Atomic lambda-calculus:
a typed lambda-calculus with explicit sharing

Tom Gundersen
CNRS & Université Paris Diderot
Bât. Sophie Germain, 8 Place FM/13
75013 Paris
teg@jklm.no

Willem Heijltjes
University of Bath
Claverton Down
Bath BA2 7AY
w.b.heijltjes@bath.ac.uk

Michel Parigot
CNRS & Université Paris Diderot
Bât. Sophie Germain, 8 Place FM/13
75013 Paris
parigot@pps.univ-paris-diderot.fr

***Abstract*—An explicit–sharing lambda-calculus is presented, based on a Curry–Howard-style interpretation of the deep inference proof formalism. Duplication of subterms during reduction proceeds 'atomically', i.e. on individual constructors, similar to optimal graph reduction in the style of Lamping. The calculus preserves strong normalisation with respect to the lambda-calculus, and achieves fully lazy sharing.**

I. Introduction

Computation in the lambda-calculus is notoriously unpredictable in its use of computational resources. Central to restraining unnecessary computation within terms is to exercise strategic control over duplication and deletion: to perform computation within a subterm before it is duplicated, and to avoid computation within subterms that will be deleted.

Several formalisms have been developed to deal with this problem. Firstly, there are the *explicit-substitution calculi* [1]: term calculi that allow greater control over the substitution process, inherent in β-reduction, by representing substitutions explicitly in the syntax of the calculus. Secondly, there is the study of *explicit sharing*, which includes various techniques to allow the sharing of common subterms within a λ-term, such as *pointer graphs* [17], *labelling systems*, and *optimal reduction graphs* [13]; see also [7] or [5] for an overview. Many of them can be expressed in term calculi, using for instance `let`-expressions (see e.g. [11], [4]).

This paper presents a term calculus of the latter kind, the *atomic lambda-calculus*, based on a Curry–Howard interpretation of a deep-inference proof system for intuitionistic logic. It is a linear lambda-calculus with a `let`-construct, called a *sharing*, extended with a further construction, the *distributor*. The distributor is a computational interpretation of the medial-rule of deep inference [8], that allows reduction steps to be *atomic*, i.e. apply to individual constructors.

The atomic lambda-calculus preserves strong normalisation with respect to the lambda-calculus (Theorem 27), preserves typing under translation with the lambda-calculus (Proposition 2) and preserves typing under reduction (Theorem 12). Additionally, it implements, in a natural way, a form of fully lazy sharing [17], [11] (Proposition 31).

Many individual components of the atomic lambda-calculus, such as the sharing-construct and many rewrite rules, are familiar from the literature, in particular from functional programming (see e.g. [14]) and from recent explicit-substitution calculi based on proof nets [12], [3]. The main innovation is the distributor, and the rewrite rules that govern it (rules 6, 10, and 11 in Section IV). These enable the duplication of an abstraction λx in a term $\lambda x.t$ independently of the body t, replacing the abstraction by a distributor while duplication of t is in progress. As a consequence of small-step (as opposed to big-step) duplication, the global rewriting dynamics of the atomic lambda-calculus differ markedly from that of explicit-substitution calculi. Details will be provided in Section IV, where also the divergence with optimal reduction graphs will be illustrated.

A preliminary version of this work has been presented in French in [10].

II. Deep inference

The distinguishing feature of deep-inference formalisms is that logical inference is applied within context, i.e. that it is not restricted to the main connective. An alternative approach consists in applying logical connectives at the level of derivations as well as formulae. The *open deduction* formalism [9] is designed around this principle. In open deduction, a *derivation* from a *premise* A to a *conclusion* C (over the connectives *conjunction* and *implication*) is constructed as follows,

$$\begin{array}{c} A \\ \Downarrow \\ C \end{array} \quad := \quad A \quad | \quad \begin{array}{c} A_1 \quad A_2 \\ \Downarrow \quad \Downarrow \\ C_1 \quad C_2 \end{array} \wedge \quad | \quad \begin{array}{c} A_1 \quad A_2 \\ \Downarrow \quad \Downarrow \\ C_1 \quad C_2 \end{array} \rightarrow \quad | \quad \begin{array}{c} A \\ \Downarrow \\ \dfrac{B}{B'}\,r \\ \Downarrow \\ C \end{array}$$

with from left to right: (i) a *formula* where $C = A$; (ii) a *conjunction* where $A = A_1 \wedge A_2$ and $C = C_1 \wedge C_2$; (iii) an *implication* where $A = C_1 \rightarrow A_2$ and $C = A_1 \rightarrow C_2$ (note that, for the negative part - the antecedent - of the implication, premise and conclusion are inverted); and (iv) a *rule-composition*, where a derivation from A to B and one from B' to C are combined using an *inference rule* r from B to B'. The general composition of a derivation from A to B and one from B to C is a derived operation.

Open deduction is a general formalism, like the sequent calculus is, in which proof systems for different logics can be formulated by choosing a particular collection of connectives

and inference rules. Here, we consider a formulation of minimal logic obtained by simply embedding its usual natural deduction system in open deduction. The dynamics will nevertheless be different, because the proof normalisation possibilities offered by open deduction will be used. The basic inference rules are *abstraction*, *application*, and (n-ary) *contraction*, below.

$$\frac{B}{A \to (A \wedge B)}\,\lambda \qquad \frac{A \wedge (A \to B)}{B}\,@ \qquad \frac{A}{A \wedge \ldots \wedge A}\,\vartriangle$$

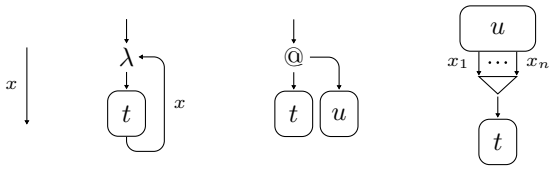
This deductive system provides a typing of the following *basic calculus*, consisting of linear lambda terms with explicit contractions, that forms the basis of the atomic lambda-calculus.

Definition 1. The *basic calculus* Λ_a^- is given by the grammar

$$t, u, v \;:=\; x \;\mid\; \lambda x.t \;\mid\; (t)u \;\mid\; u[x_1, \ldots, x_n \leftarrow t]$$

where (i) each variable may occur at most once, (ii) in $\lambda x.t$ the variable x must occur in t and becomes bound, and (iii) in $u[x_1, \ldots, x_n \leftarrow t]$ each x_i must occur in u and becomes bound. Terms of the basic calculus are called *basic terms*.

The four constructors are called *variable*, *abstraction*, *application*, and *sharing* respectively, and are displayed graphically below. A nullary sharing $u[\leftarrow t]$ is called a *weakening*.



In Figure 1 a system of typing derivations for the basic calculus is given, in the open deduction formalism, constituting a deduction system for minimal logic. Note that derivations are rotated 180° compared to the graphical depictions of the terms. For a given set $\{a, b, c, \ldots\}$ of *atomic formulae*, the following two grammars define *minimal formulae* and *conjunctive formulae* respectively.

$$A, B, C \;:=\; a \;\mid\; A \to B \qquad\qquad \Gamma, \Delta \;:=\; A \;\mid\; \top \;\mid\; \Gamma \wedge \Delta$$

In the type system for the basic calculus, as shown in Figure 1, a variable x may be typed by any minimal formula A, while the other constructors each correspond to an inference rule, used within the context of further derivations. A term t with free variables $x_1, \ldots, x_n$ is typed by a derivation from an assumption $A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}$ to a conclusion C; a *typing judgement* $t : C$ will express that t is typeable by a derivation with conclusion C. In addition, it can be observed that a derivation occurring as the antecedent of an implication is always a minimal formula.

Before introducing the atomic lambda-calculus, the basic calculus will be related to the standard lambda-calculus (Λ), defined below (using a distinct alphabet for clarity).

$$M, N \quad := \quad x \quad \mid \quad \lambda x.N \quad \mid \quad (N)M$$

Where applicable, all variables will be assumed fresh.

