An equational presentation of the local state monad

Kenji Maillard

September 21, 2015

Algebraic theories

An algebraic theory (aka abstract clone, lawvere theory) is a set of operations of finite arity containing projections and stable by composition.

A generic n-ary operation will be noted

$$\mathrm{op} \quad : \quad [n] \quad \longrightarrow \quad [1]$$

Most of the time, algebraic theories are presented by an equational theory over some signature. In that case, the algebraic theory itself correspond to the terms over that signature modulo the equations.

Examples of algebraic theories

- Non-deterministics computations

  $\vee : [2] \to [1]$ commutative, associative, idempotent

- Input-output with input language Σ_I and output language Σ_O

  $in : [n_I] \to [1]$ where $n_I = |\Sigma_I|$
  $out_c : [1] \to [1]$ where $c \in \Sigma_O$

- Probabilistic computations

Counter-example : The continuation monad

$$A \longmapsto (A \Rightarrow R) \Rightarrow R$$

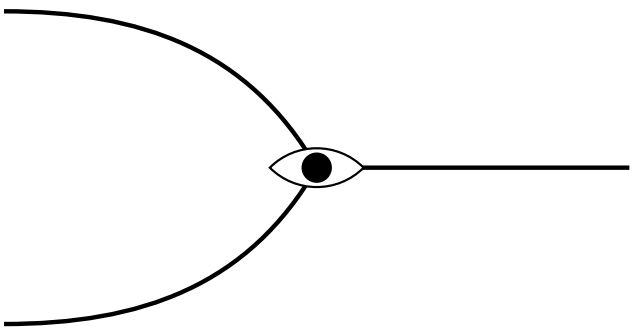
is not finitary, hence can not be presented by an algebraic theory

The theory of global state : Operations

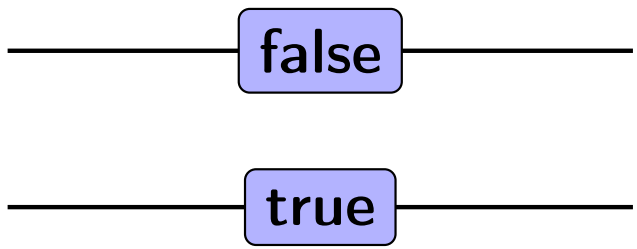
Let V be a fixed finite set of values and $n = Card(V)$.
For all the diagrams, $V = \{\textbf{true}, \textbf{false}\}$.

The theory contains a binary operation and n unary operations for each $val \in V$

$\texttt{lookup} : [n] \to [1]$



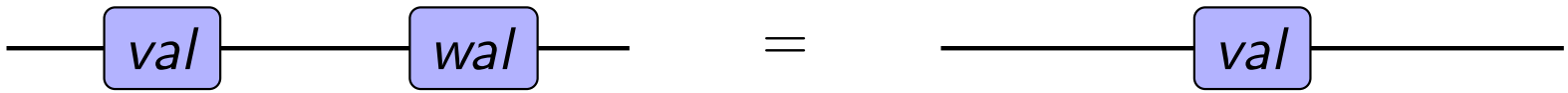
$\texttt{update}_{val} : [1] \to [1]$



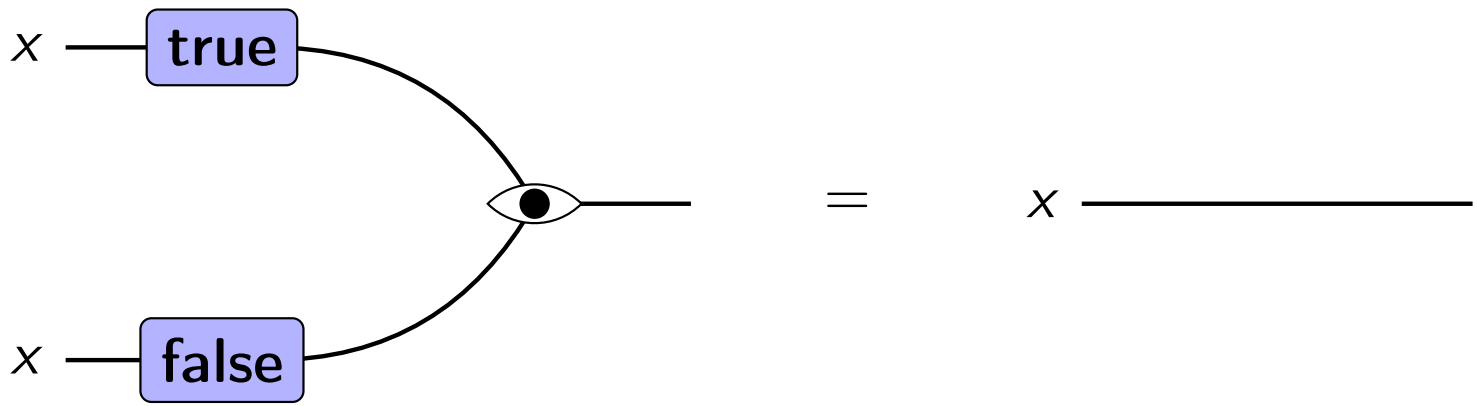
The theory of global state : Equations (1)

