

A monad for full ground reference cells

Ohad Kammar

`<ohad.kammar@cs.ox.ac.uk>`

joint work with

Paul B. Levy, Sean K. Moss, and Sam Staton

<http://arxiv.org/abs/1702.04908>

University of Oxford

Department of Computer Science

Oxford Advanced Seminar on Informatic Structures (OASIS)

10 March 2017



THE ROYAL
SOCIETY



UNIVERSITY OF
CAMBRIDGE



UNIVERSITY OF
BIRMINGHAM

Dynamic allocation of memory cells

Kinds of storage

(* ground *)

type data = **bool**

(* full ground: refs to refs, cyclic data *)

type linked_list = **1** + **ref** list_cell

type list_cell = **ref** data * **ref** linked_list

(* general: suspended computation/effectful functions *)

type process = **1** + (**1** → **ref** process)

This talk

Full ground storage only

A semantic gap

Success stories

- ▶ Operational semantics: general
- ▶ Relational semantics [Benton, Hofmann, et al., Bohr and Birkedal'06]: general with dangling pointers
- ▶ Step-indexing [Ahmed'04, Birkedal et al.'10 etc]: general small print
- ▶ Game semantics: general storage with partiality
- ▶ Parametricity semantics [Reddy-Yang'04]: full ground

A dead end? [Reynolds and Oles'82, Moggi'90, O'Hearn and Tennent'92, Stark'94, Ghica'97, Plotkin-Power'02, Levy'02]

- ▶ Sets-with-structure and structure preserving functions
- ▶ Possible-world semantics
- ▶ Monad over a bi-CCC
- ▶ Avoid dangling pointers

Goal

Semantics for full ground storage:

- ▶ Sets-with-structure and structure preserving functions
- ▶ Possible-world semantics
- ▶ Monad over a bi-CCC
- ▶ Avoid dangling pointers

Evaluation

- ▶ Effect masking
- ▶ Adequacy
- ▶ Local ground storage equations

Talk structure

- ▶ Full ground storage
- ▶ Worlds and initialisations
- ▶ A monad for hiding/encapsulation
- ▶ The monad, explicitly
- ▶ Evaluation
- ▶ Conclusion

Aspiration

Computational motivation alongside technical development

Full ground storage

Full ground signature $\langle \mathbb{m}, \mathbf{type} \rangle$

- ▶ Countably many $c \in \mathbb{m}$ **storable-type names**
- ▶ A **full ground type interpretation** $\mathbf{type} : \mathbb{m} \rightarrow \mathbb{G}$

Where the **full ground types** $\gamma \in \mathbb{G}$ are:

$$\gamma ::= \mathbf{0} \mid \gamma_1 + \gamma_2 \mid \mathbf{1} \mid \gamma_1 * \gamma_2 \mid \mathbf{ref} \ c$$

Example

$\mathbb{m} \stackrel{\text{def}}{=} \{\text{data}, \text{linked_list}, \text{list_cell}\}$

```
type data      = bool  
type linked_list = 1 + ref list_cell  
type list_cell  = ref data * ref linked_list
```

A call-by-value calculus: λ_{ref} (syntax)

$\tau ::=$	types
ref c	reference
0	empty
$\tau_1 + \tau_2$	binary sum
1	unit
$\tau_1 * \tau_2$	product
$\tau_1 \rightarrow \tau_2$	function
$v ::=$	values
ℓ	location ($\ell \in \mathbb{L}$)
x	identifier
inj $_i^{\tau_1 + \tau_2} v$	constructor
$()$	unit
(v_1, v_2)	pair
$\lambda x : \tau. t$	function

Sugar

let $(x : \tau) = t$ **in** $s \equiv (\lambda x : \tau. s) t$

new $_c t \equiv$

let $(x : \text{type } c) = t$

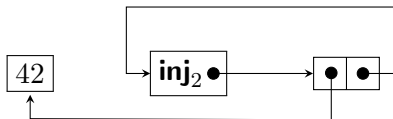
in letref $(y : \text{ref } c) := x$ **in** y

$t, s ::=$	terms
ℓ	location ($\ell \in \mathbb{L}$)
x	identifier
inj $_i^{\tau_1 + \tau_2} t$	sum constructor
$()$	unit
(t, s)	pair
$\lambda x : \tau. t$	function abstraction
match t with $\{\}^\tau$	pattern matching
match t with $\{\text{inj}_1 x_1 \mapsto s_1$ $\text{inj}_2 x_2 \mapsto s_2\}$	
prj $_i t$	projections
$t s$	function application
$t := s$	assignment
! t	dereferencing
letref $(x_1 : \text{ref } c_1) := v_1,$ $\vdots$ $(x_n : \text{ref } c_n) := v_n$	allocation
in t	

A call-by-value calculus: λ_{ref} (allocation syntax)