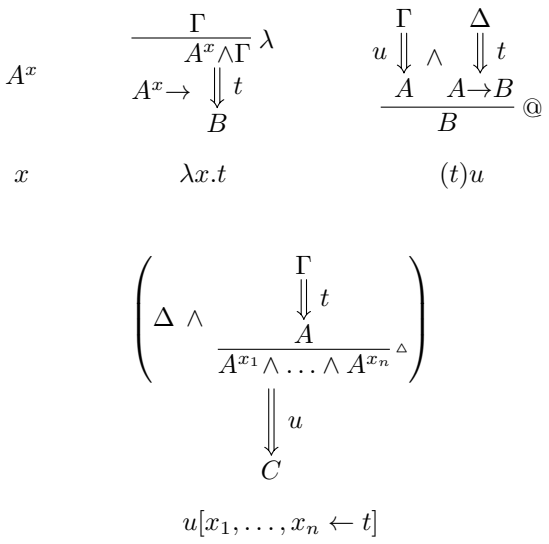


Fig. 1. Typing the basic calculus

The function $(\!|-|\!) : \Lambda \to \Lambda_a^-$ interprets lambda-terms as basic terms. Intuitively, it replaces each abstraction $\lambda x.-$ in a term by $\lambda x.-[x^1, \ldots, x^n \leftarrow x]$, where $x^1, \ldots, x^n$ replace the occurrences of x. Let $|N|_x$ denote the number of occurrences of x in N, and if $|N|_x = n$ let $N\frac{n}{x}$ denote N with the occurrences of x replaced by fresh, distinct variables $x^1, \ldots, x^n$. First, the translation of a *closed* term N is $(\!|N|\!)'$, defined below.

$$(\!|x|\!)' = x \qquad\qquad (\!|(M)N|\!)' = ((\!|M|\!)')(\!|N|\!)'$$

$$(\!|\lambda x.N|\!)' = \begin{cases} \lambda x.(\!|N|\!)' & \text{if } |N|_x = 1 \\ \lambda x.(\!|N\frac{n}{x}|\!)'[x^1, \ldots, x^n \leftarrow x] & \text{if } |N|_x = n \neq 1 \end{cases}$$

For an arbitrary term N, if $x_1, \ldots, x_k$ are the free variables of N such that $|N|_{x_i} = n_i > 1$, the translation $(\!|N|\!)$ is

$$(\!|N\frac{n_1}{x_1} \ldots \frac{n_k}{x_k}|\!)'[x_1^1, \ldots, x_1^{n_1} \leftarrow x_1] \ldots [x_k^1, \ldots, x_k^{n_k} \leftarrow x_k]$$

Note that in the case of a linear abstraction $\lambda x.t$, where x has a single occurrence in t, no sharing $[x^1 \leftarrow x]$ is introduced in the translation. This anticipates the evaluation of a unary sharing $u[x \leftarrow t]$ by a linear substitution of t for x (rule 7 in Section IV). Closed terms in the image of $(\!|-|\!)$ are the closed basic terms generated by the grammar below.

$$t \;:=\; x \;\mid\; \lambda x.t \;\mid\; (t)u \;\mid\; \lambda x.t[x_1, \ldots, x_n \leftarrow x] \quad (n \neq 1)$$

Proposition 2. *For any λ-term M, if $M : A$ then $(\!|M|\!) : A$.*

III. The atomic lambda-calculus

The atomic lambda-calculus will extend the basic calculus with an additional construct, based on two further, standard, inference rules, *co-contraction* and *medial* [16]:

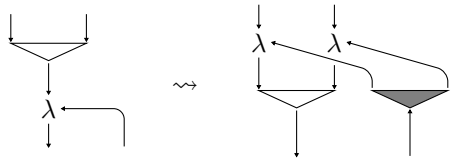
$$\frac{A \vee \ldots \vee A}{A}\,\blacktriangledown \qquad\qquad \frac{(A_1 \vee A_2) \to (B_1 \wedge B_2)}{(A_1 \to B_1) \wedge (A_2 \to B_2)}\,\mathsf{m}$$

The medial rule is a linearised version of a distribution law, where only two of the four possible conjuncts $A_i \to B_j$ are

retained in the conclusion. In deep inference, the medial rule is the one which enables to reduce contractions to their atomic case [8]. The two rules allow the following rewrite step in open deduction.

$$\dfrac{\dfrac{A \to B}{(A \to B) \wedge (A \to B)}\,\vartriangle}{}\quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{A \vee A}{A}\,\blacktriangledown \;\to\; \dfrac{B}{B \wedge B}\,\vartriangle}{(A \to B) \wedge (A \to B)}\,\mathsf{m}}{}$$

Here, a contraction inference on a formula $A \to B$ is replaced by a medial, a contraction on B (the consequent of the implication), and a co-contraction (its antecedent). The reduction step can be seen to correspond to the duplication of a λ-node in optimal reduction graphs.



The white *fan-in* nodes correspond to the contractions and the grey *fan-out* node corresponds to the co-contraction, while the assumption $A \to B$ in the derivation above left may reasonably be represented by a λ-node. The medial corresponds to the two λ-nodes in the graph on the right taken together.

To avoid introducing disjunctions and to maintain the restriction that the antecedent of an implication is a minimal formula, instead of co-contraction and medial the following *distribution* rule (d) is used, combining the former two.

$$\dfrac{A \to (B \wedge C)}{(A \to B) \wedge (A \to C)}\,\mathsf{d} \qquad \sim \qquad \dfrac{\dfrac{A \vee A}{A}\,\blacktriangledown \;\to (B \wedge C)}{(A \to B) \wedge (A \to C)}\,\mathsf{m}$$

The proof reduction step to be implemented in the atomic lambda-calculus is as follows (rewrite rule (10) in Section IV).

$$\dfrac{A \to \dfrac{\dfrac{B}{A \wedge B}\,\lambda}{C}}{(A \to C) \wedge (A \to C)}\,\vartriangle \qquad \rightsquigarrow \qquad \dfrac{\dfrac{B}{A \wedge B}\,\lambda}{(A \to C) \wedge (A \to C)}\,\mathsf{d}$$

The point of doing this is the following. Above left a contraction is applied to a subderivation for a term $\lambda x.t$ The reduction step allows to push the contraction past the abstraction, inside the body t, without duplicating any part of t itself. The abstraction λx becomes *shared* between the two conjuncts in $C \wedge C$, while the contraction continues to duplicate t stepwise. When it reaches the λ-inference at the top, the distribution inference can be eliminated by the reduction step below (omitting the assumption of the λ-rule for simplicity), corresponding to rewrite rule (11) in Section IV.

$$\dfrac{A \to \dfrac{\dfrac{}{A}\,\lambda \quad \dfrac{}{A}\,\lambda}{C \quad C}\,\vartriangle}{(A \to C) \wedge (A \to C)}\,\mathsf{d} \qquad \rightsquigarrow \qquad \dfrac{}{A \to C}\,\lambda \;\wedge\; \dfrac{}{A \to C}\,\lambda$$

During the duplication process, some subderivations can be permuted above the λ-inference in the way illustrated below, corresponding to rewrite rule (6) in Section IV. These need not be duplicated, but can remain shared, a feature that is central to obtaining fully lazy sharing.

$$\dfrac{B}{A \to \left(A \wedge \begin{matrix} B \\ \Downarrow \\ C \end{matrix}\right)}\,\lambda \qquad \rightsquigarrow \qquad \dfrac{\begin{matrix} B \\ \Downarrow \\ C \end{matrix}}{A \to (A \wedge C)}\,\lambda$$

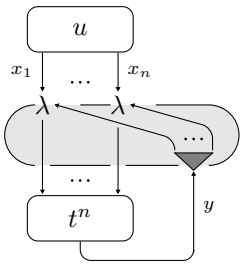
To implement reductions of this kind, two constructs are needed: one corresponding to the distribution inference, and one to allow terms of conjunction type.

Definition 3. The *atomic lambda-calculus* Λ_a extends the basic calculus with the *distributor* constructor. The following mutually recursive grammars simultaneously define the *atomic lambda terms* and the *terms of multiplicity* n, or n*-terms*, for every $n > 0$, written t^n.

$$\begin{array}{rcll} t, u, v & := & \ldots & \mid \; u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n] \quad {}^{*} \\ t^n & := & \langle t_1, \ldots, t_n \rangle & \mid \; t^n[x_1, \ldots, x_m \leftarrow u] \quad {}^{**} \\ & & & \mid \; t^n[x_1, \ldots, x_m \twoheadleftarrow \lambda y.t^m] \quad {}^{***} \end{array}$$

where (i) each variable may occur at most once, (ii) in (*) each variable x_i must occur in u and becomes bound, and y must occur in t^n and becomes bound, and (iii) in (**) and (***) each variable x_i must occur in t^n and becomes bound.

