Fixed points in programming: datatypes and protocols.

J.R.B. Cockett

Department of Computer Science
University of Calgary
Alberta, Canada

robin@cpsc.ucalgary.ca

(work with Subashis Chakraborty)

14th Colloquiumfest Saskatoon: February 2014

Where are we going?

1. Tensions from language design.
2. Datatypes deliver computation ...
3. Communication on a channel.
4. Polycategories and representability.
5. Communication on many channels.
6. Message passing.
7. Protocols.

Tensions from language design

- Legacy or lean?
- Language or library?
- Domain or dominant?
- Formal or fun?
- Network or not work?

Networks ...

In the 1970's networks, parallel, and distributed computing arrived and was going to solve everything!
Practitioners pushed back with "the fallacies"
(Joy, Lyon, Deutsch, Gosling):

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite
- The network is secure.
- Topology doesn't change.
- Transport cost is zero.
- The network is homogeneous.

Computing had blindly entered a new world of expectation and connectedness!

There was no turning back ...

Computing ...

- In the 1960's computing power and memory was expensive ...
  measurement: bits and operations per second.
  One GFLOP cost US1.1 trillion.
- By 2000 memory was dirt cheap and processors powerful ...
  measurement: moved from bits to gigabytes ....
  One GFLOP cost US1000.
- By 2013 multicore (4 or 8 cpu) is common ...
  measurement: moved to terabytes (32 bit addressing)
  One GFLOP cost US0.75.
- By 2050 kilocore and gigacore will be common ...

Computing has blindly entered a new era of parallel power!

There is no turning back ...

Implications for language design ...

Programming languages
need to support

Concurrent, Massively Parallel

and

Interactive Computing!

Practice ahead of theory ...

- Where is the mathematics of processes, concurrency, communication?
- Is this theory only develop in response to practice?
- Should theory simply be modelling practice?!!?
- Is there a need to develop new theory? ...
  .... or is it just taking time to link existing theory and practice?

Does mathematics have anything useful to say?

A brief history of process semantics ...

- Petri nets, C. A. Petri (1962).
- Communicating Sequential Processes, C. A. R. Hoare (1978).
- Calculus of Communicating Processes, R. Milner (1979) [book (1989)].
- Algebra of Communicating Processes (ACP), J. Bergstra and J. W. Klop (1982).
- Robin Milner's quest to find the "λ-calculus of concurrency" produced the π-calculus with J. Parrow (1992) [book (1999)].
- Others: ambient calculus (L. Cardelli, A.D. Gordon), PEPA (J. Hillston), the fusion calculus (J. Parrow and B. Victor), the spy calculus (M. Abadi and A. Gordon), ...

The complaint ...

What are the fundamental structures of concurrency?
We still don't know!'

"Is this profusion a scandal of our subject: I used to think so
... now I am not so sure."

Samson Abramsky (2005)

The complaint ....

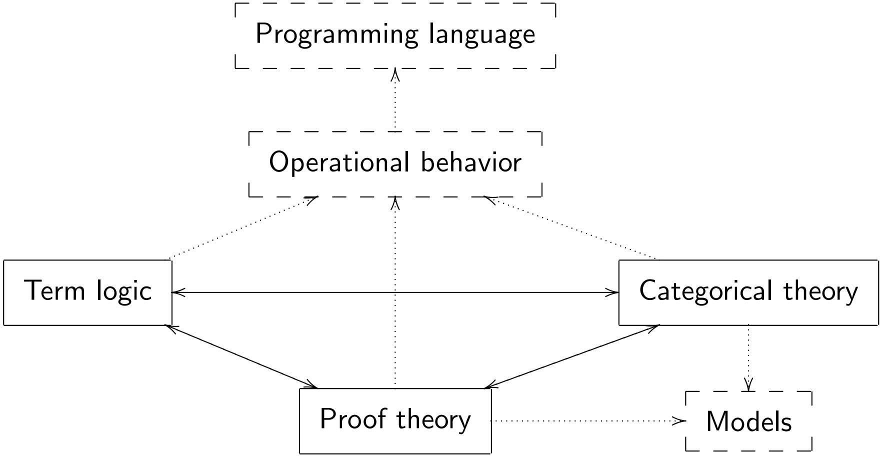
- No Church/Turing thesis for concurrency ...
- A tool kit: no unified theory ...
- Plasticity of definition, carvings in snow: no bedrock ...
- A profusion of syntax but no semantics ...
- Physics (quantum computing), biological computing, and environmental modelling are at our gates: what do we have to show?

Should we expect more than a tool kit?

- Tools are good: bisimulation, hiding, scope extrusion, ...
- The subject covers a wide range of phenomena ...

... of course we should expect more ...

So what is a good process semantics?



So what is a good process semantics?

- Proof theory: behaviour of the free term model:
  - Term construction
  - Type inference and checking.
  - Compositional behaviour from cut elimination.
- A categorical semantics:
  - Universal constructs (properties versus structure).
  - Rules of equality (local to support program transformations).
  - Compositional semantics (modular program construction).
  - Coherent framework (determines “feature” interactions).
  - Interface to mathematics (models with different properties).
  - Term logic = programming language.
  - Operational semantics = (reasonably) efficient evaluation.

So where are we?

CLAIM: The mathematical structures are there!!

... BUT the Computer Science is not!

... this is where the rubber hits the road!

... and the mathematical development is part of this!

2.

Datatypes deliver computation ...

(A basis for "strong functional programming"!)

Datatypes deliver computation

In mathematical settings computation may be delivered by rewriting ....

... and a disciplined way of adding rewriting rules is to introduce initial and final fixed points.

In the sequential world these are called *datatypes* ...

Datatypes deliver *progressive* computation ...
(i.e. essentially terminating).

Inductive datatypes

$\mathbf{X}$ a category and $F : \mathbf{X} \longrightarrow \mathbf{X}$ an endofunctor. An *inductive datatype* (or **least fixed point**) for F is an object $\mu x.F(x) \in \mathbf{X}$ together with a map

$$\mathsf{cons}_F : F(\mu x.F(x)) \longrightarrow \mu x.F(x)$$

satisfying the *inductive axiom*: Given $Z \in \mathbf{X}$ and a map $g : F(Z) \longrightarrow Z$ then there is a unique map $\{\!|g|\!\}_F$, such that

$$\begin{tikzcd}
F(\mu x.F(x)) \arrow[r, "\mathsf{cons}_F"] \arrow[d, "F(\{\!|g|\!\}_F)"'] & \mu x.F(x) \arrow[d, "\{\!|g|\!\}_F"] \\
F(Z) \arrow[r, "g"'] & Z
\end{tikzcd}$$

commutes.

