Lambda calculs et catégories

Paul-André Melliès

Master Parisien de Recherche en Informatique

Ecole Normale Supérieure
Roadmap

1 – Limits and colimits
2 – Kan extensions
3 – Monoids
4 – Mnemoids and the state monad
5 – Local mnemoids and the local state monad
Part I

Limits and colimits

A generalization of cartesian products
Cones

Suppose given a functor

\[ F : \mathcal{I} \rightarrow \mathcal{C} \]

**Definition.**

A cone from an object \( L \) to the functor \( F \)

\[ \theta : L \rightarrow F \]

is defined as a family of morphisms

\[ \theta_i : L \rightarrow F(i) \]

indexed by the objects of the category \( \mathcal{I} \) such that the diagram

commutes for every morphism \( f : i \rightarrow j \) of the category \( \mathcal{I} \).
Limits

Suppose given a functor

\[ F : J \to C \]

Definition.

A limit of the functor \( F \) is an object \( L \) together with a cone

\[ \theta : L \to F \]

such that for every other cone

\[ \nu : X \to F \]

there exists a unique morphism

\[ h : X \to L \]

such that

\[ \mu_i = \theta_i \circ h : X \to L \to F(i). \]
Example: the cartesian products

The indexing category $J$ of the diagram

\[
F : J \rightarrow C
\]

is the discrete category with two objects.
Example: the pullbacks

The indexing category $J$ of the diagram

$$F : J \rightarrow C$$

is the category

[Diagram]

- $0$
- $1$
- $2$
Cones

Suppose given a functor

\[ F : J \to \mathcal{C} \]

**Definition.**

A cone from the functor \( F \) to an object \( K \)

\[ \theta : F \to K \]

is a family of morphisms

\[ \theta_i : F(i) \to K \]

indexed by the objects of the category \( J \) such that the diagram

\[ \begin{array}{ccc}
F(i) & \xrightarrow{\theta_i} & K \\
\downarrow F(f) & & \downarrow \theta_j \\
F(j) & \xrightarrow{\theta_i} & K
\end{array} \]

commutes for every morphism \( f : i \to j \) of the category \( J \).
Colimits

Suppose given a functor
\[ F : J \rightarrow \mathcal{C} \]

**Definition.**

A colimit of the functor \( F \) is an object \( K \) together with a cone
\[ \theta : F \rightarrow K \]
such that for every other cone
\[ \nu : F \rightarrow X \]
there exists a unique morphism
\[ h : K \rightarrow X \]
such that
\[ \mu_i = h \circ \theta_i : F(i) \rightarrow K \rightarrow X. \]
Example: the sums

The indexing category $J$ of the diagram

\[ F : J \rightarrow C \]

is the discrete category with two objects.
Example: the pushouts

The indexing category $J$ of the diagram

\[
F : J \rightarrow C
\]

is the category

\[
\begin{array}{c}
1 \\
\downarrow \\
0 \\
\downarrow \\
2
\end{array}
\]
Filtered categories

Definition. A category $J$ is filtered when

(1) it is not empty

(2) for every pair of objects $i$ and $j$

there exists an object $k$ and two morphisms

(3) for every pair of morphisms $f : i \to j$ and $g : i \to j$

there exists a morphism $h : j \to k$

such that

$h \circ f = h \circ g$
Final functors

Definition  A functor

\[ F : A \rightarrow B \]

is called final when

(1) for every object \( B \in B \), there exists an object \( A \in A \) and a morphism

\[ f : B \rightarrow FA \]

(2) given an object \( B \in B \), every pair of morphisms

\[ f_1 : B \rightarrow FA_1 \quad f_2 : B \rightarrow FA_2 \]

are related by the smallest equivalence relation \( \sim_B \) satisfying

\[ B \xrightarrow{f} FA \sim_B B \xrightarrow{g} FA' \]

whenever

\[ g = Fu \circ f \]

for some \( u : A \rightarrow A' \).
Final functors

Proposition.

A functor

\[ F : A \to B \]

is final precisely when for every functor

\[ G : B \to \text{Set} \]

the canonical morphism

\[ \text{colim}(G \circ F) \to \text{colim}(G) \]

is an isomorphism.
Part II

Kan extensions

A generalization of limits and colimits
Left Kan extensions

The left Kan extension of a functor

\[ F : A \to C \]

along a functor

\[ G : A \to B \]

is defined as a functor

\[ \text{Lan}_G(F) : B \to C \]

equipped with a natural transformation

\[ \Rightarrow \eta \]

defining a bijection

\[ \text{Nat}(B, C)(\text{Lan}_G(F), H) \cong \text{Nat}(A, C)(F, H \circ G) \]

for every functor

\[ H : B \to C \]
Left Kan extensions

The functor

\[ G : A \rightarrow B \]

defines a functor

\[ - \circ G : \text{Nat}(B, C) \rightarrow \text{Nat}(A, C) \]

between categories of functors and natural transformations.

