Denotational semantics of general store and polymorphism

Jonathan Sterling[1]

Aarhus University

February 17^{th}, 2023

Funded by the European Union

[1]Joint work with Daniel Gratzer and Lars Birkedal

Denotational semantics for realistic PLs

Classical domain theory provides **the** account of general recursion, but struggled to combine more complex features, including two that are commonly dealt with operationally:

- ***higher-order* store**: where you can store functions and even other pointers in the heap;
- **concurrency**: many advances in the denotational semantics world (*e.g*. powerdomains & event structures), unfinished.

Today: I will show how to combine **guarded recursion** with **polymorphic types** to easily define denotational models of higher-order store with polymorphism.

Monadic System $\mathbf{F}^\omega$ with reference types

Our language is a version of System $\mathbf{F}^\omega$ extended by an "IO monad" with reference types:

$\mathbf{T} : \star \to \star$

$\mathbf{Ref} : \star \to \star$

$\mathbf{get} : \mathbf{Ref}\ \alpha \to \mathbf{T}\ \alpha$

$\mathbf{set} : \mathbf{Ref}\ \alpha \to \alpha \to \mathbf{T}\ ()$

$\mathbf{new} : \alpha \to \mathbf{T}\ (\mathbf{Ref}\ \alpha)$

Kripke semantics of reference types

The classic *state monad* handles a single cell of fixed type:

$$\mathbf{State}\ \sigma\ \alpha = \sigma \to (\sigma \times \alpha)$$

Our situation is harder: we can allocate new cells, and store anything we want in there.

Thus the denotation $[\![\mathbf{Ref}\ \alpha]\!]$ must depend on the "current" heap layout, which is always growing.

The solution is to *parameterize* $[\![-]\!]$ in heap layouts and require all denotations to be *monotone* in the growth of the heap (Reynolds, Oles, O'Hearn, *etc*.). Called **Kripke semantics**.

Defining the poset *World* of heap layouts

A heap layout w should map a finite set of global addresses to *semantic types*.

A semantic type A should be a (monotone) *family of sets* A_w indexed in heap layouts w, *i.e.* a **functor** from heap layouts to sets.

Defining the poset $\mathcal{World}$ of heap layouts

A heap layout w should map a finite set of global addresses to *semantic types*.

A semantic type A should be a (monotone) *family of sets* A_w indexed in heap layouts w, *i.e.* a **functor** from heap layouts to sets.

This is circular, in a bad way! When $\mathcal{U}$ is some non-trivial set of sets, we cannot solve the following system of equations:

$$\begin{aligned} \mathcal{World} &\cong \mathsf{Addr} \rightharpoonup_{fin.} \mathcal{Type} \\ \mathcal{Type} &\cong \mathbf{Functor}(\mathcal{World}, \mathcal{U}) \end{aligned}$$

This was solved using Appel and McAllester's *step-indexing* by Amal Ahmed, and further developed by many others.

A step-indexed poset of heap layouts

The idea of Appel and McAllester was, roughly, to *stratify* the definition of $\mathcal{World}$ in its unrollings of finite depth.

Idea: every set is replaced by an *antitone* ω-indexed family of sets, *i.e*. a functor $\omega^{op} \longrightarrow \mathbf{Set}$.

$$\begin{aligned}
\mathcal{World}_n &= \mathsf{Addr} \rightharpoonup_{fin.} \varprojlim_{k<n} \mathcal{Type}_k \\
\mathcal{Type}_n &= \mathbf{Functor}(\mathcal{World}_n \times \omega^{op}, \mathcal{P}(\mathsf{Val}))
\end{aligned}$$

The above is well-defined! But it is clearly a mess... **We can tame it with *guarded dependent type theory*.**

Denotational semantics in guarded type theory

Guarded dependent type theory / **GDTT** is a version of dependent type theory whose purpose is to speak of functors $\omega^{op} \longrightarrow \mathbf{Set}$.

