From normal functors to logarithmic space queries

NGUYỄN Lê Thành Dũng
LIPN, CNRS & Université Paris 13, France
nltd@nguyentito.eu

Pierre PRADIC
LIP, ENS de Lyon, France
MIMUW, University of Warsaw, Poland
pierre.pradic@ens-lyon.fr

We introduce a new approach to implicit complexity in linear logic, inspired by functional database query languages and using recent developments in effective denotational semantics of polymorphism. We give the first sub-polynomial upper bound in a type system with impredicative polymorphism; adding restrictions on quantifiers yields a characterization of logarithmic space, for which extensional completeness is established via descriptive complexity.

Long version: `https://hal.archives-ouvertes.fr/hal-02024152`**, cited as [25].**

Machine-free complexity If implicit computational complexity (ICC) can be thought of as studying complexity via functional programming, then *descriptive complexity* is its declarative programming counterpart: it consists in characterizing complexity classes as sets of *queries* – predicates over finite first-order relational structures – written in some logic. (Such structures often go by the name of *finite models*.) The field was launched by Fagin's characterization of NP by existential second-order logic [9].

This idea of representing inputs as finite first-order structures also appeared in the early history of ICC: at the same FOCS'83 meeting, Gurevich [14] showed that in this setting, a form of primitive recursion captures L, and Leivant [21] deduced from this a characterization of L in a predicative lambda-calculus. But unlike in descriptive complexity, Gurevich considers endofunctions instead of relations and queries.

In this work, we pursue an approach to ICC advocated in the above paper by Leivant, using type systems for functional languages and the proofs-as-programs correspondence.

Queries in the λ-calculus Hillebrand's PhD thesis [15] is a junction point between implicit and descriptive complexity. The idea was to represent finite models inside the simply typed λ-calculus (STλ), using them to represent the inputs to programs. By doing so, Hillebrand et al. managed to characterize P [16], PSPACE [1] and k-EXPTIME/k-EXPSPACE[1] [17] – the extensional completeness for the first two being established through descriptive complexity.

Keeping in mind the connections between finite model theory and relational databases, this can also be seen as using STλ as a functional language for database queries, expressive enough to admit translations from other languages such as Datalog, as is done in [18].

The work we present here could then be motivated as looking for a *sub-polynomial*[2] functional query language, filling a gap in the aforementioned work.

[1] k-EXPTIME(resp. k-EXPSPACE) is the class of functions which can be computed in time (resp. space) $2\uparrow^k (p(n))$, where p is a polynomial and n is the size of the input. (We use Knuth's up-arrow notation [19] for iterated exponentials: $2\uparrow^{k+1}(n) = 2^{2\uparrow^k(n)}$, and $2\uparrow^0(n) = n$.)

[2] That is, capturing a complexity class below P. To be fair, Hillebrand's thesis does define a characterization of the sub-polynomial class of *first-order queries* (FO) in STλ, but this class has very little expressivity, and our work captures a class still well above FO.

Submitted to:
DICE-FOPARA 2019

Linear logic for ICC For such a task, it is natural to turn to *linear logic*, a constructive logic born from the proofs-as-programs correspondence, in which several characterizations of sub-polynomial complexity classes have already been devised [31, 29, 6, 23, 24]. From its inception, linear logic has indeed had the ambition to "help us improve the efficiency of programs" [11, p. 3], and a landmark result in that direction was characterizing P through Light Linear Logic [13].

Here, we use Elementary Linear Logic (ELL) [13, 7], which was originally introduced to capture the class ELEMENTARY[3]. A recent line of work by Baillot et al. [2, 3, 4] shows that one can define, inside variants of ELL, types of programs which compute smaller complexity classes, such as P. We follow this approach, by introducing a type `Inp` which is essentially an abstract data type[4] for finite models. Our main result is (writing $\texttt{Bool} = 1 \oplus 1$):

Theorem 0.1. *The class of queries computed by the proofs of* $\texttt{Inp} \multimap !!\texttt{Bool}$ *in second-order Elementary Linear Logic* (ELL_2) *is* between L and NL. *Furthermore, a suitable restriction on the existential witnesses in the proof gives an* exact characterization of L.

Here NL stands for *non-deterministic logarithmic space*. Actually, we obtain a better upper bound than NL in the unrestricted case, namely the class L^{UL}, i.e. L with an UL oracle where UL stands for *unambiguous*[5] logarithmic space [28, 27] ($\mathsf{L} \subseteq \mathsf{UL} \subseteq \mathsf{L}^{\mathsf{UL}} \subseteq \mathsf{NL}$). But we believe that this is still not optimal:

Conjecture 0.2. *Even without the restriction, the class of queries obtained is* exactly L.