. . . subject to the following equations :

$$\mathtt{update}_{wal} \circ \mathtt{update}_{val} = \mathtt{update}_{val}$$



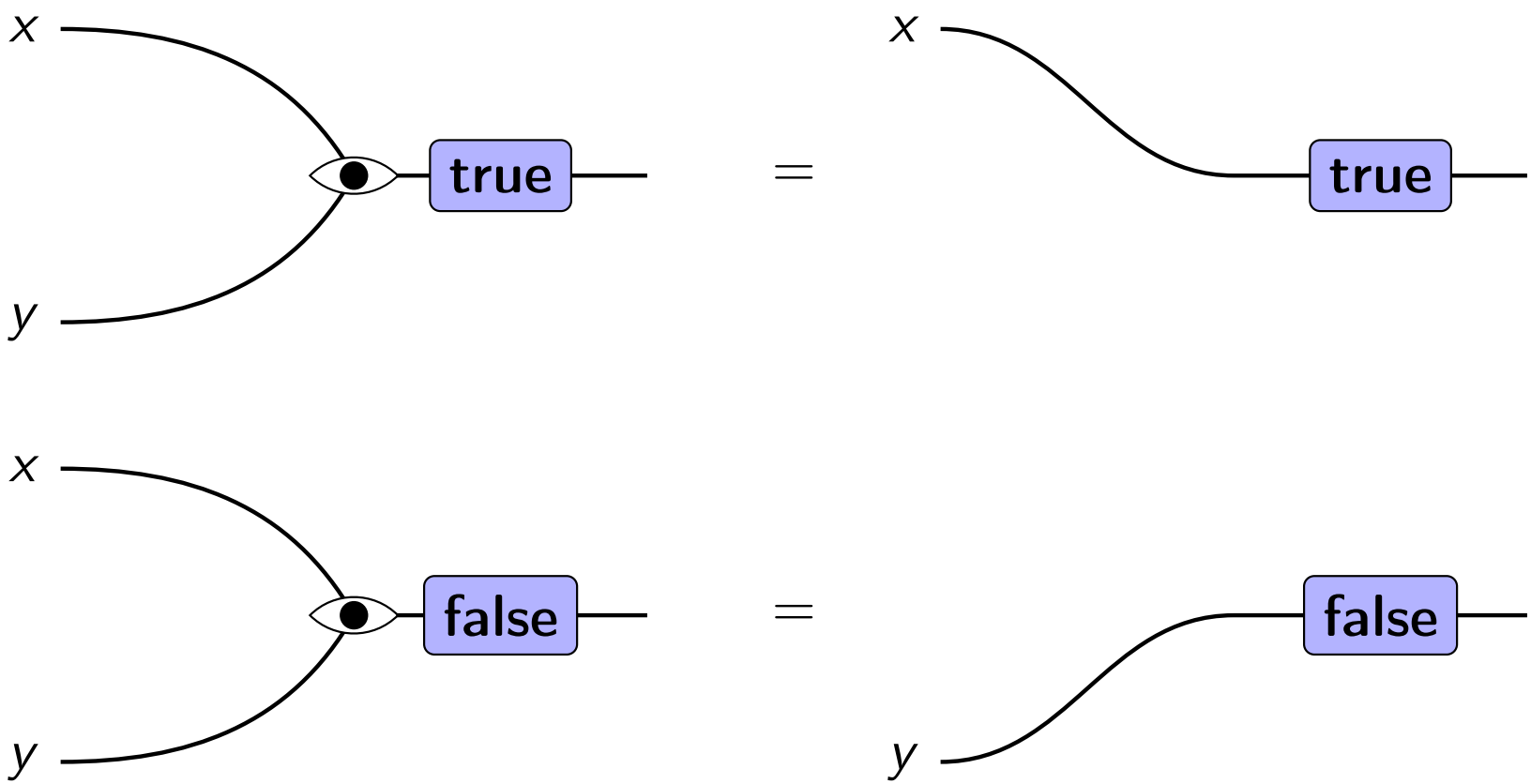
$$\mathtt{lookup} \circ \prod_{v \in V} \mathtt{update}_v \circ \delta = id$$



