An Atomic Lambda Calculus

T. Gundersen[1], W. Heijltjes[2], & M. Parigot[3]

1: CNRS & Université Paris Diderot
teg@jklm.no
2: INRIA & Laboratoire d'Informatique de l'École Polytechnique
willem.heijltjes@gmail.com
3: CNRS & Université Paris Diderot
parigot@pps.univ-paris-diderot.fr

Abstract

An explicit sharing lambda calculus is presented in which duplication of subterms proceeds on individual constructors. The calculus is given a denotation in terms of the lambda calculus, and it is shown to simulate β-reduction, preserving strong normalisation. A type system for the atomic lambda calculus is given, and preservation of typing is shown.

1. Introduction

Computation in the λ-calculus is notoriously unpredictable in its use of computational resources. Even for simply typed λ-terms, one reduction path may terminate in linear time, while another path induces hyperexponential growth of the very same term. Central to restraining such growth is to exercise strategic control over duplication and deletion: to perform computation within a term before it is duplicated, and to avoid unnecessary computation within subterms that will be deleted. One approach to this end is the use of *reduction strategies*, which attempt to choose favourably among the available reduction paths. Another is to adapt the syntax, as with *explicit substitution calculi* [ACCL91] and *optimal reduction graphs* [Lam90].

In this paper a specific aspect of control over duplication is addressed: *atomicity* of duplication. By this is meant: the ability to duplicate individual constructors; in particular, given a term $\lambda x.t$, to duplicate the abstraction λx independently of the body t. Implementing such reductions, this paper presents the *atomic lambda calculus*, inspired by proof reduction in *deep inference* [Gug07]—a style of proof representation where inference rules apply deep in context, reminiscent of term rewriting. The crucial duplication step is performed as follows, here shown in the *open deduction* formalism [GGP10].

$$\dfrac{\begin{array}{c} A^x \\ A^x \;\to\; \|_t \\ B \end{array}}{c\,\dfrac{}{[A\to B]\wedge[A\to B]}} \quad\rightsquigarrow\quad \dfrac{\begin{array}{c} A^x \\ A^x \;\to\; \|_t \\ c\,\dfrac{B}{B\wedge B} \end{array}}{d\,\dfrac{}{[A\to B]\wedge[A\to B]}}$$

Particular of open deduction is that connectives such as $\wedge$ and, here, $\to$ may combine derivations as well as formulae. The derivation above left is then composed as follows. The double vertical line from A to B represents a derivation for a term t with a free variable of type A, which is combined with an implication to form a derivation for $\lambda x.t : A \to B$. To this derivation a *contraction* (c) is applied, allowing its conclusion $A \to B$ to be used twice in a further derivation. On the right, rather than duplicating the entire subderivation for $\lambda x.t$, the contraction is pushed within the subderivation that is the consequent of the implication, by using a special *distribution* inference (d). The contraction (c) may then continue to duplicate t stepwise until it reaches the top (A), after which the distribution (d) may be replaced by the conjunction of the two derivations for $\lambda x.t$.

The main feature of the atomic lambda calculus will be a term constructor called the *distributor*, corresponding to the distribution rule above. This paper will define the calculus

and its reduction steps, and establish their basic properties: simple typing, translation to and from the lambda calculus, and preservation of strong normalisation. Use of deep inference in this paper will be limited, since it is possible to give typing derivations for the atomic lambda calculus in the—usually more familiar—sequent calculus. To aid the intuition, constructors and reduction steps will be illustrated graphically. More involved matters, such as reduction strategies and their efficiency, and in particular the relation to other studies on explicit sharing in lambda-calculi such as [Wad71], [BLM07], and [Bal12], are left for further work. As far as the authors are aware, no other term calculus currently enjoys the atomicity property; however, it is a main feature of optimal reduction graphs.

2. The term calculus

For clarity the atomic lambda calculus and the standard lambda calculus will make use of distinct alphabets. The standard lambda calculus (Λ) is defined as follows.

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

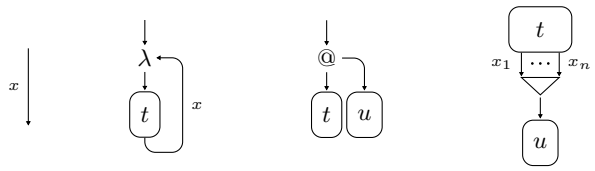
The basis of the atomic lambda calculus consists of linear lambda terms with explicit contractions. For the remainder, where applicable all variables will be assumed fresh.

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

$$s, t, u, v, w \quad := \quad x \quad | \quad \lambda x.t \quad | \quad (t)u \quad | \quad t[x_1, \ldots, x_n \leftarrow u]$$

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 $t[x_1, \ldots, x_n \leftarrow u]$, each x_i must occur in t and becomes bound.

The constructor $t[x_1, \ldots, x_n \leftarrow u]$ will be called a *sharing*; a nullary sharing $t[\leftarrow u]$ will be called a *weakening*. The four constructors (*variable*, *abstraction*, *application*, and *sharing*) are displayed graphically as follows. In the remainder, term boxes such as for t and u will be omitted.



The function $(\!|-|\!) : \Lambda \to \Lambda_A^-$ maps lambda-terms onto basic terms. The translation proceeds by inductively replacing a term $\lambda x.N$ by $\lambda x.N'[x_1, \ldots, x_n \leftarrow x]$, where N' is obtained from N by replacing the n occurrences of x by $x_1, \ldots, x_n$. In order to formally define this translation, it will be parametrised by an operation σ which assigns to each variable x of the term, a sequence of new variables $x_1, \ldots, x_n$ whose length n is the number of occurrences of x in the term, and such that the variables appearing in the image of σ are all distinct. For convenience, let $|N|_x$ denote the number of occurrences of x in N. The term $(\!|N|\!)_\sigma$ is defined inductively by

$$(\!|x|\!)_\sigma = \sigma(x)$$

$$(\!|(M)N|\!)_\sigma = ((\!|M|\!)_{\sigma'})(\!|N|\!)_{\sigma''}$$

$$(\!|\lambda x.N|\!)_\sigma = \begin{cases} \lambda x'.(\!|N|\!)_\sigma & \text{if } |N|_x = 1 \text{ and } \sigma(x) = x' \\ \lambda x.(\!|N|\!)_\sigma[x_1, \ldots, x_n \leftarrow x] & \text{if } |N|_x = n \neq 1 \text{ and } \sigma(x) = (x_1, \ldots, x_n) \end{cases},$$

where σ' and σ'' are built as follows: for each variable x in $(M)N$ with $|M|_x = m$ and $|N|_x = n$, if $\sigma(x) = (x_1, \ldots, x_{m+n})$, then $\sigma'(x) = (x_1, \ldots, x_m)$ and $\sigma''(x) = (x_{m+1}, \ldots, x_{m+n})$. 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 will facilitate the evaluation of a unary sharing $t[x \leftarrow u]$ by a linear substitution of u for x.