GDTT has so far been used to give elegant denotational semantics to *non-polymorphic* languages with general recursion, recursive types, and non-determinism.

See the work of Birkedel, Møgelberg, Paviotti, Veltri, Vezzosi, *etc*.

Naïve heap layouts, in guarded type theory

We can use **GDTT** as a "domain specific language" to replace set theory, and remove all the indices from our definition:

$$\mathcal{W}orld_n = \mathsf{Addr} \rightharpoonup_{fin.} \varprojlim_{k<n} \mathcal{T}ype_k$$

$$\mathcal{T}ype_n = \mathbf{Functor}(\mathcal{W}orld_n \times \omega^{op}, \mathcal{P}(\mathsf{Val}))$$

Naïve heap layouts, in guarded type theory

We can use **GDTT** as a "domain specific language" to replace set theory, and remove all the indices from our definition:

$$\mathcal{W}orld = \mathsf{Addr} \rightharpoonup_{fin.} \blacktriangleright \mathcal{T}ype$$
$$\mathcal{T}ype = \mathbf{Functor}(\mathcal{W}orld, \mathcal{U})$$

Naïve heap layouts, in guarded type theory

We can use **GDTT** as a "domain specific language" to replace set theory, and remove all the indices from our definition:

$$\mathcal{World} = \mathsf{Addr} \rightharpoonup_{fin.} \blacktriangleright \mathcal{Type}$$
$$\mathcal{Type} = \mathbf{Functor}(\mathcal{World}, \mathcal{U})$$

What is $\blacktriangleright$? It is a *dependent applicative functor* that is built into **GDTT** called the "later modality", interpreted as follows:

$$[\![\blacktriangleright A]\!]_n = \varprojlim_{k<n} [\![A]\!]_k$$

(Dependent applicative functors support a form of "do-notation" $\blacktriangleright[x \leftarrow u, \ldots].\, B$ where $u : \blacktriangleright A$ and $x : A, \ldots \vdash B$ is a type.)

The Ref type in guarded type theory

We can now give the denotation of the **Ref** type in **GDTT**.

$$
\begin{aligned}
&[\![\mathbf{Ref}]\!] : \mathcal{T}ype \to \mathcal{T}ype \\
&[\![\mathbf{Ref}]\!]\ A = \\
&\quad \lambda w : \mathcal{W}orld. \\
&\qquad \left\{ l \in |w| \;\middle|\; wl = \mathsf{next}\, A \right\}
\end{aligned}
$$

The Ref type in guarded type theory

We can now give the denotation of the **Ref** type in **GDTT**.

$$
\begin{aligned}
&[\![\mathbf{Ref}]\!] : \mathcal{T}ype \longrightarrow \mathcal{T}ype \\
&[\![\mathbf{Ref}]\!]\ A = \\
&\quad \lambda w : \mathcal{W}orld. \\
&\qquad \left\{ l \in |w| \;\middle|\; wl = \mathsf{next}\, A \right\}
\end{aligned}
$$

Are we done? **No.**

Defining the T monad?

Suppose we want to define $\mathbf{T}$ as a kind of state monad. First we must define what the states (heaps) are:

$\mathbf{H}_w : \mathcal{U}$ for each $w : \mathcal{W}orld$

$\mathbf{H}_w = \prod_{l \in |w|} \blacktriangleright[X \leftarrow wl].\, Xw$

A naïve attempt to define $[\![\mathbf{T}]\!]$, using the *guarded lift monad* $\mathbf{L}X = X + \blacktriangleright X$ to support recursion.

$$
\begin{aligned}
&[\![\mathbf{T}]\!] : \mathcal{T}\!ype \to \mathcal{T}\!ype \\
&[\![\mathbf{T}]\!]\, A = \\
&\quad \lambda w : \mathcal{W}\!orld. \\
&\qquad \prod_{w' \geq w} \mathbf{H}_{w'} \to \sum_{w'' \geq w'} \mathbf{L}\,(\mathbf{H}_{w''} \times A w'')
\end{aligned}
$$

