A Proof of Strong Normalisation of the Typed Atomic Lambda-Calculus

Tom Gundersen[1], Willem Heijltjes[2], and Michel Parigot[1]

[1] Laboratoire Preuves, Programmes, Systèmes
CNRS & Université Paris Diderot
teg@jklm.no, parigot@pps.univ-paris-diderot.fr
[2] University of Bath
w.b.heijltjes@bath.ac.uk

Abstract. The atomic lambda-calculus is a typed lambda-calculus with explicit sharing, which originates in a Curry-Howard interpretation of a deep-inference system for intuitionistic logic. It has been shown that it allows fully lazy sharing to be reproduced in a typed setting. In this paper we prove strong normalization of the typed atomic lambda-calculus using Tait's reducibility method.

1 Introduction

The *atomic lambda-calculus* is a typeable lambda-calculus with explicit sharing, recently introduced in [13, 12], developed as the Curry–Howard interpretation of a deep-inference proof system for intuitionistic logic. The present paper constitutes an important step in the development of its meta-theory, by extending Tait's reducibility method to show strong normalisation of the simply typed atomic lambda-calculus. The primary motivation for establishing this result is to demonstrate that the atomic lambda-calculus is a natural and well-behaved calculus, to which the main standard techniques and results apply.

Sharing is an approach to efficient computation in lambda-calculi whereby duplication of subterms is deferred in favor of reference to a common representation. It is a leading principle behind, among others, explicit substitution calculi [1, 18, 8, 9, 15, 2], term calculi with strategies or higher-order transformations [14, 3], and sharing graphs in the style of Lamping [17, 4, 21]. The atomic lambda-calculus represents a novel category in this range. As a typeable term calculus it is an alternative to explicit substitution calculi, providing a different perspective on sharing: as in sharing graphs, sharing is evaluated *atomically*, by duplicating individual constructors. A salient property is that the calculus implements *fully lazy sharing* [22, 14, 5], a degree of sharing that, while standard, had previously been achieved in lambda-calculi only by means of external transformations.

The paper [12] details how the atomic lambda-calculus and its sharing mechanisms are derived from *deep inference* [6], a proof methodology where inferences apply *in context*, reminiscent of term rewriting. Sharing in deep inference is by explicit *contraction* rules, which implement atomic duplication by interacting with individual inferences. By embedding natural deduction within the deep-inference formalism *open deduction* [11], duplication in traditional normalisation is broken up into atomic steps. The atomic lambda-calculus is a direct computational interpretation of the resulting proof system. The paper [12] further establishes the technical properties of full laziness and *PSN*, preservation of strong normalisation with respect to the lambda-calculus.

In the present paper strong normalisation for the typed atomic lambda-calculus will be proven using the Tait-reducibility method [20, 10]. Reducibility is an abstract method compatible with higher-order logic, whose application provides a deeper understanding of reduction and its dynamics. The fact that a reducibility proof can be carried out for the atomic lambda-calculus shows the generic character of this extension of the lambda-calculus.

2 The atomic lambda-calculus

The *atomic lambda-calculus* introduced in [12] is a refined lambda-calculus, in which abstraction is split into a linear abstraction and a sharing operation. Duplication and deletion proceeds locally through the evaluation of sharings. The calculus consists of a standard linear lambda-calculus with a sharing construct, extended by a further construction called the *distributor*. The distributor allows to duplicate an abstraction without duplicating its scope: it replaces the abstraction while duplication of its scope is in progress, where the duplicated parts of the scope are stored in a tuple of terms (see also the reduction rules in Section 2.1).

Definition 1. The *atomic lambda-calculus* Λ_A is defined by the grammars

$$\begin{array}{rcccccccc} s,t,u,v,w & := & x & \mid & \lambda x.t & \mid & (t)u & \mid & t[\gamma] \\ [\gamma],[\delta] & := & [x_1,\dots,x_n \leftarrow t] & \mid & [x_1,\dots,x_n \twoheadleftarrow \lambda y.t^n] & & & & \\ t^n & := & \langle t_1,\dots,t_n\rangle & \mid & t^n[\gamma] & & & & \end{array}$$

where (i) $n \geq 0$, (ii) each variable may occur at most once in a term, (iii) in $\lambda x.t$, x must be free in t and becomes bound, (iv) in $\lambda y.t^n$, y must be free in t^n and becomes bound, (v) in $t[\gamma]$ where $[\gamma]$ is $[x_1,\dots,x_n \leftarrow u]$ or $[x_1,\dots,x_n \twoheadleftarrow \lambda y.t^n]$, each x_i must be free in t and becomes bound, and (vi) likewise for $t^n[\gamma]$.

Terms t are *atomic lambda-terms*. The *closures* $[\gamma]$ are called respectively *sharing* and *distributor*, and a nullary sharing $[\leftarrow t]$ is a *weakening*. Atomic lambda-terms not containing a distributor are *basic* terms. A sequence of closures $[\gamma_1]\dots[\gamma_n]$ will be denoted $[\gamma_i]_{i\leq n}$ or $[\Gamma]$. The t^n are *terms of multiplicity* n or n-*terms*, and are of the form $\langle t_1,\dots,t_n\rangle[\Gamma]$. Where possible, terms and n-terms will not be distinguished, and both denoted t,u,v. A sequence of variables $x_1,\dots,x_n$ may be abbreviated $\vec{x}$; a sharing is then denoted $[\vec{x} \leftarrow t]$. Standard notions are: $\mathsf{FV}(u)$ is the set of free variables of u, and $u\{t/x\}$ denotes the substitution of t for x in u. A series of substitutions $\{t_1/x_1\}\dots\{t_n/x_n\}$ is abbreviated $\{t_i/x_i\}_{i\leq n}$.

Atomic lambda terms will be considered up to the congruence ($\sim$) induced by (1) below; note that due to linearity, both terms are only well-defined if both $[\gamma]$ and $[\delta]$ bind only in t.

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

The functions $(\!|-|\!) : \Lambda \to \Lambda_A$ and $[\![-]\!] : \Lambda_A \to \Lambda$ translate between atomic lambda-terms and standard lambda-terms. The former is defined below. For a formal definition of the function $(\!|-|\!)$ see [12]; intuitively, it replaces each abstraction $\lambda x.-$ in a term by $\lambda x. - [x_1,\dots,x_n \leftarrow x]$, where $x_1,\dots,x_n$ replace the occurrences of x, so that $[\![(\!|N|\!)]\!] = N$ for any lambda-term N.

