Non-uniform Polytime Computation in the Infinitary Affine Lambda-calculus

Damiano Mazza
CNRS, UMR 7030, LIPN, Université Paris 13, Sorbonne Paris Cité
Damiano.Mazza@lipn.univ-paris13.fr

Abstract

We give an implicit, functional characterization of the class of non-uniform polynomial time languages, based on an infinitary affine lambda-calculus and on previously defined bounded-complexity subsystems of linear (or affine) logic. The fact that the characterization is implicit means that the complexity is guaranteed by structural properties of programs rather than explicit resource bounds. As a corollary, we obtain a proof of the (already known) P-completeness of the normalization problem for the affine lambda-calculus which mimics in an interesting way Ladner's P-completeness proof of CIRCUIT VALUE (essentially, the argument giving the Cook-Levin theorem). This suggests that the relationship between affine and usual lambda-calculus is deeply similar to that between Boolean circuits and Turing machines, opening some interesting perspectives, which we discuss.

This work will appear in the proceedings of ICALP 2014 [Maz14]. An extended version with proofs is available on the author's web page.

Implicit computational complexity and linear logic. Loosely speaking, the aim of implicit computational complexity 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 guarantee a polynomial dependence of the runtime on the input by construction, *i.e.*, because they satisfy some syntactic condition, not because their execution is artificially stopped after a polynomial number of steps. 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].

We consider here the question of finding an implicit characterization of non-uniform polynomial time, *i.e.*, the class $\mathsf{P/poly}$. Our approach brings together two lines of work, both based on linear logic. The first is the linear-logical take at implicit computational complexity initiated by Girard [Gir98] and reformulated in the λ-calculus, for example, by Asperti and Roversi [AR02]. The second is the author's work on the infinitary affine λ-calculus [Maz12], previously considered also by Kfoury [Kfo00] and Melliès [Mel06].

For our present purposes, the essence of linear logic is in its resource awareness. Linear (or, more precisely, affine) types describe volatile data, which may be accessed only once. Accordingly, the linear (or affine) functional type $A \multimap B$ describes programs producing an output of type B by using their input of type A exactly (or at most) once. Persistent data is described by the type $!A$, which may be understood as volatile access to a bottomless pile of copies of A, thus obtaining unlimited access to A. The usual functional type $A \to B$ may then be expressed by $!A \multimap B$.

In the λ-calculus, which is the prototypical functional language, affinity takes the form of forbidding duplication, which translates into an extremely simple syntactic restriction: each variable must appear at most once in a term. Let us give a more formal presentation.

The infinitary affine λ-calculus. We start by introducing *patterns*, inductively generated by

$$\mathsf{p}, \mathsf{q} ::= a \mid x \mid \mathsf{p} \otimes \mathsf{q},$$

where a and x range over disjoint sets of *linear* and *non-linear* variables, respectively, and we stipulate that, in $\mathsf{p} \otimes \mathsf{q}$, p and q share no variables. Consider now the terms co-inductively generated by

$$t, u ::= \bot \mid a \mid x_i \mid \lambda \mathsf{p}.t \mid tu \mid t \otimes u \mid \langle u_0, u_1, u_2, \ldots \rangle,$$

where x_i is an *occurrence* of x, with $i \in \mathbb{N}$, and $\langle u_0, u_1, u_2, \ldots \rangle$ is an infinite sequence of terms, called *box*. We use $\mathbf{u}$ to succinctly denote boxes and $\mathbf{u}(i)$ to denote the i-th term. The concept of free and bound variable, along with the usual notion of α-equivalence, are defined as usual, noting that, if x is in p, then *all* occurrences of x are bound in $\lambda \mathsf{p}.t$.

We denote by $\ell\Lambda_\infty$ the set of all above-defined terms t verifying the following:

Affinity: after α-converting all bound variables so that all λs bind distinct variables, every linear variable appears at most once in t and, if x_i and x_j both appear in t, then $i \neq j$. So, for instance, $\lambda x.x_0 \otimes \lambda x.x_0 \in \ell\Lambda_\infty$, because it can be α-converted to $\lambda x.x_0 \otimes \lambda y.y_0$, whereas $\lambda x.x_0 x_0$ is excluded.

Boxes: boxes of t have no free linear variable.

Well-foundedness: the underlying tree of t is well-founded, *i.e.*, it has no infinite path. For instance, we accept $\langle x_0, x_1, x_2, \ldots \rangle$ but refuse $x_0(x_1(x_2 \ldots))$.

