The next 700 (type-theoretical) denotational models ?

Damiano Mazza [1] and Domenico Ruoppolo [2]

[1]CNRS, UMR 7030, LIPN, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France,
damiano.mazza@lipn.univ-paris13.fr
[2]LIPN, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France,
domenico.ruoppolo@lipn.univ-paris13.fr

The use of intersection type systems in the study of denotational semantics of the untyped λ-calculus has been a fruitful line of research over the past thirty-five years. In this extended abstract we present a research project aiming at broaden the boundaries of such type-theoretical approach to semantics, possibly to other idealized programming languages. The key idea is to exploit a more abstract view on what a type system is. This view finds its root in higher-dimensional category theory.

The context

Melliès and Zeilberger recently suggested a new paradigmatic way of conceiving type refinement systems [11]. One can see any category $\mathcal{T}$ as a type system/idealized programming language, by considering every object a as a type, every morphism $t : a \to b$ as a term $x : a \vdash t : b$ intrinsically typed (*à la Church*) in a single-free-variable context, and composition $t \circ u$ as substitution $t\{u/x\}$. If one wants a more refined syntax, allowing for instance multivariable contexts $x_1 : a_1, \dots, x_n : a_n$ or some specific type constructors, then it suffices to take a more refined category $\mathcal{T}$, by adding for instance a monoidal structure, a cartesian structure, some form of closure, and so on. On the other side, when the category $\mathcal{T}$ is just a monoid this view provides an untyped term system. Melliès and Zeilberger remarked that when this *'categories-as-type systems à la Church'* correspondence is taken seriously, then *every functor $F : \mathcal{D} \to \mathcal{T}$ is a type refinement system for $\mathcal{T}$*. An object α of $\mathcal{D}$ is a type refinement of the only type a of $\mathcal{T}$ such that $F(\alpha) = a$. A morphism $\delta \in \mathcal{D}(\alpha, \beta)$ is a *derivation* of refined typing $\delta :: x : \alpha \vdash t : \beta$ for the only term $x : a \vdash t : b$ such that $F(\delta) = t$. Finally, a composition $\delta \circ \psi$ is nothing but a pasting $\delta\{\psi \,/\, x : \alpha\}$ of derivation trees. This view naturally accommodates (and actually generalizes) the common notion of subtyping: the fact that $\alpha \leq \alpha'$ (actually $\alpha \leq_\psi \alpha'$) is represented by a derivation $\psi \in \mathcal{D}(\alpha, \alpha')$ such that $F(\psi) = id_a$. (Notice that when $\mathcal{T}$ is a monoid, then a functor $F : \mathcal{D} \to \mathcal{T}$ is just a type system *à la Curry* for the untyped language $\mathcal{T}$.)

Although not formalized in [11], Melliès-Zeilberger's paradigm also allows reductions of terms $t \Rightarrow t'$, and even rewritings of corresponding typing derivations $\delta \Rightarrow \delta'$, as first-class citizens. At that purpose, one takes $\mathcal{T}$ and $\mathcal{D}$ to be 2-categories, with 2-arrows playing the role of rewritings. This approach has been undertaken by the first author in a recent joint work with Pellissier and Vial [10]. More precisely, that article advocates the idea that *a type refinement system is a 2-morphism $F : \mathcal{D} \to \mathcal{T}$ of colored 2-operads*. Considering colored operads (just 'operads' from now on), i.e. multicategories, rather than monoidal categories as in [11], is mostly a matter of taste: the visualization of programs as multi-entries terms fits very naturally in an operadic framework. In particular the free variables $x_1, x_2, \dots, x_n, \dots$ of the language $\mathcal{T}$ are nothing but names for the entrances of multimorphisms, while the horizontal composition $t \circ^i u$ of the operad corresponds to the substitution $t\{u/x_i\}$.