A naïve attempt to define $[\![\mathbf{T}]\!]$, using the *guarded lift monad* $\mathbf{L}X = X + \blacktriangleright X$ to support recursion.

$$
\begin{aligned}
&[\![\mathbf{T}]\!] : \mathcal{T}\!ype \longrightarrow \mathcal{T}\!ype \\
&[\![\mathbf{T}]\!]\, A = \\
&\quad \lambda w : \mathcal{W}\!orld. \\
&\qquad {\color{red}\textstyle\prod_{w' \geq w}} \mathbf{H}_{w'} \longrightarrow {\color{red}\textstyle\sum_{w'' \geq w'}} \mathbf{L}\,(\mathbf{H}_{w''} \times A w'')
\end{aligned}
$$

The above is irredeemably ill-typed:

- we are trying to define a type $[\![\mathbf{T}]\!]\; A\; w : \mathcal{U}$
- but $\mathcal{W}\!orld$ is as big as $\mathcal{U}$, so $\mathcal{U}$ is not closed under $\mathcal{W}\!orld$-indexed products and sums!

Thus **GDTT** is *inadequate* for defining a typed denotational semantics of higher-order store, **but all that is missing is polymorphism**. Imagine:

$$[\![\mathbf{T}]\!]\; A\; w = \forall_{w' \geq w} \mathbf{H}_{w'} \to \mathbf{L} \exists_{w'' \geq w'} \mathbf{H}_{w''} \times A w''$$

Thus we are lead to develop an **impredicative** version of **GDTT**.

Impredicative guarded dependent type theory

iGDTT extends the the Birkedal–Møgelberg–Paviotti program of guarded denotational semantics to languages that combine **polymorphism** with **realistic computational effects**.

iGDTT augments **GDTT** with the “impredicative `Set`” universe from the old calculus of constructions / Coq.

The definition of iGDTT, formally

The structure of **iGDTT** is as follows:

1. a hierarchy of predicative universes $\mathbf{Type}_i$;
2. an **impredicative** universe $\mathbf{Prop} \in \mathbf{Type}_i$ of proof-irrelevant types satisfying propositional extensionality;
3. an **impredicative** universe $\mathbf{iSet} \in \mathbf{Type}_i$ with $\mathbf{Prop} \subseteq \mathbf{iSet}$;
4. all universes have $\prod$, $\sum$, $(=)$, inductive types, and $\blacktriangleright$.

Note that $\mathbf{Prop} \notin \mathbf{iSet}$ and $\mathbf{Prop}$ is *not* a subobject classifier!

Universal and existential types in iGDTT

An impredicative universe $\mathbb{X} \in \mathbf{Type}_i$ is one that is closed under *large* universal quantification:

$$\frac{A : \mathbf{Type}_i \qquad x : A \vdash Bx : \mathbb{X}}{\forall_{x:A}\, Bx : \mathbb{X}} \qquad \uparrow_{\mathbb{X}}^{\mathbf{Type}_i}(\forall_{x:A}\, Bx) \cong \prod_{x:A} \uparrow_{\mathbb{X}}^{\mathbf{Type}_i}(Bx)$$

Universal and existential types in iGDTT

An impredicative universe $\mathbb{X} \in \mathbf{Type}_i$ is one that is closed under *large* universal quantification:

$$\frac{A : \mathbf{Type}_i \qquad x : A \vdash Bx : \mathbb{X}}{\forall_{x:A} Bx : \mathbb{X}} \qquad \uparrow_{\mathbb{X}}^{\mathbf{Type}_i}(\forall_{x:A} Bx) \cong \prod_{x:A} \uparrow_{\mathbb{X}}^{\mathbf{Type}_i}(Bx)$$

If $\mathbb{X}$ is closed under $(=)$, then it is automatically closed under *existential quantification*, via the coherent impredicative encoding of Awodey, Frey, and Speight (2018).

$$(\exists_{x:A} Bx) \subseteq \prod_{C:\mathbb{X}} \prod_{k:\prod_{x:A} \prod_{b:Bx} C} C$$

Although $\forall_{x:A} Bx$ is the dependent product, it is *not* the case that $\exists_{x:A} Bx$ is the dependent sum. (It is a so-called “weak sum”.)

