A Proof of Weak Termination of the Simply-Typed $\lambda\sigma$-Calculus

Jean Goubault-Larrecq

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

A Proof of Weak Termination of the Simply-Typed $\lambda\sigma$-Calculus

Jean Goubault-Larrecq

N° 3090
Janvier 1997

THÈME 2

Rapport de recherche

ISSN 0249-6399

A Proof of Weak Termination of the Simply-Typed $\lambda\sigma$-Calculus

Jean Goubault-Larrecq *

Thème 2 — Génie logiciel
et calcul symbolique

Projet Coq

Rapport de recherche n˚3090 — Janvier 1997 — 10 pages

Abstract: We show that reducing any simply-typed $\lambda\sigma$-term by applying the rules in σ eagerly always terminates, by a translation to the simply-typed λ-calculus, and similarly for $\lambda\sigma_{\Uparrow}$-terms with $\sigma_{\Uparrow}$-eager rewrites. This holds even with term and substitution meta-variables. In fact, every reduction terminates provided that (β)-redexes are only contracted under so-called safe contexts. The previous results follow because in σ, resp. $\sigma_{\Uparrow}$-normal forms, all contexts around terms of sort T are safe.

Key-words: $\lambda\sigma$-calculus, explicit substitutions, termination, λ-calculus, simple types

(Résumé : tsvp)

*Jean.Goubault@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Une preuve de terminaison faible du $\lambda\sigma$-calcul simplement typé

Résumé : Nous montrons que réduire n'importe quel $\lambda\sigma$-terme simplement typé en appliquant toujours les règles de σ le plus tôt possible est un processus qui termine. Ceci est prouvé à l'aide d'une traduction vers le λ-calcul simplement typé, et de même pour le $\lambda\sigma_{\Uparrow}$-calcul avec réduction prioritaire par $\sigma_{\Uparrow}$. Ceci est valide même en présence de méta-variables de termes et de substitutions. En fait, toute réduction termine dès qu'on ne contracte que les (β)-rédex qui apparaissent sous des contextes dits sûrs. Les résultats énoncés plus haut s'en déduisent car dans toute forme σ-normale, resp. $\sigma_{\Uparrow}$-normale, tous les contextes autour de termes de sorte T sont sûrs.

Mots-clé : $\lambda\sigma$-calcul, substitutions explicites, terminaison, λ-calcul, types simples

Introduction

Although the simply-typed $\lambda\sigma$-calculus does not terminate strongly [Mel94, Mel95], it terminates in the weak sense: every typed term has some (unique) normal form. We present a refined proof of this fact. In fact, we prove the widely believed claim that every reduction where σ steps are applied eagerly is finite.

For the sake of generality, we prove this result even in the presence of term and substitution metavariables, and for a general $\lambda\sigma$-calculus that encompasses both the original $\lambda\sigma$-calculus [ACCL90] and the $\lambda\mathcal{E}$-calculus of [HL89]. The techniques that we use are basically the same as in [GL97], where they are used for a more complicated calculus.

The plan of the paper is as follows. We recapitulate some basic definitions in Section 1 and prove a few preliminaries in Section 2. In Section 3, we prove the main theorem of this paper: every reduction where (β)-contraction occurs only under so-called *safe* contexts always terminates. This result is applied in Section 4 to show that all normalization strategies that apply σ (resp. $\sigma_\Uparrow$) eagerly in the $\lambda\sigma$-calculus (resp. in $\lambda\sigma_\Uparrow$, a.k.a. $\lambda\mathcal{E}$) terminate.

1 Definitions

Consider the language $\lambda\sigma_\Uparrow$ defined as $T \cup S$, where the sublanguage T and that of *explicit substitutions* or *stacks* S are given by:

$$\begin{array}{rcl} T & ::= & \mathcal{V}_T \mid \lambda T \mid TT \mid T[S] \mid 1 \\ S & ::= & \mathcal{V}_S \mid S \circ S \mid id \mid T \cdot S \mid \uparrow \mid \Uparrow S \end{array}$$

where $\mathcal{V}_T$ and $\mathcal{V}_S$ are disjoint infinite sets of *term variables* x, y, z, ..., and *stack variables* X, Y, Z, etc. The set of $\lambda\sigma$-terms is the subset of $\lambda\sigma_\Uparrow$-terms where $\Uparrow$ does not occur.

