Concepts extended in time

Dusko Pavlovic
Royal Holloway

SamsonFest 60
Oxford, 28 May 2013

Outline

Outline

Montreal 1991: Category Theory

London 1993: Semantics of Computation

From Left to Right

Top Row: Iain Phillips, Guy McCusker, Simon Gay, Juarez Muylaert Filho, David Clark, Michael Huth, Thomas Jensen, Jose Fiadeiro, Paul Taylor, Steve Vickers, Julian Webster.

Middle Row: Sarah Liebert, Xiang Rong Shi, Kevin Lano, Greg Meredith, François Lamarche, Roy Crole, Abbas Edalat, Marta Kwiatkowska, John C. Reynolds, Gillian Hill, Chris Townsend.

Bottom Row: Mark Dawson, Duško Pavlović, Martín Escardó, Odinaldo Rodrigues, Samson Abramsky, Tom Maibaum, Chris Hankin, Raja Nagarajan, Ian Mackie, Lindsay Errington, Theodosis Dimitrakis.

London 1994: Semantics of Computation

Interaction Categories and the Foundations of Typed Concurrent Programming

Samson Abramsky, Simon Gay and Rajagopal Nagarajan

. . .

1.1 Semantic Paradigms

Denotational Semantics The most influential and longest established of current paradigms for the semantics of computation is denotational semantics. It is this paradigm which best approximates by far to the ideal of a mathematical theory of computation in the sense of McCarthy [45] or Scott [54].

The criticism we wish to lodge is one of scope. Despite its pretensions to universality, denotational semantics has an inherent bias towards a particular computational paradigm, that of *functional computation*. By this we mean, not only functional programming languages, but that whole sphere of computation in which the behaviour of the program is adequately abstracted as the computation of a function. This view of programs as func-

London 1994: Semantics of Computation

Process Calculi A different family of semantic paradigms has been developed for reactive systems. The most notable of these is the process calculus paradigm, pioneered by Milner and Hoare, and exemplified by CCS [49] and CSP [32]. The great achievement of this paradigm over the past 15 years has been to develop an algebraic theory of concurrency, as a basis for structural methods of description of concurrent systems. The major limitation is that no canonical theory or calculus for concurrency has emerged; there is a veritable Babel of formalisms, combinators, equivalences. This may suggest that the current methodologies for concurrency are insufficiently constrained, or perhaps that some key ideas are still missing. Some secondary, but also significant limitations of the

London 1994: Semantics of Computation

Calculi vs. Semantic Universes At this point it will be useful to contrast the methodology implicit in presenting theories of concurrency as formal "process calculi", with that in which one presents a "semantic universe" in the form of a categorical model. The formal calculus approach starts from a set of combinators generating a syntax; then one may define a structured operational semantics or a model, various notions of equivalence, etc.

The weakness of this methodology is in the very first step; why this set of combinators rather than any otherΓ If in fact a consensus had been reached that some calculus was canonical for concurrency in the same way and for the same kind of reasons that λ-calculus enjoys this status for functional computation, then this would not have been a problem. History has turned out otherwise, and should caution us to beware of availing ourselves too readily of the seductive freedoms of BNF.

In the categorical semantic approach, we define

- "objects" (types) A, B, C
- "morphisms" (programs) $f : A \to B$
- composition

$$\frac{f : A \to B \qquad g : B \to C}{f\,;g : A \to C}$$

London 1994: Semantics of Computation

1.2 The Interaction Category Paradigm

We propose *Interaction Categories* as a new paradigm for the semantics of computation. In place of sets, functions, and function composition, an Interaction Category is a semantic universe where

- Types are *process specifications* A, B, C
- Morphisms are *processes* $p : A \to B$
- Composition is *interaction*

$$\frac{p : A \to B \qquad q : B \to C}{p\,;q : A \to C}$$

Sta Maria 1997: Category Theory in Computer Science

Specifying Interaction Categories

D. Pavlović*[1] and S. Abramsky[2]

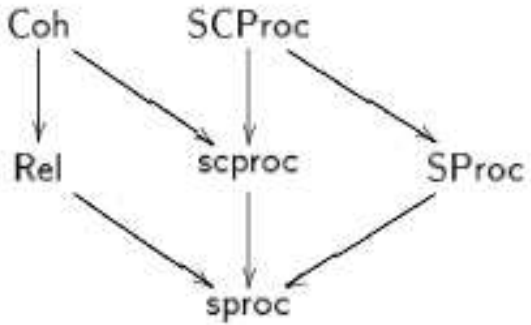
...

$$\Phi\{CR\}\Psi \iff \forall \alpha\alpha' \in A\beta\beta' \in B.\ (\alpha R\beta \wedge \alpha' R\beta') \Rightarrow (\alpha\Phi\alpha' \Rightarrow \beta\Psi\beta'). \quad (17)$$

The total category $\mathsf{Rel}^{\mathsf{C}}$ will be the familiar category Coh of coherence spaces [14]. By imposing on each set of traces $S \subseteq A^*$ (or on labelled trees, or transition systems) the coherence requirement

$$s\alpha, s\alpha' \in S \Longrightarrow \alpha\Phi\alpha' \quad (18)$$

for all $\alpha \neq \alpha' \in A$, all previously described specifications lift to Coh, and yield interaction categories with a grain of true concurrency. It is interesting to notice that already the synchronous ones can be specified in many different, meaningful ways.