The first two conditions are the important ones; well-foundedness is more technical and, at the intuitive level, may be replaced with requiring the height of terms to be finite (an example of well-founded term of infinite height is $\langle \lambda x.x_0, \lambda x.x_0 x_1, \lambda x.x_0(x_1 x_2), \ldots \rangle$).

For defining reduction, the important case is

$$(\lambda x.t)\mathbf{u} \to t[\mathbf{u}(i)/x_i],$$

i.e., $\mathbf{u}(i)$ replaces the unique (if any) occurrence x_i in t. Reduction for general patterns is defined as expected, with any term matching a linear variable and $t \otimes u$ matching $\mathsf{p} \otimes \mathsf{q}$ as soon as t matches p and u matches q (so, for instance, $(\lambda a \otimes b.t)u$ is not a redex unless u is of the form $u_1 \otimes u_2$, in which case the reduct is $t[u_1/a][u_2/b]$).

We say that a box $\mathbf{u}$ is *finite* if $\mathbf{u}(i) = \bot$ for almost all $i \in \mathbb{N}$. A term of $\ell\Lambda_\infty$ is *finite* if it contains only finite boxes. We denote by $\ell\Lambda$ the set of finite terms. It is immediate to check that, whenever $t \in \ell\Lambda$ and $t \to t'$, $t' \in \ell\Lambda$ and its size (in terms of number of symbols) is strictly smaller than that of t, so the calculus is terminating. In fact, the whole infinitary calculus $\ell\Lambda_\infty$ is affine, because no term is ever duplicated by reduction, so the fact that finite terms terminate is not surprising. Of course, infinite terms may diverge, as $\Omega := \Delta\langle \Delta, \Delta, \Delta, \ldots \rangle$ shows, with $\Delta := \lambda x.x_0\langle x_1, x_2, x_3, \ldots \rangle$.

In [Maz12], we showed how $\ell\Lambda_\infty$ may be endowed with a uniform structure[1] such that:

- the space $\ell\Lambda_\infty$ is Cauchy-complete, with $\ell\Lambda$ as a dense subspace;
- reduction is continuous (as a function from $\ell\Lambda_\infty$ to itself);
- if one restricts to the subspace generated by $t ::= \bot \mid x_i \mid \lambda x.t \mid t\mathbf{u}$ and takes the quotient under a certain partial equivalence relation enforcing uniformity,[2] one obtains exactly the pure λ-calculus.

So $\ell\Lambda_\infty$ may be seen as the completion of $\ell\Lambda$, in a way which is compatible with reduction (much like the real numbers are the completion of the rational numbers, in a way which is compatible with the basic operations), and usual λ-terms may be embedded in $\ell\Lambda_\infty$. However, the completion process introduces a host of infinitary terms which do not correspond to any usual λ-term. The reason is easily explained: to act as a persistent memory cell, a datum of type $!A$ must contain infinitely many *identical* copies of a datum of type A. Technically, the embedding of a λ-calculus application MN (with N closed) is $[\![M]\!]\langle [\![N]\!], [\![N]\!], [\![N]\!], \ldots \rangle$, where $[\![M]\!], [\![N]\!]$ are the embeddings of M, N, respectively.[3] Without further constraints, the infinitary affine λ-calculus allows memory cells whose content changes arbitrarily with each access. This is the "functional gateway" to non-uniform computation.

A characterization of P/poly. Our first objective is to "tame" the non-uniformity of the unrestricted calculus $\ell\Lambda_\infty$ so as to keep it within interesting boundaries, namely those of P/poly. Let us give an

[1] A uniform space is essentially a generalization of the concept of metric space still allowing one to speak of Cauchy sequences. This use of the word "uniform", standard in topology [Bou98], is unfortunately completely unrelated to the equally standard meaning more common in computer science (and, in particular, in circuit complexity).

[2] In the usual sense of computer science.