Terms in the image of $(\!|-|\!)$ are those of the basic calculus generated by the grammar below.

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

Definition 2. The *atomic lambda calculus* Λ_A extends the basic calculus with the *distributor* constructor. The following mutually recursive grammars simultaneously define the *terms* t of the calculus, and the *terms of multiplicity* n, written t^n, for every $n > 0$.

$$s, t, u, v, w \quad := \quad \ldots \quad | \quad s[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n]$$

$$t^n \quad := \quad \langle t_1, \ldots, t_n \rangle \quad | \quad t^n[x_1, \ldots, x_m \leftarrow u] \quad | \quad t^n[x_1, \ldots, x_m \twoheadleftarrow \lambda y.t^m]$$

where (i) each variable may occur at most once, (ii) in $s[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n]$, each variable x_i must occur in s and becomes bound, and (iii) in $t^n[x_1, \ldots, x_m \leftarrow u]$ and $t^n[x_1, \ldots, x_m \twoheadleftarrow \lambda y.t^m]$ each variable x_i must occur in t^n and becomes bound.

The sharing and distributor constructors together will be referred to as *closures*. Terms of the atomic lambda calculus will be called *atomic lambda terms* A term of multiplicity n is an n-tuple of atomic lambda terms to which closures apply as though it were an atomic lambda term. In other words, a term of multiplicity n is of the form $\langle t_1, \ldots, t_n \rangle[\Gamma]$ where $[\Gamma]$ is a sequence of sharings and distributors. Note that while closures are freely applied to both terms and terms of a given multiplicity, application and abstraction only apply to regular, singular terms. In the remainder, the distinction between terms and terms of multiplicity n will only be made where it is relevant, and both will generally be denoted by t $(u, v, \ldots)$. The distributor is illustrated below, as a term, a derivation, and a graph.

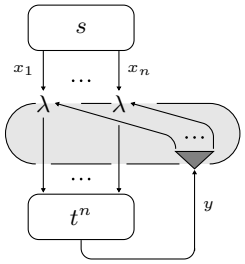
$s[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n]$

$$A^y \quad \rightarrow \quad \begin{array}{c} A^y \\ \| t^n \\ B_1 \wedge \ldots \wedge B_n \end{array}$$

$$d \; \frac{}{[A \to B_1]^{x_1} \wedge \ldots \wedge [A \to B_n]^{x_n}}$$

$$\| s$$

$$C$$



The open-deduction derivation, above centre, is constructed from: a subderivation t^n whose conclusion is an n-fold conjunction; an implication from A, corresponding to the abstraction λy within the distributor term on the left; a distributor inference d; and a subderivation s whose assumptions $A \to B_i$ correspond to the variables x_i bound by the distributor in the term s on the left. The graphical depiction of the distributor is rotated 180 degrees compared to the derivation; that t^n is of multiplicity n means it accepts n input wires.

As is suggested by the graphical representation, the distributor contains an implicit *unsharing* or *co-sharing* node, of the kind used in optimal reduction graphs. The crucial problem in making reduction graphs possible, first solved in [Lam90], 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. Currently, no other way is known that is compatible with typing: in deep inference, to implement a β-reduction involving an implicit abstraction of a distributor appears to require inference rules that are unsound. This is in line with what appears to be a more generally held belief: that it may not be possible to give typing derivations for optimal reduction graphs.

In managing sharings and distributors it will be convenient to assume certain natural equivalences between terms. First, the notation $[\Gamma]$ will be expanded and formalised. Let $[\gamma]$ denote a closure viewed as an operation on terms, taking an argument to the left, $t[\gamma]$, and let $[\Gamma]$ denote a sequence of such operators, which may be seen as a composed operation on terms, $t[\Gamma]$. The natural structure of $[\Gamma]$ is that of a partially ordered set, where $[x_1, \ldots, x_n \leftarrow s] < [y_1, \ldots, y_m \leftarrow t]$ if any of the binding variables y_i of the second sharing occurs free in the body of the first, s. Then, let the *free variables* $\mathsf{FV}[\gamma]$ and the *binding variables* $\mathsf{BV}[\gamma]$ of a closure be defined as follows.

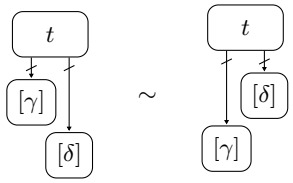
$$\mathsf{FV}[x_1, \ldots, x_n \leftarrow t] = \mathsf{FV}(t) \qquad \mathsf{BV}[x_1, \ldots, x_n \leftarrow t] = \{x_1, \ldots, x_n\}$$

$$\mathsf{FV}[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t] = \mathsf{FV}(\lambda y.t) \qquad \mathsf{BV}[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t] = \{x_1, \ldots, x_n\}$$

Atomic lambda terms will be taken up to the equivalence (1).

$$t[\gamma][\delta] \;\sim\; t[\delta][\gamma] \qquad \text{if} \quad \mathsf{FV}[\gamma] \cap \mathsf{BV}[\delta] = \emptyset \tag{1}$$

In other words, if $[\gamma] < [\delta]$ then they must be applied in the order $t[\gamma][\delta]$; otherwise they may be permuted. The equivalence (1) is illustrated below. A barred arrow ($\downarrow$) denotes a bundle of wires, and is used in place of ellipsis where space is scarce.



The function $[\![-]\!] : \Lambda_A \to \Lambda$, called *denotation*, maps the atomic lambda calculus onto the lambda calculus. Actual substitutions (of t for x in s) are denoted $s\{t/x\}$.

$$[\![x]\!] = x$$

$$[\![\lambda x.u]\!] = \lambda x.[\![u]\!]$$

$$[\![(u)t]\!] = ([\![u]\!])[\![t]\!]$$

$$[\![u[x_1, \ldots, x_n \leftarrow t]]\!] = [\![u]\!]\{t/x_1\} \ldots \{t/x_n\}$$

$$[\![u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle t_1, \ldots, t_n\rangle[\Gamma]]]\!] = [\![u]\!]\{\lambda y.[\![t_1[\Gamma]]\!]/x_1\} \ldots \{\lambda y.[\![t_n[\Gamma]]\!]/x_n\}$$

In the last case, in $t_i[\Gamma]$ the closures in $[\Gamma]$ may contain binding variables not occurring in t_i, so the linearity condition on sharing terms is strictly speaking violated; however, this does not pose a problem, since the closures in $[\Gamma]$ are immediately translated into actual substitutions.