Our characterization has a few distinctive features with respect to the previous variants of linear logic capturing logarithmic space [29, 6, 23]: it takes place in a simple pre-existing logical system, which contains only usual logical connectives, and no primitive datatypes[6]; at the price of a more involved encoding of inputs, the `Inp` type. But the main novelty, in our opinion, is the unrestricted case: to our knowledge, it is the first[7] sub-polynomial bound in a type system with *impredicative polymorphism*.

This forces our approach to be significantly different to these previous works: they all exploit some form of the Geometry of Interaction (GoI) [12, 8] as a space-efficient evaluator, whereas in our case this does not work[8] because of impredicative quantication. (In the restricted case, which is predicative, there is still an obstruction to the GoI: the *additive* connectives of linear logic.) Instead, our tool of choice is *denotational semantics*.

Semantic evaluation and polymorphism This is indeed the sequel to a previous paper [26] which studied the semantics of second-order Multiplicative-Additive Linear Logic (MALL_2) with applications in mind; in particular it proved that Girard's model of MALL_2 in *coherence spaces* [10, 11] is finite and effective. In order to establish our upper bound on complexity, we compute the denotation of a program applied to its input the coherence space model.

This *semantic evaluation* technique has been very successful before for establishing complexity bounds in $\mathrm{ST}\lambda$: it is how soundness is established in the aforementioned works of Hillebrand et al., and

[3] This is the class of elementary recursive functions, i.e. the union over $k \in \mathbf{N}$ of the classes k-EXPTIME.

[4] This term is the programming language counterpart of existential formulas in logic, cf. infra.

[5] A non-deterministic Turing machine is unambiguous iff its accepting runs are unique.

[6] Given the special status granted to unary Church integers by the "skewed iteration" rule in Schöpp's SBAL [29], it is fair to consider them to be primitive datatypes.

[7] Excluding the characterization of regular languages in the prequel paper [26], which anticipates the techniques used here, but regular languages do not form a well-behaved complexity class (for instance they are not closed under uniform AC^0 reductions).

[8] We will not enter into details here, but essentially, the GoI works by "following paths" inside a proof, and in our case, the length of these paths would be super-polynomial.

also underlies Terui's more recent result on the complexity of β-reduction in STλ at fixed order [32]. Beyond STλ, it has been applied to System T and PCF, see the survey [20]. However, these applications have been confined to monomorphic type systems for now[9].

To adapt this to polymorphic languages, one needs an effective model of polymorphism, and such models are not easy to build. First, one must first restrict to a purely linear language[10] such as MALL_2 or the semantics will not be finite. Even then, obstacles remain: for instance, the prequel [26] proved that no degenerate model of MALL_2 (in which $\otimes$ and $⅋$ are identified) can satisfy a desirable "constancy property", so this excludes the Scott model of linear logic used by [32]. Girard managed to build a semantics for System F [10][11] which later turned out to be finite and effective for MALL_2 by representing types depending on type parameters as *normal functors*.

The unambiguity of the UL appearing in our upper bound is related to the *stability* of linear maps in coherence spaces; stable maps are the "lower-dimensional analogue" of normal functors, and interestingly it seems that stability is required for the construction of models of polymorphism based on normal functors.

New complexity phenomena in MALL The bottleneck for this L^{UL} bound is the complexity of an iterated composition problem: given a MALL_2 type A and k proofs $f_1, \ldots, f_k$ of $A \vdash A$, compute their composition $f_1 \circ \ldots \circ f_k$. To illustrate the kind of complexity constraint induced by the linearity of the f_i, consider the types $\texttt{Bool} \otimes \ldots \otimes \texttt{Bool}$ (n times) and $\texttt{Bool} \,\&\, \ldots \,\&\, \texttt{Bool}$ (n times). A non-linear function does not distinguish them, whereas for linear functions:

- an iteration over $\texttt{Bool} \otimes \ldots \otimes \texttt{Bool}$ can simulate a Turing machine running in space n (minus $O(1)$ bits for the control state);
- an iteration over $\texttt{Bool} \,\&\, \ldots \,\&\, \texttt{Bool}$ can be computed in space $O(\log(nk))$.

An ad-hoc explanation: any bit of the final "& bit vector" depends on a single bit of the penultimate vector, and so on; the computation reduces to a backwards propagation in L.

