The Parsimonious Lambda-calculus and Implicit Computational Complexity

Damiano Mazza

Abstract

The parsimonious lambda-calculus arose from studying the usual lambda-calculus by means of affine approximations. It has turned out to have several interesting applications to implicit computational complexity, which we discuss here.

Reduction and complexity in the λ-calculus. A reduction step in the λ-calculus is not an atomic operation, because arbitrarily large sub-terms may be duplicated. A particularly striking example is the reduction below, where $D := \lambda d.\lambda a.dd(aa)$:

$$DDA \to DD(AA) \to DD(AA(AA)) \to DD(AA(AA)(AA(AA))) \to \cdots$$

In n steps, 2^n copies of the subterm A are created. The fact that reduction (indeed, weak head-reduction) performs an exponential amount of work in a linear number of steps casts serious doubts on its relevance in terms of computational complexity. Although recent work by Accattoli and Dal Lago [ADL14] has shown that such doubts are ill-founded,[1] it is still useful to have a deeper understanding of what goes on in λ-calculus reduction.

The infinitary affine λ-calculus. A way of gathering such an understanding is looking at the embedding of the λ-calculus into the infinitary affine λ-calculus introduced in [Maz12], and in particular at the theory of continuous approximations arising from it. Affinity means that duplication is forbidden: a reduction step may "move around" or erase subterms, not make copies of them. Finite affine terms normalize in a number of steps bounded by their size, so reduction on them is trivially sound from the point of view of complexity. Infinitary terms are needed to recover the computational power of the usual λ-calculus: the potential infinity present in usual λ-terms (offered by non-affine variables) is replaced by actual infinity (every copy of a subterm which will be made during reduction is already there from the start).

The key idea now is to topologize the space of infinitary affine terms in such a way that:

1. finite affine terms form a dense subset;
2. reduction is continuous.

Point 1 may be refined by saying that each infinitary affine term t admits a sequence of finite affine terms $(\lfloor t \rfloor_n)_{n\in\mathbb{N}}$ such that $\lim \lfloor t \rfloor_n = t$ (we say that $\lfloor t \rfloor_n$ is the *n-th approximation* of t). One may then study the "modulus of continuity" of reduction: if $t \to^l u$ (t reduces in l steps to u), continuity guarantees us that for all $n \in \mathbb{N}$, there exists $m \in \mathbb{N}$ such that $\lfloor t \rfloor_m \to^l v$ and $\lfloor v \rfloor_n = \lfloor u \rfloor_n$. In other words, if we want to know a finite approximation of u, we only need a finite approximation of t. But how is m related to n and l? While $m = O(n)$, it is easy to see that m may be exponential in l, even when $n = 0$ (the head variable of u, or the absence of such, is visible in $\lfloor u \rfloor_0$, so the 0-th approximation already contains useful information). This is in particular the case when t is the infinitary affine image of a usual λ-term such as the example given above. So affinity explains the exponential amount of duplication by the need for exponentially large approximations (the size of $\lfloor t \rfloor_m$ is linear in m).

The parsimonious λ-calculus. The contribution of [Maz14] was to find a coherent (*i.e.*, stable under reduction) subset of the infinitary affine λ-calculus for which the above "modulus of continuity" is polynomial in l. The resulting calculus, which we deem *parsimonious*, may be seen as the term calculus

[1] Accattoli and Dal Lago showed that counting the number of reduction steps to normal form is a reasonable cost measure, *i.e.*, it is polynomially related to Turing machine complexity.

$$\frac{}{\Gamma;\Delta, a : A \vdash a : A}\ \mathsf{ax}$$

$$\frac{\Gamma;\Delta, a : A \vdash t : B}{\Gamma;\Delta \vdash \lambda a.t : A \multimap B}\ {\multimap}\mathsf{I} \qquad \frac{\Gamma;\Delta \vdash t : A \multimap B \quad \Gamma';\Delta' \vdash u : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash tu : B}\ {\multimap}\mathsf{E}$$

$$\frac{\Gamma;\Delta \vdash t : A \quad \Gamma';\Delta' \vdash u : B}{\Gamma,\Gamma';\Delta,\Delta' \vdash t \otimes u : A \otimes B}\ {\otimes}\mathsf{I} \qquad \frac{\Gamma';\Delta' \vdash u : A \otimes B \quad \Gamma;\Delta, a : A, b : B \vdash t : C}{\Gamma,\Gamma';\Delta,\Delta' \vdash \mathsf{let}\, a \otimes b = u \,\mathsf{in}\, t : C}\ {\otimes}\mathsf{E}$$