Coinductive datatypes (dual)

$\mathbf{X}$ a category and $F : \mathbf{X} \longrightarrow \mathbf{X}$ an endofunctor. An *coinductive datatype* (or **greatest fixed point**) for F is an object $\nu x.F(x) \in \mathbf{X}$ together with a map

$$\mathrm{dest}_F : \nu x.F(x) \longrightarrow F(\mu x.F(x))$$

such that the *coinductive axiom* holds: Given $Z \in \mathbf{X}$ and a map $g : Z \longrightarrow F(Z)$ then there is a unique map $(\!|g|\!)_F$, such that

$$\begin{tikzcd}
Z \arrow[r, "g"] \arrow[d, "(\!|g|\!)_F"] & F(Z) \arrow[d, "F((\!|g|\!)_F)"] \\
\mu x.F(x) \arrow[r, "\mathrm{dest}_F"'] & \mu x.F(x)
\end{tikzcd}$$

commutes.

Proof rules for fixed points

The pure "Kozen rules" for fixed points are:

$$\frac{X \vdash F(\mu x.F(x))}{X \vdash \mu x.F(x)} \qquad \frac{G(\nu y.G(y)) \vdash Y}{\nu y.G(y) \vdash Y}$$

$$\frac{F(X) \vdash X}{\mu x.F(x) \vdash X} \qquad \frac{Y \vdash G(Y)}{Y \vdash \nu y.G(y)}$$

Proof rules for fixed points in inuitionistic logic

In intuitionistic logic want the Kozen rules to become:

$$\frac{\Gamma \vdash F(\mu x.F(x))}{\Gamma \vdash \mu x.F(x)} \qquad \frac{\Gamma, F(X) \vdash X}{\Gamma, \mu x.F(x) \vdash X} \quad \text{etc.}$$

where Γ is a "context". To obtain this from the pure rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Z, Z \Rightarrow X \vdash X}\,\text{eval}}{F(Z \times Z \Rightarrow X) \vdash F(X)}\,\text{functor}}{Z, F(Z \Rightarrow X) \vdash F(X)}\,\text{strength} \qquad Z, F(X) \vdash X}{F(Z \Rightarrow X) \vdash Z \Rightarrow X}}{\mu x.F(x) \vdash Z \Rightarrow X}}{\overline{Z, \mu x.F(x) \vdash X}}$$

one needs a *higher-order* setting and *strength* of the functor.

Circular rules ...

The Kozen rules are not very convenient for cut-elimination or for writing programs ...

To facilitate the description of evaluation we use a different (but more complicated) equivalent form ...

... which is reminiscent of a general recursive program (but can be type checked to ensure progress).

Inductive circular rule

A combinator:
$$\frac{f : X \longrightarrow B}{c[f] : F(X) \longrightarrow B}$$

where
$$\begin{tikzcd} F(X) \arrow[rr, "F(r)"] \arrow[dr, "{c[x]}"'] & & F(X') \arrow[dl, "{c[x']}"] \\ & B & \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} X \arrow[rr, "r"] \arrow[dr, "x"'] & & X' \arrow[dl, "x'"] \\ & B & \end{tikzcd}$$

delivers a **circular map** $\mu a.c[a] : \mu x.F(x) \longrightarrow B$ such that the following diagram commutes

$$\begin{tikzcd} F(\mu x.F(x)) \arrow[rr, "\mathsf{cons}"] \arrow[dr, "{c[h]}"'] & & \mu x.F(x) \arrow[dl, "h"] \\ & B & \end{tikzcd}$$

if and only if $h = \mu a.c[a]$. In particular cons$\mu a.c[a] = c[\mu a.c[a]]$.

Coinductive circular rule (dual)

Dually we have for coinductive datatypes the following circular style definition. Given a combinator

$$\frac{B \xrightarrow{f} X}{G(B) \xrightarrow[c[f]]{} X}\ c[\_]$$

where B is a fixed object in **X**, there is a **cocircular map** $\nu b.c[b] : B \longrightarrow \nu x.G(x)$ such that

$$\begin{tikzcd} & B \arrow[dl, "u"'] \arrow[dr, "c[u]"] & \\ \nu x.G(x) \arrow[rr, "\mathrm{dest}"'] & & G(\nu x.G(x)) \end{tikzcd}$$

commutes iff $u = \nu b.c[b]$. In particular $(\nu b.c[b])\mathrm{dest} = c[\nu b.c[b]]$.

See also Tarmo Uustalu and Varmo Vene (thesis).

A circular definition ...

This allows a natural style of programming ...

$$\begin{aligned} \text{data list}(A) \to C = {} & \text{nil: } 1 \to C \\ & \text{cons: } A \times C \to C \end{aligned}$$

```
fold append : $\mathrm{list}(A), \mathrm{list}(A) \to \mathrm{list}(A) =$
    $x, y$ by $x$ as
            nil $\to y$
            cons$(a, x') \to$ cons$(a, \mathrm{append}(x', y))$
```

The rewriting rules ...

$$\mathrm{append}(\mathrm{nil}, y) \Rightarrow y$$

$$\mathrm{append}(\mathrm{cons}(a, x'), y) \Rightarrow \mathrm{cons}(a, \mathrm{append}(x', y))$$

This (together with high-order features) gives an expressive and "useable" strong functional programming language.

... one has all the power of higher-order primitive recursion BUT programs must now "show why they terminate": in practice this can make for inefficient algorithms ....

Pragmatic solution: warn the user when he defines a function which cannot be type-inferred to terminate ....

3.

Communication on a channel

(A bit of bedrock for concurrency!)

Products and coproducts

- Abelian groups, suplattices, relations: $A + B = A \times B$ (biproducts)
- Sets, topoi, cartesian closed categories, extensive and distributive categories
  - $A + B = A \sqcup B$ (disjoint union)
  - $A \times B$ (cartesian product)
  - $A \times (B + C) \cong (A \times B) + (A \times C)$

In all these settings the product and coproducts satisfy some very special properties!

Products and coproducts

But what does $\Sigma\Pi(\mathbf{A})$ the category with free products and coproducts generated by the category $\mathbf{A}$ look like?

