A Filter Model for the State Monad (short paper)

Ugo de'Liguoro and Riccardo Treglia

Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy
ugo.deliguoro@unito.it riccardo.treglia@unito.it

Abstract. We consider an untyped computational lambda calculus equipped with primitives to read and write from a global store. The calculus is modeled into a solution of a domain equation involving the state monad. We introduce an intersection type theory with subtyping such that the induced filter model is a solution of such an equation. Finally we define a type assignment system which is sound and complete w.r.t. any model of the calculus.

Keywords: Intersection types · Computational λ-calculi · State Monad

1 An untyped imperative λ-calculus and its semantics

We conceive the calculus λ_{imp} as an untyped call-by-value λ-calculus extended with two operators to read and write from a store. To model such side-effects, we choose a monadic setting.
In Wadler's formulation [12], a monad is a triple $(\mathbf{T}, \mathit{unit}, \star)$ where $\mathbf{T}$ is a type constructor, and for all types D, E, $\mathit{unit}_D : D \to \mathbf{T}D$ and $\star_{D,E} : \mathbf{T}D \times (D \to \mathbf{T}E) \to \mathbf{T}E$ are such that (omitting subscripts and writing $\star$ as an infix operator):

$$(\mathit{unit}\ d) \star f = f\, d, \qquad a \star \mathit{unit}\ = a, \qquad (a \star f) \star g = a \star \lambda\!\!\lambda d.(f\, d \star g).$$

Instances of monads are partiality, exceptions, input/output, store, non determinism, continuations.
Following [7], an untyped computational calculus has a model that is the solution of the equation $D \cong D \to \mathbf{T}D$ in the category **Dom** of domains, namely the call-by-value reflexive object.
Such domain equation implies that we have just two types: the type of *values* D, and the type of *computations* $\mathbf{T}D$.
Since now $D \cong D \to \mathbf{T}D$, we have:

$$\star : \mathbf{T}D \times (D \to \mathbf{T}D) \to \mathbf{T}D \cong \mathbf{T}D \times D \to \mathbf{T}D$$

Therefore, as in [4] the syntax of the monadic calculus is

$$\mathit{Val} : V, W ::= x \mid \lambda x.M \qquad \mathit{Com} : M, N ::= [V] \mid M \star V$$

To model λ_{imp} we instantiate $\mathbf{T}$ to a variant of the state monad defined as $\mathbb{S}X = S \to (X \times S)_\perp$, where S is a suitable space of states, and $(\cdot)_\perp$ is the lifting monad. A natural choice for S is (a subspace of) $\mathbf{L} \to D$ (with the order induced by that one of D), where D is the intended domain of values. However, D is the solution of the domain equation $D \cong D \to \mathbb{S}D$, which is clearly circular.
To break the circularity, we define the mixed-variant bi-functor $G : \mathbf{Dom}^{op} \times \mathbf{Dom} \to \mathbf{Dom}$ by $G(X,Y) = FX \to (Y \times FY)_\perp$, where $FX = \mathbf{L} \to X$, $Fh = h \circ \_$ and $G(f,g)(x) = (g \times Fg)_\perp \circ \alpha \circ Ff$ where $f : X' \to X$, $g : Y \to Y'$ and $\alpha \in G(X,Y)$. Now it is routine to prove that G is locally continuous (see e.g. [8]) so that, by the inverse limit technique, we can find the initial solution to the domain equation $D \cong D \to G(D,D)$. Now let us define $\mathbb{S}_D X = FD \to (X \times FX)_\perp$, that is a functor and a monad, and we conclude:

Theorem 1. *There exists a domain D such that the state monad $\mathbb{S}_D$ is a solution in **Dom** to the domain equation: $D \cong D \to \mathbb{S}_D\, D$. Moreover, it is initial among all solutions to such equation.*

It remains to define the operators to read and write from the store, for which we use ideas from [9–11]: $\mathit{get}_\ell : (D \to \mathbb{S}_D D) \to \mathbb{S}_D D$ and $\mathit{set}_\ell : D \times \mathbb{S}_D D \to \mathbb{S}_D D$; so the syntax of λ_{imp} is completed by

$$\mathit{Com} : M, N ::= \ldots \mid \mathit{get}_\ell(\lambda x.M) \mid \mathit{set}_\ell(V, M) \quad (\ell \in \mathbf{L})$$

The additional operators are actually two families of operators, indexed over the denumerable set of locations: $\mathit{get}_\ell(\lambda x.M)$ reading the value V associated to the location ℓ in the current *state*, and binding x to V in M; $\mathit{set}_\ell(V, M)$ which modifies the state assigning V to ℓ, and then proceeds as M. The operational semantics is given in [5].