Proposition 3. *For any two terms u and v such that $u \sim v$, $[\![u]\!] = [\![v]\!]$.*

Proof. Simple case analysis. □

3. Sharing Reductions

The aim of the sharing and distributor constructors is to implement atomic duplication. While there are obvious similarities in notation and design between the atomic lambda calculus and expicit substitution calculi, this aim is in some sense orthogonal to that of explicit substitutions, which is to exercise more fine-grained control over the process of substitution within a term calculus. The difference in aim manifests itself in the reduction rules for closures, which will be discussed over the course of this section.

Definition 4. The relation $\rightsquigarrow_S$ is the rewrite relation on Λ_A induced by rewrite rules (2)–(12).

The first set of rewrite rules, (2)–(7) below, moves closures towards the root of the term; in particular, they can be brought out of the scope of a distributor, allowing the rewrite rule (12) to be applied. The direction of rewriting is directly opposite that of explicit substitution calculi, in which explicit substitutions move towards the leaves. Moreover, the rewrite rules (2)–(7) are equalities on the graphical representation, while rewriting in explicit substitution calculi is not, as it may duplicate closures.

$$\lambda x.t[\gamma] \;\rightsquigarrow_S\; (\lambda x.t)[\gamma] \quad \text{if} \quad x \notin \mathsf{FV}[\gamma] \tag{2}$$

$$(u[\gamma])t \;\rightsquigarrow_S\; ((u)t)[\gamma] \tag{3}$$

$$(u)t[\gamma] \;\rightsquigarrow_S\; ((u)t)[\gamma] \tag{4}$$

$$\langle t_1, \ldots, t_i[\gamma], \ldots, t_n\rangle \;\rightsquigarrow_S\; \langle t_1, \ldots, t_n\rangle[\gamma] \tag{5}$$

$$u[x_1, \ldots, x_n \leftarrow t[\gamma]] \;\rightsquigarrow_S\; u[x_1, \ldots, x_n \leftarrow t][\gamma] \tag{6}$$

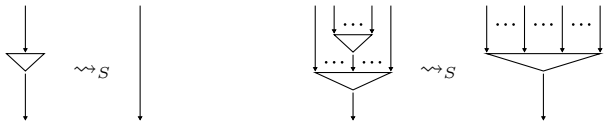
$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t[\gamma]] \;\rightsquigarrow_S\; u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t][\gamma] \quad \text{if} \quad y \notin \mathsf{FV}[\gamma] \tag{7}$$

In a second set of rewrite rules, below, unary sharings are applied as actual substitutions, and consecutive sharings are compounded.

$$u[x \leftarrow t] \;\rightsquigarrow_S\; u\{t/x\} \tag{8}$$

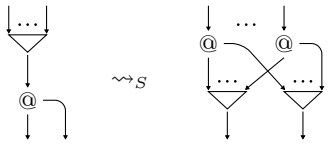
$$u[x_1, \ldots, x_n \leftarrow y_i][y_1, \ldots, y_i, \ldots, y_m \leftarrow t] \;\rightsquigarrow_S\; u[y_1, \ldots, y_{i-1}, x_1, \ldots, x_n, y_{i+1}, \ldots, y_m \leftarrow t] \tag{9}$$

Note that while unary sharings, in the rewrite (8), do not occur in terms translated from the lambda calculus, they may be created by combining for instance a binary sharing with a weakening in the rewrite (9). Both rewrite rules are illustrated graphically below.

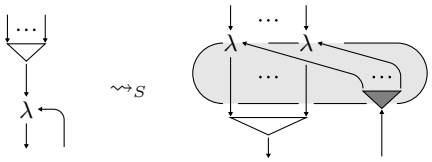


The third set of rewrite rules, (10)–(12), concerns the actual duplication of terms. These are the atomic duplication steps central to the calculus. A graphical illustration is pictured below each rewrite. Together, these give an idea of why the reduction rules are (almost) exhaustive: in (10), a sharing meets an application; in (11), a sharing meets an abstraction; and in (12) a sharing meets a co-sharing, at the end of the scope of a distributor. A missing case is when a sharing meets a distributor; the case is redundant, and has been omitted for the complexity of the terms involved (for more see Section 7).

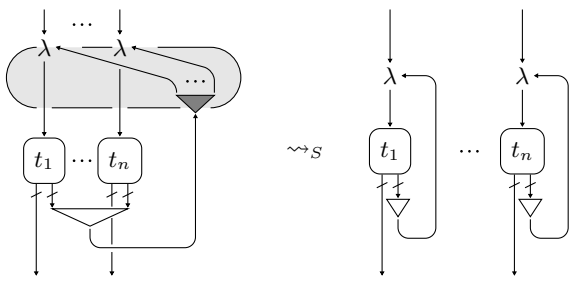
$$u[x_1, \ldots, x_n \leftarrow (v)t] \;\rightsquigarrow_S\; u\{(y_1)z_1)/x_1\} \ldots \{(y_n)z_n)/x_n\}[y_1, \ldots, y_n \leftarrow v][z_1, \ldots, z_n \leftarrow t] \tag{10}$$



$$u[x_1, \ldots, x_n \leftarrow \lambda x.t] \;\rightsquigarrow_S\; u[x_1, \ldots, x_n \twoheadleftarrow \lambda x.\langle y_1, \ldots, y_n\rangle[y_1, \ldots, y_n \leftarrow t]] \tag{11}$$



$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle t_1, \ldots, t_n\rangle[y_{1,1}, \ldots, y_{1,k_1}, \ldots, y_{n,1}, \ldots, y_{n,k_n} \leftarrow y]] \;\rightsquigarrow_S\; u\{\lambda y_1.t_1[y_{1,1}, \ldots, y_{1,k_1} \leftarrow y_1]/x_1\} \ldots \{\lambda y_n.t_n[y_{n,1}, \ldots, y_{n,k_n} \leftarrow y_n]/x_n\} \quad \text{where } \{y_{i,1}, \ldots, y_{i,k_i}\} \subseteq \mathsf{FV}(t_i) \text{ for every } i \leq n \tag{12}$$



A minor remark regarding this rule: a distributor of the form below left, where y occurs exactly once in one t_i, may be reduced as though it were of the form below right.

$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle t_1, \ldots, t_n\rangle] \quad \sim \quad u[x_1, \ldots, x_n \twoheadleftarrow \lambda y'.\langle t_1, \ldots, t_n\rangle[y \leftarrow y']]$$

It is interesting to consider the nullary instances of the last three rules in isolation.

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

With the other sharing rewrite rules, these implement reduction paths of the following kind.

$$t[\leftarrow u] \rightsquigarrow_S^\star t[\leftarrow x_1] \ldots [\leftarrow x_n] \quad \text{where} \quad \{x_1, \ldots, x_n\} = \mathsf{FV}(u)$$

The central properties of sharing reductions ($\rightsquigarrow_S$) are treated below.

Proposition 5. *The reduction $\rightsquigarrow_S$ is strongly normalising.*

The proof of Proposition 5 is postponed until Section 5, 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.

