Intersection Types for a Computational λ-Calculus with Global State

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

[1] Università di Torino, C.so Svizzera 185, 10149 Torino, Italy
ugo.deliguoro@unito.it
[2] Università di Torino, C.so Svizzera 185, 10149 Torino, Italy
riccardo.treglia@unito.it

Abstract

We study the semantics of an untyped λ-calculus equipped with operators representing read and write operations from and to a global state. We adopt the monadic approach to model side effects and treat read and write as algebraic operations over a computational monad. We introduce an operational semantics and a type assignment system of intersection types, and prove that types are invariant under reduction and expansion of term and state configurations, and characterize convergent terms via their typings.

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, there are aspects of actual programming languages that 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; a prominent case is surely side-effects.

In [Mog91] Moggi proposed a unified framework to reason about λ-calculi embodying various kinds of effects, including side-effects, that has been used by Wadler [Wad92, Wad95] 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. To a domain A of values it is associated the domain TA of computations over A, that is an embellished structure in which A can be merged, and such that any morphism f from A to TB extends by a universal construction to a map f^T from TA to TB.

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* [PP02, PP03, Pow06], Plotkin and Power have 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, namely varieties in the sense of universal algebra [HP07].

In [dT20] we have considered an untyped computational λ-calculus with two sorts of terms: *values* denoting points of some domain D, and *computations* denoting points of TD, where T is some generic monad and $D \cong D \to TD$. The goal was to show how such a calculus can be equipped with an operational semantics and an intersection type system, such that types are invariant under reduction and expansion of computation terms, and convergent computations are characterized by having non-trivial types in the system.

Here, we extend our approach and consider a variant of the *state monad* from [Mog91] and a calculus with two families of operators, indexed over a denumerable set of *locations*: $get_\ell(\lambda x.M)$ reading the value V associated to the location ℓ in the current *state*, and binding x to V in M;

$set_\ell(V, M)$ which modifies the state assigning V to ℓ, and then proceeds as M. This calculus, with minor notational differences, is called *imperative λ-calculus* in [Gav19].

As a first step, we construct a domain D that is isomorphic in a category of domains to $D \to \mathbb{S}\, D$, where $\mathbb{S}$ is the *monad of partiality and state* from [DGL17]. Then, to define the operational semantics, we consider an algebra of states which is parametric in the values, that are denoted by value-terms of the calculus. *State terms* are equated by a theory whose axioms are standard in the literature (see e.g. [Mit96], chap. 6) and are essentially, albeit not literally, the same as those ones for global state in [PP02].

Operational semantics is formalized by the evaluation or *big-step relation* $(M, s) \Downarrow (V, t)$, where M is a (closed) computation, V a value and s, t state-terms. Equivalently, we define in the SOS style a reduction or *small-step relation* $(M, s) \longrightarrow (N, t)$ among pairs of computations and state-terms, which we call *configurations*, such that $(M, s) \Downarrow (V, t)$ if and only if $(M, s) \xrightarrow{*} ([V], t)$, where $[V]$ is the computation trivially returning V.

Types and type assignment system are derived from the domain equation defining D and $\mathbb{S}\, D$, following the method of domain logic in [Abr91]. Type and typing rule definitions are guided along the path of a well understood mathematical method, which indicates both how type syntax and the subtyping relations are constructed and how to shape type assignment rules. The so obtained system is an extension of Curry style intersection type assignment system: see [BDS13] Part III.

The first result we obtain is that in our system types are invariat under reduction and expansion of configurations. This is the key step to the main theorem of this work, that is the characterization of convergence. We say that a program, namely a closed computation term, *converges* if it evaluates to a value and a final state, whatever the initial state is.

In analogy with the lazy lambda-calculus, where a term converges if and only if is typable by $\omega \to \omega$ in a suitable intersection type system, we show that a closed M converges if and only if it is typable by $\omega \to \omega \times \omega$. Consequently, type-checking in our system is undecidable.

The article is available on arXiv: [dT21].

References

[Abr91] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Log.*, 51(1-2):1–77, 1991.

[BDS13] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types.* Perspectives in logic. Cambridge University Press, 2013.

[DGL17] U. Dal Lago, F. Gavazzo, and P. B. Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[dT20] U. de'Liguoro and R. Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[dT21] U. de'Liguoro and R. Treglia. Intersection types for a computational lambda-calculus with global state. https://arxiv.org/abs/2104.01358, 2021.

[Gav19] F. Gavazzo. *Coinductive Equivalences and Metrics for Higher-order Languages with Algebraic Effects.* PhD thesis, University of Bologna, Italy, Aprile 2019.

[HP07] M. Hyland and J. Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electron. Notes Theor. Comput. Sci.*, 172:437–458, 2007.

[Mit96] J.C. Mitchell. *Foundations for Programming Languages.* MIT Press, Cambridge, MA, 1996.

[Mog91] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[Pow06] J. Power. Generic models for computational effects. *Theor. Comput. Sci.*, 364(2):254–269, 2006.

[PP02] G. D. Plotkin and J. Power. Notions of computation determine monads. In *FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[PP03] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003.

[Wad92] P. Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1992*, pages 1–14. ACM Press, 1992.

[Wad95] P. Wadler. Monads for functional programming. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995.