$$\frac{;\overline{a} : \Delta \vdash t : A}{\overline{x} : \Gamma; \vdash !t[\overline{x}_0/\overline{a}] : !A}\ \mathsf{!I} \qquad \frac{\Gamma';\Delta' \vdash u : !A \quad \Gamma, x : A;\Delta \vdash t : C}{\Gamma,\Gamma';\Delta,\Delta' \vdash \mathsf{let}\, !x = u \,\mathsf{in}\, t : C}\ \mathsf{!E}$$

$$\frac{\Gamma, x : A;\Delta, a : A \vdash t : C}{\Gamma, x : A;\Delta \vdash t^{x++}[x_0/a] : C}\ \mathsf{abs} \qquad \frac{\Gamma;\Delta \vdash t : A \quad \Gamma';\Delta' \vdash u : !A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t :: u : !A}\ \mathsf{coabs}$$

Figure 1: The simply typed parsimonious λ-calculus.

of a variant (although not a subsystem) of linear logic, based on an exponential modality verifying the isomorphism $!A \cong A \otimes !A$. Its terms are generated by

$$t, u ::= a \mid \lambda a.t \mid tu \mid \mathsf{let}\, a \otimes b = u \,\mathsf{in}\, t \mid t \otimes u \mid x_i \mid \mathsf{let}\, !x = u \,\mathsf{in}\, t \mid !t \mid t :: u,$$

where a (resp. x) ranges over *affine* (resp. *exponential*) variables, and the usual affinity constraints hold: affine variables appear at most once; every occurrence of x appears with a distinct index $i \in \mathbb{N}$; and in $!t$, t has no free affine variable. Terms of the form $!t$ are called *boxes*. Terms must additionally enjoy the following *parsimony constraint*: every exponential variable appears in at most one box and at most once therein, and the index of such an occurrence is the maximum in the term. For instance:

- $\Delta := \lambda a.\mathsf{let}\, !x = a \,\mathsf{in}\, x_0 !x_1$ is parsimonious;
- none of the following terms is parsimonious: $!x_0 \otimes !x_1$ and $!!x_0$ (x appears in two boxes); $!(x_0 \otimes x_3)$ (x appears twice in a box); $x_2 !x_1$ (the unique occurrence of x in the unique box is not the highest in the term).

If t is a parsimonious term, we denote by t^{++} the term obtained by replacing every free occurrence of exponential variable x_i with x_{i+1}. We call a term of the form $t_1 :: \cdots :: t_n :: !u$ a *stream*. We denote streams by $\mathbf{u}$ and, if $\mathbf{u}$ is the above stream, $\mathbf{u}(i)$ (with $i \in \mathbb{N}$) stands for t_{i+1} if $i < n$, or for $u^{+(i-n)}$ if $i \geq n$, where u^{+j} denotes u to which the operation $(-)^{++}$ is applied j times. In other words, we see $\mathbf{u}$ as an infinite stream $t_1 :: \cdots :: t_n :: u :: u^{++} :: u^{+2} :: u^{+3} :: \cdots$.

Reduction is defined by the following rules:

$$\begin{array}{cc}
(\lambda a.t)u \to_\beta t[u/a] & C[t]u \to_{\mathsf{c1}} C[tu] \\
\mathsf{let}\, a \otimes b = u \otimes v \,\mathsf{in}\, t \to_\otimes t[u/a, v/b] & \mathsf{let}\, \mathsf{p} = C[u] \,\mathsf{in}\, t \to_{\mathsf{c2}} C[\mathsf{let}\, \mathsf{p} = u \,\mathsf{in}\, t] \\
\mathsf{let}\, !x = \mathbf{u} \,\mathsf{in}\, t \to_! t\{\mathbf{u}/x\} &
\end{array}$$

Rules β, $\otimes$, $\mathsf{c1}$ and $\mathsf{c2}$ are standard. In the latter two, $C[\bullet]$ is a context of the form $\mathsf{let}\, \mathsf{q} = v \,\mathsf{in}\, \bullet$, and both p, q stand for either $a \otimes b$ or $!x$. In the $!$ rule, $t\{\mathbf{u}/x\}$ stands for t in which every x_i is substituted by $\mathbf{u}(i)$. For example, if Δ is the parsimonious term introduced above, we have

$$\Delta !\Delta \quad \to \quad \mathsf{let}\, !x = !\Delta \,\mathsf{in}\, x_0 !x_1 \quad \to \quad \Delta !\Delta,$$

showing that non-termination is possible.