The distributor is illustrated below; it is depicted as a box encompassing n lambda-nodes and a co-contraction (or *co-sharing*), emphasising the connection with optimal reduction graphs. An n-term is drawn as a subgraph with n input wires.



The crucial problem in making optimal reduction graphs possible, first solved in [13], is to decide when a sharing meets a co-sharing whether they duplicate one another, or annihilate. In the atomic lambda-calculus the distributor, containing the co-sharing, maintains its own scope, making this decision problem trivial. On the other hand, the lambda-nodes contained in the distributor box, unlike in optimal reduction graphs, cannot be part of a redex.

The sharing and distributor constructors together will be referred to as *closures* and abbreviated $[\gamma]$,$[\delta]$; a sequence of closures will be denoted $[\Gamma]$. An n-term is of the form $\langle t_1, \ldots, t_n \rangle[\Gamma]$: a sequence of closures applied to an n-tuple of atomic lambda-terms. The i^{th} *projection* $\pi_i(t^n)$ of an n-term $t^n = \langle t_1, \ldots, t_n \rangle[\Gamma]$ is the smallest atomic lambda-term

$$t_i[\leftarrow x_1] \ldots [\leftarrow x_m][\Gamma] \;,$$

i.e. $x_1, \ldots, x_m$ are the free variables of all t_j ($i \neq j$) that are bound by $[\Gamma]$.

Typing derivations for the distributor and for tuples are given in Figure 2. The derivations for the two constructors $t^n[\gamma]$, closures on n-terms, coincide with those for closures on atomic lambda terms ($u[\gamma]$) when the conclusion C is replaced by $C_1 \wedge \ldots \wedge C_n$ (and u by t^n).

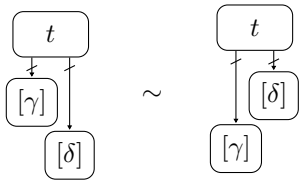
The function $[\![-]\!] : \Lambda_a \to \Lambda$, called *denotation*, maps the atomic lambda-calculus onto the lambda-calculus. Actual substitutions (of t for x in u) are denoted $u\{t/x\}$, and a family of substitutions $\{t_1/x_1\}\ldots\{t_n/x_n\}$ is denoted $\{t_i/x_i\}_{1\leq i\leq n}$.

$$
\begin{aligned}
[\![x]\!] &= x \\
[\![\lambda x.u]\!] &= \lambda x.[\![u]\!] \\
[\![(u)t]\!] &= ([\![u]\!])[\![t]\!] \\
[\![u[x_1, \ldots, x_n \leftarrow t]]\!] &= [\![u]\!]\{[\![t]\!]/x_i\}_{1\leq i\leq n} \\
[\![u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n]]\!] &= [\![u]\!]\{\lambda y.[\![\pi_i(t^n)]\!]/x_i\}_{1\leq i\leq n}
\end{aligned}
$$

Atomic lambda terms will be considered up to the congruence induced by (1) below, denoted ($\sim$).

$$t[\gamma][\delta] \;\sim\; t[\delta][\gamma] \tag{1}$$

Note that due to the uniqueness of variables, both terms are only well-defined if both $[\gamma]$ and $[\delta]$ bind only in t; the congruence is then illustrated below. A barred arrow ($\downarrow$) denotes a bundle of wires, and is used in place of ellipsis where space is scarce.



The following facts are easily observed.

Proposition 4. *If $u \sim v$ then $[\![u]\!] = [\![v]\!]$.*

Proposition 5. *If $u \sim v$ and $u : A$, then $v : A$.*

Although the atomic lambda-calculus has its roots in distinguishing features of the open deduction formalism, a sequent-calculus typing system is available, displayed in Figure 3, for which subject reduction holds [10]. There, the contexts Γ and Δ denote sets of typing judgements on variables, $x : A$. A proof of a sequent $x_1 : A_1, \ldots x_n : A_n \vdash t : C$ can be seen to construct an open-deduction derivation from $A_1 \wedge \ldots \wedge A_n$ to C for the term t. The inference rules for closures applied to terms of multiplicity n are similar to the regular rules for closures, replacing u with t^n and C with $C_1 \wedge \ldots \wedge C_n$.

IV. Sharing Reductions

The central feature of the atomic lambda-calculus is to implement reduction steps that are atomic, i.e. that apply to individual constructors. While there are obvious similarities in notation and design between the atomic lambda-calculus and explicit substitution calculi, the dynamics of the former is in some sense orthogonal to that of the latter. The difference in dynamics is given by the reduction rules for closures, which will be discussed over the course of this section.

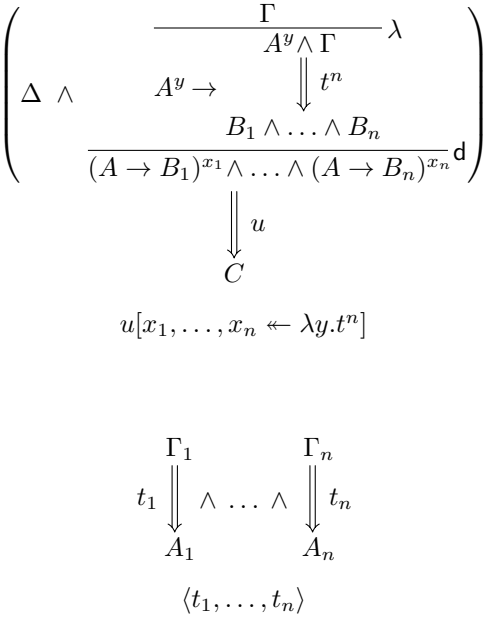


Fig. 2. Typing the atomic lambda-calculus

Definition 6. The relation $\rightsquigarrow_S$ is the rewrite relation on Λ_a induced by rewrite rules (2)–(11).

The first set of rewrite rules, (2)–(6) below, moves closures towards the ouside of the term; in particular, they can be brought out of the scope of a distributor, allowing rule (11) to be applied. The direction of rewriting is directly opposite that of explicit-substitution calculi, in which explicit substitutions move towards the inside of the term. Moreover, the rewrite rules (2)–(6) are equalities on the graphical representation, while rewriting in explicit-substitution calculi is not, as it may duplicate closures. In view of a closure as abstract notation representing a redex, the rewrite rules (2)–(6) correspond to *permutations* as explored for the standard lambda-calculus in [15], [2].

$$\lambda x.t[\gamma] \rightsquigarrow_S (\lambda x.t)[\gamma] \quad \text{if } x \in \mathsf{FV}(t) \tag{2}$$
$$(u[\gamma])t \rightsquigarrow_S ((u)t)[\gamma] \tag{3}$$
$$(u)t[\gamma] \rightsquigarrow_S ((u)t)[\gamma] \tag{4}$$
$$u[x_1, \ldots, x_n \leftarrow t[\gamma]] \rightsquigarrow_S u[x_1, \ldots, x_n \leftarrow t][\gamma] \tag{5}$$
$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t[\gamma]] \rightsquigarrow_S u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t][\gamma] \quad \text{if } y \in \mathsf{FV}(t) \tag{6}$$

The presence of the rules (2)–(6) is justified by the need to lift closures out of a distributor, as opposed to duplicating them. This requires to lift closures out of every constructor, except tuples, for which no rule is added. In a second set of

$$\frac{}{x : A \vdash x : A}\,\mathsf{ax} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,\lambda \qquad \frac{\Gamma \vdash t : A \to B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash (t)u : B}\,@ \qquad \frac{\Gamma_1 \vdash t_1 : A_1 \qquad \cdots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma_1, \ldots, \Gamma_n \vdash \langle t_1, \ldots, t_n \rangle : A_1 \wedge \ldots \wedge A_n}\,\langle\,\rangle$$

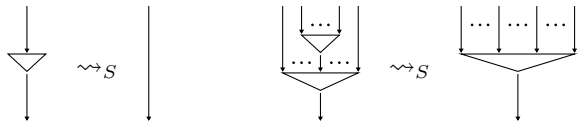
$$\frac{\Gamma, x_1 \ldots x_n : A \vdash u : C \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash u[x_1, \ldots, x_n \leftarrow t] : C}\,\triangle \qquad \frac{\Gamma, x_1 : A \to B_1, \ldots, x_n : A \to B_n \vdash u : C \qquad \Delta, y : A \vdash t^n : B_1 \wedge \ldots \wedge B_n}{\Gamma, \Delta \vdash u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n] : C}\,\mathsf{d}$$

