Call-by-value, Elementary Time and Intersection Types

Erika De Benedetti[1]

Dipartimento di Informatica
Università degli Studi di Torino
Torino, Italy

Simona Ronchi Della Rocca[2]

Dipartimento di Informatica
Università degli Studi di Torino
Torino, Italy

Abstract

We present a type assignment system for the call-by-value lambda-calculus, such that typable terms reduce to normal form in a number of steps which is elementary in the size of the term itself and in the rank of the type derivation. Types are built through non-idempotent and non-associative intersection, and the system is loosely inspired by elementary affine logic. The result is due to the fact that the lack of associativity in intersection supplies a notion of stratification of the reduction, which recalls the similar notion of stratification in light logics.

Keywords: Implicit computational complexity, type assignment systems, intersection types.

1 Introduction

The interest in a careful control of resource usage has been growing in the last few decades, due to the finite amount of *time* and *space* available for computing. Since classical complexity theory is based on a particular computational model, the complexity measure is in a sense "given" from outside and cannot be formalised easily; a way to overcome these problems is to employ formal methods, in order to give a proper account of complexity classes and their properties, in a way which is independent of any machine model: this allows to give an *implicit characterisation* of complexity classes, through the design of systems where the complexity of programs is statically verified, beside their usual correctness criterions, and the impact of various programming features (such as higher-order types, recursive definitions

[1] Email: debenede@di.unito.it
[2] Email: ronchi@di.unito.it

$$\sigma \wedge \sigma = \sigma \ \textbf{(I)} \qquad \sigma \wedge \tau = \tau \wedge \sigma \ \textbf{(C)} \qquad (\sigma \wedge \tau) \wedge \rho = \sigma \wedge (\tau \wedge \rho) \ \textbf{(A)}$$

Fig. 1. Properties of intersection types.

or I/O operations) on complexity can be analysed.
The idea of using a (functional) programming approach to complexity theory was first explored in [14] [17] [4], introducing the concept of *stratification*, i.e. the distribution of the computation over different strata. Later, a very interesting and fruitful tool was found in various restrictions of linear logic [13] via the proofs-as-programs correspondence induced by the Curry-Howard isomorphism; here programs are divided into strata by the modality !, and the complexity of the normalisation procedure depends on the depth of such strata. Moreover, by turning the logical system into a type assignment system for λ-calculus, it is possible to analyse the duplication and sharing of arguments in typed terms. Such direction is explored e.g. in `DLAL` [3], whose types are a proper subset of formulae of `LAL` [1], and the system `STA` [12], whose types are a proper subset of `SLL` [16] formulae.
A crucial point is the extensional completeness of such languages, i.e. the actual set of programs that we are able to type. The languages described above suffer from a common lack of expressivity, due to the constraints imposed by the underlying logical system: indeed they are able to capture only a subset of all the algorithms representing a function in the desired complexity class. Ideally, we would like to type as many "allowed" programs as possible, so a possible approach would be to enrich the features offered by the programming language (e.g. higher-order types, polymorphism and the handling of exceptions). However, if one wants to maintain pure λ-calculus as programming language, one possible approach is to enrich the typing system in order to expand the class of typable terms: then a natural tool can be found in *intersection types* [6].
The idea of intersection types stems from the observation that the conjunction can be interpreted in a way unrelated to the Curry-Howard correspondence, so that a term might be assigned many types at the same time. Historically, intersection types have been considered modulo the three equations given in Figure 1. Indeed, intersection types were born with the aim of studying qualitative properties of λ-terms, such as solvability, normalisation and strong normalisation. Lately, however, the interest has shifted towards less standard formulations of intersection types enjoying various combinations of the three properties shown in Figure 1. In fact, by removing some properties it is possible to reason also about *quantitative properties* of terms, such as bounds on the number of normalisation steps or on the size of the normal form with respect to the initial term or derivation. In particular, non idempotent intersection types (see [7]) seem to be suited to this purpose, since they retain a precise information about the number of copies of a term which will be needed during reduction; see e.g.[15] [10] [5].
In this short paper we further this idea, by proposing a type assignment system based on both linear logic (in theory) and intersection types (in practice), where the quantitative properties of typed terms can be computed statically and, when a suitable arithmetic is considered, allow the characterization of some interesting complexity classes.