This kind of phenomenon surfaced when we tried to obtain bounds on our ELL_2 queries; we are not aware of a previous mention in the literature. Coherence spaces are sensitive to this (e.g. the interpretation of $\otimes$ and & bit vectors have respective sizes 2^n and $2n$) and thus manage to give a systematic sub-polynomial (but not L) bound on iterations.

For now, we have only managed to find a logarithmic space algorithm for those iterations in very specific cases of A, subsuming the above example. These cases still leave enough room for an extensional completeness result, leading to our exact characterization of L. But even in propositional MALL, the complexity of iterations remains mysterious.

How this result came to be To wrap up, we now explain how have been led accidentally to the work presented here.

Coming back to the work of Hillebrand et al., the motivation was not only database theory: they also wanted to overcome expressivity limits in STλ, such as Statman's classical result that equality cannot be defined on STλ Church integers (see the introduction to [18]). In fact Hillebrand and Kanellakis [17]

[9]That said, there have been some uses of rather different semantic techniques for implicit complexity in presence of polymorphism, e.g. realizability [5].

[10]The type $\forall X.\, X \to (X \to X) \to X$ of polymorphic Church integers – more generally, any infinite data type whose destructors are definable – has an inifinite denotation in any semantics of System F.

[11]In fact, coherence spaces gave birth to linear logic, since the latter was discovered by studying the connectives supported by the former.

later proved another limit of this kind: by using Church encodings for the input and output types, one can only decide *regular languages* in STλ. Such restrictions seem drastic since the β-equivalence problem for STλ is not in ELEMENTARY [30, 22], hinting that its computational power should be much greater. The encoding of finite models in STλ by Hillebrand, Kanellakis and Mairson [18] shows a way out: by changing the input representation, they manage to express all ELEMENTARY queries.

In the prequel [26], it was shown that at "fixed depth"[12], ELL_2 suffers from similar limitations: the functions from Church binary strings to $!!(1 \oplus 1)$ decide exactly regular languages. (Both this and the result by Hillebrand and Kanellakis are proved by semantic evaluation.) Transposing a successful idea for STλ in the setting of ELL_2, we replaced these Church encodings by finite models. From this followed the developments exposed above.

References

[1] Serge Abiteboul & Gerd Hillebrand (1995): *Space usage in functional query languages*. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Georg Gottlob & Moshe Y. Vardi, editors: *Database Theory — ICDT '95*, 893, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 439–454, doi:10.1007/3-540-58907-4_33.

[2] Patrick Baillot (2015): *On the expressivity of elementary linear logic: Characterizing Ptime and an exponential time hierarchy*. *Information and Computation* 241, pp. 3–31, doi:10.1016/j.ic.2014.10.005.

[3] Patrick Baillot, Erika De Benedetti & Simona Ronchi Della Rocca (2018): *Characterizing polynomial and exponential complexity classes in elementary lambda-calculus*. *Information and Computation* 261, pp. 55–77, doi:10.1016/j.ic.2018.05.005.

[4] Patrick Baillot & Alexis Ghyselen (2018): *Combining Linear Logic and Size Types for Implicit Complexity*. In: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, pp. 9:1–9:21, doi:10.4230/LIPIcs.CSL.2018.9.

[5] Ugo Dal Lago & Martin Hofmann (2011): *Realizability models and implicit complexity*. *Theoretical Computer Science* 412(20), pp. 2029–2047, doi:10.1016/j.tcs.2010.12.025.

[6] Ugo Dal Lago & Ulrich Schöpp (2016): *Computation by interaction for space-bounded functional programming*. *Information and Computation* 248, pp. 150–194, doi:10.1016/j.ic.2015.04.006.

[7] Vincent Danos & Jean-Baptiste Joinet (2003): *Linear logic and elementary time*. *Information and Computation* 183(1), pp. 123–137, doi:10.1016/S0890-5401(03)00010-5.

[8] Vincent Danos & Laurent Regnier (1999): *Reversible, irreversible and optimal λ-machines*. *Theoretical Computer Science* 227(1), pp. 79–97, doi:10.1016/S0304-3975(99)00049-3.

[9] Ronald Fagin (1973): *Contributions to the model theory of finite structures*. Ph.D. thesis, University of California, Berkeley.

[10] Jean-Yves Girard (1986): *The system F of variable types, fifteen years later*. *Theoretical Computer Science* 45, pp. 159–192, doi:10.1016/0304-3975(86)90044-7.

[11] Jean-Yves Girard (1987): *Linear logic*. *Theoretical Computer Science* 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[12] Jean-Yves Girard (1989): *Geometry of Interaction 1: Interpretation of System F*. In R. Ferro, C. Bonotto, S. Valentini & A. Zanardo, editors: *Studies in Logic and the Foundations of Mathematics*, *Logic Colloquium '88* 127, Elsevier, pp. 221–260.

