



HAL
open science

A strong call-by-need calculus

Thibaut Balabonski, Antoine Lanco, Guillaume Melquiond

► **To cite this version:**

Thibaut Balabonski, Antoine Lanco, Guillaume Melquiond. A strong call-by-need calculus. Proceedings of the 6th International Conference on Formal Structures for Computation and Deduction, Jul 2021, Buenos Aires, Argentina. pp.1-22, 10.4230/LIPIcs.FSCD.2021.9 . hal-03149692v1

HAL Id: hal-03149692

<https://inria.hal.science/hal-03149692v1>

Submitted on 23 Feb 2021 (v1), last revised 4 May 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A strong call-by-need calculus

Thibaut Balabonski ✉

Université Paris-Saclay, CNRS, LMF, France

Antoine Lanco ✉

Université Paris-Saclay, CNRS, Inria, LMF, France

Guillaume Melquiond ✉

Université Paris-Saclay, CNRS, Inria, LMF, France

Abstract

We present a *call-by-need* λ -calculus that enables *strong* reduction (that is, reduction inside the body of abstractions) and guarantees that arguments are only evaluated if needed and at most once. This calculus uses explicit substitutions and subsumes the existing strong-call-by-need strategy, but allows for more reduction sequences, and often shorter ones, while preserving the *neededness*.

The calculus is shown to be *normalizing* in a strong sense: Whenever a λ -term t admits a normal form n in the λ -calculus, then *any* reduction sequence from t in the calculus eventually reaches the normal form n .

We also exhibit a restriction of this calculus that has the *diamond* property and that only performs reduction sequences of minimal length, which makes it systematically better than the existing strategy.

We have used the Abella proof assistant to formalize part of this calculus, and discuss how this experiment affected its design.

2012 ACM Subject Classification Theory of computation \rightarrow Program semantics; Theory of computation \rightarrow Operational semantics

Keywords and phrases strong reduction, call-by-need, evaluation strategy, normalization

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Lambda-calculus is seen as the standard model of computation in functional programming languages, once equipped with an *evaluation strategy* [23]. The most famous evaluation strategies are *call-by-value*, that eagerly evaluates the arguments of a function before resolving the function call, *call-by-name*, where the arguments of a function are evaluated when they are needed, and *call-by-need* [25, 4], which extends call-by-name with a memoization or sharing mechanism to remember the value of an argument that has already been evaluated.

The strength of call-by-name is that it only evaluates terms whose value is effectively needed, at the (possibly huge) cost of evaluating some terms several times. Conversely, the strength *and* weakness of call-by-value (by far the most used strategy in actual programming languages) is that it evaluates each function argument exactly once, even when its value is not actually needed and when its evaluation does not terminate. At the cost of memoization, call-by-need combines the benefits of call-by-value and call-by-name, by only evaluating needed arguments and evaluating them only once.

A common point of these strategies is that they are concerned with *evaluation*, that is computing *values*. As such they operate in the subset of λ -calculus called *weak reduction*, in which there is no reduction inside λ -abstractions, the latter being already considered to be values. Some applications however, such as proof assistants or partial evaluation, require reducing inside λ -abstractions, and possibly aiming for the actual normal form of a λ -term.

The first known abstract machine computing the normal form of a term is due to Crégut [14] and implements normal order reduction. More recently, several lines of work



© Thibaut Balabonski and Antoine Lanco and Guillaume Melquiond;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

46 have transposed the known evaluation strategies to strong reduction strategies or abstract
 47 machines: call-by-value [16, 9, 2], call-by-name [1] or call-by-need [8, 10]. Some non-advertised
 48 strong extensions of call-by-name or call-by-need can also be found in the internals of proof
 49 assistants, notably Coq.

50 These strong strategies are mostly conservative over their underlying weak strategy, and
 51 often proceed by *iteratively* applying a weak strategy to open terms. As such they use a
 52 restricted form of strong reduction to enable reduction to normal form, but do not try to
 53 take advantage of strong reduction to obtain shorter reduction sequences. However, since
 54 call-by-need has been shown to capture optimal weak reduction [7], it is known that the
 55 deliberate use of strong reduction [17] is the only way of allowing shorter reduction sequences.

56 This paper presents a strong call-by-need calculus, which obeys the following guidelines.
 57 First, it reduces only needed redexes. Second, it keeps a level of sharing at least equal to
 58 that of call-by-value and call-by-need. Third, it enables strong reduction as generally as
 59 possible. This calculus builds on the syntax and a part of the meta-theory of λ -calculus with
 60 explicit substitutions, which we recall in Section 2.

61 Neededness of a redex is undecidable in general, thus the first and third guidelines are
 62 antagonist. Section 3 resolves this tension by exposing a simple syntactic criterion capturing
 63 more needed redexes than what is already used in call-by-need strategies. Through reducing
 64 only needed redexes, our calculus enjoys a normalization preservation theorem that is stronger
 65 than usual: Any λ -term that is *weakly* normalizing in the pure λ -calculus (there is at least
 66 one reduction sequence to a normal form, but some other sequences may diverge) will be
 67 *strongly* normalizing in our calculus (any reduction sequence is normalizing). This strong
 68 normalization theorem, related to the usual *completeness* results of call-by-name or call-by-
 69 need strategies, is completely dealt with using a system of non-idempotent intersection types.
 70 This allows us to avoid the traditional tedious syntactic commutation lemmas to provide
 71 more elegant proofs. Our proof improves the technique used in [19, 8].

72 While our calculus contains the strong call-by-need strategy introduced in [8], it also
 73 allows more liberal call-by-need strategies that anticipate some strong reduction steps in
 74 order to reduce the overall length of the reduction to normal form. Section 4 presents a
 75 restriction of the calculus that guarantees reduction sequences of minimal length and in
 76 particular builds shorter reduction sequences

77 Finally, Section 5 presents a formalization of parts of our results in Abella [5]. Beyond the
 78 proof safety provided by such a tool, this formalization has also influenced the design of our
 79 strong call-by-need calculus itself in a positive way. In particular, it promoted a presentation
 80 based on SOS-style local reduction rules [24], which later became a lever for a more efficient
 81 use of non-idempotent intersection types. The formalization can be found at the following
 82 address: <https://www.lri.fr/~blsk/SCBNd-FSCD2021.zip>

83 **2** The host calculus λ_c

84 Our strong call-by-need calculus is included in an already known calculus λ_c , that serves as
 85 a technical tool in [8] and which we name our *host calculus*. This calculus gives the general
 86 shape of reduction rules allowing memoization and comes with a system of non-idempotent
 87 intersection types. It includes however no notion of neededness.

88 The λ_c -calculus uses the following syntax of λ -terms with explicit substitutions, which is
 89 isomorphic to the original syntax of the call-by-need calculus using let-bindings [4].

$$90 \quad t \in \Lambda_c \quad ::= \quad x \mid \lambda x.t \mid t t \mid t[x \setminus t]$$

91 The free variables $\text{fv}(t)$ of a term t are defined as usual. We write \mathcal{C} for a context, that is a
 92 term with exactly one hole \square , and L for a context with the specific shape $\square[x_1 \setminus t_1] \dots [x_n \setminus t_n]$.
 93 We write $\mathcal{C}[t]$ for the term obtained by plugging the subterm t in the hole of the context \mathcal{C}
 94 (with possible capture of free variables of t by binders in \mathcal{C}), or tL when the context is of
 95 the specific shape L . We also write $\mathcal{C}[[t]]$ for plugging a term t whose free variables are not
 96 captured by \mathcal{C} . The *values* we consider are λ -abstractions.

97 Reduction in λ_c is defined by the following three reduction rules, applied in any context.
 98 Rather than using traditional propagation rules for explicit substitutions [18], it builds on
 99 the *Linear Substitution Calculus* [22, 3] that is more similar to the let-constructs commonly
 100 used for defining call-by-need.

$$\begin{array}{lll}
 (\lambda x.t)L u & \rightarrow_{\text{dB}} & t[x \setminus u]L \\
 \mathcal{C}[[x][x \setminus v]L] & \rightarrow_{\text{lsv}} & \mathcal{C}[[v][x \setminus v]L] \quad \text{with } v \text{ value} \\
 t[x \setminus u] & \rightarrow_{\text{gc}} & t \quad \text{with } x \notin \text{fv}(t)
 \end{array}$$

102 The rule \rightarrow_{dB} describes β -reduction “at a distance”. It applies to a β -redex whose λ -
 103 abstraction is possibly hidden by a list of explicit substitutions. This rule is a combination
 104 of a single use of β -reduction with a repeated use of the structural rule lifting the explicit
 105 substitutions at the left of an application. The rule \rightarrow_{lsv} describes the linear substitution
 106 of a value, that is the substitution of one occurrence of the variable x bound by an explicit
 107 substitution. It has to be understood as a lookup operation. Similarly to \rightarrow_{dB} this rule
 108 embeds a repeated use of a structural rule for unnesting explicit substitutions. Remark
 109 that this calculus only allows the substitution of λ -abstractions, and not of variables as
 110 is it sometimes seen [21]. This restricted behavior is enough for the main results of this
 111 paper, and will allow a more compact presentation. Finally, the rule \rightarrow_{gc} describes garbage
 112 collection of an explicit substitution for a variable that does not live anymore.

113 A term t of λ_c is related to a pure λ -term t^* by the unfolding operation which applies all
 114 the explicit substitutions. We write $t\{x \setminus u\}$ for the meta-level substitution of x by u in t .

$$\begin{array}{ll}
 x^* & = x & (t u)^* & = t^* u^* \\
 (\lambda x.t)^* & = \lambda x.(t^*) & (t\{x \setminus u\})^* & = (t^*)\{x \setminus u^*\}
 \end{array}$$

116 Through unfolding, any reduction step $t \rightarrow_c u$ in λ_c is related to a sequence of reductions
 117 $t^* \rightarrow_{\beta}^* u^*$ in the pure λ -calculus.