Andre Joyal: *Free bicomplete categories.*
Cockett and Seely: *The logic of sums and products* $\Sigma\Pi$

Products

$$\begin{tikzcd}
 & & Y_1 \\
X \arrow[rru, "g_1"] \arrow[r, dotted, "(g_i)_i"'] \arrow[rrd, "g_n"'] & \prod_i Y_i \arrow[ru, "\Pi_1"'] \arrow[rd, "\Pi_n"] & \vdots \\
 & & Y_N
\end{tikzcd}$$

$$\begin{aligned}
\Pi_k(f);g &= \Pi_k(f;g) \\
f;(g_i)_{i\in I} &= (f;g_i)_{i\in I} \\
(g_i)_{i\in I};\Pi_k(f) &= g_k;f
\end{aligned}$$

where

$$\begin{tikzcd}
\prod_i Y_i \arrow[r, "\Pi_k"] \arrow[rd, "\Pi_k(f)"'] & Y_k \arrow[d, "f"] \\
 & Z
\end{tikzcd}$$

$$\begin{aligned}
\Pi_k(f) &:= \Pi_k;f \\
\Pi_k &= \Pi_k(1_{Y_k})
\end{aligned}$$

Coproducts

$$\begin{tikzcd}
X_1 \arrow[dr, "\amalg_1"'] \arrow[drr, "g_1"] & & \\
\vdots & \coprod_i X_i \arrow[r, dotted, "\langle g_i \rangle"'] & Y \\
X_n \arrow[ur, "\amalg_n"] \arrow[urr, "g_n"'] & &
\end{tikzcd}$$

$$\begin{aligned}
f;\amalg_k(g) &= \amalg_k(f;g)\\
\langle f_j\rangle_{j\in J};g &= \langle f_j;g\rangle_{j\in J}\\
\amalg_k(f);\langle g_j\rangle_{j\in J} &= f;g_k
\end{aligned}$$

where

$$\begin{tikzcd}
X_k \arrow[r, "\amalg_k"] & \coprod_j Y_j \\
Z \arrow[u, "f"] \arrow[ur, "\amalg_k(f)"'] &
\end{tikzcd}$$

$$\begin{aligned}
\amalg_k(f) &:= f;\amalg_k\\
\amalg_k &= \amalg_k(1_{X_k})
\end{aligned}$$

Interactions

$$\begin{tikzcd}
X_1 \arrow[r, "h_{11}"'] \arrow[ddr, "h_{n1}"] & Y_1 \\
\vdots & \vdots \\
X_n \arrow[r, "h_{nm}"'] \arrow[uur, "h_{1m}"'] & Y_n
\end{tikzcd}$$

gives $\coprod_i X_i \xrightarrow{\langle (h_{ij})_i \rangle_j = (\langle h_{ij} \rangle_j)_i} \prod_j Y_j$

and the basic equalities:

$$\Pi_i(\amalg_j(f)) = \amalg_j(\Pi_i(f)) \qquad \Pi_k((g_i)_i) = (\Pi_k(g_i))_i$$

$$\amalg_k(\langle f_j \rangle_j) = \langle \amalg_k(f_j) \rangle_j$$

Logic of products and coproducts

$$\frac{}{A \vdash_{1_A} A}\ \text{id}$$

$$\frac{\{X_j \vdash_{f_j} Y\}_{j\in J}}{\coprod_j X_j \vdash_{\langle f_j\rangle_{j\in J}} Y}\ \text{cotuple} \qquad \frac{\{X \vdash_{g_i} Y_i\}_{i\in I}}{X \vdash_{(g_i)_{i\in I}} \prod_i Y_i}\ \text{tuple}$$

$$\frac{X \vdash_f Y_k}{X \vdash_{\amalg_k(f)} \coprod_{i\in I} Y_i}\ \text{coproj} \qquad \frac{X_k \vdash_f Y}{\prod_{i\in I} X_i \vdash_{\Pi_k(f)} Y}\ \text{proj}$$

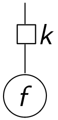
$$\frac{X \vdash_f Y \quad Y \vdash_g Z}{X \vdash_{f;g} Z}\ \text{cut}$$

Cut elimination

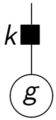
... is rewriting modulo equations:

$$\begin{array}{rcl}
f;1 & \Longrightarrow & f \\
1;f & \Longrightarrow & f \\
f;\amalg_k(g) & \Longrightarrow & \amalg_k(f;g) \\
\sqcap_k(f);g & \Longrightarrow & \sqcap_k(f;g) \\
\langle f_i\rangle_i;g & \Longrightarrow & \langle f_i;g\rangle_i \\
f;(g_i)_i & \Longrightarrow & (f;g_i)_i \\
\amalg_k(f);\langle g_i\rangle_i & \Longrightarrow & f;g_k \\
(f_i)_i;\sqcap_k(g) & \Longrightarrow & f_k;g
\end{array}
\qquad
\begin{array}{rcl}
\amalg_k(\langle f_j\rangle_j) & \Longleftrightarrow & \langle \amalg_k(f_j)\rangle_j \\
\sqcap_k((f_i)_i) & \Longleftrightarrow & (\sqcap_k(f_i))_i \\
\sqcap_i(\amalg_j(f)) & \Longleftrightarrow & \amalg_j(\sqcap_i(f)) \\
(\langle f_{ij}\rangle_i)_j & \Longleftrightarrow & \langle (f_{ij})_j\rangle_i
\end{array}$$

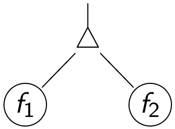
Process reading ...



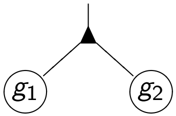
Output "k" on the right $= \amalg_k(f)$



Output "k" on the left $= \Pi_k(g)$



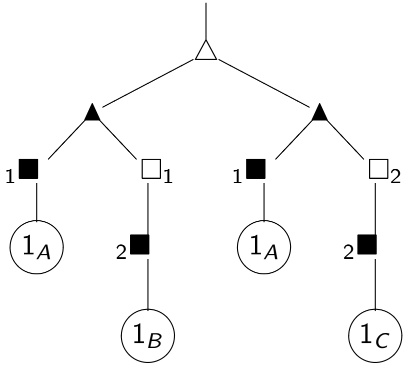
Listen for input on the left $= \langle f_1, f_2 \rangle$



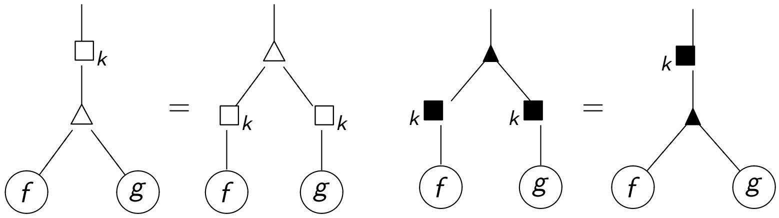
Listen for input on the right $= (g_1, g_2)$

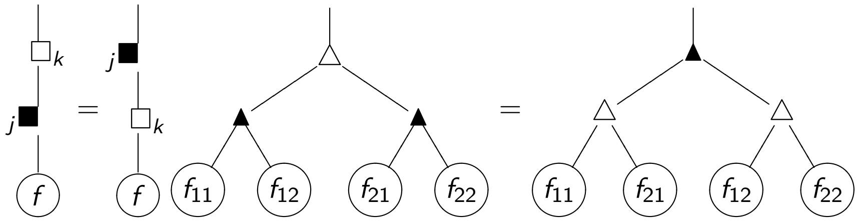
Process reading of a map ...

$$(A \times B) + (A \times C) \xrightarrow{\langle(\Pi_1(1_A),\amalg_1(\Pi_2(1_B))),(\Pi_1(1_A),\amalg_2(\Pi_2(1_C)))\rangle} A \times (B + C)$$



Process reading of the identities ...