The Kan extension property means that the natural transformation

\[ \eta : F \rightarrow \text{Lan}_G(F) \circ G \]

represents the functor

\[ \text{Nat}(B, C) \rightarrow \text{Sets} \]

which transports every functor

\[ H : B \rightarrow C \]

to the set of natural transformations

\[ \text{Nat}(A, B) (F, H \circ G) \]
Right Kan extension

The right Kan extension of a functor

$$F : A \to C$$

along a functor

$$G : A \to B$$

is defined as a functor

$$\text{Ran}_G(F) : B \to C$$

equipped with a natural transformation

$$\varepsilon$$

defining a bijection

$$\text{Nat}(B, C)(H, \text{Ran}_G(F)) \cong \text{Nat}(A, C)(H \circ G, F)$$

for every functor

$$H : B \to C$$
Slogan

Kan extension = 2-dimensional closure

Indeed, one observes that

\[ \text{Ran}_G(F) = F \rightleftharpoons G \]

in the spirit of cartesian closed categories.
Finitary functors

Definition. A functor

\[ F : \text{Set} \longrightarrow \text{Set} \]

is called finitary when it defines together with the identity 2-cell

\[
\begin{array}{ccc}
\text{Set} & \xrightarrow{\text{id}} & \text{FinSet} \\
F & \downarrow & \downarrow \iota \\
\iota \circ F & \longrightarrow & \iota \\
\end{array}
\]

a left Kan extension of the functor \( F \circ \iota \) along the functor \( \iota \).

Theorem. A functor is finitary if and only if it preserves the filtered colimits.
Illustration

The identity functor together with the identity 2-cell

\[
\begin{array}{ccc}
Set & \rightarrow & \text{FinSet} \\
\downarrow^{\text{id}} & \Rightarrow & \downarrow^{\text{Id}} \\
\text{FinSet} & \rightarrow & \text{Set}
\end{array}
\]

defines a left Kan extension of the functor \( t \) along the functor \( t \).

This establishes that the identity functor

\[
\text{Id} : \text{Ens} \rightarrow \text{Ens}
\]

is finitary.
Preservation of Kan extension

A functor

\[ F : \text{C} \rightarrow \text{D} \]

preserves a left Kan extension
Preservation of Kan extension

when the functor

\[ F \circ H : B \to C \]

together with the composite 2-cell

\[ F \circ \eta : F \circ G \Rightarrow F \circ H \]

defines a left Kan extension

of the functor \( F \circ G \) along the functor \( \ell \).
Alternative definition

A functor

\[ F : \text{Set} \to \text{Set} \]

is finitary precisely when it preserves the previous left Kan extension.

This means that the composite 2-cell

\[
\begin{array}{ccc}
\text{Set} & \xrightarrow{\text{id}} & \text{FinSet} \\
\text{id} & \Rightarrow & \text{Id} \\
\text{Set} & \xrightarrow{\iota} & \text{Set}
\end{array}
\]

defines a left Kan extension of the functor \( F \circ \iota \) along the functor \( \iota \).
Arrow category

Every pair of functors

\[ A \xrightarrow{F} C \]

induces a category

\[ F \downarrow G \]

together with a natural transformation

\[ \pi_1 \Rightarrow \pi_2 \]
Arrow category

The arrow category $F \downarrow G$ has the triples

$$(A, B, f)$$

as objects, where $A \in A$, $B \in B$ and $f$ is a morphism

$$FA \xrightarrow{f} GB$$

in the category $C$. Its morphisms

$$(A_1, B_1, f_1) \rightarrow (A_2, B_2, f_2)$$

are the pairs $(u, v)$ of morphisms

$$A_1 \xrightarrow{u} A_2 \quad B_1 \xrightarrow{v} B_2$$

making the diagram below commute:

\[
\begin{array}{ccc}
F(A_1) & \xrightarrow{Fu} & F(A_2) \\
\downarrow f_1 & & \downarrow f_2 \\
G(B_1) & \xrightarrow{Gv} & G(B_2)
\end{array}
\]
Fundamental and useful observation

