

Linear types as a semantics for concurrency: passing messages and defining protocols.

J.R.B. Cockett

Department of Computer Science
University of Calgary
Alberta, Canada

robin@cpsc.ucalgary.ca

(work with Subashis Chakraborty)

Chocola Lyon: 14 March 2013

What is a good semantics for concurrency?

1. Where are we? Where should we be?
2. Communication on a channel.
3. Polycategories and representability.
4. Communication on many channels.
5. Message passing.
6. Protocols.

1.

Where are we?

Where should we be?

Distributed computing: the reality ...

In the 1970's networks, parallel, and distributed computing were 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 the reality ...

- ▶ 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 power!

There is no turning back ...

Practice ahead of theory ...

Where is the theory?

- ▶ Where was the mathematics of processes, concurrency, communication?
- ▶ Was the theory only develop in response to practice?
- ▶ Was theory simply modeling practice? Should it?
- ▶ Was there a need to develop new theory ...
.... or was it just taking time to link existing theory and practice?

Does mathematics have anything insightful to say about communicating processes?

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

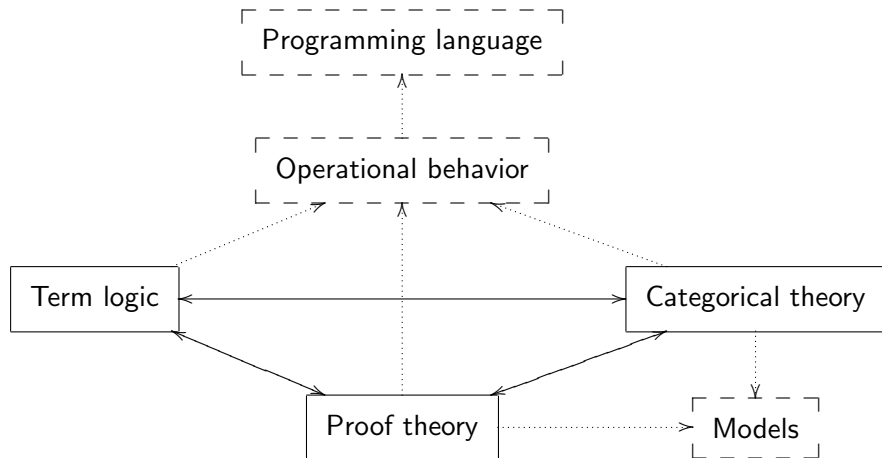
- ▶ No Church/Turing thesis for concurrency ...
- ▶ A tool kit: no unified theory ...
- ▶ Plasticity of definition, carvings in snow: no bedrock ...
- ▶ A profusion of syntax but no semantics ...
- ▶ Physics (quantum computing), biological computing, and environmental modeling are at our gates: what 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: (localized) program transformations.
 - ▶ Compositional semantics: allowing program construction.
 - ▶ Modular description: allows controlled “feature” addition.
 - ▶ Interface to mathematics: models with different properties.
 - ▶ Term logic = programming language (Subashis).
 - ▶ Operational semantics = abstract machines, efficient evaluation.

So where are we?

The mathematics is largely there!!

... BUT the Computer Science is not!
... this is where the rubber hits the road!

2.

Communication on a channel

(A first bit of bedrock!)

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!

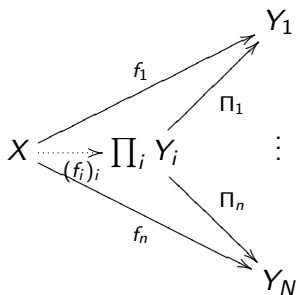
Products and coproducts

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.*

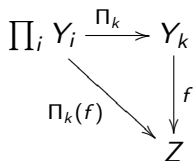
Cockett and Seely: *The logic of sums and products* $\Sigma\Pi$

Products



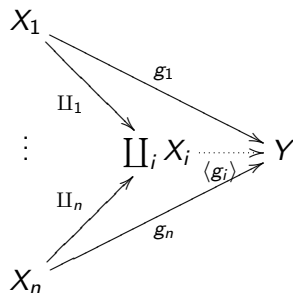
$$\begin{aligned} \Pi_k(f); g &= \Pi_k(f; g) \\ f; (g_i)_{i \in I} &= (f; g_i)_{i \in I} \\ (f_i)_{i \in I}; \Pi_k(g) &= f_k; g \end{aligned}$$

where



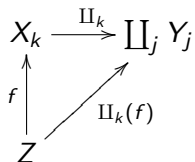
$$\Pi_k; f = \Pi_k(f) \text{ and } \Pi_k = \Pi_k(1_{Y_k})$$