What is really relevant here is the **Cat**-enriched perspective: for all objects $\alpha_1, \dots, \alpha_n, \beta$ of the operad $\mathcal{D}$ such that $F(\alpha_i) = a_i$ and $F(\beta) = b$, we are given a small category $\mathcal{D}(\alpha_1, \dots, \alpha_n; \beta)$

whose arrows are derivation rewritings $\delta \Rightarrow \delta'$, a small category $\mathcal{L}(a_1, \ldots, a_n; b)$ whose arrows are term rewritings $t \Rightarrow t'$, and a functor $F_{(a_1,\ldots,a_n;b)} : \mathcal{D}(\alpha_1, \ldots, \alpha_n; \beta) \to \mathcal{T}(a_1, \ldots, a_n; b)$ between them. The functor $F_{(a_1,\ldots,a_n;b)}$ describes how every rewriting ρ of a derivation δ for a term $t = F(\delta)$

$$\delta \left(:: \quad x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash t : \beta \right) \quad \stackrel{\rho}{\Longrightarrow} \quad \delta' \left(:: \quad x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash t' : \beta \right)$$

follows the underlying reduction of the program t into the program $t' = F(\delta')$

$$\left(x_1 : a_1, \ldots, x_n : a_n \vdash t : b \right) \quad \stackrel{F_{(a_1,\ldots,a_n;b)}(\rho)}{\Longrightarrow} \quad \left(x_1 : a_1, \ldots, x_n : a_n \vdash t' : b \right).$$

Mazza, Pellissier and Vial require that such a functor satisfies three natural properties:

- fixed δ and δ' as above, different rewritings $\delta \Rightarrow \delta'$ must correspond to different underlying reductions $t \Rightarrow t'$ (categorically, this means that the functor $F_{(a_1,\ldots,a_n;b)}$ is faithful);
- only an 'empty' rewriting $\delta \Rightarrow \delta$ can correspond to the 'empty' reduction $t \Rightarrow t$ (categorically, this is a notion of *identity lifting*: if $F_{(a_1,\ldots,a_n;b)}(\rho) = id_t$ then $\rho = id_\delta$);
- if the reduction $t \Rightarrow t'$ that underlies $\rho : \delta \Rightarrow \delta'$ factorizes as $t \Rightarrow u \Rightarrow t'$, then also ρ factorizes accordingly (categorically, this gives a notion of *factorization lifting*).

These three properties, which we adopt here, correspond to a notion of *Niefield fibration*.

Definition (Mazza, Pellissier, Vial [10])**.** *A* type system *is a* 2*-morphism* $F : \mathcal{D} \to \mathcal{T}$ *of (colored)* 2*-operads such that for all objects* $\alpha_1, \ldots, \alpha_n, \beta$ *of* $\mathcal{D}$ *the functor* $F_{(a_1,\ldots,a_n;b)}$ *is a Niefield fibration.*

Provided with a suitable notion of morphism, the collection of all such type systems for a fixed 2-operad $\mathcal{T}$ form a category $\mathbf{TS}(\mathcal{T})$. Through a kind of construction *à la Grothendieck*, it is proved in [10] that $\mathbf{TS}(\mathcal{T})$ is equivalent to a quite different category. This equivalence provides an alternative formalization of the very same notion of type systems, as described here below.

Firstly, one defines a bioperad $\mathcal{R}el\mathcal{D}ist$. Its objects are all small categories. A morphism $\mathcal{A}_1, \ldots, \mathcal{A}_n \to \mathcal{B}$ in $\mathcal{R}el\mathcal{D}ist$ is a *relational distributor*, i.e. a notion of profunctor $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n \times \mathcal{B}^{op} \to \mathbf{Rel}$ that goes into the category **Rel** of sets and relations (rather then the usual **Set**). We omit the description of the 2-arrows, which are of course suitable natural transformations. Then, the following is an alternative characterization of the definition above

Definition (Mazza, Pellissier, Vial [10])**.** *A* type system *is a lax morphism* $\mathscr{F} : \mathcal{T} \to \mathcal{R}el\mathcal{D}ist$ *of bicategories.*