118 The host calculus λ_c comes with a system of non-idempotent intersection types [13, 15]
 119 originally defined in [20]. A type τ may be a type variable α or an arrow type $\mathcal{M} \rightarrow \tau$, where
 120 \mathcal{M} is a multiset $\{\{\sigma_1, \dots, \sigma_n\}\}$ of types. A typing environment Γ associates to each variable in
 121 its domain a multiset of types. This multiset contains one type for each potential use of the
 122 variable, and may be empty if the variable is not actually used. A typing judgment $\Gamma \vdash t : \tau$
 123 assigns exactly one type to the term t . As shown by the typing rules in Fig. 1, an argument of
 124 an application or of an explicit substitution may be typed several times in a derivation. This
 125 type system is known to characterize λ -terms that are weakly normalizing for β -reduction, if
 126 associated with the side condition that the empty multiset $\{\{\}\}$ does not appear at a positive
 127 position in the typing judgment. Positive type occurrences $\mathcal{T}_+(\Gamma \vdash t : \tau)$ and negative type
 128 occurrences $\mathcal{T}_-(\Gamma \vdash t : \tau)$ of a typing judgment are defined by the following equations.

$$\begin{array}{ll}
 \mathcal{T}_+(\alpha) & = \{\alpha\} & \mathcal{T}_-(\alpha) & = \emptyset \\
 \mathcal{T}_+(\mathcal{M}) & = \{\mathcal{M}\} \cup \bigcup_{\sigma \in \mathcal{M}} \mathcal{T}_+(\sigma) & \mathcal{T}_-(\mathcal{M}) & = \bigcup_{\sigma \in \mathcal{M}} \mathcal{T}_-(\sigma) \\
 \mathcal{T}_+(\mathcal{M} \rightarrow \sigma) & = \{\mathcal{M} \rightarrow \sigma\} \cup \mathcal{T}_-(\mathcal{M}) \cup \mathcal{T}_+(\sigma) & \mathcal{T}_-(\mathcal{M} \rightarrow \sigma) & = \mathcal{T}_+(\mathcal{M}) \cup \mathcal{T}_-(\sigma) \\
 \mathcal{T}_+(\Gamma \vdash t : \sigma) & = \mathcal{T}_+(\sigma) \cup \bigcup_{x \in \text{dom}(\Gamma)} \mathcal{T}_-(\Gamma(x))
 \end{array}$$

130

$$\begin{array}{c}
\frac{}{x : \{\sigma\} \vdash x : \sigma} \\
\frac{\Gamma; x : \mathcal{M} \vdash t : \tau}{\Gamma \vdash \lambda x.t : \mathcal{M} \rightarrow \tau} \\
\frac{\Gamma \vdash t : \mathcal{M} \rightarrow \tau \quad (\Delta_\sigma \vdash u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash t u : \tau} \\
\frac{\Gamma; x : \mathcal{M} \vdash t : \tau \quad (\Delta_\sigma \vdash u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash t[x \setminus u] : \tau}
\end{array}$$

■ **Figure 1** Typing rules for λ_c

131 ► **Theorem 1** (Typability [8]). *If the pure λ -term t is weakly normalizing for β -reduction,*
132 *then there is a typing judgment $\Gamma \vdash t : \tau$ such that $\{\!\!\}\notin \mathcal{T}_+(\Gamma \vdash t : \tau)$.*

133 A typing derivation Φ for a typing judgment $\Gamma \vdash t : \tau$ (written $\Phi \triangleright \Gamma \vdash t : \tau$) defines
134 in t a set of *typed positions*, which are the positions for which the derivation Φ contains a
135 typing judgment. These typed positions have an important property: They satisfy a *weighted*
136 *subject reduction theorem* ensuring a decreasing derivation size, which we will use in the
137 next section.

138 ► **Theorem 2** (Weighted subject reduction [8]). *If $\Phi \triangleright \Gamma \vdash t : \tau$ and $t \rightarrow_c t'$ by reduction of a*
139 *redex at a typed position, then there is a derivation $\Phi' \triangleright \Gamma \vdash t' : \tau$ with Φ' smaller than Φ .*

140 3 Strong call-by-need calculus λ_{sn}

141 Our strong call-by-need calculus is defined by the same terms and reduction rules as λ_c ,
142 with restrictions on where the reduction rules can be applied. These restrictions ensure in
143 particular that only needed redexes are reduced. Notice that **gc**-reduction is never needed in
144 this calculus and will thus be ignored from now on.

145 3.1 Structures and frozen variables

146 The starting point is the same as the one for the original (weak) call-by-need calculus. Since
147 the argument of a function is not always needed, we do not reduce in advance the right
148 part of an application $t u$. Instead, we first evaluate t to an answer $(\lambda x.t')L$, then apply
149 a β -reduction to put the argument u in the environment of t' , and then go on with the
150 resulting term $t'[x \setminus u]L$, evaluating u only if and when it is required.

151 The previous principle is enough for weak reduction, but new behaviors appear with
152 strong reduction. Suppose for instance that we get an abstraction $\lambda x.x t u$ at top-level.
153 Given its position, this abstraction will never be applied to an argument. This means in
154 particular that its variable x will never be substituted by anything; it is blocked and is now
155 part of the rigid structure of the term. Following [8], we call this variable x *frozen*. Now
156 regarding the arguments t and u given to the frozen variable x , these will always remain at
157 their respective positions and never get into contact with other parts of the term; they are
158 in *top-level-like* positions and can be treated as independent terms. In particular, if t is an
159 abstraction $\lambda y.t'$ then the variable y will never be substituted and can be seen as frozen.

160 We call *structure* an application $x t_1 \dots t_n$ led by a frozen variable x , possibly interlaced
161 with explicit substitutions. As implied by the last rule in Fig. 2, an explicit substitution in a
162 structure may even affect the leading variable, provided that the content of the substitution
163 is itself a structure. We write \mathcal{S}_φ the set of structures under a set φ of frozen variables. It
164 differs from the notion in [8] in that it does not require the term to be in normal form.

$$\frac{x \in \varphi}{x \in \mathcal{S}_\varphi} \quad \frac{t \in \mathcal{S}_\varphi}{t u \in \mathcal{S}_\varphi} \quad \frac{t \in \mathcal{S}_\varphi}{t[x \setminus u] \in \mathcal{S}_\varphi} \quad \frac{t \in \mathcal{S}_{\varphi \cup \{x\}} \quad u \in \mathcal{S}_\varphi}{t[x \setminus u] \in \mathcal{S}_\varphi}$$

■ **Figure 2** Structures of λ_{sn} .

165 Frozen variables in a term t are either free variables of t , or variables introduced by
 166 binders in t . As such they obey the usual renaming conventions. In particular, the third and
 167 fourth rules in Fig. 2 implicitly require that the variable x bound by the explicit substitution
 168 is *not* in the set φ . We keep this *freshness convention* in all the definitions of the paper.

169 3.2 Reduction in λ_{sn}

170 As a result, the reduction relation of our calculus has three parameters:

- 171 ■ a rule ρ , which can be **dB**, **lsv**, or others that we will introduce shortly;
- 172 ■ a set φ of frozen variables, used to define structures and top-level-like positions;
- 173 ■ a flag μ , which is \top for reduction at a top-level-like position, and \perp for reduction at an
 174 arbitrary position.

175 The rules are given in Fig. 3. Rule **@-LEFT** makes reduction always possible on the left of
 176 an application, but as shown by the premise this position is not a \top position. Rule **@-RIGHT**
 177 on the other hand shows that reduction on the right of an application is possible only in the
 178 context of a frozen structure. In this case however the position is \top .

179 Rules **λ -TOP** and **λ -BOT** make reduction always possible inside a λ -abstraction, *i.e.*,
 180 unconditional strong reduction. If the abstraction is in a \top position, its variable is added to
 181 the set of frozen variables.

182 Rules **ES-LEFT** and **ES-LEFT- φ** show that it is always possible to reduce a term affected
 183 by an explicit substitution. If the substitution contains a structure, the variable bound by
 184 the substitution can be added to the set of frozen variables.

185 Rule **ES-RIGHT** restricts reduction inside a substitution to the case where an occurrence
 186 of the substituted variable is at a position eligible for reduction. It uses an auxiliary rule **id_x**
 187 that propagates using the same inference rules as the rules **dB** and **lsv**, to probe a term for
 188 the presence of some variable x at a reduction position. By freshness, $x \notin \varphi$. This auxiliary
 189 rule does not modify the term to which it applies, as witnessed by its base case **ID**.

190 Rules **DB** and **LSV** are the base cases for applying **dB**- or **lsv**-reduction. Using the notations
 191 of λ_c , they will allow the following reductions.

$$\begin{array}{l} (\lambda x.t)L u \xrightarrow{\text{dB}, \varphi, \mu}_{\text{sn}} t[x \setminus u]L \\ \mathcal{C}[[x]][x \setminus v]L \xrightarrow{\text{lsv}, \varphi, \mu}_{\text{sn}} \mathcal{C}[[v]][x \setminus v]L \quad \text{with } v \text{ value, and } \mathcal{C} \text{ a suitable context} \end{array}$$

193 Each is defined using an auxiliary reduction relation dealing with the list L of explicit
 194 substitutions. These auxiliary reductions are given in Fig. 4.

195 Rules **DB-BASE** and **LSV-BASE** describe the base cases of the auxiliary reductions, where
 196 the list L is empty. Note that, while **DB-BASE** is an axiom, the inference rule **LSV-BASE**
 197 uses as a premise a $\frac{\rho, \varphi, \mu}{\text{sn}}$ -reduction using a new reduction rule **sub_{x \setminus v}**. This reduction rule
 198 substitutes one occurrence of the variable x at a position eligible for reduction by the value v
 199 (with, by freshness, $x \notin \varphi$). As seen for **id_x** above, this reduction rule propagates using the
 200 same inference rules as **dB** and **lsv**, and its base case is the rule **SUB** in Fig. 3. The presence

23:6 A strong call-by-need calculus

$$\begin{array}{c}
\textcircled{A}\text{-LEFT} \\
\frac{t \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} t'}{t u \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t' u} \\
\textcircled{A}\text{-RIGHT} \\
\frac{t \in \mathcal{S}_\varphi \quad u \xrightarrow{\rho, \varphi, \top}_{\text{sn}} u'}{t u \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t u'} \\
\lambda\text{-TOP} \\
\frac{t \xrightarrow{\rho, \varphi \cup \{x\}, \top}_{\text{sn}} t'}{\lambda x. t \xrightarrow{\rho, \varphi, \top}_{\text{sn}} \lambda x. t'} \\
\lambda\text{-BOT} \\
\frac{t \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} t'}{\lambda x. t \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} \lambda x. t'} \\
\text{ES-LEFT} \\
\frac{t \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'}{t[x \setminus u] \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'[x \setminus u]} \\
\text{ES-LEFT-}\varphi \\
\frac{t \xrightarrow{\rho, \varphi \cup \{x\}, \mu}_{\text{sn}} t' \quad u \in \mathcal{S}_\varphi}{t[x \setminus u] \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'[x \setminus u]} \\
\text{ES-RIGHT} \\
\frac{t \xrightarrow{\text{id}_x, \varphi, \mu}_{\text{sn}} t \quad u \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} u'}{t[x \setminus u] \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t[x \setminus u]} \\
\text{ID} \\
\frac{}{x \xrightarrow{\text{id}_x, \varphi, \mu}_{\text{sn}} x} \\
\text{SUB} \\
\frac{}{x \xrightarrow{\text{sub}_{x \setminus v}, \varphi, \mu}_{\text{sn}} v} \\
\text{dB} \\
\frac{t \rightarrow_{\text{db}} t'}{t \xrightarrow{\text{dB}, \varphi, \mu}_{\text{sn}} t'} \\
\text{LSV} \\
\frac{t \xrightarrow{\varphi, \mu}_{\text{lsv}} t'}{t \xrightarrow{\text{lsv}, \varphi, \mu}_{\text{sn}} t'}
\end{array}$$