Denotational Semantics. The triple $(\mathbb{S}_D, \mathit{unit}, \star)$ is a monad, where $(\mathit{unit}\, d)\, \varsigma = (d, \varsigma)$ and $(m \star d)\, \varsigma : \mathit{let}\ (d', \varsigma') = m\, \varsigma\ \mathit{in}\ (d\, d')\varsigma$ which is $\perp$ if $m\, \varsigma = \perp$, and $d \in D \cong D \to \mathbb{S}_D D$.
The semantics à la Plotkin and Power [9–11] of get_ℓ and set_ℓ, is given by $[\![\mathit{get}_\ell]\!] : (\mathbb{S}_D D)^D \to \mathbb{S}_D D$ where $[\![\mathit{get}_\ell]\!]\, d\, \varsigma = d\, (\varsigma\, \ell)\, \varsigma$, and $[\![\mathit{set}_\ell]\!] : D \times \mathbb{S}_D D \to \mathbb{S}_D D$ where $[\![\mathit{set}_\ell]\!](d, c)\, \varsigma = c(\varsigma[\ell \mapsto d])$ and $c \in \mathbb{S}_D D = S \to (D \times S)_\perp$ and $\varsigma[\ell \mapsto d]$ is the store sending ℓ to d and it is equal to ς, otherwise.
Then we interpret values from *Val* in D and computations from *Com* in $\mathbb{S}_D D$ via the maps $[\![\cdot]\!]^D : \mathit{Val} \to \mathit{Env} \to D$ and $[\![\cdot]\!]^{\mathbb{S}D} : \mathit{Com} \to \mathit{Env} \to \mathbb{S}_D D$, where $\mathit{Env} = \mathit{Var} \to D$ is the set of environments interpreting term variables.

Definition 1. *A λ_{imp}-**model** is a structure $\mathcal{D} = (D, \mathbb{S}_D, [\![\cdot]\!]^D, [\![\cdot]\!]^{\mathbb{S}D})$ such that:*

1. *D is a domain s.t. $D \cong D \to \mathbb{S}_D D$, where $\mathbb{S}_D$ is the state monad;*

2. *for all $e \in Env$, $V \in Val$ and $M \in Com$:*

$$\begin{array}{ll} [\![x]\!]^D e = e(x) & [\![\lambda x.M]\!]^D e = \lambda d \in D.\, [\![M]\!]^{\mathbb{S}D} e[x \mapsto d] \\ [\![\,[V]\,]\!]^{\mathbb{S}D} e = unit\ ([\![V]\!]^D e) & [\![M \star V]\!]^{\mathbb{S}D} e = ([\![M]\!]^{\mathbb{S}D} e) \star ([\![V]\!]^D e) \\ [\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e = [\![get_\ell]\!]([\![\lambda x.M]\!]^D e) & [\![set_\ell(V,M)]\!]^{\mathbb{S}D} e = [\![set_\ell]\!]([\![V]\!]^D e, [\![M]\!]^{\mathbb{S}D} e) \end{array}$$

By unravelling definitions and applying to an arbitrary store $\varsigma \in S$, the last two clauses can be written:

$$\begin{array}{r} [\![get_\ell(\lambda x.M)]\!]^{\mathbb{S}D} e\, \varsigma = [\![M]\!]^{\mathbb{S}D} (e[x \mapsto \varsigma(\ell)])\, \varsigma \\ [\![set_\ell(V,M)]\!]^{\mathbb{S}D} e\, \varsigma = [\![M]\!]^{\mathbb{S}D} e\, (\varsigma[\ell \mapsto [\![V]\!]^D e]) \end{array}$$

For $M, N \in Com$, we say that the equation $M = N$ is *true* in $\mathcal{D}$, notation $\models^{\mathcal{D}} M = N$, if $[\![M]\!]^{\mathbb{S}D} e = [\![N]\!]^{\mathbb{S}D} e$ for all $e \in Env$.

