Functional models of full ground, and general, reference cells

Work in Progress

Ohad Kammar
<ohad.kammar@cs.ox.ac.uk>
joint work with
Sean Moss

The 5th ACM SIGPLAN Workshop on
Higher-Order Programming with Effects
18 September 2016

Kinds of local state

Semantic complications with dynamic allocation of arbitrary type:

- Locality: freshness of newly allocated cell.
- Non-ground: stored values can manipulate the memory store. E.g. **ref** (**bool** $\to$**bool**).
- Full storage: stored values may depend on store shape. E.g., inhabitants **ref** (**ref bool**) require inhabitants of **ref bool** .

This talk: full ground local state.

Success stories

- Operational semantics
- Step-indexing [Birkedal et al.'10 etc]
- Strategies over games

Goal

Denotational semantics for full ground state

- Sets with structure and structure preserving functions.
- Monadic (or adjunctive, following Levy'02).
- Extensional.

Applications

- Validation of compiler transformations.
- Analysis of ML's value restriction.
- Semantic correctness of Haskell's **runST** .

This talk

~~Contribution~~ Tutorial and discussion

- General setting.
- Effect masking.
- Monads (not-quite) for full ground references.
- Denotational semantics for Haskell's **runST** .

Ground types [Levy'04,Murawski and Tzevelekos'12]

Ground types

Parameterised by a pair $\langle \mathbb{C}, \mathit{Type} \rangle$, where

- $\mathbb{C}$ — a countable set of storable type names C;
- $\mathit{Type} : \mathbb{C} \to \mathbb{G}$ function

where the set $\mathbb{G}$ of ground types is:

$$G ::= 0 \mid G_1 + G_2 \mid 1 \mid G_1 \times G_2 \mid \mathbf{ref}\ C$$

Rationale

We can include circular data structures without complicating the semantics further.

For example, taking $\mathbb{C} := \{\mathbf{linked\_list}\}$, and:

$$\mathit{Type}(\mathbf{linked\_list}) := 1 + (\mathbf{bool}\ \times \mathbf{ref}\ \mathbf{linked\_list})$$

Worlds

The category $\mathbb{W}$

- Worlds w consist of:
  - $\underline{w} = \{0, \ldots, w - 1\}$ a finite ordinal; and
  - a function $w : \underline{w} \to \mathbb{C}$

  For example with $C = \{ \textbf{int}, \textbf{linked\_list} \}$ and

$$w = \{0 : \textbf{int}, 1 : \textbf{linked\_list}, 2 : \textbf{int}\}$$
$$w' = \{0 : \textbf{linked\_list}, 1 : \textbf{int}, 2 : \textbf{int}, 3 : \textbf{int}\}$$

- Morphism $\rho : w \to w'$ are type-name-preserving injections
  - $\rho : \underline{w} \rightarrowtail \underline{w}'$, such that:
  - for all $\ell \in \underline{w}$, we have $w'(\rho(\ell)) = w$.

  In the example above, $\rho : w \to w'$:

$$\rho(i) := (i - 1) \bmod 4$$

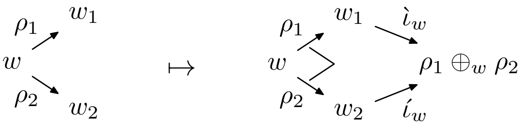
Independence structure

$\mathbb{W}$ has a monoidal structure given by ordinal addition and relabelling:

$$\underline{w_1 \oplus w_2} := \underline{|w_1| + |w_2|}$$

$$w_1 \oplus w_2(\ell) := \begin{cases} w_1(\ell_1) & \ell = \ell_1 \in \underline{w_1} \\ w_2(\ell_2) & \ell = |\underline{w_1}| + \ell_2 \end{cases}$$

And its coslices $w/\mathbb{W}$ have monoidal structure:



whose action on the ordinals is given by:

$$\underline{\rho_1 \oplus_w \rho_2} := \underline{|w_1| + |w_2| - |w|}$$

General setting

The functor category $\mathcal{V} := [\mathbb{W}, \mathbf{Set}]$

- Bi-cartesian closed: interpret finite sums, products, and function spaces.
- Interpret ground reference types:

$$[\![\mathbf{ref}\ C]\!]\, w := w^{-1}[C] \quad [\![\mathbf{ref}\ C]\!]\, \rho := \rho|_{w^{-1}[C]}$$

For example with $C = \{\ \mathbf{int}\ , \mathbf{linked\_list}\}$ and

$$w = \{0 :\ \mathbf{int}\ , 1 : \mathbf{linked\_list}, 2 :\ \mathbf{int}\ \}$$

we have

$$[\![\mathbf{ref}\ \ \mathbf{int}\ ]\!]\, w := \{0, 2\}$$

We want a monad $T : \mathcal{V} \to \mathcal{V}$.

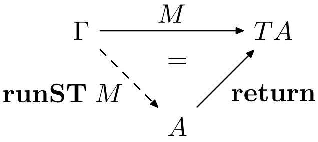
Correctness criteria

Semantics for local state

- Allocation, dereferencing, assignment.
- Usual equations [Levy'08].
- Adequacy.

Effect masking

A monad $T : \mathcal{V} \to \mathcal{V}$ validates effect masking when, for every two constant functors $\Gamma, A : \mathbb{W} \to \textbf{Set}$, every morphism $M : \Gamma \to TA$ factors uniquely:



(Natural, and holds for the ground state monad.)