■ **Figure 3** Reduction rules for λ_{sn} .

$$\begin{array}{c}
\text{dB-BASE} \\
\frac{}{(\lambda x. t) u \rightarrow_{\text{db}} t[x \setminus u]} \\
\text{LSV-BASE} \\
\frac{t \xrightarrow{\text{sub}_{x \setminus v}, \varphi, \mu}_{\text{sn}} t' \quad v \text{ value}}{t[x \setminus v] \xrightarrow{\varphi, \mu}_{\text{lsv}} t'[x \setminus v]} \\
\text{dB-}\sigma \\
\frac{t u \rightarrow_{\text{db}} v}{t[x \setminus w] u \rightarrow_{\text{db}} v[x \setminus w]} \\
\text{LSV-}\sigma \\
\frac{t[x \setminus u] \xrightarrow{\varphi, \mu}_{\text{lsv}} t'}{t[x \setminus u[y \setminus w]] \xrightarrow{\varphi, \mu}_{\text{lsv}} t'[y \setminus w]} \\
\text{LSV-}\sigma\text{-}\varphi \\
\frac{t[x \setminus u] \xrightarrow{\varphi \cup \{y\}, \mu}_{\text{lsv}} t' \quad w \in \mathcal{S}_\varphi}{t[x \setminus u[y \setminus w]] \xrightarrow{\varphi, \mu}_{\text{lsv}} t'[y \setminus w]}
\end{array}$$

■ **Figure 4** Auxiliary reduction rules for λ_{sn} .

201 of this premise $t \xrightarrow{\text{sub}_{x \setminus v}, \varphi, \mu}_{\text{sn}} t'$ in the rule is the primary reason why the auxiliary relation
202 $\xrightarrow{\varphi, \mu}_{\text{lsv}}$ is parameterized by φ and μ . The combination of the rules LSV and LSV-BASE makes
203 it possible, in the case of a lsv-reduction, to resume the search for a reducible variable in the
204 context in which the substitution has been found (instead of resetting the context). In [8], a
205 similar effect was achieved using a more convoluted condition on a composition of contexts.

206 Rule dB- σ makes it possible to float out an explicit substitution applied to the left part
207 of an application. That is, if a dB-reduction is possible without the substitution, then the
208 reduction is performed and the substitution is applied to the result.

209 Rules LSV- σ and LSV- σ - φ achieve the same effect with the nested substitutions applied to
210 the value substituted by an lsv-reduction step. As with rule ES-LEFT- φ , if the substitution is
211 a structure, the variable can be frozen. Note that this last feature distinguishing LSV- σ and
212 LSV- σ - φ can be ignored for now. It will prove useful in Sec. 4 only.

Here is a sample derivation using our inference rules. Its left branch checks that an occurrence of the variable x is actually at a needed position, while its right branch reduces

$$\begin{array}{c}
\frac{x \in \varphi}{x \in \mathcal{N}_\varphi} \qquad \frac{t \in \mathcal{N}_\varphi \quad t \in \mathcal{S}_\varphi \quad u \in \mathcal{N}_\varphi}{t u \in \mathcal{N}_\varphi} \qquad \frac{t \in \mathcal{N}_{\varphi \cup \{x\}}}{\lambda x.t \in \mathcal{N}_\varphi} \\
\\
\frac{t \in \mathcal{N}_{\varphi \cup \{x\}} \quad u \in \mathcal{N}_\varphi \quad u \in \mathcal{S}_\varphi}{t[x \setminus u] \in \mathcal{N}_\varphi} \qquad \frac{t \in \mathcal{N}_\varphi}{t[x \setminus u] \in \mathcal{N}_\varphi}
\end{array}$$

■ **Figure 5** Normal forms of λ_{sn} .

the argument of the substitution.

$$\begin{array}{c}
\text{ID} \\
\text{@-RIGHT} \frac{a \in \mathcal{S}_{\{a\}} \quad \frac{x \xrightarrow{\text{id}_x, \{a\}, \top}_{sn} x}}{a x \xrightarrow{\text{id}_x, \{a\}, \top}_{sn} a x}}{\lambda a.a x \xrightarrow{\text{id}_x, \emptyset, \top}_{sn} \lambda a.a x}} \quad \frac{\text{DB-BASE} \quad \frac{(\lambda y.t) v \xrightarrow{\text{db}} t[y \setminus v]}{(\lambda y.t)[z \setminus u] v \xrightarrow{\text{db}} t[y \setminus v][z \setminus u]} \quad \text{DB-}\sigma}{(\lambda y.t)[z \setminus u] v \xrightarrow{\text{dB}, \emptyset, \perp}_{sn} t[y \setminus v][z \setminus u]} \quad \text{DB}}{\text{ES-RIGHT} \quad \frac{(\lambda a.a x)[x \setminus (\lambda y.t)[z \setminus u] v] \xrightarrow{\text{dB}, \emptyset, \top}_{sn} (\lambda a.a x)[x \setminus t[y \setminus v][z \setminus u]}}
\end{array}$$

213 The actual reduction, written \rightarrow_{sn} , is either $\xrightarrow{\text{dB}, \varphi, \top}_{sn}$ or $\xrightarrow{\text{lsv}, \varphi, \top}_{sn}$, with φ typically being
214 empty when reducing closed terms, or containing the free variables otherwise.

215 ► **Lemma 3** (Simulation). *If $t \rightarrow_{sn} t'$ then $t^* \rightarrow_{\beta}^* t'^*$.*

216 **Proof.** By induction on the reduction $t \xrightarrow{\rho, \varphi, \mu}_{sn} t'$. Included in our Abella development. ◀

217 The normal forms of λ_{sn} correspond to the normal forms of the strong call-by-need
218 strategy [8]. They can be characterized by the inductive definition given in Fig. 5.

219 ► **Lemma 4.** *$t \in \mathcal{N}_\varphi$ if and only if there is no reduction $t \xrightarrow{\rho, \varphi, \mu}_{sn} t'$.*

220 **Proof.** The first part (a term cannot be in normal form and reducible) has been proved in
221 Abella, by induction on the reduction rules. The second part (any term is either a normal
222 form or a reducible term) is by induction on t (not in Abella yet). ◀

223 ► **Lemma 5.** *If $t \in \mathcal{N}_\varphi$ then t^* is a normal form in the λ -calculus.*

224 **Proof.** By induction on $t \in \mathcal{N}_\varphi$. Done in Abella. ◀

225 Finally, note that the strong call-by-need strategy introduced in [8] is included in our
226 calculus. One can recover this strategy by imposing two restrictions on $\xrightarrow{\rho, \varphi, \mu}_{sn}$:

- 227 ■ remove the rule λ -BOT, since the strategy never reduces under an abstraction that is not
- 228 yet in top-level-like position,
- 229 ■ restrict the rule @-RIGHT to the case where the left member of an application is not only
- 230 a structure, but rather a structure in normal form, since the strategy imposes left-to-right
- 231 reduction of structures.

$$\begin{array}{c}
\text{TY-VAR} \\
\hline
x : \{\!\!\{\sigma\}\!\!\} \vdash_{\varphi}^{\mu} x : \sigma
\end{array}
\qquad
\begin{array}{c}
\text{TY-}\lambda\text{-}\perp \\
\frac{\Gamma; x : \mathcal{M} \vdash_{\varphi}^{\perp} t : \tau}{\Gamma \vdash_{\varphi}^{\perp} \lambda x.t : \mathcal{M} \rightarrow \tau}
\end{array}
\qquad
\begin{array}{c}
\text{TY-}\lambda\text{-}\top \\
\frac{\Gamma; x : \mathcal{M} \vdash_{\varphi \cup \{x\}}^{\top} t : \tau}{\Gamma \vdash_{\varphi}^{\top} \lambda x.t : \mathcal{M} \rightarrow \tau}
\end{array}$$

$$\begin{array}{c}
\text{TY-}\textcircled{\text{A}} \\
\frac{\Gamma \vdash_{\varphi}^{\perp} t : \mathcal{M} \rightarrow \tau \quad (\Delta_{\sigma} \vdash_{\varphi}^{\perp} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_{\sigma} \vdash_{\varphi}^{\mu} t u : \tau}
\end{array}
\qquad
\begin{array}{c}
\text{TY-}\textcircled{\text{S}} \\
\frac{\Gamma \vdash_{\varphi}^{\perp} t : \mathcal{M} \rightarrow \tau \quad t \in \mathcal{S}_{\varphi} \quad (\Delta_{\sigma} \vdash_{\varphi}^{\top} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_{\sigma} \vdash_{\varphi}^{\mu} t u : \tau}
\end{array}$$

$$\begin{array}{c}
\text{TY-ES} \\
\frac{\Gamma; x : \mathcal{M} \vdash_{\varphi}^{\mu} t : \tau \quad (\Delta_{\sigma} \vdash_{\varphi}^{\perp} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_{\sigma} \vdash_{\varphi}^{\mu} t[x \setminus u] : \tau}
\end{array}
\qquad
\begin{array}{c}
\text{TY-ES-}\mathcal{S} \\
\frac{\Gamma; x : \mathcal{M} \vdash_{\varphi \cup \{x\}}^{\mu} t : \tau \quad u \in \mathcal{S}_{\varphi} \quad (\Delta_{\sigma} \vdash_{\varphi}^{\perp} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_{\sigma} \vdash_{\varphi}^{\mu} t[x \setminus u] : \tau}
\end{array}$$

■ **Figure 6** Annotated system for non-idempotent intersection types.

232 3.3 Completeness

233 Our strong call-by-need calculus is complete with respect to normalization in the λ -calculus
234 in a strong sense: Whenever a λ -term t admits a normal form in the pure λ -calculus, every
235 reduction path in λ_{sn} eventually reaches a representative of this normal form. This section is
236 devoted to proving this completeness result (Th. 11). The proof relies on the non-idempotent
237 intersection type system in the following way:

- 238 1. Typability (Th. 1) ensures that any weakly normalizing λ -term admits a typing derivation
239 (with no positive occurrence of $\{\!\!\{\}\!\!\}$).
- 240 2. We prove here that any λ_{sn} -reduction in a typed λ_{sn} -term t (with no positive occurrence
241 of $\{\!\!\{\}\!\!\}$) is at a typed position of t (Th. 10).
- 242 3. Then weighted subject reduction (Th. 2) provides a decreasing measure for λ_{sn} -reduction.
243 Finally, the obtained normal form is related to the β -normal form using Lemmas 3, 4, and 5.