Coproducts



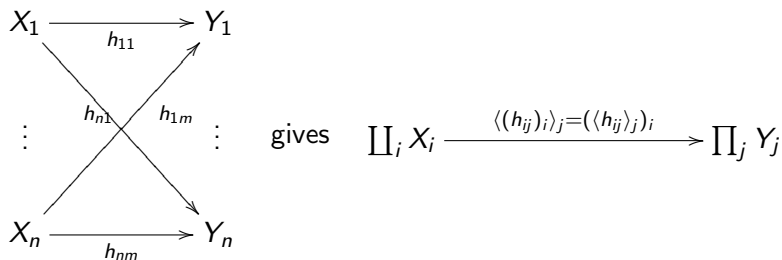
$$\begin{aligned}
 f; \Pi_k(g) &= \Pi_k(f; g) \\
 \langle f_j \rangle_{j \in J}; g &= \langle f_j; g \rangle_{j \in J} \\
 \Pi_k(f); \langle g_j \rangle_{j \in J} &= f; g_k
 \end{aligned}$$

where



$$f; \Pi_k = \Pi_k(f) \text{ and } \Pi_k = \Pi_k(1_{X_k})$$

Interactions



and the basic equalities:

$$\prod_i (\prod_j (f)) = \prod_j (\prod_i (f)) \quad \prod_k (\langle (g_i)_i \rangle) = (\prod_k (g_i))_i$$

$$\prod_k (\langle (f_j)_j \rangle) = \langle (\prod_k (f_j)) \rangle_j$$

Logic of products and coproducts

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

$$\frac{\{X_j \vdash_{f_j} Y\}_{j \in J}}{\prod_j X_j \vdash_{\langle f_j \rangle_{j \in J}} Y} \text{ cotuple}$$

$$\frac{\{X \vdash_{g_i} Y_i\}_{i \in I}}{X \vdash_{(g_i)_{i \in I}} \prod_i Y_i} \text{ tuple}$$

$$\frac{X \vdash_f Y_k}{X \vdash_{\Pi_k(f)} \prod_{i \in I} Y_i} \text{ coproj}$$

$$\frac{X_k \vdash_f Y}{\prod_{i \in I} X_i \vdash_{\Pi_k(f)} Y} \text{ proj}$$

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

Cut elimination

... is rewriting modulo equations:

$$f; 1 \Longrightarrow f$$

$$1; f \Longrightarrow f$$

$$f; \Pi_k(g) \Longrightarrow \Pi_k(f; g)$$

$$\Pi_k(f); g \Longrightarrow \Pi_k(f; g)$$

$$\langle f_i \rangle_i; g \Longrightarrow \langle f_i; g \rangle_i$$

$$f; (g_i)_i \Longrightarrow (f; g_i)_i$$

$$\Pi_k(f); \langle g_i \rangle_i \Longrightarrow f; g_k$$

$$(f_i)_i; \Pi_k(g) \Longrightarrow f_k; g$$

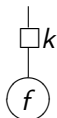
$$\Pi_k(\langle f_j \rangle_j) \longmapsto \langle \Pi_k(f_j) \rangle_j$$

$$\Pi_k(\langle f_i \rangle_i) \longmapsto \langle \Pi_k(f_i) \rangle_i$$

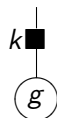
$$\Pi_i(\Pi_j(f)) \longmapsto \Pi_j(\Pi_i(f))$$

$$\langle \langle f_{ij} \rangle_i \rangle_j \longmapsto \langle \langle f_{ij} \rangle_j \rangle_i$$

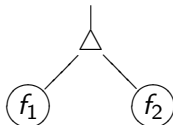
Process reading ...