Two not-quite-right monads

Not enough structure

A store is given by:

$$\mathbb{S}(w', w) := \prod_{\ell \in \underline{w'}} \int^{w/\mathbb{W}} [\![Type(w'(\ell))]\!]$$

with the covariant action given by the independence monoidal structure $\oplus_w$ and the monad is given by:

$$TAw := \mathbb{S}(w, w) \to \int^{w/\mathbb{W}} \mathbb{S} \times A$$

- Analogous to ground case.
- Validates effect masking.
- No obvious interpretation for dereferencing.

Two not-quite-right monads

Too much structure

A store is given by:

$$\mathbb{S}(w', w) := \prod_{\ell \in \underline{w'}} [\![Type(w'(\ell))]\!]\, w$$

and the monad is given by:

$$TAw := \int_{\rho' : w \to w'} \mathbb{S}(w', w') \to \int^{\rho'' : w \to w''} \mathbb{S}(\rho' \oplus_w \rho'', \rho' \oplus_w \rho'') \times A(\rho' \oplus_w \rho'')$$

- More natural store.
- Explicit use of $\oplus_w$
- Interprets the operations.

- Doesn't validate effect masking.

runST -Haskell syntax

Syntax

$M ::=$	term	
x	variable	
$\mid\ \iota_i^{A_1+A_2} M$	coproducts	deconstructors
$\mid\ ()$	unit value	
$\mid\ \langle M_1, M_2 \rangle$	pairing	
$\mid\ \textbf{absurd}\ M$	empty	deconstructors
$\mid\ \textbf{match}\ M\ \textbf{with}\ \{\iota_1 x \to M_x, \iota_2 y \to M_y\}$	coproducts	
$\mid\ \textbf{match}\ M_1\ \textbf{with}\ ()\ \textbf{in} M_2$	unit type	
$\mid\ \textbf{match}\ M_1\ \textbf{with}\ \langle x, y \rangle\ \textbf{in} M_2$	pairs	
$\mid\ \lambda x.M$	abstraction	
$\mid\ M_1\ M_2$	application	
$\mid\ \textbf{return}\ M$	monadic return	
$\mid\ M_1 \gg= M_2$	monadic bind	
$\mid\ \alpha.\textbf{letref}\ x_1 := M_1, \ldots, x_n := M_n\ \textbf{in}\ M$	allocation [Lev'02]	
$\mid\ !M$	dereferencing	
$\mid\ M_1 := M_2$	assignment	
$\mid\ \textbf{runST}\ M$	runST	

runST -Haskell kinds and types

Syntax

α, β		region variables
$\Delta ::= \alpha_1, \ldots, \alpha_n$		kinds
$A ::=$		types
	G	ground types
	$\mid A_1 + A_2$	coproducts
	$\mid A_1 \times A_2$	products
	$\mid A_1 \to A_2$	functions
	$\mid T_\alpha A$	ST monad
$G ::= 0 \mid G_1 + G_2 \mid 1 \mid G_1 \times G_2 \mid \mathbf{ref}\ C$		ground types

runST -Haskell kind and type system

Kinding judgements $\Delta \vdash A$

$$\frac{\alpha_1, \ldots, \alpha_n \vdash A}{\alpha_1, \ldots, \alpha_n \vdash T_{\alpha_i} A}$$

Typing judgements $\Delta; \Gamma \vdash M : A$

$$\frac{\begin{array}{c}\Delta; \Gamma, x_1 : \mathbf{ref}_\alpha\ C_1, \ldots, x_n : \mathbf{ref}_\alpha\ C_n \vdash M : T_\alpha A \\ \text{for all i: } \Delta; \Gamma, x_1 : \mathbf{ref}_\alpha\ C_1, \ldots, x_n : \mathbf{ref}_\alpha\ C_n \vdash M_i : Type\, C_i\end{array}}{\Delta; \Gamma \vdash \alpha.\mathbf{letref}\ x_1 := M_1, \ldots, x_n := M_n\ \mathbf{in}\ M : T_\alpha A}$$

$$\frac{\Delta; \Gamma \vdash M_1 : T_\alpha \mathbf{ref}_\alpha\ C}{\Delta; \Gamma \vdash M_1 := M_2 : \mathbf{unit}} \qquad \frac{\Delta; \Gamma \vdash M : \mathbf{ref}_\alpha\ C}{\Delta; \Gamma \vdash !M : Type C}$$

$$\frac{\Delta \vdash \Gamma, A \qquad \Delta, \alpha; \Gamma \vdash M : T_\alpha A}{\Delta; \Gamma \vdash \mathbf{runST}\ M : A}$$

runST -Haskell semantics

Kinds denote categories of worlds:

$$[\![\vec{\alpha}]\!] := \prod_{i \in |\vec{\alpha}|} \mathbb{W}$$

Types $\Delta \vdash A$ denote objects in $\mathcal{V}_{[\![\Delta]\!]} := [[\![\Delta]\!], \mathbf{Set}]$. The monad constructor is interpreted by:

$$[\![T_{\alpha_i} A]\!] \langle w_1, \ldots, w_n \rangle := T(A \langle w_1, \ldots, w_{i-1}, -, w_{i+1}, \ldots, w_n \rangle) w_i$$

Terms $\Delta; \Gamma \vdash M : A$ denote $\mathcal{V}_{[\![\Delta]\!]}$ morphisms:

$$[\![M]\!] : [\![\Gamma]\!] \to [\![A]\!]$$

Weakening of a type by α:

$$\frac{\Delta \vdash A}{\Delta, \alpha \vdash A}$$

is interpreted by a presheaf constant in the α-argument, and so the effect masking property allows us to interpret **runST** .

Concluding remarks

- Still work in progress!
- Two monads (not) for local full ground references.
- Effect masking property and its applications.