Quantitative Perspectives on Generalized Applications

Loïc Peyrot

Université de Paris, CNRS, IRIF, Paris, France
lpeyrot@irif.fr

Joachimski and Matthes λ-*calculus with generalized applications* ΛJ [10, 11] is a Curry-Howard interpretation of von Plato's natural deduction with generalized elimination rules [16]. The difference between ΛJ and the usual λ-calculus lies in the application constructor: its structure is generalized to $t(u, y.r)$, where t, u, r are subterms and y is a variable bound in r. This extended form comes from the generalized implication elimination rule:

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A \qquad \Gamma, y : B \vdash r : C}{\Gamma \vdash t(u, y.r) : C}\ (\to_e)$$

Intuitively, $y.r$ can be seen as a "continuation" waiting for the result of the application of t to u. Once this result is available, it will be substituted in r for y. This can be seen in the following β_j-rule:

$$(\lambda x.t)(u, y.r) \to_{\beta_j} \{\{u/x\}t/y\}r,$$

where the result is simply the β-reduction of $(\lambda x.t)u$. An additional permutative rule π: $t(u, x.r)(u', y.r') \to_\pi t(u, x.r(u', y.r'))$ enables to reduce terms to a *fully normal form*, where the left side of an argument is always a variable, such as in the term $x(u, y.r)$. The calculus ΛJ has its origins in natural deduction, but it also has links with the sequent calculus [8] and calculi with explicit substitutions [12].

In this talk, I look at generalized applications through the lens of *quantitative types*. Quantitative types, also known as non-idempotent intersection types, were introduced by Gardner [9], and later independently by Kfoury [14] and De Carvalho [4]. Like in Coppo and Dezani's idempotent intersection types [3], normalization of some reduction relation $\mathcal{R}$ can be characterized by typability in a quantitative system: a term normalizes for $\mathcal{R}$ *iff* it is typable in an appropriate type system. This semantical flavor is of great use to easily derive equivalences between different operational relations. On top of these qualitative properties, non-idempotence brings a quantitative analysis. In particular, the size of type derivations usually provides an upper bound on the length of the longest reduction sequence to normal form.

An *idempotent* intersection type system for ΛJ was already defined by Matthes [15]. But the change of perspective to non-idempotence gives new insights on the calculus and leads us to define a variant λJ_n[1], differing from the original calculus in two main ways.

First, we use a *distant* β_j ($\mathtt{d}\beta_j$) rule, where distance is a paradigm we adopt from calculi with explicit substitutions [1]. As in these calculi, some redexes of ΛJ can be hidden by the syntax, such as in the term $x(u, y.\lambda z.r)(u', y'.r')$, where the computation between $\lambda z.r$ and u' is stuck. Conversion π enables to unblock this redex:

$$x(u, y.\lambda z.r)(u', y'.r') \to_\pi x(u, y.(\lambda z.r)(u', y'.r')) \to_{\beta_j} x(u, y.\{\{u'/z\}r/y'\}r').$$

But in our approach, permutations are simply considered as a tool to overcome syntactical limitations, leaving the computational content inside the β_j-reduction itself. Indeed, only β_j-reductions create divergence. Such an approach is motivated by graphical formalisms such as

[1] The subscript n emphasizes the call-by-name nature of the calculus, as a promising call-by-value calculus with generalized applications was recently defined by Espírito Santo [5].

proof nets, where these kinds of syntactical obstructions do not occur. Concretely, we only integrate the necessary permutations in the distant β_j rule, so that the calculus relies on a unique computational step. This is more relevant for the quantitative analysis, since typing should not be affected by permutations.

