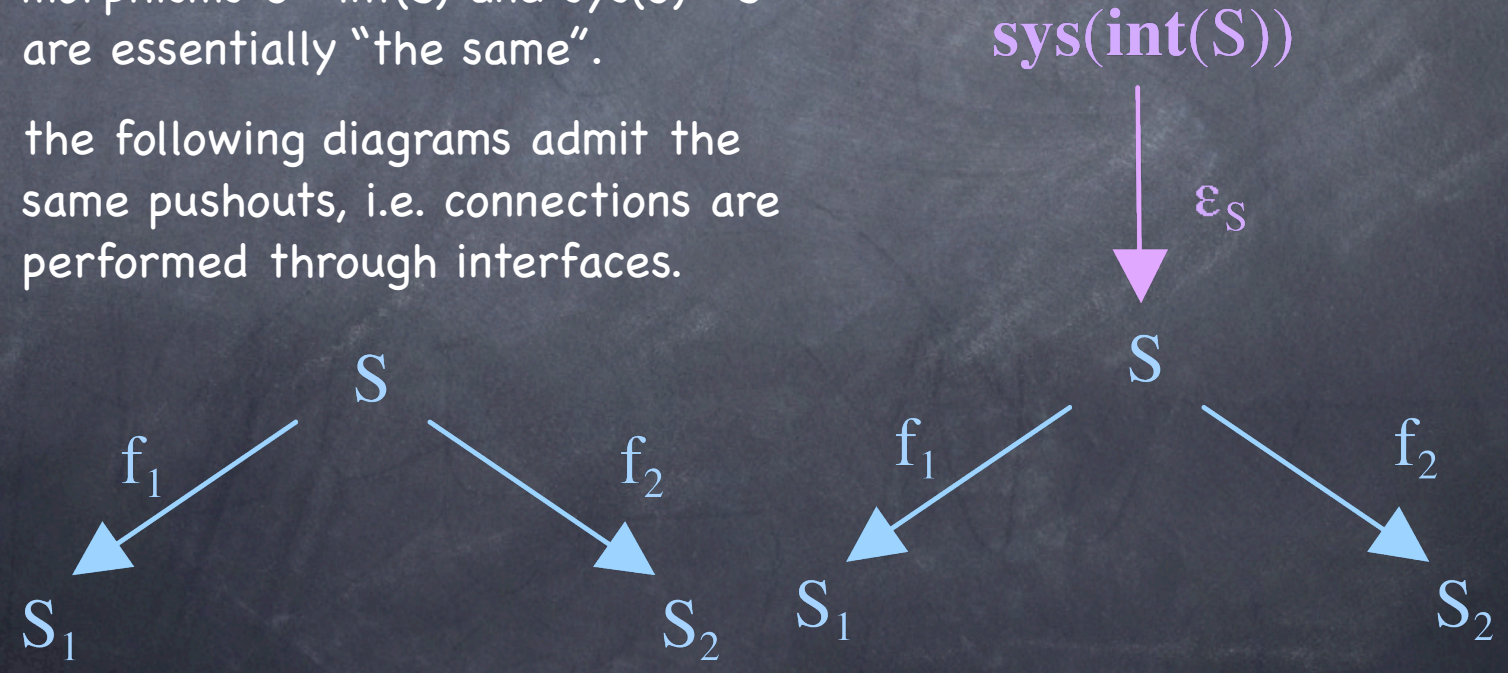


Discrete lifts

- Given a faithful functor (concrete category) $\varphi:D\to C$, a discrete lift for $c:C$ is $d:D$ such that $c=\varphi(d)$ and id_c is a $\varphi(d)$-reflection, i.e. for every $f:c\to\varphi(d')$, there is a (unique) $g:d\to d'$ such that $\varphi(g)=f$.
- The functor (concrete category) is said to have discrete structures whenever every $c:C$ has a discrete lift.
- A concrete category with discrete structures is reflective and the co-reflections are epis.

Interfaces

- Consider a functor int:SYS→INT mapping system models to interfaces that is faithful and has discrete structures, with reflector sys:INT→SYS.
  - morphisms C→int(S) and sys(C)→S are essentially "the same".
  - the following diagrams admit the same pushouts, i.e. connections are performed through interfaces.



Coordinated categories

- A faithful functor (concrete category) $\varphi: D \to C$ is said to be coordinated (or that D is coordinated over C) if it has discrete structures and lifts colimits.
- THEO is coordinated over SET
- PROC is (co)coordinated over $SET_{\perp}$