Definition 2. The functions $[\![-]\!]$ and $\{\!|-|\!\}$ interpret atomic lambda-terms and closures respectively as lambda-terms and substitutions. For a sequence of closures $[\Gamma] = [\gamma][\Gamma']$ with $[\Gamma']$ non-empty, let $\{\!|\Gamma|\!\} = \{\!|\gamma|\!\}\{\!|\Gamma'|\!\}$.

$$[\![x]\!] = x \qquad [\![\lambda x.t]\!] = \lambda x.[\![t]\!] \qquad [\![(t)u]\!] = ([\![t]\!])[\![u]\!] \qquad [\![t[\gamma]]\!] = [\![t]\!]\{\!|\gamma|\!\}$$

$$\{\!|x_1,\dots,x_n \leftarrow t|\!\} = \{[\![t]\!]/x_i\}_{i\le n}$$

$$\{\!|x_1,\dots,x_n \twoheadleftarrow \lambda y.\langle t_1,\dots,t_n\rangle[\Gamma]|\!\} = \{\lambda y.[\![t_i]\!]\{\!|\Gamma|\!\}/x_i\}_{i\le n}$$

2.1 Reduction rules

Reduction in the atomic lambda-calculus, denoted $\rightsquigarrow$, consists of two parts: (i) linear β-reduction, denoted $\rightsquigarrow_\beta$: the usual rule (rule β below) applied linearly; (ii) sharing reductions, denoted $\rightsquigarrow_S$, comprising two kinds of rule: (a) *permutations* taking closures outward (rules 2–6), and (b) local *transformations* that evaluate closures (rules 7–10).

Linear β-reduction:

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

Permutations of closures:

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

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

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

$$u[x_1,\dots,x_n \leftarrow t[\gamma]] \;\rightsquigarrow_S\; u[x_1,\dots,x_n \leftarrow t][\gamma] \tag{5}$$

$$u[x_1,\dots,x_n \twoheadleftarrow \lambda y.t^n[\gamma]] \;\rightsquigarrow_S\; u[x_1,\dots,x_n \twoheadleftarrow \lambda y.t^n][\gamma] \quad \text{if } y \in \mathsf{FV}(t^n) \tag{6}$$

Transformations on closures:

$$u[\vec{y} \leftarrow y][\vec{x}, y, \vec{z} \leftarrow t] \;\rightsquigarrow_S\; u[\vec{x}, \vec{y}, \vec{z} \leftarrow t] \tag{7}$$

$$u[x_1,\dots,x_n \leftarrow (v)t] \;\rightsquigarrow_S\; u\{(y_i)z_i/x_i\}_{i\le n}[y_1,\dots,y_n \leftarrow v][z_1,\dots,z_n \leftarrow t] \tag{8}$$

$$u[x_1,\dots,x_n \leftarrow \lambda x.t] \;\rightsquigarrow_S\; u[x_1,\dots,x_n \twoheadleftarrow \lambda x.\langle y_1,\dots,y_n\rangle[y_1,\dots,y_n \leftarrow t]] \tag{9}$$

$$u[x_1,\dots,x_n \twoheadleftarrow \lambda y.\langle t_1,\dots,t_n\rangle[\vec{z} \leftarrow y]] \;\rightsquigarrow_S\; u\{\lambda y_i.t_i[\vec{z}_i \leftarrow y_i]/x_i\}_{i\le n}$$

$$\text{where } \{\vec{z}_i\} = \{\vec{z}\} \cap \mathsf{FV}(t_i) \text{ for every } i \le n \tag{10}$$

The fact that a term u reduces to v in exactly n steps will be denoted $u \rightsquigarrow^n v$, while an arbitrary number of steps is indicated simply by $\rightsquigarrow$. A term u is called *strongly normalisable* if all the reduction sequences starting with u are finite. The set of strongly normalisable terms is denoted $\mathcal{N}$. Reduction in the atomic lambda-calculus commutes 1–1 with substitution, due to the linearity condition on free variables.

Lemma 3. *For atomic lambda-terms u, u', v and v' and variable $x \in \mathsf{FV}(u)$, if $u \rightsquigarrow^1 u'$, then $u\{v/x\} \rightsquigarrow^1 u'\{v/x\}$; and if $v \rightsquigarrow^1 v'$, then $u\{v/x\} \rightsquigarrow^1 u\{v'/x\}$.*

2.2 Basic properties of the atomic lambda-calculus

We collect in this section the main basic properties we are using in the strong normalisation proof. The two main properties are (i) the strong normalisation property of the sharing reduction, and (ii) the decomposition of the computational content of sharings and distributors.

Theorem 4 ([12, Theorem 11]). *The reduction $\rightsquigarrow_S$ is strongly normalising and confluent.*

Sharing reductions preserve the denotation $[\![t]\!]$ of a term [12, Prop. 10]. The normal form under $\rightsquigarrow_S$ of an atomic lambda-term t is called its *unfolding* $\mathsf{u}(t)$. It is a basic term (i.e., no distributors occur) of the form $u[\Gamma]$, where sharing in u occurs only as $\lambda y.v[\vec{x} \leftarrow y]$, of bound variables immediately within the scope of their binder, and where $[\Gamma]$ are sharings $[\vec{x} \leftarrow y]$ of the free variables y in t that occur in shared subterms [12, Prop. 9]. For closed terms, $\mathsf{u}(t) =_\alpha (\![[\![t]\!]]\!)$.

Definition 5. The *unfolded body* $\mathsf{ub}(t)$ of t is the largest subterm of $\mathsf{u}(t)$ not of the form $u[\gamma]$.

The unfolded body of a term is what is duplicated during reduction. To identify the various copies, let a *variant* of a term t be any term obtained from t by renaming certain (bound or free) variables. A variant is *fresh* if all its variables are fresh, and t^i is the fresh variant of t obtained by replacing each variable x by a fresh variable x^i.

For an n-term $t^n = \langle t_1, \ldots, t_n\rangle[\Gamma]$ let the i^{th} *projection* $\pi_i(t^n)$ be the atomic lambda term $t_i[\Gamma_i]$ where $[\Gamma_i]$ is obtained by removing the binders from $[\Gamma]$ binding in any t_j $(i \neq j)$, and iteratively removing binders in s_k when x_k is removed from a distributor $[x_1, \ldots, x_k, \ldots, x_m \twoheadleftarrow \lambda y.\langle s_1, \ldots, s_k, \ldots, s_m\rangle[\Gamma']]$.

The following basic facts then characterise ub.