The theory of global state : Equations (2)

As well as this one :

$$\texttt{update}_{val} \circ \texttt{lookup} = \mathit{update}_{val} \circ \pi_{val}$$



Working on multiple cells

If we want to work with a **fixed** number k of cells we can :

- Change the set of value V and set

$$\tilde{V} = V^k$$

- Apply k times a state transformer

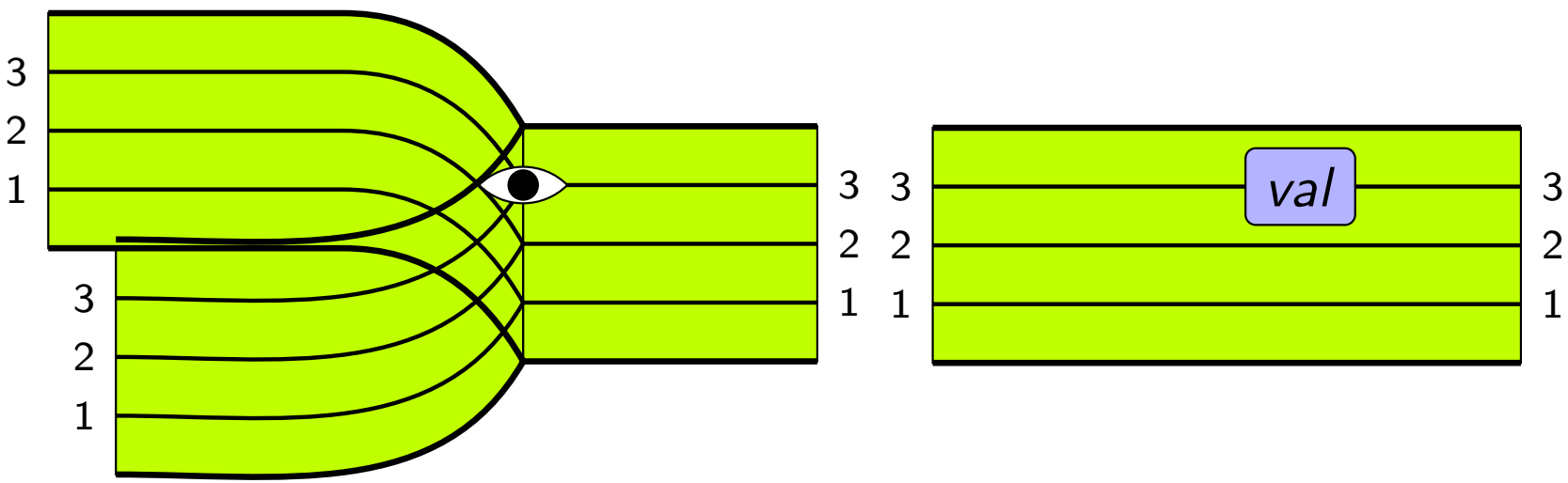
It was shown by Hyland, Plotkin and Power that the two coincides and can be derived from an operation $\otimes$ on algebraic theories.

Working on multiple cells diagrammatically

We index each operation by the cell *loc* on which it operates :

$\texttt{lookup}_{loc} : [n] \to [1]$

$\texttt{update}_{loc,val} : [1] \to [1]$



Local state : enabling allocation and deallocation

We want to add operations

$$\texttt{alloc}_{loc} : [1] \to [1] \qquad \texttt{dealloc}_{loc} : [1] \to [1]$$

morally giving and restraining access to a new cell at a fresh location *loc*

Of course, we want to forbid the following code

```
let l = alloc in
let () = dealloc l in
lookup l
```

In order to do so, we need to track the allocated locations.

State-indexed types

We assume an infinite family of sorts :

$$\langle 0 \rangle \qquad \langle 1 \rangle \qquad \langle 2 \rangle \qquad \ldots \qquad \langle p \rangle \qquad \ldots$$

which will index our types.

We can now think of a type A as a family

$$A_{\langle 0 \rangle} \qquad A_{\langle 1 \rangle} \qquad A_{\langle 2 \rangle} \qquad \ldots \qquad A_{\langle n \rangle} \qquad \ldots$$

where $A_{\langle p \rangle}$ is the type of terms knowing that there are exactly p memory locations allocated.

