Heaps denote finitely partitioned forests

Paul Blain Levy

University of Birmingham

July 13, 2018

Outline

References

- A reference is a memory address storing a value.
  It may be read or updated.
- ML-like languages allows generation of references.

References

- A reference is a memory address storing a value.
  It may be read or updated.
- ML-like languages allows generation of references.
- ML provides general references; they may store values of any type, even functions and thunks.
- Sometimes we study languages that allow only ground references, i.e. references to integers or booleans.
- Sometimes we study languages with full ground references, i.e. references to integers and also references to references to integers, but not references to code.

Denotational inspiration

Some denotational models of higher-order languages with references:

- Game semantics of full ground references, using strong nominal sets (Laird; Murawski and Tzevelekos)
- Game semantics of "good" general references, using strong nominal sets (Laird; Murawski and Tzevelekos)
- Kripke semantics of full ground references, using ends and coends (Kammar, Levy, Moss and Staton)

Denotational inspiration

Some denotational models of higher-order languages with references:

- Game semantics of full ground references, using strong nominal sets (Laird; Murawski and Tzevelekos)
- Game semantics of "good" general references, using strong nominal sets (Laird; Murawski and Tzevelekos)
- Kripke semantics of full ground references, using ends and coends (Kammar, Levy, Moss and Staton)

This talk is just about a first-order language with full ground references.

I hope this will eventually yield an improved account of these higher-order models.

One reference type

We consider a first order language—no function types. The type syntax is.

$$A \quad ::= \quad \mathtt{0} \mid A + A \mid \mathtt{1} \mid A \times A \mid \mathtt{Ref} \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A$$

A reference (memory location) has type `Ref` and stores a value of type

$$D \stackrel{\text{def}}{=} \mathtt{bool} \times \mathtt{Ref} \times \mathtt{Ref} + \mathtt{nat}$$

Fancy version

Several sorts of references, e.g. red and blue.
A red reference has type $\mathtt{Ref}_{\mathsf{red}}$ and stores a value of type D_{red}.
A blue reference has type $\mathtt{Ref}_{\mathsf{blue}}$ and stores a value of type D_{blue}.

Typing rules

The calculus is “fine-grain call-by-value”.

The judgements are

- $w, \Gamma \vdash^{\mathsf{v}} V : A$ for values
- $w, \Gamma \vdash^{\mathsf{c}} M : A$ for commputations.

Here w is a *world*, which is a number—the size of the memory.
Or a finite sequence of sorts, if there are several sorts.

Term syntax

Here is the term syntax:

$$
\begin{array}{rcl}
V, W & ::= & \mathtt{x} \mid \mathtt{l} \mid \mathtt{in}_i\, V \mid \langle\rangle \mid \langle V, W \rangle \mid \mathtt{roll}\ V \\
M, N & ::= & \mathtt{return}\ V \mid M\ \mathtt{to}\ \mathtt{x}.\ M \\
 & & \quad \mid \mathtt{match}\ V\ \mathtt{as}\ \cdots \\
 & & \quad \mid \mathtt{if}\ V = W\ \mathtt{then}\ M\ \mathtt{else}\ N \\
 & & \quad \mid V := W.\ M \\
 & & \quad \mid \mathtt{read}\ V\ \mathtt{as}\ \mathtt{x}.\ M \\
 & & \quad \mid \mathtt{new}\ \overrightarrow{\mathtt{x} := V}.\ M
\end{array}
$$

Operational Semantics

For a computation $w \vdash^{\mathsf{c}} M : A$ we have

$$w, M \Downarrow w', V$$

where $w' \sqsupseteq w$ and $w' \vdash^{\mathsf{v}} V : A$.

Operational Semantics

For a computation $w \vdash^{\mathsf{c}} M : A$ we have

$$w, M \Downarrow w', V$$

where $w' \sqsupseteq w$ and $w' \vdash^{\mathsf{v}} V : A$.

The observational preorder is obtained from programs of boolean type.