Fig. 3. Sequent calculus inference rules for the atomic lambda-calculus

rewrite rules, unary sharings are applied as substitutions, and consecutive sharings are compounded, illustrated below.

$$u[x \leftarrow t] \;\rightsquigarrow_S\; u\{t/x\} \tag{7}$$

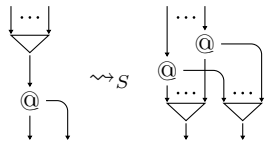
$$u[x_1, \ldots, x_n \leftarrow y_i][y_1, \ldots, y_m \leftarrow t] \;\rightsquigarrow_S\; u[y_1, \ldots, y_{i-1}, x_1, \ldots, x_n, y_{i+1}, \ldots, y_m \leftarrow t] \tag{8}$$



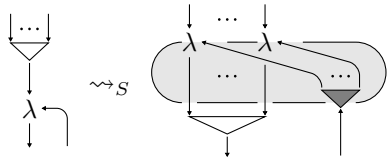
Note that while unary sharings, in rule (7), do not occur in terms translated from the lambda-calculus, they are created during reduction, e.g. by combining a binary sharing with a weakening in rule (8).

The atomic duplication steps central to the calculus are given in a third set of rewrite rules, (9)–(11), below, each followed by a graphical illustration. The definition of rule (11) uses the abbreviation $\vec{z}$ for $z_1, \ldots, z_k$.

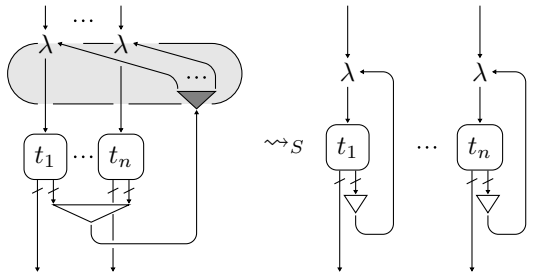
$$u[x_1, \ldots, x_n \leftarrow (v)t] \;\rightsquigarrow_S\; u\{(y_i)z_i/x_i\}_{1 \le i \le n}[y_1, \ldots, y_n \leftarrow v][z_1, \ldots, z_n \leftarrow t] \tag{9}$$



$$u[x_1, \ldots, x_n \leftarrow \lambda x.t] \;\rightsquigarrow_S\; u[x_1, \ldots, x_n \twoheadleftarrow \lambda x.\langle y_1, \ldots, y_n \rangle[y_1, \ldots, y_n \leftarrow t]] \tag{10}$$



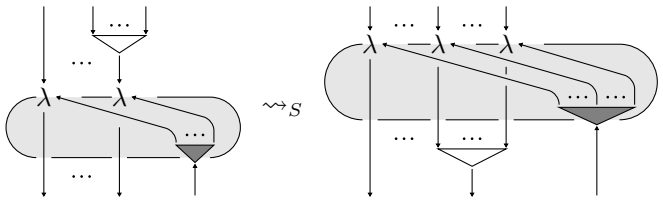
$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle t_1, \ldots, t_n \rangle[\vec{z} \leftarrow y]] \;\rightsquigarrow_S\; u\{\lambda y_i.t_i[\vec{z}_i \leftarrow y_i]/x_i\}_{1 \le i \le n} \quad \text{where } \{\vec{z}_i\} = \{\vec{z}\} \cap \mathsf{FV}(t_i) \text{ for every } i \le n \tag{11}$$



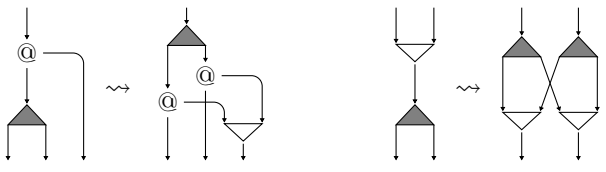
The graphical illustrations may help seeing why the reduction rules are (almost) exhaustive: in (9), a sharing meets an application; in (10), a sharing meets an abstraction; and in (11) a sharing meets a co-sharing, at the end of the scope of a distributor. The remaining case is when a sharing meets a distributor, in a term

$$u[x_1, \ldots, x_n \leftarrow y_i][y_1, \ldots, y_m \twoheadleftarrow \lambda z.t^m] \;.$$

The appropriate reduction step would be the one illustrated below: it closes the confluence diagram where two sharing nodes, one above another, meet an abstraction. The case is redundant, as the distributor will eventually be eliminated, and for simplicity the corresponding rule will be omitted.



In several aspects, the graphical representation of the reduction in the atomic lambda-calculus resembles sharing-graph reduction. However, there are a few differences, which lead to significantly different overall reduction behaviour. Firstly, in sharing graphs, applications are duplicated by co-sharing nodes, not sharing nodes, as illustrated below left. Secondly, unlike the atomic lambda-calculus, sharing graphs allow the mutual duplication of a sharing and a co-sharing, illustrated below right.



It is interesting to consider the nullary instances of the duplication rules in isolation. They are:

$$\begin{aligned} t[\leftarrow (u)v] &\rightsquigarrow_S t[\leftarrow u][\leftarrow v] \\ t[\leftarrow \lambda x.u] &\rightsquigarrow_S t[\twoheadleftarrow \lambda x.\langle\rangle[\leftarrow u]] \\ t[\twoheadleftarrow \lambda y.\langle\rangle[\leftarrow y]] &\rightsquigarrow_S t \end{aligned}$$

With the other sharing rewrite rules, these implement reduction paths of the following kind, where $\mathsf{FV}(u) = \{x_1, \ldots, x_n\}$.

$$t[\leftarrow u] \rightsquigarrow_S^\star t[\leftarrow x_1] \ldots [\leftarrow x_n]$$

It should be noted that the rule (7) allows to reduce unary sharings by actual substitutions, instead of reducing them atomically as the other sharings. This rule is in fact not necessary to compute the terms. It is only an optimisation, that one can choose to include or not include in the atomic lambda-calculus.

The central properties of sharing reductions are treated below.

Proposition 7. *Sharing reductions preserve typing.*

Proposition 8. *The reduction $\rightsquigarrow_S$ is strongly normalising.*

The proof of Proposition 8 is postponed until Section V, as it is best expressed using notation introduced in that section. Atomic lambda terms in normal form with respect to $\rightsquigarrow_S$ are said to be in *sharing normal form*. These correspond closely to the terms in the image of $(\!|-|\!)$, as expressed in the following proposition—here, the reason for the restriction to closed terms is to exclude atomic lambda terms with free variables inside a weakening (e.g., $x[\leftarrow y]$).

Proposition 9. *For $N \in \Lambda$ and closed, sharing-normal $t \in \Lambda_a$*

$$[\![(\!|N|\!)]\!] = N \qquad (\!|[\![t]\!]|\!) = t \qquad \exists M \in \Lambda.\ t = (\!|M|\!)\ .$$

Proposition 10. *If $t \rightsquigarrow_S u$, then $[\![t]\!] = [\![u]\!]$.*

Taken together, these propositions yield the following theorem.