2 From logic to intersection based systems

Our starting points are SLL [16] and ELL [13], characterising respectively polynomial and elementary time computation, where the number of steps of the normalisation procedure is intrinsically bound. Such bound is based on the fact that proofs are naturally *stratified* and the number of such strata (i.e. the depth) is invariant under reduction; indeed, during the normalisation procedure through cut-elimination, some subproofs identified by the ! modality can be duplicated. Then a bound on the size of the normalised proof can be statically computed from the initial proof, and a bound on the time complexity of the normalisation procedure is computed by examining the size of the proof after each cut-elimination step.
In particular, by considering a suitable notion of data having fixed depth, a characterisation of polynomial and elementary time has been obtained for SLL and ELL respectively. Moreover, by decorating the previous logics through λ-calculus, some interesting type assignment systems were obtained, for which the complexity properties of the logics are transferred to the typed terms: examples of such type systems are STA [12] and ETAS [8], where the set of typable terms characterise polynomial and elementary time, respectively.
Starting from STA, in [9] we used idempotent and non-associative intersection in order to design a typing systems for pure λ-calculus characterising FPTIME, but with more expressive power that STA. Now our aim is to fix ETAS as a starting point and try to apply the same reasoning, thus extending the typing system through intersection in order to obtain a larger class of typable terms.

3 A characterisation of elementary time

Let us consider call-by-value λ-calculus, where the set Λ of terms coincide with that of λ-calculus and values belong to the set $\Upsilon = \mathtt{Var} \cup \{\lambda \mathtt{x}.\mathtt{M} \mid \mathtt{M} \in \Lambda\}$. The call-by-value reduction $\rightarrow$ is the contextual closure of the rule $(\lambda \mathtt{x}.\mathtt{M})\mathtt{N} \rightarrow_v \mathtt{M}[\mathtt{N}/\mathtt{x}]$, where $\mathtt{N} \in \Upsilon$. The intuition behind this choice is the fact that ELL cannot be decorated in a straightforward way with λ-calculus without losing the subject reduction property; nonetheless, as in ETAS, restricting to call-by-value is enough to ensure that such crucial property is satisfied.
Our aim is to extend ETAS through intersection, which is considered without idempotence and associativity, but modulo an equivalence relation preserving the *stratification* of types; in particular, non-idempotence supplies a bound on the size of type derivations for values, while stratification corresponds to the depth of the derivation. The grammar of types is the following:

$$\begin{array}{lll} \mathtt{A},\mathtt{B},\mathtt{C} & ::= \mathtt{a} \mid \sigma \rightarrow \sigma & \text{(linear types)} \\ \sigma,\rho & ::= \mathtt{A} \mid [\sigma_1,...,\sigma_n] & \text{(multiset types)} \end{array}$$

where, informally, two types are equivalent if their linear components occur in them with the same depth, i.e., $[[\mathtt{A}],[\mathtt{B}],\mathtt{C}] \cong [[\mathtt{A},\mathtt{B}],\mathtt{C}]$, since $\mathtt{A},\mathtt{B}$ are at depth 2, while $\mathtt{C}$ is at depth 1.
The rules of system $\mathcal{M}$ are given in Figure 2. Note that we consider two different contexts $\Gamma \mid \Delta$, the first hosting linear variables (occurring only once) and variables

$$\frac{}{\mathtt{x} : [\mathtt{A}] \mid \emptyset \vdash \mathtt{x} : \mathtt{A}}\ (Ax) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \sigma \quad (\star)}{\Gamma \sqcup \Gamma' \mid \Delta \sqcup \Delta' \vdash \mathtt{M} : \sigma}\ (w) \qquad \frac{\Gamma, \mathtt{x} : [\mathtt{A}] \mid \Delta \vdash \mathtt{M} : \sigma}{\Gamma \mid \Delta \vdash \lambda\mathtt{x}.\mathtt{M} : \mathtt{A} \to \sigma}\ (\to I_L)$$

$$\frac{\Gamma \mid \Delta, \mathtt{x} : \tau \vdash \mathtt{M} : \sigma}{\Gamma \mid \Delta \vdash \lambda\mathtt{x}.\mathtt{M} : \tau \to \sigma}\ (\to I_M) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \tau \to \sigma \quad \Gamma' \mid \Delta' \vdash \mathtt{N} : \tau \quad (\star)}{\Gamma \sqcup \Gamma' \mid \Delta \sqcup \Delta' \vdash \mathtt{MN} : \sigma}\ (\to E)$$

$$\frac{\Gamma_i \mid \Delta_i \vdash \mathtt{M} : \sigma_i \quad (1 \leq i \leq n) \quad (\bullet)}{\emptyset \mid \sqcup_{i=1}^{n} \Gamma_i \sqcup_{i=1}^{n} [\Delta_i] \vdash \mathtt{M} : [\sigma_1, ..., \sigma_n]}\ (m)$$

$$(\star)\ (\Gamma \sqcup \Gamma')\#(\Delta \sqcup \Delta') \qquad (\bullet)\ \Gamma_i\#\Delta_j \text{ for every } i, j$$

Fig. 2. The system $\mathcal{M}$.

in an intermediate state, while the second context contains variables occurring possibly more than once. $\Gamma\#\Delta$ denotes that the sets of variables in Γ and Δ are disjoint. We denote by $\Pi \triangleright \Gamma \mid \Delta \vdash \mathtt{M} : \sigma$ a derivation Π proving the statement $\Gamma \mid \Delta \vdash \mathtt{M} : \sigma$, where $\mathtt{M}$ is the subject of the derivation and σ is its type.