Slogan

The proof theory of products and coproducts

IS

the basic calculus of communication on a channel!!

Joyal: "... mathematics is saying something."

Different readings ...

TYPE	CATEGORY	PROOF	PROCESS	GAME
Type	Object	Proposition	Protocol	Game
Terms	Map	Proof	Process	Mediator
Substitute	Compose	Cut	Communicate	Compose
Variables	Identities	Axioms	Relay	Copy cat

Joyal and Santocanale used the reading of games (Blass) ...
Cockett and Seely used the reading of proofs and categories ...
Pastro used the reading as protocols and processes ...

... just products and coproducts ...

History:

(1) Cockett and Seely, *Finite sum-product logic*, TAC 8, (2001)
Not everything was sorted out!! No decision procedure for units ...

(2) Cockett and Santocanale, *On the word problem for $\Sigma\Pi$-categories, and the properties of two-way communication.* CSL 2009.
Proposed a feasible but "intricate" procedure to decide equality with units.

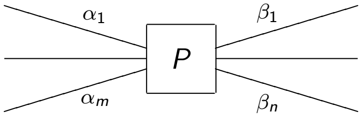
(3) Heijltjes, *Proof nets for additive linear logic with units.* Proc. LICS 2011
(Awarded the LICS 2011 Kleene award for best student paper)
Gave a clean feasible decision procedure for the units.

Without units there is no finite communication!!

4.

Polycategories and representation

Processes connected to many channels



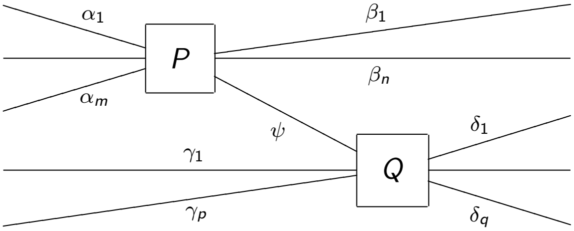
$$\alpha_1 : X_1, \ldots, \alpha_m : X_m \vdash_P \beta_1 : Y_1, \ldots, \beta_n : Y_n$$

$X_1, .., X_m, Y_1, ..., Y_n$ are *protocols* ...
These are types determine which events can happen next on each channel (e.g. given by products and coproduct types).

A process can listen or output to any channel to which it is attached. The process is the *system* and it communicates with its *environment*.

Communication

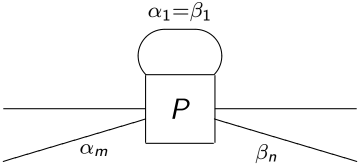
Plugging processes together ...



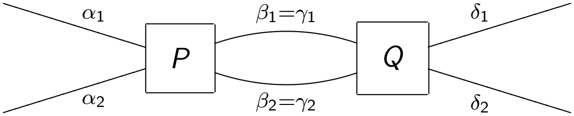
The combined processes become a composite process with the communication on ψ hidden.

Miscommunications ...

Plugging processes together in the wrong way can cause deadlock or livelock ...



Don't plug a process to itself..



Don't connect two processes in two different ways ..

... the correctness criterion for linear logic ...

Polycategories

A polycategory $\mathbf{P}$ consists of the data

- objects: $X_1, \ldots, Y_1, \ldots \in \mathbf{P}_0$
- polymaps: $\forall m, n \in \mathbb{N}$ a set

$$\mathbf{P}(X_1, \ldots, X_m \; ; \; Y_1, \ldots, Y_n)$$

- identities: for each $X \in \mathbf{P}_0$ a polymap $1_X \in \mathbf{P}(X; X)$.
- composition (cut): A map

$$\mathbf{P}(\Gamma; \Delta_1, X, \Delta_2) \times \mathbf{P}(\Gamma_1, X, \Gamma_2; \Delta) \longrightarrow \mathbf{P}(\Gamma_1, \Gamma, \Gamma_2; \Delta_1, \Delta, \Delta_2)$$

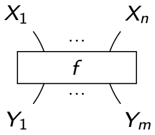
  where Γ_1 or Δ_1 is empty and Γ_2 or Δ_2 is empty.

such that identities *are* identities and cut satisfies associativity and interchange.

A polycategory is symmetric in case $\mathbf{P}(\sigma\Gamma; \tau\Delta) = \mathbf{P}(\Gamma; \Delta)$ for permutations σ and τ, and certain obvious coherence conditions hold.

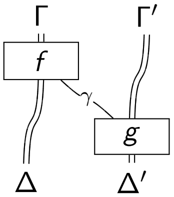
Polycategories

$$X_1, \ldots, X_n \vdash_f Y_1, \ldots, Y_m$$



Composition is modelled by the cut rule

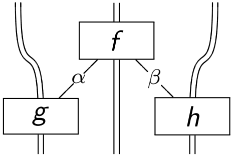
$$\frac{\Gamma \vdash_f \Delta, \gamma : Z \quad \gamma : Z, \Gamma' \vdash_g \Delta}{\Gamma, \Gamma' \vdash_{f;_\gamma g} \Delta, \Delta'}$$

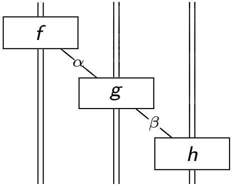


Composition must have identities (these are wires) ..

Polycategories

Composition must satisfy the interchange and associative laws





When polycategories are symmetric crossing wires are allowed.

Pure proof theory of cut-elimination

Symmetric polycategories are the categorical proof theory for cut-elimination.

$$\frac{}{A \vdash_{1_A} A}\ \mathsf{id}$$

$$\frac{\Gamma_1, X_1, X_2, \Gamma_2 \vdash \Gamma}{\Gamma_1, X_2, X_1, \Gamma_2 \vdash \Gamma}\ \text{exchange} \qquad \frac{\Gamma \vdash \Gamma_1, X_1, X_2, \Gamma_2}{\Gamma \vdash \Gamma_1, X_2, X_1, \Gamma_2}\ \text{exchange}$$

$$\frac{\Gamma_1 \vdash \Gamma_2, X \quad X, \Delta_1 \vdash \Delta_2}{\Gamma_1, \Delta_1 \vdash \Gamma_2, \Delta_2}\ \text{cut} \qquad \frac{\Gamma_1 \vdash X, \Gamma_2 \quad \Delta_1, X \vdash \Delta_2}{\Delta_1, \Gamma_1 \vdash \Delta_2, \Gamma_2}\ \text{cut}$$

$$\frac{\Gamma \vdash X \quad \Delta_1, X, \Delta_2 \vdash \Delta}{\Delta_1, \Gamma, \Delta_2 \vdash \Delta}\ \text{cut} \qquad \frac{\Gamma \vdash \Gamma_1, X, \Gamma_2 \quad X \vdash \Delta}{\Gamma \vdash \Gamma_1, \Delta, \Gamma_2}\ \text{cut}$$