In fact, the untyped parsimonious λ-calculus is Turing-complete. However, contrarily to the usual λ-calculus, the parsimonious λ-calculus has a much more interesting behavior in terms of (implicit) computational complexity.

Types. As mentioned above, the parsimonious λ-calculus may be seen as the Curry-Howard image of a variant of (intuitionistic) linear logic. The propositional case yields a simply-typed system, given in Fig. 1 and called **PL** (for "parsimonious logic"). In the $\mathsf{!I}$ rule, t^{x++} is defined just as t^{++} but only the free occurrences of x are re-indexed. The distinguishing property of the exponential modality $!(-)$ of parsimonious logic is the isomorphism $!A \cong A \otimes !A$, which we call *Milner's law.* It is given by the terms

$$\begin{array}{rcl} \lambda a.\mathsf{let}\,!x = a\,\mathsf{in}\,x_0 \otimes !x_1 & : & !A \multimap A \otimes !A \\ \lambda c.\mathsf{let}\,a \otimes b = c\,\mathsf{in}\,\mathsf{let}\,!x = b\,\mathsf{in}\,a :: !x_0 & : & A \otimes !A \multimap !A \end{array}$$

Milner's law does not hold in linear logic. Its name comes from the resemblance with the behavior of replication in the π-calculus (indeed, the parsimonious λ-calculus satisfies the structural equivalence generated by $!t \equiv t :: !t^{++}$). It says that $!A$ is the type of streams of elements of type A. The forward implication is "pop", the backward is "push". The fact that these are true inverses of each other is why we speak of (infinite) streams rather than (finite) stacks.

The types of Booleans, Church (unary) integers and binary strings may be defined as

$$\mathsf{Bool} := o \otimes o \multimap o \otimes o \qquad\qquad \mathsf{Str} := !(o \multimap o) \multimap !(o \multimap o) \multimap o \multimap o$$

with o a fixed ground type. Now, if we denote by PL the class of languages decidable by terms of **PL** of type $\mathsf{Str}[A/o] \multimap \mathsf{Bool}$, with A arbitrary, we have

Theorem 1 ([Maz15]) $\mathsf{PL} = \mathsf{L}$ *(deterministic logarithmic space).*

Completeness ($\mathsf{L} \subseteq \mathsf{PL}$) uses the descriptive complexity characterization of logspace (we encode first-order logic with deterministic closure), whereas soundness ($\mathsf{PL} \subseteq \mathsf{L}$) is established by executing terms on a token machine, using the geometry of interaction in the style of [DLS10].

We stress that this is arguably the simplest higher-order language capturing logarithmic space currently known. Moreover, the analogous question for the usual simply-typed λ-calculus, or for propositional linear logic, lacks to our knowledge such a straightforward, standard answer.[2] This, we believe, supports the claim that parsimony is an interesting and natural notion.

The Cook-Levin theorem and continuity. The main content of the Cook-Levin theorem is that "computation is local". The essence of the idea is present also in Ladner's proof of P-completeness of Circuit Value [Lad75]. A deterministic Turing machine whose runtime is bound by $p(n)$ induces, on inputs of size n, a square array of size $p(n)^2$ consisting of the "space-time" of the execution of the machine, the i-th line representing the contents of the tape at step i. The key observation now is that line $i+1$ is induced by line i by means of purely local rules, implementable by a constant-depth Boolean circuit (depending on the machine). Under the convention that the machine writes its acceptance/rejection bit in a fixed position of the tape, each input x of length n determines a circuit C of size $O(p(n)^2)$ with one output and whose inputs are set to match x. Therefore, computing the value of C is tantamount to deciding acceptance/rejection of x. If p is a polynomial, the size of C is also polynomial and its description may be given in logarithmic space (or even less), showing P-hardness of Circuit Value.

