A complete polynomial λ-calculus

Erika De Benedetti Simona Ronchi Della Rocca

Università degli Studi di Torino
Dipartimento di Informatica
Corso Svizzera 185, 10149 Torino
{debenede,ronchi}@di.unito.it

Abstract. We propose a system of stratified types, inspired by intersection types but without associativity, which is correct and complete for polynomial time computations, while typing all the strongly normalizing terms, so increasing the expressivity w.r.t. the previous proposals.

1 Introduction

This work is in the field of Implicit Computational Complexity. One of the aims of Implicit Computational Complexity is the design of programming languages with bounded computational complexity. In fact, guaranteeing and certifying a limited resources usage is of central importance for various aspects of computer science. One of the more promising approaches to this aim is based on the use of lambda-calculus as paradigmatic programming language, and on the design of type assignment systems for lambda-terms, where types guarantee, besides the functional correctness, also the desired complexity bound. In this spirit, some systems characterizing polynomial time complexity have been designed, inspired by the Light Logics. The problem of these characterizations is that, while the systems are functionally complete, their expressivity is very small, in the sense that few algorithms can be coded. In particular, a proper subset of strongly normalizing terms can be typed.

There are in the literature two characterizations of PTIME through type assignments for λ-calculus, DLAL ([2], [3]) based on Light Affine Logic [1], an affine version of Light Linear Logic [7], and STA [5, 6], based on the Soft Linear Logic of Lafont [8]. Both these characterizations are correct and complete with respect to PTIME, namely every typed term reduces to normal form in a number of steps which is polynomial in its size, and moreover all and only the polynomial functions can be coded by a typed term. However, the completeness in both systems is functional, not algorithmic, in the sense that for every polynomial function there is at least one algorithm that can be typed; even though the algorithmic completeness is undecidable, we would like to design more expressive systems.

2 The system ISTA

Here we propose a type assignment system for λ-calculus, whose types are *stratified* types, defined as follows.

Definition 1. *i) The set $\mathcal{T}$ of types is defined by the following syntax:*

$$\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A} \quad \textit{(linear types)}$$

$$\sigma ::= \mathtt{A} \mid \{\underbrace{\sigma, ..., \sigma}_{n}\} \quad n > 0 \textit{ (stratified types)}$$

ii) Let $\equiv$ denote the syntactical equality between (stratified) types. Types will be considered modulo the following equivalence relation:

$$\begin{array}{ll} \mathtt{A} \equiv \mathtt{B} \Rightarrow \mathtt{A} = \mathtt{B} & \\ \sigma \multimap \mathtt{A} = \tau \multimap \mathtt{B} & \mathtt{iff}\ \sigma = \tau \quad \mathtt{and} \quad \mathtt{A} = \mathtt{B} \\ \{\sigma_1, ..., \sigma_n\} = \{\tau_1, ..., \tau_m\} & \mathtt{iff}\ \forall \sigma_i . \exists \tau_j . \sigma_i = \tau_j \quad \mathtt{and} \quad \forall \tau_j . \exists \sigma_i . \sigma_i = \tau_j \end{array}$$

i.e., a stratified type represents a set.

iii) The system ISTA *proves judgments of the kind $\Gamma \vdash \mathtt{M} : \sigma$, where Γ is a partial function associating to variables a linear type or a stratified type $\{\sigma_1, ..., \sigma_n\}$ for $n > 0$. $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$. The system is defined in Table 1.*

$$\frac{}{\mathtt{x} : \mathtt{A} \vdash \mathtt{x} : \mathtt{A}}\ (Ax)$$

$$\frac{\Gamma \vdash \mathtt{M} : \mathtt{B} \quad \mathtt{x} \notin \mathrm{dom}(\Gamma)}{\Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{M} : \mathtt{B}}\ (w) \qquad \frac{\Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : \mathtt{B}}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap \mathtt{B}}\ (\multimap I)$$