Proposition 6.

$$\mathsf{ub}(x) = x \qquad \mathsf{ub}(\lambda x.t) = \lambda x.\mathsf{ub}(t)[\vec{x} \leftarrow x] \qquad \mathsf{ub}((u)v) = (\mathsf{ub}(u))\mathsf{ub}(v)$$

$$\mathsf{ub}(u[x_1, \ldots, x_n \leftarrow t]) = \mathsf{ub}(u)\{\mathsf{ub}(t)^i/x_i\}_{i \leq n}$$

$$\mathsf{ub}(u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n]) = \mathsf{ub}(u)\{\mathsf{ub}(\lambda y.\pi_i(t^n))^i/x_i\}_{i \leq n}$$

Proposition 7. *For* $t = \langle t_1, \ldots, t_n\rangle[\Gamma]$, $\mathsf{ub}(\pi_i(t)) = \mathsf{ub}(t_i[\leftarrow x_1]\ldots[\leftarrow x_m][\Gamma])$ *where* $x_1, \ldots, x_m$ *are the free variables of all* t_j $(i \neq j)$*.*

To characterise the effects of duplication on the free variables of a term t or an abstracted n-term $\lambda y.t^n$, let $\mathsf{FV}(t) = \mathsf{FV}(\lambda y.t^n) = \{y_1, \ldots, y_k\}$, and let $\mathsf{FV}(\mathsf{ub}(t)^i) = \mathsf{FV}(\mathsf{ub}(\lambda y.\pi_i(t^n))^i) = \{\vec{y}_1^{\,i}, \ldots, \vec{y}_k^{\,i}\}$. Define the *renamings* of t and $\lambda y.t^n$ to be the sharings $[\vec{y}_i^{\,1}, \ldots, \vec{y}_i^{\,n} \leftarrow y_i]_{i\leq k}$, denoted $[\mathsf{rn}(t) : 1, \ldots, n]$ and $[\mathsf{rn}(\lambda y.t^n) : 1, \ldots, n]$ and abbreviated $[\mathsf{rn}(t)]$ and $[\mathsf{rn}(\lambda y.t^n)]$ where possible. The unfolded body and the renamings give the following key decomposition properties of the computational content of closures.

Lemma 8. $u[x_1, \ldots, x_n \leftarrow t] \rightsquigarrow u\{\mathsf{ub}(t)^i/x_i\}_{i\leq n}[\mathsf{rn}(t)]$

Lemma 9. $u[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n] \rightsquigarrow u\{\mathsf{ub}(\lambda y.\pi_i(t^n))^i/x_i\}_{i\leq n}[\mathsf{rn}(\lambda y.t^n)]$

3 Typed atomic lambda-calculus

The simply typed atomic lambda-calculus S_a is defined by the following rules (see [13, 12]). Terms, including variables, are typed $t : A$ with A a *minimal* formula, one built over $\rightarrow$, while n-terms are typed by *conjunctive* formulae, $t^n : A_1 \wedge \cdots \wedge A_n$. With the notation t^* indicating either a term or an n-term, a *judgment* is of the form $x_1 : A_1, \ldots, x_n : A_n \vdash t^* : B$, where $x_1, \ldots, x_n$ are the free variables of t^*. The antecedent $x_1 : A_1, \ldots, x_n : A_n$ of a judgement is treated as a set, denoted Γ, Δ, and abbreviated $(x_i : A_i)_{i\leq n}$, or $\vec{x} : A$ if $A_i = A$ for all i.

Typing rules of S_a**:**

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

$$\frac{}{x : A \vdash x : A}\,\mathsf{ax} \qquad \frac{\Gamma_1 \vdash t_1 : A_1 \qquad \cdots \qquad \Gamma_n \vdash t_n : A_n}{\Gamma_1, \ldots, \Gamma_n \vdash \langle t_1, \ldots, t_n\rangle : A_1 \wedge \cdots \wedge A_n}\,\langle\,\rangle_n$$

$$\frac{\Gamma, (x_i : B)_{i\leq n} \vdash t^* : A \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash t^*[x_1, \ldots, x_n \leftarrow u] : A}\leftarrow$$

$$\frac{\Gamma, (x_i : A \rightarrow B_i)_{i\leq n} \vdash s^* : C \qquad \Delta, y : A \vdash t^n : B_1 \wedge \cdots \wedge B_n}{\Gamma, \Delta \vdash s^*[x_1, \ldots, x_n \twoheadleftarrow \lambda y.t^n] : C}\twoheadleftarrow$$

The type system S_a of the atomic lambda-calculus is a refinement of the simply typed lambda-calculus S: the rules ax, λ, and @ are the rules of S restricted by the linearity condition. The rule for sharing, $\leftarrow$, is a standard cut-rule combined with contraction on the left. Similarly, the rule for the distributor, $\twoheadleftarrow$, is a cut-rule, albeit a highly non-standard one. It contracts on A, the antecedent of the implications, but not on their consequent, integrating a limited amount of deepness.

The typed atomic lambda-calculus S_a enjoys the usual properties of typed systems, in particular subject reduction.

Theorem 10 ([13]). *If $\Gamma \vdash u : A$ and $u \rightsquigarrow v$, then $\Gamma \vdash v : A$.*

Moreover, types are preserved in the interpretation of standard lambda-terms as atomic lambda-terms, by inserting sharing-inferences ($\leftarrow$) where required.

Proposition 11. *If $\Gamma \vdash N : A$, then $\Gamma \vdash (\!|N|\!) : A$*

Despite the fact that sharing reductions are strongly normalising, commute with denotation, and preserve typing, preservation of strong normalisation (PSN) is not immediate since infinite reduction may take place within weakenings: consider the denotation $[\![x[\leftarrow t]]\!] = x$ where t is not SN.

4 Proof of Strong Normalisation for Simple Types

In this section we prove the strong normalisation theorem for atomic lambda-terms, typed in the system S_a, using Tait's reducibility method. The proof of the main proposition (Proposition 18) relies on closure properties of the reducibility sets (Lemma 17), which again relies on closure properties on the set of strongly normalisable atomic lambda terms proved in Section 5.

For simplifying the presentation, we consider in the remainder of this paper, that no beta-reduction happens inside n-tuples, that n-tuples are unfolded, and that all their free variables are captured by closures. This property is preserved by reduction and it is natural in the context of sharing calculus. In particular all the useful computation strategies satisfy it, including the one reproducing fully lazy sharing.

Definition 12. The *value* $|A|$ of a formula A is defined inductively by:

$$\begin{array}{ll} |X| & = \mathcal{N} \\ |A \to B| & = \{u \mid u \text{ is a term and, for each term } v \in |A|,\ (u)v \in |B| \ \} \end{array}$$

Values are extended to conjunctive formulae by the following clauses, for $n > 0$. We denote by $\mathcal{V}$ the set of variables, and note that $\langle\rangle$ is the empty tuple.

$$\begin{array}{ll} |A_1 \wedge \ldots \wedge A_n| & = \{t^n \mid \text{for each } i \leq n,\ \pi_i(t) \in |A_i|\} \\ |\top| & = \{\langle\rangle[\Gamma] \mid \text{for any } x \in \mathcal{V},\ x[\Gamma] \in \mathcal{N}\} \end{array}$$

Values of formulae are called *reducibility sets*. Note that if $t \in |A|$ and t' is a variant of t, then $t' \in |A|$.