(19)

Palo Alto 2000: Evolving Specifications

Composition and Refinement of Behavioral Specifications

Dusko Pavlovic and Douglas R. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304 USA

Abstract

This paper presents a mechanizable framework for specifying, developing, and reasoning about complex systems. The framework combines features from algebraic specifications, abstract state machines, and refinement calculus, all couched in a categorical setting. In particular, we show how to extend algebraic specifications to evolving specifications (especs) in such a way that composition and refinement operations extend to capture the dynamics of evolving, adaptive, and self-adaptive software development, while remaining efficiently computable. The framework is partially implemented in the Epoxi system.

1 Introduction

Especs grew out of higher-order algebraic specifications as implemented in Specware [11], the evolving algebras of Gurevich (aka abstract state machines) [4], as well as the classical axiomatic semantics of Floyd/Hoare/Dijkstra. Especs go beyond all three, not only allowing the capture of logical structure and behavior, but also the composition of systems and their refinement to code. Of course the composition and refinement operations are meaning-preserving, so that any code produced by means of composition and refinement is guaranteed to be consistent with the initial especs.

The paper is structured straightforwardly. We first discuss how to extend logical specifications to model behavior, and then define especs and how to refine

Palo Alto 2000: Evolving Specifications

```
espec GCD-0 is
  import GCD-base
  spec                    ;; the spec extends the spec from GCD-base with a theorem
    thm gcd(x,x) = x      ;; this theorem follows from axiom gcd-spec
  end-spec

  prog                    ;; the keyword prog encloses the state machine
    stad One init[X-in,Y-in] is    ;; the initial state receives X-in and Y-in
    end-stad

    stad Two fin[Z] is     ;; this stad extends the global spec with a local axiom
       axiom Z = gcd(X-in,Y-in)
    end-stad

    step Out : One -> Two is    ;; transition from stad One to stad Two
      Z |-> gcd(X-in,Y-in)
    end-step
  end-prog
end-espec
```

Note that the steps are expressed in terms of symbol translations. Because of the connection between translations and transitions, we will henceforth use assignments instead; i.e. write `x := e` instead of `x |-> e`.

```
  axiom Z = gcd(X-in,Y-in)
end-stad

step initialize : One -> Loop is
  X := X-in
```

Palo Alto 2000: Evolving Specifications

3 Especs

The concept of espec is now formally defined.

Definition 3.1 *A* graph s *consists of two sets* edge_s *and* node_s*, and two functions,* dom_s *and* cod_s *from* edge_s *to* node_s*.*

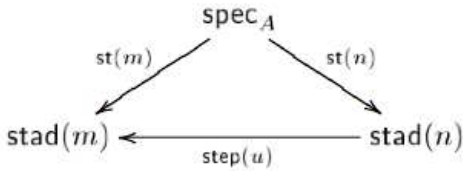
A shape *is a graph* s*, which is moreover*

- *reflexive, in the sense that there is a function* $\mathsf{id}_s : \mathsf{node}_s \longrightarrow \mathsf{edge}_s$*, which assigns a distinguished loop to each node;*
- *distinguished initial node* i*, and a set* O *of final nodes* o*;*

Together with the morphisms preserving all displayed structure, shapes form the category Shape.

Definition 3.2 *An* evolving spec*, or* **espec** *A consists of*

- *a spec* spec_A *and*
- stad assigns to each shape-node n a *state description* $\mathsf{stad}(n)$, which comes with a translation $\mathsf{st}_A(n) : \mathsf{spec}_A \longrightarrow \mathsf{stad}(n)$;
- step assigns to each shape-edge $u : m \to n$ a *step* (or *transition*) $\mathsf{step}(u) : \mathsf{stad}(m) \leftarrow \mathsf{stad}(n)$, keeping S invariant, in the sense that the following diagram commutes.



4 Refinements

We now define the concept of a refinement (or morphism) between two especs. A characteristic of espec refinements is that logical structure and behavior refine contravariantly, in opposite directions. If A refines

Palo Alto 2000: Evolving Specifications

spec_A and spec_B to their extensions stad_A and stad_B. Just like spec_B refines spec_A because it proves all formulas in the image $f_{\mathsf{spec}}[\mathsf{spec}_A]$, each $\mathsf{stad}_B(n)$ refines $\mathsf{stad}_A(f_{\mathsf{shape}}(n))$ because it proves all formulas in the image $f_{\mathsf{stad}}(n)[\mathsf{stad}_A(f_{\mathsf{shape}}(n))]$. The *structural* refinement is thus extended from $f_{\mathsf{spec}} : \mathsf{spec}_A \longrightarrow \mathsf{spec}_B$ to $f_{\mathsf{stad}} : \mathsf{stad}_A \longrightarrow \mathsf{stad}_B$. Its naturality ensures that each transition $\mathsf{step}_B(v)$ of B extends the transition $\mathsf{step}_A(f_{\mathsf{shape}}(v))$ of A.

The guard condition ensures that every behavior of B maps to a behavior of A. There are stronger versions of the guard condition that also ensure that B simulates all of A's behaviors, and others that eliminate nondeterminism. Rather than commit to one such definition, we use several, but the guard condition above is sufficient for the purposes of this paper.

