Atomic Lambda Calculus and its connections with Sharing Graphs

David R. Sherratt

The purpose of sharing in the λ-calculus is to have better control over duplication. Sharing is the use of a single representation for multiple instances of a common subterm. During evaluation, instead of duplicating a term, we can share it. This allows us to evaluate all the copies of the subterm simultaneously, by evaluating their shared representation [5]. Sharing is implemented using various techniques, including `let`-expressions in term calculi. In the atomic λ-calculus [2], sharing is expressed as $u[y_1, ..., y_n \leftarrow t]$ where t is being shared in u, and the variables y_1 through y_n in u represent the shared instances of t.

With sharing, duplication can be delayed until the point where the instances of a shared function are applied to distinct arguments. Since a function is destroyed upon evaluation, at this point duplication becomes necessary. The question is then how much do we duplicate: do we need to copy the whole term or can parts remain shared? With simple sharing mechanisms such as the above, the maximum that can remain shared upon duplication are the *maximal free subexpressions* (MFS) of a term (the largest subterms that, if they contain a variable x, they also contain its binder λx). A formalism that duplicates all but the maximal free subexpressions of a term is called *fully lazy*.

Atomic λ-calculus achieves full laziness by effectively integrating the duplication process with MFS-extraction. MFS-extraction is obtained through the duplication process which is atomic. At any point, a shared term is either a MFS where it is permuted so it is not duplicated but shared, or it is not a MFS in which case we begin duplicating.

Atomic λ-calculus extends λ-calculus with explicit sharing and a *distributor* construct [2]. The introduction of the distributor allows us to duplicate terms atomically and refine the computation of terms into smaller steps. The distributor works directly on the λ-abstractions $\lambda x.t$. The calculus uses the distributor to duplicate the body t n times into the tuple $\langle t_1, .., t_n \rangle$ while maintaining one copy of the constructor to obtain $\lambda x.\langle t_1, .., t_n \rangle$, and then to distribute to obtain n copies of $\lambda x.t$. Atomic λ-calculus duplicates terms via sharing reductions:

$$\lambda x.t \;\rightsquigarrow\; \lambda x.\langle y_1, ..., y_n \rangle [y_1, ..., y_n \leftarrow t] \;\rightsquigarrow^*\; \lambda x.\langle t_1, ..., t_n \rangle \;\rightsquigarrow\; \lambda x.t, ..., \lambda x.t$$

The calculus is a Curry-Howard interpretation of a deep inference [1] proof system for intuitionistic logic. The *distribution rule* enables the atomicity property in deep inference: it allows a contraction on an implication to be reduced locally i.e. without duplicating the whole subproof. This property is what allows the introduction of the distributor in term calculus, a computational interpretation of the distribution rule (a combination of the medial rule and the co-contraction rule).

$$\frac{A \to (B \wedge C)}{(A \to B) \wedge (A \to C)}\, d \qquad\qquad \frac{A \to B}{(A \to B) \wedge (A \to b)}\, c \;\sim\; \frac{\mathrm{A} \to \dfrac{B}{B \wedge B}\, c}{(A \to B) \wedge (A \to B)}\, d$$

A natural intuitive graphical interpretation of atomic λ-calculus is used for illustration purposes, but they have interesting similarities with sharing graphs [3]. Sharing graphs implement λ-expression reduction that avoid any copying that could later cause duplication of work, an *optimal* [4] algorithm for λ-calculus reduction. This is achieved through graph reduction techniques. However, there are significant differences between the two. The calculus is fully lazy but not optimal like sharing graphs, however the calculus has global typing unlike sharing graphs which have only local typing.

The aims of my research is to make precise the connection between the atomic λ-calculus and sharing graphs. We wish to make the intuitive graphs used to illustrate atomic λ-calculus more precise, formally connecting the atomic λ-calculus and sharing graphs. To achieve this one would have to refine the control mechanisms for the atomic λ-calculus i.e. explore efficient control mechanisms for a graph-based version of the atomic λ-calculus. This will allow us to compare more directly the control mechanisms of the calculus and sharing graphs: to characterize the distinction and similarities with the formal graphical illustration of atomic λ-calculus and sharing graphs, and to explore the area between full laziness and optimality.

References

[1] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. *A proof calculus which reduces syntactic bureaucracy.* (2010): 135-150.

[2] Tom Gundersen, Willem Heijltjes, and Michel Parigot. *Atomic lambda calculus: A typed lambda-calculus with explicit sharing.* Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science. IEEE Computer Society, 2013.

[3] John Lamping. *An algorithm for optimal lambda calculus reduction.* Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1989.

[4] Jean-Jacques Lévy. *Optimal reductions in the lambda-calculus.* To HB Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism (1980): 159-191.

[5] Christopher Peter Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* Diss. University of Oxford, 1971.