The Geometry of Approximations in Programming Languages

Ph.D. Thesis Proposal

Damiano Mazza
CNRS, LIPN, Université Sorbonne Paris Nord

1 Context

In the last couple of decades, the idea that a programming language may be approximated by means of a (multi)linear calculus has seen an increasing number of applications. The original, and perhaps best known example is the so-called Taylor expansion of the untyped λ-calculus by means of the resource λ-calculus introduced by Ehrhard and Regnier [ER06, ER08]. This was the starting point of a long series of results, on semantics [dC18b], pure λ-calculus [BM20] and probabilistic computation [ETP14, TAO17], just to give a brief, far from exhaustive list of examples.

Another important source of applications is the link of approximations with the theory of intersection types. First discovered by de Carvalho for the non-idempotent case [dC07], this was later extended by myself and my Ph.D. students to cover the idempotent case as well, in great generality [MPV18]. In this way, de Carvalho's idea to use intersection types for inferring exact bounds on the running time of programs [dC18a] may be extended to wider contexts, allowing for analysis of memory usage [ADLV20, ADLV21a, ADLV21b], analyzing concurrent programs [DLdVMY19], or going as far as proving the Cook-Levin theorem using type systems [Maz17].

I recently proposed an axiomatic framework for approximations in programming languages [Maz21], which we are starting to develop with my Ph.D. student Boris Eng. Although the exact formulation requires a bit of category theory, the basic idea may be explained informally as follows. An approximation relation $t \sqsubset M$ between an approximating program t and a program M should induce a sort of "adjunction" between computations and approximated computations: for every program M and approximation u,

$$\frac{M \text{ evaluates to } N \text{ such that } u \sqsubset N}{\text{there exists } t \sqsubset M \text{ such that } t \text{ evaluates to } u} \text{ iff}$$

or, diagrammatically,

$$\begin{array}{ccc} u & & t \longrightarrow u \\ \sqcap & \text{iff} & \sqcap \\ M \longrightarrow N & & M \end{array} \tag{1}$$

where arrows represent evaluation of programs. The intuition is simple: if we consider approximations as pieces of information, and read $t \sqsubset M$ as "M contains information t", an

approximation relation ensures that a program M evaluates to something containing a piece of information u iff an approximation of M evaluates to u itself. Notice the resemblance with topological continuity here.

2 Idea

Although the axiomatic framework nicely encapsulates all instances of program approximations known so far, as well as others which had not been hitherto considered, it does not shed any light on an important, albeit informal question: *where do approximations come from*?

The starting point of this thesis proposal is the idea that, in the context of programming languages, approximations have a geometric origin. More precisely, an approximation relation $t \sqsubset M$ should arise from the existence of some kind of "*étale map*" from t to M. The notion of étale map is well-known in several geometric contexts (smooth manifolds, schemes, toposes...) as a suitable reformulation of the notion of *local homeomorphism* from topology.

The above intuition results from looking at what happens in program approximations : in all notions of approximations, $t \sqsubset M$ precisely when the syntactic constructs of t map onto similar syntactic constructs of M, respecting the neighboring structure and, moreover, for each syntactic construct of M, there may be several corresponding constructs in t. This seems to indicate that t is an "étalé space" over M, in a sense to be made precise.

Notice that, for the above viewpoint to make sense, approximating and approximated programs must live in the same world, otherwise one would not be able to speak of maps between them. This may seem surprising at first: for example, in the primordial example of Ehrhard and Regnier's Taylor expansion [ER06, ER08], approximations are resource λ-terms and approximated programs are λ-terms, and these live in two distinct calculi with distinct operational semantics. However, as shown in my work on the infinitary affine λ-calculus [Maz12], it is possible to see a usual program M as an infinitary program of the same of sort of the approximating programs t, so there is no inconsistency in taking them as inhabitants of the same world; using distinct programming languages is more of a convenience than a mathematical necessity.

