The Spinal Atomic λ-Calculus

Willem Heijltjes[1] and David R. Sherratt[1,2]

[1] University of Bath, England
[2] Friedrich-Schiller University Jena, Germany
`w.b.heijltjes@bath.ac.uk` `david.rhys.sherratt@uni-jena.de`

We investigate the computational meaning of the following *switch* rule of intuitionistic deep-inference proof theory [11, 6].

$$\frac{(A \to B) \wedge C}{A \to (B \wedge C)}\, s$$

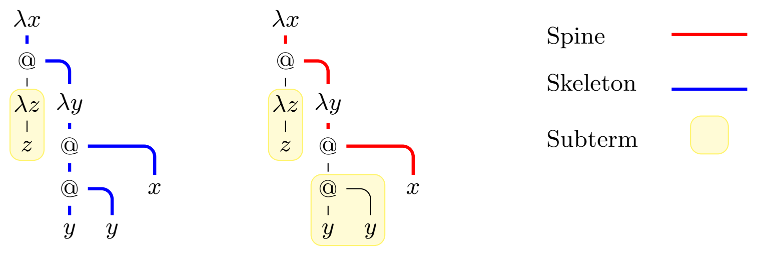
On its own, it corresponds to an *end-of-scope* marker in λ-calculus. This is a special annotation of a subterm, to indicate that a given variable does not occur free, so that a substitution on that variable can be aborted early. In the above rule, A corresponds to the variable and C to the subterm where it doesn't occur, while B represents those subterms where it does occur.

The main thrust of our work is to incorporate this rule, and its computational interpretation as a term construct, into the *atomic λ-calculus* [8]. This calculus results from an investigation of the following *medial* rule:

$$\frac{(A \vee B) \to (C \wedge D)}{(A \to C) \wedge (B \to D)}\, m$$

The medial rule enables duplication to proceed *atomically*: on individual constructors (abstraction and application) rather than entire subterms. The atomic lambda-calculus implements *full laziness*, a standard notion of sharing where only the *skeleton* of a term needs to be duplicated. Given a term t which needs to be duplicated, full laziness allows to share all maximal subterms $u_1, \ldots, u_k$ of t that do not contain occurrences of a variable bound in t outside u_i. The constructors in t not in u_i are part of the skeleton.

Our investigation is then focused on the interaction of switch and medial. Based on this we develop the *spinal atomic λ-calculus*, a natural evolution of the atomic λ-calculus. The new calculus improves on full laziness with *spinal full laziness*. The spine of an abstraction are the direct paths from the binder to bound variables [1]. The graph below provides an example of this for the term $\lambda x.(\lambda z.z)\lambda y.(yy)x$, where the spine of λx is the very thick red line and the largest subterms that could be identied by an end-of-scope operator in the term calculus or the switch rule in the typing system are enclosed by yellow boxes.



1 The λ-Calculus with Scope

We present now the typing system for a simpler variant of our calculus, and later will expand on this. The typing system is provided in *open deduction* [7], a formalism of deep inference. We now give a more formal definition of a derivation in open deduction, with a premise A and conclusion C. We work modulo symmetry, associativity, and unit laws of conjunction. A derivation is constructed with the following syntax.

$$\begin{array}{c} A \\ \Downarrow \\ C \end{array} \quad ::= \quad A \quad \Big| \quad \begin{array}{c} A_1 \\ \Downarrow \\ C_1 \end{array} \wedge \begin{array}{c} A_2 \\ \Downarrow \\ C_2 \end{array} \quad \Big| \quad \begin{array}{c} A_1 \\ \Uparrow \\ C_1 \end{array} \to \begin{array}{c} A_2 \\ \Downarrow \\ C_2 \end{array} \quad \Big| \quad \begin{array}{c} A \\ \Downarrow \\ \dfrac{B_1}{B_2}\,r \\ \Downarrow \\ C \end{array}$$

where from left to right, (1) the premise and the conclusion can be the same formula i.e. $A = C$. (2) We can compose derivations horizontally with a conjunction $\wedge$, where $A = A_1 \wedge A_2$ and $C = C_1 \wedge C_2$. (3) We can compose derivations horizontally with an implication $\to$ where $A = A_1 \to A_2$ and $C = C_1 \to C_2$. Note that the derivation on the antecedent of the implication is inverted, it can be interpreted as a derivation where we treat the premise as the conclusion and the conclusion as the premise. Lastly (4) derivations can be composed vertically with an inference rule r from B_1 to B_2.

For intuition, take de Bruijn indices [5], where variables are represented by natural numbers that denote the number of binders in scope between the variable and its corresponding binder. Following the idea of [3], instead of the natural numbers we use '0' for zero and 'S' the successor operator, then each S would correspond to an operator expressing the end of the scope for an abstraction. For example, in the term $\lambda.S0$, the variable 0 is free since the binding effect of λ is countered by the S. By doing this we make alpha conversion during substitution redundant [2], while preserving the semantics with respect to the λ-calculus [9]. This notion of scope is also used in [12] as part of an optimal implementation in the sense of Lévy [10].