244 The proof of the forthcoming typed reduction (Th. 10) uses a refinement of the non-
245 idempotent intersection types system of λ_c , given in Fig. 6. Both systems derive the same
246 typing judgments with the same typed positions. The refined system however features an
247 annotated typing judgment $\Gamma \vdash_{\varphi}^{\mu} t : \tau$ embedding additional context information, such as a
248 set φ of frozen variables and a marker μ of top-level-like positions. These annotations are
249 faithful counterparts to the corresponding annotations of λ_{sn} reduction rules.

250 In particular, the rule for typing an abstraction is split in two versions $\text{TY-}\lambda\text{-}\perp$ and
251 $\text{TY-}\lambda\text{-}\top$, the latter being applicable to \top positions and thus freezing the variable bound
252 by the abstraction (in both rules, by freshness convention we assume $x \notin \varphi$). The rule for
253 typing an application is also split in two version: $\text{TY-}\textcircled{\text{S}}$ is applicable when the left part
254 of the application is a structure and marks the right part as a \top position, while $\text{TY-}\textcircled{\text{A}}$ is
255 applicable otherwise. Finally, the rule for typing an explicit substitution is similarly split in
256 two versions, depending on whether the contents of the substitution is a structure or not,
257 and handling the set of frozen variables accordingly. We write $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t : \tau$ if there is a
258 derivation Φ of the annotated typing judgment $\Gamma \vdash_{\varphi}^{\mu} t : \tau$.

259 ► **Lemma 6** (Typing derivation annotation). *If there is a derivation $\Phi \triangleright \Gamma \vdash t : \tau$, then for
260 any φ and μ there is a derivation $\Phi' \triangleright \Gamma \vdash_{\varphi}^{\mu} t : \tau$ such that the sets of typed positions in Φ
261 and Φ' are equal.*

262 **Proof.** By induction on Φ , since annotations do not interfere with typing. ◀

263 The converse property is also true, by erasing of the annotations, but is not used in the proof
264 of the completeness result.

265 The most crucial part of the proof is ensuring that any argument of a typed structure is
266 itself at a typed position. This follows from the following three lemmas.

267 ► **Lemma 7** (Typed structure). *If $\Gamma \vdash_{\varphi}^{\mu} t : \tau$ and $t \in \mathcal{S}_{\varphi}$, then there is $x \in \varphi$ such that*
268 *$\tau \in \mathcal{T}_{+}(\Gamma(x))$.*

269 **Proof.** By induction on the structure of t .¹ The most interesting case is the one of an explicit
270 substitution $t_1[x \setminus t_2]$. The induction hypothesis applied on t_1 can give the variable x which
271 does not appear in the conclusion, but in that case t_2 is guaranteed to be a structure whose
272 type contains τ . ◀

273 Given a derivation Φ , we write $\text{fzt}(\Phi)$ the set of types associated to frozen variables in
274 judgments of the derivation Φ .

275 ► **Lemma 8** (Subformula property).

- 276 1. *If $\Phi \triangleright \Gamma \vdash_{\varphi}^{\top} t : \tau$ then*
$$\begin{cases} \mathcal{T}_{+}(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_{+}(\Gamma(x)) \cup \mathcal{T}_{-}(\tau) \\ \mathcal{T}_{-}(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_{-}(\Gamma(x)) \cup \mathcal{T}_{+}(\tau) \end{cases}$$
- 277 2. *If $\Phi \triangleright \Gamma \vdash_{\varphi}^{\perp} t : \tau$ then*
$$\begin{cases} \mathcal{T}_{+}(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_{+}(\Gamma(x)) \\ \mathcal{T}_{-}(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_{-}(\Gamma(x)) \end{cases}$$

278 **Proof.** By mutual induction on the typing derivations.¹ Most cases are fairly straightforward.
279 The only difficult case comes from the rule TY-@-S , in which there is a premise $\Delta \vdash_{\varphi}^{\top} u : \sigma$
280 with mode \top but with a type σ that does not clearly appear in the conclusion. Here we need
281 the typed structure (Lem. 7) to conclude. ◀

282 ► **Lemma 9** (Typed structure argument). *If $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t : \tau$ with $\{\!\!\}\notin \mathcal{T}_{+}(\Gamma \vdash t : \tau)$, then*
283 *every typing judgment of the shape $\Gamma' \vdash_{\varphi'}^{\mu'} s : \mathcal{M} \rightarrow \sigma$ in Φ with $s \in \mathcal{S}_{\varphi'}$ satisfies $\mathcal{M} \neq \{\!\!\}$.*

284 **Proof.** Let $\Gamma' \vdash_{\varphi'}^{\mu'} s : \mathcal{M} \rightarrow \sigma$ in Φ with $s \in \mathcal{S}_{\varphi'}$. By Lemma 7 there is $x \in \varphi'$ such
285 that $\mathcal{M} \rightarrow \sigma \in \mathcal{T}_{+}(\Gamma'(x))$. Then $\mathcal{M} \in \mathcal{T}_{-}(\Gamma'(x))$ and $\mathcal{M} \in \mathcal{T}_{-}(\text{fzt}(\Phi))$. By Lemma 8
286 $\mathcal{M} \in \mathcal{T}_{+}(\Gamma \vdash_{\varphi}^{\mu} t : \tau)$, thus $\mathcal{M} \neq \{\!\!\}$. ◀

287 The main theorem on typed reduction is then as follows.

288 ► **Theorem 10** (Typed reduction). *If $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t : \tau$ with $\{\!\!\}\notin \mathcal{T}_{+}(\Gamma \vdash t : \tau)$, then every*
289 *λ_{sn} -reduction $t \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'$ is at a typed position.*

290 **Proof.** We prove by induction on $t \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'$ that, if $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t : \tau$ with Φ such that any
291 typing judgment $\Gamma' \vdash_{\varphi'}^{\mu'} s : \mathcal{M} \rightarrow \sigma$ in Φ with $s \in \mathcal{S}_{\varphi'}$ satisfies $\mathcal{M} \neq \{\!\!\}$, then $t \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t'$
292 reduces at a typed position (the restriction on Φ is enabled by Lemma 9). Since all other
293 reduction cases concern positions that are systematically typed, we focus here on @-RIGHT
294 and ES-RIGHT .

295 ■ Case @-RIGHT : $t u \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t u'$ with $t \in \mathcal{S}_{\varphi}$ and $u \xrightarrow{\rho, \varphi, \top}_{\text{sn}} u'$, assuming $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t u : \sigma$.
296 By inversion of the last rule in Φ we know there is a subderivation $\Phi' \triangleright \Gamma' \vdash_{\varphi'}^{\perp} t : \mathcal{M} \rightarrow \sigma$
297 and by hypothesis $\mathcal{M} \neq \{\!\!\}$. Then u is typed in Φ and we can conclude by induction
298 hypothesis.

¹ See appendix for the complete proof.

299 ■ Case ES-RIGHT: $t[x \setminus u] \xrightarrow{\rho, \varphi, \mu}_{\text{sn}} t[x \setminus u']$ with $t \xrightarrow{\text{id}_x, \varphi, \mu}_{\text{sn}} t'$ and $u \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} u'$, assuming
 300 $\Phi \triangleright \Gamma \vdash_{\varphi}^{\mu} t[x \setminus u] : \tau$. By inversion of the last rule in Φ we know there is a subderivation
 301 $\Phi' \triangleright \Gamma'; x : \mathcal{M} \vdash_{\varphi}^{\mu} t : \tau$. By induction hypothesis we know that reduction $t \xrightarrow{\text{id}_x, \varphi, \mu}_{\text{sn}} t'$ is
 302 at a typed position in Φ' , thus x is typed in t and $\mathcal{M} \neq \{\!\!\}\}$. Then u is typed in Φ and
 303 we can conclude by induction hypothesis on $u \xrightarrow{\rho, \varphi, \perp}_{\text{sn}} u'$. ◀

304 ▶ **Theorem 11 (Completeness).** *If a λ -term t is weakly normalizing in the λ -calculus, then
 305 t is strongly normalizing in λ_{sn} . Moreover, if n_{β} is the normal form of t in the λ -calculus,
 306 then any normal form n_{sn} of t in λ_{sn} is such that $n_{\text{sn}}^* = n_{\beta}$.*

307 **Proof.** Let t be a pure λ -term that admits a normal form n_{β} for β -reduction. By Theorem 1
 308 there exists a derivable typing judgment $\Gamma \vdash t : \tau$ such that $\{\!\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$. Thus by
 309 Theorems 10 and 2, the term t is strongly normalizing for \rightarrow_{sn} . Let $t \rightarrow_{\text{sn}}^* n_{\text{sn}}$ be a maximal
 310 reduction in λ_{sn} . By Lemma 4, $n_{\text{sn}} \in \mathcal{N}_{\varphi}$, and by Lemma 5, n_{sn}^* is a normal form in the
 311 λ -calculus. Moreover, by simulation (Lem. 3), there is a reduction $t^* \rightarrow_{\beta}^* n_{\text{sn}}^*$. By uniqueness
 312 of the normal form in the λ -calculus, $n_{\text{sn}}^* = n_{\beta}$. ◀

313 4 Relatively optimal strategies

314 Our proposed λ_{sn} -calculus guarantees that, in the process of reducing a term to its strong
 315 normal form, only needed redexes are ever reduced. This does not tell anything about the
 316 length of reduction sequences, though. Indeed, a term might be substituted several times
 317 before being reduced, thus leading to duplicate computations. To prevent this duplication, we
 318 introduce a notion of *local normal form*, which will be used to restrict the *value* criterion in
 319 the LSV-BASE rule of Fig. 4. We then show that this restriction is strong enough to guarantee
 320 the diamond property. Finally, we explain why our restricted calculus, named $\lambda_{\text{sn}+}$, only
 321 produces minimal sequences, among all the reduction sequences allowed by λ_{sn} . This makes
 322 it a relatively optimal strategy.

