# Kripke semantics for full ground references (work in progress)

Levy, Paul

# Kripke semantics for full ground references (work in progress)

Paul Blain Levy, University of Birmingham

"Full ground references" means references to integers and to other references, but not functions or thunks. Game semantics for full ground references was given in [1]. This work aims to give Kripke semantics.

**Language** We fix a set $\mathcal{S}$ of *sorts*. We extend the call-by-push-value types with a reference type $\mathsf{Ref}_s$ for each sort $s$. We then define

$$\text{full ground types} \quad D ::= \quad 0 \mid D + D \mid \textstyle\sum_{i \in \mathbb{N}} D_i \mid 1 \mid D \times D \mid \mathsf{Ref}_s \ (s \in \mathcal{S})$$

and fix, for each sort $s$, a full ground type $D_s$, intended to be the type of values stored in a reference cell of sort $s$.

A *world* is a finite set over $\mathcal{S}$. (Alternatively: a finite sequence of sorts.) The judgements are $w, \Gamma \vdash^{\mathsf{v}} V : A$ for values and $w, \Gamma \vdash^{\mathsf{c}} M : \underline{B}$ for computations, where $w$ is a world and $\Gamma$ a typing context (finite set of identifiers with an associated value type). Term syntax and operational semantics is defined as usual. Evaluation terminates because of the restriction to full ground references.

**Denotational semantics** A value type $A$ will denote a functor from the category $\mathsf{Inj}$ of worlds and injections to **Set**. Intuitively, $[\![A]\!]w$ is a semantic domain for closed values in world $w$, and such values can be renamed along an injection $w \to w'$. In particular $[\![\mathsf{Ref}_s]\!]w$ is the set of $s$-sorted cells in $w$.

A $w$-store associates to each cell $l$ in $w$ a value $w \vdash^{\mathsf{v}} s_l : D_{\mathsf{sort}(l)}$. The semantic domain for these is

$$Sw \stackrel{\text{def}}{=} \prod_{l \in w} [\![D_{\mathsf{sort}(l)}]\!]w$$

Although for ground references $S$ is a functor $\mathsf{Inj}^{\mathsf{op}} \to \mathbf{Set}$, that is not so for full ground references.

Computation types have more subtle semantics. To understand it, say that a *$w$-store* $s$ associates to each cell $l$ in $w$ a value $w \vdash^{\mathsf{v}} s_l : D_{\mathsf{sort}(l)}$. An *SC-configuration* $\Gamma \vdash^{\mathsf{sc}} x, s, M : \underline{B}$ consists of

- a world $x$—we think of cells in $x$ as local/private, whereas those in $w$ are global/public
- a $w + x$-store $s$
- and a computation $w + x, \Gamma \vdash^{\mathsf{c}} M : \underline{B}$.

These arise in the operational semantics of a language that has both $w$-many global cells and generation of local cells.

We want $[\![\underline{B}]\!]w$ to be a semantic domain for closed SC-configurations in world $w$. What category should $[\![\underline{B}]\!]$ be a functor from?

Let's start with the ground ref setting. An *initialization* $(i, p) : w \to w'$ consists of

- an injection $i : w \to w'$—we write $\mathsf{new}(i)$ for the cells in $w'$ not in the range of $i$
- and an element $p \in \prod_{l \in \mathsf{new}(i)} [\![D_{\mathsf{sort}(l)}]\!]$

These form a category $\mathsf{Init}$. A *partial initialization* is defined similarly, except that $i$ is a partial injection. The latter form a category $\mathsf{PInit}$ that contains both $\mathsf{Inj}^{\mathsf{op}}$ and $\mathsf{Init}$, and indeed is freely generated by these subcategories modulo two equations.

A partial initialization $i : w \to w'$ converts an SC-configuration in world $w$ to one in world $w'$, by

- hiding the cells in $w$ that are not in the domain of $i$
- renaming each cell in the domain of $i$ to one in the range
- creating each cell $l \in \mathsf{new}(i)$ with value $p_l$.