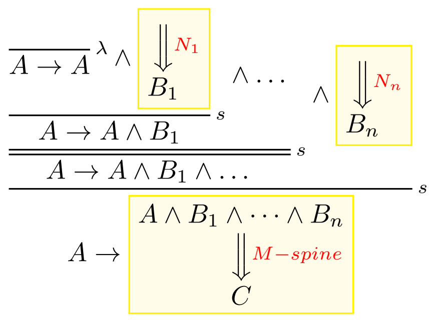
Using open deduction we can type these terms with the following inference rules, *abstraction*, *application*, *switch*, and *n-ary contraction*. A *typing judgement* $M : C$ will express that M is typable by a derivation with conclusion C.

$$\frac{}{A \to A}\,\lambda \qquad \frac{(A \to B) \wedge A}{B}\,@ \qquad \frac{(A \to B) \wedge C}{A \to (B \wedge C)}\,s \qquad \frac{A}{A \wedge \cdots \wedge A}\,\triangle$$

This deductive system provides the typing system for the calculus previously described, which has the following grammar.

$$M, N \quad ::= \quad 0 \quad | \quad \lambda M \quad | \quad (M)N \quad | \quad S(M)$$

The four constructors are *variable*, *abstraction*, *application*, and *successor*. We can type a variable with a formula, and application can be typed directly with the application rule. We look at typing abstraction and the successor function together. Consider a term λM, with n subterms $S(N_1), \ldots, S(N_n)$. This abstraction can then be typed with the following derivation, where the derivation of M is the composition of the derivations of each subterm N_i as well as the derivation of the spine of M.



Note that since there is explicit substitution in the calculus, the contraction rule has no equivalent term constructor. It is debatable where to put it in a derivation, but basically the two options are:

1. Directly below each abstraction rule. Then to correspond to an S-constructor, a switch must operate on all formulas corresponding to the same variable.

2. Directly above an application rule. Then a switch must operate on a single formula in the conjunct/context. This is arguably closer to the spirit of the term calculus.

2 Phantom Abstractions

For our calculus, we consider item 1. We consider proofs such that each abstraction has at most one corresponding switch rule, and that all the subterms are brought 'into scope' together.

Additionally, we remind the reader here that this calculus will atomically duplicate terms. This means that an abstraction λx in a term $\lambda x.t$ is duplicated independently of the body t. In our calculus, duplicating an abstraction creates *phantom-abstractions*. These are partially duplicated abstractions, i.e. the abstraction λx has been duplicated but not the bound variable x in t. The phantom-abstraction, $c\langle\, x_1, \dots, x_n \,\rangle.t$, has three components. The *phantom-variable*, c, is the name of the phantom-abstraction, and when the phantom-abstraction is *exorcised* (converted back into an abstraction) this is the variable that it will bind. The *cover*, $x_1, \dots, x_n$, are the variables that are located in the body t that will be replaced with terms that might contain the phantom-variable. Intuitively, a phantom-abstraction can be thought of as an abstraction with 'holes' that can be filled with terms, and the abstraction may capture any variables free in those terms. Phantom-abstractions are created when we duplicate a shared abstraction, and the body is populated as we proceed with duplication on the next constructors in the body. Consider the following grammar for terms

$$s, t \quad ::= \quad x \quad | \quad (s)t \quad | \quad c\langle\, x_1, \dots, x_n \,\rangle.t \quad | \quad t[x_1, \dots, x_n \leftarrow s]$$

We have the constructors *variable*, *application*, *abstraction*, and *sharing*. The spinal atomic λ-calculus is linear, meaning that each variable can only occur at most once. An abstraction is a phantom-abstraction if the cover is not the phantom-variable. A phantom-abstraction is 'exorcised' by creating a sharing construct that shared the phantom-variable to capture the variables in the cover, and replacing the variables in the cover with the phantom-variable.

$$c\langle\, x_1, \dots, x_n \,\rangle.t \quad \Rightarrow_e \quad c\langle\, c \,\rangle.t[x_1, \dots, x_n \leftarrow c]$$

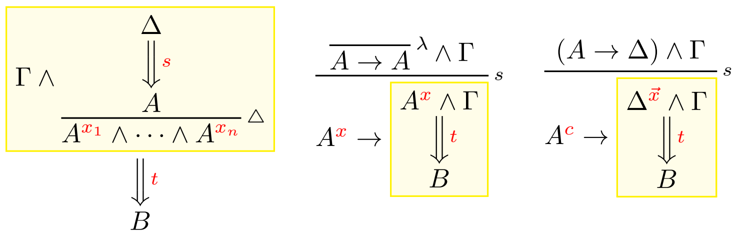
A phantom-abstraction may need some *book-keeping*, i.e. updating the variables in the cover. This happens when substituting is replacing one of the variables in the cover found free in the body. We would then replace the variable in the cover with the free variables of the term being substituted in. Substitution is denoted by $s\{t/x\}$, where we replace x for t in s. Then this book-keeping is shown below, where $z_1, \dots, z_m$ are the free variables in s.