323 4.1 Local normal forms

324 In λ_{c} and λ_{sn} , substituted terms can be arbitrary values. In particular, they might be
 325 abstractions whose body contains some redexes. Since substituted variables can appear
 326 multiple times, this would cause the redex to be reduced several times if the value is
 327 substituted too soon. Let us illustrate this phenomenon on the following example, where
 328 $\text{id} = \lambda x.x$. The sequence of reductions does not depend on the set φ of frozen variables nor
 329 on the position μ , so we do not write them to lighten a bit the notations. Subterms that are
 330 about to be substituted or reduced are underlined.

$$\begin{array}{l}
 \underline{(\lambda w.w w)} (\lambda y.\text{id } y) \xrightarrow{\text{db}}_{\text{sn}} (\underline{w} w)[w \setminus \lambda y.\text{id } y] \\
 \xrightarrow{\text{lsv}}_{\text{sn}} ((\lambda y.\underline{\text{id } y}) w)[w \setminus \lambda y.\text{id } y] \\
 \xrightarrow{\text{db}}_{\text{sn}} ((\lambda y.x[x \setminus y]) w)[w \setminus \lambda y.\text{id } y] \\
 \xrightarrow{\text{db}}_{\text{sn}} \underline{x}[x \setminus y][y \setminus w][w \setminus \lambda y.\text{id } y] \\
 \xrightarrow{\text{lsv}^3}_{\text{sn}} (\lambda y.\underline{\text{id } y})[x \setminus \lambda y.\text{id } y][y \setminus \lambda y.\text{id } y][w \setminus \lambda y.\text{id } y] \\
 \xrightarrow{\text{db}}_{\text{sn}} (\lambda y.x[x \setminus y])[x \setminus \lambda y.\text{id } y][y \setminus \lambda y.\text{id } y][w \setminus \lambda y.\text{id } y]
 \end{array}$$

332 Notice how $\text{id } y$ is reduced twice, which would not have happened if the second reduction
 333 had focused on the body of the abstraction.

$$\begin{array}{c}
\frac{x \in \varphi \cup \omega}{x \in \mathcal{N}_{\varphi, \omega, \mu}} \text{VAR} \quad \frac{t \in \mathcal{N}_{\varphi \cup \{x\}, \omega, \top}}{\lambda x. t \in \mathcal{N}_{\varphi, \omega, \top}} \lambda\text{-}\varphi \quad \frac{t \in \mathcal{N}_{\varphi, \omega \cup \{x\}, \perp}}{\lambda x. t \in \mathcal{N}_{\varphi, \omega, \perp}} \lambda\text{-}\omega \quad \frac{t \in \mathcal{N}_{\varphi, \omega, \mu}}{t[x \setminus u] \in \mathcal{N}_{\varphi, \omega, \mu}} \text{ES} \\
\\
\frac{t \in \mathcal{N}_{\varphi, \omega, \mu} \quad t \in \mathcal{S}_{\varphi} \quad u \in \mathcal{N}_{\varphi, \omega, \top}}{t u \in \mathcal{N}_{\varphi, \omega, \mu}} @\text{-}\varphi \quad \frac{t \in \mathcal{N}_{\varphi, \omega, \mu} \quad t \in \mathcal{S}_{\omega}}{t u \in \mathcal{N}_{\varphi, \omega, \mu}} @\text{-}\omega \\
\\
\frac{t \in \mathcal{N}_{\varphi \cup \{x\}, \omega, \mu} \quad u \in \mathcal{N}_{\varphi, \omega, \top} \quad u \in \mathcal{S}_{\varphi}}{t[x \setminus u] \in \mathcal{N}_{\varphi, \omega, \mu}} \text{ES-}\varphi \quad \frac{t \in \mathcal{N}_{\varphi, \omega \cup \{x\}, \mu} \quad u \in \mathcal{S}_{\omega}}{t[x \setminus u] \in \mathcal{N}_{\varphi, \omega, \mu}} \text{ES-}\omega
\end{array}$$

■ **Figure 7** Local normal forms.

$$\frac{t \xrightarrow{\text{sub}_{x \setminus v, \varphi, \mu}}_{\text{sn}} t' \quad v \in \mathcal{N}_{\varphi, \emptyset, \perp}}{t[x \setminus v] \xrightarrow{\varphi, \mu}_{\text{lsv}} t'[x \setminus v]} \text{LSV-BASE}$$

■ **Figure 8** New rule LSV-BASE for $\lambda_{\text{sn}+}$.

334 This suggests that a substitution should only be allowed if the substituted term is in
335 normal form. But such a strong requirement is incompatible with our calculus, as it would
336 prevent the abstraction $\lambda y. y \Omega$ (with Ω a diverging term) to ever be substituted in the
337 following example, thus preventing normalization (with a a closed term).

$$\begin{array}{l}
\underline{w} (\lambda x. a)[w \setminus \lambda y. y \Omega] \xrightarrow{\text{lsv}}_{\text{sn}} (\lambda y. y \Omega) (\lambda x. a)[w \setminus \lambda y. y \Omega] \\
\quad \xrightarrow{\text{db}}_{\text{sn}} (y \Omega)[y \setminus \lambda x. a][w \setminus \lambda y. y \Omega] \\
338 \quad \xrightarrow{\text{lsv}}_{\text{sn}} ((\lambda x. a) \Omega)[y \setminus \lambda x. a][w \setminus \lambda y. y \Omega] \\
\quad \xrightarrow{\text{db}}_{\text{sn}} a[x \setminus \Omega][y \setminus \lambda x. a][w \setminus \lambda y. y \Omega]
\end{array}$$

339 Notice how the sequence of reductions has progressively removed all the occurrences of Ω ,
340 until the only term left to reduce is the closed term a .

341 To summarize, substituting any value is too permissive and can cause duplicate computa-
342 tions, while substituting only normal forms is too restrictive as it prevents normalization. So,
343 we need some relaxed notion of normal form, which we call *local normal form*. The intuition
344 is as follows. The term $\lambda y. y \Omega$ is not in normal form, because it could be reduced if it were
345 in a \top position. But in a \perp position, variable y is not frozen, which prevents any further
346 reduction of $y \Omega$. The inference rules are presented in Fig. 7.

347 If an abstraction is in a \top position, its variable is added to the set φ of frozen variables,
348 as in Fig. 3. But if an abstraction is in a \perp position, its variable is added to a new set ω , as
349 shown in rule $\lambda\text{-}\omega$ of Fig. 7. That is what will happen to y in $\lambda y. y \Omega$.

350 For an application, the left part is still required to be a structure. But if the leading
351 variable of the structure is not frozen (and thus in ω), our λ_{sn} -calculus guarantees that no
352 reduction will occur in the right part of the application. So, this part does not need to be
353 constrained in any way. This is rule $@\text{-}\omega$ of Fig. 7. It applies to our example, since $y \Omega$ is a
354 structure led by $y \in \omega$. Substitutions are handled in a similar way, as shown by rule $\text{ES-}\omega$.

355 Now that we have characterized the local normal forms. Let us revisit our λ_{sn} -calculus.
356 The rules of $\lambda_{\text{sn}+}$ are the same as the ones of λ_{sn} presented in Fig. 4 and Fig. 3, except that
357 a new rule replaces LSV-BASE. This rule is shown in Fig. 8.

358 **4.2 Diamond property**

359 As mentioned before, in both λ_c and λ_{sn} , terms might be substituted as soon as they are
 360 values, thus potentially causing duplicate computations. As a consequence, these calculi
 361 cannot have the diamond property, as shown on the following example.

$$\begin{array}{ccc}
 & & ((\lambda x.(\lambda y.y) x) w)[w \setminus \lambda x.(\lambda y.y) x] \longrightarrow_3 ((\lambda x.y[y \setminus x]) w)[w \setminus \lambda x.(\lambda y.y) x] \\
 & \nearrow_1 & \\
 (w w)[w \setminus \lambda x.(\lambda y.y) x] & & \\
 & \searrow_2 & \\
 & & (w w)[w \setminus \lambda x.y[y \setminus x]] \xrightarrow{4} ((\lambda x.y[y \setminus x]) w)[w \setminus \lambda x.y[y \setminus x]]
 \end{array}$$

362
 363 In λ_{sn} , the leftmost term can be reduced, either by rule **lsv** (arrow 1) because the
 364 substituted term is a value, or by rule **dB** (arrow 2). The top term can only be reduced
 365 by rule **dB** (arrow 3) because the substitution variable is not reachable. The bottom term
 366 can only be reduced by rule **lsv** (arrow 4) because the substituted term is not reducible.
 367 The two new terms are different, thus breaking the diamond property. It would take one
 368 more reduction step (in λ_c) for the top sequence to reach the bottom-right term. But in
 369 our restricted calculus λ_{sn+} , arrow 1 is forbidden, since the substituted term is not in local
 370 normal form. By preventing such sequences, the diamond property is restored.

371 ► **Theorem 12 (Diamond).** *Suppose $t \xrightarrow{\rho_1, \varphi, \mu}_{sn+} t_1$ and $t \xrightarrow{\rho_2, \varphi, \mu}_{sn+} t_2$. Assume that, if ρ_1
 372 and ρ_2 are *sub* or *id*, then they apply to separate variables. Then there exists t' such that
 373 $t_1 \xrightarrow{\rho_2, \varphi, \mu}_{sn+} t'$ and $t_2 \xrightarrow{\rho_1, \varphi, \mu}_{sn+} t'$.*

374 **Proof.** The statement has first to be generalized so that the steps $t \rightarrow t_1$ and $t \rightarrow t_2$ can use
 375 the main reduction $\xrightarrow{\rho, \varphi, \mu}_{sn}$ or the auxiliary reductions \rightarrow_{db} and $\xrightarrow{\varphi, \mu}_{lsv}$. Then it becomes a
 376 tedious but rather unsurprising induction on t , with reasoning by case on the last inference
 377 rule applied on each side. One notable case is when the two reductions are respectively given
 378 by rules **@-LEFT** and **@-RIGHT**. Indeed, the reduction on the left does not interfere with the
 379 reduction on the right thanks to a stability property of structures (Lem. 13 below). ◀

380 ► **Lemma 13 (Stability of structures).** *If $t \in \mathcal{S}_\varphi$ and $t \xrightarrow{\rho, \varphi, \mu}_{sn+} t'$ then $t' \in \mathcal{S}_\varphi$*

381 **4.3 Minimal path length**

382 The λ_{sn+} -calculus is a restriction of λ_{sn} that causes term to be reduced to local normal form
 383 before they can be substituted (Fig. 8), thus reducing the amount of duplicated work. Except
 384 for this restriction, any reduction sequence of λ_{sn} is admissible in λ_{sn+} . So, any λ_{sn} -reduction
 385 sequence to a normal form can be turned into a λ_{sn+} one that is no longer. Since the diamond
 386 property ensures that all the sequences to a normal form have the same length in λ_{sn+} , this
 387 guarantees that the reduction sequences are of minimal length.

388 ► **Theorem 14 (Minimal length).** *With $t' \in \mathcal{N}_\varphi$, if $t \xrightarrow{n}_{sn} t'$ and $t \xrightarrow{m}_{sn+} t'$ then $m \leq n$.*

389 Remark that this minimality result is relative to λ_{sn} . The reduction sequences of λ_{sn+}
 390 are not necessarily optimal with respect to the unconstrained λ_c or λ -calculi. For instance,
 391 neither λ_{sn+} nor λ_{sn} allow reducing r in the term $(\lambda x.x (x a)) (\lambda y.y r)$ prior to its duplication.