Example

```
letref (payload      : ref data      ) := 42,  
      (cyclic_list: ref linked_list) := inj2head,  
      (head       : ref list_cell  ) := (payload, cyclic_list)  
in cyclic_list
```



A call-by-value calculus: λ_{ref} (type system)

Typing judgements $\boxed{\Gamma \vdash_w t : \tau}$

Type contexts: $\Gamma = x_1 : \tau_1, \dots, x_n : \tau_n$

Heap layouts: $w = \{\ell_1 : c_1, \dots, \ell_n : c_n\}$

$\underline{w} \stackrel{\text{def}}{=} \{\ell_1, \dots, \ell_n\}$

$$\frac{(\ell : c) \in w}{\Gamma \vdash_w \ell : \mathbf{ref} \ c}$$

$$\frac{\Gamma \vdash_w t : \mathbf{ref} \ c \quad \Gamma \vdash_w s : \mathbf{type} \ c}{\Gamma \vdash_w t := s : \mathbf{1}}$$

$$\frac{\Gamma \vdash_w t : \mathbf{ref} \ c}{\Gamma \vdash_w !t : \mathbf{type} \ c}$$

for $i = 1, \dots, n$: $\Gamma, x_1 : \mathbf{ref} \ c_1, \dots, x_n : \mathbf{ref} \ c_n \vdash_w v_i : \mathbf{type} \ c_i$

$$\frac{\Gamma, x_1 : \mathbf{ref} \ c_1, \dots, x_n : \mathbf{ref} \ c_n \vdash_w t : \tau}{\Gamma \vdash_w \mathbf{letref}(x_1 : \mathbf{ref} \ c_1) := v_1,$$

$\vdots$

$(x_n : \mathbf{ref} \ c_n) := v_n \text{ in } t : \tau$

A call-by-value calculus: λ_{ref} (operational semantics)

$$\eta = \langle v_\ell \rangle_{\ell \in \underline{w}_\eta}$$

$$\overline{\langle \ell, \eta \rangle \Downarrow \langle \ell, \eta \rangle}$$

$$\frac{\langle t, \eta \rangle \Downarrow \langle \lambda x : \tau.t', \eta_1 \rangle \quad \langle s, \eta_1 \rangle \Downarrow \langle v', \eta_2 \rangle \quad \langle t'[x \mapsto v'], \eta_2 \rangle \Downarrow \langle v, \eta' \rangle}{\langle t \ s, \eta \rangle \Downarrow \langle v, \eta' \rangle}$$

$$\frac{\langle t, \eta \rangle \Downarrow \langle v, \eta' \rangle}{\langle \text{inj}_i^{\tau_1 + \tau_2} t, \eta \rangle \Downarrow \langle \text{inj}_i^{\tau_1 + \tau_2} v, \eta' \rangle}$$

$$\overline{\langle (), \eta \rangle \Downarrow \langle (), \eta \rangle}$$

$$\frac{\langle t, \eta \rangle \Downarrow \langle \ell, \hat{\eta} \rangle \quad \langle s, \hat{\eta} \rangle \Downarrow \langle v, \eta' \rangle}{\langle t := s, \eta \rangle \Downarrow \langle (), \eta'[\ell \mapsto v] \rangle} (\ell \in \underline{w}_{\eta'})$$

$$\frac{\langle t_1, \eta \rangle \Downarrow \langle v_1, \hat{\eta} \rangle \quad \langle t_2, \hat{\eta} \rangle \Downarrow \langle v_2, \eta' \rangle}{\langle (t_1, t_2), \eta \rangle \Downarrow \langle (v_1, v_2), \eta' \rangle}$$

$$\frac{\langle t, \eta \rangle \Downarrow \langle \ell, \eta' \rangle}{\langle !t, \eta \rangle \Downarrow \langle \eta'(\ell), \eta' \rangle} (\ell \in \underline{w}_{\eta'})$$

$$\overline{\langle \lambda x : \tau.t, \eta \rangle \Downarrow \langle \lambda x : \tau.t, \eta \rangle}$$

$$\langle t, \eta \rangle \Downarrow \langle \text{inj}_i^{\tau_1 + \tau_2} \hat{v}, \hat{\eta} \rangle$$

$$\langle s_i[x_i \mapsto \hat{v}], \hat{\eta} \rangle \Downarrow \langle v, \eta' \rangle$$

$$\frac{\text{match } t \text{ with}}{\langle \text{inj}_1 x_1 \mapsto s_1, \eta \rangle \Downarrow \langle v, \eta' \rangle}$$

$$\langle \text{inj}_2 x_2 \mapsto s_2 \rangle$$

$$\frac{\langle t, \eta \rangle \Downarrow \langle (v_1, v_2), \eta' \rangle}{\langle \text{prj}_i t, \eta \rangle \Downarrow \langle v_i, \eta' \rangle}$$

$$\frac{\langle t[\theta], \eta \left[\ell_i \mapsto v_i[\theta] \right]_{i=1}^n \rangle \Downarrow \langle v, \eta' \rangle}{\text{letref} \quad \text{where } \theta \stackrel{\text{def}}{=} [x_i \mapsto \ell_i]_{i=1}^n} (\# \underline{w}_\eta \langle \ell_1, \dots, \ell_n \rangle)$$

$$\langle \begin{array}{l} (x_1 : \text{ref } c_1) := v_1, \\ \vdots \\ (x_n : \text{ref } c_n) := v_n \end{array} \text{ in } t, \eta \rangle \Downarrow \langle v, \eta' \rangle$$