Let us return to the example in Section 2. In the refinement from GCD-0 to GCD-1, f_{spec} is a simple inclusion, and f_{shape} is given by the stad map

$$\begin{array}{rcl} One & \mapsto & One \\ Loop & \mapsto & One \\ Two & \mapsto & Two \end{array}$$

condition is also straightforward; e.g. for step Loop1 in GCD-1, the guard condition instantiates to

$$Loop \vdash X > Y \implies true$$

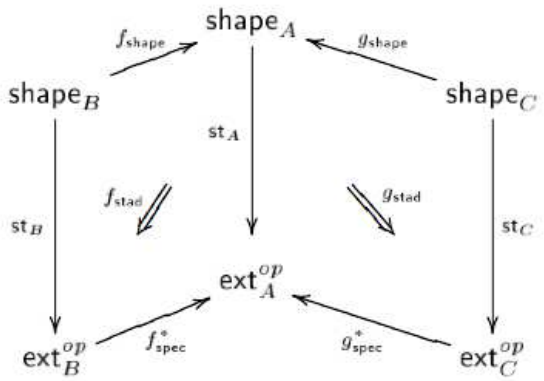
where the consequent is the guard on step id_{One} in GCD-0.

5 Colimits

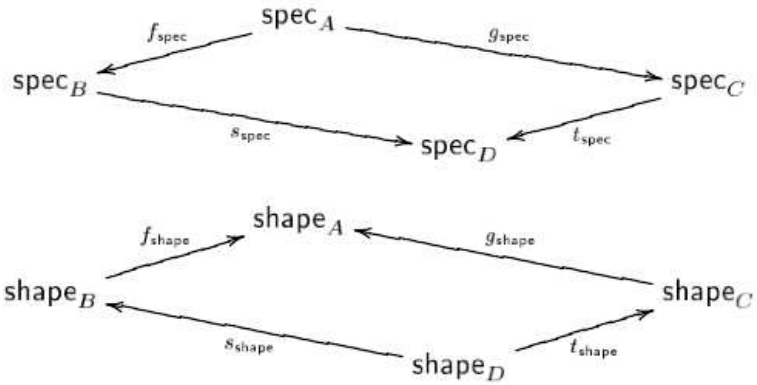
Composition of especs is carried out by the colimit operation. Colimits in ESpec are constructed from the colimits in Spec, the limits in Shape, plus some wiring to connect them in Cat. First of all, recall that all colimits can be derived from the initial object and the pushouts. Of course, the initial espec consists of the empty spec, and a one-state-one-step program (with the state represented by the empty spec).

To describe the pushout of especs, suppose we are

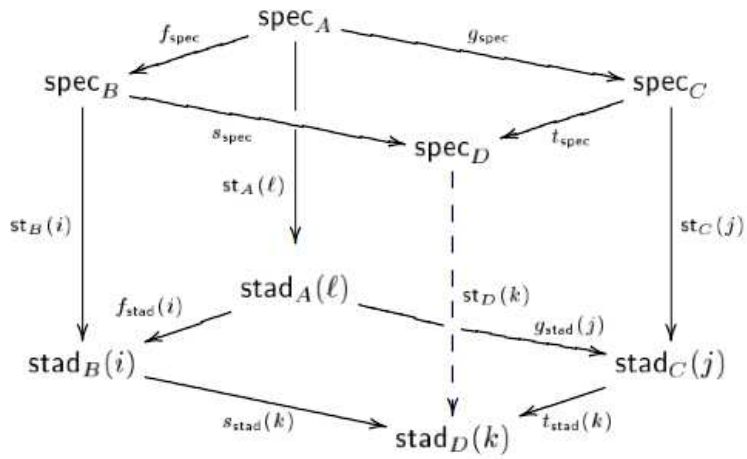
Palo Alto 2000: Evolving Specifications



To compute the pushout, we first compute the corresponding pushout of specs and the pullback of shapes.



It is easy to see that $M : \mathsf{Spec}^{op} \longrightarrow \mathsf{Cat}$ maps the



Since shape_D is the pullback of f_{shape} and g_{shape}, the node k corresponds to a pair $\langle i, j\rangle$ of the nodes from shape_B and shape_C, identified in shape_A as the node $\ell = f_{\mathsf{shape}}(i) = g_{\mathsf{shape}}(j)$. Of course, $i = s_{\mathsf{shape}}(k)$ and $j = t_{\mathsf{shape}}(k)$.