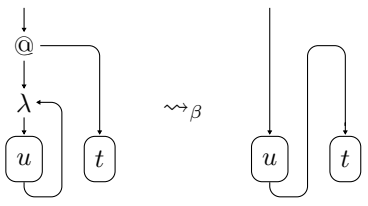
Theorem 11. *The reduction relation $\rightsquigarrow_S$ is strongly normalising and confluent.*

Proof: By Proposition 8 ($\rightsquigarrow_S$) is strongly normalising; by Proposition 9 normal forms are in 1–1 correspondence with denotations; and by Proposition 10 denotation is preserved under reduction. It follows that ($\rightsquigarrow_S$) is confluent. ∎

V. Beta-reduction

The definition of β-reduction in the atomic lambda-calculus, written ($\rightsquigarrow_\beta$) for clarity, is identical to that of the regular lambda-calculus; it is given and illustrated below.

$$(\lambda x.u)t \rightsquigarrow_\beta u\{t/x\}$$



$$\begin{aligned} [\![\{\cdot\}_{x_1,\ldots,x_n}]\!] &= (\{\cdot\}, \varnothing) \\ [\![\lambda x.u\{\cdot\}]\!] &= (\lambda x.M\{\cdot\}, \sigma) \\ [\![(u\{\cdot\})t]\!] &= ((M\{\cdot\})[\![t]\!], \sigma) \\ [\![(t)u\{\cdot\}]\!] &= (([\![t]\!])M\{\cdot\}, \sigma) \\ [\![t[y_1, \ldots, y_m \leftarrow u\{\cdot\}]]\!] &= ([\![t]\!]\{M\{\cdot\}/y_i\}_{1\le i\le m}, \sigma) \\ [\![u\{\cdot\}[y_1, \ldots, y_m \leftarrow t]]\!] &= (M\{\cdot\}\tau, \sigma\tau|_{\{x_1,\ldots,x_n\}}) \\ &\quad \text{where } \tau = \{[\![t]\!]/y_i\}_{1\le i\le n} \\ [\![u\{\cdot\}[y_1, \ldots, y_m \twoheadleftarrow \lambda z.t^m]]\!] &= (M\{\cdot\}\tau, \sigma\tau|_{\{x_1,\ldots,x_n\}}) \\ &\quad \text{where } \tau = \{[\![\pi_i(t^m)]\!]/y_i\}_{1\le i\le m} \\ [\![t[y_1, \ldots, y_m \twoheadleftarrow \lambda z.t^m\{\cdot\}]]\!] &= \\ &\quad ([\![t]\!]\{N_i/y_i\}_{1\le i\le m},\ \sigma_1 \ldots \sigma_m|_{\{x_1,\ldots,x_n\}}) \\ &\quad \text{where } (N_i, \sigma_i) = [\![\lambda_z.\pi_i(t^m\{\cdot\})]\!] \end{aligned}$$

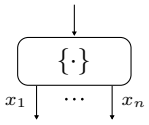
Fig. 4. The denotation $[\![-]\!]$ of atomic term contexts (Definition 15)

However, its effect is very different: in the atomic lambda-calculus β-reduction is a linear operation, since the bound variable x occurs exactly once in the body u. Any duplication of the term t in the atomic lambda-calculus proceeds via the sharing reductions ($\rightsquigarrow_S$).

Theorem 12. *β-reduction preserves typing.*

Proof: The proof is standard: by constructing matching typing derivations for either side of the reduction rule. ∎

In order to relate reduction steps in the atomic lambda-calculus to steps in the lambda-calculus, a notion of *one-hole context* is introduced for atomic lambda terms. The hole, denoted by $\{\cdot\}$, may be seen as a special, unique subterm, indicating a certain position within a term. For use with closures, a hole will be indexed with a sequence of variables $x_1, \ldots, x_n$—the free variables of the hole. Graphically, these may be pictured as a bundle of output wires, as follows.



Definition 13. *Atomic term contexts* $\Lambda_a^{\{\cdot\}}$ are constructed by the grammar of atomic lambda terms extended with the *hole* construct $\{\cdot\}_{x_1,\ldots,x_n}$, in which the variables $x_1, \ldots, x_n$ are considered free, and which must occur exactly once in an atomic term context.

$$t\{\cdot\}_{x_1,\ldots,x_n} \quad := \quad \ldots \quad | \quad \{\cdot\}_{x_1,\ldots,x_n}$$

The atomic lambda term $t\{u\}$ is obtained from the term context $t\{\cdot\}_{x_1,\ldots,x_n}$ by substituting u for the hole, under the condition that $\mathsf{FV}(u) = \{x_1, \ldots, x_n\}$.

The subscript indicating the free variables of a hole will be omitted where possible. The purpose of the hole is to identify a specific subterm u in an atomic lambda term $t\{u\}$, and in particular to obtain independent translations of $t\{\cdot\}$ and u into the lambda-calculus. There are two important details to this translation. Firstly, since the hole may occur in a subterm that is shared, a corresponding context in the lambda-calculus may need any number of holes. Secondly, a free variable of a hole in an atomic context may be bound by a closure $[\gamma]$ (but also by an abstraction λx). Since in translation the action of $[\gamma]$ is naturally captured by a substitution, the translation of an atomic term context $t\{\cdot\}_{x_1,\ldots,x_n}$ will be a pair $(N\{\cdot\},\sigma)$ of a regular lambda-term context (defined below), and a substitution map σ that replaces the variables $x_1,\ldots,x_n$ with regular lambda terms. Then if the context $t\{\cdot\}_{x_1,\ldots,x_n}$ translates to $(N\{\cdot\},\sigma)$ and the atomic lambda term u with free variables $\{x_1,\ldots,x_n\}$ translates to M, the translation of $t\{u\}$ will be $N\{M\sigma\}$, the context $N\{\cdot\}$ where the holes are filled by the lambda term obtained by applying σ to M.

Definition 14. *Lambda term contexts* $\Lambda^{\{\cdot\}}$ extend the lambda-calculus Λ with the *hole* constructor $\{\cdot\}$.

$$N\{\cdot\} \quad := \quad \ldots \quad | \quad \{\cdot\}$$

For lambda term contexts there will be no restriction on the number of occurrences of the hole, nor will the hole contain free variables.

Definition 15. The *denotation* $[\![u\{\cdot\}_{x_1,\ldots,x_n}]\!]$ of an atomic term context is a pair $(M\{\cdot\},\sigma)$, where $M\{\cdot\} \in \Lambda^{\{\cdot\}}$ and $\sigma\colon \{x_1,\ldots,x_n\} \to \Lambda$ is a substitution map. The function $[\![-]\!]$ is inductively defined in Figure 4, for $[\![u\{\cdot\}]\!] = (M\{\cdot\},\sigma)$.

Lemma 16. *If* $[\![u\{\cdot\}_{x_1,\ldots,x_n}]\!] = (M\{\cdot\},\sigma)$ *and* $\mathsf{FV}(t) = \{x_1,\ldots,x_n\}$ *then* $[\![u\{t\}]\!] = M\{[\![t]\!]\sigma\}$.

Proof: Simple case analysis. ∎

A first use of contexts is the proof of Proposition 8, that was postponed in Section IV.

Proposition 8 (restatement). *The reduction* $\rightsquigarrow_S$ *is strongly normalising.*

Proof: Each closure $[\gamma]$ in a term u is assigned two measures: the *depth*, which is the distance from the subterm to the root of the term when the term is viewed as a directed acyclic graph modulo Equation (1); and the *weight*, which for a distributor is ½, and for a sharing is defined as follows. Given a sharing $[\gamma] = [x_1,\ldots,x_n \leftarrow t]$ within a term u, let $u = s\{t\}$ and let $[\![u\{\cdot\}]\!] = (N\{\cdot\},\sigma)$. The weight of $[\gamma]$ is then the size of $[\![t]\!]\sigma$, measured in the number of abstractions and applications. Note that both measures are invariant under the congruence $(\sim)$.

The measure for a term t is the pair (weights,depths) consisting of the multiset of the weights of all closures in t, and the multiset of the depths of all closures. A simple case analysis shows that for every sharing reduction rule either

- the depth of one closure $[\delta]$ is reduced, while all weights and all other depths in the term remain unchanged (rewrite rules (2)–(6)); or
- one sharing is removed, while the weights of the others remain unchanged (rewrite rules (7)–(8)); or
- one or more closures $[\delta]$ are replaced with some of strictly lower weight: in rewrite rule (9), one sharing of weight n is replaced by two of weight $n-1$; in rewrite rule (10) a sharing of weight $(n \geq 1)$ is replaced by a distributor (of weight ½) and a sharing of weight $n-1$; and in rewrite rule (11) a distributor of weight ½ (plus a sharing of weight zero) are replaced by several sharings of weight zero.

It follows that the measure of a term is strictly decreasing under $\rightsquigarrow_S$, proving the statement. ∎

One β-reduction step in the atomic lambda-calculus corresponds to zero or more β-steps in the regular lambda-calculus.

Lemma 17. *If* $t \rightsquigarrow_\beta u$ *then* $[\![t]\!] \rightsquigarrow_\beta^\star [\![u]\!]$.