**Proposition.** The functor $\text{Lan}_\ell(F) \circ G$ together with the composite 2-cell

$$\eta \Rightarrow \ell \downarrow G \Rightarrow \pi_2 \rightarrow C$$

defines a left Kan extension of the functor $F \circ \pi_1$ along the functor $\pi_2$. 
Application

The image of the object $B \in \mathcal{B}$ by the functor $\text{Lan}_\ell (F)$

\[
\begin{array}{ccc}
\text{Set} & \xrightarrow{\text{Lan}_\ell (F)} & \mathcal{B} \\
F & \xrightarrow{\eta} & \mathcal{A} \\
\downarrow & & \downarrow \ell \\
\mathcal{A} & \xrightarrow{\pi_1} & \mathcal{B} \\
\downarrow \ell & & \downarrow \pi_2 \\
\mathcal{B} & \xrightarrow{\pi_2} & \mathbb{1} \\
\end{array}
\]

is the colimit of the functor

\[
\ell \downarrow \mathcal{B} \xrightarrow{\pi_1} \mathcal{A} \xrightarrow{F} \text{Set}
\]
Key example

Every set $A$ is the filtered colimit of the functor

$$FinSet \downarrow A \xrightarrow{\pi_1} A \xrightarrow{\iota} Set$$
One direction of the equivalence

Corollary.

A functor $F : \text{Set} \to \text{Set}$ which preserves all filtered colimits is finitary.
The other direction

**Proposition.** A finitary functor $F : \text{Set} \to \text{Set}$ preserves all filtered colimits.

**Proof.** Suppose given a filtered diagram $G : \mathcal{J} \to \text{Set}$.

The finitary functor $F$ preserves the left Kan extension below:

![Diagram](image_url)
The other direction

Let us denote

\[ A := \text{colim}(G) \]

the filtered colimit of the diagram \( G \).

The cone between the diagram \( G \) and its colimit \( A \) induces a functor

\[ L : \text{FinSet} \downarrow G \longrightarrow \text{FinSet} \downarrow A \]

One shows that this functor \( L \) is final.

Now, recall that the set \( FA \) is a colimit of the diagram

\[ \text{FinSet} \downarrow A \xrightarrow{\pi_1} \text{FinSet} \xrightarrow{i} \text{Set} \xrightarrow{F} \text{Set} \]

From this follows that \( FA \) is also a colimit of the diagram

\[ \text{FinSet} \downarrow G \xrightarrow{L} \text{FinSet} \downarrow A \xrightarrow{\pi_1} \text{FinSet} \xrightarrow{i} \text{Set} \xrightarrow{F} \text{Set} \]

thus of the diagram

\[ \text{FinSet} \downarrow G \xrightarrow{\pi_1} \text{FinSet} \xrightarrow{i} \text{Set} \xrightarrow{F} \text{Set} \]

This concludes the proof.
Part III

Monoids

Algebraic presentation vs. free monoid monad
Monoids

A set $M$ equipped with a multiplication and a unit $e$ satisfying:

\[
(x \cdot y) \cdot z = x \cdot (y \cdot z)
\]

\[
x \cdot e = x = e \cdot x
\]

for all elements $x, y, z \in M$. 
Monoids

A set $M$ equipped with a **multiplication** and a **unit** map:

\[
m : M \times M \rightarrow M \\
(x, y) \mapsto x \cdot y
\]

\[
e : \{\ast\} \rightarrow M \\
\ast \mapsto e
\]
Monoids

A set $M$ equipped with a **multiplication** and a **unit** map:

$$m : M^2 \rightarrow M \\
(x, y) \mapsto x \cdot y$$

$$e : M^0 \rightarrow M \\
* \mapsto e$$
Monoids

such that the **associativity** diagram commutes:

\[
\begin{array}{ccc}
M \times M \times M & \xrightarrow{M \times m} & M \times M \\
\downarrow m \times M & & \downarrow m \\
M \times M & \xrightarrow{m} & M
\end{array}
\]

\[(x \cdot y) \cdot z = x \cdot (y \cdot z)\]
Monoids

such that the **associativity** diagram commutes:

\[
\begin{array}{ccc}
  x & y & z \\
  \downarrow & \downarrow & \downarrow \\
  x \cdot y & z \\
\end{array}
\xrightarrow{m \times M}

\begin{array}{ccc}
  x \cdot y & z \\
  \downarrow & \downarrow \\
  (x \cdot y) \cdot z = x \cdot (y \cdot z)
\end{array}
\]
Monoids

and such that the two unity diagrams commute:

\[
x \cdot e = x = e \cdot x
\]
In string diagrams

\[ m \circ m \circ m = m \]

\[ e \circ m = m = m \circ e \]
The “free monoid” monad

The set of finite words on an alphabet $A$ defines a monoid

$$TA = A^0 + A^1 + A^2 + \cdots + A^n + \cdots$$

with concatenation as multiplication and the empty word as unit.