So a computation type $\underline{B}$ should denote a functor $\mathsf{PInit} \to \mathbf{Set}$.

Turning to full ground references, an initialization $(i,p) : w \to w'$ consists of

- an injection $i : w \to w'$
- and an element $p \in \prod_{l \in \mathsf{new}(i)} [\![D_{\mathsf{sort}(l)}]\!](w + \mathsf{new}(i))$

These form a category $\mathsf{Init}$, and $S$ is a functor $\mathsf{Init} \to \mathbf{Set}$. But partial initializations are more subtle. Let's first say that a *stateful value* $\Gamma \vdash^{\mathsf{sv}} x, s, V : A$ consists of

- a world $x$—again, cells in $x$ are local/private, whilst those in $x$ are global/public
- for each local cell $l$ in $x$, a value $w + x \vdash^{\mathsf{v}} s_l : D_{\mathsf{sort}(l)}$
- and a value $w + x \vdash^{\mathsf{v}} V : A$.

For any functor $A : \mathsf{Inj} \to \mathbf{Set}$, define

$$(\Psi A)\, w \stackrel{\text{def}}{=} \int^{x \in \mathsf{Init}} \prod_{l \in x} [\![D_{\mathsf{sort}(l)}]\!](w + x) \times [\![A]\!](w + x)$$

so that $(\Psi[\![A]\!])w$ is a semantic domain for closed stateful values of type $A$ in world $w$. If $A$ is a constant functor, then $(\Psi A)w \cong Aw$. A *partial initialization* $(i,p) : w \to w'$ consists of

- a partial injection $i : w \to w'$
- and an element $p \in (\Psi \prod_{l \in \mathsf{new}(i)} [\![D_{\mathsf{sort}(l)}]\!])(w + \mathsf{new}(i))$.

These form a category $\mathsf{PInit}$ that contains both $\mathsf{Inj}^{\mathsf{op}}$ and $\mathsf{Init}$ and indeed is freely generated by these subcategories modulo two equations. A computation type $\underline{B}$ should denote a functor $\mathsf{PInit} \to \mathbf{Set}$. Also, if $A : \mathsf{Inj} \to \mathbf{Set}$ then $\Psi A : \mathsf{PInit}^{\mathsf{op}} \to \mathbf{Set}$.

Semantics of judgements:

- A value $w, \Gamma \vdash^{\mathsf{v}} V : A$ denotes a family of functions $[\![\Gamma]\!](w+x) \to [\![A]\!](w+x)$, natural in $x \in \mathsf{Inj}$.
- A computation $w, \Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a family of functions $S(w+x) \times [\![\Gamma]\!](w+\overline{x}) \to [\![\underline{B}]\!](w+x)$, natural in $x \in \mathsf{Init}$. Here overline represents the forgetful functor $\mathsf{Init} \to \mathsf{Inj}$.

Semantics of types:

$$\begin{aligned}
[\![FA]\!]w &\stackrel{\text{def}}{=} \int^{x \in \mathsf{Init}} S(w+x) \times [\![A]\!](w+\overline{x}) \\
[\![U\underline{B}]\!]w &\stackrel{\text{def}}{=} \int_{x \in \mathsf{Init}} S(w+x) \to [\![\underline{B}]\!](w+x) \\
[\![\prod_{i \in I} \underline{B}_i]\!]w &\stackrel{\text{def}}{=} \prod_{i \in I} [\![\underline{B}_i]\!]w \\
[\![A \to \underline{B}]\!]w &\stackrel{\text{def}}{=} \int_{x \in \mathsf{Init}} [\![A]\!](w+\overline{x}) \to [\![\underline{B}]\!](w+x) \\
[\![U \prod_{i \in I}(A_i \to \underline{B}_i)]\!]w &\cong \int_{x \in \mathsf{Init}} S(w+x) \to \prod_{i \in I}([\![A_i]\!](w+\overline{x}) \to [\![\underline{B}_i]\!](w+x))
\end{aligned}$$

# References


[1] Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2012.