Proof: An atomic lambda term with a β-redex is of the form $v\{(\lambda x.u)t\}$. Let $[\![v\{\cdot\}]\!] = (M\{\cdot\},\sigma)$; then, using Lemma 16 (and with n the number of holes in $M\{\cdot\}$),

$$\begin{aligned} [\![v\{(\lambda x.u)t\}]\!] &= M\{((\lambda x.[\![u]\!])[\![t]\!])\sigma\} \\ &\rightsquigarrow_\beta^n M\{([\![u]\!]\{[\![t]\!]/x\})\sigma\} \\ &= [\![v\{u\{t/x\}\}]\!] \,. \end{aligned}$$

∎

Moreover, any β-step in the denotation of an atomic lambda term may be simulated in the atomic lambda-calculus by a combination of sharing reductions and a β-reduction.

Theorem 18. *If* $N \rightsquigarrow_\beta M$ *then* $(\!|N|\!) \rightsquigarrow_\beta \rightsquigarrow_S^\star (\!|M|\!)$.

Proof: A lambda-term with a β-redex is of the form $M\{(\lambda x.N)P\}$, where $M\{\cdot\}$ has exactly one hole. Let $(\!|M\{\cdot\}|\!) = v\{\cdot\}$; then, by the definition of $(\!|-|\!)$ and by Proposition 9,

$$\begin{aligned} (\!|M\{(\lambda x.N)P]\}|\!) &= v\{(\lambda x.(\!|N|\!)[x_1,\ldots,x_n \leftarrow x])(\!|P|\!)\} \\ &\rightsquigarrow_\beta v\{(\!|N|\!)[x_1,\ldots,x_n \leftarrow (\!|P|\!)]\} \\ &\rightsquigarrow_S^\star v\{(\!|N\{P/x\}|\!)\} \\ &= (\!|M\{N\{P/x\}\}|\!) \end{aligned}$$

where $n \neq 1$ is the number of occurrences of x in N. The case for $n = 1$ is trivial. ∎

VI. Preservation of strong normalisation

The main result for the atomic lambda-calculus presented here will be the preservation of strong normalisation with respect to the lambda-calculus (PSN). The challenge, as with many implementations of sharing, is that reduction in the atomic lambda-calculus may take place inside a weakening. A beta-step within a weakening is simulated by zero beta-steps in a corresponding lambda term, where weakenings are not retained, thus frustrating the direct construction of an infinite lambda reduction from an infinite atomic reduction.

To separate the problem of weakened reductions from the details of the sharing mechanism, a lambda-calculus with explicit weakening is introduced, the *w-calculus*, as an

intermediate between the atomic lambda-calculus and the lambda-calculus.[1] A beta-step in the atomic lambda-calculus will correspond to at least one step in the w-calculus; then PSN for the latter will imply the same for the former.

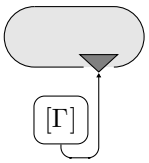
Definition 19. The *w-terms* of the *w-calculus* are

$$T, U, V \;:=\; x \;\mid\; \lambda x.T^{\,*} \;\mid\; (U)V \;\mid\; T[\leftarrow U] \;\mid\; \bullet$$

where $(^*)$ $x \in \mathsf{FV}(T)$.

A variable or subterm inside a weakening $[\leftarrow U]$ is *weakened*. The bullet ($\bullet$) will interpret the variable bound in a nullary distributor, such as y in $[\Gamma]$ in the example illustrated below.

$$[\twoheadleftarrow \lambda y.\langle\rangle[\Gamma]]$$



Definition 20. The function $[\![-]\!]_w$ interprets atomic lambda terms as w-terms.

$$[\![x]\!]_w = x \qquad [\![\lambda x.t]\!]_w = \lambda x.[\![t]\!]_w \qquad [\![u(v)]\!]_w = ([\![u]\!]_w)[\![v]\!]_w$$

$$[\![t[x_1,\ldots,x_n \leftarrow u]]\!]_w = \begin{cases} [\![t]\!]_w[\leftarrow [\![u]\!]_w] & \text{if } n = 0 \\ [\![t]\!]_w\{[\![u]\!]_w/x_i\}_{1\le i\le n} & \text{otherwise} \end{cases}$$

$$[\![t[x_1,\ldots,x_n \twoheadleftarrow \lambda y.t^n]]\!]_w =$$

$$\begin{cases} [\![t[\Gamma]]\!]_w\{\bullet/y\} & \text{if } n = 0,\ t^n = \langle\rangle[\Gamma] \\ [\![t]\!]_w\{\lambda y.[\![\pi_i(t^n)]\!]_w/x_i\}_{1\le i\le n} & \text{otherwise} \end{cases}$$

As an interpretation of the atomic lambda-calculus, the w-calculus is *denotational* with respect to sharing, but *operational* with respect to weakening: sharing is modelled by duplication (via substitution), while deletion is via an elaborate set of rewrite rules, designed to mimick the behaviour of the atomic lambda-calculus. Beta-reduction in the w-calculus is as normal, and terms are taken up to permutation of weakenings:

$$(\lambda x.T)U \rightsquigarrow_\beta T\{U/x\} \qquad t[\leftarrow U][\leftarrow V] \sim_w t[\leftarrow V][\leftarrow U]\ .$$

Deletion of weakenings ($\rightsquigarrow_w$) proceeds as follows.

$$\begin{array}{rcll} \lambda x.T[\leftarrow U] & \rightsquigarrow_w & (\lambda x.T)[\leftarrow U] & \text{if } x \notin \mathsf{FV}(U) \\ (U[\leftarrow T])V & \rightsquigarrow_w & ((U)V)[\leftarrow T] & \\ (U)V[\leftarrow T] & \rightsquigarrow_w & ((U)V)[\leftarrow T] & \\ T[\leftarrow U[\leftarrow V]] & \rightsquigarrow_w & T[\leftarrow U][\leftarrow V] & \\ T[\leftarrow \lambda x.U] & \rightsquigarrow_w & T[\leftarrow U\{\bullet/x\}] & \\ T[\leftarrow (U)V] & \rightsquigarrow_w & T[\leftarrow U][\leftarrow V] & \\ T[\leftarrow \bullet] & \rightsquigarrow_w & T & \\ T[\leftarrow U] & \rightsquigarrow_w & T & \text{if } U \text{ is a subterm of } T \end{array}$$

[1] A similar technique is used in [3]; unfortunately, their weakening calculus is not general enough for the present purpose.

The above rules correspond directly to rules in $(\rightsquigarrow_S)$, as is easily observed; the final rule merits an explanation. Firstly, it corresponds to rewrite rule (8) for the case where a weakening is incorporated into a sharing. Secondly, the side-condition ensures that the occurrence requirement $(^*)$ of Definition 19 is maintained; it could be relaxed, requiring only that $\mathsf{FV}(U) \subseteq \mathsf{FV}(T)$. Thirdly, in the translation $[\![-]\!]_w$ from the atomic lambda-calculus to the w-calculus any weakening within the scope of a sharing, e.g. $[\leftarrow v]$ in $t[x, y \leftarrow u[\leftarrow v]]$, is duplicated. In contrast, rewriting within the atomic lambda-calculus never causes duplication of weakenings. The rule allows deletion of duplicated weakenings, maintaining commutativity of translation and rewriting. The latter is expressed by the following lemma.

Lemma 21. *If $t \rightsquigarrow_\beta u$ then $[\![t]\!]_w \rightsquigarrow_\beta^+ [\![u]\!]_w$. If $t \rightsquigarrow_S u$ then $[\![t]\!]_w \rightsquigarrow_w^\star [\![u]\!]_w$.*

Proof: Omitted. ∎

The natural translation from a w-term to a lambda-term, which discards all weakenings $[\leftarrow U]$, is denoted $\lfloor - \rfloor$. The interpretation of a lambda-term as a w-term by the function $(\!|-|\!)_w$ is defined below.

$$(\!|x|\!)_w = x \qquad (\!|(N)M|\!)_w = ((\!|N|\!)_w)(\!|M|\!)_w$$

$$(\!|\lambda x.N|\!)_w = \begin{cases} \lambda x.(\!|N|\!)_w & \text{if } x \in \mathsf{FV}(N) \\ \lambda x.(\!|N|\!)_w[\leftarrow x] & \text{otherwise} \end{cases}$$

The following are easily observed.