$$(c\langle\, x_1, \dots, x_n, y \,\rangle.t)\{s/y\} \quad \Rightarrow_{bk} \quad c\langle\, x_1, \dots, x_n, z_1, \dots, z_m \,\rangle.t\{s/y\}$$

Beta reduction then only occurs with abstractions, and not phantom-abstractions. A phantom-abstraction can be part of a redex only after is has been exorcised.

$$(x\langle\, x \,\rangle.t)s \rightsquigarrow_\beta t\{s/x\}$$

Typing these terms in open deduction is similar to as described before. Sharing is the constructor that corresponds with the contraction rule. An abstraction, $x\langle\, x \,\rangle.t$ is typed as a combination of the abstraction rule and the switch rule, where the variables brought into scope with the switch rule are the types of the maximal subterms of t that do not contain x as a free variable. A phantom-abstraction, $c\langle\, \vec{x} \,\rangle.t$, is typed similarly, but without the abstraction rule. The formulas not brought into scope with the switch are the types of the variables in the cover. In the derivations below, Γ and Δ are *conjunctive formula*, i.e. multiple (possibly zero) formulas connected by $\wedge$.

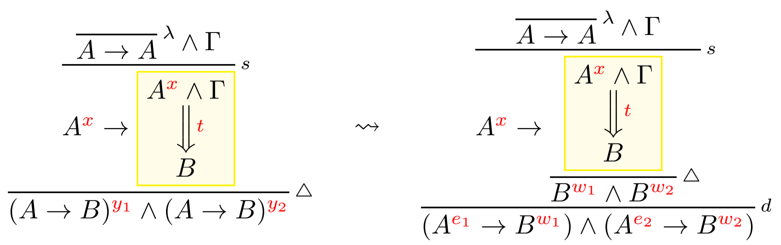


3 Distributor

Phantom-abstractions are partially duplicated abstractions. Here we remind the reader of the result of [8], which observes the computational interpretation of the medial rule. In classical logic, the medial rule is the key to reducing contractions to their atomic case during proof normalisation [4]. This result also translates into intuitionistic logic, and is the key factor to allowing atomic reduction. The medial rule studied in [8] uses disjunction, and to avoid introducing disjunction into the typing system the authors instead use the following *distribution* rule, which combines the medial with a *co-contraction* rule (the reversed contraction rule that works with disjunction rather than conjunction).

$$\frac{A \to (B_1 \wedge \cdots \wedge B_n)}{(A \to B_1) \wedge \cdots \wedge (A \to B_n)}\, d$$

The distribution rule is introduced during proof normalisation when we wish to duplicate a (phantom-)abstraction. When the conclusion of an abstraction meets a contraction rule, we introduce the distribution rule which creates n phantom-abstractions that represent the copies of the duplicated abstraction. Below is the proof rewriting rule, showing the introduction of the distributor. On the right hand side, we introduce the distirbutor and two new phantom-abstractions, $e_1\langle\, w_1 \,\rangle.w_1$ and $e_2\langle\, w_2 \,\rangle.w_2$. The derivations of the phantom-abstractions do not have a switch rule because there are no variables in the body that are not mentioned in the cover.



In the term calculus, the *distributor*, $u[e_1\langle \vec{w_1}\rangle,\dots,e_n\langle \vec{w_n}\rangle \mid x\langle x\rangle\, \overline{[\Gamma]}]$, is the computational interpretation of the distribution rule, where $\overline{[\Gamma]}$ are a bundle of sharings and nested distributors which we call an *environment*. The distributor captures the phantom-variables e_i in u, and the covers associated with those phantom-variables are captured by the environment $\overline{[\Gamma]}$. The generalised rewrite rule for the mentioned proof rewrite step above.

$$\begin{aligned}&u[y_1,\dots,y_n \leftarrow x\langle x\rangle.t] \rightsquigarrow_D \\ &\quad u\{e_1\langle w_1\rangle.w_1/y_1\}\dots\{e_n\langle w_n\rangle.w_n/y_n\}[e_1\langle w_1\rangle,\dots,e_n\langle w_n\rangle \mid x\langle x\rangle\,[w_1,\dots,w_n \leftarrow t]]\end{aligned} \qquad (d_2)$$