Proposition 6. *The closed atomic lambda terms in sharing normal form are precisely the image of $(\!|-|\!)$ (on closed lambda-terms); moreover, the translations $(\!|-|\!)$ and $[\![-]\!]$ are each other's inverse for closed terms in sharing normal form.*

Proof. Simple case analysis. □

Proposition 7. *For any atomic lambda terms u and v, if $u \rightsquigarrow_S v$, then $[\![u]\!] = [\![v]\!]$.*

Proof. Simple case analysis. □

Theorem 8. *The reduction $\rightsquigarrow_S$ is strongly normalising and confluent.*

Proof. The reduction is strongly normalising by Proposition 5. By Proposition 7 its denotation is preserved under reductions; and by Proposition 6 normal forms are in 1–1 correspondence with denotations (while the proposition only deals with closed terms, the result generalises to arbitrary terms immediately by closing them). It follows that $\rightsquigarrow_S$ is confluent. □

4. Typing

A system of simple types for the atomic lambda calculus is available via the typing rules presented below. As for the atomic lambda calculus there are two kinds of expressions. The first are *typing judgements for atomic lambda terms*, $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$, where t is an atomic lambda term, $A_1, \ldots, A_n, B$ are formulae with root connective $\rightarrow$, and $x_1, \ldots, x_n$ are distinct variables which are exactly the free variables of t. The second are *typing judgements for terms of multiplicity $k > 0$*, $x_1 : A_1, \ldots x_n : A_n \vdash t^k : B_1 \wedge \cdots \wedge B_k$, where t^k is a term of multiplicity k, $A_1, \ldots, A_n, B_1, \ldots, B_k$ are formulae with root connective $\rightarrow$, and $x_1, \ldots, x_n$ are distinct variables which are exactly the free variables of t. The antecedent $x_1 : A_1, \ldots, x_n : A_n$ of a typing judgement is considered a set, and denoted Γ or Δ. In addition, $x_1 \ldots x_n : B$ abbreviates $x_1 \colon B, \ldots, x_n : B$.

Typing rules for atomic lambda terms:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,\lambda \qquad \frac{\Gamma \vdash u : A \to B \qquad \Delta \vdash v : A}{\Gamma, \Delta \vdash (u)v : B}\,@$$

$$\frac{}{x : A \vdash x : A}\,\mathsf{ax} \qquad \frac{\Gamma, x_1 \dots x_n : B \vdash u : A \qquad \Delta \vdash t : B}{\Gamma, \Delta \vdash u[x_1, \dots, x_n \leftarrow t] : A}\,\leftarrow$$

$$\frac{\Gamma, x_1 : A \to B_1, \dots, x_n : A \to B_n \vdash s : C \qquad \Delta, y : A \vdash t^n : B_1 \wedge \dots \wedge B_n}{\Gamma, \Delta \vdash s[x_1, \dots, x_n \twoheadleftarrow \lambda y.t^n] : C}\,\twoheadleftarrow$$

Typing rules for terms of multiplicity k:

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \cdots \qquad \Gamma_k \vdash t_k : A_k}{\Gamma_1, \dots, \Gamma_k \vdash \langle t_1, \dots, t_k \rangle : A_1 \wedge \dots \wedge A_k}\,\langle\,\rangle_k$$

$$\frac{\Gamma, x_1 \dots x_n : B \vdash u^k : A_1 \wedge \dots \wedge A_k \qquad \Delta \vdash t : B}{\Gamma, \Delta \vdash u^k[x_1, \dots, x_n \leftarrow t] : A_1 \wedge \dots \wedge A_k}\,\leftarrow_k$$

$$\frac{\Gamma, x_1 : A \to B_1, \dots, x_n : A \to B_n \vdash u^k : C_1 \wedge \dots \wedge C_k \qquad \Delta, y : A \vdash t^n : B_1 \wedge \dots \wedge B_n}{\Gamma, \Delta \vdash u^k[x_1, \dots, x_n \twoheadleftarrow \lambda y.t^n] : C_1 \wedge \dots \wedge C_k}\,\twoheadleftarrow_k$$

Proposition 9. *Sharing reductions preserve typing.*

Proof. Of the rewrite rules (2)–(12) only the duplication steps, (10)–(12), will be treated; the others are similar (but smaller).

For rewrite rule (10) the typing derivations of the left and right side of the rule are given in Figure 1. In the second derivation, π_1' is obtained from π_1 by replacing each axiom $x_i : C \vdash x_i : C$ by a proof of $z_i : B, y_i : B \to C \vdash (y_i)z_i : C$.

For rewrite rule (11) the typing derivations are displayed in Figure 2.

For rewrite rule (12) the derivations are given in Figure 3. In the second derivation, π_1' is obtained from π_1 by replacing each axiom $x_i : B \to C \vdash x_i : B \to C$ by a proof of $\Delta_i \vdash \lambda y_i.t_i[y_{i,1}, \dots, y_{i,k_i} \leftarrow y_i] : B \to C$. □

The following facts are easily observed.

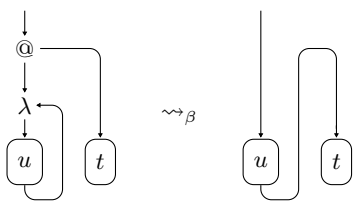
Proposition 10. *For any two terms u and v such that $u \sim v$, if $v : A$ then $u : A$.*

Proposition 11. *For any lambda-term M, if $M : A$ then $(\!|M|\!) : A$.*

5. Beta-reduction

The definition of β-reduction in the atomic lambda calculus is identical to that of the regular lambda calculus; it is given and illustrated below.