A call-by-value calculus: λ_{ref} (meta-theory)

Functoriality of types

If $w \leq w'$ as partial functions, then $\Gamma \vdash_w t : \tau \implies \Gamma \vdash_{w'} t : \tau$. So

$$\begin{array}{lll} \tau^v w \subseteq \tau^v w' & \text{and} & \tau^t w \subseteq \tau^t w' \quad \text{where} \\ \tau^v w \stackrel{\text{def}}{=} \{v \mid \vdash_w v : \tau\} & \text{and} & \tau^t w \stackrel{\text{def}}{=} \{t \mid \vdash_w t : \tau\} \end{array}$$

Non-functoriality of heaps

For $\mathbb{H}w := \{\eta \mid \underline{w}_\eta = \underline{w}, \forall (\ell : c) \in w. \vdash_w \eta(\ell) : \mathbf{type} \ c\}$ and $w \leq w'$ we **lack** functions

$$\mathbb{H}w \rightarrow \mathbb{H}w' \quad \text{and} \quad \mathbb{H}w \leftarrow \mathbb{H}w'$$

A call-by-value calculus: λ_{ref} (meta-theory)

Non-functoriality of heaps

For $\mathbb{H}w := \{\eta \mid \underline{w}_\eta = \underline{w}, \forall (\ell : c) \in w. \vdash_w \eta(\ell) : \mathbf{type} \ c\}$ and $w \leq w'$ we **lack** functions

$$\mathbb{H}w \rightarrow \mathbb{H}w' \quad \text{and} \quad \mathbb{H}w \leftarrow \mathbb{H}w'$$

as we lack **initialisation data** and want to avoid **dangling references**:

$$\begin{aligned} \langle \rangle \in \mathbb{H}\emptyset &\mapsto ? \in \mathbb{H} \{\ell : \mathbf{ptr}\} \\ ? \in \mathbb{H} \{\ell_1 : \mathbf{ptr}\} &\leftarrow \left\langle \begin{array}{l} \ell_1 \mapsto \ell_2, \\ \ell_2 \mapsto \ell_1 \end{array} \right\rangle \in \mathbb{H} \{\ell_1, \ell_2 : \mathbf{ptr}\} \end{aligned}$$

for **type** $\mathbf{ptr} = \mathbf{ref} \ \mathbf{ptr}$.

A call-by-value calculus: λ_{ref} (meta-theory)

Theorem (totality)

All well-typed closed programs fully evaluate:

$$\begin{aligned} \forall \vdash_w t : \tau, w_1 \geq w, \eta_1 \in \mathbb{H}w_1. \\ \implies \exists w_2 \geq w_1, \vdash_{w_2} v : \tau, \eta_2 \in \mathbb{H}w_2. \langle t, \eta_1 \rangle \Downarrow \langle v, \eta_2 \rangle \end{aligned}$$

Proof highlights

Use Tait's method with Kripke logical predicates, and a w -indexed predicate on $\mathbb{H}w$ that is **not** Kripke.

Allocation case: initialisation data $\vdash_{w \oplus \{\ell_1 : c_1, \dots, \ell_n : c_n\}} v_1, \dots, v_n$ needs to extend $\eta \in \mathbb{H}w_1, w_1 \geq w$:

$$\begin{array}{ccc} w & \leq & w \oplus \{\ell_1 : c_1, \dots, \ell_n : c_n\} \\ \mid \wedge & & \mid \wedge \\ w' & \leq & w' \oplus \{\ell_1 : c_1, \dots, \ell_n : c_n\} \end{array}$$

Talk structure

- ▶ Full ground storage
- ▶ **Worlds and initialisations**
- ▶ A monad for hiding/encapsulation
- ▶ The monad, explicitly
- ▶ Evaluation
- ▶ Conclusion

Aspiration

Computational motivation alongside technical development

The category $\mathbb{W}$

Objects: layouts w

Morphisms $\rho : w \rightarrow w'$: label preserving injections $\rho : \underline{w} \rightarrow \underline{w'}$