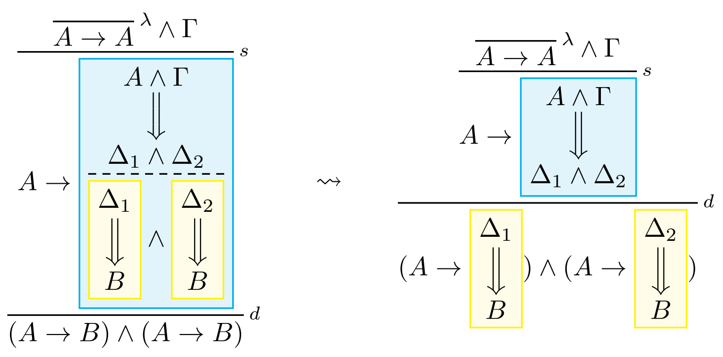
Duplication would then proceed up the derivation of t (a.k.a. on the term t). Duplicating an application is the same rewrite rule as in [8]. The term rewrite rule is shown below.

$$u[x_1\dots x_n \leftarrow s\,t] \rightsquigarrow_D u\{z_1\,y_1/x_1\}\dots\{z_n\,y_n/x_n\}[z_1\dots z_n \leftarrow s][y_1\dots y_n \leftarrow t] \qquad (d_1)$$

As dupliucation proceeds in the distributor, it produces more terms to be substituted into u, specifically into the bodies of the phantom-abstractions bound by the distributor. Substitutions occur when reducing the environment in the distributor. As a result, the substitution would need to 'escape' the distributor and proceed on the context of the distirbutor.

$$u[\overrightarrow{e\langle \vec{w}\rangle} \mid z\langle z\rangle\,\{t/x\}\overline{[\Gamma]}] \Rightarrow u\{t/x\}[\overrightarrow{e\langle \vec{w}\rangle} \mid z\langle z\rangle\,\overline{[\Gamma]}]$$

Book-keeping becomes essential here to maintain the correctness of the term calculus. Book-keeping in the proof theory is known as *flushing*. As the contraction rule scales up the derivation, it gradually creates multiple copies of derivations that sit under the contraction rule and above the distribution rule. We 'flush' these derivations under the distribution rule and into context. This corresponds to substituting into the body of the phantom-abstraction. The covers of these phantom-abstractions are then maintained such that the variables listed do occur within the body, and are captured by the environment of the corresponding distributor.



In the above derivations, the derivations in the yellow boxes are ‘flushed’ underneath the distributor. The variables in the cover of the phantom-abstractions are then updated to be variables corresponding to Δ_1 (and Δ_2). This corresponds to the book-keeping done by substitutions, i.e. updating the covers of the corresponding phantom-abstractions.

When duplicating the spine, we may need to lift a *closures* (a sharing or a distributor) out of the distributor. These correspond to lifting the maximal subexpressions that do not contain the bound variable. In general, we use the following rewrite rule for lifting where $[\Gamma]$ is a closure. In the case of (l_3), we must check that the variables in the cover (regardless of whether the abstraction is a phantom or not) do not occur in the closure in order to lift the closure.

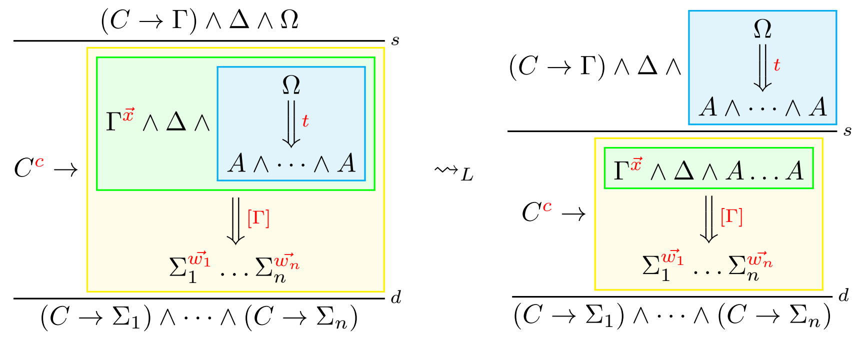
$$s[\Gamma]\,t \rightsquigarrow_L (s\,t)[\Gamma] \tag{l_1}$$

$$s\,t[\Gamma] \rightsquigarrow_L (s\,t)[\Gamma] \tag{l_2}$$

$$\begin{gathered} d\langle\,\vec{x}\,\rangle.t[\Gamma] \rightsquigarrow_L (d\langle\,\vec{x}\,\rangle.t)[\Gamma] \\ \text{iff } x \in \vec{x} \rightarrow x \in (t)_{fv} \text{ and } d \notin ([\Gamma])_{fv} \end{gathered} \tag{l_3}$$

