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 → 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 \( \langle C, Type \rangle \), where

- \( C \) — a countable set of storable type names \( C \);
- \( Type : C \rightarrow G \) function

where the set \( G \) of ground types is:

\[
G ::= 0 \mid G_1 + G_2 \mid 1 \mid G_1 \times G_2 \mid \text{ref } C
\]

Rationale
We can include circular data structures without complicating the semantics further.
For example, taking \( C := \{ \text{linked_list} \} \), and:

\[
Type(\text{linked_list}) := 1 + (\text{bool} \times \text{ref linked_list})
\]
The category $\mathbf{W}$

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

For example with $C = \{\text{int}, \text{linked_list}\}$ and

$$w = \{0 : \text{int}, 1 : \text{linked_list}, 2 : \text{int}\}$$

$$w' = \{0 : \text{linked_list}, 1 : \text{int}, 2 : \text{int}, 3 : \text{int}\}$$

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

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

$$\rho(i) := (i - 1) \mod 4$$
\( \mathbb{W} \) has a monoidal structure given by ordinal addition and relabelling:

\[
\begin{align*}
w_1 \oplus w_2 & := |w_1| + |w_2| \\
w_1 \oplus w_2(\ell) & := \begin{cases} 
    w_1(\ell_1) & \ell = \ell_1 \in w_1 \\
    w_2(\ell_2) & \ell = |w_1| + \ell_2
\end{cases}
\]

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

whose action on the ordinals is given by:

\[
\rho_1 \oplus_w \rho_2 := |w_1| + |w_2| - |w|
\]
The functor category $\mathcal{V} := [\mathcal{W}, \text{Set}]$

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

$$\lbrack \text{ref } C \rbrack w := w^{-1}[C] \quad \lbrack \text{ref } C \rbrack \rho := \rho|_{w^{-1}[C]}$$

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

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

we have

$$\lbrack \text{ref int } \rbrack w := \{0, 2\}$$

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 $\Gamma, A : \mathcal{W} \to \mathbf{Set}$, every morphism $M : \Gamma \to TA$ factors uniquely:

\[
\begin{array}{c}
\Gamma \\
\downarrow^M \\
\dashrightarrow \downarrow \quad = \\
\dashrightarrow \downarrow^\text{runST} M \\
\downarrow^A \\
\end{array}
\]

\[
\begin{array}{c}
\end{array}
\]

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

Ohad Kammar <ohad.kammar@cs.ox.ac.uk>
Two not-quite-right monads

Not enough structure
A store is given by:

\[ S(w', w) := \prod_{\ell \in w'} \int^{w/\text{w}} \left[ \text{Type}(w'(\ell)) \right] \]

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

\[ TA_w := S(w, w) \to \int^{w/\text{w}} S \times A \]

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

Ohad Kammar <ohad.kammar@cs.ox.ac.uk>
Functional reference cells
Two not-quite-right monads

Too much structure
A store is given by:

\[ S(w', w) := \prod_{\ell \in w'} \left[ \text{Type}(w'(\ell)) \right] w \]

and the monad is given by:

\[ TA_w := \int_{\rho': w \to w'} S(w', w') \to \int^{\rho'': w \to w''} S(\rho' \oplus w \rho'', \rho' \oplus w \rho'') \times A(\rho' \oplus w \rho'') \]

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

- Doesn’t validate effect masking.
Syntax

\[ M ::= \]

\( x \) \hspace{1cm} \text{variable}

\( \iota_i^{A_1 + A_2} M \) \hspace{1cm} \text{coproducts deconstructors}

\( () \) \hspace{1cm} \text{unit value}

\( \langle M_1, M_2 \rangle \) \hspace{1cm} \text{pairing}

\textbf{absurd} \hspace{1cm} \text{empty deconstructors}

\textbf{match} \ M \textbf{ with } \{ \iota_1 x \rightarrow M_x, \iota_2 y \rightarrow M_y \} \hspace{1cm} \text{coproducts}

\textbf{match} \ M_1 \textbf{ with } () \textbf{ in } M_2 \hspace{1cm} \text{unit type}

\textbf{match} \ M_1 \textbf{ with } \langle x, y \rangle \textbf{ in } M_2 \hspace{1cm} \text{pairs}

\( \lambda x. M \) \hspace{1cm} \text{abstraction}

\( M_1 \ M_2 \) \hspace{1cm} \text{application}

\textbf{return} \ M \hspace{1cm} \text{monadic return}

\( M_1 \gg= M_2 \) \hspace{1cm} \text{monadic bind}

\( \alpha. \textbf{letref } \ x_1 := M_1, \ldots, x_n := M_n \textbf{ in } M \) \hspace{1cm} \text{allocation [Lev'02]}

\( ! M \) \hspace{1cm} \text{dereferencing}

\( M_1 := M_2 \) \hspace{1cm} \text{assignment}

\textbf{runST} \ M \hspace{1cm} \text{runST}
runST - Haskell kinds and types

Syntax

\[ \alpha, \beta \]
\[ \Delta ::= \alpha_1, \ldots, \alpha_n \]
\[ A ::= \\
    G \\
    A_1 + A_2 \\
    A_1 \times A_2 \\
    A_1 \rightarrow A_2 \\
    T_\alpha A \]
\[ G ::= 0 | G_1 + G_2 | 1 | G_1 \times G_2 | \text{ref} \ C \]

region variables
kinds

types
ground types
coproducts
products
functions

ST monad

ground types

Ohad Kammar <ohad.kammar@cs.ox.ac.uk>  Functional reference cells
runST - Haskell kind and type system

Kinding judgements $\Delta \vdash A$

\[
\begin{align*}
\alpha_1, \ldots, \alpha_n & \vdash A \\
\alpha_1, \ldots, \alpha_n & \vdash T_{\alpha_i} A
\end{align*}
\]

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

\[
\begin{align*}
\sum_{\alpha} \Delta; \Gamma, x_1 : \text{ref}_\alpha C_1, \ldots, x_n : \text{ref}_\alpha C_n & \vdash M : T_{\alpha} A \\
\text{for all } i & : \sum_{\alpha} \Delta; \Gamma, x_1 : \text{ref}_\alpha C_1, \ldots, x_n : \text{ref}_\alpha C_n & \vdash M_i : \text{TypeC}_i
\end{align*}
\]

\[
\Delta; \Gamma \vdash \alpha. \text{letref } x_1 := M_1, \ldots, x_n := M_n \text{ in } M : T_{\alpha} A
\]

\[
\Delta; \Gamma \vdash M_1 : T_{\alpha} \text{ref}_\alpha C
\]

\[
\Delta; \Gamma \vdash M_1 := M_2 : \text{unit}
\]

\[
\Delta; \Gamma \vdash M : \text{ref}_\alpha C
\]

\[
\Delta; \Gamma \vdash !M : \text{TypeC}
\]

\[
\begin{align*}
\Delta \vdash \Gamma, A & \\
\Delta, \alpha; \Gamma \vdash M : T_{\alpha} A
\end{align*}
\]

\[
\Delta; \Gamma \vdash \text{runST } M : A
\]
Kinds denote categories of worlds:

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

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

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

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

\[ [M] : [\Gamma] \rightarrow [A] \]

Weakening of a type by \( \alpha \):

\[
\frac{\Delta \vdash A}{\Delta, \alpha \vdash A}
\]

is interpreted by a presheaf constant in the \( \alpha \)-argument, and so the effect masking property allows us to interpret \( \text{runST} \).
Still work in progress!
- Two monads (not) for local full ground references.
- Effect masking property and its applications.