Although being less intuitive, this characterization has the following nice feature: it explicitly organizes the subtyping relations as categories. Remember that every object a of $\mathcal{T}$ represents a type *à la* Church of the programming language $\mathcal{T}$. Then the objects of $\mathscr{F}(a)$ are exactly the refinement types of a. A morphism $f \in \mathscr{F}(a)(\alpha, \beta)$ witnesses the fact that α is a subtyping of β, namely $\alpha \leq_f \beta$. For a term $t \in \mathcal{T}(a_1, \ldots, a_n; b)$, the corresponding profunction

$$\mathscr{F}(t) \ : \ \mathscr{F}(b_1) \times \cdots \times \mathscr{F}(b_n) \times \mathscr{F}(a)^{op} \ \to \ \mathbf{Rel}$$

has the following meaning. Given $(\beta_1, \ldots, \beta_n, \alpha) \in Ob(\mathscr{F}(b_1)) \times \cdots \times Ob(\mathscr{F}(b_n)) \times Ob(\mathscr{F}(a))$, the set $\mathscr{F}(t)(\vec{\beta}, \alpha)$ contains all the typing derivations $\delta :: x_1 : \beta_1, \ldots, x_n : \beta_n \vdash t : \alpha$ that the system $\mathscr{F}$ gives to the term $x_1 : b_1, \ldots, x_n : b_n \vdash t : a$. As concerns the action of $\mathscr{F}(t)$ on morphisms $(f_1, \ldots, f_n) \in \mathscr{F}(b_1)(\beta_1, \beta_1') \times \cdots \times \mathscr{F}(b_n)(\beta_n, \beta_n')$ and $f \in \mathscr{F}(a)^{op}(\alpha, \alpha') = \mathscr{F}(a)(\alpha', \alpha)$, the relation $\mathscr{F}(t)(\vec{f}, f) \subseteq \mathscr{F}(t)(\vec{\beta}, \alpha) \times \mathscr{F}(t)(\vec{\beta'}, \alpha')$ contains pairs (δ, δ') such that the derivation δ differs from δ' only in the application of the subtyping rules $\vec{f}, f$ in a certain way. As an example, let $\mathcal{T}$ be the 2-monoid representing the untyped λ-calculus. Let us take as $\mathscr{F}$ a refinement type system for $\mathcal{T}$ allowing arrow types (denoted by $\Rightarrow$) and the usual subtyping rule

$$\frac{\Gamma \vdash t : \psi \qquad \psi \leq_f \varphi}{\Gamma \vdash t : \varphi} \ \text{sub}$$

(For instance, $\mathscr{F}$ could be one of the traditional intersection type systems.) Suppose that this system has three specific types α, β, γ provided with the subtyping morphism $\beta \leq_f \gamma \Rightarrow \alpha$. Formally this means that there is a morphism $f : \beta \to \gamma \Rightarrow \alpha$ in the category $\mathscr{F}(*)$, where $*$ is the only object of $\mathcal{T}$. Let us see a pair (δ, δ') in the relation $\mathscr{F}(x_1x_2)\big(f, id_\gamma, id_\alpha\big) \subseteq \mathscr{F}(x_1x_2)\big(\beta, \gamma, \alpha\big) \times \mathscr{F}(x_1x_2)\big(\gamma \Rightarrow \alpha, \gamma, \alpha\big)$. Consider the derivation $\delta' \in \mathscr{F}(x_1x_2)\big(\gamma \Rightarrow \alpha, \gamma, \alpha\big)$ given as

$$\frac{x_1 : \gamma \Rightarrow \alpha \vdash x_1 : \gamma \Rightarrow \alpha \qquad x_2 : \gamma \vdash x_2 : \gamma}{x_1 : \gamma \Rightarrow \alpha\,,\, x_2 : \gamma \vdash x_1x_2 : \alpha}$$