Reference non-use and swap:

$$M \quad \simeq \quad \mathtt{new}\ \overrightarrow{\mathtt{x} := V}.\ M$$

$$\begin{aligned} &\mathtt{new}\ \overrightarrow{\mathtt{x} := V}, \mathtt{y} := W, \mathtt{y}'{:=}W', \overrightarrow{\mathtt{z} := V'}.\ M \\ \simeq\ &\mathtt{new}\ \overrightarrow{\mathtt{x} := V}, \mathtt{y}' := W', \mathtt{y}{:=}W, \overrightarrow{\mathtt{z} := V'}.\ M \end{aligned}$$

Freyd category

Fine-grain call-by-value is modelled by a distributive Freyd category.

- A category $\mathcal{C}$
- A category $\mathcal{K}$ with the same objects as $\mathcal{C}$
- An identity-on-objects functor $\iota : \mathcal{C} \to \mathcal{D}$
- Extra structure for the product and sum types.

A type denotes an object of $\mathcal{C}$.

A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism $[\![\Gamma]\!] \to [\![A]\!]$.

A computation $\Gamma \vdash^{\mathsf{c}} M : A$ denotes a $\mathcal{K}$-morphism $[\![\Gamma]\!] \to [\![A]\!]$.

More categories are required for terms $w,\Gamma \vdash^{\mathsf{v}} V : A$ and $w,\Gamma \vdash^{\mathsf{c}} M : A$.

Basic semantics of a computation (Levy 2002)

What is the meaning of a computation $w, \Gamma \vdash^{\mathsf{c}} M : A$?

Basic semantics of a computation (Levy 2002)

What is the meaning of a computation $w, \Gamma \vdash^{\mathsf{c}} M : A$?

Given

- some local cells $\overrightarrow{l}$
- an environment ρ for Γ, using cells in $w, \overrightarrow{l}$
- a state s for $w, \overrightarrow{l}$

the computation terminates with

- some more local cells $\overrightarrow{l'}$
- a value v of type A, using cells in $w, \overrightarrow{l}, \overrightarrow{l'}$
- a state s' for $w, \overrightarrow{l}, \overrightarrow{l'}$

In summary $\llbracket M \rrbracket : \overrightarrow{l}, \rho, s \mapsto \overrightarrow{l'}, v, s'$.

Equivariance

In the basic model, a computation denotes a function:

$$[\![M]\!] : \overrightarrow{l}, \rho, s \mapsto \overrightarrow{l'}, v, s'$$

In the strong nominal set model and the end/coend model

- functions are identified up to reference non-use and swap
- functions are constrained to propagate reference non-use and swap.

Aim

To reformulate the equivariant model in an explicit way:

- the homset $\mathcal{K}(A, B)$ will be of the form $\prod_{x \in X} Y(x)$
  "A computation denotes a function".
- elements of X and $Y(x)$ will not be equivalence classes.

Template sets

Let **Inj** be the category of finite sets and injections.

Then $\mathbf{Fam}(\mathbf{Inj}^{\mathrm{op}})$ is the category of template sets.

(Equivalent to strong nominal sets.)
A type denotes a template set, i.e. a family of finite sets.

Templates for $\mathtt{bool} \times \mathtt{Ref} \times \mathtt{Ref} + \mathtt{nat}$

$$\begin{array}{rll} \mathtt{inl}\ \langle b, -_0, -_0 \rangle & (b \in \mathbb{B}) & \text{Arity} = 1 \\ \mathtt{inl}\ \langle b, -_0, -_1 \rangle & (b \in \mathbb{B}) & \text{Arity} = 2 \\ \mathtt{inr}\ n & (n \in \mathbb{N}) & \text{Arity} = 0 \end{array}$$

The template set is indexed by $\mathbb{B} + \mathbb{B} + \mathbb{N}$.

Semantics of a value $\Gamma \vdash^{\mathsf{v}} V : A$

Let $[\![\Gamma]\!] = (X_i)_{i \in I}$ and $[\![A]\!] = (Y_j)_{j \in J}$.

Then V denotes a map of template sets $(X_i)_{i \in I} \to (Y_j)_{j \in J}$.

For every template i, we obtain

- a template j
- a map of references $Y_j \to X_i$.

Reason: every reference in $V[W_{\mathtt{x}}/\mathtt{x}]_{\mathtt{x} \in \Gamma}$ arises from one in $(W_{\mathtt{x}})_{\mathtt{x} \in \Gamma}$.