Interpreting full ground types

Set $\mathbf{V} := [\mathbb{W}, \mathbf{Set}]$ and define:

$$\llbracket \mathbf{0} \rrbracket \stackrel{\text{def}}{=} 0 \qquad \llbracket \gamma_1 + \gamma_2 \rrbracket \stackrel{\text{def}}{=} \llbracket \gamma_1 \rrbracket + \llbracket \gamma_2 \rrbracket$$

$$\llbracket \mathbf{1} \rrbracket \stackrel{\text{def}}{=} 1 \qquad \llbracket \gamma_1 \times \gamma_2 \rrbracket \stackrel{\text{def}}{=} \llbracket \gamma_1 \rrbracket \times \llbracket \gamma_2 \rrbracket$$

$$\llbracket \mathbf{ref} \ c \rrbracket w \stackrel{\text{def}}{=} \{ \ell \in w \mid w(\ell) = c \} \qquad \llbracket \mathbf{ref} \ c \rrbracket \rho(\ell) \stackrel{\text{def}}{=} \rho(\ell)$$

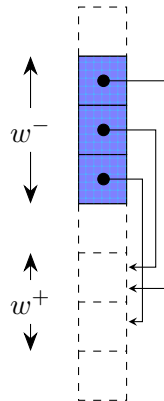
Store fragments

Define $\underline{\$} : \mathbb{W}^{\text{op}} \times \mathbb{W} \rightarrow \mathbf{Set}$:

$$\underline{\$}(w^-, w^+) \stackrel{\text{def}}{=} \prod_{(\ell:c) \in w^-} \llbracket \mathbf{type} \ c \rrbracket w^+$$

Stores

$$\$w \stackrel{\text{def}}{=} \underline{\$}(w, w)$$



Combining store fragments

Independent coproducts [Simpson, unpublished]

Given $\# : \mathbb{N} \xrightarrow{\cong} \mathbb{L}$, define:

$$\#w \stackrel{\text{def}}{=} \min\{n \in \mathbb{N} \mid \forall i \geq n. \#i \notin w\}$$

$$\underline{w_1} \oplus \underline{w_2} \stackrel{\text{def}}{=} \underline{w_1} \cup \{\#(\#w + n) \mid \#n \in w_2\}$$

$$\iota_i^\oplus : w_i \rightarrow \underline{w_1} \oplus \underline{w_2}$$

$$\iota_1^\oplus : \ell_1 \mapsto \ell_1$$

$$\iota_2^\oplus : \#n \mapsto \#(\#w + n)$$

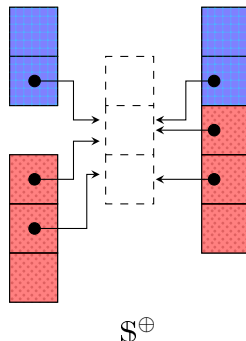
$$(w_1 \oplus w_2)(\iota_i^\oplus \ell) \stackrel{\text{def}}{=} w_i(\ell)$$

and we have: $w_1 \xrightarrow{\iota_1^\oplus} w_1 \oplus w_2 \xleftarrow{\iota_2^\oplus} w_2$ in $\mathbb{W}$

Canonical isomorphisms

$$\mathbb{S}^\oplus : \underline{\mathbb{S}}(w_1, w) \times \underline{\mathbb{S}}(w_2, w) \xrightarrow{\cong} \underline{\mathbb{S}}(w_1 \oplus w_2, w)$$

$$\mathbb{S}^\emptyset : \mathbb{1} \xrightarrow{\cong} \underline{\mathbb{S}}(\emptyset, w)$$



Initialisations

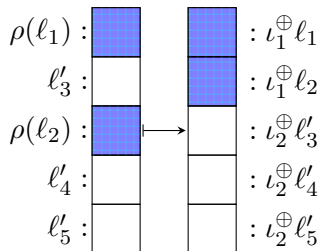
Complements

Given $\rho : w_1 \rightarrow w_2$, define:

$$\underline{w_2 \ominus \rho} \stackrel{\text{def}}{=} \underline{w_2} \setminus \text{Im}(\rho)$$

$$w_2 \ominus \rho(\ell) \stackrel{\text{def}}{=} w_2(\ell)$$

$$\begin{aligned} \rho^{\complement} : w_2 \ominus \rho &\rightarrow w_2 \\ \ell &\mapsto \ell \end{aligned}$$

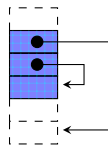


$$\underline{\mathbb{S}}(w_2, w) \xrightarrow{\underline{\mathbb{S}}(\cong, w)} \underline{\mathbb{S}}(w_1 \oplus (w_2 \ominus \rho), w)$$

The category $\mathbb{E}$ of initialisations

Objects: layouts w

Morphisms $\varepsilon : w \rightarrow w'$: injections $u\varepsilon : w \rightarrow w'$ and
initialisation data $\sigma_\varepsilon \in \underline{\mathbb{S}}(w' \ominus u\varepsilon, w')$