Representability

A polycategory is *representable* in case there are *polynatural* equivalences

$$
\begin{array}{rcl}
\mathbf{P}(\Gamma_1, X, Y, \Gamma_2; \Delta) & \xrightarrow[\sim]{r_\otimes} & \mathbf{P}(\Gamma_1, X \otimes Y, \Gamma_2; \Delta) \\
\mathbf{P}(\Gamma_1, \Gamma_2; \Delta) & \xrightarrow[\sim]{r_\top} & \mathbf{P}(\Gamma_1, \top, \Gamma_2; \Delta) \\
\mathbf{P}(\Gamma; \Delta_1, X, Y, \Delta_2) & \xrightarrow[\sim]{r_\oplus} & \mathbf{P}(\Gamma; \Delta_1, X \oplus Y, \Delta_2) \\
\mathbf{P}(\Gamma; \Delta_1, \Delta_2) & \xrightarrow[\sim]{r_\perp} & \mathbf{P}(\Gamma; \Delta_1, \perp, \Delta_2)
\end{array}
$$

Replace the commas with “bundled” types ...

Polynatural means that the transformation is invariant under cutting into the non-active position ...

Representability (Burroni, Hermida) simplifies coherence. In a polycategory having tensors is a *property* not *structure*.

The multiplicatives

Representability can be presented by sequent calculus rules of inference:

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, \top, \Gamma_2 \vdash \Delta}\ \text{split } \top \qquad \frac{\Gamma \vdash \Delta_1, \Delta_2}{\Gamma \vdash \Delta_1, \bot, \Delta_2}\ \text{split } \bot$$

$$\frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta}\ \text{split } \otimes \qquad \frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, A \oplus B, \Delta_2}\ \text{split } \oplus$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \top}\ \text{fork } \top \qquad \frac{\vdash \Delta}{\bot \vdash \Delta}\ \text{fork } \bot$$

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \otimes B, \Delta_2}\ \text{fork } \otimes \qquad \frac{\Gamma_1, A \vdash \Delta_1 \quad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, A \oplus B, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \text{fork } \oplus$$

Linear distribution ...

Here is a derivation of the linear distribution of $\otimes$ over $\oplus$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B} \quad \dfrac{}{C \vdash C}}{B \oplus C \vdash B, C} \qquad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{B \vdash B}}{A, B \vdash A \otimes B}}{A, B \oplus C \vdash A \otimes B, C}}{A \otimes (B \oplus C) \vdash (A \otimes B) \oplus C}$$

Linearly distributive categories

Representable polycategories correspond precisely to linearly distributive categories.

There are natural coherence requirements. A typical coherence requirement is:

$$\begin{tikzcd}
A \otimes (B \otimes (C \oplus D)) \arrow[r, "a_\otimes"'] \arrow[d, "1 \otimes \delta_L"] & (A \otimes B) \otimes (C \oplus D)) \arrow[dd, "\delta_L"] \\
A \otimes ((B \otimes C) \oplus D) \arrow[d, "\delta_L"'] & \\
(A \otimes (B \otimes C)) \oplus D \arrow[r, "a_\otimes \oplus 1"'] & ((A \otimes B) \otimes C) \oplus D
\end{tikzcd}$$

Examples of linearly distributive categories

- A distributive lattice $\wedge = \otimes$, $\vee = \oplus$ (*-autonomous=Boolean lattice).
- A distributive category is a linearly distributive category (with respect to the product and coproduct and the obvious linear distribution) if and only if it is a poset.
- Any monoidal category is a degenerate linear distributive category ("compact": tensor = par).
- Any $*$-autonomous category is a linearly distributive category.
- A compact closed category is a degenerate $*$-autonomous category ("compact": tensor and par).
- (Joyal) Bicompletions of monoidal / linearly distributive categories are linearly distributive (generally not *-autonomous).

Units again .. this time multiplicative!

- If you are French you pretend they don't exist! This is not wise!!
  ... because even if you don't mention them they are implicit.
- If you are Canadian they are the main interest!!
- A decision procedure for map/proof equality *in the symmetric case* is known.
- The decision problem is PSPACE complete (Heijltjes and Houston, manuscript 2014)...

Some history

(1) Barr $*$-*autonomous categories* LMS 752 (1979)
(2) Girard *Linear logic* TCS (1987)
(3) Seely *Linear logic, *-autonomous categories and cofree coalgebras* (1989)
(4) Blute, Cockett, Seely, Trimble *Natural deduction and coherence for linearly distributive categories.* JPAA (1996).
(5) Schneck *Natural deduction and coherence for non-symmetric linearly distributive categories.* TAC (1999)
(6) Koh, Ong *Explicit substitution internal languages for autonomous and $*$-automonous categories.* ENTCS 26 (1999)
(7) Lamarche, Strassburger *Proof nets for multiplicative linear logic with units* LNCS 3210 (2004)
(8) Dominic Hughes *Simple free star-autonomous categories and full coherence* JPPA (2012)

5.

Communication on many channels (Mulipicatives and additives)

Products and coproducts for polycategories

There are polynatural equivalences

$$\mathbf{P}(\Gamma_1, X + Y, \Gamma_2; \Delta) \xrightarrow[\sim]{r_+} \mathbf{P}(\Gamma_1, X, \Gamma_2; \Delta) \times \mathbf{P}(\Gamma_1, Y, \Gamma_2; \Delta)$$

$$\mathbf{P}(\Gamma_1, 0, \Gamma_2; \Delta) \xrightarrow[\sim]{r_0} 1$$

$$\mathbf{P}(\Gamma; \Delta_1, X \times Y, \Delta_2) \xrightarrow[\sim]{r_\times} \mathbf{P}(\Gamma; \Delta_1, X, \Delta_2) \times \mathbf{P}(\Gamma; \Delta_1, Y, \Delta_2)$$

$$\mathbf{P}(\Gamma; \Delta_1, \Delta_2) \xrightarrow[\sim]{r_1} 1$$

When **P** is representable we have distributive laws:

$$X \otimes (A+B) \cong (X \otimes A)+(X \otimes B) \quad \text{and} \quad (A \times B) \oplus Y \cong (A \oplus Y) \times (B \oplus Y).$$

e.g.

$$\cfrac{\cfrac{\cfrac{\mathbf{P}(\Gamma, X \otimes A, \Gamma'; \Delta)}{\mathbf{P}(\Gamma, X, A, \Gamma'; \Delta)} \quad \cfrac{\mathbf{P}(\Gamma, X \otimes B, \Gamma'; \Delta)}{\mathbf{P}(\Gamma, X, B, \Gamma'; \Delta)}}{\mathbf{P}(\Gamma, X, (A+B), \Gamma'; \Delta)}}{\mathbf{P}(\Gamma, X \otimes (A+B), \Gamma'; \Delta)}$$

Notation designed to shock ...