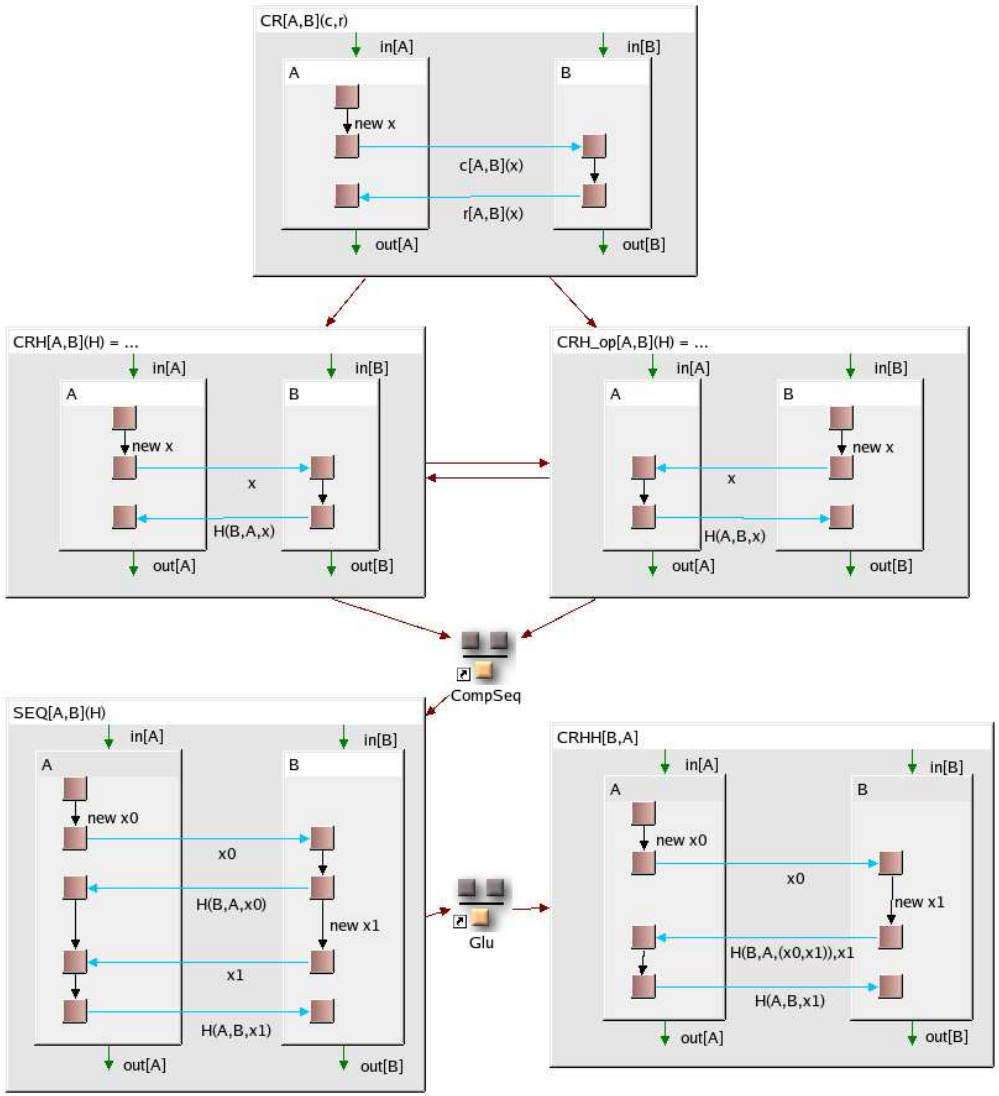
This construction gives the node part stad_D of $\mathsf{st}_D : \mathsf{shape}_D \longrightarrow \mathsf{ext}^{op}_D$, as well as the components of s_{stad} and t_{stad}. The arrow part step_D is induced by the fact that the bottom of the cube is a pushout, using the naturality of f_{stad} and g_{stad}. This also yields the naturality of s_{stad} and t_{stad}. Finally we construct the guards for the edges of shape_D. Given an edge $w : k \to k'$ of shape_D define

Palo Alto 2000: Evolving Specifications

Category of specifications

- *objects*: specifications
- *morphisms*: processes / interpretations
- *colimits*: composite specifications
- *limits*: composite processes

Palo Alto 2000–2006: Specifying Security



PDA (Protocol Derivation Assistant)

Question 1

What are categories good for?

Answer 1

Categories are good for working mathematicians

Answer 1

Categories are good for working mathematicians:

- general abstract nonsense

Answer 2

Categories are good for working computer scientists:

- **concrete** abstract nonsense

Answer 2

Categories are good for working computer scientists:

- **concrete** abstract nonsense
  - variants:
    general concrete nonsense,
    particular abstract nonsense. . .

Outline

Question 2

Are working computer scientists good for categories?

Zürich 1966

A collection of informal reports and seminars
Edited by A. Dold, Heidelberg and B. Eckmann, Zürich

Series: Forschungsinstitut für Mathematik, ETH, Zürich · Adviser: K. Chandrasekharan