392 **5 Formalization in Abella**

393 We used the Coq proof assistant for our first attempts to formalize our results. We experi-
 394 mented both with the locally nameless approach [11] and parametric higher-order abstract

395 syntax [12]. While we might eventually have succeeded using the locally nameless approach,
 396 having to manually handle binders felt way too cumbersome. So, we turned to a dedicated formal
 397 system, Abella [5], in the hope that it would make syntactic proofs more straightforward.
 398 This section describes our experience with this tool.²

399 5.1 Nominal variables and λ -tree syntax

400 Our initial motivation for using Abella was the availability of nominal variables through
 401 the `nabla` quantifier. Indeed, in order to open a bound term, one has to replace the bound
 402 variable with a fresh global variable. This freshness is critical to avoid captures; but handling
 403 it properly causes a lot of bureaucracy in the proofs. By using nominal variables, which are
 404 guaranteed to be fresh by the logic, this issue disappears.

405 Here is an excerpt of our original definition of the `nf` predicate, which states that a term
 406 is in normal form for our calculus. The second line states that any nominal variable is in
 407 normal form, while the third line states that an abstraction is in normal form, as long as the
 408 abstracted term is in normal form for any nominal variable.

```
409 Define nf : trm -> prop by
410   nabla x, nf x;
411   nf (abs U) := nabla x, nf (U x);
412   ...
413
414
```

415 Note that Abella is based on a λ -tree approach (higher-order abstract syntax). In the
 416 above excerpt, `U` has a bound variable and `(U x)` substitutes it with the fresh variable `x`.
 417 More generally, `(U V)` is the term in which the bound variable is substituted with the term `V`.

418 This approach to fresh variables was error-prone at first. Several of our formalized
 419 theorems ended up being pointless, despite seemingly matching the statements of our pen-
 420 and-paper formalization. Consider the following example. This proposition states that, if `T`
 421 is a structure with respect to `x`, and if `U` is related to `T` by the unfolding relation `star`, then
 422 `U` is also a structure with respect to `x`.

```
423 forall T U, nabla x,
424   struct T x -> star T U -> struct U x.
425
426
```

427 Notice that the nominal variable `x` is quantified after `T`. As a consequence, its freshness
 428 ensures that it does not occur in `T`. Thus, the proposition is vacuously true, since `T` can
 429 only be a structure with respect to a variable that occurs in it. Had the quantifiers been
 430 exchanged, the statement would have been fine. Unfortunately, Abella kind of requires
 431 universal quantifiers to happen before nominal ones in theorem statements, thus exacerbating
 432 the issue. The correct way to state the above proposition is by carefully lifting any term in
 433 which a given free variable could occur:

```
434 forall T U, nabla x,
435   struct (T x) x -> star (T x) (U x) -> struct (U x) x.
436
437
```

438 Once one has overcome these hurdles, advantages become apparent. For example, to state
 439 that some free variable does not occur in a term, not lifting this term is sufficient. And if it
 440 needed to be lifted for some other reason, one can always equate it to a constant λ -tree. For
 441 instance, one of our theorems needed to state that the free variable `x` occurring in `T` could
 442 not occur in `U`, by virtue of `star`. This was expressed as follows (with `y \ V` denoting `y ↦ V`):

```
443 star (T x) (U x) -> exists V, U = (y \ V).
444
445
```

² See appendix for all the definitions and the statement of the main theorems.

446 **5.2 Judgments, contexts, and derivations**

447 Abella provides two levels of logic: a minimal logic used for specifications and an intuitionistic
 448 logic used for inductive reasoning over relations. At first, we only used the reasoning logic.
 449 By doing so, we were using Abella as if we were using Coq, except for the additional `nabla`
 450 quantifiers. We knew of the benefits of the specification logic when dealing with judgments
 451 and contexts; but in the case of the untyped λ -calculus, we could not see any use for those.

452 Our point of view started to shift once we had to manipulate sets of free variables, in
 453 order to distinguish which of them were frozen. We could have easily formalized such sets by
 454 hand; but since Abella is especially designed to handle sets of binders, we gave it a try. Let
 455 us consider the above predicate `nf` anew, except that it is now defined using λ -Prolog rules
 456 (`␣` is the universal quantifier in the specification logic).

```
457 nf X :- frozen X.  
458 nf (abs U) :- ␣ x \ frozen x => nf (U x).  
459 ...  
460 ...  
461 ...
```

462 Specification-level propositions have the form $\{L \mid P\}$, with P a proposition defined in
 463 λ -Prolog and L a list of propositions representing the context of P . Consider the proposition
 464 $\{L \mid \text{nf } (\text{abs } T)\}$. If there were only the two rules above, there would be only three ways of
 465 deriving the proposition. Indeed, it can be derived from $\{L \mid \text{frozen } (\text{abs } T)\}$ (first rule).
 466 It can also be derived from `nabla x`, $\{L, \text{frozen } x \mid \text{nf } (T \ x)\}$ (second rule). Finally,
 467 the third way to derive it is if `nf (abs T)` is already a member of the context L .

468 The second and third derivations illustrate how Abella automates the handling of contexts.
 469 But where Abella shines is that some theorems come for free when manipulating specification-
 470 level properties, especially when it comes to substitution. Suppose that one wants to prove
 471 $\{L \mid P (T \ U)\}$, *i.e.*, term T whose bound variable was replaced with U satisfies predicate P
 472 in context L . The simplest way is if one can prove `nabla x`, $\{L \mid P (T \ x)\}$. In that case,
 473 one can instantiate the nominal variable x with U and conclude.

474 But more often that not, x occurs in the context, *e.g.*, $\{L, Q \ x \mid P (T \ x)\}$ instead of
 475 $\{L \mid P (T \ x)\}$. Then, proving $\{L \mid P (T \ U)\}$ is just a matter of proving $\{L \mid Q \ x\}$. But,
 476 what if the latter does not hold? Suppose one can only prove $\{L \mid R \ x\}$, with $R \ V \text{ :- } Q \ V$.
 477 In that case, one can reason on the derivation of $\{L, Q \ x \mid P (T \ x)\}$ and prove that $\{L,$
 478 $R \ x \mid P (T \ x)\}$ necessarily holds, by definition of R . This ability to inductively reason on
 479 derivations is a major strength of Abella.

480 Having to manipulate contexts led us to revisit most of our pen-and-paper concepts. For
 481 example, a structure was no longer defined as a relation with respect to its leading variable
 482 (*e.g.*, `struct T x`) but with respect to all the frozen variables (*e.g.*, $\{\text{frozen } x \mid \text{struct } T\}$).
 483 In turn, this led us to handle live variables purely through their addition to contexts: $\varphi \cup \{x\}$.
 484 Our freshness convention is a direct consequence, as in Fig. 2 for example.

485 Performing specification-level proofs does not come without its own set of issues, though.
 486 As explained earlier, a proposition $\{L \mid \text{nf } (\text{abs } T)\}$ is derivable from the consequent
 487 being part of the context L , which is fruitless. The way around it is to define a predicate
 488 describing contexts that are well-formed, *e.g.*, L contains only propositions of the form $(\text{nf } x)$
 489 with x nominal. As a consequence, the case above can be eliminated because `(abs T)` is
 490 not a nominal variable. Unfortunately, defining these predicates and proving the associated
 491 helper lemmas is tedious and extremely repetitive. Thus, the user is encouraged to reuse
 492 existing context predicates rather than creating dedicated new ones, hence leading to sloppy
 493 and convoluted proofs. Having Abella provide some automation for handling well-formed
 494 contexts would be a welcome improvement.

5.3 Functions and relations

Our Abella formalization assumes a type `trm` and three predefined ways to build elements of that type: application, abstraction, and explicit substitution. For example, a term $t[x \backslash u]$ of our calculus will be denoted `(es (x \ t) u)` with `t` containing some occurrences of `x`.

```

499 type app trm -> trm -> trm.
500 type abs (trm -> trm) -> trm.
501 type es (trm -> trm) -> trm -> trm.

```

Since Abella does not provide functions, we instead use a relation to define the unfolding function $t \mapsto t^*$. Of particular interest is the way binders are handled; they are characterized by stating that they are their own image: `star x x`.

```

507 star (app U V) (app X Y) :- star U X, star V Y.
508 star (abs U) (abs X) :- pi x \ star x x => star (U x) (X x).
509 star (es U V) (X Y) :- star V Y, pi x \ star x x => star (U x) (X x).

```

Since this is just a relation, we have to prove that it is defined over all the closed terms of our calculus, that it maps only to pure λ -terms, and that it maps to exactly one λ -term. Needless to say, all of that would be simpler if Abella had native support for functions.

6 Conclusion

This paper presents a λ -calculus dedicated to strong reduction. In the spirit of a call-by-need strategy with explicit substitutions, it builds on a linear substitution calculus [3]. Our calculus, however, embeds a syntactic criterion that ensures that only needed redexes are considered. Moreover, by delaying substitutions until they are in so-called local normal forms rather than just values, all the reduction sequences are of minimal length.

Properly characterizing these local normal forms proved difficult and lots of iterations were needed until we reached the presented definition. Our original approach relied on evaluation contexts, as in the original presentation of a strong call-by-need strategy [8]. While tractable, this made the proof of the diamond property long and tedious. It is the use of Abella that led us to reconsider this approach. Indeed, the kind of reasoning Abella favors forced us to give up on evaluation contexts and look for reduction rules that were much more local in nature. In turn, these changes made the relation with typing more apparent. In hindsight, this would have avoided a large syntactic proof in [8].

Due to decidability, our syntactic criterion can characterize only part of the needed redexes at a given time. All the needed reductions will eventually happen, but detecting the neededness of a redex too late might prevent the optimal reduction. It is an open question whether there is a different yet simple criterion that characterizes more needed redexes, and thus potentially allow for even shorter sequences than our calculus.

Even with the current criterion, there is still work to be done. First and foremost, the Abella formalization should be completed to at least include the diamond property. There are also some potential improvements to consider. For example, our calculus could be made to not substitute variables that are not applied (rule LSV-BASE), following [26, 2] but it opens the question of how to characterize the normal forms then. Another venue for investigation is how this work interacts with fully lazy sharing, which avoids more duplications but whose properties are tightly related to weak reduction [6]. Finally, this paper stops at describing the reduction rules of our calculus and does not investigate what an efficient abstract machine would look like.