Proposition 1. *The following equations are true in any λ_{imp}-model $\mathcal{D}$:*

1. $[V] \star (\lambda x.M) = M[V/x]$
2. $M \star \lambda x.[x] = M$
3. $(L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N)$
4. $get_\ell(\lambda x.M) \star W = get_\ell(\lambda x.(M \star W))$
5. $set_\ell(V, M) \star W = set_\ell(V, M \star W)$

The above proposition states that the three monadic equations (parts *(1)* to *(3)*) are true in any model, while parts *(4)* and *(5)* imply that the operators *set* and *get* are algebraic.

2 The filter model construction

Following Abramsky [1], domains in a suitable category can be described via an intersection type theory. A *type theory* is a structure $Th_A = (\mathcal{L}_A, \wedge, \leq_A, \omega_A)$ where $\mathcal{L}_A$ is the language of Th_A, inducing the algebraic domain A (see below), namely a set of type expressions closed under $\wedge$; $\omega_A \in \mathcal{L}_A$ is a special constant, and $\leq_A$ is a pre-order over $\mathcal{L}_A$ such that $\alpha \leq_A \omega_A$ for all $\alpha \in \mathcal{L}_A$ and $\alpha \wedge \alpha'$ is the meet of α, α' w.r.t. $\leq_A$.
A non empty $F \subseteq \mathcal{L}_A$ is a *filter* of Th_A if it is upward closed and closed under intersection; let $\mathcal{F}_A$ be the set of filters of Th_A; then $(\mathcal{F}_A, \subseteq)$ is a domain. We seek theories Th_D and Th_S such that $\mathcal{F}_D \cong [\mathcal{F}_D \to \mathcal{F}_S \to (\mathcal{F}_D \times \mathcal{F}_S)_\perp]$.
Actually, we consider four theories, whose definitions are mutually inductive, derived by applying operators in the above equation. Indeed these are specific functors that can be represented as type theoretical constructors. We recall that if Th_A, Th_B are type theories, then we may define for $\alpha \in \mathcal{L}_A$ and $\beta \in \mathcal{L}_B$ the following languages:

$$\begin{array}{lll} \mathcal{L}_{A_\perp} & \psi ::= \alpha \mid \psi \wedge \psi' \mid \omega_{A_\perp} & \text{bottom theory} \\ \mathcal{L}_{A\times B} & \pi ::= \alpha \times \beta \mid \pi \wedge \pi' \mid \omega_{A\times B} & \text{product theory} \\ \mathcal{L}_{A\to B} & \phi ::= \alpha \to \beta \mid \phi \wedge \phi' \mid \omega_{A\to B} & \text{arrow theory} \\ \mathcal{L}_{S,A} & \sigma ::= \langle \ell : \alpha \rangle \mid \sigma \wedge \sigma' \mid \omega_S & \text{store theory} \end{array}$$

Then the respective theories are:

$$
\begin{array}{lll}
Th_{A_\perp} & : & \alpha \leq_A \alpha' \Longrightarrow \alpha \leq_{A_\perp} \alpha' \\
Th_{A\times B} & : & \omega_{A\times B} \leq_{A\times B} \omega_A \times \omega_B \quad (\alpha\times\beta)\wedge(\alpha'\times\beta') \leq_{A\times B} (\alpha\wedge\alpha')\times(\beta\wedge\beta') \\
 & & \alpha \leq_A \alpha' \quad \beta \leq_B \beta' \Longrightarrow \alpha\times\beta \leq_{A\times B} \alpha'\times\beta' \\
Th_{A\to B} & : & \omega_{A\to B} \leq_{A\to B} \omega_A \to \omega_B \quad (\alpha\to\beta)\wedge(\alpha\to\beta') \leq_{A\to B} \alpha\to(\beta\wedge\beta') \\
 & & \alpha' \leq_A \alpha \quad \beta \leq_B \beta' \Longrightarrow \alpha\to\beta \leq_{A\to B} \alpha'\to\beta' \\
Th_{S,A} & : & \langle \ell : \alpha\rangle \wedge \langle \ell : \alpha'\rangle \leq_{S,A} \langle \ell : \alpha\wedge\alpha'\rangle \quad \alpha \leq_A \alpha' \Longrightarrow \langle \ell:\alpha\rangle \leq_{S,A} \langle \ell:\alpha'\rangle
\end{array}
$$