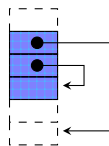


Initialisations

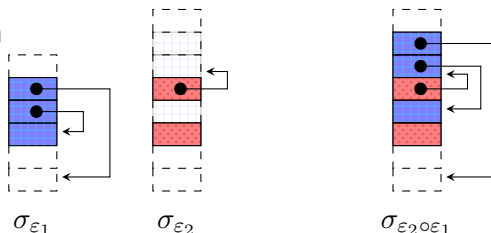
The category $\mathbb{E}$ of initialisations

Objects: layouts w

Morphisms $\varepsilon : w \rightarrow w'$: injections $u\varepsilon : w \rightarrow w'$ and
initialisation data $\sigma_\varepsilon \in \underline{\mathbb{S}}(w' \ominus u\varepsilon, w')$



Composition



Functoriality of stores

$$\mathbb{S}w = \underline{\mathbb{S}}(w, w) \cong \mathbb{E}(\emptyset, w)$$

so $\mathbb{S}$ is a representable functor in $\mathbf{C} \stackrel{\text{def}}{=} [\mathbb{E}, \mathbf{Set}]$

Talk structure

- ▶ Full ground storage
- ▶ Worlds and initialisations
- ▶ A monad for hiding/encapsulation
- ▶ The monad, explicitly
- ▶ Evaluation
- ▶ Conclusion

Aspiration

Computational motivation alongside technical development

The hiding monad

Comma category

For the forgetful $u : \mathbb{E} \rightarrow \mathbb{W}$, the comma $w \downarrow u$ is:

Objects: $\mathbb{W}$ -morphisms $\rho : w \rightarrow w'$

Morphisms $(\rho_1 \downarrow_{w_1}^w) \xrightarrow{\varepsilon} (\rho_2 \downarrow_{w_2}^w)$: Initialisations $w'_1 \xrightarrow{\varepsilon} w'_2$ s.t. $w \begin{matrix} \xrightarrow{\rho_1} w'_1 \\ \xrightarrow{\rho_2} w'_2 \end{matrix} \begin{matrix} \downarrow u\varepsilon \\ \end{matrix} w_2$

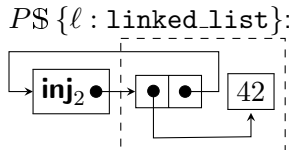
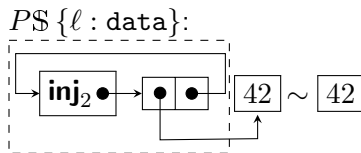
Object map

Define for $A \in \mathbf{C} = [\mathbb{E}, \mathbf{Set}]$:

$$PAw \stackrel{\text{def}}{=} \int^{w \rightarrow w' \in w \downarrow u} A$$

$$\stackrel{\text{def}}{=} \left(\sum_{w \rightarrow w' \in w \downarrow u} Aw' \right) / \sim$$

In $[\rho : w \rightarrow w', x] \in PAw$ the locations in $w' \ominus \rho$ are private to x



The hiding monad

Comma category

For the forgetful $u : \mathbb{E} \rightarrow \mathbb{W}$, the comma $w \downarrow u$ is:

Objects: $\mathbb{W}$ -morphisms $\rho : w \rightarrow w'$

Morphisms $(\rho_1 \downarrow_{w_1}^w) \xrightarrow{\varepsilon} (\rho_2 \downarrow_{w_2}^w)$: Initialisations $w'_1 \xrightarrow{\varepsilon} w'_2$ s.t.

A commutative triangle diagram representing the comma category. On the left is the object w . On the top right is the object w_1 . On the bottom right is the object w_2 . An arrow labeled ρ_1 points from w to w_1 . An arrow labeled ρ_2 points from w to w_2 . An arrow labeled $u\varepsilon$ points from w_1 to w_2 . An equals sign $=$ is placed between the two arrows originating from w , indicating the commutativity of the triangle.

Object map

Define for $A \in \mathbf{C} = [\mathbb{E}, \mathbf{Set}]$:

$$PAw \stackrel{\text{def}}{=} \int^{w \rightarrow w' \in w \downarrow u} A$$
$$\stackrel{\text{def}}{=} \left(\sum_{w \rightarrow w' \in w \downarrow u} Aw' \right) / \sim$$

In $[\rho : w \rightarrow w', x] \in PAw$ the locations in $w' \ominus \rho$ are private to x

Subtleties [standard]

- ▶ $PAw = \text{Colim}_{w \rightarrow w' \in w \downarrow u} Aw'$
convenient coend notation
- ▶ The injections $\rho : w \rightarrow w'$
define the quotient

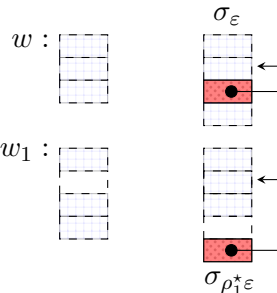
The initialisation $\rho^\star \varepsilon$