Product of template sets A and B

If $[\![A]\!] = (X_i)_{i \in I}$ and $[\![B]\!] = (Y_j)_{j \in J}$,

a template for $A \times B$ consists of

- a template i for A
- a template j for B
- a matching—i.e. equivalence relation R on $A + B$, discrete on each component

and the arity of (i, j, R) is $A + B \ / \ R$.

Indistinguishable inputs to $\mathtt{x} : \mathtt{bool} + \mathtt{Ref} \times \mathtt{Ref} \vdash^{\mathsf{c}} M : A$

$$
\begin{array}{rcl}
\mathtt{l}_0, \mathtt{l}_1, \mathtt{l}_2, \mathtt{l}_3 & & \\
\mathtt{x} & \mapsto & \mathtt{inr}\ \langle \mathtt{l}_1, \mathtt{l}_2 \rangle \\
\mathtt{l}_0 & := & \mathtt{inr}\ 17 \\
\mathtt{l}_1 & := & \mathtt{inl}\ \langle \mathtt{true}, \mathtt{l}_1, \mathtt{l}_1 \rangle \\
\mathtt{l}_2 & := & \mathtt{inl}\ \langle \mathtt{false}, \mathtt{l}_0, \mathtt{l}_2 \rangle \\
\mathtt{l}_0, \mathtt{l}_1, \mathtt{l}_2, \mathtt{l}_3, \mathtt{l}_4 & & \\
\mathtt{x} & \mapsto & \mathtt{inr}\ \langle \mathtt{l}_4, \mathtt{l}_0 \rangle \\
\mathtt{l}_0 & := & \mathtt{inl}\ \langle \mathtt{false}, \mathtt{l}_2, \mathtt{l}_0 \rangle \\
\mathtt{l}_1 & := & \mathtt{inl}\ \langle \mathtt{false}, \mathtt{l}_0, \mathtt{l}_0 \rangle \\
\mathtt{l}_2 & := & \mathtt{inr}\ 17 \\
\mathtt{l}_3 & := & \mathtt{inr}\ 5 \\
\mathtt{l}_4 & := & \mathtt{inl}\ \mathtt{true}, \mathtt{l}_4, \mathtt{l}_4
\end{array}
$$

Heap = finitely partitioned tree

A reference stores a value of type D, and $[\![D]\!] = (C_k)_{k \in K}$.

Heap = finitely partitioned tree

A reference stores a value of type D, and $[\![D]\!] = (C_k)_{k \in K}$.

For each element $x \in X_i$ we need a template $r(x) \in K$

For each $c_0 \in C_{r(x)}$ we need a template $r(x, c_0) \in K$

For each $c_1 \in C_{r(x,c)}$ we need a template $r(x, c_0, c_1) \in K$

Etc.

Heap = finitely partitioned tree

A reference stores a value of type D, and $[\![D]\!] = (C_k)_{k \in K}$.

For each element $x \in X_i$ we need a template $r(x) \in K$

For each $c_0 \in C_{r(x)}$ we need a template $r(x, c_0) \in K$

For each $c_1 \in C_{r(x,c)}$ we need a template $r(x, c_0, c_1) \in K$

Etc.

In summary, we need a (non-well-founded) forest with X_i roots, where each node is labelled with $k \in K$ and has C_k children.

Heap = finitely partitioned tree

A reference stores a value of type D, and $[\![D]\!] = (C_k)_{k \in K}$.

For each element $x \in X_i$ we need a template $r(x) \in K$

For each $c_0 \in C_{r(x)}$ we need a template $r(x, c_0) \in K$

For each $c_1 \in C_{r(x,c)}$ we need a template $r(x, c_0, c_1) \in K$

Etc.

In summary, we need a (non-well-founded) forest with X_i roots, where each node is labelled with $k \in K$ and has C_k children.

This gives a family of finite sets, one at each node.
Each node is a reference. Reference equality is observable, so the nodes are partitioned into finitely many parts.

In summary, a heap is a finitely partitioned forest.

Summary

A computation $\Gamma \vdash^{\mathsf{c}} M : A$ denotes a function

taking a template (environment) and finitely partitioned forest (heap)

to a template (value) and finitely partitioned forest (heap).

No equivariance constraint or identification are used,

as a finitely partitioned forest gives the observable data about a heap

and does not mention unreachable references or reference order.

We thus obtain a Freyd category on $\mathbf{Fam}(\mathbf{Inj}^{\mathrm{op}})$.