Then a possible $\delta \in \mathscr{F}(x_1x_2)\big(\beta, \gamma, \alpha\big)$ such that $(\delta, \delta') \in \mathscr{F}(x_1x_2)\big(f, id_\gamma, id_\alpha\big)$ is

$$\frac{\dfrac{x_1 : \beta \vdash x_1 : \beta \qquad \beta \leq_f \gamma \Rightarrow \alpha}{x_1 : \beta \vdash x_1 : \gamma \Rightarrow \alpha}\,{\scriptstyle \text{sub}} \qquad x_2 : \gamma \vdash x_2 : \gamma}{x_1 : \beta\,,\, x_2 : \gamma \vdash x_1x_2 : \alpha}$$

The problem

Since the introduction of the first filter model by Barendregt, Coppo and Dezani [1] in the early '80s, intersection type systems have become a significant tool for denotational semantics of the untyped λ-calculus. There is indeed a form of Stone duality between certain classes of models of the λ-calculus and certain classes of intersection type systems. Such a correspondence tells us that the interpretation $[\![M]\!]$ of a λ-term M in one of these models can be characterized in a type-theoretical form, following the intuition: $\vdash M : \alpha \iff \alpha \in [\![M]\!]$. The inference rules of the typing system reflect the structure of the corresponding model, and *vice versa* the elements of the model can be seen as some specific sets of types. The most known instance is the one of filter models [2, 13], a large class of models living in Scott-continuous semantics. In their case the elements of the model can be defined as filters on the set of types ordered by the subtyping preorder. Another class of Scott-continuous models provided with a Stone duality is the class of K-models, where K stands for Krivine [8]. In their case the elements of the model can be defined as initial segments on the set of types ordered by the subtyping preorder. (Actually, the initial segments can even be replaced by anti-chains, as clarified in [3].) In recent years a similar form of Stone duality has appeared for relational semantics, a resource-sensitive interpretation of the λ-calculus coming from the study of linear logic [6]. Many relational models can be characterized by means of relevant (i.e., without weakening) intersection type systems, where the intersection is actually a non-idempotent operator (corresponding to the tensor $\otimes$ of linear logic). Typically, in these models the subtyping is just an equivalence relation. The idea, already present in [5], is that in the absence of idempotency and partial orders the type $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta$ can be seen as a relation associating the multiset $[\alpha_1, \ldots, \alpha_n]$ with the element β. Paolini, Ronchi and Piccolo called these models strongly linear relational models [12]. The second author, together with Manzonetto and Breuvart, extensively studied a vast class of them, called relational graph models [14, 4]. In their case the elements of the model can just be seen as equivalence classes $[\alpha]_\simeq$ of types modulo the subtyping relation $\simeq$.

Inspired by the type-theoretical semantics reminded above, we ask the following question. Consider a type system in the abstract sense depicted in the previous section, no matter if as a 2-morphism $F : \mathcal{D} \to \mathcal{T}$ or in the form of a lax morphism $\mathscr{F} : \mathcal{T} \to \mathcal{R}el\mathcal{D}ist$. Suppose that this system enjoys subject expansion and subject reduction w.r.t. the notion of reduction carried by the 2-arrows of the operad $\mathcal{T}$. (In the case of $F : \mathcal{D} \to \mathcal{T}$ this hypothesis corresponds to a lifting/op-lifting property, whereas in the case of $\mathscr{F} : \mathcal{T} \to \mathcal{R}el\mathcal{D}ist$ it means that the target bicategory of $\mathscr{F}$ can be specialized to a certain sub-bicategory of $\mathcal{R}el\mathcal{D}ist$.) *Is it possible to define a denotational model of the programming language $\mathcal{T}$ via this typing system?*

In the semantics of the λ-calculus seen above, which we use as blueprints for our problem, the models are always built in one way or another from the set of refinement types ordered by the subtyping relation. As a consequence, we find more natural to use here the formalization $\mathscr{F} : \mathcal{T} \to \mathcal{R}el\mathcal{D}ist$, which groups together refinement types and subtyping morphisms in explicit small categories $\mathscr{F}(a)$.