Local independent coproducts

Given $w_1 \xleftarrow{\rho_1} w \xrightarrow{\rho_2} w_2$, define:

$$\rho_1 \oplus_w \rho_2 := w \oplus (w_1 \ominus \rho_1) \oplus (w_2 \ominus \rho_2)$$

$$\begin{array}{ccc} w & \xrightarrow{\rho_2} & w_2 \\ \rho_1 \downarrow & = & \downarrow \rho_1^\star \rho_2 \\ w_1 & \xrightarrow{\rho_1^\star \rho_2} & \rho_1 \oplus_w \rho_2 \end{array}$$



when $\rho_2 = u\varepsilon$ there is an initialisation $\rho_1^\star \varepsilon : w_1 \rightarrow \rho_1 \oplus_w \rho_2$ with $u\rho_1^\star \varepsilon = \rho_1^\star u\varepsilon$.

Semantic counterpart of

$$\begin{array}{c} w \leq w \oplus \{\ell_1 : c_1, \dots, \ell_n : c_n\} \\ | \wedge \qquad \qquad \qquad | \wedge \\ w' \leq w' \oplus \{\ell_1 : c_1, \dots, \ell_n : c_n\} \end{array}$$

The hiding monad

Object map

Define for $A \in \mathbf{C} = [\mathbb{E}, \mathbf{Set}]$:

$$PAw \stackrel{\text{def}}{=} \int^{w \rightarrow w' \in w \downarrow u} A$$

$$\stackrel{\text{def}}{=} \left(\sum_{w \rightarrow w' \in w \downarrow u} Aw' \right) / \sim$$

Functorial action

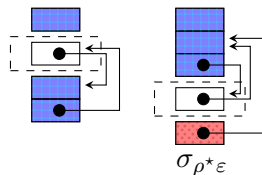
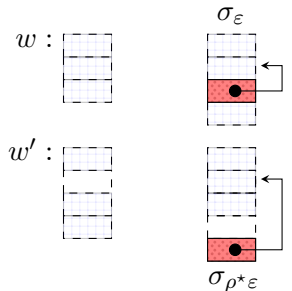
For $\varepsilon : w_1 \rightarrow w_2$ in $\mathbb{E}$:

$$PA\varepsilon : PAw_1 \rightarrow PAw_2$$

$$[\rho : w_1 \rightarrow w', a] \mapsto [\varepsilon^* \rho, A(\rho^* \varepsilon)(a)]$$

where

$$\begin{array}{ccc}
 w_1 & \xrightarrow{\varepsilon} & w_2 \\
 \rho \downarrow & = & \downarrow \varepsilon^* \rho \\
 w' & \xrightarrow{\rho^* \varepsilon} & \rho \oplus_w u \varepsilon
 \end{array}$$



$$P\$w_1 \xrightarrow{P\$ \varepsilon} P\$w_2$$

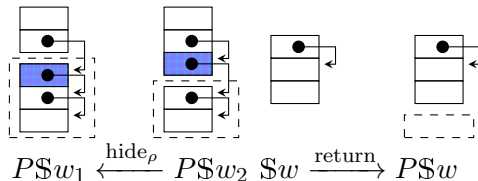
The hiding monad

Object map

Define for $A \in \mathbf{C} = [\mathbb{E}, \mathbf{Set}]$:

$$PAw \stackrel{\text{def}}{=} \int^{w \rightarrow w' \in w \downarrow u} A$$

$$\stackrel{\text{def}}{=} \left(\sum_{w \rightarrow w' \in w \downarrow u} Aw' \right) / \sim$$



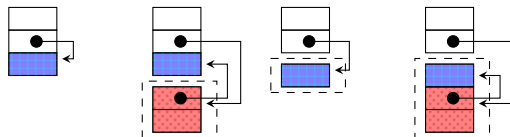
Hiding

For $\rho : w_1 \rightarrow w_2$ we have

$$\text{hide}_\rho : PAw_1 \leftarrow PAw_2$$

Return

$$\text{return}_w a \stackrel{\text{def}}{=} [\text{id}_w, a]$$



Bind

For $g : A \rightarrow PB$, define:

$$([\rho : w \rightarrow w', a] \gg g) \stackrel{\text{def}}{=} \text{hide}_\rho(g_{w'}(a))$$

A state transformer

Enrichment

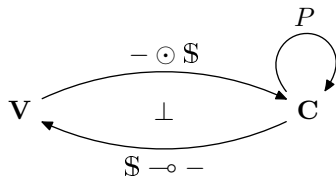
$\mathbf{C} = [\mathbb{E}, \mathbf{Set}]$ is enriched over $\mathbf{V} = [\mathbb{W}, \mathbf{Set}]$ with tensors:

$$X \odot A \stackrel{\text{def}}{=} (X \circ u) \times A$$
$$(A \multimap B)w \stackrel{\text{def}}{=} \int_{w \rightarrow w' \in w \downarrow u} Aw' \Rightarrow (Bw')$$