[3] The attentive reader will at this point have noticed how the terms Δ and Ω introduced above are just the embeddings of their namesake standard λ-terms. The reader acquainted with linear logic will have gone even further and noticed that we are just applying Girard's translation of intuitionistic logic in linear logic.

informal description of what this means. Using (an adaptation of) the standard λ-calculus encodings of binary strings, we may say that a term t *decides* $L \subseteq \{0,1\}^*$ in $\ell\Lambda_\infty$ if, given $w \in \{0,1\}^*$, $t\underline{w} \to^* \underline{b}$ with $b \in \{0,1\}$ according to whether w belongs to L ($\underline{w}$ is the encoding of w). Now, t is generally infinite, but we may define a canonical sequence $\lfloor t \rfloor_n$ of approximations of t, which are finite affine terms such that $\lim \lfloor t \rfloor_n = t$. Intuitively, $\lfloor t \rfloor_n$ behaves like t in which every internal memory cell is limited to at most n accesses. We may then appeal to the continuity of reduction, by which, if we let u_n be the normal form of $\lfloor t \rfloor_n \underline{w}$, we have that $\underline{b} = \lim u_n$. But our topology is such that pieces of data like $\underline{b}$ are isolated points, so there exists $m \in \mathbb{N}$ such that $u_n = \underline{b}$ for all $n \geq m$. This means that a finite approximation of t suffices to compute $t\underline{w}$. The size of $\lfloor t \rfloor_m$ is linear in m, so the question is: How big is m? Can we relate it to $|w|$? If we can make m be polynomial in $|w|$, the language decided by t is in $\mathsf{P/poly}$: we may use the $\lfloor t \rfloor_m$ as (polynomial) advice and then normalize $\lfloor t \rfloor_m \underline{w}$, which may be done in polynomial time in $|w|$ because it is a finite affine term.

There exist several λ-calculus characterizations of P based on linear logic (most notably Girard's light linear logic [Gir98] and Lafont's soft linear logic [Laf04]) and the naive idea to polynomially bound m would be to reuse the recipes given therein. However, non-uniformity in the λ-calculus is extremely subtle and the approach "take your favorite λ-calculus characterization of P and add non-uniformity" does not necessarily yield $\mathsf{P/poly}$. The most surprising aspect is that polytime non-uniformity seems to refuse the logical principle of contraction (expressed by the formula $!A \multimap !A \otimes !A$): in its presence, m may be exponentially big and we may therefore decide any language (an intuitive explanation is given below). This rules out Girard's approach. Lafont's system does not use contraction but appears to have the opposite problem: we are currently unaware of whether the expressiveness of its non-uniform version reaches $\mathsf{P/poly}$.

The key to our solution is a new structural constraint on terms, which we call *parsimony*. In $\ell\Lambda_\infty$, contraction (corresponding to duplication) is implemented using "Hilbert's hotel": from an infinite family $(x_i)_{i\in\mathbb{N}}$ representing an argument of type $!A$, we make two infinite families, as in the term $\lambda x.\langle x_0, x_2, x_4, \ldots\rangle \otimes \langle x_1, x_3, x_5, \ldots\rangle : !A \multimap !A \otimes !A$. By discarding one of the two families, we obtain $\lambda x.\langle x_1, x_3, x_5, \ldots\rangle : !A \multimap !A$, which, iterated n times, yields a family whose first element is $x_{O(2^n)}$, causing the exponential growth rate of m mentioned above (we point out that both the latter term and its iteration would be allowed following the approach of light linear logic). A very high level description of parsimony is that boxes may not "waste" occurrences: a box contains either finitely many x_i or almost all of them (*i.e.*, a co-finite family). Parsimony therefore refuses contraction, which requires infinite co-infinite families. Instead, it allows an asymmetric form of contraction, also known as absorption, expressed by the formula $!A \multimap !A \otimes A$. Unfortunately, the exact definition of parsimony is a bit too technical for this abstract; the details may be found in [Maz14] (available on the author's web page).

Parsimony is coupled with *stratification*, which is a staple of Girard's work [Gir98]. Stratification partitions a program into rigid levels which may not interact and, very roughly speaking, forbids the self-reference that makes the λ-calculus Turing powerful. Alone, it guarantees termination (in elementary time, in the uniform case). Without it, parsimonious terms may diverge and the question of bounding m may not make sense.

By restricting to parsimonious stratified terms one obtains an infinitary calculus $\ell\Lambda^{\mathrm{ps}}_\infty$ with the following property:

Theorem 1 *The class of languages decidable by terms of $\ell\Lambda^{\mathrm{ps}}_\infty$ (w.r.t. a suitable adaptation of the usual λ-calculus representations of binary strings and Booleans) is exactly $\mathsf{P/poly}$.*

Soundness (*i.e.*, that every language decidable in $\ell\Lambda^{\mathrm{ps}}_\infty$ is in $\mathsf{P/poly}$) is obtained by polynomially bounding the parameter m, as delineated above. Completeness is shown by encoding deterministic polytime Turing machines with polynomial advice in $\ell\Lambda^{\mathrm{ps}}_\infty$. The encoding of polytime machines (without advice) is done in the uniform fragment of the calculus, which may actually be seen as the image (via Curry-Howard) of a subsystem of linear logic with a monoidal § modality (as light linear logic) and a monoidal ! modality enjoying weakening and the law $!A \multimap !A \otimes \S A$ (contrarily to the ! modality of light linear logic, which enjoys weakening and contraction but is not monoidal). Hence, we obtain as a by-product a new linear-logical characterization of P.