We call (untyped) $\lambda\sigma$-calculus the rewrite system of Figure 1. It is a slight generalization, in the sense that it can simulate every reduction, of both the $\lambda\sigma$-calculus of [ACCL90] and of the $\lambda\sigma_\Uparrow$-calculus of [HL89] (where it is named $\lambda\mathcal{E}$). Notice that, because of rule $(\eta \Uparrow)$, all the other rules with $\Uparrow$ in their names are superfluous. The $\lambda\sigma_\Uparrow$-calculus is the rewrite system defined by all rules of $\lambda\sigma$ without the three rules with an η in their names. (The latter is basically group (B) of [GL96], while the former is group (B) plus part of group (H)).

(β)	$(\lambda u)v \to u[v \cdot id]$	$(\eta \Uparrow)$	$\Uparrow w \to 1 \cdot (w \circ \uparrow)$
$([id])$	$u[id] \to u$	$(\eta \cdot)$	$1 \cdot \uparrow \to id$
$(\circ id)$	$u \circ id \to u$	$(\eta \cdot \circ)$	$(1[u]) \cdot (\uparrow \circ u) \to u$
$(id \circ)$	$id \circ u \to u$	$(1 \Uparrow)$	$1[\Uparrow u] \to 1$
$([])$	$(u[v])[w] \to u[v \circ w]$	$(1 \Uparrow \circ)$	$1[\Uparrow u \circ v] \to 1[v]$
$(\circ)$	$(u \circ v) \circ w \to u \circ (v \circ w)$	$(\uparrow\Uparrow)$	$\uparrow \circ \Uparrow u \to u \circ \uparrow$
(1)	$1[u \cdot v] \to u$	$(\uparrow\Uparrow \circ)$	$\uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v)$
$(\uparrow)$	$\uparrow \circ (u \cdot v) \to v$	$(\Uparrow\Uparrow)$	$\Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v)$
$(\cdot)$	$(u \cdot v) \circ w \to (u[w]) \cdot (v \circ w)$	$(\Uparrow\Uparrow \circ)$	$\Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w$
(λ)	$(\lambda u)[w] \to \lambda(u[\Uparrow w])$	$(\Uparrow \cdot)$	$\Uparrow u \circ (v \cdot w) \to v \cdot (u \circ w)$
(app)	$(uv)[w] \to (u[w])(v[w])$	$(\Uparrow id)$	$\Uparrow id \to id$

Figure 1: The $\lambda\sigma_\Uparrow$-calculus

The purpose of this paper is to show that the simply-typed $\lambda\sigma$-calculus terminates, when rewrites are restricted to be *σ-eager rewrites*, i.e. rewrites of the form:

$$u \xrightarrow{\sigma}{}^{*} u_1 \xrightarrow{(\beta)} u'_1 \xrightarrow{\sigma}{}^{*} u_2 \xrightarrow{(\beta)} u'_2 \xrightarrow{\sigma}{}^{*} \ldots \xrightarrow{\sigma}{}^{*} u_k \xrightarrow{(\beta)} u'_k \xrightarrow{\sigma}{}^{*} \ldots$$

where u_1, u_2, ..., u_k, ... are σ-normal. And similarly, that the simply-typed $\lambda\sigma_\Uparrow$-calculus terminates, when rewrites are restricted to be $\sigma_\Uparrow$-eager rewrites, defined similarly.

The typing rules are given in Figure 2. Types are either *term types* or *stack types*. Term types are *sequents* $\Gamma \vdash \tau$, where τ is a formula and Γ is a finite list of formulas $\tau_1, \ldots, \tau_n, \cdot$, where $\cdot$ marks the end of the list; formulas τ are either atoms A, B, ..., or implications (arrow types) $\tau_1 \rightarrow \tau_2$. Stack types are similar sequents $\Gamma \vdash \Delta$, but Δ is a finite list of formulas as well. For simplicity, we shall assume that the calculus is given in Church style: variables $x_{\Gamma \vdash \tau}$ are annotated with their type $\Gamma \vdash \tau$; we write them x when no confusion arises. (Writing terms in Church style will allow us to translate $\lambda\sigma$-terms to λ-terms in Section 3; otherwise, we would need to translate typing derivations of $\lambda\sigma$-terms to typing derivations of λ-terms, which is not that different, but would be much less readable.)

