The Characterization of Convergence in Computational λ-Calculi via Intersection Types

Ugo de'Liguoro[1] and Riccardo Treglia[2,*]

[1] Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Torino, Italy
ugo.deliguoro@unito.it

[2] DISI, Università di Bologna, Mura Anteo Zamboni,7, I-40126, Bologna
riccardo.treglia@unibo.it

λ-calculus and Monads. Since Strachey and Scott's work in the 60's, λ-calculus and denotational semantics, together with logic and type theory, have been recognized as the mathematical foundations of programming languages. Nonetheless, aspects of actual programming languages have shown to be quite hard to treat, at least with the same elegance as the theory of recursive functions and of algebraic data structures. In [8] Moggi proposed a unified framework to reason about λ-calculi embodying various kinds of effects, including side-effects, that has been used by Wadler (i.e. in [11]) to cleanly implement non-functional aspects into Haskell, a purely functional programming language. Moggi's approach is based on the categorical notion of *computational monad*: instead of adding impure effects to the semantics of a pure functional calculus, effects are subsumed by the abstract concept of "notion of computation" represented by the monad T. The basic idea is to distinguish among *values* of some type D and *computations* over such values, the latter having type TD. In [4] we considered an untyped computational λ-calculus, called *computational core* in [7], $\lambda_{©}$, where T is some generic monad and $D \cong D \to TD$. Consequently, $\lambda_{©}$ is a calculus of two sorts of expressions: $\mathit{Val} : V, W ::= x \mid \lambda x.M$ and $\mathit{Com} : M, N ::= [V] \mid M \star V$, where x ranges over a denumerable set Var of variables. Differently from the equational theory in [8], we introduce an operational semantics based on a reduction relation over computation terms and a type assignment system of intersection types such that types are invariant under conversions and convergent computations are characterized by having non-trivial types in the system. Monadic operations of merging values into computations, i.e. the *unit* of the monad T, and of extension model how morphisms from values to computations compose, but do not tell anything about how the computational effects are produced. In the theory of *algebraic effects* (i.e. in [10]), it is shown under which conditions effect operators live in the category of algebras of a computational monad, which is isomorphic to the category of models of certain equational specifications. Toward studying how to extend the type-theoretic characterization of convergence in $\lambda_{©}$ when computations are endowed with operators for algebraic effects, in [6, 5] we have considered the case of the state monad, together with (denumerably many) operations get_ℓ and set_ℓ, indexed over an infinite set of locations, to access a global store. The syntax of the calculus, named *imperative λ-calculus*, λ_{imp}, is then obtained by the extending computations by $\mathit{Com}_{\mathbb{S}} : M, N ::= [V] \mid M \star V \mid get_\ell(\lambda x.M) \mid set_\ell(V, M)\ (\ell \in \mathbf{L})$.

Intersection type theory. Among the various type systems of intersection types, we adopt the BCD system in [2], which is equipped with a subtyping pre-order axiomatizing an inf-semilattice of types with the type ω as the top element, where ω is the type of any term. The characteristic of the BCD system is type invariance under both reduction and expansion of terms, which is at the heart of the filter model construction, where the meaning of a term is

*Speaker.

fully characterized by the set of its types. As the subsequent developments have shown, relevant properties of pure λ-terms can be characterized via their types in BCD and related type systems, like weak and strong normalizability, as well as reducing to head normal form. The aim of our study is to achieve similar results in the case of computational λ-calculi, which seems non-trivial given the intrinsic complexity of calculi with effects, and the severe limitations introduced in [3], for example, especially when using intersection types for typing references and side-effects. However, key properties of BCD like systems can be recovered in the setting of computational λ-calculi, provided that the type system is constructed on the basis of the denotational semantics of the calculus, exploiting Abramsky's idea of domain logic [1]. The process of synthesizing a type system from semantics is sketched in [5] in the case of λ_{imp}. Once the type system has been properly constructed, the characterization of convergence can be obtained via the standard technique of Tait's computability.

Results. We see the computational λ-calculi and, in particular, $\lambda_{©}$as a generalization of Plotkin's call-by-value λ-calculus [9]. In this perspective, a "value" is an abstraction, and a term converges if it reduces to a value. In the case of $\lambda_{©}$, however, reduction is among computations only and never produces a value. Adapting the convergence predicate, we say that a computation M converges to a value V, written $M \Downarrow V$ if M reduces to the trivial computation $[V]$. Then we show that in the type discipline obtained out the "logical" description of $D \simeq D \to TD$ a computation M converges if and only if M can be typed by a type τ which is a proper subtype of ω, hence non-trivial. In the case of the state monad such a result can be strengthened since a single, non-trivial type is sufficient for the characterization. Such a result, which is new for calculi with side-effects, encompasses known results about static analysis of imperative programs, since a convergent computation in λ_{imp} cannot depend on reading from uninitialised locations.

References

[1] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Log.*, 51(1-2):1–77, 1991.

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

[3] M. Dezani-Ciancaglini, P. Giannini, and S. Ronchi Della Rocca. Intersection, universally quantified, and reference types. *Computer Science Logic, 23rd international Workshop, CSL 2009. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2009.

[4] U. de'Liguoro and R. Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[5] U. de'Liguoro and R. Treglia. From semantics to types: the case of the imperative lambda-calculus. Proceedings 37th Conference on *MFPS 2021*, volume 351 of *Electronic Proceedings in Theoretical Computer Science*, pages 168–183. Open Publishing Association, 2021.

[6] U. de'Liguoro and R. Treglia. Intersection types for a λ-calculus with global store. *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming 2021*, pages 5:1–5:11. ACM, 2021.

[7] C. Faggian, G. Guerrieri, U. de'Liguoro, and R. Treglia. On reduction and normalization in the computational core. *CoRR*, abs/2104.10267, 2021.

[8] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[9] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

[10] J. Power. Generic models for computational effects. *Theor. Comput. Sci.*, 364(2):254–269, 2006.

[11] P. Wadler. Monads for functional programming. In *Adv. Fun. Prog., First International Spring School on Adv. Fun. Prog. Tech.*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995.