Coercions between state-index

Not only do we want to know how many cells were allocated, but we also want to track the lifetime of each memory cell.

We introduce coercions between sorts :

$$f \quad : \quad \langle p \rangle \quad \triangleright \quad \langle q \rangle$$

- We want a coercion $\langle p \rangle \triangleright \langle p \rangle$ for each permutation of the memory space
- We also want to track the modifications when going from one sort to another

$$\langle p \rangle \triangleright \langle p + 1 \rangle$$

Definition

A coercion $f : \langle p \rangle \triangleright \langle q \rangle$ is an injection

$$f \quad : \quad [p] \quad \hookrightarrow \quad [q]$$

State-indexed types : examples

- The type *Bool* of booleans

$$Bool_{\langle p \rangle} = \{\mathbf{true}, \mathbf{false}\} \quad Bool_{\langle f \rangle} = id : Bool_{\langle p \rangle} \to Bool_{\langle q \rangle}$$

- The type *Loc* of locations

$$Loc_{\langle p \rangle} = \{1, \ldots, p\} \qquad Loc_{\langle f \rangle} = f : Loc_{\langle p \rangle} \to Loc_{\langle q \rangle}$$

- The product $A \times B$ of two indexed types A and B

$$(A \times B)_{\langle p \rangle} = A_{\langle p \rangle} \times B_{\langle p \rangle} \qquad (A \times B)_{\langle f \rangle} = A_{\langle f \rangle} \times B_{\langle f \rangle}$$

- The function space $A \Rightarrow B$ between two indexed types

$$(A \Rightarrow B)_{\langle p \rangle} = A \times \mathcal{Y}(p) \to B$$

where $\mathcal{Y}(p)_{\langle q \rangle} = \{f : \langle p \rangle \rhd \langle q \rangle\}$

The legacy local state monad

Given such an indexed-type A, Plotkin and Power explain that we can compute the local state monad $\mathcal{T}$ as follows :

$$(\mathcal{T}A)_{\langle p \rangle} = V^p \Rightarrow \int^{q \in Inj} A_{\langle q \rangle} \times V^q \times Inj(p, q)$$

Regrettably the understanding of this formula seems to be reserved to a handful of caterrorists.

An introduction to theories over the arity ΣInj^{op}

The relevant notion of "algebraic theory" for our setting is called a *theory with arity* (c.f. Weber, Clemens, Melliès).

When comparing to the traditional case :

- it is multi-sorted with an infinite family $\langle p \rangle$ of sorts
- it contains at least the coercions $f : \langle p \rangle \triangleright \langle q \rangle$
- but the restriction to a specific sort $\langle p \rangle$ should be an algebraic theory

The operations are then of the form

$$\langle p_1 \rangle + \ldots + \langle p_k \rangle \longrightarrow \langle q \rangle$$

that is an operation takes as input a finite number of arguments of eventually distinct sorts and returns an ouput at any sort.

A presentation of the local state monad

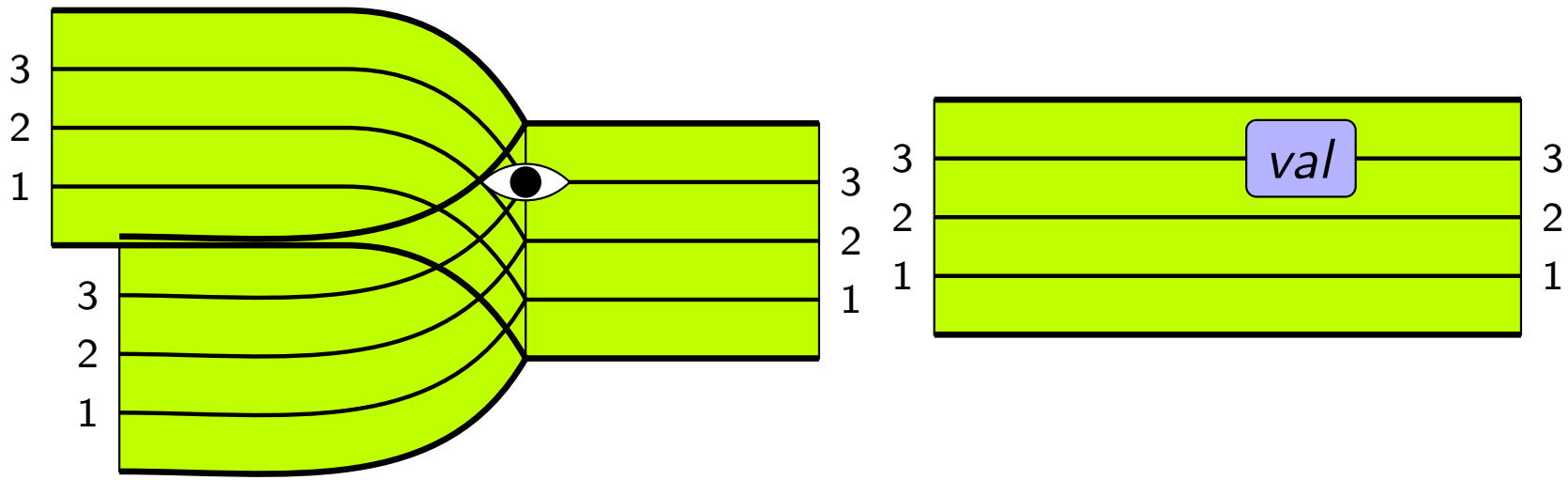
All the content that follows is taken from Melliès'14.
The presentation can be logically divided as follows :