Proposition 22. *For $N \in \Lambda$ and $t \in \Lambda_a$,*

$$\lfloor [\![t]\!]_w \rfloor = [\![t]\!] \qquad (\!|N|\!)_w = [\![(\!|N|\!)]\!]_w \qquad \lfloor (\!|N|\!)_w \rfloor = N\ .$$

PSN for the w-calculus will be shown using the notion of a *perpetual strategy* [6]. The main property of such a reduction strategy is that, from a given term, if it reaches a normal form, then this term is strongly normalising. By translating a non-normalising perpetual reduction on w-terms to an infinite reduction on lambda terms, PSN then follows.

Definition 23. The *perpetual strategy* on a w-term U is the sequence $U = U_1 \rightsquigarrow U_2 \rightsquigarrow \ldots$ defined by $U_{i+1} = \omega(U_i)$:

$$\begin{array}{lcll} \omega(x) & = & x & \\ \omega(\lambda x.T) & = & \lambda x.\omega(T) & \\ \omega((T)U) & = & (T)\omega(U) & \text{if } T \text{ normal} \\ \omega((\lambda x.T)U) & = & \begin{cases} (\lambda x.T)\omega(U) & \text{if } x \text{ weakened and } U \text{ not normal} \\ T\{U/x\} & \text{otherwise} \end{cases} & \\ \omega(((T)U)V) & = & (\omega((T)U))V & \text{if } (T)U \text{ not normal} \\ \omega((T[\leftarrow V])U) & = & (T)U[\leftarrow V] & \\ \omega(T[\leftarrow U]) & = & \begin{cases} \omega(T)[\leftarrow U] & \text{if } U \text{ normal} \\ (T)[\leftarrow \omega(U)] & \text{otherwise} \end{cases} & \end{array}$$

where *normal* is with respect to $\rightsquigarrow_\beta$ and the rewrite rule

$$(T[\leftarrow V])U \rightsquigarrow_w ((T)U)[\leftarrow V]\ .$$

This strategy can be characterised as *leftmost-outermost*. In the final case, for $T[\leftarrow U]$, the choice to reduce first in U before reducing T is inconsequential to the argument: as will be shown, for the relevant cases U will already be normal. A crucial property of this strategy is that when it applies a reduction inside a subterm T of a term U, then further reduction inside T will be independent of reduction outside it; in particular, T will not be duplicated, nor have its free variables instantiated. This is expressed in the following lemma.

Contexts for w-terms $U\{\cdot\}$ are used in the standard way except, as with atomic term contexts, the hole may carry free variables $\{\cdot\}_{x_1,\ldots,x_n}$, which will be left implicit.

Lemma 24. *If $\omega(U\{T\}) = U\{\omega(T)\}$ and $U\{T\} \rightsquigarrow V$, then $V = U'\{T'\}$ with $U\{\} \rightsquigarrow U'\{\}$ and $T \rightsquigarrow T'$.*

Proof: By induction on the definition of ω. ∎

Lemma 25. *If U has an infinite reduction then so does $\omega(U)$.*

Proof: Let $U = U_1, U_2, \ldots$ be an infinite reduction path, and let $U = U'\{S\}$ with S the smallest subterm of U such that $\omega(U'\{S\}) = U'\{\omega(S)\}$. Assume that S is of the form $(T[\leftarrow W])V$; the case where $S = (\lambda x.V)T$ are analogous, while S cannot be a variable as then U would be β-normal.

If the redex S is never reduced in the path $U_1, U_2, \ldots$ then one of $U'\{\}$, T, W or V must have an infinite path by Lemma 24, and so must $U'\{(T)V[\leftarrow W]\}$.

If the redex S is reduced in the k-th step, then by Lemma 24 the first $k+1$ terms of the infinite reduction path are

$$U_1, U_2, \ldots, U_k\{(T_k[\leftarrow W_k])V_k\},\ U_k\{(T_k)V_k[\leftarrow W_k]\}$$

where $U'\{\} \rightsquigarrow^\star U_k\{\}$, $T \rightsquigarrow^\star T_k$, $V \rightsquigarrow^\star V_k$ and $W \rightsquigarrow^\star W_k$. Then $U'\{(T)V[\leftarrow W]\} \rightsquigarrow^\star U_k\{(T_k)V_k[\leftarrow W_k]\}$. ∎

Lemma 26. *If a lambda-term N is SN, so is the w-term $(\!|N|\!)_w$.*

Proof: By contradiction. Let $U = (\!|N|\!)_w$ have an infinite reduction path; then its perpetual strategy $U = U_1, U_2, \ldots$ has an infinite number of beta steps. Moreover, in each U_i every weakening is β-normal: in U weakenings are of the form $[\leftarrow x]$ with x a variable, and if x is instantiated with T in $U_{i-1} \rightsquigarrow U_i$ then 1) by the definition of ω, T is β-normal, and 2) by Lemma 24 no free variable in T is instantiated in any U_j ($j > i$). Since all β-steps in $U_1, U_2, \ldots$ occur outside weakenings, the reduction $\lfloor U_1 \rfloor, \lfloor U_2 \rfloor, \ldots$ on $N = \lfloor U \rfloor$ (Proposition 22) contains infinitely many β-steps. ∎

Theorem 27. *The atomic lambda-calculus satisfies PSN.*

Proof: Let $N \in \Lambda$ be SN. By Lemma 26 $(\!|N|\!)_w$ is SN; by Proposition 22 $(\!|N|\!)_w = [\![(\!|N|\!)]\!]_w$; it follows by Lemma 21 that $(\!|N|\!)$ is SN. ∎

VII. Fully lazy sharing

The atomic lambda-calculus provides fine-grained control over which parts of a shared subterm to duplicate during normalisation. A natural strategy is to only reduce closures when necessary in order to perform beta reductions, and to prefer pushing closures towards the outside of the term rather than duplicating them. This strategy, which is detailed below, implements *fully lazy sharing*. The notion of *fully lazy sharing*, for an implementation of the regular lambda-calculus, requires that in the duplication of a lambda-term the *maximal free subexpressions* remain shared [17], [11] (see also [14], [5]). The latter, for a term $\lambda x.N$, are the largest subterms of N containing neither x nor any free variable that is bound in N.

Definition 28. A *(maximal) V-free subexpression* t of u, for a set of variables V, is a (maximal) subterm t of u such that $\mathsf{FV}(t) \subset \mathsf{FV}(u) - V$. A *free subexpression* is a $\varnothing$-free subexpression, and a *maximal free subexpression* of a term $\lambda x.N$ is a maximal $\{x\}$-free subexpression of N.

The *skeleton* (or *iterated scope* or *extended scope*) of a regular lambda term $\lambda x.N$ is $\lambda x.N'$, where N' is the result of replacing the maximal $\{x\}$-free subexpressions $M_1, \ldots, M_n$ of N by fresh variables $y_1, \ldots, y_n$. Then

$$\lambda x.N = (\lambda x.N')\{M_1/y_1\} \ldots \{M_n/y_n\}\ .$$

For the atomic lambda-calculus, the notion of skeleton will only be needed for basic terms; it is defined similarly, except that a skeleton is sharing-normal, and excludes the top-level sharings of free variables. A formal inductive definition is given below.

A *variant* of a term t is any term obtained from t by changing the name of certain (bound or free) variables. A variant is *fresh* if its variables are fresh, and t^i will denote a fresh variant of t obtained by replacing each variable x in t by a fresh one x^i.

Definition 29. The *V-skeleton* $\mathsf{skel}_V(t)$ of a basic term t, with V a set of variables that do not occur bound in t, is defined inductively as follows (where y is a fresh variable).