	Additives		Multiplicatives	
	Product	Coproduct	Tensor	Par
Linear Logic (Girard)	$\&$	$\oplus$	$\otimes$	$\parr$
Categorical (Cockett/Seely)	$\times/\prod$	$+/\coprod$	$\otimes$	$\oplus$
Categorical (Egger)	$\wedge$	$\vee$	$\owedge$	$\ovee$

Vive la différence!!!

Girard notation aligned for the distributive law:

$$A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

$*$-autonomous with "exponentials"

Categorical notion aligned along dualities:

$+ \rightleftarrows \times$ and $\oplus \rightleftarrows \otimes$

not $*$-autonomous no exponentials

Poly-calculus of products and coproducts

$$\frac{}{A \vdash_{1_A} A}\ \text{id}$$

$$\frac{\{\Gamma_1, \alpha : X_j, \Gamma_2 \vdash_{P_j} \Gamma_3\}_j}{\Gamma_1, \alpha : \coprod_j X_j, \Gamma_2 \vdash_{\alpha\langle P_j\rangle_j} \Gamma_3}\ \text{cotuple} \quad \frac{\{\Gamma_1 \vdash_{Q_i} \Gamma_2, \alpha : Y_i, \Gamma_3\}_i}{\Gamma_1 \vdash_{\alpha\langle Q_i\rangle_i} \Gamma_2, \alpha : \prod_i Y_i, \Gamma_3}\ \text{tuple}$$

$$\frac{\Gamma_1 \vdash_P \Gamma_2, \alpha : Y_k, \Gamma_3}{\Gamma_1 \vdash_{\alpha[k]\cdot P} \Gamma_2, \alpha : \coprod_i Y_i, \Gamma_3}\ \text{coproj} \quad \frac{\Gamma_1, \alpha : X_k, \Gamma_2 \vdash_Q \Gamma_3}{\Gamma_1, \alpha : \prod_i X_i, \Gamma_2 \vdash_{\alpha[k]\cdot Q} \Gamma_3}\ \text{proj}$$

$$\frac{\Gamma_1 \vdash_P \Gamma_2, \alpha : X, \Gamma_3 \quad \Delta_1, \alpha : X, \Delta_2 \vdash_Q \Delta_3}{\Delta_1, \Gamma_1, \Delta_2 \vdash_{P;_\alpha Q} \Gamma_2, \Delta_3, \Gamma_3}\ \text{cut}$$

Cut elimination ...

... is rewriting modulo equations (but now you have to worry about which channel!!!):

$\alpha \neq \beta$

$$\begin{array}{rcl}
(\alpha[k] \cdot P) \;;_\gamma\; Q & \Longrightarrow & \alpha[k] \cdot (P \;;_\gamma\; Q) \\
P \;;_\gamma\; (\beta[k] \cdot Q) & \Longrightarrow & \beta[k] \cdot (P \;;_\gamma\; Q) \\
\alpha\langle P_i \rangle_i \;;_\gamma\; Q & \Longrightarrow & \alpha\langle P_i \;;_\gamma\; Q \rangle_i \\
P \;;_\gamma\; \beta\langle Q_j \rangle_j & \Longrightarrow & \beta\langle P \;;_\gamma\; Q_j \rangle_j \\
\gamma[k] \cdot P \;;_\gamma\; \gamma\langle Q_j \rangle_j & \Longrightarrow & P \;;_\gamma\; Q_k \\
\gamma\langle P_i \rangle_i \;;_\gamma\; \gamma[k] \cdot Q & \Longrightarrow & P_k \;;_\gamma\; Q \\
\alpha\langle \beta\langle P_{ij} \rangle_j \rangle_i & \longmapsfrom\!\!\longmapsto & \beta\langle \alpha\langle P_{ij} \rangle_i \rangle_j \\
\alpha[k] \cdot \beta\langle P_j \rangle_j & \longmapsfrom\!\!\longmapsto & \beta\langle \alpha[k] \cdot P_j \rangle_j \\
\alpha[k] \cdot \beta[l] \cdot P & \longmapsfrom\!\!\longmapsto & \beta[l] \cdot \alpha[k] \cdot P
\end{array}$$

Forking and splitting

$$\frac{\Gamma_1, \alpha_1 : X, \alpha_2 : Y, \Gamma_2 \vdash_P \Gamma_3}{\Gamma_1, \alpha : X \otimes Y, \Gamma_2 \vdash_{\alpha\langle \alpha_1, \alpha_2 \mapsto P\rangle} \Gamma_3}$$

$$\frac{\gamma_1 : \Gamma_1 \vdash_P \gamma_2 : \Gamma_2, \alpha_1 : X \quad \delta_1 : \Delta_1 \vdash_Q \alpha_2 : Y, \delta_2 : \Delta_2}{\gamma_1 : \Gamma_1, \delta_1 : \Delta_1 \vdash_{\alpha\left\langle \begin{array}{l} \alpha_1 \mid \gamma_1, \gamma_2 \mapsto P \\ \alpha_2 \mid \delta_1, \delta_2 \mapsto Q \end{array} \right\rangle} \gamma_2 : \Gamma_2, \alpha : X \otimes Y, \delta_2 : \Delta_2}$$

Program syntax:

$$\alpha\langle \alpha_1, \alpha_2 \mapsto P\rangle \equiv \texttt{split } \alpha \texttt{ into } (\alpha_1, \alpha_2) \texttt{ in } P$$

$$\alpha\left\langle \begin{array}{l} \alpha_1 \mid \gamma_1, \gamma_2 \mapsto P \\ \alpha_2 \mid \delta_1, \delta_2 \mapsto Q \end{array} \right\rangle \equiv \texttt{fork } \alpha \texttt{ as } \begin{array}{l} \alpha_1 \texttt{ with } \gamma_1, \gamma_2 \mapsto P \\ \alpha_2 \texttt{ with } \delta_1, \delta_2 \mapsto Q \end{array}$$

More rewrites and identities

- $\gamma\left\langle \begin{array}{l} \alpha \mid \Lambda \mapsto f \\ \beta \mid \Phi \mapsto g \end{array}\right\rangle ;_{\gamma}\ \gamma\langle(\alpha,\beta)\mapsto h\rangle \Longrightarrow g;_{\beta}(f;_{\alpha} h))$
- $\alpha\left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto \beta\left(\begin{array}{l} a_1 \mapsto g_1 \\ a_2 \mapsto g_2 \end{array}\right) \end{array}\right\rangle \mathrel{\vert\!=\!=\!\vert} \beta\left(\begin{array}{l} a_1 \mapsto \alpha\left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto g_1 \end{array}\right\rangle \\ a_2 \mapsto \alpha\left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto g_2 \end{array}\right\rangle \end{array}\right)$

There are more identities. See:
Cockett and Pastro, *A language for multiplicative-additive linear logic*, ENTCS, 2005.