- For each sort $\langle p \rangle$, we need the operations for manipulating p memory cells
- We need operations of allocation and deallocation that move from a sort $\langle p \rangle$ to another sort $\langle q \rangle$
- We then have to explain how the two different group of operations interact

Operations manipulating p memory cells

We get back our old friends from the algebraic theory of global state :

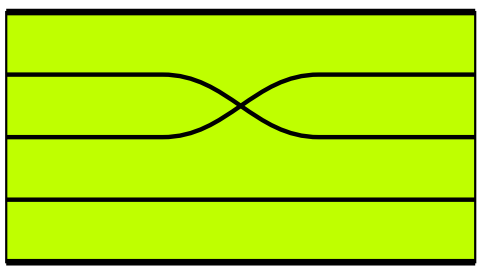
$$\texttt{lookup}_{loc} : \underbrace{\langle p \rangle + \ldots + \langle p \rangle}_{n \text{ times}} \to \langle p \rangle \qquad \texttt{update}_{loc,val} : \langle p \rangle \to \langle p \rangle$$

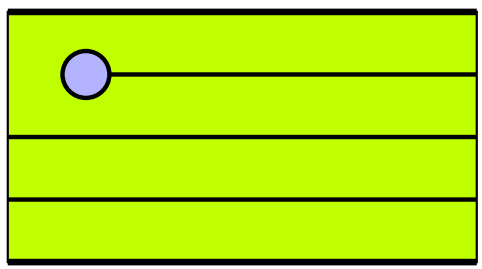


Deallocation & permutation

Let $p > 0$ and $loc, loc' \in \{1, \ldots, p\}$, then we have

$\texttt{permute}_{loc,loc'} : \langle p \rangle \to \langle p \rangle$ $\qquad$ $\texttt{dealloc}_{loc} : \langle p \rangle \to \langle p + 1 \rangle$





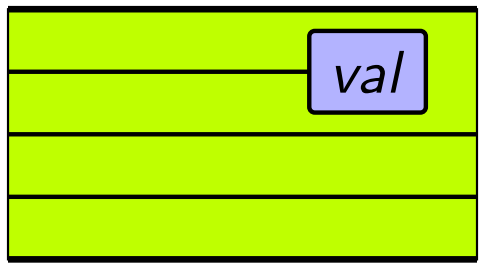
Here on the diagrams, we are at sort $\langle 3 \rangle$, $loc = 3$ and $loc' = 2$.

The coercions are in fact exactly the terms derived from these two families of operations.

Allocation

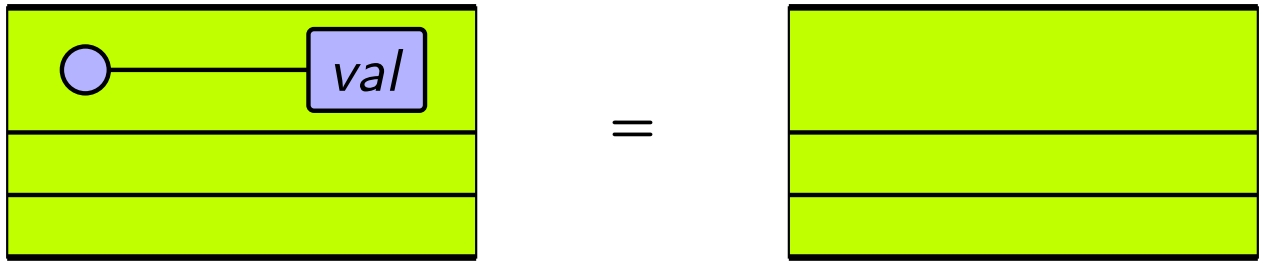
Suppose $p > 0$, $loc \in \{1, \ldots, p\}$ and $val \in V$, then we have :

$$\texttt{alloc}_{loc,val} : \langle p + 1 \rangle \longrightarrow \langle p \rangle$$