Proposition 13. *For each minimal formula A, $\mathcal{V} \subseteq |A| \subseteq \mathcal{N}$*

Proposition 14. *For any formulae $A_1, \ldots, A_n$, $|A_1 \wedge \ldots \wedge A_n| \subseteq \mathcal{N}$.*

Lemma 15. *For any formula A, if $u \in |A|$ and $u \rightsquigarrow v$, then $v \in |A|$.*

Proof. Immediate by induction on A. $\square$

Let $\{\cdot\}\overline{w}$ denote a term context consisting of repeated applications, so that $\{u\}\overline{w}$ is $(\ldots(u)w_1\ldots)w_n$.

Lemma 16. *For any formula B, if $u \in |B|$ then $u[\vec{x} \leftarrow y] \in |B|$.*

Proof. By induction on B, if $(u)\overline{w} \in |B|$ then $(u[\vec{x} \leftarrow y])\overline{w} \in |B|$. $\square$

Lemma 17.

(i) *If $u\{v/x\} \in |B|$ and $x \in \mathsf{FV}(u)$, then $(\lambda x.u)v \in |B|$.*

(ii) *If $u\{\mathsf{ub}(t)^i/x_i\}_{i\leq n}[\mathsf{rn}(t)] \in |B|$ and $t \in \mathcal{N}$, then $u[x_1,\ldots,x_n \leftarrow t] \in |B|$.*

(iii) *If $u\{\mathsf{ub}(\lambda y.\pi_i(t^n))^i/x_i\}_{i\leq n}[\mathsf{rn}(\lambda y.t^n)] \in |B|$ and $t^n \in \mathcal{N}$, then $u[x_1,\ldots,x_n \twoheadleftarrow \lambda y.t^n] \in |B|$.*

Proof. Each case is proved by induction on B, using context $\{\cdot\}\overline{w}$.

(i) If B is a variable, $|B| = \mathcal{N}$ and the result is given by Lemma 21. Otherwise, let $B = C \to D$. Suppose $(u\{v/x\})\overline{w} \in |C \to D|$ and $x \in \mathsf{FV}(u)$. Let $t \in |C|$. We prove that $(((\lambda x.u)v)\overline{w})t \in |D|$. Because $(u\{v/x\})\overline{w} \in |C \to D|$ and $t \in |C|$, we have $((u\{v/x\})\overline{w})t \in |D|$ and by the induction hypothesis, $(((\lambda x.u)v)\overline{w})t \in |D|$. It follows that $((\lambda x.u)v)\overline{w} \in |C \to D|$.

(ii) If B is a variable, $|B| = \mathcal{N}$ and the result is given by Lemma 24. Otherwise, let $B = C \to D$. Let $u' = (u\{\mathsf{ub}(t)^i/x_i\}_{i\leq n}[\mathsf{rn}(t)])\overline{w} \in |C \to D|$ and $t \in \mathcal{N}$. Let $v \in |C|$. We prove that $((u[x_1,\ldots,x_n \leftarrow t])\overline{w})v \in |D|$. By the definition of $|C \to D|$ we have $(u')v \in |D|$. By the induction hypothesis $((u[x_1,\ldots,x_n \leftarrow t])\overline{w})v \in |D|$. It follows that $(u[x_1,\ldots,x_n \leftarrow t])\overline{w} \in |C \to D|$.

(iii) The proof is similar to that of (ii). $\square$

Proposition 18. *If $(x_i : A_i)_{i\leq n} \vdash u : B$ and $v_i \in |A_i|$, then $u\{v_i/x_i\}_{i\leq n} \in |B|$.*

Proof. We proceed by induction on the derivation of $(x_i : A_i)_{i\leq n} \vdash u : B$.

1. The last rule is ax, with conclusion $x : A \vdash x : A$. For $v \in |A|$ we have $x\{v/x\} = v \in |A|$.

2. The last rule is

$$\frac{(x_i : C_i)_{i\leq n} \vdash t : A \to B \qquad (y_j : D_j)_{j\leq m} \vdash u : A}{(x_i : C_i)_{i\leq n}, (y_j : D_j)_{j\leq m} \vdash (t)u : B}\,@$$

Let $v_i \in |C_i|$ and $w_j \in |D_j|$ for $i \leq n$ and $j \leq m$. By the induction hypothesis, $t\{v_i/x_i\}_{i\leq n} \in |A \to B|$ and $u\{w_j/y_j\}_{j\leq m} \in |A|$. By the definition of $|-|$,

$$((t)u)\{v_i/x_i\}_{i\leq n}\{w_j/y_j\}_{j\leq m} = (t\{v_i/x_i\}_{i\leq n})u\{w_j/y_j\}_{j\leq m} \in |B| \ .$$

3. The last rule is
$$\frac{(x_i : C_i)_{i\leq n}, x : A \vdash t : B}{(x_i : C_i)_{i\leq n} \vdash \lambda x.t : A \to B}\,\lambda$$
Let $v_i \in |C_i|$ for $i \leq n$, and suppose $w \in |A|$. By the induction hypothesis we have $t\{v_i/x_i\}_{i\leq n}\{w/x\} \in |B|$. By Lemma 17, $(\lambda x.t\{v_i/x_i\}_{i\leq n})w \in |B|$. It follows that
$$(\lambda x.t)\{v_i/x_i\}_{i\leq n} = \lambda x.t\{v_i/x_i\}_{i\leq n} \in |A \to B|\ .$$

4. The last rule is
$$\frac{(y_i : C_i)_{i\leq k}, (x_i : B)_{i\leq n} \vdash u : A \qquad (z_i : D_i)_{i\leq m} \vdash t : B}{(y_i : C_i)_{i\leq k}, (z_i : D_i)_{i\leq m} \vdash u[x_1, \dots, x_n \leftarrow t] : A}\leftarrow$$
Let $v_i \in |C_i|$ and $w_j \in |D_j|$ for $i \leq k$ and $j \leq m$, and let $u' = u\{v_i/y_i\}_{i\leq k}$ and $t' = t\{w_j/z_j\}_{j\leq m}$. We have to prove that the following term is in $|A|$:
$$(u[x_1, \dots, x_n \leftarrow t])\{v_i/y_i\}_{i\leq k}\{w_j/z_j\}_{j\leq m} = u'[x_1, \dots, x_n \leftarrow t']\ .$$
By the induction hypothesis, $t' \in |B|$; then by Lemma 15 also the unfolded body $\mathsf{ub}(t')$ is in $|B|$. Let $\mathsf{ub}(t')^1, \dots, \mathsf{ub}(t')^n$ be fresh variants. By the induction hypothesis, $u'\{\mathsf{ub}(t')^i/x_i\}_{i\leq n} \in |A|$, and $u'\{\mathsf{ub}(t')^i/x_i\}_{i\leq n}[\mathsf{rn}(t')] \in |A|$ by Lemma 16. It follows by Lemma 17 that $u'[x_1, \dots, x_n \leftarrow t'] \in |A|$.