We assume that $\wedge$ and $\times$ take precedence over $\to$, and that $\to$ associates to the right, so that $\delta \to \tau \wedge \tau'$ reads as $\delta \to (\tau \wedge \tau')$ and $\delta' \to \sigma' \to \delta'' \times \sigma''$ reads as $\delta' \to (\sigma' \to (\delta'' \times \sigma''))$.
We obtain a solution of the domain equation in terms of domains of filters.

Theorem 2. *If $Th_D = Th_{D\to\mathbb{S}D}$, $Th_S = Th_{S,D}$, $Th_C = Th_{(D\times\mathbb{S}D)_\perp}$, $Th_{\mathbb{S}D} = Th_{S\to C}$ are defined by mutual induction, then $\mathcal{F}_D \cong [\mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D}]$.*

A λ_{imp}-model. To show that $\mathcal{F}_D$ is a λ_{imp}-model we have to define the following operators:

$$
\begin{array}{ll}
unit : \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D} & \star : \mathcal{F}_{\mathbb{S}D} \times \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D} \\
get_\ell : \mathcal{F}_D \to \mathcal{F}_{\mathbb{S}D} & set_\ell : \mathcal{F}_D \times \mathcal{F}_{\mathbb{S}D} \to \mathcal{F}_{\mathbb{S}D}
\end{array}
$$

This can be done as follows:

$$
\begin{array}{ll}
unit^{\mathcal{F}} F & = \mathtt{Filt}\{\sigma \to \delta\times\sigma \in \mathcal{L}_{\mathbb{S}D} \mid \delta \in F\} \\
G \star^{\mathcal{F}} F & = \mathtt{Filt}\{\sigma \to \delta''\times\sigma'' \in \mathcal{L}_{\mathbb{S}D} \mid \exists\ \delta',\sigma'.\ \sigma \to \delta\times\sigma' \in G\ \ \&\ \ \delta'\to\sigma'\to\delta''\times\sigma'' \in F\} \\
get_\ell^{\mathcal{F}}(F) & = \mathtt{Filt}\{(\langle\ell:\delta\rangle\wedge\sigma)\to\kappa \in \mathcal{L}_{\mathbb{S}D} \mid \delta\to(\sigma\to\kappa)\in F\} \\
set_\ell^{\mathcal{F}}(F,G) & = \mathtt{Filt}\{\sigma'\to\kappa \in \mathcal{L}_{\mathbb{S}D} \mid \exists\delta\in F.\ \langle\ell:\delta\rangle\wedge\sigma'\to\kappa\in G \quad \&\quad \ell\notin dom(\sigma')\}
\end{array}
$$

where $\mathtt{Filt}X$ is the least filter including the set X.

Theorem 3. *$(\mathcal{F}_D, \mathbb{S}_{\mathcal{F}_D}, [\![\cdot]\!]^{\mathcal{F}_D}, [\![\cdot]\!]^{\mathcal{F}_{\mathbb{S}D}})$ is a λ_{imp}-model.*

Type assignment system. A *typing context* is a finite set $\Gamma = \{x_1 : \delta_1, \ldots, x_n : \delta_n\}$ with pairwise distinct x_i's and with $\delta_i \in \mathcal{L}_D$ for all $i = 1,\ldots,n$; with Γ as before, we set $dom(\Gamma) = \{x_1,\ldots,x_n\}$; finally, by $\Gamma, x:\delta$ we mean $\Gamma \cup \{x:\delta\}$ for $x \notin dom(\Gamma)$. Type *judgments* are of either forms $\Gamma \vdash V : \delta$ or $\Gamma \vdash M : \tau$. Define $\ell \in dom(\sigma)$ if $\exists\delta \neq \omega_D.\ \sigma \leq_S \langle \ell : \delta\rangle$; then the rules of the type assignment system are listed below.