Perspectives. Let $L \in \mathsf{P}$. By Theorem 1, we know that L is decided by a (usual, finite but not necessarily affine) λ-term term M whose infinitary affine embedding $[\![M]\!]$ is in $\ell\Lambda^{\mathrm{ps}}_\infty$, so deciding whether

$w \in L$ amounts to normalizing $[\![M]\!]\underline{w}$. But, for this, we know that it is enough to normalize $\lfloor [\![M]\!] \rfloor_m \underline{w}$ with m polynomial in $|w|$. One may see that building $\lfloor [\![M]\!] \rfloor_m$ from M may be done in logarithmic space (in $|w|$), much like building the circuit representing the computation of a polytime Turing machine from the trace of its execution on w.

We have therefore given an alternative proof of the P-completeness of the normalization problem for the affine λ-calculus (given an affine λ-term, decide whether its normal form is the Boolean $\mathtt{tt}$). Mairson [Mai04] showed this by encoding Boolean circuits in affine λ-terms. The interest of the above proof is that it is virtually identical to the usual P-completeness proof of CIRCUIT VALUE [Lad75], which is essentially the Cook-Levin theorem and does not rest on the P-completeness of another problem. It is also noteworthy that the "locality of computation" is reflected in the continuity of normalization.

Our results seem to suggest the following "equation":

$$\frac{\text{affine } \lambda\text{-terms}}{\text{(infinitary affine) } \lambda\text{-terms}} = \frac{\text{Boolean circuits}}{\text{Turing machines (with advice)}}$$

The relationship between Boolean circuits and affine calculi was of course already known [Mai04, Ter04]. However, we are seeing a connection here which is deeper than what was shown by any previous result.

An interesting perspective given by the above "equation" is to study the notion of uniformity of families of Boolean circuits via the uniformity of the infinitary affine λ-calculus. This may be defined in a purely algebraic way: the terms which are embeddings of usual λ-terms may be characterized by means of a partial equivalence relation, as in [Maz12]. This might be turned into a notion of uniform family of Boolean circuits which is purely intrinsic, *i.e.*, it depends only on the "shape" of the circuits in the family and does not invoke external algorithms producing the circuits themselves.

Another line of research is the development of a denotational semantics for a simply-typed version of $\ell\Lambda^{\mathrm{ps}}_{\infty}$. In the usual λ-calculus, semantic arguments have been used to show the impossibility of representing certain functions (*e.g.* subtraction) with simple types [BDS13]. The interest of such arguments is that they are not natural in the sense of Razborov and Rudich: denotational semantics allows one to exploit extensional properties of functions, breaking the so-called *largeness* property. Preliminary results (via [Ter04]) show that the class of languages decidable in simply-typed $\ell\Lambda^{\mathrm{ps}}_{\infty}$, call it $\lambda\mathsf{C}^0$, is such that $\mathsf{TC}^0 \subseteq \lambda\mathsf{C}^0 \subseteq \mathsf{AC}^1$ (non-uniform). Any application of denotational semantics showing that some $L \notin \lambda\mathsf{C}^0$ would therefore give us a non-natural proof of $L \notin \mathsf{TC}^0$, an extremely interesting perspective.

References

[AR02] Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Trans. Comput. Log.*, 3(1):137–175, 2002.

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

[BDS13] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Cambridge University Press, 2013.

[Bou98] Nicolas Bourbaki. *General Topology: Chapters 1–4*. Springer, 1998.

[Gir98] Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

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

[Kfo00] A. J. Kfoury. A linearization of the lambda-calculus and consequences. *J. Log. Comput.*, 10(3):411–436, 2000.

[Lad75] Richard E. Ladner. The circuit value problem is log-space complete for P. *SIGACT News*, 6(2):18–20, 1975.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, 2004.

[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*, 2014. To appear.

[Mel06] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *Theor. Comput. Sci.*, 358(2-3):200–228, 2006.

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