543 — References

- 544 1 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. In Xinyu
545 Feng and Sungwoo Park, editors, *Programming Languages and Systems*, volume 9458 of *Lecture*
546 *Notes in Computer Science*, pages 231–250, 2015. doi:10.1007/978-3-319-26529-2_13.
- 547 2 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is
548 reasonable, implisively, February 2021. arXiv:2102.06928.
- 549 3 Beniamino Accattoli and Delia Kesner. The structural λ -calculus. In Anuj Dawar and Helmut
550 Veith, editors, *Computer Science Logic*, pages 381–395, 2010. doi:10.5555/1887459.1887491.
- 551 4 Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. A
552 call-by-need lambda calculus. In *22nd ACM SIGPLAN-SIGACT Symposium on Principles of*
553 *Programming Languages*, POPL '95, pages 233–246, 1995. doi:10.1145/199448.199507.
- 554 5 David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen
555 Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *Journal*
556 *of Formalized Reasoning*, 7(2):1–89, December 2014. doi:10.6092/issn.1972-5787/4650.
- 557 6 Thibaut Balabonski. A unified approach to fully lazy sharing. In John Field and Michael Hicks,
558 editors, *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*,
559 POPL, pages 469–480, January 2012. doi:10.1145/2103656.2103713.
- 560 7 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and
561 Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming*,
562 ICFP'13, pages 263–274, September 2013. doi:10.1145/2500365.2500606.
- 563 8 Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of
564 strong call by need. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017. doi:10.1145/3110264.
- 565 9 Malgorzata Biernacka, Dariusz Biernacki, Witold Charatonik, and Tomasz Drab. An abstract
566 machine for strong call by value. September 2020. arXiv:2009.06984.
- 567 10 Malgorzata Biernacka and Witold Charatonik. Deriving an Abstract Machine for Strong Call
568 by Need. In Herman Geuvers, editor, *4th International Conference on Formal Structures for*
569 *Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in*
570 *Informatics (LIPIcs)*, pages 8:1–8:20, 2019. doi:10.4230/LIPIcs.FSCD.2019.8.
- 571 11 Arthur Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*,
572 49(3):363–408, October 2012. doi:10.1007/s10817-011-9225-2.
- 573 12 Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *13th*
574 *ACM SIGPLAN International Conference on Functional Programming*, ICFP, pages 143–156,
575 September 2008. doi:10.1145/1411204.1411226.
- 576 13 M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the
577 λ -calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980. doi:10.1305/ndjfl/
578 1093883253.
- 579 14 Pierre Crégut. An abstract machine for lambda-terms normalization. In *ACM Conference on*
580 *LISP and Functional Programming*, LFP '90, page 333–340, 1990. doi:10.1145/91556.91681.
- 581 15 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and
582 John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, pages 555–574, 1994.
- 583 16 Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In
584 *7th ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, page
585 235–246, 2002. doi:10.1145/581478.581501.
- 586 17 Carsten Kehler Holst and Darsten Krogh Gomard. Partial evaluation is fuller laziness. In *ACM*
587 *SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*,
588 PEPM '91, page 223–233, 1991. doi:10.1145/115865.115890.
- 589 18 Delia Kesner. A theory of explicit substitutions with safe and full composition. *Logical Methods*
590 *in Computer Science*, 5(3), May 2009. doi:10.2168/LMCS-5(3:1)2009.
- 591 19 Delia Kesner. Reasoning about call-by-need by means of types. In Bart Jacobs and Christof
592 Löding, editors, *Foundations of Software Science and Computation Structures*, pages 424–441,
593 2016. doi:10.1007/978-3-662-49630-5_25.

- 594 20 Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus.
 595 In Josep Diaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science*,
 596 volume 8705 of *Lecture Notes in Computer Science*, pages 296–310, 2014. doi:10.1007/
 597 978-3-662-44602-7_23.
- 598 21 John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *J.*
 599 *Funct. Program.*, 8(3):275–317, May 1998. doi:10.1017/S0956796898003037.
- 600 22 Robin Milner. Local bigraphs and confluence: Two conjectures. *Electron. Notes Theor. Comput.*
 601 *Sci.*, 175(3):65–73, June 2007. doi:10.1016/j.entcs.2006.07.035.
- 602 23 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer*
 603 *Science*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.
- 604 24 Gordon D. Plotkin. A structural approach to operational semantics. Technical report, DAIMI
 605 FN-19, Computer Science Department, Aarhus University, 1981.
- 606 25 Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis,
 607 Oxford, 1971.
- 608 26 Nobuko Yoshida. Optimal reduction in weak-lambda-calculus with shared environments. In
 609 *Conference on Functional Programming Languages and Computer Architecture*, FPCA '93,
 610 page 243–252, 1993. doi:10.1145/165180.165217.

611 A Formal definitions

612 This appendix describes the main definitions of the Abella formalization. The reduction
 613 rules of λ_{sn} and λ_{sn+} presented in Fig. 3 are as follows.

```

614 step R top (abs T) (abs T') :- pi x \ frozen x => step R top (T x) (T' x).
615 step R B (abs T) (abs T') :- pi x \ step R bot (T x) (T' x).
616 step R B (app T U) (app T' U) :- step R bot T T'.
617 step R B (app T U) (app T U') :- struct T, step R B U U'.
618 step R B (es T U) (es T' U) :- pi x \ step R B (T x) (T' x).
619 step R B (es T U) (es T' U) :-
620   pi x \ frozen x => step R B (T x) (T' x), struct U.
621 step R B (es T U) (es T U') :-
622   pi x \ active x => step (idx x) B (T x) (T x), step R B U U'.
623
624
625 step (idx X) B X X :- active X.
626 step (sub X V) B X V :- active X.
627 step db B T T' :- step_db T T'.
628 step lsv B T T' :- step_lsv B T T'.

```

630 A small difference with the core of the paper is the predicate `active`, which characterizes
 631 the variable being considered id_x (`idx`) and $sub_{x \setminus v}$ (`sub`). This predicate is just a cheap way
 632 of remembering that the active variable is fresh yet not frozen.

633 Another difference is rule λ -BOT. While the antecedent of the rule is at position \perp as in
 634 the paper, the consequent is in any position rather than just \perp . Since any term reducible in
 635 position \perp is provably reducible in position \top , this is just a conservative generalization of
 636 the rule.

637 The auxiliary rules for λ_{sn+} , as given in Fig. 4 and Fig. 8 for rule LSV-BASE, are the same
 638 as in the core of the paper.

```

639 step_db (app (abs T) U) (es T U).
640 step_db (app (es T W) U) (es T' W) :- pi x \ step_db (app (T x) U) (T' x).
641
642
643 step_lsv B (es T (abs V)) (es T' (abs V)) :-
644   pi x \ active x => step (sub x (abs V)) B (T x) (T' x), lnf bot (abs V).

```

23:18 A strong call-by-need calculus

```

645 step_lsv B (es T (es U W)) (es T' W) :-
646   pi x\ step_lsv B (es T (U x)) (T' x).
647 step_lsv B (es T (es U W)) (es T' W) :-
648   pi x\ frozen x => step_lsv B (es T (U x)) (T' x), struct W.
649

```

650 Finally, an actual reduction is just comprised of rules DB and LSV in a \top position:

```

651 red T T' :- step db top T T'.
652 red T T' :- step lsv top T T'.
653

```

655 The normal forms of λ_{sn} and λ_{sn+} , given in Fig. 5, are as follows.

```

656 nf X :- frozen X.
657 nf (app U V) :- nf U, nf V, struct U.
658 nf (abs U) :- pi x\ frozen x => nf (U x).
659 nf (es U V) :- pi x\ frozen x => nf (U x), nf V, struct V.
660 nf (es U V) :- pi x\ nf (U x).
661
662

```

663 They make use of structures (`struct`), as given in Fig. 2.

```

664 struct X :- frozen X.
665 struct (app U V) :- struct U.
666 struct (es U V) :- pi x\ struct (U x).
667 struct (es U V) :- pi x\ frozen x => struct (U x), struct V.
668
669

```

670 The local norm forms of Fig. 7 are as follows. As for the `step` relation, one of the rules
671 for abstraction was generalized a bit with respect to the paper. This time, it is for the \top
672 position, since any term that is locally normal in a \top position is also locally normal in a \perp
673 position.

```

674 lnf B X :- frozen X.
675 lnf B X :- omega X.
676 lnf B (app T U) :- lnf B T, struct T, lnf top U.
677 lnf B (app T U) :- lnf B T, struct_omega T.
678 lnf B (abs T) :- pi x\ frozen x => lnf top (T x).
679 lnf bot (abs T) :- pi x\ omega x => lnf bot (T x).
680 lnf B (es T U) :- pi x\ lnf B (T x).
681 lnf B (es T U) :- pi x\ frozen x => lnf B (T x), lnf B U, struct U.
682 lnf B (es T U) :- pi x\ omega x => lnf B (T x), struct_omega U.
683

```

685 Structures with respect to the set ω use a dedicated predicate `struct_omega`, which is
686 just a duplicate of `struct`. Another approach, perhaps more elegant, would have been to
687 parameterize `struct` with either `frozen` or `omega`.

```

688 struct_omega X :- omega X.
689 struct_omega (app U V) :- struct_omega U.
690 struct_omega (es U V) :- pi x\ struct_omega (U x).
691 struct_omega (es U V) :-
692   pi x\ omega x => struct_omega (U x), struct_omega V.
693
694

```

695 Normal forms of the λ -calculus are defined as follows:

```

696 notabs T :- frozen T.
697 notabs (app T U).
698 nfb X :- frozen X.
699 nfb (abs T) :- pi x\ frozen x => nfb (T x).
700 nfb (app T U) :- nfb T, nfb U, notabs T.
701
702

```

703 Finally, let us remind the definitions of a pure λ -term, of a β -reduction, of a sequence of
704 zero or more β -reductions, and of the unfolding operation from λ_c to λ .

```
705 pure (app U V) :- pure U, pure V.
706 pure (abs U) :- pi x \ pure x => pure (U x).
707
708
709 beta (app M N) (app M' N) :- beta M M'.
710 beta (app M N) (app M N') :- beta N N'.
711 beta (abs R) (abs R') :- pi x \ beta (R x) (R' x).
712 beta (app (abs R) M) (R M).
713
714 betas M M.
715 betas M N :- beta M P, betas P N.
716
717 star (app U V) (app X Y) :- star U X, star V Y.
718 star (abs U) (abs X) :- pi x \ star x x => star (U x) (X x).
719 star (es U V) (X Y) :- star V Y, pi x \ star x x => star (U x) (X x).
720
```

721 **B** Formally verified properties

722 This appendix states the theorems that were fully proved using Abella. First comes the
723 simulation property (Lem. 3), which states that, if $T \rightarrow_{\text{sn}+} U$, then $T^* \rightarrow_{\beta}^* U^*$.

```
724 Theorem simulation' : forall T U T* U*,
725   {star T T*} -> {star U U*} -> {red T U} -> {betas T* U*}.
726
727
```

728 Then comes the fact that (local) normal forms are not reducible in $\lambda_{\text{sn}+}$ (part of Lemma 4).

```
729 Theorem lnf_nand_red : forall T U,
730   {lnf top T} -> {red T U} -> false.
731
```

```
732 Theorem nf_nand_red : forall T U,
733   {nf T} -> {red T U} -> false.
734
735
```

736 Finally, if T is a normal form of λ_{sn} , then T^* is a normal form of the λ -calculus (Lem. 5).

```
737 Theorem correctness1' : forall T T*,
738   {nf T} -> {star T T*} -> {nfb T*}.
739
740
```

741 **C** Proof of the subformula properties

742 We recall here Lemma 7:

743 If $\Gamma \vdash_{\varphi}^{\mu} t : \tau$ and $t \in \mathcal{S}_{\varphi}$, then there is $x \in \varphi$ such that $\tau \in \mathcal{T}_+(\Gamma(x))$.