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



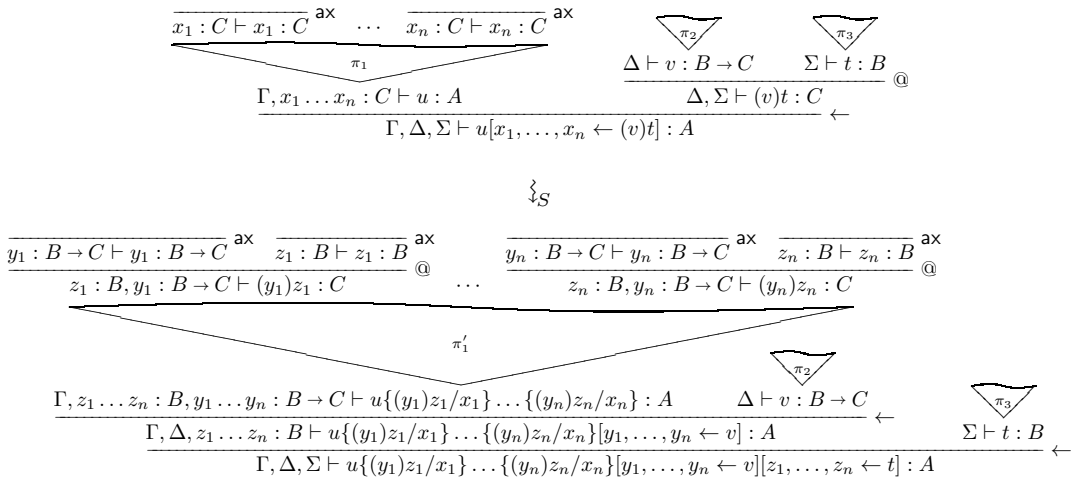


Figure 1: Derivations for rewrite rule (10) in the proof of Proposition 9

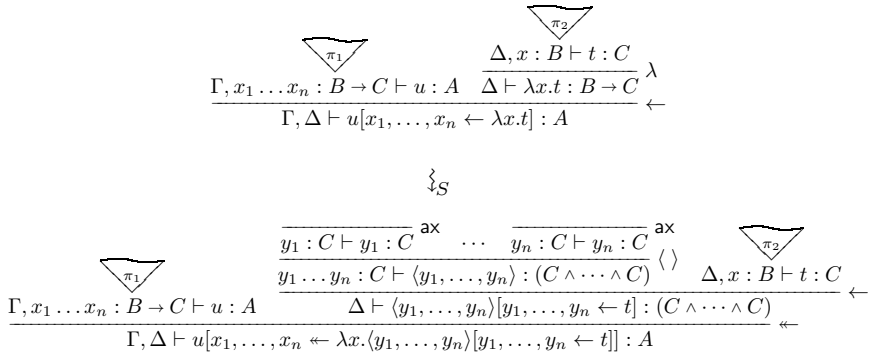


Figure 2: Derivations for rewrite rule (11) in the proof of Proposition 9

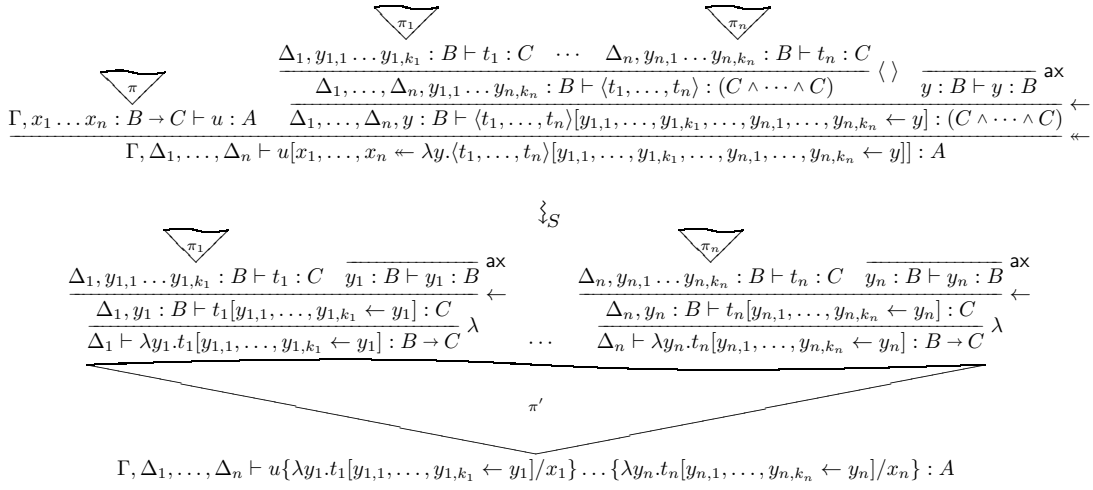
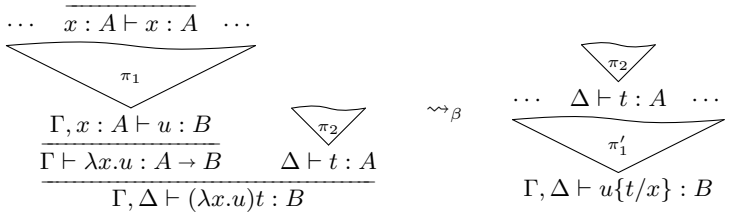


Figure 3: Derivations for rewrite rule (12) in the proof of Proposition 9

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 ($\rightsquigarrow_S$).

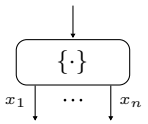
Theorem 12. *β-reduction preserves typing.*

Proof. By the following derivations, where π_1' is obtained from π_1 by replacing each axiom $x : A \vdash x : A$ by the derivation π_2 of $\Delta \vdash t : A$.



□

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 a 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, defined as follows. Here, let $[\![u\{\cdot\}]\!] = (M\{\cdot\}, \sigma)$, and let $[\![v\{\cdot\}[\Gamma]]\!] = (N\{\cdot\}, \tau)$.

$$\begin{aligned}
[\![\{\cdot\}_{x_1,\dots,x_n}]\!] &= (\{\cdot\},\varnothing)\\
[\![\lambda x.u\{\cdot\}]\!] &= (\lambda x.M\{\cdot\},\sigma)\\
[\![(u\{\cdot\})t]\!] &= ((M\{\cdot\})[\![t]\!],\sigma)\\
[\![(t)u\{\cdot\}]\!] &= (([\![t]\!])M\{\cdot\},\sigma)\\
[\![t[y_1,\dots,y_m \leftarrow u\{\cdot\}]]\!] &= ([\![t]\!]\{M\{\cdot\}/y_1\}\dots\{M\{\cdot\}/y_m\},\sigma)\\
[\![u\{\cdot\}[y_1,\dots,y_m \leftarrow t]]\!] &= (M\{\cdot\}\{[\![t]\!]/y_1\}\dots\{[\![t]\!]/y_n\},\\
&\qquad \sigma\{[\![t]\!]/y_1\}\dots\{[\![t]\!]/y_m\}|_{\{x_1,\dots,x_n\}})\\
[\![t[y_1,\dots,y_m \twoheadleftarrow \lambda z.\langle t_1,\dots,v\{\cdot\},\dots,t_m\rangle[\Gamma]]]\!] &= ([\![t]\!]\{N_1/y_1\}\dots\{\lambda z.N\{\cdot\}/y_i\}\dots\{N_m/y_m\},\sigma)\\
&\qquad \text{where } N_i = \lambda z.[\![t_i[\Gamma]]\!]\\
[\![u\{\cdot\}[y_1,\dots,y_m \twoheadleftarrow \lambda z.\langle t_1,\dots,t_m\rangle[\Gamma]]]\!] &= (M\{\cdot\}\{N_1/y_1\}\dots\{N_m/y_m\},\\
&\qquad \sigma\{N_1/y_1\}\dots\{N_m/y_m\}|_{\{x_1,\dots,x_n\}})\\
&\qquad \text{where } N_i = \lambda z.[\![t_i[\Gamma]]\!]
\end{aligned}$$

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

Proof. Simple case analysis. □

A first use of contexts is the proof of Proposition 5, that was postponed in Section 3.