24

Joachim Lambek

McGill University, Montreal
Forschungsinstitut für Mathematik, ETH, Zürich

Completions of Categories

Seminar lectures given 1966 in Zürich

1966

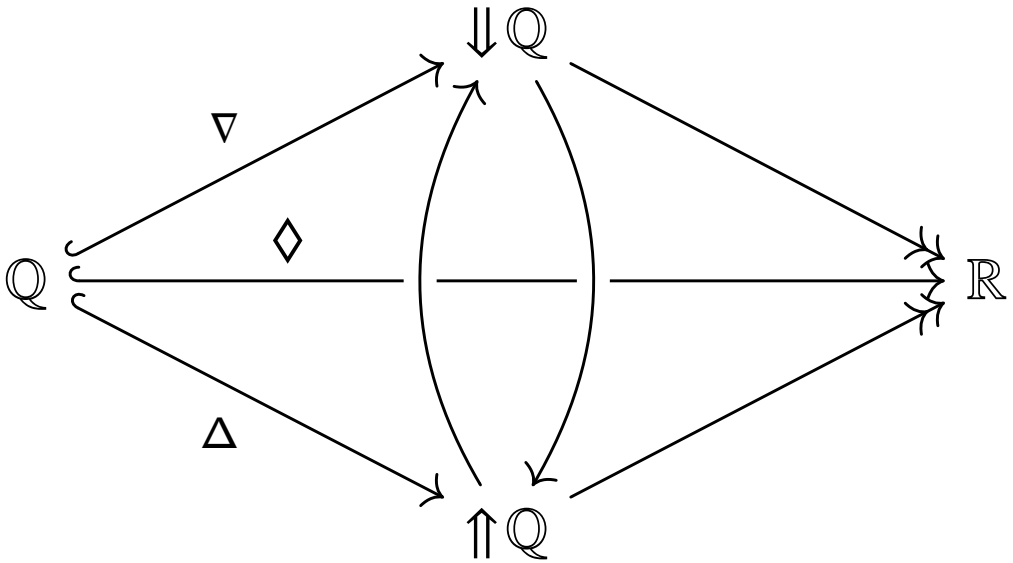
Springer-Verlag · Berlin · Heidelberg · New York

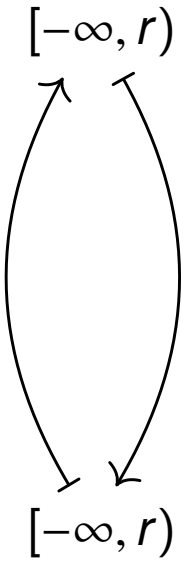
Zürich 1966

It is an open problem whether there exists a sup- and inf-complete category $\underline{A}''''$ with a sup- and inf-dense embedding $\underline{A} \to \underline{A}''''$, in analogy to the Dedekind completion of an ordered set.

Dedekind completion of rational numbers

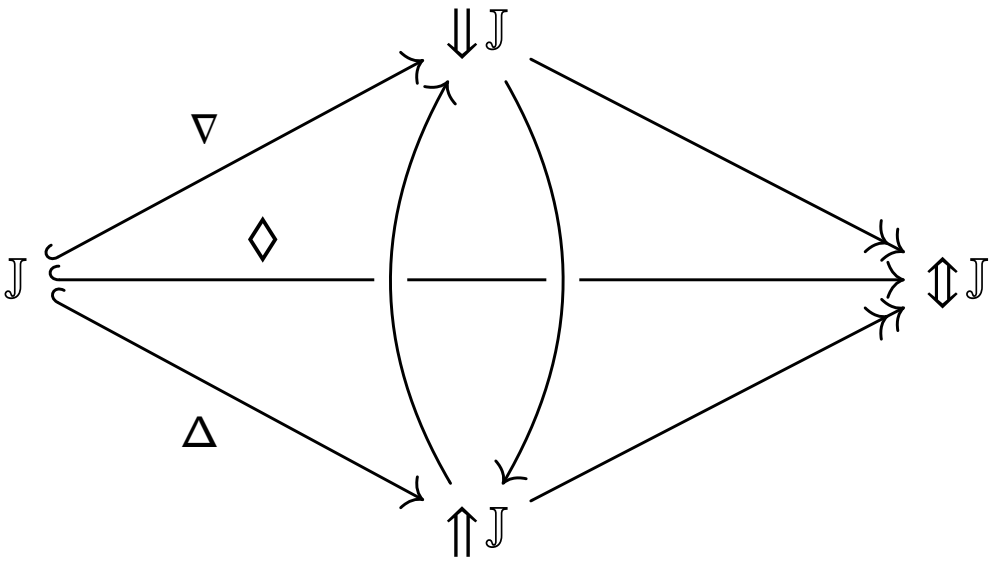
Construct the reals:

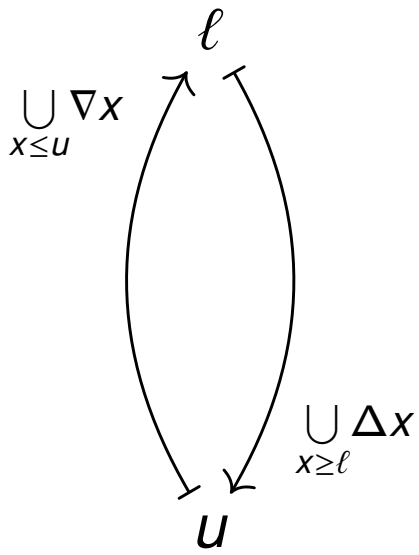




MacNeille completion of a poset

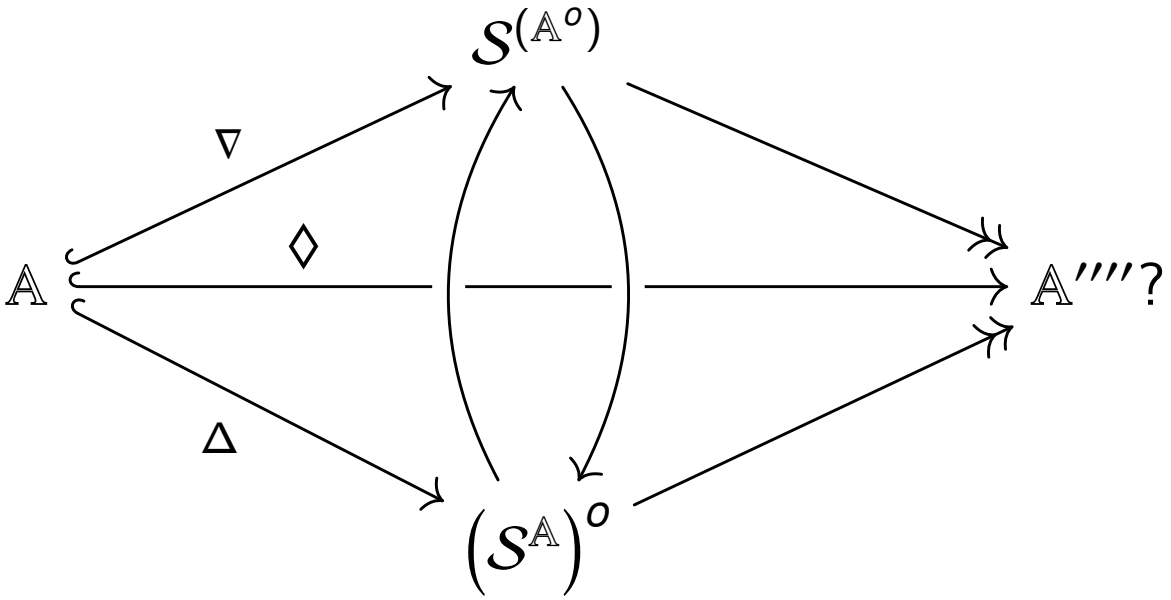
Adjoin $\bigwedge$ and $\bigvee$ — preserve those that exist

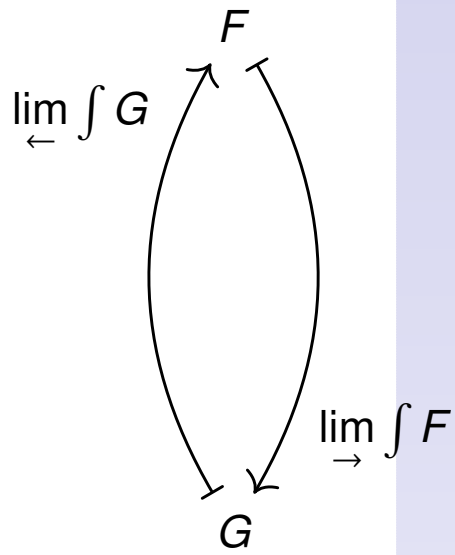




Lambek's question

Adjoin $\varinjlim_{\leftarrow}$ and $\varinjlim_{\rightarrow}$ — preserve those that exist





Isbell's answer

Small Subcategories and Completeness

by

JOHN R. ISBELL

Case Western Reserve University
Cleveland, Ohio

. . .

Lambek has asked [7] whether every small category $\mathscr{A}$ has a small-complete extension $\mathscr{C}$ in which every object is both a limit and a colimit of objects of $\mathscr{A}$. Such an extension is normal and has all the strong completeness properties (and sharpness). I think they merit study and propose to call them (small-complete) *Lambek extensions*. However, our immediate concern is negative.

3.1. *No Lambek extension of the one-object category Z_4 has finite limits.*

Proof. For any group $\mathscr{A}$, call a diagram $\mathscr{D} \to \mathscr{A}$ that is constant on each

Intuition

Category of specifications

- *objects*: specifications
- *morphisms*: transitions / interpretations
- *colimits*: composite specifications
- *limits*: composite processes

Intuition

Category of specifications

- *objects*: specifications
- *morphisms*: transitions / interpretations
- *colimits*: composite specifications
- *limits*: composite processes