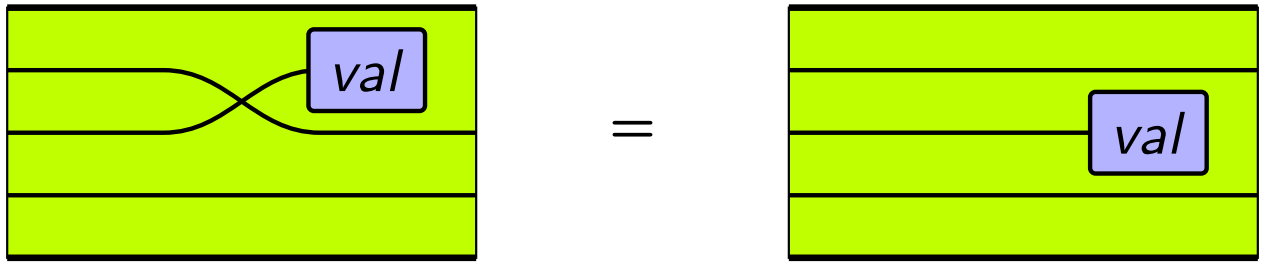


Equations for *Block*

`dealloc-alloc`



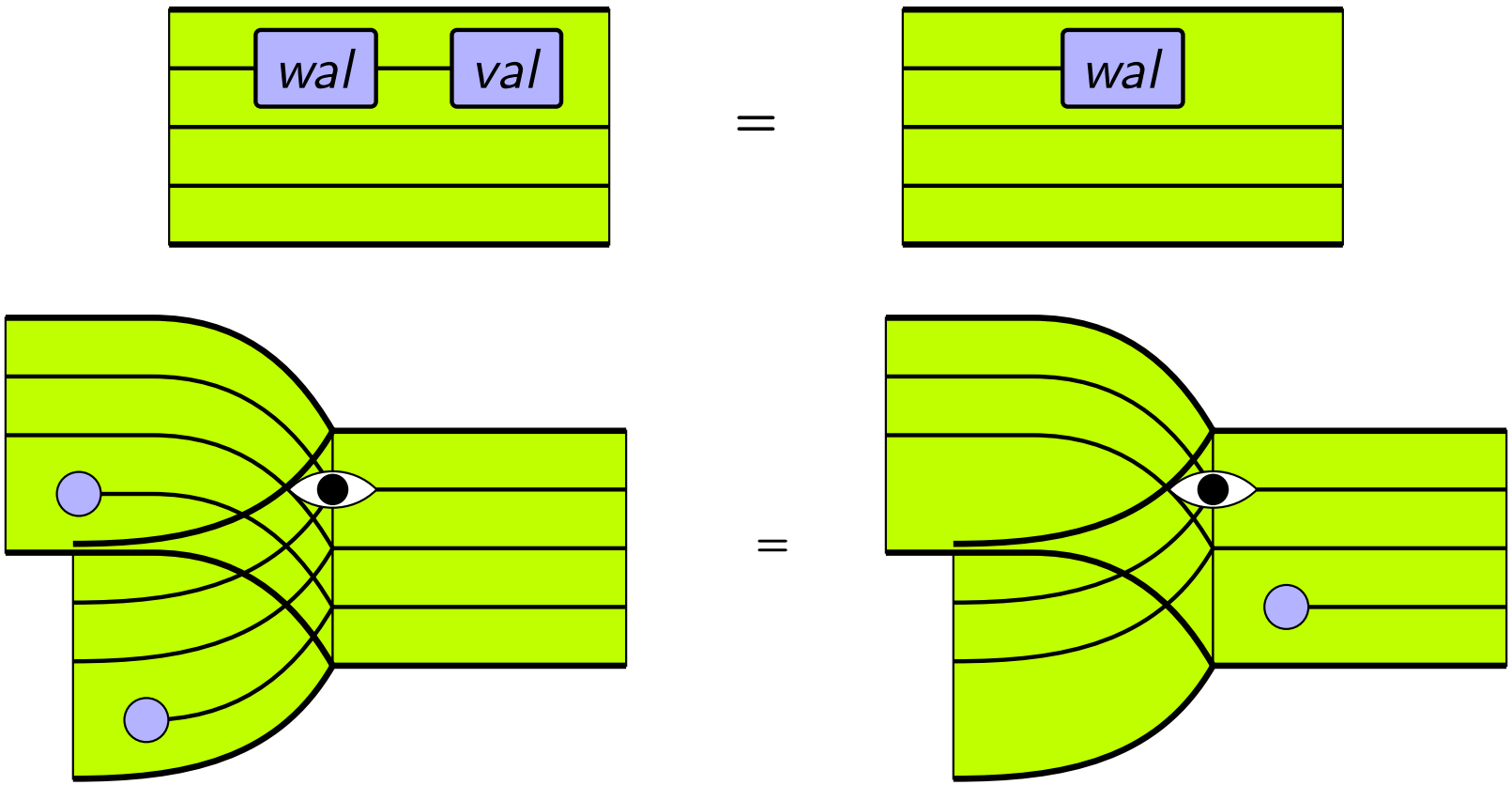
`permute-alloc`



Plus the equations for commutations when the operations are not on the same locations.

Equations for the distributive law

We now consider *some* equations involving a `lookup` or `update` with a *Block* operation :



Going back to the local state monad

Now that we have a presentation of the monad, we can try to have a new understanding of its formula.

$$(\mathcal{T}A)_{\langle p \rangle} = V^p \Rightarrow \left(V^p \times \int^{q,r \in Inj} A_{\langle q \rangle} \times Inj(q, p+r) \times V^r \right)$$

Indeed $(TA)_{\langle p \rangle}$ is the set of syntactical trees over the operations with root at sort $\langle p \rangle$ and leaves in A modulo the equations.

The normal forms can be understood as the following process

reading $\succ$ writing $\succ$ deallocating & permuting $\succ$ allocating