$$\eta_A : A \rightarrow TA$$

$a \mapsto [a]$

$$\mu_A : TTA \rightarrow TA$$

$$[[ab][aa][b]] \mapsto [abaab]$$
Free monoid

By free monoid, one means that for every function

\[ f : A \to M \]

there exists a \textbf{unique} homomorphism

\[ h : TA \to M \]

making the diagram commute:
Algebras of a monad

A set $A$ equipped with a function

$$\text{alg} : TA \rightarrow A$$

making the two diagrams below commute:
The category of monoids

Let $\mathcal{M}$ denote the category

- with monoids as objects
- with homomorphisms as maps

Folklore theorem

$\mathcal{M}$ is equivalent to the category of algebras of the monad $T$
Part IV

Mnemoids

An algebraic presentation of memory accesses
The state monad

A program accessing one register with a finite set $V$ of values

$$A \xrightarrow{\text{impure}} B$$

is interpreted as a function

$$V \times A \xrightarrow{\text{pure}} V \times B$$

thus as a function

$$A \xrightarrow{\text{pure}} V \Rightarrow (V \times B)$$

Hence, the state monad

$$T : A \leftrightarrow V \Rightarrow (V \times A)$$
What is an algebra of the state monad?
Mnemoids

A mnemoid is a set $A$ equipped with a $V$-ary operation $\text{lookup} : A^V \rightarrow A$ and a unary operation $\text{update}_{\text{val}} : A \rightarrow A$ for each value $\text{val} \in V$. 
Mnemoids

Typically, the lookup operation on the boolean space

$$V = \{ \text{false, true} \}$$

is binary:

$$\text{lookup} : A \times A \rightarrow A$$

The intuition is that

$$\text{lookup} (u, v) = \begin{cases} u & \text{when the register is false} \\ v & \text{when the register is true} \end{cases}$$
Annihilation lookup – update

\[
\text{term} = \text{lookup} \left[ \text{val} \mapsto \text{update}_{(\text{val})} \text{term} \right]
\]
Annihilation lookup – update

Here, the map

$$\text{update}_\langle V \rangle : A \rightarrow A^V$$

is the $V$-ary vector

$$\left[ \text{update}_\langle 0 \rangle , \cdots , \text{update}_\langle V-1 \rangle \right]$$

also noted

$$\left[ \text{val} \mapsto \text{update}_\langle \text{val} \rangle \right]$$
Annihilation lookup – update

\[ x \quad \square \quad = \quad x \quad \text{true} \quad \circ \quad x \quad \text{false} \]
Interaction update – update

\[
\text{update}_{\langle \text{val}_2 \rangle} \circ \text{update}_{\langle \text{val}_1 \rangle} = \text{update}_{\langle \text{val}_2 \rangle}
\]
Interaction update – update
Interaction update – lookup

\[ \text{update}_{\langle \text{val} \rangle} \circ \text{lookup} \left[ x \mapsto \text{term}(x) \right] = \text{update}_{\langle \text{val} \rangle} \circ \text{term}(\text{val}) \]
Interaction update – lookup

\[ x \quad \text{true} \quad y \quad = \quad x \quad \text{true} \]

55
Key theorem (Plotkin & Power)

the category of mnemoids

is equivalent to

the category of algebras of the state monad

Provides an algebraic presentation of the state monad!
Store with several locations

Algebraic presentation of the state monad
The global state monad

The state monad is generally defined as

\[ T : A \mapsto S \Rightarrow (S \times A) \]

for a set of states

\[ S = V^L \]

induced by a set \( L \) of locations and a \textbf{finite} set \( V \) of values.
Global store (Plotkin & Power)

A global store is a family of compatible mnemoidal structures

\[
\text{lookup}_{(loc)} : A^V \rightarrow A
\]

\[
\text{update}_{(loc,val)} : A \rightarrow A
\]

one for each location \(loc \in L\).