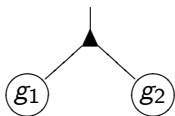
Output “k” on the right = $\Pi_k(f)$



Output “k” on the left = $\Pi_k(g)$



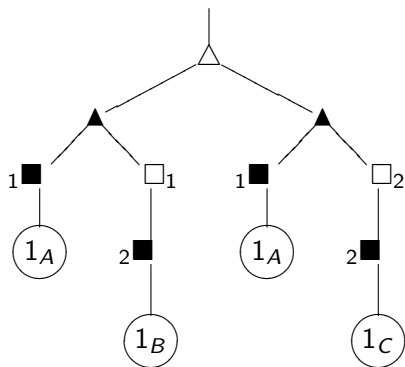
Listen for input on the left = $\langle f_1, f_2 \rangle$



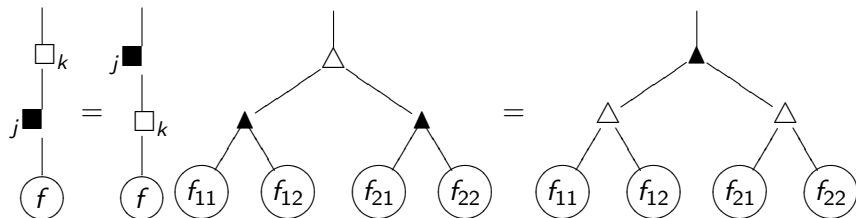
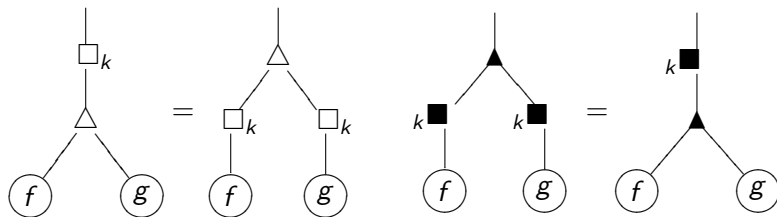
Listen for input on the right = (g_1, g_2)

Process reading of a map ...

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



Process reading of the identities ...



Slogan

The proof theory of products and coproducts

IS

the basic calculus of communication on a channel.

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:

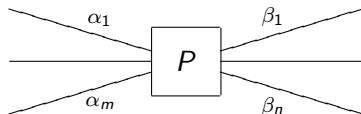
- (1) Cockett and Seely, *Finite sum-product logic*, TAC 8, (2001)
Not everything was sorted out!! No decision procedure for units ...
- (2) Cockett and Santocanale, *On the word problem for $\Sigma\Pi$ -categories, and the properties of two-way communication*. CSL 2009.
Proposed a feasible but “intricate” 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!!

3.

Polycategories and representation

Processes connected to many channels



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

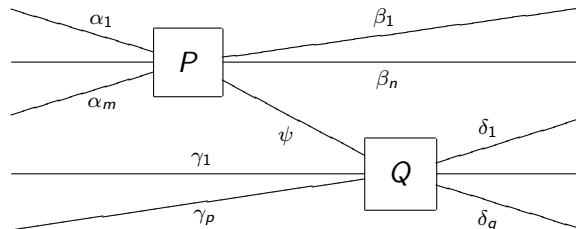
$X_1, \dots, X_m, Y_1, \dots, Y_n$ are *protocols* ...

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

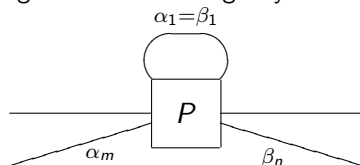
Plugging processes together ...



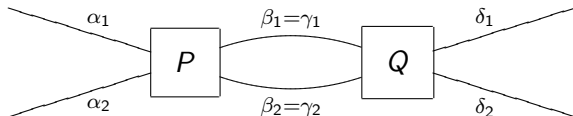
The combined processes become a composite process with the communication on ψ hidden.

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

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

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

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

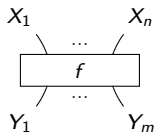
where Γ_1 or Δ_1 is empty and Γ_2 or Δ_2 is empty.

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.