$$\frac{}{X_{\Gamma \vdash \Delta} : \Gamma \vdash \Delta}\,(Var_S) \qquad \frac{}{x_{\Gamma \vdash \tau} : \Gamma \vdash \tau}\,(Var_T)$$

$$\frac{}{id : \Gamma \vdash \Gamma}\,(I)$$

$$\frac{u : \Gamma \vdash \tau \quad v : \Gamma \vdash \Delta}{u \cdot v : \Gamma \vdash \tau, \Delta}\,(,I) \qquad \frac{}{1 : \tau, \Gamma \vdash \tau}\,(,E_1)$$

$$\frac{u : \Gamma \vdash \Delta}{\Uparrow u : \tau, \Gamma \vdash \tau, \Delta}\,(\Uparrow) \qquad \frac{}{\uparrow : \tau, \Gamma \vdash \Gamma}\,(,E_2)$$

$$\frac{u : \tau_1, \Gamma \vdash \tau_2}{\lambda u : \Gamma \vdash \tau_1 \rightarrow \tau_2}\,(\rightarrow I) \qquad \frac{u : \Gamma \vdash \tau_1 \rightarrow \tau_2 \quad v : \Gamma \vdash \tau_1}{uv : \Gamma \vdash \tau_2}\,(\rightarrow E)$$

$$\frac{u : \Lambda \vdash \Delta \quad v : \Gamma \vdash \Lambda}{u \circ v : \Gamma \vdash \Delta}\,(Cut_S) \qquad \frac{u : \Lambda \vdash \tau \quad v : \Gamma \vdash \Lambda}{u[v] : \Gamma \vdash \tau}\,(Cut_T)$$

Figure 2: Typing rules

To get a calculus in Church style, we also need to annotate the terms 1, $\uparrow$, *id* and $\Uparrow u$ with types agreeing with the rules of Figure 2. We shall usually write these terms without their type annotations when context permits, however.

Then, every typed term has a unique type. There is a unique way of lifting the rules of Figure 1 to the typed case so that the resulting typed calculus has the subject reduction property: see Figure 3.