Pushing the above idea further, we are led to wonder: if approximations are étale maps, maybe evaluations themselves (the horizontal arrows in the diagram (1)) should be seen as maps. By taking seriously the idea that a map takes a syntactic construct to a construct of the same form, we see that program evaluation (in λ-calculus-like languages or, more generally, in rewriting systems) induces a map *in the opposite direction* of the evaluation itself. This is exemplified by so-called β-reduction, the fundamental evaluation step of the λ-calculus:

$$(\lambda x.M)N \ \to \ M[N/x],$$

where $M[N/x]$ denotes the term M in which every free occurrence of the variable x is replaced with a copy of the program N. In this rule, every syntactic construct on the right hand side may be mapped to a unique syntactic construct of the same kind on the left hand side, defining a map (again, in a sense to be made precise) $\rho : M[N/x] \to (\lambda x.M)N$ (this is implicitly used in rewriting theory, where an inverse image of a syntactic construct c via ρ is known as a *residue* of c).

If we redraw diagram (1) according the above ideas, we obtain

(2)

where now arrows represent maps, the vertical ones being étale and the horizontal ones corresponding to evaluation. This looks like a factorization theorem of some sort.

3 Plan

The main objective of this thesis is to transform the above idea into a precise theory. Here is a rough plan of how it may be developed:

- formally define the notion of "map" between programs, in the context of the pure λ-calculus at first, so that approximations correspond to suitable étale maps and evaluations to other particular maps, and such that the desired factorization theorem holds.
- Use the above framework to give a geometric explanation of program approximations. For example, Ehrhard and Regnier's Taylor expansion of a λ-term M should appear as an "étale cover" of M by means of linear approximations.
- Investigate the notion of étale cover more systematically, in particular understanding whether this yields a site (in the sense of Grothendieck) on the category of programs and maps, and then study what kind of generalized programs correspond to the sheaves on that site or, in a more speculative direction, whether cohomological methods arising from this setting have any meaning in programming languages.
- Extend the approach outside of the pure λ-calculus, for example to typed languages, or to languages with side effects (non-determinism, probability...). A particularly interesting direction is that of concurrent programming: the work [DLdVMY19] on intersection types showed that the notion of approximation may be fruitfully brought to this setting, but it barely scratched the surface of what seems to be possible.

4 Required Skills

Albeit grounded in computer science applications, the present thesis has a relatively high mathematical content and requires a student with solid background in category theory, basic topology and, if possible, some acquaintance with sheaves in the general context of Grothendieck topologies.

References

[ADLV20] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The machinery of interaction. In *Proceedings of PPDP*, pages 4:1–4:15, 2020.

[ADLV21a] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The (in)efficiency of interaction. *Proceedings of the ACM on Programming Languages*, 5(POPL), 2021.

[ADLV21b] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The space of interaction (long version). *CoRR*, abs/2104.13795, 2021.

[BM20] Davide Barbarossa and Giulio Manzonetto. Taylor subsumes scott, berry, kahn and plotkin. *Proc. ACM Program. Lang.*, 4(POPL):1:1–1:23, 2020.

[dC07] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. Ph.d. thesis, Université de la Méditerranée – Aix-Marseille 2, 2007.

[dC18a] Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.*, 28(7):1169–1203, 2018.

[dC18b] Daniel de Carvalho. Taylor expansion in linear logic is invertible. *Log. Methods Comput. Sci.*, 14(4), 2018.

[DLdVMY19] Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *Proc. ACM Program. Lang.*, 3(POPL):7:1–7:29, 2019.

[ER06] Thomas Ehrhard and Laurent Regnier. Böhm trees, krivine's machine and the taylor expansion of lambda-terms. In *Proceedings of CiE*, pages 186–197, 2006.

[ER08] Thomas Ehrhard and Laurent Regnier. Uniformity and the taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.

[ETP14] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proceedings of POPL*, pages 309–320, 2014.

[Maz12] Damiano Mazza. An infinitary affine lambda-calculus isomorphic to the full lambda-calculus. In *Proceedings of LICS*, pages 471–480, 2012.

[Maz17] Damiano Mazza. *Polyadic Approximations in Logic and Computation*. *Habilitation* thesis, Université Paris 13, 2017.

[Maz21] Damiano Mazza. An axiomatic notion of approximation for programming languages and machines. `https://lipn.univ-paris13.fr/~mazza/papers/ApxAxiom.pdf`, 2021.

[MPV18] Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proceedings of the ACM on Programming Languages*, 2(POPL:6), 2018.

[TAO17] Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *Proceedings of LICS*, pages 1–12, 2017.