Denotational semantics of state in iGDTT

Finally our denotational semantics can be defined!

$$
\begin{aligned}
&\mathcal{W}\!\mathit{orld} = \mathsf{Addr} \rightharpoonup_{\mathit{fin.}} \blacktriangleright\mathcal{T}\!\mathit{ype} \\
&\mathcal{T}\!\mathit{ype} = \mathbf{Functor}(\mathcal{W}\!\mathit{orld}, \mathbf{iSet}) \\
&\mathbf{H}_w = \textstyle\prod_{l \in |w|} \blacktriangleright[X \leftarrow wl].\, Xw
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathbf{Ref}]\!] : \mathcal{T}\!\mathit{ype} \longrightarrow \mathcal{T}\!\mathit{ype} \\
&[\![\mathbf{Ref}]\!]\; A\; w = \left\{ l \in |w| \;\middle|\; wl = \mathsf{next}\, A \right\}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathbf{T}]\!] : \mathcal{T}\!\mathit{ype} \longrightarrow \mathcal{T}\!\mathit{ype} \\
&[\![\mathbf{T}]\!]\; A\; w = \forall_{w' \geq w}\, \mathbf{H}_{w'} \to \mathbf{L}\, \exists_{w'' \geq w'}\, \mathbf{H}_{w''} \times Aw''
\end{aligned}
$$

How do we know this is OK?

The original **GDTT** was justified in the topos of trees $\mathbf{Functor}(\omega^{op}, \mathbf{Set})$. What about **iGDTT**?

1. Take *any* non-trivial realizability topos $\mathscr{E}$;
2. Take *any* non-trivial internal well-founded poset $\mathbb{O}$ in $\mathscr{E}$;
3. Then the category of *internal* diagrams $\mathbf{Functor}_{\mathscr{E}}(\mathbb{O}^{op}, \mathscr{E})$ contains a non-trivial model of **iGDTT**.

Further directions

1. Our model construction *also* justifies a version of **iGDTT** with an **T** monad! (Important for languages like Idris 2 and Lean 4, which currently have no semantics.)
2. **Easy to extend** with additional computational effects, via a call-by-push-value decomposition. (See our manuscript.)

Future work:

1. Try combining with the Møgelberg–Vezzosi guarded powerdomains.
2. Adapt **Iris**-style program logics to denotational semantics (ongoing with Aagaard, Birkedal).
3. Experiment with a *resumption-style* version of our monad, to prepare for concurrency.

Bibliography I

Ahmed, Amal Jamil (2004). "Semantics of Types for Mutable State". PhD thesis. Princeton University. URL: `http://www.ccs.neu.edu/home/amal/ahmedthesis.pdf`.

Appel, Andrew W. and David McAllester (Sept. 2001). "An Indexed Model of Recursive Types for Foundational Proof-carrying Code". In: *ACM Transactions on Programming Languages and Systems* 23.5, pp. 657–683. ISSN: 0164-0925. DOI: `10.1145/504709.504712`.

Appel, Andrew W., Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon (2007). "A Very Modal Model of a Modern, Major, General Type System". In: *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Nice, France: Association for Computing Machinery, pp. 109–122. ISBN: 1-59593-575-4.

Awodey, Steve, Jonas Frey, and Sam Speight (2018). "Impredicative Encodings of (Higher) Inductive Types". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. Oxford, United Kingdom: Association for Computing Machinery, pp. 76–85. ISBN: 978-1-4503-5583-4. DOI: `10.1145/3209108.3209130`.