Proposition 5 (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 $1/2$, and for a sharing is the size of the denotation of the shared term, which is defined as follows. Given a sharing $[\gamma] = [x_1,\dots,x_n \leftarrow t]$ 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. Notice that both measures are invariant under Equation (1).

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)–(7)); or
- one sharing is removed, while the weights of the others remain unchanged (rewrite rules (8)–(9)); or
- one or more closures $[\delta]$ are replaced with some of strictly lower weight: in rewrite rule (10), one sharing of weight n is replaced by two of weight $n-1$; in rewrite rule (11) a sharing of weight $(n \geq 1)$ is replaced by a distributor (of weight $1/2$) and a sharing of weight $n-1$; and in rewrite rule (12) a distributor of weight $1/2$ and 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. *For any terms* u *and* v*, if* $u \rightsquigarrow_\beta v$ *then* $[\![u]\!] \rightsquigarrow_\beta^\star [\![v]\!]$.

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,

$$[\![v\{(\lambda x.u)t]\}]\!] \;=\; M\{((\lambda x.[\![u]\!])[\![t]\!])\sigma\} \;\rightsquigarrow_\beta^n\; M\{([\![u]\!]\{[\![t]\!]/x\})\sigma\} \;=\; [\![v\{u\{t/x\}\}]\!]$$

where n is the number of holes in $M\{\cdot\}$. □

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 $[\![u]\!] \rightsquigarrow_\beta M$ then there exist atomic lambda terms v and t such that $u \rightsquigarrow_S^\star v \rightsquigarrow_\beta t$ and $[\![t]\!] = M$.*

Proof. Take v to be the sharing normal form of u. □

6. Preservation of strong normalisation

The main result for the atomic lambda calculus presented in this paper will be the preservation of strong normalisation with respect to the lambda calculus. The challenge, as with many sharing mechanisms and explicit substitution calculi, is that reduction in the atomic lambda calculus may take place inside a weakening $[\leftarrow t]$. Beta-steps within a weakening are 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.

In order to separate the problem of weakened reductions from the details of the sharing mechanisms of the atomic lambda calculus, the proof is split into two stages. In the first stage, a lambda calculus with weakening (the *explicit weakening calculus*) will be defined, as a denotation for the atomic lambda calculus. It will be shown that a beta-step in the atomic lambda calculus corresponds to at least one step in the explicit weakening calculus. In the second stage it will be shown that the explicit weakening calculus satisfies preservation of strong normalisation of lambda-terms, implying the same for the atomic lambda calculus.

Definition 19. The *explicit weakening calculus*, or *w-calculus*, is given by the following grammar.

$$S, T, U \quad := \quad x \quad | \quad \lambda x.T \quad \text{where } x \in \mathsf{FV}(T) \quad | \quad (S)U \quad | \quad S[\leftarrow U] \quad | \quad \bullet$$

Terms of the w-calculus are called *w-terms*. Beta-reduction in the w-calculus is as normal:

$$(\lambda x.T)U \quad \rightsquigarrow_\beta \quad T\{U/x\}$$

Deletion of weakenings in the w-calculus proceeds as follows.

$$\begin{array}{rcl@{\qquad}rcl}
S[\leftarrow \lambda x.U] & \rightsquigarrow_w & S[\leftarrow U\{\bullet/x\}] & \lambda x.T[\leftarrow U] & \rightsquigarrow_w & (\lambda x.T)[\leftarrow U]\ (^{**}) \\
S[\leftarrow (T)U] & \rightsquigarrow_w & S[\leftarrow T][\leftarrow U] & (S[\leftarrow U])T & \rightsquigarrow_w & ((S)T)[\leftarrow U] \\
S[\leftarrow \bullet] & \rightsquigarrow_w & S & (S)T[\leftarrow U] & \rightsquigarrow_w & ((S)T)[\leftarrow U] \\
S[\leftarrow U] & \rightsquigarrow_w & S\ (^{*}) & S[\leftarrow T[\leftarrow U]] & \rightsquigarrow_w & S[\leftarrow T][\leftarrow U]
\end{array}$$

* if U is a subterm of S $\qquad$ ** if $x \notin \mathsf{FV}(U)$

As an interpretation of the atomic lambda calculus, the w-calculus is *denotational* with respect to sharing, but *operational* with respect to weakening: sharing will be modelled by duplication (via substitution), while the main feature of the w-calculus is an elaborate set of deletion rules designed to mimick those of the atomic lambda calculus.

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

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

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

$$[\![t[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle u_1, \ldots, u_n\rangle[\Gamma]]]\!]_w = \begin{cases} [\![t[\Gamma]]\!]_w\{\bullet/y\} & \text{if } n = 0 \\ [\![t]\!]_w\{\lambda y.[\![u_1[\Gamma]]\!]_w/x_1\} \ldots \{\lambda y.[\![u_n[\Gamma]]\!]_w/x_n\} & \text{otherwise} \end{cases}$$

In the translation 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 weakened terms $[\leftarrow v]$. The rewrite rule $S[\leftarrow U] \rightsquigarrow_w S$ 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. First, the statement is shown for rewriting at the root of a term. The case for β-reduction follows by the equation below; note that here, x occurs exactly once in t, and at least once in $[\![t]\!]_w$.

$$[\![(\lambda x.t)u]\!]_w \;=\; (\lambda x.[\![t]\!]_w)[\![u]\!]_w \;\rightsquigarrow_\beta\; [\![t]\!]_w\{[\![u]\!]_w/x\} \;=\; [\![t\{u/x\}]\!]_w$$

Of the sharing permutations (2)–(7) most cases are easily verified. The case (2) (for $n \neq 0$) is treated explicitly, below; that for (7) is similar.

$$\begin{array}{rcl}
[\![t[x_1, \ldots, x_n \leftarrow u[\leftarrow v]]]\!]_w & = & [\![t]\!]_w\{[\![u]\!]_w[\leftarrow [\![v]\!]_w]/x_1\} \ldots \{[\![u]\!]_w[\leftarrow [\![v]\!]_w]/x_n\} \\
& \rightsquigarrow_w^+ & [\![t]\!]_w\{[\![u]\!]_w/x_1\} \ldots \{[\![u]\!]_w/x_n\}[\leftarrow [\![v]\!]_w] \ldots [\leftarrow [\![v]\!]_w] \\
& \rightsquigarrow_w^+ & [\![t]\!]_w\{[\![u]\!]_w/x_1\} \ldots \{[\![u]\!]_w/x_n\}[\leftarrow [\![v]\!]_w] \\
& = & [\![t[x_1, \ldots, x_n \leftarrow u][\leftarrow v]]\!]_w
\end{array}$$