$$\frac{\Gamma_1 \vdash \mathtt{M} : \sigma \multimap \mathtt{A} \quad \Gamma_2 \vdash \mathtt{N} : \sigma \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \vdash \mathtt{MN} : \mathtt{A}}\ (\multimap E)$$

$$\frac{\Gamma \vdash \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash \mathtt{M} : \forall \mathtt{a}.A}\ (\forall I) \qquad \frac{\Gamma \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Gamma \vdash \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Gamma, \mathtt{x}_1 : \mathtt{A}, ..., \mathtt{x}_n : \mathtt{A} \vdash \mathtt{M} : \tau}{\Gamma \mathtt{x} : \{\mathtt{A}\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau}\ (m_l) \qquad \frac{\Gamma, \mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_n : \sigma_n \vdash \mathtt{M} : \tau \quad \sigma_i \text{ not linear}}{\Gamma, \mathtt{x} : \{\sigma_1, ..., \sigma_n\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau}\ (m_s)$$

$$\frac{\Gamma_i \vdash \mathtt{M} : \sigma_i \quad n \geq 1 \quad \sigma_i \neq \sigma_j}{\cup_i \{\Gamma_i\} \vdash \mathtt{M} : \{\sigma_1, ..., \sigma_n\}}\ (sp)$$

Table 1. The ISTA Type Assignment system.

The system is inspired to STA (in Table 2), where the modality ! has been replaced by the stratification. So, while in STA the multiplexor can contract only premises with the same type, here the stratification allows to contract also different premises, in case of stratified types. In case of linear types the stratification behaves like the !, and this is essential for typing in an uniform way the data types, in particular binary numbers. Stratified types allow us to exploit some good properties of intersection types, for which the intersection is idempotent and commutative, but not associative.

$$\frac{}{\mathtt{x} : \mathtt{A} \vdash_{\mathtt{STA}} \mathtt{x} : \mathtt{A}}\ (Ax) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \quad \mathtt{x} \notin dom\Theta}{\Theta, \mathtt{x} : A \vdash_{\mathtt{STA}} \mathtt{M} : \mu}\ (w) \qquad \frac{\Theta, \mathtt{x} : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{A}}{\Theta \vdash_{\mathtt{STA}} \lambda\mathtt{x}.\mathtt{M} : \mu \multimap \mathtt{A}}\ (\multimap I)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \multimap A \quad \Xi \vdash_{\mathtt{STA}} \mathtt{N} : \mu \quad \Theta \# \Xi}{\Theta, \Xi \vdash_{\mathtt{STA}} \mathtt{MN} : \mathtt{A}}\ (\multimap E) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin FV(\Theta)}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall\mathtt{a}.\mathtt{A}}\ (\forall I)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall\mathtt{a}.\mathtt{B}}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E) \qquad \frac{\Theta, \mathtt{x}_1 : \mu, \ldots, \mathtt{x}_n : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \nu}{\Theta, \mathtt{x} :! \mu \vdash_{\mathtt{STA}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \nu}\ (m) \qquad \frac{\Theta \vdash \mathtt{M} : \mu}{!\Theta \vdash_{\mathtt{STA}} \mathtt{M} :! \mu}\ (sp)$$

Table 2. The STA Type Assignment system.

3 Properties

The system ISTA enjoys the following properties.

Theorem 1. *i) Let $\pi : \Gamma \vdash \mathtt{M} : \sigma$. Then* M *reduces to normal form in a number of steps $\in O(|\mathtt{M}|^{3d})$, where $|\mathtt{M}|$ is the size of* M *and d is the depth of π, i.e., the number of nested applications of rule (sp).*
ii) The system ISTA *givets type to all and only the strongly normalizing terms.*

Observe that these two properties are not in contradiction! So ISTA is more powerful than STA, which can type a proper subset of strongly normalizing terms.