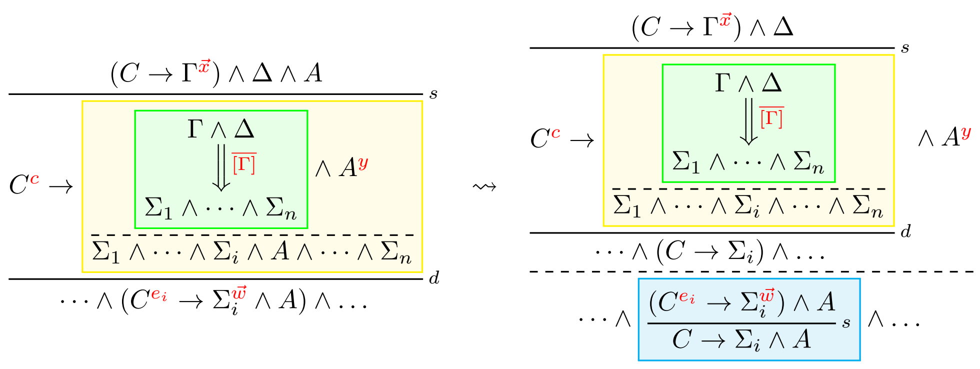
$$u[x_1,\dots,x_n \leftarrow t[\Gamma]] \rightsquigarrow_L u[x_1,\dots,x_n \leftarrow t][\Gamma] \tag{l_4}$$

The remaining case to discuss is lifting a closure outside of a distributor. We discuss here to lifting the a sharing, but the lifting of a nested distributor is defined similarly. We lift by combining two rewrite rules. The first is the actual lifting of the closure, and the second is generating book-keeping steps. Similarly to (l_3), we may only lift the closure iff the variables in the cover of the distributor do not occur freely inside the closure.


$(C \to \Gamma) \wedge \Delta \wedge \Omega$
s
$C^c \to$
$\Gamma^{\vec{x}} \wedge \Delta \wedge$
Ω
t
$A \wedge \cdots \wedge A$
$[\Gamma]$
$\Sigma_1^{\vec{w_1}} \dots \Sigma_n^{\vec{w_n}}$
d
$(C \to \Sigma_1) \wedge \cdots \wedge (C \to \Sigma_n)$
$\rightsquigarrow_L$
$(C \to \Gamma) \wedge \Delta \wedge$
Ω
t
$A \wedge \cdots \wedge A$
s
$C^c \to$
$\Gamma^{\vec{x}} \wedge \Delta \wedge A \dots A$
$[\Gamma]$
$\Sigma_1^{\vec{w_1}} \dots \Sigma_n^{\vec{w_n}}$
d
$(C \to \Sigma_1) \wedge \cdots \wedge (C \to \Sigma_n)$

$$\begin{gathered} u[e_1\langle\,\vec{w_1}\,\rangle \dots e_n\langle\,\vec{w_n}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\vec{y} \leftarrow t]] \rightsquigarrow_L \\ u[e_1\langle\,\vec{w_1}\,\rangle \dots e_n\langle\,\vec{w_n}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\vec{y} \leftarrow t] \\ \text{iff } x \in \vec{x} \rightarrow x \notin (t)_{fv} \end{gathered} \tag{$l_{5.1}$}$$

In the example, each A corresponds to a variable in $\vec{y}$. If the A is in the cover of the phantom-abstraction, we can remove it since we now know that the shared term that we lifted does not contain the binding variable, i.e. it is not part of the spine of the abstraction. In this case, after lifting we introduce the switch rule for each phantom-abstraction. This allows to remove each A from the scope of the distributor while maintaining the bodies of the phantom-abstractions, by keeping A within the scope of these abstractions.


$(C \to \Gamma^{\vec{x}}) \wedge \Delta \wedge A$
s
$C^c \to$
$\Gamma \wedge \Delta$
$\overline{[\Gamma]}$
$\Sigma_1 \wedge \cdots \wedge \Sigma_n$
$\wedge A^y$
$\Sigma_1 \wedge \cdots \wedge \Sigma_i \wedge A \wedge \cdots \wedge \Sigma_n$
d
$\cdots \wedge (C^{e_i} \to \Sigma_i^{\vec{w}} \wedge A) \wedge \ldots$
$\rightsquigarrow$
$(C \to \Gamma^{\vec{x}}) \wedge \Delta$
s
$C^c \to$
$\Gamma \wedge \Delta$
$\overline{[\Gamma]}$
$\Sigma_1 \wedge \cdots \wedge \Sigma_n$
$\wedge A^y$
$\Sigma_1 \wedge \cdots \wedge \Sigma_i \wedge \cdots \wedge \Sigma_n$
d
$\cdots \wedge (C \to \Sigma_i) \wedge \ldots$
$\cdots \wedge$
$(C^{e_i} \to \Sigma_i^{\vec{w}}) \wedge A$
$C \to \Sigma_i \wedge A$
s
$\wedge \ldots$