744 **Proof.** By induction on the structure of t .

745 ■ Case $t = x$. By inversion of $x \in \mathcal{S}_{\varphi}$ we deduce $x \in \varphi$. Moreover the only rule applicable
746 to derive $\Gamma \vdash_{\varphi}^{\mu} x : \tau$ is TY-VAR, which gives the conclusion.

747 ■ Case $t = t_1 t_2$. By inversion of $t_1 t_2 \in \mathcal{S}_{\varphi}$ we deduce $t_1 \in \mathcal{S}_{\varphi}$. Moreover the only
748 rules applicable to derive $\Gamma \vdash_{\varphi}^{\mu} t_1 t_2 : \tau$ are TY-@ and TY-@-S. Both have a premise
749 $\Gamma' \vdash_{\varphi}^{\mu} t_1 : \mathcal{M} \rightarrow \tau$ with $\Gamma' \subseteq \Gamma$, to which the induction hypothesis applies, ensuring
750 $\mathcal{M} \rightarrow \tau \in \mathcal{T}_+(\Gamma'(x))$ and thus $\tau \in \mathcal{T}_+(\Gamma'(x))$ and $\tau \in \mathcal{T}_+(\Gamma(x))$.

751 ■ Case $t = t_1[x \setminus t_2]$. We reason by case on the last rules applied to derive $t_1[x \setminus t_2] \in \mathcal{S}_{\varphi}$ and
752 $\Gamma \vdash_{\varphi}^{\mu} t_1[x \setminus t_2] : \tau$. There are two possible rules for each.

- 753 ■ Case where $t_1[x \setminus t_2] \in \mathcal{S}_\varphi$ is deduced from $t_1 \in \mathcal{S}_\varphi$ (with $x \notin \varphi$) and $\Gamma \vdash_\varphi^\mu t_1[x \setminus t_2] : \tau$
754 comes from rule TY-ES. This rule has in particular a premise $\Gamma' \vdash_\varphi^\mu t_1 : \tau$ for a
755 $\Gamma' = \Gamma''; x : \mathcal{M}$ such that $\Gamma'' \subseteq \Gamma$. We thus have by induction hypothesis on t_1 that
756 $\tau \in \mathcal{T}_+(\Gamma'(y))$ for some $y \in \varphi \cap \text{dom}(\Gamma')$. Since $y \in \varphi$ and $x \notin \varphi$ we have $y \neq x$. Then
757 $y \in \text{dom}(\Gamma'')$ and $y \in \text{dom}(\Gamma)$, and $\Gamma(y) = \Gamma''(y)$.
- 758 ■ In the three other cases, we have:
- 759 1. a hypothesis $t_1 \in \mathcal{S}_\varphi$ or $t_1 \in \mathcal{S}_{\varphi \cup \{x\}}$, from which we deduce $t_1 \in \mathcal{S}_{\varphi \cup \{x\}}$,
 - 760 2. a hypothesis $\Gamma' \vdash_\varphi^\mu t_1 : \tau$ or $\Gamma' \vdash_{\varphi \cup \{x\}}^\mu t_1 : \tau$ (for a $\Gamma' = \Gamma''; x : \mathcal{M}$ such that $\Gamma'' \subseteq \Gamma$),
761 from which we deduce $\Gamma' \vdash_{\varphi \cup \{x\}}^\mu t_1 : \tau$, and
 - 762 3. a hypothesis $t_2 \in \mathcal{S}_\varphi$, coming from the derivation of $t_1[x \setminus t_2]$ or the derivation of
763 $\Gamma \vdash_\varphi^\mu t_1[x \setminus t_2] : \tau$ (or both).
- 764 Then by induction hypothesis on t_1 we have $\tau \in \mathcal{T}_+(\Gamma'(y))$ for some $y \in \varphi \cup \{x\}$.
- 765 * If $y \neq x$, then $y \in \varphi$ and $\Gamma(y) = \Gamma''(y)$, which allows a direct conclusion.
 - 766 * If $y = x$, then $\tau \in \mathcal{T}_+(\Gamma'(x))$ implies $\mathcal{M} \neq \{\!\!\}\}$. Let $\sigma \in \mathcal{M}$ with $\tau \in \mathcal{T}_+(\sigma)$. The
767 instance of the rule TY-ES or TY-ES- \mathcal{S} we consider thus has at least one premise
768 $\Delta \vdash_\varphi^\perp t_2 : \sigma$ with $\Delta \subseteq \Gamma$. Since $t_2 \in \mathcal{S}_\varphi$, by induction hypothesis on t_2 there is
769 $z \in \varphi \cap \text{dom}(\Delta)$ such that $\sigma \in \mathcal{T}_+(\Delta(z))$. Then $\tau \in \mathcal{T}_+(\Delta(z))$, and $\tau \in \Gamma$. ◀

770 We recall here Lemma 8:

- 771 1. If $\Phi \triangleright \Gamma \vdash_\varphi^\top t : \tau$ then $\begin{cases} \mathcal{T}_+(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_+(\Gamma(x)) \cup \mathcal{T}_-(\tau) \\ \mathcal{T}_-(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_-(\Gamma(x)) \cup \mathcal{T}_+(\tau) \end{cases}$
- 772 2. If $\Phi \triangleright \Gamma \vdash_\varphi^\perp t : \tau$ then $\begin{cases} \mathcal{T}_+(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_+(\Gamma(x)) \\ \mathcal{T}_-(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_-(\Gamma(x)) \end{cases}$

773 **Proof.** By mutual induction on the typing derivations.

- 774 ■ Both properties are immediate in case TY-VAR, where $\text{fzt}(\Phi) = \{\sigma\}$.
- 775 ■ Cases for abstractions.
- 776 ■ If $\Phi \triangleright \Gamma \vdash_\varphi^\perp \lambda x.t : \mathcal{M} \rightarrow \tau$ by rule TY- λ - \perp with premise $\Phi' \triangleright \Gamma; x : \mathcal{M} \vdash_\varphi^\perp t : \tau$.
777 Write $\Gamma' = \Gamma; x : \mathcal{M}$. By induction hypothesis we have $\mathcal{T}_+(\text{fzt}(\Phi')) \subseteq \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma'(y))$.
778 Since $x \notin \varphi$ by renaming convention, we deduce that $\mathcal{T}_+(\text{fzt}(\Phi')) \subseteq \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma(y))$ and
779 $\mathcal{T}_+(\text{fzt}(\Phi)) \subseteq \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma(y))$. The same applies to negative type occurrences, which
780 concludes the case.
- 781 ■ If $\Phi \triangleright \Gamma \vdash_\varphi^\top \lambda x.t : \mathcal{M} \rightarrow \tau$ by rule TY- λ - \top with premise $\Phi' \triangleright \Gamma; x : \mathcal{M} \vdash_{\varphi \cup \{x\}}^\top t : \tau$.
782 Write $\Gamma' = \Gamma; x : \mathcal{M}$. By induction hypothesis we have

$$\begin{aligned}
\mathcal{T}_+(\text{fzt}(\Phi')) &\subseteq \bigcup_{y \in (\varphi \cup \{x\})} \mathcal{T}_+(\Gamma'(y)) \cup \mathcal{T}_-(\tau) \\
&= \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_+(\mathcal{M}) \cup \mathcal{T}_-(\tau) \\
&= \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_-(\mathcal{M} \rightarrow \tau)
\end{aligned}$$

784 Thus $\mathcal{T}_+(\text{fzt}(\Phi)) \subseteq \bigcup_{y \in \varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_-(\mathcal{M} \rightarrow \tau)$. The same applies to negative
785 occurrences, which concludes the case.

786 ■ Cases for application.

- 787 ■ Cases for TY-@ are by immediate application of the induction hypotheses.
- 788 ■ If $\Phi \triangleright \Gamma \vdash_\varphi^\mu t u : \tau$ by rule TY-@- \mathcal{S} , with premises $\Phi_t \triangleright \Gamma_t \vdash_\varphi^\perp t : \mathcal{M} \rightarrow \tau$, $t \in \mathcal{S}_\varphi$
789 and $\Phi_\sigma \triangleright \Delta_\sigma \vdash_\varphi^\top u : \sigma$ for $\sigma \in \mathcal{M}$, with $\Gamma_t \subseteq \Gamma$ and $\Gamma_\sigma \subseteq \Gamma$ for all $\sigma \in \mathcal{M}$.
790 Independently of the value of μ , we show that $\mathcal{T}_+(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_+(\Gamma(x))$ and
791 $\mathcal{T}_-(\text{fzt}(\Phi)) \subseteq \bigcup_{x \in \varphi} \mathcal{T}_-(\Gamma(x))$ to conclude on both sides of the mutual induction.
792 By immediate use of the induction hypothesis, $\mathcal{T}_+(\text{fzt}(\Phi_t)) \subseteq \bigcup_{x \in \varphi} \Gamma_t(x) \subseteq \mathcal{T}_+(\text{fzt}(\Phi))$.
793 By induction hypothesis on the other premises we have $\mathcal{T}_+(\text{fzt}(\Phi_\sigma)) \subseteq \bigcup_{x \in \varphi} \Gamma_\sigma(x) \cup$

794 $\mathcal{T}_-(\tau)$ for $\sigma \in \mathcal{M}$. We immediately have $\bigcup_{x \in \varphi} \Gamma_\sigma(x) \subseteq \bigcup_{x \in \varphi} \Gamma(x)$. We conclude by
 795 showing that $\mathcal{T}_-(\sigma) \subseteq \mathcal{T}_+(\Gamma_t(x))$ for some $x \in \varphi$. Since $t \in \mathcal{S}_\varphi$, by the first subformula
 796 property and the typing hypothesis on t we deduce that there is a $x \in \varphi$ such that
 797 $\mathcal{M} \rightarrow \tau \in \mathcal{T}_+(\Gamma_t(x))$. By closeness of type occurrences sets $\mathcal{T}_+(\tau)$ this means $\mathcal{T}_+(\mathcal{M} \rightarrow$
 798 $\tau) \subseteq \mathcal{T}_+(\Gamma_t(x))$. By definition $\mathcal{T}_+(\mathcal{M} \rightarrow \tau) = \mathcal{T}_-(\mathcal{M}) \cup \mathcal{T}_+(\tau) = \bigcup_{\sigma \in \mathcal{M}} \mathcal{T}_-(\sigma) \cup \mathcal{T}_+(\tau)$,
 799 which allows us to conclude the proof that $\bigcup_{x \in \varphi} \Gamma_\sigma(x) \cup \mathcal{T}_-(\tau) \subseteq \bigcup_{x \in \varphi} \Gamma(x)$.

800 The same argument also applies to negative positions, and concludes the case.

801 ■ Cases for explicit substitutions are by immediate application of the induction hypothesis.

802