5. The last rule is
$$\frac{(y_i : C_i)_{i\leq k}, (x_i : A \to B)_{i\leq n} \vdash u : C \qquad (z_i : D_i)_{i\leq m}, y : A \vdash t^n : B \wedge \dots \wedge B}{(y_i : C_i)_{i\leq k}, (z_i : D_i)_{i\leq m} \vdash u[x_1, \dots, x_n \twoheadleftarrow \lambda y.t^n] : C}\twoheadleftarrow$$
Let $v_i \in |C_i|$ and $w_j \in |D_j|$ for $i \leq k$ and $j \leq m$, and let $u' = u\{v_i/y_i\}_{i\leq k}$ and $t' = t^n\{w_j/z_j\}_{j\leq m}$. We have to prove that the following term is in $|C|$:
$$(u[x_1, \dots, x_n \twoheadleftarrow \lambda y.t^n])\{v_i/y_i\}_{i\leq k}\{w_j/z_j\}_{j\leq m} = u'[x_1, \dots, x_n \twoheadleftarrow \lambda y.t']\ .$$
By the induction hypothesis, $t'\{s/y\} \in |B \wedge \dots \wedge B|$ for each $s \in |A|$, and therefore $\pi_i(t'\{s/y\}) = \pi_i(t')\{s/y\} \in |B|$. Then, for each $s \in |A|$, $(\lambda y.\pi_i(t'))s \in |B|$ by Lemma 17, and by definition of $|-|$, $\lambda y.\pi_i(t') \in |A \to B|$. By Lemma 15 also the unfolding of $\lambda y.\pi_i(t')$ belongs to $|A \to B|$, as does any variant $\mathsf{ub}(\lambda y.\pi_i(t'))^i$. By the induction hypothesis, $u'\{\mathsf{ub}(\lambda y.\pi_i(t'))^i/x_i\}_{i\leq n}$ is in $|C|$, and by Lemma 16, $u'\{\mathsf{ub}(\lambda y.\pi_i(t'))^i/x_i\}_{i\leq n}[\mathsf{rn}(\lambda y.t^n)] \in |C|$. It follows by Lemma 17 that $u'[x_1, \dots, x_n \twoheadleftarrow \lambda y.t'] \in |C|$.

□

Theorem 19. *If* $(x_i : A_i)_{i\leq n} \vdash u : B$ *then* $u \in \mathcal{N}$.

Proof. Suppose $(x_i : A_i)_{i\leq n} \vdash u : B$. By Proposition 13, we have $x_i \in |A_i|$ for $i \leq n$. Therefore by Proposition 18, $u\{x_i/x_i\}_{i\leq n} \in |B|$, i.e. $u \in |B|$, and by Proposition 13 we have $u \in \mathcal{N}$. □

5 Closure properties of Strongly Normalisable Atomic Lambda Terms

In this section we prove closure properties for the set of strongly normalisable atomic lambda terms which are used in Section 4. To strengthen the induction hypothesis in several lemmata, we define a further context, $\{\cdot\}\overline{w[\Delta]}$, which is given by the following grammar.

$$* \quad := \quad \{\cdot\} \quad | \quad (*)u \quad | \quad *[\delta]$$

The terms within a context $\{\cdot\}\overline{w[\Delta]}$ are denoted $\overline{w} = w_1, \dots, w_n$, and the sharings are denoted $\overline{[\Delta]} = [\delta_1], \dots, [\delta_m]$.

Lemma 20. *If $x \in \mathcal{V}$ and in $\{\cdot\}\overline{w}$ each $w_i \in \mathcal{N}$, then $(x)\overline{w} \in \mathcal{N}$.*

For each term $t \in \mathcal{N}$, we denote by $\mathcal{R}(t)$ the sum of the number of reduction steps in all reduction sequences of t to its normal form. For any term t, we denote by $\mathcal{S}(t)$ the number of sharing-reduction steps in all reduction paths to $\mathsf{u}(t)$.

Lemma 21. *If $(u\{v/x\})\overline{w} \in \mathcal{N}$ and $x \in \mathsf{FV}(u)$, then $((\lambda x.u)v)\overline{w} \in \mathcal{N}$.*

Proof. To obtain a suitable induction hypothesis, the context $\{\cdot\}\overline{w}$ is strengthened to $\{\cdot\}\overline{w[\Delta]}$, and further closures are inserted. It will be shown by induction on $(\mathcal{R}(T'), \mathcal{S}(T))$ that if $T' \in \mathcal{N}$ then $T \in \mathcal{N}$, where

$$T = (((\lambda x.u)[\Gamma])v)\overline{w[\Delta]} \qquad T' = (u\{v/x\}[\Gamma])\overline{w[\Delta]}\ .$$

It will be shown that for any term U reached by a reduction step $T \rightsquigarrow^1 U$, there is a term U' reached by a reduction $T' \rightsquigarrow U'$, such that the induction hypothesis applies to U and U' and $(\mathcal{R}(U'), \mathcal{S}(U)) < (\mathcal{R}(T'), \mathcal{S}(T))$, giving $U \in \mathcal{N}$. Since this holds for any term U, it follows that $T \in \mathcal{N}$.

The first, special case, is $T' = U$ (with $[\Gamma]$ empty), for which $U \in \mathcal{N}$ is immediate. For the remaining cases, we have to verify that U and U' have the right form, that the measure decreases and that $T' \rightsquigarrow U'$, which implies that $U' \in \mathcal{N}$. In the following cases, $\mathcal{R}(U') < \mathcal{R}(T')$.