We can update the covers in the term calculus explicitly with an explicit book-keeping operation $u\{\vec{w}/e\}_b$. This will replace the cover of the phantom-abstraction with phantom-variable e with the variables $\vec{w}$ in the term u.

$$\begin{gathered} u[e\{e_i\langle\, \vec{w} \cdot y \,\rangle\} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_L \\ u\{\vec{w}/e_i\}_b[e\{e_i\langle\, \vec{w} \,\rangle\} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \\ \text{iff } y \in (u[e\{e_i\langle\, \vec{w} \cdot z \,\rangle\} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}])_{fv} \end{gathered} \tag{$l_{5.2}$}$$

By doing this, eliminating the distributor simply means applying creating exorcisms when the environment in the distributor is exactly one sharing that is sharing the bound variable only.

$$\begin{aligned} &u[e_1\langle\, \vec{w_1} \,\rangle \ldots e_n\langle\, \vec{w_n} \,\rangle \,|\, c\langle\, c \,\rangle\, [\vec{w_1}, \ldots, \vec{w_n} \leftarrow c]] \\ &\qquad\qquad \rightsquigarrow_D u\{e_1\langle\, \vec{w_1} \,\rangle\}_e \ldots \{e_n\langle\, \vec{w_n} \,\rangle\}_e \end{aligned} \tag{d_3}$$

In the proof theory, this corresponds to replacing the distributor with multiple abstraction rules, where the number of abstraction rules are equal to the number of phantom-abstractions.

We also include the following rules in which unary sharings are applied as substitutions, and consecutive sharings are compounded.

$$\begin{aligned} &u[w_1, \ldots, w_m \leftarrow y_i][y_1, \ldots, y_n \leftarrow t] \rightsquigarrow_C \\ &\quad u[y_1, \ldots, y_{i-1}, w_1, \ldots, w_m, y_{i+1}, \ldots, y_n \leftarrow t] \end{aligned} \tag{c_1}$$

$$u[x \leftarrow t] \rightsquigarrow_C u\{t/x\} \tag{c_2}$$

We write $\rightsquigarrow_S$ to mean any one of these reduction rules (excluding β-reduction). After observing the correspondence between proof rewritings and reductions steps in the term calculus, the following proposition becomes trivial.

Proposition 1 (Subject Reduction)**.** *If $s \rightsquigarrow_{(\beta,S)} t$ and $s : A$, then $t : A$*

4 Relationship with λ-calculus

We provide the semantics of this calculus by providing a translation into the λ-calculus. The spinal atomic λ-calculus is linear, so translating a sharing means just performing the actual substitutions replacing the variables bound by the sharing. The main difference to notice is the effects of the distributor. The distributor $[e_1\langle\, \vec{w_1} \,\rangle, \ldots, e_n\langle\, \vec{w_n} \,\rangle \,|\, x\langle\, x \,\rangle\, \overline{[\Gamma]}]$, captures n phantom-abstractions. When translating the distributor, we will collapse the environment $\overline{[\Gamma]}$, generating substitutions. After the environment is collapsed, the free variables in the bodies of these phantom-abstractions ($\vec{w_i}$) will be replaced with terms. The variable bound by the abstraction, x, might be occurring somewhere in the body after substitution. The phantom-abstraction into the λ-calculus will be interpreted as an abstraction, that binds the phantom-variable. Therefore, in order to maintain binding, we need to rename the bound variable x to e_i for each phantom-abstraction. The phantom-abstraction is a partial copy of the abstraction being duplicated in the distributor, and each copy will rename the bound variable differently.

We utilise two interpretations here. Given a term $N : \Lambda$, a substitution map $S : V \to \Lambda$, and a variable-indexed renaming $R : V \to V \to$, i.e. for a given x, $R_x : V \to V$ is a renaming function, we can define the interpretation $[\![\, t \,]\!] = (\Lambda, R)$ that interprets a spinal atomic λ terms as a pair consisting of a λ-term and renamings, and will use the auxiliary translation $\|[\Gamma]\|$ which interprets closures as a pair (S, R).

A renaming R *modifies* a substitution S by $S\{R\} : V \to \Lambda$, such that $S\{R\}(x) = S(x)\{R_x\}$. So if S has $\{N/x\}$, then $S\{R\}$ has $\{N\{R_x\}/x\}$. Renamings R and Q can be *composed* $R \cdot Q = R_x \cdot Q_x$ for every x. Renaming can be *joined* as $R + Q$ if R_x and Q_x agree where they are both defined. We thus define the translation.