We represent binary numbers in Church style, so the number w is represented by $\underline{w} = \lambda\mathtt{s}_0\mathtt{s}_1\mathtt{x}.\mathtt{s}_{i_1}(...(\mathtt{s}_{i_{\lfloor \log w \rfloor + 1}}\mathtt{x})...)$ where $i_j \in \{0, 1\}$, for any $w \neq 0$, and $\underline{0} = \lambda\mathtt{s}_0\mathtt{s}_1\mathtt{x}.\mathtt{x}$.
In STA, binary numbers are typed uniformely by the type

$$\mathbf{W} = \forall a.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a}$$

and moreover, $\underline{w}$ can be given the type

$$\mathbf{W}_{n,m} = \forall a.!^n(\mathtt{a} \multimap \mathtt{a}) \multimap !^m(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a}\ \text{, for all } n, m \geq 1$$

Observe that the possibility of non uniform typings for binary numbers is essential to limit the expressivity of the language, in that it does not allow nesting iterations of functions. Similarly, in ISTA we define types for binary numbers to be

$$\mathbf{WI}_{n,m} = \forall a.\{^n\mathtt{a} \multimap \mathtt{a}\}^n \multimap \{^m\mathtt{a} \multimap \mathtt{a}\}^m \multimap \mathtt{a} \multimap \mathtt{a}\ \text{, for all } n, m \geq 1$$

so all Church numerals have in particular the type

$$\mathbf{WI} = \forall a.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\} \multimap \mathtt{a} \multimap \mathtt{a}$$

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p. Then the term M represents ϕ in an untyped setting if and only if $\mathtt{M}\underline{n_1}...n_p = \phi(n_1, ..., n_p)$. In a typed setting, M needs to satisfy also typing constraints. Namely, in STA it needs to be typed as

$$\mathtt{x}_1 :!^{i_1}\mathbf{W}_{j_1,k_1}, ..., \mathtt{x}_p :!^{i_p}\mathbf{W}j_pk_p \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{W}_{j,k}$$

for some j, k, p, j_h, k_h $(1 \leq h \leq p)$.

In ISTA the corresponding typing must be

$$\mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_p : \sigma_p \vdash \mathtt{M}\mathtt{x}_1 ... \mathtt{x}_p : \mathbf{WI}_{h,k}$$

such that, for all $i \in \{1, ..., p\}$, either $\sigma_i = \mathbf{WI}_{h_i,k_i}$ (so σ_i is linear), or $\bar{\sigma}_i = \{\mathbf{WI}_{h_1^i,k_1^i}, ..., \mathbf{WI}_{h_{q_i}^i,k_{q_i}^i}\}$, for some $h, k, h_i, k_i, q_i, h_r^i, k_r^i$ $(1 \leq r \leq q_i)$.

Following an approach similar to the one in [4], we prove the following result:

Theorem 2. *The numerical functions definable in* ISTA *are all and only the numerical functions definable in* STA.

The consequences of this theorem are interesting and also quite surprising. The first one is that, as expected, ISTA is sound and complete with respect to PTIME. But also a further notion of completeness arises. In fact ISTA is polynomially complete with respect to strongly normalizing terms, i.e., all the numerical polynomial algorithms that can be expressed by a strongly normalizing term can be typed in it.

References

1. Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.
2. P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS 2004. IEEE Computer Society*, pages 266–275, 2004.
3. P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, 2009.
4. Antonio Bucciarelli, Adolfo Piperno, and Ivano Salvo. Intersection types and lambda-definability. *Mathematical Structures in Computer Science*, 13:15–53, 2003.
5. M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for λ-calculus. In *Computer Science Logic, 21st International Workshop, CSL 07, 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.
6. Marco Gaboardi and Simona Ronchi Della Rocca. From light logics to type assignements: a case study. *Logic Journal of the IGPL, Special Issue on LSFA 2007*, 17:499 – 530, 2009.
7. J-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
8. Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.