1. If $T \rightsquigarrow^1 U$ is due to $u \rightsquigarrow^1 u'$, then U and U' are as follows.

$$U = (((\lambda x.u')[\Gamma])v)\overline{w[\Delta]} \qquad U' = (u'\{v/x\}[\Gamma])\overline{w[\Delta]}$$

2. If $T \rightsquigarrow^1 U$ is due to $v \rightsquigarrow^1 v'$, then U and U' are as follows.

$$U = (((\lambda x.u)[\Gamma])v')\overline{w[\Delta]} \qquad U' = (u\{v'/x\}[\Gamma])\overline{w[\Delta]}$$

3. If $T \rightsquigarrow^1 U$ is due to a rewrite step entirely inside $[\Gamma]$ or inside $\overline{w[\Delta]}$—which covers any rule except (8) and (10)—then U and U' are as follows.

$$U = (((\lambda x.u)[\Gamma'])v)\overline{w'[\Delta']} \qquad U' = (u\{v/x\}[\Gamma'])\overline{w'[\Delta']}$$

4. If $T \rightsquigarrow^1 U$ is due to an application of rule (8) or (10) to $[\Gamma]$ with subsitutions in u, then U and U' are as follows.

$$U = (((\lambda x.u')[\Gamma'])v)\overline{w[\Delta]} \qquad U' = (u'\{v/x\}[\Gamma'])\overline{w[\Delta]}$$

5. If $T \rightsquigarrow^1 U$ is due to an application of rule (8) or (10) to $\overline{w[\Delta]}$ with subsitutions anywhere in $((\lambda x.u)[\Gamma])v$, then U and U' are as follows.

$$U = (((\lambda x.u')[\Gamma'])v')\overline{w'[\Delta']} \qquad U' = (u'\{v'/x\}[\Gamma'])\overline{w'[\Delta']}$$

For the remaining cases, $\mathcal{R}(U') \leq \mathcal{R}(T')$ and $\mathcal{S}(U) < \mathcal{S}(T)$.

6. If $T \rightsquigarrow^1 U$ is an application of permutation rule (2) to $[\gamma]$ in $\lambda x.u'[\gamma]$, where $u = u'[\gamma]$, then U and U' are as follows (note that $T' = U'$).

$$U = (((\lambda x.u')[\gamma][\Gamma])v)\overline{w[\Delta]} \qquad U' = (u'\{v/x\}[\gamma][\Gamma])\overline{w[\Delta]}$$

7. If $T \rightsquigarrow^1 U$ is an application of permutation rule (3) to $[\gamma]$ in $((\lambda x.u)[\Gamma'][\gamma])v$, where $[\Gamma] = [\Gamma'][\gamma]$, then U and U' are as follows (note that $T' = U'$).

$$U = ((((\lambda x.u)[\Gamma'])v)[\gamma])\overline{w[\Delta]} \qquad U' = (u\{v/x\}[\Gamma'][\gamma])\overline{w[\Delta]}$$

8. If $T \rightsquigarrow^1 U$ is an application of permutation rule (4) to $[\gamma]$ in $((\lambda x.u)[\Gamma])v'[\gamma]$, where $v = v'[\gamma]$, then U and U' are as below. Note that $T' \rightsquigarrow U'$ by permuting $[\gamma]$ outward, from $u\{v'[\gamma]/x\}$ to $u\{v'/x\}[\gamma]$, and $T' = U'$ if $u = x$.

$$U = ((((\lambda x.u)[\Gamma])v')[\gamma])\overline{w[\Delta]} \qquad U' = (u\{v'/x\}[\Gamma][\gamma])\overline{w[\Delta]}$$

□

For the following proofs, we associate with each closure $[\gamma]$ its *body* $\mathsf{b}[\gamma]$ and its *computation* $[\gamma]^{\mathsf{c}}$, defined as follows.

$$\begin{aligned} \mathsf{b}[\vec{x} \leftarrow t] &= t \\ \mathsf{b}[\vec{x} \twoheadleftarrow \lambda y.t^n] &= \lambda y.t^n \\ [x_1, \dots, x_n \leftarrow t]^{\mathsf{c}} &= \{\mathsf{ub}(t)^i/x_i\}_{i \leq n}[\mathsf{rn}(t)] \\ [x_1, \dots, x_n \twoheadleftarrow \lambda y.t^n]^{\mathsf{c}} &= \{\mathsf{ub}(\lambda y.\pi_i(t^n))^i/x_i\}_{i \leq n}[\mathsf{rn}(\lambda y.t^n)] \end{aligned}$$

Lemma 22.
1. *If z is free in t, then $u[\vec{x} \leftarrow t]^{\mathsf{c}}\{w/z\} \rightsquigarrow u[\vec{x} \leftarrow t\{w/z\}]^{\mathsf{c}}$.*
2. *If z is free in t^n, then $u[\vec{x} \twoheadleftarrow \lambda y.t^n]^{\mathsf{c}}\{w/z\} \rightsquigarrow u[\vec{x} \twoheadleftarrow \lambda y.t\{w/z\}]^{\mathsf{c}}$.*

Proof. Immediate from the definitions, Lemma 8, and Lemma 9. □

The notation $[\gamma]^*$ will indicate a either $[\gamma]$ or $[\gamma]^{\mathsf{c}}$. For a sequence of closures $[\Gamma] = [\gamma_1]\underline{\dots}[\gamma_p]$, we denote by $[\Gamma]^*$ a *partial computation* $[\gamma_1]^*\underline{\dots}[\gamma_p]^*$. Analogously, $\{\cdot\}\overline{w[\Delta]^*}$ denotes a partial computation for a context $\{\cdot\}\overline{w[\Delta]}$.

In order to measure the number of reduction steps in a context $\overline{w[\Delta]}$, we use the notion of *applicative n-term*, defined by the following grammar.

$$T^n \quad := \quad \langle t_1, \ldots, t_n \rangle \quad | \quad T^n[\gamma] \quad | \quad (T^n)t$$

Rewrite rules apply to applicative n-terms as normal, but reduction within the tuple is permitted. Then for a term $(u)\overline{w[\Delta]}$, reduction in the context $\overline{w[\Delta]}$ is separated from that in u by considering reduction in the applicative n-term $\langle x_1, \ldots, x_n \rangle \overline{w[\Delta]}$, where $\{x_1, \ldots, x_n\} = \mathsf{FV}(u)$.

Lemma 23. *For any terms t, v, and w, if $t \rightsquigarrow^1 v$ and $t \rightsquigarrow_S w$ then $w \rightsquigarrow \mathsf{u}(v)$.*

Proof. There are two cases.

1. If $t \rightsquigarrow^1_S v$, then $\mathsf{u}(v) = \mathsf{u}(w)$, as sharing reduction is confluent and strongly normalising by Theorem 4.
2. If $t \rightsquigarrow^1_\beta v$, by [12, Lemma 17 and Theorem 18] the unfolding of w beta-reduces (in zero or more beta-steps) to a term w' such that $\mathsf{u}(w') = \mathsf{u}(v)$.

