Affine Functional Programs as Higher-order Boolean Circuits

Damiano Mazza*
CNRS, LIPN, Université Paris 13, Sorbonne Paris Cité
Damiano.Mazza@lipn.univ-paris13.fr

Implicit computational complexity. Loosely speaking, the aim of implicit computational complexity (ICC) is to replace clocks (or other explicit resource bounds) with certificates. For example, if we consider polynomial time computation, the idea is to define a structured programming language whose programs are guaranteed to be polytime by construction, *e.g.*, because they are well-typed, not because their execution is artificially stopped after a polynomial number of steps (which is the "explicit" definition used in computational complexity). At the same time, such a programming language must be expressive enough so that every polynomial time function may be somehow implemented. Notable early examples of such methodology are the work of Bellantoni and Cook [BC92], Leivant and Marion [LM93], and Jones [Jon99].

Although there is an annual international workshop on the topic,[1] ICC is a niche research subject which rarely appears in mainstream programming languages conferences. Nevertheless, it is thematically relevant, both in terms of techniques (type systems, proof theory, semantics...) and potential applications (mostly program analysis [JHLH10, DLG11], but people are starting to develop applications to the verification of cryptosystems [NZ15]).

Non-uniform computation. To keep the definition of cost model as simple and incontrovertible as possible, the "default" models used in computational complexity (Turing machines, RAMs...) are extremely low level. This has the effect of virtually erasing the fundamental structures present in data and algorithms: in Turing machines, there are no data types, just strings; and all algorithms, albeit containing recursion, control structures, subroutines and such, become sets of tuples.

Although complexity theorists have been able to develop their research seemingly unimpeded by the above remark, in truth the theory has been most successful in proving non-trivial lower bounds when algorithms are expressed in computational models offering more structure than just sets of tuples or assembly-like instructions. Decision trees, Boolean formulas and circuits are typical examples of such models (see for instance Part II of [AB09]). These all share the characteristic of being strongly finitary, in the sense that one program may only handle finitely many inputs (usually, all inputs of a given size). In order to compute a function in the usual sense (defined on data of arbitrary size), one has to consider infinite families of such programs (one for each input size). To stay within the realm of computable functions, one must impose a *uniformity* constraint, *i.e.*, that the members of the family are generated by a single "usual" program (*e.g.* a Turing machine, perhaps with bounded resources). There are well known connection theorems between, for instance, time- or space-bounded Turing machines and uniform families of circuits with restrictions on their size or depth (for instance, deterministic polytime Turing machines correspond to uniform polysize circuits).

However, the role of uniformity in practice is unclear: most lower bound proofs simply ignore it, *i.e.*, they apply to non-uniform families. Indeed, the majority of existing proof techniques exploit the local structure of each member of the family, regardless of the other members. The fact that the family globally forms a coherent whole (*i.e.*, an algorithm in the usual sense instead of infinitely many unrelated micro-algorithms) is surprisingly unhelpful.

Affine functional programs as higher-order Boolean circuits. We propose to discuss the idea of non-uniformity from the point of view of functional programming and ICC, following recent work by the author (and co-authors) on

*Partially supported by project COQUAS ANR-12-JS02-006-01.

[1] Developments in Implicit Computational Complexity (DICE), http://perso.ens-lyon.fr/patrick.baillot/DICE/. Incidentally, the author is the PC chair of this year's edition.

an infinitary affine λ-calculus [Maz12, Maz14, MT15]. The starting point is the "equation"

$$\frac{\text{linear/affine } \lambda\text{-terms}}{\lambda\text{-terms}} \quad = \quad \frac{\text{Boolean circuits}}{\text{Turing machines}}$$

The relationship between Boolean circuits and affine calculi is not new [Mai04, Ter04]. The key novelty of the present line of work is the idea of *continuous approximation*: every usual λ-term M induces an infinite family of affine λ-terms converging to it (technically, they converge to the infinitary affine representation of M, in a suitable topology [Maz12]). Additionally, term evaluation (*i.e.*, β-reduction) is continuous, which has the following consequence. Suppose that M decides a language: for every Church string $\underline{w}$, $M\underline{w} \to^* \underline{b}$ with $\underline{b}$ a Boolean. Then, continuity guarantees that there is an affine approximation $t_{|w|}$ of M (where $|w|$ is the length of w) such that $t_{|w|}\ \underline{w} \to^* \underline{b}$. If M is compared to a Turing machine, then $t_{|w|}$ is akin to the circuit approximating the behavior of the machine on inputs of size $|w|$. Using this fact, one may re-establish certain fundamental complexity-theoretic results, such as the Cook-Levin theorem, in a purely functional setting. Moreover, working along these lines one may find implicit characterizations of certain standard non-uniform complexity classes (namely, non-uniform polytime P/poly [Maz14] and non-uniform logspace L/poly [MT15]). Capturing such complexity classes is a novelty in ICC.

There are two main perspectives which we wish to put forward:

- within the theory of programming languages, one may wonder whether the notion of continuous approximation has applications in static analysis and control flow analysis (CFA). Essentially, ICC is a kind of static analysis: runtime properties of programs (their membership to a complexity class) are inferred at compile-time. Does the non-uniform view provide new methods for refining ICC to more sophisticated analyses, *e.g.* of the kind provided in [DLG11, GS14] (which already come from ICC)? Are affine approximations related to the approximations used in k-CFA? One should also mention Ghica's *geometry of synthesis* (`http://www.veritygos.org/`), which is based on the idea of extracting circuits from higher-order programs.
- A more speculative (and much more ambitious) question is whether the higher-order view of non-uniformity may shed some light on the role (or lack thereof) of uniformity in lower bound proofs. Is there a more abstract formulation of the notion of approximation (categorical, type-theoretic, or any other traditional tool of PL theory) which helps explaining why current lower bound proof methods fail at exploiting uniformity?

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. CUP, 2009.

[BC92] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[DLG11] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. *Log. Methods Comput. Sci.*, 8(4), 2011.

[GS14] Dan R. Ghica and Alex I. Smith. Bounded linear types in a resource semiring. In *Proceedings of ESOP*, pages 331–350, 2014.

[JHLH10] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. Static determination of quantitative resource usage for higher-order programs. In *Proceedings of POPL*, pages 223–236, 2010.

[Jon99] Neil D. Jones. Logspace and ptime characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, 1999.

[LM93] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1/2), 1993.

[Mai04] Harry G. Mairson. Linear lambda calculus and ptime-completeness. *J. Funct. Program.*, 14(6):623–633, 2004.

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

[Maz14] Damiano Mazza. Non-uniform polytime computation in the infinitary affine lambda-calculus. In *Proceedings of ICALP, Part II*, pages 305–317, 2014.

[MT15] Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In *Proceedings of ICALP, Part II*, pages 350–361, 2015.

[NZ15] David Nowak and Yu Zhang. Formal security proofs with minimal fuss: Implicit computational complexity at work. *Information and Computation*, 241:96–113, 2015.

[Ter04] Kazushige Terui. Proof nets and boolean circuits. In *Proceedings of LICS*, pages 182–191, 2004.