So, given a type system $\mathscr{F} : \mathcal{T} \to \mathcal{R}el\mathcal{D}ist$, we wish to define a 2-morphism $[\![\ ]\!]_{\mathscr{F}} : \mathcal{T} \to \mathcal{O}$, where the 2-operad $\mathcal{O}$ represents the mathematical universe in which the interpretations live. (The natural candidate for $\mathcal{O}$ is **Set** itself, seen as a 2-operad.) The 2-morphism $[\![\ ]\!]_{\mathscr{F}} : \mathcal{T} \to \mathcal{O}$ should satisfy the property that every 2-arrow $\rho : t \Rightarrow t'$ of $\mathcal{T}$ is sent to the identity of $[\![t]\!]_{\mathscr{F}} = [\![t']\!]_{\mathscr{F}}$, since turning program reduction into an invariant of the model is the key idea of denotational semantics. (At very least $[\![\rho]\!]_{\mathscr{F}}$ should be an automorphism in $\mathcal{O}$.)

The blueprint semantics suggest that for a given type a of $\mathcal{T}$, its interpretation $[\![a]\!]_{\mathscr{F}}$ must have something to do with the representable presheaves $\mathscr{F}(a)(-, \alpha) : \mathscr{F}(a)^{op} \to \mathbf{Set}$ and similar notions [9].. Intuitively, such a functor tells us if and *in how many ways* any refinement type β in $\mathscr{F}(a)$ is a subtype of α.

Perspectives

Achieving the definition of the 2-morphism $[\![\ ]\!]_{\mathscr{F}} : \mathcal{T} \to \mathcal{O}$ in the most general setting presented here would be a breakthrough. It could allow to extend to other term rewriting systems/idealized programming languages a type-theoretical approach to semantics so far only confined to the untyped λ-calculus, to our knowledge.

We would be satisfied to solve the problem even just for specific case where $\mathcal{T}$ is the 2-operad representing the untyped λ-calculus. That could possibly help us in finding new denotational models of the λ-calculus, by exploiting the fact of having a category, rather then just a preorder, of types and subtypings. Moreover, such a solution would allow us to explore the connection with Hyland's recent algebraic take on the untyped λ-calculus [7], where the language (and more generally every λ-theory) is seen as a Lawvere theory, equivalently as an operad $\mathcal{T}$, whereas its models are defined as algebras over that Lawvere theory, i.e. exactly as functors $[\![\ ]\!] : \mathcal{T} \to \mathbf{Set}$.

References

[1] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983.

[2] H.P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[3] F. Breuvart. On the characterization of models of $\mathcal{H}^*$: the semantical aspect. *Logical Methods in Computer Science*, 12(2), 2016.

[4] Flavien Breuvart, Giulio Manzonetto, and Domenico Ruoppolo. Relational graph models at work. *to appear in Logical Methods in Computer Science*, abs/1703.10382, 2017.

[5] D. de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. Draft available at `http://arxiv.org/abs/0905.4251`, 2009.

[6] J.-Y. Girard. Linear logic. *Th. Comp. Sci.*, 50:1–102, 1987.

[7] J. M. E. Hyland. Classical lambda calculus in modern dress. *Mathematical Structures in Computer Science*, 27(5):762–781, 2017.

[8] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood, New York, 1993. Translated from the ed. Masson, 1990, French original.

[9] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic a First Introduction to Topos Theory*. Springer, New York, NY, 1992.

[10] Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *PACMPL*, 2(POPL):6:1–6:28, 2018.

[11] Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 3–16, 2015.

[12] L. Paolini, M. Piccolo, and S. Ronchi Della Rocca. Essential and relational models. *Mathematical Structures in Computer Science*, 2015. Doi:10.1017/S0960129515000316.

[13] S. Ronchi Della Rocca and L. Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[14] D. Ruoppolo. *Relational Graph Models and Morris's Observability: resource-sensitive semantic investigations on the untyped λ-calculus*. Thése de doctorat, Université Paris 13, 2016.