$\mathsf{skel}_V(x) = x$

$\mathsf{skel}_V(\lambda x.u) = \lambda x.(\mathsf{skel}_{V\cup\{x\}}(u)[x^1, ..., x^m \leftarrow x])$ *

$\mathsf{skel}_V((u)v) =$

$$\begin{cases} y & \text{if } \mathsf{FV}((u)v) \cap V = \varnothing; \text{otherwise} \\ (\mathsf{skel}_V(u))y & \text{if } \mathsf{FV}(v) \cap V = \varnothing \\ (y)\mathsf{skel}_V(v) & \text{if } \mathsf{FV}(u) \cap V = \varnothing \\ (\mathsf{skel}_V(u))\mathsf{skel}_V(v) & \text{otherwise} \end{cases}$$

$\mathsf{skel}_V(u[\vec{x} \leftarrow v]) =$

$$\begin{cases} y & \text{if } \mathsf{FV}(u[\vec{x} \leftarrow v]) \cap V = \emptyset \\ \mathsf{skel}_V(u) & \text{if } \mathsf{FV}(v) \cap V = \emptyset \text{ and } \mathsf{FV}(u) \cap V \neq \emptyset \\ \mathsf{skel}_{V\cup\{\vec{x}\}}(u)\sigma & \text{otherwise } ^{**} \end{cases}$$

where (*) the variants $x^1, \ldots, x^m$ originate in the substitutions σ in the final case above, and (**) $\sigma = \{(\mathsf{skel}_V(v))^i/x_i\}_{1\leq i\leq n}$. The *skeleton* $\mathsf{skel}(t)$ of a term t is defined as $\mathsf{skel}_\emptyset(t)$.

Lemma 30. *For a set of variables V and a term t of the basic calculus, a term $u[x_1, \ldots, x_n \leftarrow t]$ can be reduced to*

$$u\{\mathsf{skel}_V(t)^i/x_i\}_{1\leq i\leq n}[\Gamma][\Delta]$$

where $[\Gamma]$ is made up of, for each $y \in \mathsf{FV}(t) \cap V$, a sharing of the form $[y_1, \ldots, y_m \leftarrow y]$.

Proof: The proof is by induction on t. For t a variable, the statement holds by the empty reduction. The remaining cases are routine, following the inductive definition of $\mathsf{skel}_V(t)$, with the most interesting ones being the case for abstraction and case for nested sharing:

Let $t = \lambda y.v$ and $s = \mathsf{skel}_{V\cup\{x\}}(v)$.

$$
\begin{aligned}
&u[x_1,\dots,x_n \leftarrow \lambda y.v] \\
&\rightsquigarrow_S u[x_1,\dots,x_n \nleftarrow \lambda y.\langle y_1,\dots,y_n\rangle[y_1,\dots,y_n \leftarrow v]] && (10)\\
&\rightsquigarrow^\star_S u[x_1,\dots,x_n \nleftarrow \lambda y.\langle s^1,\dots,s^n\rangle[\vec{z} \leftarrow y][\Gamma][\Delta]] && (\text{ih})\\
&\rightsquigarrow^\star_S u[x_1,\dots,x_n \nleftarrow \lambda y.\langle s^1,\dots,s^n\rangle[\vec{z} \leftarrow y]][\Gamma][\Delta] && (6)\\
&\rightsquigarrow_S u\{\lambda y^i.s^i[\vec{z}_i \leftarrow y^i]/x_i\}_{1\le i\le n}[\Gamma][\Delta] && (11)\\
&\quad = \quad u\{(\mathsf{skel}_V(\lambda y.v))^i/x_i\}_{1\le i\le n}[\Gamma][\Delta]
\end{aligned}
$$

where the $\vec{z}_i$ are as in rewrite rule (11).

Let $t = w[y_1,\dots,y_m \leftarrow v]$, and consider the case where $\mathsf{FV}(w)\cap V \neq \emptyset$ and $\mathsf{FV}(v)\cap V \neq \emptyset$. Let $\vec{y} = y_1,\dots,y_m$ and let $\sigma = \{(\mathsf{skel}_V(v))^j/y_j\}_{1\le j\le m}$; then

$$
\begin{aligned}
&u[x_1,\dots,x_n \leftarrow w[y_1,\dots,y_m \leftarrow v]] \\
&\rightsquigarrow_S u[x_1,\dots,x_n \leftarrow w][y_1,\dots,y_m \leftarrow v] && (5)\\
&\rightsquigarrow^\star_S u\{(\mathsf{skel}_{V\cup\{\vec{y}\}}(w))^i/x_i\}_{1\le i\le n}[\vec{y}_1 \leftarrow y_1]\dots[\vec{y}_m \leftarrow y_m] \\
&\qquad\qquad [\Gamma_1][\Delta_1][y_1,\dots,y_m \leftarrow v] && (\text{ih})\\
&\rightsquigarrow^\star_S u\{(\mathsf{skel}_{V\cup\{\vec{y}\}}(w))^i/x_i\}_{1\le i\le n}[\vec{y}_1 \leftarrow (\mathsf{skel}_V(v))^1]\dots \\
&\qquad\qquad \dots[\vec{y}_m \leftarrow (\mathsf{skel}_V(v))^m][\Gamma_1][\Delta_1][\Gamma_2][\Delta_2] && (\text{ih})\\
&\rightsquigarrow^\star_S u\{(\mathsf{skel}_{V\cup\{\vec{y}\}}(w)\sigma)^i/x_i\}_{1\le i\le n}[\Gamma_3][\Delta_3][\Gamma_1][\Delta_1][\Gamma_2][\Delta_2] \\
&\qquad\qquad && (\text{ih})\\
&\quad = \quad u\{(\mathsf{skel}_V(w[y_1,\dots,y_m \leftarrow v]))^i/x_i\}_{1\le i\le n}[\Gamma][\Delta]
\end{aligned}
$$

where $[\Gamma] = [\Gamma_1][\Gamma_2][\Gamma_3]$ and $[\Delta] = [\Delta_1][\Delta_2][\Delta_3]$. The last application of the inductive hypothesis above follows from the fact that $\mathsf{skel}_V(-)$ is idempotent. ∎

The strategy implemented in the proof of the above lemma reduces basic terms to basic terms, eliminating every distributor it introduces, as can be observed from the case for abstraction.

Proposition 31. *The atomic lambda-calculus implements fully lazy sharing: a basic term $u[x_1,\dots,x_n \leftarrow \lambda y.t]$ reduces to a basic term of the form*

$$u\{(\mathsf{skel}(\lambda y.t))^i/x_i\}_{1\le i\le n}[\Gamma]$$

Proof: By Lemma 30. ∎

VIII. Conclusions and Further Work

The atomic lambda-calculus presented here is a typed term calculus with explicit sharing. It implements reduction steps that are atomic, similar to duplication in optimal reduction-graphs, to obtain a form of fully lazy sharing. Originating in a Curry–Howard style interpretation of the open deduction formalism, the calculus exposes a connection between proof reduction in deep inference and the sharing mechanisms of optimal reduction graphs.

The motivation behind the atomic lambda-calculus has been to capture the basic reduction behaviour of the medial-rule of open deduction within a simple term construct, the distributor. A natural direction for future work is to construct term calculi corresponding to more fine-grained proof systems in open deduction, in order to explore further possible proof reductions.

Acknowledgements: The authors would like to thank the anonymous referees for their helpful and insightful comments. This work was supported by the ANR–FWF project Structural.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] Beniamino Accattoli and Delia Kesner. The permutative lambda-calculus. In *LPAR*, pages 23–36, 2012.

[3] Beniamino Accattoli and Delia Kesner. Preservation of strong normalisation modulo permutations for the structural lambda-calculus. *LMCS*, 8(1), 2012.

[4] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7:265–301, 1997.

[5] Thibaut Balabonski. A unified approach to fully lazy sharing. In *POPL*, pages 469–480, 2012.

[6] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[7] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus revisited. In *Reflections on Type Theory, Lambda Calculus, and the Mind*, pages 41–50, 2007.

[8] Kai Brünnler and Alwen Tiu. A local system for classical logic. In *LPAR*, volume 2250 of *LNCS*, pages 347–361, 2001.

[9] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In *RTA*, pages 135–150, 2010.

[10] Tom Gundersen, Willem Heijltjes, and Michel Parigot. Un lambda-calcul atomique. In *Journés Francophones des Langages Applicatifs*, 2013.

[11] R.J.M. Hughes. Super-combinators: a new implementation method for applicative languages. In *ACM Symposium on Lisp and Functional Programming*, pages 1–10, 1982.

[12] Delia Kesner and Stéphane Lengrand. Resource operators for lambda-calculus. *Information and Computation*, 205(4):419–473, 2007.

[13] John Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.

[14] Simon L. Peyton-Jones. *The implementation of functional programming languages*. Prentice Hall, 1987.

[15] Laurent Regnier. Une équivalence sur les lambda-termes. *Theor. Comput. Sci.*, 126(2):281–292, 1994.

[16] Alwen Tiu. A local system for intuitionistic logic. In *LPAR*, 2006.

[17] Christopher Peter Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.