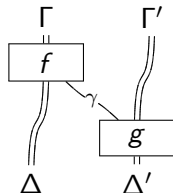
Polycategories

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



Composition is modeled by the cut rule

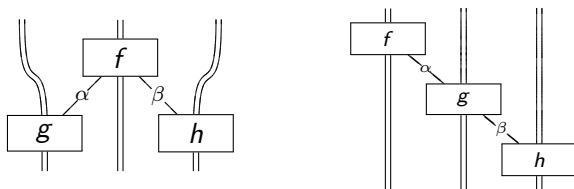
$$\frac{\Gamma \vdash_f \Delta, \gamma : Z \quad \gamma : Z, \Gamma' \vdash_g \Delta'}{\Gamma, \Gamma' \vdash_{f; \gamma g} \Delta, \Delta'}$$



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.

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

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

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

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

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

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

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

Representability

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

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

Replace the commas with “bundled” types ...

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

Representability (Burrioni, 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:

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, \top, \Gamma_2 \vdash \Delta} \text{ split } \top$$

$$\frac{\Gamma \vdash \Delta_1, \Delta_2}{\Gamma \vdash \Delta_1, \perp, \Delta_2} \text{ split } \perp$$

$$\frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta} \text{ split } \otimes$$

$$\frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, A \oplus B, \Delta_2} \text{ split } \oplus$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \top} \text{ fork } \top$$

$$\frac{\vdash \Delta}{\perp \vdash \Delta} \text{ fork } \perp$$

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

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

Linear distribution ...

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

$$\frac{\frac{\overline{B \vdash B} \quad \overline{C \vdash C}}{B \oplus C \vdash B, C} \quad \frac{\overline{A \vdash A} \quad \overline{B \vdash B}}{A, B \vdash A \otimes B}}{A, B \oplus C \vdash A \otimes B, C}}{A \otimes (B \oplus C) \vdash (A \otimes B) \oplus C}$$

Linearly distributive categories

Representable polycategories correspond precisely to linearly distributive categories.

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

$$\begin{array}{ccc}
 A \otimes (B \otimes (C \oplus D)) & \xrightarrow{a_{\otimes}} & (A \otimes B) \otimes (C \oplus D) \\
 \downarrow 1 \otimes \delta_L & & \downarrow \delta_L \\
 A \otimes ((B \otimes C) \oplus D) & & \\
 \downarrow \delta_L & & \\
 (A \otimes (B \otimes C)) \oplus D & \xrightarrow{a_{\otimes} \oplus 1} & ((A \otimes B) \otimes C) \oplus D
 \end{array}$$

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 (exponential).
- ▶ The complexity of this decision problem is *still* unknown ...
- ▶ A decision procedure for equality of maps in *special cases* of the non-symmetric case is known.

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 *-autonomous 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)

4.

Communication on many channels (Multiplicatives 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 \mathbf{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.

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

Notation designed to shock ...

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

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:

$$+ \rightleftarrows \times \text{ and } \oplus \rightleftarrows \otimes$$

not *-autonomous no exponentials

Poly-calculus of products and coproducts

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

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

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

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

Cut elimination ...

... is rewriting modulo equations:

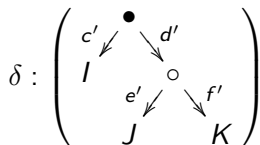
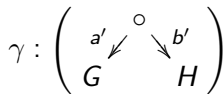
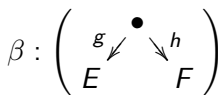
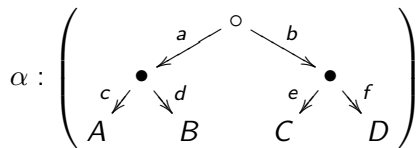
$\alpha \neq \beta$

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

A multi-channel process

$$P : A, E \longrightarrow G, I \qquad Q : B, E \longrightarrow G, J$$

$$R : C, F \longrightarrow H, I \qquad S : D, F \longrightarrow H, K$$



$$\alpha \left\langle \begin{array}{l} a \mapsto \beta[g](\gamma[a'](\delta \left(\begin{array}{l} c' \mapsto \alpha[c](P) \\ d' \mapsto \alpha[d](\delta[e'](Q)) \end{array} \right))) \\ b \mapsto \beta[h](\gamma[b'](\delta \left(\begin{array}{l} c' \mapsto \alpha[e](R) \\ d' \mapsto \alpha[f](\delta[f'](S)) \end{array} \right))) \end{array} \right\rangle$$