The set of typable terms is a proper subset of the strongly normalising terms, and moreover $\mathcal{M}$ has stronger typability power than ETAS, the latter being directly derived from ELL; for example, the term $(\lambda\mathtt{x}.\mathtt{xx})(\lambda\mathtt{y}.\mathtt{y})$ is typable in $\mathcal{M}$ but not in ETAS, while $(\lambda\mathtt{x}.\mathtt{xx})(\lambda\mathtt{y}.\mathtt{y})(\lambda\mathtt{y}.\mathtt{y})$ cannot be typed despite being a strongly normalising term.

It is possible to define some measures over derivations and their subject; the *depth* and the *size* of a derivation are respectively the maximum nesting of applications of rule (m) and the number of applications of rule (Ax) in it, while the size of the subject is the sum of its sizes at every depth of the derivation. The property we enforce is that, whenever a derivation types a value with a multiset type, the number of its linear components is bounded by the size of the derivation.
Observe that the shape of rule (m) is such that there might be many "virtual" copies of the same redex, i.e. many premises of rule (m) whose subject is the considered redex; we thus identify the depth of a redex as the minimum depth of all virtual copies of that redex. It is easy to observe the following behaviour of system $\mathcal{M}$, mimicking ETAS:

Lemma 3.1 *By performing a reduction at depth i, the size of the term*

- *is unchanged at depths less than i;*
- *decreases at depth i;*
- *increases at depth greater than i, as a function of the size of the derivation at all depths greater than i.*

Let $|\Pi|$ and $|\mathtt{M}|$ be the size of Π and $\mathtt{M}$ respectively; the main result is the following:

Theorem 3.2 (Soundness) *Let $\Pi \triangleright \Gamma \mid \Delta \vdash \mathtt{M} : \sigma$; then there are elementary functions $f, g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that the length of reduction sequences starting from $\mathtt{M}$ is at most $f(|\mathtt{M}|, |\Pi|)$ and the length of every reduct is at most $g(|\mathtt{M}|, |\Pi|)$.*

The fundamental ingredients of the proof are Lemma 3.1 and the choice of a depth-by-depth reduction strategy, as in [2].
A completeness result could also be achieved, by taking into consideration that non-idempotent intersection causes data type to be typable only in a non-uniform

way; for example, the Church numeral $\underline{n} = \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}^n\mathtt{z}$ can be assigned the type $\mathrm{N}_n = [\underbrace{\mathtt{a} \to \mathtt{a}, ..., \mathtt{a} \to \mathtt{a}}_{n}] \to [\mathtt{a} \to \mathtt{a}]$, so that the size of the type depends on the size of the numeral itself. This problem could be tackled by using a non-uniform arithmetic, possibly adjoining an explicit iterator, or by enriching the types by a suitable form of universal quantification; for example, in the former case we would have a successor term $\lambda \mathtt{x}.\lambda \mathtt{s}.(\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{y}(\mathtt{sz}))(\mathtt{xs})$ of type $\mathrm{N}_n \multimap \mathrm{N}_{n+1}$, for every $n \geq 0$.

References

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

[2] Patrick Baillot, Erika De Benedetti, and Simona Ronchi Della Rocca. Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. In Diaz et al. [11], pages 151–163.

[3] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda-calculus. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 266–275. IEEE Computer Society, 2004.

[4] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 283–293. ACM, 1992.

[5] Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2011.

[6] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the lambda-calculus. *Notre-Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[7] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and lambda-calculus semantics. In *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, pages 535–560. Academic Press, London, 1980.

[8] Paolo Coppola, Ugo Dal Lago, and Simona Ronchi Della Rocca. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science*, 4(4), 2008.

[9] Erika De Benedetti and Simona Ronchi Della Rocca. A type assignment for lambda-calculus complete both for fptime and strong normalization. *Information & Computation*, to appear, 2014.

[10] Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *MSCS*, to appear, 2009.

[11] Josep Diaz, Ivan Lanese, and Davide Sangiorgi, editors. *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*. Springer, 2014.

[12] Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for *lambda*-calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.

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

[14] Neil D. Jones. *Computability and complexity - from a programming perspective*. Foundations of computing series. MIT Press, 1997.

[15] Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In Diaz et al. [11], pages 296–310.

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

[17] D. Leivant. Predicative recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.