[13] Jean-Yves Girard (1998): *Light Linear Logic*. *Information and Computation* 143(2), pp. 175–204, doi:10.1006/inco.1998.2700.

[14] Yuri Gurevich (1983): *Algebras of feasible functions*. In: *24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, Tucson, AZ, USA, pp. 210–214, doi:10.1109/SFCS.1983.5.

[12] When the output type is $!^k(1 \oplus 1)$ for a fixed k.

[15] Gerd G. Hillebrand (1994): *Finite Model Theory in the Simply Typed Lambda Calculus*. Ph.D. thesis, Brown University, Providence, RI, USA.

[16] Gerd G. Hillebrand & Paris C. Kanellakis (1994): *Functional Database Query Languages As Typed Lambda Calculi of Fixed Order (Extended Abstract)*. In: *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '94, ACM, New York, NY, USA, pp. 222–231, doi:10.1145/182591.182615.

[17] Gerd G. Hillebrand & Paris C. Kanellakis (1996): *On the Expressive Power of Simply Typed and Let-Polymorphic Lambda Calculi*. In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, IEEE Computer Society, pp. 253–263, doi:10.1109/LICS.1996.561337.

[18] Gerd G. Hillebrand, Paris C. Kanellakis & Harry G. Mairson (1996): *Database Query Languages Embedded in the Typed Lambda Calculus*. *Information and Computation* 127(2), pp. 117–144, doi:10.1006/inco.1996.0055.

[19] Donald E. Knuth (1976): *Mathematics and Computer Science: Coping with Finiteness*. *Science* 194(4271), pp. 1235–1242, doi:10.1126/science.194.4271.1235.

[20] Lars Kristiansen (2012): *Higher Types, Finite Domains and Resource-bounded Turing Machines*. *Journal of Logic and Computation* 22(2), pp. 281–304, doi:10.1093/logcom/exq009.

[21] Daniel Leivant (1983): *Reasoning about functional programs and complexity classes associated with type disciplines*. In: *24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, Tucson, AZ, USA, pp. 460–469, doi:10.1109/SFCS.1983.50.

[22] Harry G. Mairson (1992): *A simple proof of a theorem of Statman*. *Theoretical Computer Science* 103(2), pp. 387–394, doi:10.1016/0304-3975(92)90020-G.

[23] Damiano Mazza (2015): *Simple Parsimonious Types and Logarithmic Space*. In: *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, pp. 24–40, doi:10.4230/LIPIcs.CSL.2015.24.

[24] Damiano Mazza & Kazushige Terui (2015): *Parsimonious Types and Non-uniform Computation*. In: *Automata, Languages, and Programming*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 350–361, doi:10.1007/978-3-662-47666-6_28.

[25] Lê Thành Dũng Nguyễn & Pierre Pradic (2019): *From normal functors to logarithmic space queries*. Available at `https://hal.archives-ouvertes.fr/hal-02024152`. Submitted.

[26] Lê Thành Dũng Nguyễn, Thomas Seiller, Paolo Pistone & Lorenzo Tortora De Falco (2019): *Finite semantics of polymorphism, complexity and the power of type fixpoints*. Available at `https://hal.archives-ouvertes.fr/hal-01979009`. Submitted.

[27] A. Pavan, Raghunath Tewari & N. V. Vinodchandran (2012): *On the power of unambiguity in log-space*. *computational complexity* 21(4), pp. 643–670, doi:10.1007/s00037-012-0047-3.

[28] K. Reinhardt & E. Allender (2000): *Making Nondeterminism Unambiguous*. *SIAM Journal on Computing* 29(4), pp. 1118–1131, doi:10.1137/S0097539798339041.

[29] Ulrich Schöpp (2007): *Stratified Bounded Affine Logic for Logarithmic Space*. In: *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pp. 411–420, doi:10.1109/LICS.2007.45.

[30] Richard Statman (1979): *The typed λ-calculus is not elementary recursive*. *Theoretical Computer Science* 9(1), pp. 73–81, doi:10.1016/0304-3975(79)90007-0.

[31] Kazushige Terui (2004): *Proof Nets and Boolean Circuits*. In: *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pp. 182–191, doi:10.1109/LICS.2004.1319612.

[32] Kazushige Terui (2012): *Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus*. In: *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, pp. 323–338, doi:10.4230/LIPIcs.RTA.2012.323.