A tensor product of algebraic theories
Annihilation lookup – update

\[
\text{lookup}_{(\text{loc})} \left[ \text{val} \mapsto \text{update}_{(\text{loc}, \text{val})} \text{term} \right] = \text{term}
\]
Annihilation lookup – update
Interaction update – update

\[
\text{update}_{\langle \text{loc}, \text{val} \rangle} \circ \text{update}_{\langle \text{loc}, \text{val}' \rangle} = \text{update}_{\langle \text{loc}, \text{val}' \rangle}
\]
Interaction update – update

\[ \text{val}_2 \text{val}_1 = \text{val}_2 \]

63
**Interaction update – lookup**

\[
\text{update}_{\langle \text{loc}, \text{val} \rangle} \circ \text{lookup}_{\langle \text{loc} \rangle} \left[ x \mapsto \text{term}(x) \right] = \text{update}_{\langle \text{loc}, \text{val} \rangle} \circ \text{term}(\text{val})
\]
Interaction update – lookup

\[ x \quad \text{true} \quad y = x \]
Interaction update – lookup

\[ x = \text{false} \]

\[ y \]

\[ y = \text{false} \]

\[ y = \text{false} \]
Commutation update – update

\[ \text{update}^{\langle \text{loc}, \text{val} \rangle} \rightarrow \text{update}^{\langle \text{loc}', \text{val}' \rangle} \]

\[ \text{update}^{\langle \text{loc}', \text{val}' \rangle} \rightarrow \text{update}^{\langle \text{loc}, \text{val} \rangle} \]

\[ \text{update}^{\langle \text{loc}, \text{val} \rangle} \text{ update}^{\langle \text{loc}', \text{val}' \rangle} x = \]

\[ \text{update}^{\langle \text{loc}', \text{val}' \rangle} \text{ update}^{\langle \text{loc}, \text{val} \rangle} x \]

\[ \text{when} \quad \text{loc} \neq \text{loc}' \]
Commutation update – update

\[ \text{val} = \text{val}' \]
Corollary (Plotkin & Power)

the category of objects with a global store

is equivalent to

the category of algebras of the state monad
Part V

Local states

An algebraic presentation of local variables
Functorial semantics

**Key idea:** interpret a type $A$ as a family of sets

$$A[0] \quad A[1] \quad \cdots \quad A[n] \quad \cdots$$

indexed by natural numbers, where each set $A[n]$ contains the programs of type $A$ which have access to $n$ variables.
Functorial semantics

This defines a covariant presheaf

$$A_{[n]} : \text{Inj} \rightarrow \text{Sets}$$

on the category $\text{Inj}$ of natural numbers and injections.

The action of the injections on $A$ are induced by the operations

$$\text{collect}_{\langle \text{loc} \rangle} : A_{[n]} \rightarrow A_{[n+1]}$$

defined for $0 \leq \text{loc} \leq n$. 
Functorial semantics

The intuition is that

\[ A_{[n]} \]

contains the programs with \( n \) locations. Every injection

\[ f : p \to q \]

defines a function

\[ A_{[f]} : A_{[p]} \to A_{[q]} \]

which transports every program

\[ P \in A_{[p]} \]

to the program

\[ P[\text{loc} \leftarrow f(\text{loc})] \in A_{[q]}. \]
A cartesian closed category

The category $[\text{Inj}, \text{Set}]$ is cartesian

$$A \times B : n \mapsto A_{[n]} \times B_{[n]}$$

and closed

$$A \Rightarrow B : n \mapsto [\text{Inj}, \text{Set}] ( A \times \text{Inj}(n, -) , B )$$
A monoidal closed category

The category \([\text{Inj}, \text{Set}]\) is symmetric monoidal

\[
A \otimes B : n \mapsto \int_{p,q \in \text{Inj}} A[p] \times B[q] \times \text{Inj}(p + q, n)
\]

and closed

\[
A \rightarrow B : n \mapsto [\text{Inj}, \text{Set}](A, B_{[n+1]})
\]
Local stores (Plotkin – Power)

The slightly intimidating monad

\[ TA : n \mapsto S^n \Rightarrow \left( \int^{p \in \text{Inj}} S^p \times A^p \times \text{Inj}(n, p) \right) \]

on the presheaf category \([\text{Inj}, \text{Set}]\) where the contravariant presheaf

\[ S^p = V^p \]

describes the states available at degree \( p \).
**Local mnemoids**

A **local mnemoid** is a family of sets

\[ A_0 \quad A_1 \quad \ldots \quad A_n \quad \ldots \]

equipped with the following operations

- **lookup** \(_{\langle \text{loc} \rangle} \) : \( A^n \rightarrow A^n \)
  \( 0 \leq \text{loc} \leq n - 1 \)