Birkedal, Lars, Aleš Bizjak, et al. (2019). "Guarded Cubical Type Theory". In: *Journal of Automated Reasoning* 63.2, pp. 211–253. DOI: `10.1007/s10817-018-9471-7`.

Birkedal, Lars, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring (2011). "First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees". In: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*. Washington, DC, USA: IEEE Computer Society, pp. 55–64. ISBN: 978-0-7695-4412-0. DOI: `10.1109/LICS.2011.16`. arXiv: `1208.3596 [cs.LO]`.

Bibliography II

Birkedal, Lars, Kristian Støvring, and Jacob Thamsborg (2010). "Realisability semantics of parametric polymorphism, general references and recursive types". In: *Mathematical Structures in Computer Science* 20.4, pp. 655–703. DOI: 10.1017/S0960129510000162.

Bizjak, Aleš et al. (2016). "Guarded Dependent Type Theory with Coinductive Types". In: *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*. Ed. by Bart Jacobs and Christof Löding. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 20–35. ISBN: 978-3-662-49630-5. DOI: 10.1007/978-3-662-49630-5_2. arXiv: 1601.01586 [cs.LO].

Levy, Paul Blain (2003a). "Adjunction Models For Call-By-Push-Value With Stacks". In: *Electronic Notes in Theoretical Computer Science* 69. CTCS'02, Category Theory and Computer Science, pp. 248–271. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(04)80568-1.

— (Jan. 1, 2003b). *Call-by-Push-Value: A Functional/Imperative Synthesis*. Kluwer, Semantic Structures in Computation, 2. ISBN: 1-4020-1730-8.

Møgelberg, Rasmus Ejlers and Marco Paviotti (2016). "Denotational Semantics of Recursive Types in Synthetic Guarded Domain Theory". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: Association for Computing Machinery, pp. 317–326. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2934516.

Bibliography III

Møgelberg, Rasmus Ejlers and Niccolò Veltri (Jan. 2019). "Bisimulation as Path Type for Guarded Recursive Types". In: *Proceedings of the ACM on Programming Languages* 3.POPL. DOI: `10.1145/3290317`.

Møgelberg, Rasmus Ejlers and Andrea Vezzosi (Dec. 2021). "Two Guarded Recursive Powerdomains for Applicative Simulation". In: *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics*. Vol. 351. Electronic Proceedings in Theoretical Computer Science, pp. 200–217. DOI: `10.4204/EPTCS.351.13`.

Palombi, Daniele and Jonathan Sterling (2022). "Classifying topoi in synthetic guarded domain theory". In: *Proceedings 38th Conference on Mathematical Foundations of Programming Semantics, MFPS 2022*. To appear. arXiv: `2210.04636 [math.CT]`.

Paviotti, Marco (2016). "Denotational semantics in Synthetic Guarded Domain Theory". PhD thesis. Denmark: IT-Universitetet i København. ISBN: 978-87-7949-345-2.

Paviotti, Marco, Rasmus Ejlers Møgelberg, and Lars Birkedal (2015). "A Model of PCF in Guarded Type Theory". In: *Electronic Notes in Theoretical Computer Science* 319.Supplement C. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI), pp. 333–349. ISSN: 1571-0661. DOI: `10.1016/j.entcs.2015.12.020`.

Sterling, Jonathan, Daniel Gratzer, and Lars Birkedal (July 2022). "Denotational semantics of general store and polymorphism". Unpublished manuscript. DOI: `10.48550/arXiv.2210.02169`.

Bibliography IV

Veltri, Niccolò and Andrea Vezzosi (2020). "Formalizing π-Calculus in Guarded Cubical Agda". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. New Orleans, LA, USA: Association for Computing Machinery, pp. 270–283. ISBN: 978-1-4503-7097-4. DOI: 10.1145/3372885.3373814.