In the second step of the above sequence the multiple copies of $[\leftarrow [\![v]\!]_w]$ are permuted towards the root of $[\![t]\!]_w$. This is possible since no λy inside t can bind in v, by the structure of the original term $t[x_1, \ldots, x_n \leftarrow u[\leftarrow v]]$. Then in the third step duplicate instances of $[\leftarrow [\![v]\!]_w]$ are deleted. Next, the case for the rewrite rule (8) is immediate by the following.

$$[\![u[x \leftarrow t]]\!]_w \;=\; [\![u]\!]_w\{[\![t]\!]_w/x\} \;=\; [\![u\{t/x\}]\!]_w$$

For the rewrite rule (9) there are several sub-cases: that for $n \neq 0$ is immediate since sharings are interpreted by substitutions; that for $n = 0$ and $m > 1$ follows by the sequence below, and that for $n = 0$ and $m = 1$ follows similarly.

$$\begin{array}{rcl}
[\![u[\leftarrow y_i][y_1, \ldots, y_i, \ldots, y_m \leftarrow t]]\!]_w & = & [\![u]\!]_w[\leftarrow [\![t]\!]_w]\{[\![t]\!]_w/y_1\} \ldots \{[\![t]\!]_w/y_m\} \\
& \rightsquigarrow_w & [\![u]\!]_w\{[\![t]\!]_w/y_1\} \ldots \{[\![t]\!]_w/y_m\} \\
& = & [\![u[y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m \leftarrow t]]\!]_w
\end{array}$$

Note that the weakening $[\leftarrow [\![t]\!]_w]$ may be removed by the rule $S[\leftarrow U] \rightsquigarrow_w S$ since the substitutions $\{[\![t]\!]_w/y_i\}$ ensure that $[\![t]\!]_w$ occurs in the body of the term.

Then for the actual duplication steps, the case (10) will be omitted. For the rewrite rule (11) there are again two cases. The case $n = 0$ is as follows,

$$\begin{array}{rcl}
[\![u[\leftarrow \lambda x.t]]\!]_w & = & [\![u]\!]_w[\leftarrow \lambda x.[\![t]\!]_w] \\
& \rightsquigarrow_w & [\![u]\!]_w[\leftarrow [\![t]\!]_w\{\bullet/x\}] \\
& = & [\![u[\twoheadleftarrow \lambda x.\langle\rangle[\leftarrow t]]]\!]_w
\end{array}$$

and the case $n \neq 0$ is as follows.

$$\begin{array}{rcl}
[\![u[x_1, \ldots, x_n \leftarrow \lambda x.t]]\!]_w & = & [\![u]\!]_w\{\lambda x.[\![t]\!]_w/x_1\} \ldots \{\lambda x.[\![t]\!]_w/x_n\} \\
& = & [\![u]\!]_w\{\lambda x.y_1\{[\![t]\!]_w/y_1\}/x_1\} \ldots \{\lambda x.y_n\{[\![t]\!]_w/y_n\}/x_n\} \\
& = & [\![u[x_1, \ldots, x_n \twoheadleftarrow \lambda x.\langle y_1, \ldots, y_n\rangle[y_1, \ldots, y_n \leftarrow t]]]\!]_w
\end{array}$$

Similarly, for the rewrite rule (12) the case $n = 0$ is as follows,

$$[\![u[\twoheadleftarrow \lambda y.\langle\rangle[\leftarrow y]]]\!]_w \;=\; [\![u]\!]_w[\leftarrow \bullet] \;\rightsquigarrow_w\; [\![u]\!]_w$$

and the case $n \neq 0$ is as follows.

$$\begin{array}{rcll}
\multicolumn{4}{l}{[\![u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.\langle t_1, \ldots, t_n\rangle[y_{1,1}, \ldots, y_{1,k_1}, \ldots, y_{n,1}, \ldots, y_{n,k_n} \leftarrow y]]]\!]_w} \\
\quad = & [\![u]\!]_w\{T_1/x_1\} \ldots \{T_n/x_n\} & \text{where } T_i \;=\; \lambda y.[\![t_i]\!]_w\{y/y_{i,1}\} \ldots \{y/y_{i,k_i}\} \\
\quad =_\alpha & [\![u\{t'_i/x_1\} \ldots \{t'_n/x_n\}]\!]_w & \text{where } t'_i \;=\; \lambda y_i.t_i[y_{i,1}, \ldots, y_{i,k_i} \leftarrow y_i]
\end{array}$$

Finally, for rewrite steps not at the root of a term, it is easily observed that since the translation $[\![-]\!]_w$ is non-deleting, if $[\![t]\!]_w \rightsquigarrow [\![u]\!]_w$ then $[\![s\{t\}]\!]_w \rightsquigarrow^+ [\![s\{u\}]\!]_w$. □

The natural interpretation of a w-term by a lambda term, which discards all weakened terms $[\leftarrow U]$, is denoted $\lfloor - \rfloor$. The following is easily observed.

Proposition 22. *For any atomic lambda term t, $\lfloor [\![t]\!]_w \rfloor = [\![t]\!]$.*

Preservation of strong normalisation for the explicit weakening calculus will be shown using the notion of a *perpetual strategy* [Bar84]. The main property of such a reduction strategy is that if it terminates for a given term, then this term is strongly normalising. By translating an infinite perpetual reduction on w-terms to an infinite reduction on lambda terms, preservation of strong normalisation then follows.

Definition 23. The *perpetual strategy* on a w-term U is defined as a sequence

$$U = U_1 \rightsquigarrow U_2 \rightsquigarrow U_3 \rightsquigarrow \dots$$

where $U_{i+1} = \omega(U_i)$, defined as follows:

$$\omega(x) = x \qquad \omega(\lambda x.T) = \lambda x.\omega(T) \qquad \omega(T[\leftarrow U]) = \omega(T)[\leftarrow U]$$

$$\omega((T)U) = \begin{cases} (T')U[\leftarrow V] & \text{if } T = T'[\leftarrow V] \\ T'\{U/x\} & \text{if } T = \lambda x.T' \text{ and } x \in \mathsf{FV}\lfloor T' \rfloor \\ T'\{U/x\} & \text{if } T = \lambda x.T' \text{ and } x \notin \mathsf{FV}\lfloor T' \rfloor \text{ and } U \text{ is on normal form} \\ (T)\omega(U) & \text{if } T = \lambda x.T' \text{ and } x \notin \mathsf{FV}\lfloor T' \rfloor \text{ and } U \text{ is not on normal form} \\ (\omega(T))U & \text{otherwise} \end{cases}$$