A programming interlude

$$\alpha[a] := \text{put } a \text{ on } \alpha$$

$$\delta \left(\begin{array}{l} c \mapsto R \\ d \mapsto S \end{array} \right) := \begin{array}{l} \text{match } \delta \text{ as} \\ c \rightarrow R \\ d \rightarrow S \end{array}$$

$$\delta \left\langle \begin{array}{l} c \mapsto R \\ d \mapsto S \end{array} \right\rangle := \begin{array}{l} \text{match } \delta \text{ as} \\ c \rightarrow R \\ d \rightarrow S \end{array}$$

Note: in programming syntax distinction between products and coproducts are suppressed ...

A first programming interlude

match α as

$a \rightarrow$ put g on β ; put a' on γ

 match δ as

$c' \rightarrow$ put c on α

 call $P(\alpha, \beta \Rightarrow \gamma, \delta)$

$d' \rightarrow$ put d on α ; put e' on δ

 call $Q(\alpha, \beta \Rightarrow \gamma, \delta)$

$b \rightarrow$ put h on β ; put b' on γ

 match δ as

$c' \rightarrow$ put e on α

 call $R(\alpha, \beta \Rightarrow \gamma, \delta)$

$d' \rightarrow$ put f on α ; put f' on δ

 call $S(\alpha, \beta \Rightarrow \gamma, \delta)$

An example of cut-elimination

Suppose $f : A \rightarrow D$, $g : B \rightarrow C$, and $h : D \rightarrow E$ are atomic axioms.

$$\begin{array}{c} \alpha \\ \circ \\ \swarrow \quad \searrow \\ a \quad b \\ \downarrow \quad \downarrow \\ A \quad B \end{array} \xrightarrow{\alpha \left\langle \begin{array}{l} a \mapsto \gamma[c](f) \\ b \mapsto \gamma[d](g) \end{array} \right\rangle} \begin{array}{c} \gamma \\ \circ \\ \swarrow \quad \searrow \\ c \quad d \\ \downarrow \quad \downarrow \\ C \quad D \end{array} \xrightarrow{\gamma \left(\begin{array}{l} c \mapsto \beta[f](1_C) \\ d \mapsto \beta[e](h) \end{array} \right)} \begin{array}{c} \beta \\ \circ \\ \swarrow \quad \searrow \\ e \quad f \\ \downarrow \quad \downarrow \\ E \quad C \end{array}$$

An example cont.

$$\begin{aligned}
 & \alpha \left\langle \begin{array}{l} a \mapsto \gamma[c](f) \\ b \mapsto \gamma[d](g) \end{array} \right\rangle ; \gamma \quad \gamma \left(\begin{array}{l} c \mapsto \beta[f](h) \\ d \mapsto \beta[e](1_C) \end{array} \right) \\
 & \Rightarrow \alpha \left\langle \begin{array}{l} a \mapsto \gamma[c](f) \quad ; \gamma \quad \gamma \left(\begin{array}{l} c \mapsto \beta[f](h) \\ d \mapsto \beta[e](1_C) \end{array} \right) \\ b \mapsto \gamma[d](g) \quad ; \gamma \quad \gamma \left(\begin{array}{l} c \mapsto \beta[f](h) \\ d \mapsto \beta[e](1_C) \end{array} \right) \end{array} \right\rangle \\
 & \Rightarrow \alpha \left\langle \begin{array}{l} a \mapsto f ; \gamma \beta[f](h) \\ b \mapsto g ; \gamma \beta[e](1_C) \end{array} \right\rangle \Rightarrow \alpha \left\langle \begin{array}{l} a \mapsto \beta[f](f ; \gamma h) \\ b \mapsto \beta[e](g ; \gamma 1_C) \end{array} \right\rangle \Rightarrow \alpha \left\langle \begin{array}{l} a \mapsto \beta[f](f ; \gamma h) \\ b \mapsto \beta[e](g) \end{array} \right\rangle
 \end{aligned}$$



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:

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

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

More rewrites and identities

$$\blacktriangleright \gamma \left\langle \begin{array}{l} \alpha \mid \Lambda \mapsto f \\ \beta \mid \Phi \mapsto g \end{array} \right\rangle ;_{\gamma} \gamma \langle (\alpha, \beta) \mapsto h \rangle \Longrightarrow g ;_{\beta} (f ;_{\alpha} h)$$

$$\blacktriangleright \alpha \left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto \beta \left(\begin{array}{l} a_1 \mapsto g_1 \\ a_2 \mapsto g_2 \end{array} \right) \end{array} \right\rangle \equiv \beta \left(\begin{array}{l} a_1 \mapsto \alpha \left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto g_1 \end{array} \right\rangle \\ a_2 \mapsto \alpha \left\langle \begin{array}{l} \alpha_1 \mid \Lambda_1 \mapsto f \\ \alpha_2 \mid \Lambda_2 \mapsto g_2 \end{array} \right\rangle \end{array} \right)$$

There are more identities. See:

Cockett and Pasto, *A language for multiplicative-additive linear logic*, ENTCS, 2005.

Some history

The multiplicative-additive fragment has map equality decidable. However, the precise complexity of deciding equality is still unknown (it is in PSPACE).

- (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 analyse 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!

5.

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 coproducts)
- ▶ The logic of message passing: the concurrent world of computation (e.g. Linearly distributive category (LDC) with products/coproducts)

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

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

Linear actegory

- ▶ A cartesian \mathbf{C} closed category with coproducts to represent the sequential world
- ▶ A linearly distributive category \mathbf{L} with products and coproducts to represent the concurrent world
- ▶ Two actions:
 $_ \circ _ : \mathbf{C} \times \mathbf{L} \rightarrow \mathbf{L}$ and $_ \bullet _ : \mathbf{C}^{\text{op}} \times \mathbf{L} \rightarrow \mathbf{L}$ 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):

$$\frac{X \rightarrow A \circ Y}{A \bullet X \rightarrow Y}$$

Sequential and concurrent worlds

(A) Sequential sequent:

$$\Psi \rightarrow A$$

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

$$\frac{x:A, \Psi \mid \alpha::X, \Gamma \vdash \Delta}{\Psi \mid \alpha::A \circ X, \Gamma \vdash \Delta}$$

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

$$\frac{\Psi \rightarrow t:A \quad \Psi \mid \alpha::X, \Gamma \vdash \Delta}{\Psi \mid \alpha::A \bullet X, \Gamma \vdash \Delta}$$

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

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).
The programmer has to assign “polarities” to facilitate type inference ...

Message passing rules

Augmenting multiplicative rules with sequential contexts:

$$\frac{\Psi \mid \Gamma_1, \Gamma_2 \vdash \Delta}{\Psi \mid \Gamma_1, \top, \Gamma_2 \vdash \Delta} \text{split } \top$$

$$\frac{\Psi \mid \Gamma \vdash \Delta_1, \Delta_2}{\Psi \mid \Gamma \vdash \Delta_1, \perp, \Delta_2} \text{split } \perp$$

$$\frac{\Psi \mid \Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Psi \mid \Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta} \text{split } \otimes$$

$$\frac{\Psi \mid \Gamma \vdash \Delta_1, A, B, \Delta_2}{\Psi \mid \Gamma \vdash \Delta_1, A \oplus B, \Delta_2} \text{split } \oplus$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \top} \text{fork } \top$$