$$
\frac{}{\Gamma, x:\delta \vdash x : \delta}\ (var) \qquad \frac{\Gamma, x:\delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta\to\tau}\ (\lambda)
$$

$$
\frac{\Gamma \vdash V : \delta}{\Gamma \vdash [V] : \sigma \to \delta\times\sigma}\ (unit) \qquad \frac{\Gamma \vdash M : \sigma \to \delta'\times\sigma' \quad \Gamma \vdash V : \delta'\to\sigma'\to\delta''\times\sigma''}{\Gamma \vdash M \star V : \sigma \to \delta''\times\sigma''}\ (\star)
$$

$$
\frac{\Gamma, x:\delta \vdash M : \sigma\to\kappa}{\Gamma \vdash get_\ell(\lambda x.M) : (\langle\ell:\delta\rangle\wedge\sigma)\to\kappa}\ (get) \qquad \frac{\Gamma \vdash V : \delta \quad \Gamma \vdash M : (\langle\ell:\delta\rangle\wedge\sigma)\to\kappa \quad \ell\notin dom(\sigma)}{\Gamma \vdash set_\ell(V,M) : \sigma\to\kappa}\ (set)
$$

$$\frac{}{\Gamma \vdash T : \omega}\,(\omega) \qquad \frac{\Gamma \vdash T : \varphi \quad \Gamma \vdash T : \varphi'}{\Gamma \vdash T : \varphi \wedge \varphi'}\,(\wedge) \qquad \frac{\Gamma \vdash T : \varphi \quad \varphi \leq \varphi'}{\Gamma \vdash T : \varphi'}\,(\leq)$$

See [5] for an explanation of the rules of this system.

Definition 2 (Type interpretation). *Let $TypeEnv = TypeVar \to \wp(D)$ be the set of the interpretations of type variables α, ranged over by ξ; then we define the sets: $[\![\delta]\!]_\xi \subseteq D, [\![\sigma]\!]_\xi \subseteq S = [\boldsymbol{L} \to D], [\![\kappa]\!]_\xi \subseteq C = (D \times S)_\perp, [\![\tau]\!]_\xi \subseteq \mathbb{S}_D D$ by the following inductive clauses:*

1. $[\![\alpha]\!]_\xi = \xi(\alpha)$ *and* $[\![\delta \to \tau]\!]_\xi = \{f \in [D \to \mathbb{S}_D D] \mid f([\![\delta]\!]_\xi) \subseteq [\![\tau]\!]_\xi\}$,
2. $[\![\langle \ell : \delta \rangle]\!]_\xi = \{s \in S \mid s(\ell) \in [\![\delta]\!]_\xi\}$,
3. $[\![\delta \times \sigma]\!]_\xi = [\![\delta]\!]_\xi \times [\![\sigma]\!]_\xi$,
4. $[\![\sigma \to \kappa]\!]_\xi = \{g \in [S \to (D \times S)_\perp] \mid g([\![\sigma]\!]_\xi) \subseteq [\![\kappa]\!]_\xi\}$,
5. $[\![\varphi \wedge \varphi']\!]_\xi = [\![\varphi]\!]_\xi \cap [\![\varphi']\!]_\xi$, *for φ, φ' of the same sort,*
6. $[\![\omega_D]\!]_\xi = D$, $[\![\omega_S]\!]_\xi = S$, $[\![\omega_C]\!]_\xi = C = (D \times S)_\perp$, *and* $[\![\omega_{\mathbb{S}D}]\!]_\xi = \mathbb{S}_D D$.

The following definition is standard in literature, see [2] Def. 17.1.3.