The corresponding problem in the λ-calculus setting is Affine NF: given a finite affine λ-term, decide whether its normal form is the Boolean "true". Indeed, Mairson showed this problem to be P-complete, by reducing Circuit Value to it [Mai04]. Nevertheless, it is interesting to see what happens if we try to prove Ladner's theorem for Affine NF, *i.e.*, ignoring the existence of other non-trivial P-complete problems which we may reduce to it. We start from a language L decidable in polynomially many reduction steps by a λ-term t. Deciding whether $x \in L$ amounts to deciding whether $t\,\underline{x}$ normalizes to "true" (where $\underline{x}$ is the Church encoding of x). We may of course embed $t\,\underline{x}$ in the infinitary affine λ-calculus and consider approximations. Since Booleans are determined by their head variable, it is enough to compute the 0-th approximation of the normal form of $t\,\underline{x}$ which, by continuity, means computing the normal form of the finite affine term $\lfloor t \rfloor_m\,\underline{x}$ for some m. Here is where the proof breaks: we saw that m may be exponentially big in spite of the reduction being polynomially long in $|x|$.

By contrast, the proof goes perfectly through in the parsimonious λ-calculus: by Turing-completeness, we may assume t to be parsimonious, so m, and hence the size of $\lfloor t \rfloor_m$, will be polynomial, just like the circuit C in Ladner's proof. We see that proving P-completeness of Affine NF is essentially the same as proving P-completeness of Circuit Value, with the "locality of computation" being replaced by the continuity of reduction. This requires, however, a shift from the usual λ-calculus to the parsimonious λ-calculus.

[2] All we know is that the analogous λ-calculus class is strictly contained in $\mathsf{LINTIME}$.

Non-uniform computation. The above suggests that a parsimonious λ-term t deciding a language in P may be seen as a family of finite affine terms $\lfloor t \rfloor_n$ of polynomial size. What if we take arbitrary polynomial-size families, not coming from λ-terms? The answer was given, in a slightly different but equivalent form, by Terui [Ter04]. We call APN the class of languages decidable by polynomial-size families of simply-typed finite affine terms, and by APN^0 the subclass in which we further restrict the height of the types to be constant. Then:

Theorem 2 ([Ter04]) $\mathsf{APN} = \mathsf{P/poly}$ *and* $\mathsf{APN}^0 = \mathsf{AC}^0[\textsc{ustconn}_2]$, *where the latter denotes the class of languages decidable by polynomial-size and bounded-depth Boolean circuits with gates deciding the reachability problem for degree-2 undirected graphs.*

This above approach to non-uniformity mimics the use of circuits. We may call it the "family approach". The infinitary roots of the parsimonious λ-calculus allow for a different approach to non-uniformity, which we may call the "individual approach". This is exemplified by the definition of non-uniform classes in terms of machines with advice. In one view, we have an infinite family of finite programs; in the other, one infinite program. This latter approach is closer in spirit to ICC and it is where the parsimonious λ-calculus brought its contribution.

In fact, non-uniformity may be injected into the system **PL** by considering boxes of the form

$$!_f(u_1, \ldots, u_k),$$

where f is an arbitrary surjective function from $\mathbb{N}$ to $\{1, \ldots, k\}$. Such a box must be seen as a stream

$$u_{f(0)} :: u^{++}_{f(1)} :: u^{+2}_{f(2)} :: u^{+2}_{f(3)} :: \cdots ,$$

i.e., as a sort of "infinite word" on the alphabet $\{u_1, \ldots, u_k\}$, the "letters" being given by the function f. The fact that f is arbitrary (even uncomputable) is what makes the boxes non-uniform. If we call nuPL the class of languages decidable by simply-typed non-uniform parsimonious terms, we have

Theorem 3 ([MT15]) $\mathsf{nuPL} = \mathsf{APN}^0 = \mathsf{L/poly}$.

This result has several highlights: it gives the first ICC characterization of the non-uniform class $\mathsf{L/poly}$ (to our knowledge, the only other ICC work considering non-uniformity is [Maz14], Terui's work [Ter04] not being truly classifiable as ICC); it uses continuous approximations in the λ-calculus to reconcile the family/individual approaches to non-uniformity; and it gives a more straightforward characterization of Terui's class APN^0.[3]

References

[ABC⁺09] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory Comput. Syst.*, 45(4):675–723, 2009.

[ADL14] Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In *Proceedings of CSL-LICS*, pages 8:1–8:10, 2014.

[DLS10] Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Proceedings of ESOP*, pages 205–225, 2010.

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

[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.

[Maz15] Damiano Mazza. Simple parsimonious types and logarithmic space, 2015. Available on the author's web page.

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

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

[3]The fact that $\textsc{ustconn}_2$ is L-complete is a consequence of [ABC⁺09] but was unknown at the time of [Ter04]. The proof of Theorem 3 does not use this fact anyway.