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:

\[
\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A \quad \Gamma, y : B \vdash r : C}{\Gamma \vdash t(u, y. r) : C}
\]

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 βj-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 \( \pi \): \( t(u, x.r)(u', y.r') \to_{\pi} t(u, x.r(u', y.r')) \) 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 \( R \) can be characterized by typability in a quantitative system: a term normalizes for \( R \) 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 \( \Lambda J_n \)\(^1\), differing from the original calculus in two main ways.

First, we use a distant βj (dβj) 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 \( \pi \) 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 \( \beta_j \)-reduction itself. Indeed, only βj-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 \( \beta_j \) 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 \( \Lambda J \) is that distance is not in fact based on the usual permutation \( \pi \). This rule is not sound for a quantitative semantics: subject reduction fails in our type system. Intuitively, \( \pi \) introduces a sharing of subterms fit for a call-by-value calculus, rather than call-by-name like \( \Lambda J \). While in the call-by-name \( \lambda \)-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 \( p_2: t(u,y,\lambda z.r) \rightarrow_{p_2} \lambda x.t(u,y,r) \), originating from previous studies on generalized applications [6]. Our previous example is then reduced in one step as: 
\[
x(u,y,\lambda z.r)(u',y',r') \rightarrow_{\beta_j} x(u,y,\{u'/z\}r)/y' r'.
\]

In summary, we replace the two original rules \( \beta_j \) and \( \pi \) by a single computational one based on \( \beta_j \) and permutation \( p_2 \). We show that the new calculus \( \lambda J_n \) 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 \( \Lambda J \) and \( \lambda J_n \), 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 \( \lambda \)-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 \( \lambda \)-calculus, by a simple proof of preservation of typability.

References