□

Lemma 24. *If $(u[\gamma]^{\mathsf{c}})\overline{w} \in \mathcal{N}$ and $\mathsf{b}[\gamma] \in \mathcal{N}$, then $(u[\gamma])\overline{w} \in \mathcal{N}$.*

Proof. The following stronger statement will be proved: given

$$T = (u)\overline{w[\Delta]} \qquad \text{and} \qquad T' = (u)\overline{w[\Delta]^*} \ ,$$

let T^n be the applicative n-term $\langle x_1, \ldots, x_n \rangle \overline{w[\Delta]}$ where $\mathsf{FV}(u) = \{\vec{x}\}$. If $T' \in \mathcal{N}$ and $T^n \in \mathcal{N}$, then $T \in \mathcal{N}$.

We proceed by induction on the measure $(\mathcal{R}(T'), \mathcal{R}(T^n))$. For each term U reached by a reduction step $T \rightsquigarrow^1 U$ it will be shown that $U \in \mathcal{N}$, proving that $T \in \mathcal{N}$. This will be done by giving a term U' reachable by a reduction $T' \rightsquigarrow U'$, to which the induction hypothesis applies; note that since $T' \in \mathcal{N}$ also $U' \in \mathcal{N}$, but it must also be shown that the corresponding applicative n-term U^n is in $\mathcal{N}$. The induction hypothesis for U and U' then gives $U \in \mathcal{N}$.

1. If the reduction step $T \rightsquigarrow^1 U$ takes place inside u, then U and U' are as follows.
$$U = (u')\overline{w[\Delta]} \qquad U' = (u')\overline{w[\Delta]^*}$$
Then $\mathcal{R}(U') < \mathcal{R}(T')$, and since $\mathsf{FV}(u) = \mathsf{FV}(u')$ we have $U^n = T^n \in \mathcal{N}$.
2. If the reduction step $T \rightsquigarrow^1 U$ takes place inside the context $\overline{w[\Delta]}$, then $\mathcal{R}(U^n) < \mathcal{R}(T^n)$. Let U and U' be
$$U = (u')\overline{w'[\Delta']} \qquad U' = (u')\overline{w'[\Delta']^*}$$
where every closure in $\overline{w'[\Delta']^*}$ is computed. The reduction $T \rightsquigarrow^1 U \rightsquigarrow_S U'$ corresponds 1-1 to a reduction from $T^n = \langle x_1, \ldots, x_n \rangle \overline{w[\Delta]}$ of the form
$$T^n \rightsquigarrow^1 V \rightsquigarrow_S \mathsf{u}(V) \ .$$

(Given that only unfolded terms are instantiated into the n-tuple in the reduction $V \rightsquigarrow_S \mathsf{u}(V)$, which holds due to the restriction on tuples instated in the beginning of Section 4.) Similarly, for the reduction $T \rightsquigarrow_S T'$ there is a corresponding $T^n \rightsquigarrow_S W$. For these reduction paths, Lemma 23 gives a reduction $W \rightsquigarrow_S \mathsf{u}(V)$. The corresponding reduction path $T' \rightsquigarrow U'$ gives $\mathcal{R}(U') \leq \mathcal{R}(T')$, so that the induction hypothesis applies.

3. If the reduction step $T \rightsquigarrow^1 U$ is a beta-step where $u = \lambda x.u'$ is the function, and the argument v is the first element of the context $\{\cdot\}\overline{w[\Delta]}$, then U and U' are as follows.

$$U = (u'\{v/x\})\overline{w'[\Delta]} \qquad U' = (u'\{v/x\})\overline{w'[\Delta]^*}$$

Here, $\overline{w'[\Delta]}$ is $\overline{w[\Delta]}$ with the first application v removed; it follows that $U^n = \langle \vec{x}, \vec{y} \rangle \overline{w'[\Delta]} \in \mathcal{N}$ because $T^n = (\langle \vec{x} \rangle v)\overline{w'[\Delta]} \in \mathcal{N}$, where $\vec{x}$ and $\vec{y}$ are the free variables of $\lambda x.u'$ and v respectively. The induction hypothesis applies since $\mathcal{R}(U') < \mathcal{R}(T')$.

4. Let the reduction step $T \rightsquigarrow^1 U$ be an application of rule (7), combining two sharings $[\gamma] = [\vec{y} \leftarrow y]$ and $[\delta] = [\vec{x}, y, \vec{z} \leftarrow t]$ into one $[\delta'] = [\vec{x}, \vec{y}, \vec{z} \leftarrow t]$, where $u = u'[\gamma]$ and $[\delta]$ is the first element of the context $\overline{w[\Delta]}$. Then U and U' are as follows.

$$U = u'[\delta']\overline{w[\Delta]} \qquad U' = u'[\delta']^*\overline{w[\Delta]^*}$$

Then $T' = u'[\gamma][\delta]^*\overline{w[\Delta]^*} \rightsquigarrow U'$, and hence $\mathcal{R}(U') < \mathcal{R}(T')$. The difference between T^n and U^n is that between the following n-terms.

$$\langle \vec{x}, y, \vec{z} \rangle [\vec{x}, y, \vec{z} \leftarrow t] \qquad \langle \vec{x}, \vec{y}, \vec{z} \rangle [\vec{x}, \vec{y}, \vec{z} \leftarrow t]$$

While t may be duplicated more times in U^n than in T^n, since no interaction is possible between the elements of a tuple it follows that $U^n \in \mathcal{N}$, so that the induction hypothesis applies.

5. Finally, there is one case where $u = u'[\gamma]$ and a reduction step forces the closure $[\gamma]$ into the context $\overline{w[\Delta]}$. Since the context $\overline{w[\Delta]}$ consists of closures and applications; moving $[\gamma]$ into it means it must be permuted past a closure $[\delta]$ or an application $(\cdot)v$. In the former case, $u'[\gamma][\delta] \sim u'[\delta][\gamma]$ is an equivalence, not a rewrite step; thus the reduction step must be an application of rewrite rule (3). But because of the congruence $\sim$ on terms, the application $(\cdot)v$ need not be the first element of $\overline{w[\Delta]}$: there may be closures $[\Gamma]$ such that $u[\gamma][\Gamma] \sim u[\Gamma][\gamma]$. Then consider the following rewrite step.

$$(u'[\gamma][\Gamma])v \sim (u'[\Gamma][\gamma])v \rightsquigarrow^1 ((u'[\Gamma])v)[\gamma]$$

Then T, T', U and U' are as follows.