$$\frac{\vdash \Delta}{\perp \vdash \Delta} \text{fork } \perp$$

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

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

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
  b0(x0) ↦ P1
  b1(x1) ↦ P2

```

Diffie-Hellman key exchange:

Alice : (Key, Key) Key • T, Msg • T \Rightarrow Key • (Key ◦ (cMsg • T))

Alice (k_1, k_2) $\alpha, \beta \Rightarrow \gamma$

get a on α

put $\text{mod}_{k_1}(k_2^a)$ on γ

get b on γ

get m on β

put $\text{encode}(m, \text{mod}_{k_1}(b^a))$ on γ

close α

close β

end γ

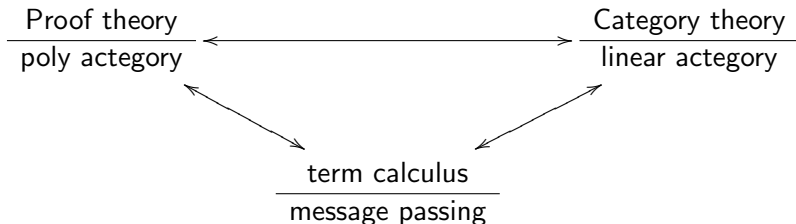
*Given two public keys k_1 and k_2 , Alice gets a secret key, a , on secure channel α . Does a key exchange on insecure channel γ talking to Bob!
Gets a plain text message on (secure) channel β , encrypts it with now agreed key; sends this on the insecure channel γ to Bob*

A Reference:

Cockett and Pastro

The logic of message passing

Science of computer programming 74 (2009) 498-533



(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!

Inductive datatypes

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

$$\text{cons}_F : F(\mu x.F(x)) \rightarrow \mu x.F(x)$$

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

$$\begin{array}{ccc}
 F(\mu x.F(x)) & \xrightarrow{\text{cons}_F} & \mu x.F(x) \\
 \downarrow F(\{\!\{g}\!\}_F) & & \downarrow \{\!\{g}\!\}_F \\
 F(Z) & \xrightarrow{g} & Z
 \end{array}$$

commutes.

Coinductive datatypes

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

$$\text{dest}_F : \nu x.F(x) \rightarrow F(\nu x.F(x))$$

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

$$\begin{array}{ccc}
 Z & \xrightarrow{g} & F(Z) \\
 \downarrow (\llbracket g \rrbracket)_F & & \downarrow F(\llbracket g \rrbracket)_F \\
 \nu x.F(x) & \xrightarrow{\text{dest}_F} & \nu x.F(x)
 \end{array}$$

commutes.


Rules for fixed points

The “Kozen rules” for fixed points are:

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

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

Problem: these do not work well² in multi- or poly- type theories!

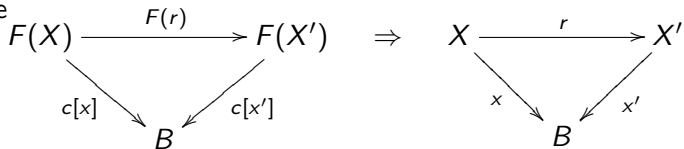
²Unless you can flip propositions from one side to the other ... 

Inductive circular datatypes

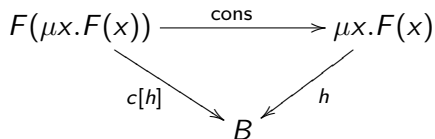
A combinator:

$$\frac{f : X \rightarrow B}{c[f] : F(X) \rightarrow B}$$

where



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



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

Coinductive circular datatypes

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 \mathbf{X} , there is a **cocircular map** $\nu b.c[b] : B \rightarrow \nu x.G(x)$ such that

$$\begin{array}{ccc}
 & B & \\
 u \swarrow & & \searrow c[u] \\
 \nu x.G(x) & \xrightarrow{\text{dest}} & G(\nu x.G(x))
 \end{array}$$

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

See also Tarmo Uustalu and Varmo Vene (thesis).

Circular rules for polycategories

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

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

$\frac{X = \mu x.P(x) \mid \Gamma, X, \Gamma' \vdash_X \Delta}{\Gamma, P(X), \Gamma' \vdash \Delta} \vdots$	$\frac{X = \nu x.Q(x) \mid \Gamma \vdash_X \Delta, X, \Delta'}{\Gamma \vdash \Delta, Q(X), \Delta'} \vdots$
$\Gamma, \mu x.P(x), \Gamma' \vdash \Delta$	$\Gamma \vdash \Delta, \nu x.Q(x), \Delta'$

Expressiveness of protocols

Adding datatypes increases expressiveness dramatically!

One can define 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 undecidable.

Programming with protocols

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

drive

duplicator:: $\text{Talk}(A, B * B) \Rightarrow \text{Talk}(A, B), \text{Talk}(A, B)$

duplicator:: $\alpha \Rightarrow \beta, \gamma$ by $\alpha =$

$\#talk:$

get x on α

put $\#talk$ on β ; put $\#talk$ on γ

put x on β ; put x on γ

get y_1 on β ; get y_2 on γ

put (y_1, y_2) on α

call duplicator($\alpha \Rightarrow \beta, \gamma$)

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 on the semantics of concurrency ...

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

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