Definition 3 (Semantic Satisfiability). *Let e be a term environment and ξ a type environment:*

1. $e, \xi \models^D \Gamma$ *if* $e(x) \in [\![\Gamma(x)]\!]^D_\xi$ *for all* $x \in \mathrm{dom}(\Gamma)$;
2. $\Gamma \models^D V : \delta$ *($\Gamma \models^D M : \tau$) if* $e, \xi \models^D \Gamma$ *implies* $[\![V]\!]^D e \in [\![\delta]\!]_\xi$ *($[\![M]\!]^{\mathbb{S}D} e \in [\![\tau]\!]_\xi$);*
3. $\Gamma \models V : \delta$ *($\Gamma \models M : \tau$) if $\Gamma \models^D V : \delta$ ($\Gamma \models^D M : \tau$) for all models D.*

Theorem 4 (Soundness and Completeness).

$$\Gamma \vdash V : \delta \;\Leftrightarrow\; \Gamma \models V : \delta \quad \textit{and} \quad \Gamma \vdash M : \tau \;\Leftrightarrow\; \Gamma \models M : \tau$$

The proof of soundness is by induction on the type derivation; the completeness is proved using Theorem 3 and the fact that the interpretation of a term in the filter model coincides with the set of types that can be assigned to it.

Final remarks. In the present work we gave a sketch of some of the issues that can be encountered in investigating the semantics of an untyped imperative λ-calculus and, albeit not in full details, outlined how a filter model for the state monad can be derived once the domain equation defining the specific call-by-value reflexive object is understood and dissected.
Although filter models have been extensively discussed in the literature, see e.g. [6], to our knowledge this is the first construction of such a model for an imperative lambda calculus.
In [3] we constructed a filter model for the pure computational λ-calculus, namely without operations nor constants, where the underlying monad is generic. In that work we highlighted that this construction requires some conditions on interpretation of intersection types. Here such conditions are naturally satisfied by the type theory and assignment system tailored for the state monad. An open issue is the investigation whether there is a uniform construction of filter models for calculi with algebraic operators over generic monads.

References

1. Abramsky, S.: Domain theory in logical form. Ann. Pure Appl. Log. **51**(1-2), 1–77 (1991). https://doi.org/10.1016/0168-0072(91)90065-T
2. Barendregt, H.P., Dekkers, W., Statman, R.: Lambda Calculus with Types. Perspectives in logic, Cambridge University Press (2013)
3. de'Liguoro, U., Treglia, R.: Intersection types for the computational lambda-calculus. CoRR **abs/1907.05706** (2019)
4. de'Liguoro, U., Treglia, R.: The untyped computational λ-calculus and its intersection type discipline. Theor. Comput. Sci. **846**, 141–159 (2020). https://doi.org/10.1016/j.tcs.2020.09.029, `https://doi.org/10.1016/j.tcs.2020.09.029`
5. de'Liguoro, U., Treglia, R.: Intersection types for a computational lambda-calculus with global state (2021), `https://arxiv.org/abs/2104.01358`
6. Dezani-Ciancaglini, M., Honsell, F., Alessi, F.: A complete characterization of complete intersection-type preorders. ACM Trans. Comput. Log. **4**(1), 120–147 (2003)
7. Moggi, E.: Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland (1988)
8. Pierce, B.C.: Basic category theory for computer scientists. Foundations of computing, MIT Press (1991)
9. Plotkin, G.D., Power, J.: Notions of computation determine monads. In: FOSSACS 2002. Lecture Notes in Computer Science, vol. 2303, pp. 342–356. Springer (2002). https://doi.org/10.1007/3-540-45931-6_24, `https://doi.org/10.1007/3-540-45931-6\_24`
10. Plotkin, G.D., Power, J.: Algebraic operations and generic effects. Appl. Categorical Struct. **11**(1), 69–94 (2003). https://doi.org/10.1023/A:1023064908962
11. Power, J.: Generic models for computational effects. Theor. Comput. Sci. **364**(2), 254–269 (2006)
12. Wadler, P.: Monads for functional programming. In: Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques. Lecture Notes in Computer Science, vol. 925, pp. 24–52. Springer (1995). https://doi.org/10.1007/3-540-59451-5_2