The Block and the distributive law

The previous formula can be decomposed as $\mathcal{T} = \mathcal{F} \circ \mathcal{B}$ with :

$$(\mathcal{F}A)_{\langle p \rangle} = V^p \to A_{\langle p \rangle} \times V^p$$

and

$$(\mathcal{B}A)_{\langle p \rangle} = \int^{q,r \in Inj} A_{\langle q \rangle} \times Inj(q, p + r) \times V^r$$

They are then recombined via a distributive law :

$$\lambda \quad : \quad \mathcal{B} \circ \mathcal{F} \quad \longrightarrow \quad \mathcal{F} \circ \mathcal{B}$$

A more modular presentation, but can we go further ?

Partial injections and rewriting

A partial injection from $[n]$ to $[m]$ is a pair

$$\langle p, f \rangle \quad : \quad [n] \quad \longrightarrow \quad [m]$$

$$p \in \mathbb{N} \qquad\qquad f : [n] \hookrightarrow [m + p]$$

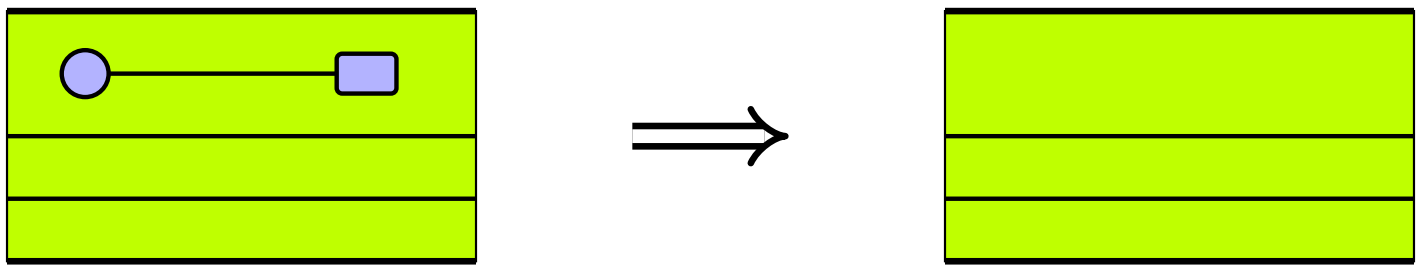
A rewriting between partial injections

$$\alpha \quad : \quad \langle p, f \rangle \quad \longrightarrow \quad \langle q, g \rangle$$