$$
\begin{array}{rll}
[\![\, x \,]\!] &= (x, 0) & \\
[\![\, st \,]\!] &= (NM, R+Q) & [\![\, s \,]\!] = (N, R), [\![\, t \,]\!] = (M, Q) \\
[\![\, c\langle\, \vec{x} \,\rangle.t \,]\!] &= (\lambda c.N, R) & [\![\, t \,]\!] = (N, R) \\
[\![\, t[\Gamma] \,]\!] &= (N\{S\{R\}\}, R \cdot Q) & [\![\, t \,]\!] = (N, R), \|[\Gamma]\| = (S, Q)
\end{array}
$$

$$
\begin{array}{rll}
\|[x_1, \dots, x_n \leftarrow t]\| &= (\{N/x_1\} \dots \{N/x_n\}, R) & [\![\, t \,]\!] = (N, R) \\
\|[e_1\langle\, \vec{w_1} \,\rangle, \dots, e_n\langle\, \vec{w_n} \,\rangle \mid c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}]\| &= (S\{R\}, R \cdot Q) & \|\overline{[\Gamma]}\| = (S, Q) \\
& & \text{each } R_x = \{e_i/c\} \text{ for each } x \in \vec{w_i}
\end{array}
$$

4.1 Strong normalisation of sharing reductions

The interpretation $[\![\, - \,]\!]$ collapses shared terms by duplicating them wholly. This is what is done atomically by the $\rightsquigarrow_S$ reduction steps. The following Lemma can be proven on a case basis.

Lemma 2 (Sharing reduction preserves denotation)**.** *If $s \rightsquigarrow_S t$, then $[\![\, s \,]\!] = [\![\, t \,]\!]$.*

We can invoke a *measure* on spinal atomic terms. This measure would consist of three parts, the *weight*, the *height* and the number of closures. The height of a closure is the number of constructors you would recursively pass through until you reach the closure. The height measure is then the collection of the heights of every closure in a term. The rewrite rules (l_1) (l_2) (l_3) (l_4) ($l_{5.1}$ + $l_{5.2}$) strictly decrease the height. The rewrite rules (c_1) and (c_2) strictly decrease the number of closures.

The weight intuitively counts the number of constructors that would appear in the corresponding λ-term obtained by $[\![\, - \,]\!]$. As we duplicate, the weight would decrease. For example, (d_1) duplicates the application constructor. Before duplicating it, the constructor may have a weight of at least n (where n is the number of variables bound by the sharing). This is because $[\![\, - \,]\!]$ will produce at least n new application constructors when interpreting the sharing. After duplicating, the weight will be distributed among the n copies, therefore the weight of a copy is strictly less than the original. The reduction steps (d_1) (d_2) (d_3) strictly reduces the weight of the term. This combined with that the other rewrite rules do not change the weight of a term, is the base of the proof of the following Theorem.

Theorem 3. *Sharing reduction $\rightsquigarrow_S$ is strongly normalising*

5 Spine duplication

We lastly deifne the *spine* formally, and show that our calculus is capable of duplicating the spine and only the spine of a term. We first define the spine for the regular λ-calculus. Given a set of variables V, $\text{spine}_V(t)$ is the term such that the only free variables in both t and $\text{spine}_V(t)$ are exactly V.

$$\text{spine}_V(x) = x$$

$$\text{spine}_V(\lambda x.M) = \begin{cases} y & (M)_{fv} \cap V = \{\} \\ \lambda x.\text{spine}_V(M) & \text{otherwise} \end{cases}$$

$$\text{spine}_V(M\,N) = \begin{cases} y & (M\,N)_{fv} \cap V = \{\} \\ (\text{spine}_V(M))\,y & (N)_{fv} \cap V = \{\} \\ (y)\,\text{spine}_V(N) & (M)_{fv} \cap V = \{\} \\ (\text{spine}_V(M))\,\text{spine}_V(N) & \text{otherwise} \end{cases}$$

We now define the spine for spinal atomic λ-terms. The spine of a term will possibly contain phantom-abstractions. The variables in the cover of these phantom-abstractions will be fresh if they are not the variables in the given set V. Therefore, given a variable x, a term t and the spine of the term t', the following function will determine the variables in the spine that are in place of the subterm freely containing the variable x.

$$UBK(x\,,\,t\,,\,y) = \{y\}$$

$$UBK(x\,,\,c\langle\,\vec{y}\,\rangle.t\,,\,c\langle\,\vec{z}\,\rangle.t') = UBK(x\,,\,t\,,\,t')$$

$$UBK(x\,,\,s\,t\,,\,s'\,t') = \begin{cases} UBK(x\,,\,s\,,\,s') & x \in (s)_{fv} \\ UBK(x\,,\,t\,,\,t') & \text{otherwise} \end{cases}$$

$$UBK(x\,,\,s[z_1,\ldots,z_n \leftarrow t]\,,\,u) = \begin{cases} \bigcup\limits_{j\leq n} UBK(x_j\,,\,s\{t^i/z_i\}_{i\leq n}\,,\,u) & x \in (t)_{fv} \\ UBK(x\,,\,s\,,\,u) & \text{otherwise} \end{cases}$$

$$UBK(x\,,\,t[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\,,\,u) = UBK(x\,,\,t\overline{[\Gamma]}\,,\,u)$$

A fresh variant of a term, t, is a copy of t where all variables are renamed with fresh variables and bindings are maintained, and is denoted t^i.