Some history

The multiplicative-additive fragment has map equality decidable.
The complexity of deciding equality is PSPACE complete.

(1) The problem was first explored in:
Girard *Proof-nets for additives*, manuscript, 1994.

(2) The "unit-free" case was handled very neatly by:
Hughes and Glabbeek, *Proof nets for unit-free multiplicative-additive linear logic*, LICS 2003.

(3) A solution for the case with all units was described in:
Cockett and Pastro, *A language for multiplicative-additive linear logic*, ENTCS, 2005.
However, the procedure was exponential and no attempt to analyze the complexity was made.

Where are we?

REMARKABLY:

almost [1]

NO CHOICES HAVE BEEN MADE!!

... everything is free and canonical ...

The initial setting for concurrency is just MALL ...

[1]... we chose symmetry for the polycategory!

6.

Message passing

Key feature of concurrency!

Message passing

A two-tier logic:

- The logic of messages: the sequential world of computation (e.g. Cartesian closed category (CCC) with datatypes)
- The logic of message passing: the concurrent world of computation
  (e.g. Linearly distributive category (LDC) with protocols)

Categorically the sequential world **acts** on the concurrent world

... that is it is a **linear actegory** ...

Linear actegory

- A cartesian $\mathbf{C}$ closed category (minimally with coproducts) to represent the sequential world
- A linearly distributive category $\mathbf{L}$ (minimally with products and coproducts) to represent the concurrent world
- Two actions:
  $\_ \circ \_ : \mathbf{C} \times \mathbf{L} \longrightarrow \mathbf{L}$ and $\_ \bullet \_ : \mathbf{C}^{\mathrm{op}} \times \mathbf{L} \longrightarrow \mathbf{L}$ so that
  $(A \times B) \circ L \cong A \circ (B \circ L)$ and $(A \times B) \bullet L \cong A \bullet (B \bullet L)$
  with obvious coherences.
- $A \bullet \_$ (putting out a message on the left) is left adjoint to $A \circ \_$ (putting out a message on the right):

$$\frac{X \longrightarrow A \circ Y}{\overline{A \bullet X \longrightarrow Y}}$$

Sequential and concurrent worlds

The proof theory:

(A) Sequential sequent:

$$\Psi \longrightarrow A$$

where Ψ is a sequential context - a list of types.

(B) Concurrent sequent:

$$\Psi \mid \Gamma \vdash \Delta$$

- Ψ is the sequential context (a sequence of types)
- Γ and Δ are the concurrent contexts (a sequence of protocols)

Message passing rules

$$\frac{x{:}A, \Psi \mid \alpha{::}X, \Gamma \vdash \Delta}{\Psi \mid \alpha{::}A \circ X, \Gamma \vdash \Delta} \quad \text{get } x \text{ on } \alpha \quad \frac{x{:}A, \Psi \mid \Gamma \vdash \alpha{::}Y, \Delta}{\Psi \mid \Gamma \vdash \alpha{::}A \bullet Y, \Delta}$$

$$\frac{\Psi \longrightarrow t{:}A \quad \Psi \mid \alpha{::}X, \Gamma \vdash \Delta}{\Psi \mid \alpha{::}A \bullet X, \Gamma \vdash \Delta} \quad \text{put } t \text{ on } \alpha \quad \frac{\Psi \longrightarrow t{:}A \quad \Psi \mid \Gamma \vdash Y, \Delta}{\Psi \mid \Gamma \vdash \alpha{::}A \circ Y, \Delta}$$

Programs do not distinguish "put" / "get" on the left or right ...
The adjunction guarantees they are equivalent ...

BUT THE TYPES ARE DIFFERENT!!!
The type depends on whether a channel has an "input polarity" (channel on left) or an "output polarity" (channel on right).

Why polarities?

A communication channel attached to a process can be viewed two ways:

A. From the processes perspective ...

B. From the external world (or environments) perspective ...

These are dual. A process must specify how it views its channels:

$$\begin{aligned} \text{Output Polarity} &= \text{Process Perspective} \\ \text{Input Polarity} &= \text{Environment perspective} \end{aligned}$$

Message passing rules

Augmenting multiplicative rules with sequential contexts:

$$\frac{\Psi \mid \Gamma_1, \Gamma_2 \vdash \Delta}{\Psi \mid \Gamma_1, \top, \Gamma_2 \vdash \Delta}\ \text{split } \top \qquad \frac{\Psi \mid \Gamma \vdash \Delta_1, \Delta_2}{\Psi \mid \Gamma \vdash \Delta_1, \bot, \Delta_2}\ \text{split } \bot$$

$$\frac{\Psi \mid \Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Psi \mid \Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta}\ \text{split } \otimes \qquad \frac{\Psi \mid \Gamma \vdash \Delta_1, A, B, \Delta_2}{\Psi \mid \Gamma \vdash \Delta_1, A \oplus B, \Delta_2}\ \text{split } \oplus$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \top}\ \text{fork } \top \qquad \frac{\vdash \Delta}{\bot \vdash \Delta}\ \text{fork } \bot$$

$$\frac{\Psi \mid \Gamma_1 \vdash \Delta_1, A \quad \Psi \mid \Gamma_2 \vdash B, \Delta_2}{\Psi \mid \Gamma_1, \Gamma_2 \vdash \Delta_1, A \otimes B, \Delta_2}\ \text{fork } \otimes$$

$$\frac{\Psi \mid \Gamma_1, A \vdash \Delta_1 \quad \Psi \mid \Gamma_2, B \vdash \Delta_2}{\Psi \mid \Gamma_1, A \oplus B, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \text{fork } \oplus$$

Message passing rules

Augmenting with sequential context is straightforward ... mostly!

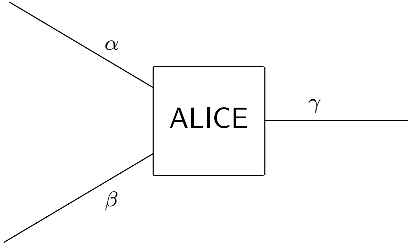
Significantly the sequential coproduct interacts with the concurrent world allowing sequential control to have concurrent effect:

$$\frac{\Psi, A \mid \Gamma \vdash \Delta \quad \Psi, A \mid \Gamma \vdash \Delta}{\Psi, A + B \mid \Gamma \vdash \Delta}$$

Program construct:

```
case $t$  of
          $b_0(x_0) \mapsto P_1$
          $b_1(x_1) \mapsto P_2$
```

Diffie-Hellman key exchange:



Given two public keys k_1 and k_2, Alice gets a secret key, *a*, on secure channel α. Does a key exchange on insecure channel γ talking to Bob! Gets a plain text message on (secure) channel β, encrypts it with now agreed key; sends this on the insecure channel γ to Bob ....