(directly/as a $\mathbf{V}$ -actegory [Gordon-Power'97, Janelidze-Kelly'01])

A monad for full ground storage

Take $T \stackrel{\text{def}}{=} \$ \multimap P(- \odot \$)$



Talk structure

- ▶ Full ground storage
- ▶ Worlds and initialisations
- ▶ A monad for hiding/encapsulation
- ▶ The monad, explicitly
- ▶ Evaluation
- ▶ Conclusion

Aspiration

Computational motivation alongside technical development

The full ground storage monad

Outline

For every $X \in \mathbf{V} = [\mathbb{W}, \mathbf{Set}]$:

$$TXw \subseteq \prod_{w \rightarrow w' \in \mathbb{W}} \$w' \Rightarrow (\sum_{w' \rightarrow w'' \in \mathbb{W}} Xw'' \times \$w'') / \sim$$

The monad

$$(TX)w = \int_{w \rightarrow w' \in w \downarrow u} \$w' \Rightarrow \left(\int_{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \$w'' \right)$$

Subtleties

- ▶ $\int_{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \w'' is contravariant in w' , but the end is taken w.r.t. the covariant action of P
- ▶ The functorial action $TX\rho$ is given via the end

Analogous to $T_{\mathbf{ground}}X\rho$ in the ground case [cf. Plotkin-Power'02]:

$$T_{\mathbf{ground}}Xw = \$_{\mathbf{ground}}w \Rightarrow \int^{w \rightarrow w' \in w / \mathbb{W}} X \times \$_{\mathbf{ground}}$$

The full ground storage monad

Outline

For every $X \in \mathbf{V} = [\mathbb{W}, \mathbf{Set}]$:

$$TXw \subseteq \prod_{w \rightarrow w' \in \mathbb{W}} \$w' \Rightarrow (\sum_{w' \rightarrow w'' \in \mathbb{W}} Xw'' \times \$w'') / \sim$$

The monad

$$(TX)w = \int_{w \rightarrow w' \in w \downarrow u} \$w' \Rightarrow \left(\int_{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \$w'' \right)$$

Return

$$(\pi_{\rho:w \rightarrow w'} \circ \text{return } x)(\sigma) = [\text{id}_{w'}, (X\rho)x, \sigma]$$

The full ground storage monad

Outline

For every $X \in \mathbf{V} = [\mathbb{W}, \mathbf{Set}]$:

$$TXw \subseteq \prod_{w \rightarrow w' \in \mathbb{W}} \$w' \Rightarrow (\sum_{w' \rightarrow w'' \in \mathbb{W}} Xw'' \times \$w'') / \sim$$

The monad

$$(TX)w = \int_{w \rightarrow w' \in w \downarrow u} \$w' \Rightarrow \left(\int_{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \$w'' \right)$$

Bind

Given $f : X \rightarrow TY$ in $\mathbf{V}$ and $\alpha \in TX$,

$$(\pi_{\rho:w \rightarrow w'} \alpha \gg f)(\sigma') = [\rho'' \circ \rho', y, \sigma''']$$

where

$$\begin{aligned} (\pi_{\rho} \alpha) \sigma' &= [\rho' : w' \rightarrow w'', x, \sigma''] \\ (\pi_{\text{id}_{w''}} \circ f_{w''}(x))(\sigma'') &= [\rho'' : w'' \rightarrow w''', y, \sigma'''] \end{aligned}$$

The full ground storage monad

Outline

For every $X \in \mathbf{V} = [\mathbb{W}, \mathbf{Set}]$:

$$TXw \subseteq \prod_{w \rightarrow w' \in \mathbb{W}} \$w' \Rightarrow (\sum_{w' \rightarrow w'' \in \mathbb{W}} Xw'' \times \$w'') / \sim$$

The monad

$$(TX)w = \int_{w \rightarrow w' \in w \downarrow u} \$w' \Rightarrow \left(\int_{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \$w'' \right)$$

State manipulation

$$\text{get}_c : \llbracket \mathbf{ref} \ c \rrbracket \rightarrow T \llbracket \mathbf{type} \ c \rrbracket$$

$$(\pi_\rho \circ \text{get}_c(\ell))(\sigma) = [\text{id}_{w'}, \sigma(\rho(\ell)), \sigma]$$

$$\text{set}_c : \llbracket \mathbf{ref} \ c \rrbracket \times \llbracket \mathbf{type} \ c \rrbracket \rightarrow T\mathbb{1}$$

$$(\pi_\rho \circ \text{set}_c(\ell, a))(\sigma) = [\text{id}_{w'}, \star, \sigma[\rho(\ell) \mapsto \llbracket \mathbf{type} \ c \rrbracket \rho(a)]]$$

where $\sigma(\ell) \stackrel{\text{def}}{=} \pi_\ell \sigma$, and

$$\sigma[\ell \mapsto x](\ell') = \begin{cases} x & \ell' = \ell \\ \sigma(\ell') & \text{otherwise} \end{cases}$$