(β)	$(\lambda u)v \rightarrow u[v \cdot id_{\Gamma \vdash \Gamma}]$	$(\eta \Uparrow)$	$(\Uparrow w)_{\tau, \Gamma \vdash \tau, \Delta} \rightarrow 1_{\tau, \Gamma \vdash \tau} \cdot (w \circ \uparrow_{\tau, \Gamma \vdash \Gamma})$
	where $v : \Gamma \vdash \tau$		
$([id])$	$u[id_{\Gamma \vdash \Gamma}] \rightarrow u$	$(\eta \cdot)$	$1_{\tau, \Gamma \vdash \tau} \cdot \uparrow_{\tau, \Gamma \vdash \Gamma} \rightarrow id_{\tau, \Gamma \vdash \tau, \Gamma}$
$(\circ id)$	$u \circ id_{\Gamma \vdash \Gamma} \rightarrow u$	$(\eta \cdot \circ)$	$(1_{\tau, \Gamma \vdash \tau}[u]) \cdot (\uparrow_{\tau, \Gamma \vdash \Gamma} \circ u) \rightarrow u$
$(id \circ)$	$id_{\Gamma \vdash \Gamma} \circ u \rightarrow u$	$(1 \Uparrow)$	$1_{\tau, \Delta \vdash \tau}[(\Uparrow u)_{\tau, \Gamma \vdash \tau, \Delta}] \rightarrow 1_{\tau, \Gamma \vdash \tau}$
$([])$	$(u[v])[w] \rightarrow u[v \circ w]$	$(1 \Uparrow \circ)$	$1_{\tau, \Delta \vdash \tau}[(\Uparrow u)_{\tau, \Gamma \vdash \tau, \Delta} \circ v] \rightarrow 1_{\tau, \Gamma \vdash \tau}[v]$
$(\circ)$	$(u \circ v) \circ w \rightarrow u \circ (v \circ w)$	$(\uparrow \Uparrow)$	$\uparrow_{\tau, \Delta \vdash \Delta} \circ (\Uparrow u)_{\tau, \Gamma \vdash \tau, \Delta} \rightarrow u \circ \uparrow_{\tau, \Gamma \vdash \Gamma}$
(1)	$1_{\Gamma \vdash \tau}[u \cdot v] \rightarrow u$	$(\uparrow \Uparrow \circ)$	$\uparrow_{\tau, \Delta \vdash \Delta} \circ ((\Uparrow u)_{\tau, \Gamma \vdash \tau, \Delta} \circ v) \rightarrow u \circ (\uparrow_{\tau, \Gamma \vdash \Gamma} \circ v)$
$(\uparrow)$	$\uparrow_{\Gamma \vdash \Delta} \circ (u \cdot v) \rightarrow v$	$(\Uparrow \Uparrow)$	$(\Uparrow u)_{\tau, \Lambda \vdash \tau, \Delta} \circ (\Uparrow v)_{\tau, \Gamma \vdash \tau, \Lambda} \rightarrow (\Uparrow (u \circ v))_{\tau, \Gamma \vdash \tau, \Delta}$
$(\cdot)$	$(u \cdot v) \circ w \rightarrow (u[w]) \cdot (v \circ w)$	$(\Uparrow \Uparrow \circ)$	$(\Uparrow u)_{\tau, \Lambda \vdash \tau, \Delta} \circ ((\Uparrow v)_{\tau, \Gamma \vdash \tau, \Lambda} \circ w) \rightarrow (\Uparrow (u \circ v))_{\tau, \Gamma \vdash \tau, \Delta} \circ w$
(λ)	$(\lambda u)[w] \rightarrow \lambda(u[(\Uparrow w)_{\tau, \Gamma \vdash \tau, \Delta}])$	$(\Uparrow \cdot)$	$(\Uparrow u)_{\tau, \Gamma \vdash \tau, \Delta} \circ (v \cdot w) \rightarrow v \cdot (u \circ w)$
	where $w : \Gamma \vdash \Delta,\ u : \tau, \Delta \vdash \tau'$		
(app)	$(uv)[w] \rightarrow (u[w])(v[w])$	$(\Uparrow id)$	$(\Uparrow id_{\Gamma \vdash \Gamma})_{\tau, \Gamma \vdash \tau, \Gamma} \rightarrow id_{\tau, \Gamma \vdash \tau, \Gamma}$

Figure 3: The typed reduction rules

2 Other Preliminaries

Lemma 1 *σ is a terminating rewrite system.*

Proof: We use Zantema's distribution elimination technique [Zan94]. (Refer to this paper for notations and concepts; our σ is slightly more complicated than the σ that Zantema considers, but the argument is similar.) We actually show that σ is totally terminating, i.e. that the reduction relation for σ can actually be extended to a *total* well-founded quasi-ordering on terms. Total termination implies *simple* termination, which is the fact that σ plus all rules of the form $f(\ldots, u, \ldots) \to u$, for every function symbol f, terminates.

Now consider the following rewrite system σ', which is obtained from σ by replacing $u[v]$ by $u \circ v$, uv by $u \cdot v$, id by $1 \cdot \uparrow$, merging duplicate rules, eliminating rule $(\eta\cdot)$ and adding rule $(d \Uparrow)$:

$$
\begin{array}{lll}
(d \Uparrow) & \Uparrow u \circ (v \cdot w) \to (\Uparrow u \circ v) \cdot (\Uparrow u \circ w) & (\eta \Uparrow) & \Uparrow w \to 1 \cdot (w \circ \uparrow) \\
(\circ id) & u \circ (1 \cdot \uparrow) \to u & (\eta \cdot \circ) & (1 \circ u) \cdot (\uparrow \circ u) \to u \\
(id \circ) & (1 \cdot \uparrow) \circ u \to u & (1 \Uparrow) & 1 \circ \Uparrow u \to 1 \\
& & (1 \Uparrow \circ) & 1 \circ (\Uparrow u \circ v) \to 1 \circ v \\
(\circ) & (u \circ v) \circ w \to u \circ (v \circ w) & (\uparrow\Uparrow) & \uparrow \circ \Uparrow u \to u \circ \uparrow \\
(1) & 1 \circ (u \cdot v) \to u & (\uparrow\Uparrow \circ) & \uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v) \\
(\uparrow) & \uparrow \circ (u \cdot v) \to v & (\Uparrow\Uparrow) & \Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v) \\
(\cdot) & (u \cdot v) \circ w \to (u \circ w) \cdot (v \circ w) & (\Uparrow\Uparrow \circ) & \Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w \\
(\lambda) & (\lambda u) \circ w \to \lambda(u \circ \Uparrow w) & (\Uparrow \cdot) & \Uparrow u \circ (v \cdot w) \to v \cdot (u \circ w) \\
& & (\Uparrow id) & \Uparrow (1 \cdot \uparrow) \to (1 \cdot \uparrow)
\end{array}
$$