Using this function, we can formally define the function that inputs a set of variables V, and a term in the spinal atomic λ-calculus, and will return the term that corresponds to the spine. It replaces the maximal subexpressions that do not contain a variable in V with a fresh, distinct variable.

$$\begin{aligned}
\text{spine}_V(x) &= x \\
\text{spine}_V(c\langle\,\vec{x}\,\rangle.t) &= y \\
&\qquad \text{if } (t)_{fv} \cap V = \{\} \\
\text{spine}_V(c\langle\,\vec{x}\,\rangle.t) &= c\langle\,\vec{y}\,\rangle.\text{spine}_V(t) \\
&\qquad \text{where } \{\vec{y}\} = \bigcup_{x\in\vec{x}} UBK(x\,,\,t\,,\,\text{spine}_V(t))
\end{aligned}$$

$$\text{spine}_V(u[\overrightarrow{e\langle\,\vec{w}\,\rangle} \mid c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]) = \text{spine}_V(u\overline{[\Gamma]})$$

$$\text{spine}_V(s\,t) = \begin{cases} y & \text{if } (s\,t)_{fv} \cap V = \{\};\ \text{otherwise} \\ (\text{spine}_V(s))\,y & \text{if } (t)_{fv} \cap V = \{\} \\ (y)\,\text{spine}_V(t) & \text{if } (s)_{fv} \cap V = \{\} \\ (\text{spine}_V(s))\,\text{spine}_V(t) & \text{otherwise} \end{cases}$$

$$\text{spine}_V(s[x_1 \dots x_n \leftarrow t]) = \begin{cases} y & \text{if } (s[x_1 \dots x_n \leftarrow t])_{fv} \cap V = \{\} \\ (\text{spine}_{V\cup U}(s\,\sigma)) & \text{otherwise} \end{cases}$$

where σ are the substitutions $\{t^i/x_i\}_{1\leq i\leq n}$
and U is such that $z \in (V \cap (t)_{fv}) \iff \forall_{1\leq i\leq n} z_i \in U$
The following proposition demonstrates that we have a good definition of spine.

Proposition 4.

$$\mathit{spine}_V(\llbracket\, t \,\rrbracket) = \llbracket\, \mathit{spine}_V(t) \,\rrbracket$$

The following Lemma is crucial to showing that our calculus can indeed duplicate the spine of an abstraction. It can be proven by induction on the weight of the term t.

Lemma 5. *Given a term t, and a set of variables V such that $v \in V \to v \notin (t)_{fc}$*

$$u[x_1, \dots, x_n \leftarrow t] \rightsquigarrow^*_S u\{\mathit{spine}_V(t)^i/x_i\}_{1\leq i\leq n}\overline{[\Gamma]}[\Delta]$$

where the environment $\overline{[\Gamma]}$ is made up of, for each $y \in (t)_{fv} \cap V$, a sharing of the form $[y_1, \dots, y_m \leftarrow y]$

Theorem 6. *The spinal atomic λ-calculus implements spine duplication: a term $u[x_1, \dots, x_n \leftarrow c\langle\,c\,\rangle.t]$ reduced to a term of the form $u\{(c\langle\,c\,\rangle.\mathit{spine}_{\{c\}}(t))^i/x_i\}_{1\leq i\leq n}[\Delta]$*

References

[1] Thibaut Balabonski. A unified approach to fully lazy sharing. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 469–480, New York, NY, USA, 2012. ACM.

[2] Klaus. J. Berkling. *A Symmetric Complement to the Lambda Calculus.* Bonn Interner Bericht ISF. Gesellschaft für Mathematik und Datenverarbeitung mbH, 1976.

[3] Richard S. Bird and Ross Paterson. de bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999.

[4] Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 2250 of *Lecture Notes in Computer Science*, pages 347–361. Springer-Verlag, 2001.

[5] Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.

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

[7] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In Christopher Lynch, editor, *21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 135–150. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[8] Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda calculus: A typed lambda-calculus with explicit sharing. In Orna Kupferman, editor, *28th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 311–320. IEEE, 2013.

[9] Dimitri Hendriks and Vincent van Oostrom. adbmal. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 136–150, 2003.

[10] Jean-Jacques Lévy. Optimal reductions in the lambda calculus. *To HB Curry: Essays on Combinatory Logic, Lambda Coalculus and Formalism*, pages 159–191, 1980.

[11] Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2):4:1–24, 2006.

[12] 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 (ALPS)*, 2004.