Diffie-Hellman key exchange:

```
$\text{Alice} :: (\text{Key}, \text{Key}) \;\; \text{Key} \bullet \top, \text{Msg} \bullet \top \Rightarrow \text{Key} \bullet (\text{Key} \circ (\text{cMsg} \bullet \top))$
$\text{Alice}\ (k_1, k_2) \;\; \alpha, \beta \Rightarrow \gamma$
        get $a$ on $\alpha$
        put $\text{mod}_{k_1}(k_2^a)$ on $\gamma$
        get $b$ on $\gamma$
        get $m$ on $\beta$
        put $\text{encode}(m, \text{mod}_{k_1}(b^a))$ on $\gamma$
        close $\alpha$
        close $\beta$
        end $\gamma$
```

A Reference:

Cockett and Pastro
The logic of message passing
Science of computer programming 74 (2009) 498-533

$$\begin{tikzcd} \frac{\text{Proof theory}}{\text{poly actegory}} \arrow[rr, leftrightarrow] \arrow[dr, leftrightarrow] & & \frac{\text{Category theory}}{\text{linear actegory}} \arrow[dl, leftrightarrow] \\ & \frac{\text{term calculus}}{\text{message passing}} & \end{tikzcd}$$

(Lots of history for message passing!!)

6.

Protocols

... for interactions continuing through time ...

Protocols

So far we have only introduced *basic* protocols using:

(a) products

(b) coproducts

(c) message passing: binding messages to channels.

These allow the modelling of *finite* interactions.

For *REAL* programming need sophisticated protocols which are possibly infinite in time ...

These can be delivered through fixed points.

Protocols and fixed points

Initial/final datatypes in the concurrent world = (categorical) fixed points in the concurrent world

=

protocols!

There are two formulations:

- Lambek style datatypes: the fixed point formulation of inductive and coinductive datatypes.
- Mendler style datatypes (Vene, Uustalu) and circular style datatypes (Santocanale).

... for polycategories (without negation) only the second formulation works!

Circular rules for polycategories

Given functors P and Q here are the circular rules for a polycategory:

$$\frac{\Gamma \vdash \Delta, P(\mu x.P(x)), \Delta'}{\Gamma \vdash \Delta, \mu x.P(x), \Delta'}\ \mu\text{-Cons} \qquad \frac{\Gamma, Q(\nu x.Q(x)), \Gamma' \vdash_A \Delta}{\Gamma, \nu x.Q(x), \Gamma' \vdash \Delta}\ \nu\text{-Cons}$$

$$\begin{array}{c}
\boxed{\begin{array}{c} X = \mu x.P(x) \ \mid \ \Gamma, X, \Gamma' \vdash_X \Delta \\ \hline \\ \vdots \\ \hline \Gamma, P(X), \Gamma' \vdash \Delta \end{array}} \\ \Gamma, \mu x.P(x), \Gamma' \vdash \Delta
\end{array}
\qquad
\begin{array}{c}
\boxed{\begin{array}{c} X = \nu x.Q(x) \ \mid \ \Gamma \vdash_X \Delta, X, \Delta' \\ \hline \\ \vdots \\ \hline \Gamma \vdash \Delta, Q(X), \Delta' \end{array}} \\ \Gamma \vdash \Delta, \nu x.Q(x), \Delta'
\end{array}$$

Expressiveness of protocols

Adding datatypes increases expressiveness dramatically!
One can defne the Burroni natural numbers by:

$$\mathbb{N}(A) = \mu X.A + X$$

Having the Burroni natural numbers means:

- All primitive recursive functions on the natural numbers are present (Pare and Roman)!
- SO the decision problem for equality of maps is immediately undecideable.

Programming with protocols

```
protocol Talk(A,B) ⇒ $C
    #talk:: put(A) (get(B) $C) ⇒ $C
                –(initial fixed point $\mu x.A \bullet (B \circ x)$)

drive
  duplicator::Talk(A,B*B) ⇒ Talk(A,B), Talk(A,B)
  duplicator:: $\alpha \Rightarrow \beta, \gamma$ by $\alpha$ =
    #talk:
        get $x$ on $\alpha$
        put #talk on $\beta$ ; put #talk on $\gamma$
        put $x$ on $\beta$ ; put $x$ on $\gamma$
        get $y_1$ on $\beta$ ; get $y_2$ on $\gamma$
        put $(y_1, y_2)$ on $\alpha$
        call duplicator( $\alpha \Rightarrow \beta, \gamma$)
```

Memory cell ... mutable memory

```
protocol Memory (a) ⇒ $C =
     #rcv :: put a $C ⇒ $C
     #snd :: get a $C ⇒ $C

drive
 Memorycell :: (a) Memory (a) ⇒
 Memorycell (n) ch ⇒ by ch =
     #rcv: get a on ch. call Memorycell (a) ch ⇒
     #snd: put  n  on  ch; call  Memorycell (n) ch ⇒
```

Reconfiguring ... memory locking

```
protocol Talker (A) ⇒ $C =
      #talk:: put A get A  $C ⇒ $C
protocol Passer $M ⇒ $C =
      #pass :: ($M (+) (Neg($M) (x) $C)) ⇒ $C
drive
 P1 :: () Passer (Memory (A)), Talker (A) ⇒ Memory (A)
 P1 c1,in2 ⇒ c2 by c1=
     #pass: match in2 as
           #talk : get x on in2.
           #snd on c2; get y on c2. #rcv on c2
             put x on c2; put y on in2
             fork c1 as
                   x1 with c2 . x1==c2
                   x2 with in2 . split x2 into (nm, x3). plug on x1
                        P1 x3, in2 ⇒ x1 to Neg x1 == nm
```

Reconfiguring ... memory locking

```
drive
 P2 :: () Talker (A) ⇒ Passer (Memory (A))
 P2  in1 ⇒ c by in1=
     #talk : get x on in1.
             #pass on c;
             split c into (m, x4).
                 #snd on m
                 get y on m.  put y on in1
                 #rcv on m; put x on m
                 fork x4 as
                    nm with m. (Neg(m)==nm)
                    x5 with in1. call P2 in1⇒ x5
```

Conclusions ...

Was this carving in snow!!???

- Logic of products and coproducts precisely describes communication on a channel.
- Polycategories (the logic of cut) and additives model communication on many channels.
- Multiplicatives given by representability.
- Messages determined by (adjoint action).
- Protocols given by datatypes.
- *There was no choice!!!!*

Conclusions ..

Linearly actegories provide a semantics of concurrency:

- communication on channels
- message passing
- fixed points give sophisticated protocols.

AND ... non-deterministic semantics for distributed computing obtained by compacting features (e.g tensor = par)?

Conclusions ...

Mathematics *has* something to say on the semantics of concurrency ...

... the mathematics involved is (largely) available ...

... the problem is to deploy it in Computer Science!!!