Bicompletion

- adjoins all derivable concepts
- preserving all specified concepts

Categorical mystery

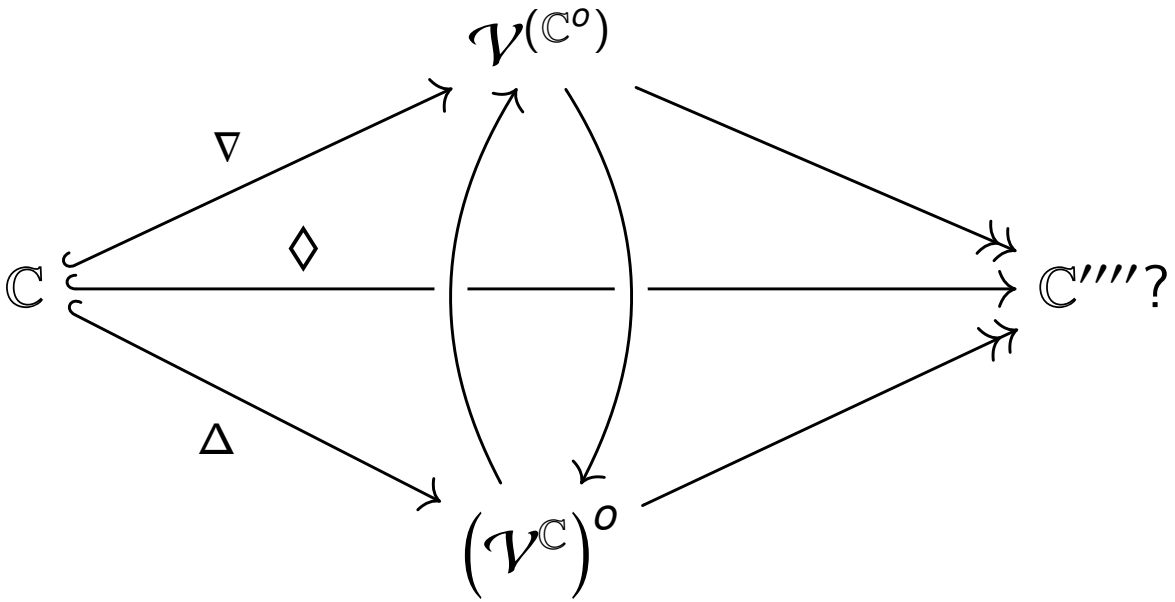
We cannot conservatively extend

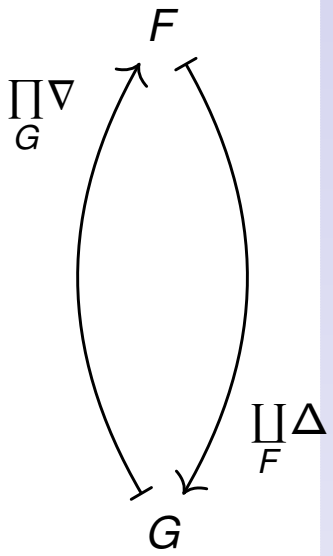
- all specified concepts by
- all derivable concepts

in a category.

Bicompletion of an enriched category

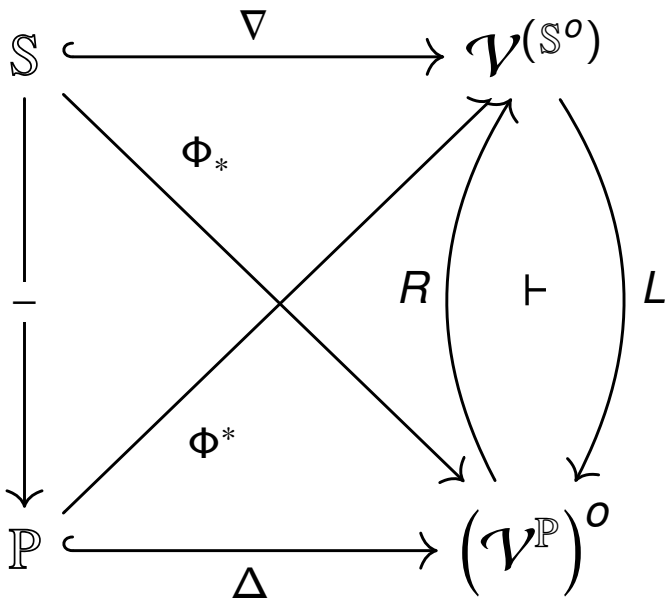
Adjoin $\prod_W$ and $\coprod_W$ — preserve those that exist

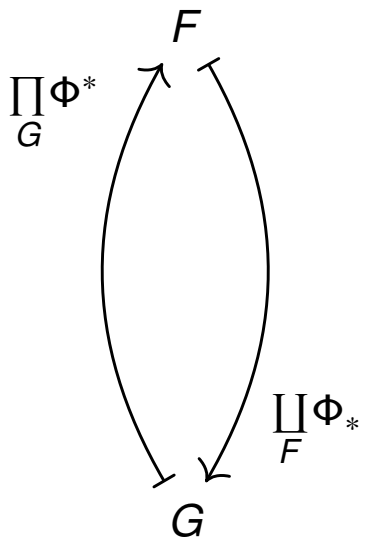




Bicompletion of matrices

Cocomplete specifications, complete processes





of the matrix $\mathbb{S}^o \times \mathbb{P} \xrightarrow{\Phi} \mathcal{V}$

Remark

A category $\mathbb{C}$ is a square matrix

$$\begin{array}{rcl} \mathbb{C}^o \times \mathbb{C} & \longrightarrow & \mathcal{V} \\ \langle a, b \rangle & \mapsto & \mathbb{C}(a, b) \end{array}$$

What do we know about bicompletions

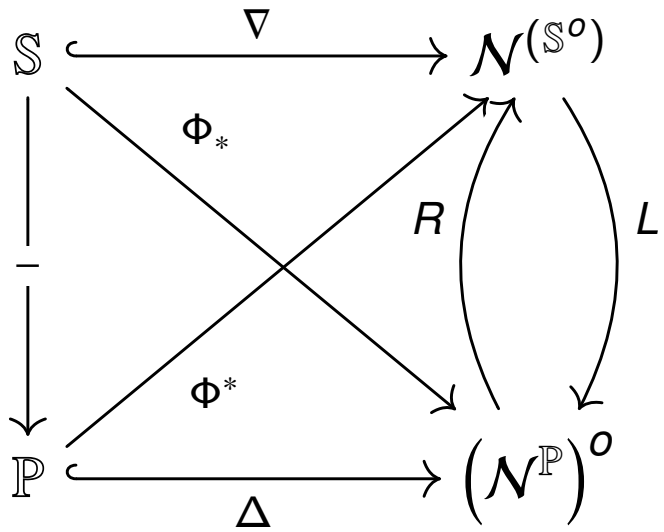
$(\mathcal{V}, \longrightarrow, \otimes)$	$\mathcal{V}$-cat. bicompl.	$\mathcal{V}$-matrix bicompl.
$(\{0 < 1\}, \leq, \wedge)$	Dedekind-MacNeille	Formal Concept Analysis
$([0 < 1], \leq, \times)$	ICFCA 2012	
$([0 < \infty], \geq, +)$	Samson 60	
$([\mathcal{S}, \longrightarrow, \times)$	Isbell: may not exist	????