- **update** \(_{\langle \text{loc}, \text{val} \rangle} \) : \( A^n \rightarrow A^n \)
  \( 0 \leq \text{loc} \leq n - 1 \)

- **fresh** \(_{\langle \text{loc}, \text{val} \rangle} \) : \( A^{n+1} \rightarrow A^n \)
  \( 0 \leq \text{loc} \leq n \)

- **collect** \(_{\langle \text{loc} \rangle} \) : \( A^n \rightarrow A^{n+1} \)
  \( 0 \leq \text{loc} \leq n \)

- **permute** \(_{\langle \text{loc}, \text{loc}' \rangle} \) : \( A^n \rightarrow A^n \)
  \( 0 \leq \text{loc} < \text{loc}' \leq n - 1 \)

satisfying a series of basic equations.
Interaction update – update
Interaction update – lookup
Interaction update – lookup

\[ x = y \]
Interaction fresh – dispose
Interaction fresh – update
Interaction fresh – lookup
Interaction fresh – lookup
Interaction fresh – permutation
Modules over the category $Inj$

An $Inj$-module is a category $C$ equipped with an action

$$\ast : Inj \times C \rightarrow C$$

$$(m, A) \quad m \ast A$$

satisfying the expected properties:

$$(p + q) \ast A = p \ast (q \ast A)$$

$$0 \ast A = A$$
Modules over the category $\text{Inj}$

A category $\mathcal{A}$ equipped with an endofunctor

$$D : C \rightarrow C$$

and two natural transformations

$$\text{permute} : D \circ D \rightarrow D \circ D \quad \text{collect} : \text{Id} \rightarrow D$$

depicted as follows in the language of string diagrams:
Yang-Baxter equation

\[
\begin{array}{c}
\text{Diagram 1:} \\
\text{Diagram 2:}
\end{array}
\]

= 

88
Symmetry

\[ \begin{array}{ccc}
D & \circ \circ & D \\
D & & D \\
D & \circ \circ & D \\
D & & D
\end{array} = \begin{array}{ccc}
D & \hline & D \\
D & \hline & D \\
D & \hline & D \\
D & \hline & D
\end{array} \]
**Alternative formulation**

A **dynamic mnemoid** is a pair of sets

\[ A[0] \quad A[1] \]

equipped with the following operations

- **lookup** : \( A^V_{[1]} \rightarrow A_{[1]} \)
- **update\(_{(val)}\)** : \( A_{[1]} \rightarrow A_{[1]} \)
- **fresh\(_{(val)}\)** : \( A_{[1]} \rightarrow A_{[0]} \)
- **collect** : \( A_{[0]} \rightarrow A_{[1]} \)

satisfying a series of basic equations.
Garbage collect: fresh – collect

\[
\begin{align*}
A_0 & \xrightarrow{id} A_0 \\
A_0 & \xleftarrow{\text{collect}} A_1 \\
A_1 & \xrightarrow{\text{fresh}_{\text{val}}} A_0
\end{align*}
\]

\[
\text{fresh}_{\text{val}} \circ \text{collect} (\text{term}) = \text{term}
\]
Garbage collect: fresh – collect

![Diagram](image-url)
Interaction fresh – update

\[
\text{fresh}_{\langle \text{val}_1 \rangle} \circ \text{update}_{\langle \text{val}_2 \rangle} = \text{fresh}_{\langle \text{val}_2 \rangle}
\]
Interaction fresh – update

\[ \text{val}_2 \quad \text{val}_1 \quad = \quad \text{val}_2 \]
Interaction fresh – lookup

\[
\begin{array}{c}
A^V_{[1]} \xrightarrow{\text{lookup}} A_{[1]} \\
A_{[1]} \xrightarrow{\text{fresh}_{\langle\text{val}\rangle}} A_{[0]}
\end{array}
\]

\[
fresh_{\langle\text{val}\rangle} \text{ lookup } (\text{wal} \mapsto term[\text{wal}]) = fresh_{\langle\text{val}\rangle} (term[\text{val}])
\]
Interaction fresh – lookup

\[
\begin{align*}
\text{true} &= x \\
\text{false} &= y
\end{align*}
\]
The equation $D(A \times B) \cong DA \times DB$ means that space commutes with time ramification !!!