$$T = ((u'[\gamma][\Gamma])v)\overline{w'[\Delta']} \qquad T' = ((u'[\gamma][\Gamma]^*)v)\overline{w'[\Delta']^*}$$

$$U = ((u'[\Gamma])v)[\gamma]\overline{w'[\Delta']} \qquad U' = ((u'[\Gamma]^*)v)[\gamma]\overline{w'[\Delta']^*}$$

Here, the context $\{\cdot\}\overline{w[\Delta]} = ((\{\cdot\}[\Gamma])v)\overline{w'[\Delta']}$. Since $T' \rightsquigarrow^1 U'$ we have that $\mathcal{R}(T') < \mathcal{R}(U')$. To apply the induction hypothesis to U and U', due to the presence of $[\Gamma]^*$ we are forced to include $[\gamma]$ into the context $\overline{w[\Delta]}$. It must then be shown that the n-term U^n is in $\mathcal{N}$, given that $T^n \in \mathcal{N}$; however, U^n includes $[\gamma]$ where T^n does not:

$$T^n = ((\langle x_1, \ldots, x_n, y_1, \ldots, y_m\rangle[\Gamma])v)\overline{w'[\Delta']}$$

$$U^n = ((\langle x_1, \ldots, x_n, z_1, \ldots, z_k\rangle[\Gamma])v)[\gamma]\overline{w'[\Delta']}\ .$$

Here, $\mathsf{FV}(\mathsf{b}[\gamma]) = \{y_1, \ldots, y_m\}$ and $\mathsf{FV}(u') = \{x_1, \ldots, x_n, z_1, \ldots, z_k\}$, with the z_i bound by $[\gamma]$.

In case $\overline{w'[\Delta']}$ does not bind in $[\gamma]$, it follows that $U^n \in \mathcal{N}$ because $\mathsf{b}[\gamma] \in \mathcal{N}$ (as it is a subterm of $T' \in \mathcal{N}$) and $T^n \in \mathcal{N}$.

Otherwise, let $\overline{w'[\Delta']}$ bind in $[\gamma]$. The n-term B' below is obtained from T' by replacing $u = u'[\gamma]$ by the tuple $\langle x_1, \ldots, x_n, \mathsf{b}[\gamma]\rangle$.

$$B' = ((\langle x_1, \ldots, x_n, \mathsf{b}[\gamma]\rangle[\Gamma]^*)v)\overline{w'[\Delta']^*} \in \mathcal{N}$$

Recall that the x_i are the free variables of u' not bound by $[\gamma]$; then each element of the tuple is a subterm of u. Then since $T' \in \mathcal{N}$, also $B' \in \mathcal{N}$, and since $\overline{w'[\Delta']}$ binds in $[\gamma]$, the computations or closures in $\overline{w'[\Delta']^*}$ binding in $\mathsf{b}[\gamma]$ create reductions in $u[\gamma]$ that have no counterpart in $\langle x_1, \ldots, x_n, \mathsf{b}[\gamma]\rangle$, so that $\mathcal{R}(B') < \mathcal{R}(T')$. Then the induction hypothesis can be applied for B' and the term B below, with $B^n = T^n \in \mathcal{N}$, giving $B \in \mathcal{N}$.

$$B = ((\langle x_1, \ldots, x_n, \mathsf{b}[\gamma]\rangle[\Gamma])v)\overline{w'[\Delta']}$$

From this it follows that $U^n \in \mathcal{N}$, by the following argument. Let $B^1, \ldots, B^m$ be variants of B. Then a reduction step in U^n must do one of three things:
(a) if it duplicates a part of $\mathsf{b}[\gamma]$ from $[\gamma]$ into a y_i, it is a sharing step, of which there are only finitely many until a step of kind (b) or (c) is performed,
(b) if it applies to an x_i or outside the tuple, there is a corresponding step in each B^j,
(c) if it applies to a (part of) $\mathsf{b}[\gamma]^i$ that has been duplicated into the tuple, there is a corresponding step in B^i.

□

6 Conclusions and further work

The present result, of strong normalisation for the simply typed atomic lambda-calculus, emphasises how the calculus is a natural and well-behaved formalisation of sharing in the lambda-calculus. Future investigations will expand in three directions: strengthening the current strong normalisation result; adapting the atomic lambda-calculus to address further notions of sharing; and investigating the practical use of the calculus in computation, for instance in compiling or implementing functional programming languages.

The present work strongly suggests two angles for future research. A natural extension would be to characterise the strongly normalisable atomic lambda-terms by an intersection typing discipline [7, 19, 16], to which the current reducibility proof is expected to extend naturally. In a second direction, it is expected that the type system and strong normalisation proof can be extended to the second-order case—although subject reduction is not immediately obvious.

For the atomic lambda-calculus in general, further work will focus on variations on the calculus that more closely approach the reduction dynamics of sharing graphs, to encompass further degrees of sharing. Another direction would be the inclusion of general recursion in the calculus, and the investigation of its interaction with the sharing constructs, as a prerequisite of making the calculus useful in practice to the implementation of functional programming languages.

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 structural lambda-calculus. In *CSL*, 2010.
3. Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *POPL*, 1995.
4. Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge University Press, 1998.
5. Thibaut Balabonski. A unified approach to fully lazy sharing. In *POPL*, 2012.
6. Kai Brünnler and Alwen Tiu. A local system for classical logic. In *LPAR*, volume 2250 of *LNCS*, pages 347–361, 2001.
7. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.
8. Rene David and Bruno Guillaume. A λ-calculus with explicit weakening and explicit substitution. *MSCS*, 11(1):169–206, 2001.
9. Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *MSCS*, 2003.
10. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.
11. Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In *RTA*, pages 135–150, 2010.
12. Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda-calculus: a typed lambda-calculus with explicit sharing. In *LICS*, 2013.
13. Tom Gundersen, Willem Heijltjes, and Michel Parigot. Un lambda-calcul atomique. In *Journées Francophones des Langages Applicatifs*, 2013.
14. 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.

15. Delia Kesner and Stéphane Lengrand. Resource operators for lambda-calculus. *Information and Computation*, 205(4):419–473, 2007.
16. Jean-Louis Krivine. *Lambda-calculus types and models*. Ellis Horwood, Chichester, UK, 1993.
17. John Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.
18. P. Lescanne. From lambda-sigma to lambda-upsilon, a journey through calculi of explicit substitutions. In *POPL*, 1994.
19. G. Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, pages 561–577. Academic Press, London, 1980.
20. W.W. Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.
21. Vincent van Oostrom, Kees-Jan van de Looij, and Marijn Zwitserlood. Lambdascope: another optimal implementation of the lambda-calculus. In *Workshop on Algebra and Logic on Programming Systems*, 2004.
22. Christopher Peter Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.