The full ground storage monad

Outline

For every $X \in \mathbf{V} = [\mathbb{W}, \mathbf{Set}]$:

$$TXw \subseteq \prod_{w \rightarrow w' \in \mathbb{W}} \$w' \Rightarrow (\sum_{w' \rightarrow w'' \in \mathbb{W}} Xw'' \times \$w'') / \sim$$

The monad

$$(TX)w = \int_{w \rightarrow w' \in w \downarrow u} \$w' \Rightarrow \left(\int^{w' \rightarrow w'' \in w \downarrow u} X \circ uw'' \times \$w'' \right)$$

Allocation

Using

$$\begin{aligned} \partial_{w_0} : \mathbf{V} &\rightarrow \mathbf{V} & \text{init}_{w_0} : \prod_{(\ell:c) \in w_0} \partial_{w_0} [\mathbf{type } c] &\rightarrow \mathbb{E}(w, w \oplus w_0) \\ X &\mapsto X(- \oplus w_0) \end{aligned}$$

define

$$\begin{aligned} \text{new}_{w_0} : \prod_{(\ell:c) \in w_0} \partial_{w_0} [\mathbf{type } c] &\rightarrow T \prod_{(\ell:c) \in w_0} [\mathbf{ref } c] \\ (\pi_\rho \circ \text{new}_{w_0}(\langle a_\ell \rangle))\sigma &= [\varepsilon^* \rho, (\langle \varepsilon^* \rho(\ell) \rangle)_{\ell \in w_0}, \$(\rho^* \varepsilon)(\sigma)] \end{aligned}$$

where $\varepsilon \stackrel{\text{def}}{=} \text{init } \langle a_\ell \rangle$.

Talk structure

- ▶ Full ground storage
- ▶ Worlds and initialisations
- ▶ A monad for hiding/encapsulation
- ▶ The monad, explicitly
- ▶ **Evaluation**
- ▶ Conclusion

Aspiration

Computational motivation alongside technical development

Garbage collection

Constant functors

Functors $X \in \mathbf{V}$ whose action $X\rho$ is a bijection.

Theorem (effect masking)

For every pair of constant functors $\Gamma, X \in \mathbf{V}$, every morphism $f : \Gamma \rightarrow TX$ factors uniquely through the monadic unit:

$$\begin{array}{ccc} \Gamma & \xrightarrow{f} & TX \\ & \searrow \text{runST } f & \nearrow \text{return}^T \\ & X & \end{array} \quad \begin{array}{c} \\ \\ = \end{array}$$

Interprets a multi-monadic-metalanguage with a **runST** construct [Launchbury-Peyton Jones'94]

Garbage collection

Proof.

$$\begin{array}{ll} \mathbb{1} \longrightarrow \$ \multimap P(X \odot \$) & \text{in } \mathbf{V} \\ \hline \$ \longrightarrow P(X \odot \$) & \text{in } \mathbf{C} \\ \hline \$ \longrightarrow X \odot P\$ & \text{by lemma} \\ \hline \mathbb{E}(\underline{0}, -) \longrightarrow X \odot P\$ & \text{in } \mathbf{C} \\ \hline \mathbb{1} \longrightarrow (X \odot P\$)\underline{0} & \text{in } \mathbf{Set}, \text{ by Yoneda} \\ \hline \mathbb{1} \longrightarrow X\underline{0} & \text{by lemma} \\ \hline \mathbb{1} \longrightarrow X & \text{in } \mathbf{V} \end{array}$$

and chase a generic morphism upwards.



A call-by-value calculus: λ_{ref} (denotational semantics)

With a standard denotational semantics we have:

Theorem (adequacy)

For all terms $\Gamma \vdash_w t_1, t_2 : \tau$, if $\llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket$ then $\Gamma \vdash_w t_1 \simeq_{\text{ctx}} t_2 : \tau$.

Program transformations

The semantics validates the Hilbert-Post complete program equations for local ground state [Staton'10], which can be divided into three groups [*loc.cit.*, Melliés'14]:

- ▶ Global state equations [Plotkin-Power'02]
- ▶ Block algebra equations (allocation only)
- ▶ Local state equations (state-allocation interaction)

Conclusions

- ▶ A monad for full ground storage
- ▶ Heaps/stores functoriality over initialisations
- ▶ Decomposition into state-transformed encapsulation monad
- ▶ Evaluation: effect masking theorem and local ground state equations

Further work

- ▶ Haskell's **runST** and effect masking
- ▶ A denotational account of the value restriction [Wright'95]
- ▶ General storage
- ▶ Algebraic presentations, Hilbert-Post completeness, handlers
- ▶ Independence structures
- ▶ Combining effects
- ▶ Further analysis
- ▶ Garbage collectors and mallocators
- ▶ Comparison with ground case
- ▶ Nominal accounts