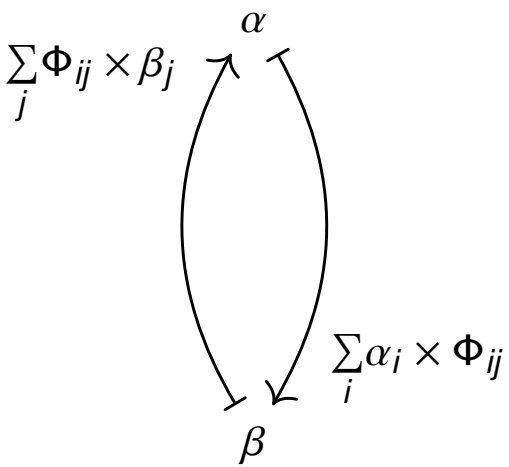
Concrete abstract nonsense

Consider $\left(\mathcal{N}, \longrightarrow, \otimes\right)$-enriched categories where

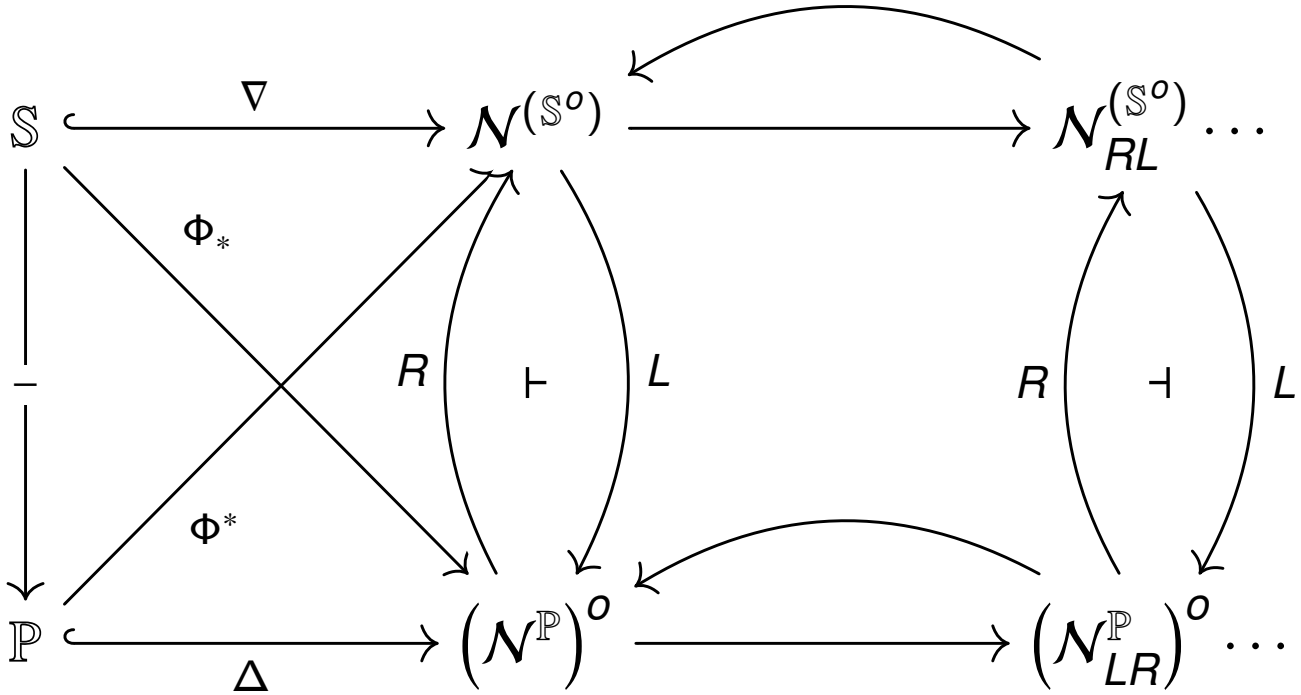
$$\begin{aligned} |\mathcal{N}| &= \left\{n = \{0, 1, \ldots, n-1\}\right\} \\ \mathcal{N}(m, n) &= \mathbb{N}^{m \times n} \\ m \otimes n &= \{0, 1, \ldots, m \times n - 1\} \end{aligned}$$

Concrete abstract nonsense





Concrete abstract nonsense



Concrete abstract nonsense

Latent Semantic Analysis in Categories

- Spectral Decomposition of RL and LR
- Singular Value Decomposition of Φ

Outline

Claim

Categories are good for friendship.