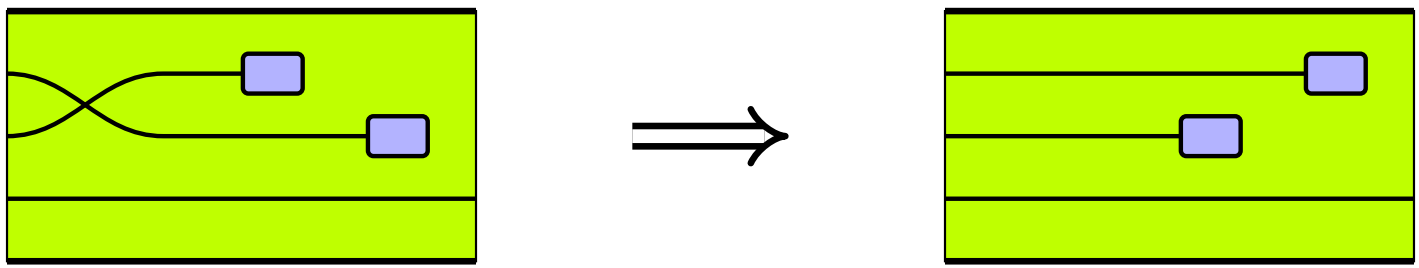
is an injection

$$\alpha \quad : \quad [q] \quad \longrightarrow \quad [p] \qquad \text{such that} \qquad f \quad = \quad (id_m + \alpha) \circ g$$

Diagrammatic rewriting : deallocation vs. allocation



Diagrammatic rewriting : permutation vs. allocation



A 2-dimensional algebraic theory for block

Now we can associate :

- To each natural number $n \in \mathbb{N}$ the monad

$$A \quad \longmapsto \quad S^n \Rightarrow A \times S^n$$

- To each partial injection $\langle r, f \rangle$ the monad transformer

applying r times the state monad transformer
then forgetting and permuting cells according to f

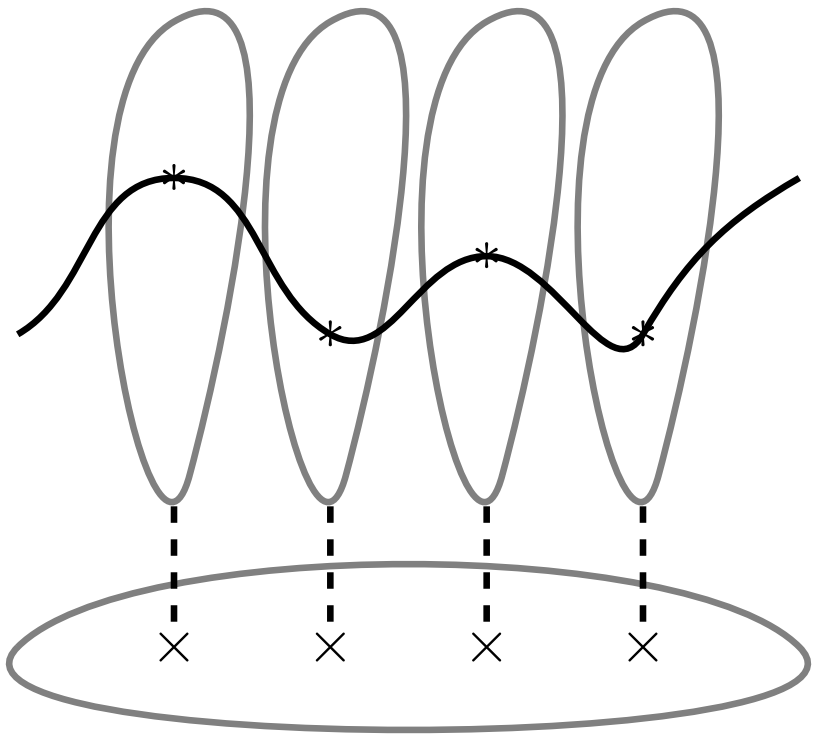
- To each rewriting path α between two partial injections a parametric function "optimising" the source monad transformers to the target monad transformers

Theorem

This 2-dimensional presentation captures exactly the algebras of the local state monad.

What we got until now

- A good understanding of an equational presentation of the monad of local state
- A tentative implementation of the local state monad in Ocaml
- A cute little categorical theorem about fibrations of monads



What's next ?

- Which programming language am I talking about ?
- How comes that the categorical description of operation is naturally in continuation passing style ?
- The monad transformer analogy is still a little too handwaving
- What is the relation to kleisli structures aka. relative monads ?

Questions ? . . . Answers ?