The second divergence between our approach and ΛJ is that distance is not in fact based on the usual permutation π. This rule is not sound for a quantitative semantics: subject reduction fails in our type system. Intuitively, π introduces a sharing of subterms fit for a call-by-value calculus, rather than call-by-name like ΛJ. While in the call-by-name λ-calculus it is enough to reject permutative rules altogether, the situation is a bit more complex with generalized applications: we still need to unblock redexes. We choose to base the distant rule on a different permutation called p_2: $t(u, y.\lambda x.r) \to_{p_2} \lambda x.t(u, y.r)$, originating from previous studies on generalized applications [6]. Our previous example is then reduced in one step as: $x(u, y.\lambda z.r)(u', y'.r') \to_{\mathtt{d}\beta_j} \{x(u, y.\{u'/z\}r)/y'\}r'$.

In summary, we replace the two original rules β_j and π by a single computational one based on β_j and permutation p_2. We show that the new calculus λJ_n is sound and complete with respect to the quantitative semantics by characterizing strong normalization with a quantitative type system. We also show the equivalence of strong normalization for ΛJ and λJ_n, by introducing inductive definitions of strong normalization. These results appear in [7], in collaboration with José Espírito Santo and Delia Kesner.

After helping in revisiting the dynamics of the general calculus, quantitative types are still useful to analyze finer reduction relations. A notable one is *head* reduction, which characterizes *solvability* operationally: a term is solvable *iff* it has a head-normal form [17]. Solvability is a crucial property identifying *meaningful* terms [2], *i.e.* terms that can influence the observational behavior of a reduction. Denotationally, a model of the λ-calculus identifying all unsolvable terms (*e.g.* as bottom) is consistent, while identifying only non-normalizing terms is not. In [13], Delia Kesner and I give a reduction relation, generalizing head reduction, which characterizes an original notion of solvability for generalized applications operationally. We then define an appropriate quantitative type system. This logical characterization enables us to prove preservation of solvability to and from the λ-calculus, by a simple proof of preservation of typability.

References

[1] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In *Computer Science Logic*, pages 381–395. Springer Berlin Heidelberg, 2010.

[2] Henk Pieter Barendregt. *The Lambda Calculus - Its Syntax and Semantics*. Elsevier, 1984.

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

[4] Daniel De Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, January 2017.

[5] José Espírito Santo. The call-by-value lambda-calculus with generalized applications. In *28th EASCL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2020.

[6] José Espírito Santo and Luís Pinto. Permutative conversions in intuitionistic multiary sequent calculi with cuts. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, pages 286–300. Springer, 2003.

[7] José Espírito Santo, Delia Kesner, and Loïc Peyrot. A faithful and quantitative notion of distant reduction for generalized applications. In *Proc. of the 24th Int. Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, LNCS, 04 2022.

[8] José Espírito Santo and Luís Pinto. A calculus of multiary sequent terms. *ACM Transactions on Computational Logic*, 12(3):1–41, May 2011.

[9] Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Lecture Notes in Computer Science*, pages 555–574. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[10] Felix Joachimski and Ralph Matthes. Standardization and confluence for a lambda calculus with generalized applications. In *Rewriting Techniques and Applications*, pages 141–155. Springer Berlin Heidelberg, 2000.

[11] Felix Joachimski and Ralph Matthes. Short proofs of normalization for the simply-typed λ-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42(1):59–87, 2003.

[12] Delia Kesner. A theory of explicit substitutions with safe and full composition. *Logical Methods in Computer Science*, 5(3), 2009.

[13] Delia Kesner and Loïc Peyrot. Solvability for generalized applications. Unpublished, 2022.

[14] Assaf Kfoury. A linearization of the lambda-calculus and consequences. *Journal of Logic and Computation*, 10(3):411–436, 2000.

[15] Ralph Matthes. Characterizing strongly normalizing terms for a lambda calculus with generalized applications via intersection types. In *Proc. ICALP 2000 (Geneva), volume 8 of Proceedings in Informatics*, pages 339–353, 2000.

[16] Jan von Plato. Natural deduction with general elimination rules. *Archive for Mathematical Logic*, 40(7):541–567, 2001.

[17] Christopher P. Wadsworth. The relation between computational and denotational properties for scott's D_∞-models of the lambda-calculus. *SIAM Journal on Computing*, 5(3):488–521, September 1976.