Lemma 24. *If $\omega(U\{S\}) = U\{\omega(S)\}$, then no reduction path starting from $U\{S\}$ can duplicate S nor instantiate free variables in S. More formally, if $\omega(U\{S\}) = U\{\omega(S)\}$, then any V such that $U\{S\} \rightsquigarrow V$ is of the form $V = U'\{S'\}$, such that $U\{\bullet\} \rightsquigarrow U'\{\bullet\}$ and $S \rightsquigarrow S'$.*

Proof. By induction on the definition of ω. □

Lemma 25. *If N is a lambda term, and $U_1, U_2, U_3, \dots$ is the perpetual strategy on $U_1 = [\![(\!|N|\!)]\!]_w$, then $U_i = U_{i+1}$ if and only if U_i is on normal form.*

Proof. Note that the weakenings in U_1 consist only of variables, and that any redex outside of a weakening will eventually be reduced in the perpetual strategy. By the definition of ω and the above Lemma, any weakened terms appearing in the perpetual reduction path from U_1 will always be on normal form. □

Lemma 26. *For N a lambda term and $i \geq 0$, if $\omega^i([\![(\!|U|\!)]\!]_w)$ has an infinite reduction path, then the perpetual stategy on $\omega^{i+1}([\![(\!|N|\!)]\!]_w)$ does not terminate.*

Proof. Let $U_1, U_2, U_3, \dots$ be an infinite reduction path from $\omega^i([\![(\!|N|\!)]\!]_w) = U_1$, and let S be the smallest subterm of U_1 such that $\omega(U_1\{S\}) = U_1\{\omega(S)\}$. Assume $S = (T[\leftarrow W])V$; the two cases where $S = (\lambda x.V)T$ are analogous. There are two cases to consider: the case where the redex S is eventually reduced in the infinite path, and the case where it is not.

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

  $$U_1,\ U_2,\ \dots,\ U_k\{(T_k[\leftarrow W]_k)V_k\},\ U_k\{(T_k)V_k[\leftarrow W]_k\}$$

  where $U_1\{\bullet\} \rightsquigarrow U_k\{\bullet\}$, $T \rightsquigarrow T_k$, $V \rightsquigarrow V_k$ and $W \rightsquigarrow W_k$. It then follows that $U_1\{(T)V[\leftarrow W]\} \rightsquigarrow^\star U_k\{(T_k)V_k[\leftarrow W]_k\}$.
- If S is never reduced, then one of $U_1\{\bullet\}$, T, W or V must have an infinite path by Lemma 24, and so must $U_1\{(T)V[\leftarrow W]\}$.

□

Lemma 27. *For any strongly normalising lambda term N, the explicit weakening term $[\![(\!|N|\!)]\!]_w$ is strongly normalising.*

Proof. By contradiction. Let $M = [\![(\!|N|\!)]\!]_w$ have an infinite reduction path, then the perpetual strategy on M has an infinite number of beta steps. Moreover, no reduction step happens inside a weakening. Let M_1, M_2, ..., be the perpetual strategy on M. As $N = \lfloor M \rfloor$, it follows that $\lfloor M_1 \rfloor$, $\lfloor M_2 \rfloor$, ..., is an infinite reduction path on N. □

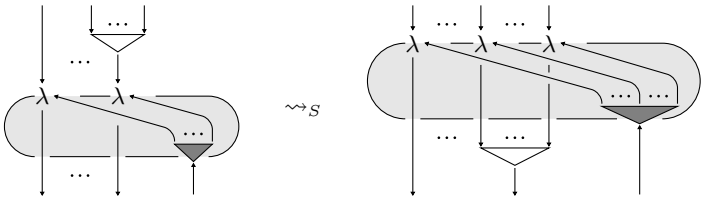
Theorem 28. *The atomic lambda calculus satisfies preservation of strong normalisation.*

Proof. For any lambda term N, $(\!|N|\!)$ is strongly normalising if $[\![(\!|N|\!)]\!]_w$ is, by Lemma 21, and $[\![(\!|N|\!)]\!]_w$ is strongly normalising if N is, by Lemma 27. □

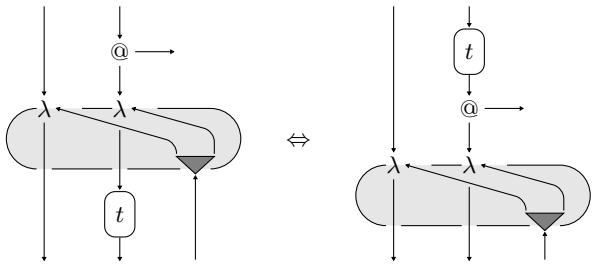
7. Conclusions and Further Work

The atomic lambda calculus presented in this paper is, to the best of the authors' knowledge, the first implementation of the lambda calculus in a term calculus enjoying the atomicity property, the ability to duplicate and delete individual term constructors during reductions. While the main innovation of the calculus, the *distributor* term constructor, may at first appear confounded, it is supported by clear graphical intuitions and solid proof-theoretic foundations. The result is to bring reduction techniques previously confined to optimal reduction graphs within the domain of term calculi that admit typing.

The present exposition has been aimed at establishing the basic theory of the atomic lambda calculus, including the translation to and from the standard lambda calculus, simulation of β-reduction, and preservation of strong normalisation. Future work will be to investigate the reduction properties of the atomic lambda calculus and variations of it. A simple adaptation, here left out for simplicity of exposition, is to allow the reduction step below. Currently, a distributor blocks a sharing that meets it from above, for example in the term $u[x_1, \ldots, x_n \leftarrow y_i][y_1, \ldots, y_m \twoheadleftarrow \lambda z.t^m]$, until the duplication within the scope of the distributor has been resolved completely. The reduction step below amends this in a natural way: it closes the confluence diagram where two sharing nodes, one above another, meet an abstraction.



A more serious drawback of the present calculus is that, as mentioned in Section 2, the implicit abstractions within a distributor cannot be part of a redex. One promising way around this problem is as follows: when an application connects to a distributor, it may be possible to *permute* subterms from inside this combination to outside it, or vice versa, as suggested by the illustration below. For the standard lambda calculus, permutations of this kind are explored in [Reg94, AK12]. The atomic lambda calculus is a natural match for them.



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.

Bibliography

[ACCL91] M. Abadi, Luca Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[AK12] Beniamino Accattoli and Delia Kesner. The permutative λ-calculus. In *LPAR*, pages 23–36, 2012.

[Bal12] Thibaut Balabonski. A unified approach to fully lazy sharing. POPL, pages 469–480, 2012.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[BLM07] 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.

[GGP10] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In *LIPIcs*, volume 6, pages 135–150, 2010.

[Gug07] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.

[Reg94] Laurent Regnier. Une équivalence sur les lambda-termes. *Theor. Comput. Sci.*, 126(2):281–292, 1994.

[Wad71] Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, Oxford, 1971.