If σ' is totally terminating, then it is clear that σ is totally terminating as well. Let indeed $>$ be any total well-founded quasi-ordering extending the rewrite relation of σ'. This induces a total well-founded quasi-ordering on $\lambda\sigma$-terms that extends all the rules in σ except $(\eta\cdot)$ (for which both sides are equivalent). Let $>'$ be the quasi-ordering such that $u >' v$ if and only if u has more occurrences of $\cdot$ than v: then the lexicographic product of $>$ and $>'$ is a well-founded total quasi-ordering extending the rewrite relation of σ.

Now call a rule *embedding* if and only if its right-hand side is homeomorphically embedded in its left-hand side (i.e., it can be simulated by a sequence of rewrite steps of the form $f(\ldots, u, \ldots) \to u$). The rules $(\circ id)$, $(id\circ)$, (1), $(\uparrow)$, $(\eta \cdot \circ)$, $(1 \Uparrow)$, $(1 \Uparrow \circ)$, $(\Uparrow id)$ are embedding. Since total termination implies simple termination, σ' is totally terminating if and only if σ' minus these rules is. Moreover, $(\Uparrow \cdot)$ can be simulated by $(d \Uparrow)$ plus embedding, so we can ignore the former for purposes of total termination. To sum up, σ' is totally terminating if and only if the following system σ'' is totally terminating:

$$
\begin{array}{llll}
(d \Uparrow) & \Uparrow u \circ (v \cdot w) \to (\Uparrow u \circ v) \cdot (\Uparrow u \circ w) & (\eta \Uparrow) & \Uparrow w \to 1 \cdot (w \circ \uparrow) \\
(\circ) & (u \circ v) \circ w \to u \circ (v \circ w) & (\uparrow\Uparrow) & \uparrow \circ \Uparrow u \to u \circ \uparrow \\
& & (\uparrow\Uparrow \circ) & \uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v) \\
& & (\Uparrow\Uparrow) & \Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v) \\
(\cdot) & (u \cdot v) \circ w \to (u \circ w) \cdot (v \circ w) & (\Uparrow\Uparrow \circ) & \Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w \\
(\lambda) & (\lambda u) \circ w \to \lambda(u \circ \Uparrow w) & &
\end{array}
$$

Now, in σ'' the rules $(d \Uparrow)$ and $(\cdot)$ are *distribution rules* for $\cdot$, and no other rule features $\cdot$ on the left-hand side. Zantema's theorem [Zan94] then states that the rewrite system σ''' obtained from σ'' by dropping these distribution rules and replacing each rule with some subterm $u \cdot v$ on the right-hand side by two rules, one with $u \cdot v$ replaced by u, the other with $u \cdot v$ replaced by v, is totally terminating if and only if σ'' is. Here is σ''':

$$
\begin{array}{ll}
\Uparrow w \to 1 & \Uparrow w \to w \circ \uparrow \\
(u \circ v) \circ w \to u \circ (v \circ w) & \uparrow \circ \Uparrow u \to u \circ \uparrow \\
\uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v) & \Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v) \\
\Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w & (\lambda u) \circ w \to \lambda(u \circ \Uparrow w)
\end{array}
$$

σ''' is then proved totally terminating by using two polynomial interpretations P_1 and P_2, ordered lexicographically, as in [HL89]; we let:

	P_1	P_2
$u \circ v$	uv	$u(v+1)$
$\Uparrow u$	u	$3u$
1	2	1
$\uparrow$	2	1
λu	$u+1$	u

Using the fact that $P_1(u) \geq 2$ for every term u, and $P_2(u) \geq 1$, a quick calculation shows that for every rule $l \to r$ but the last one (i.e., (λ)), $P_1(l) \geq P_1(r)$ and $P_2(l) > P_2(r)$, while in the case of (λ) $P_1(l) > P_1(r)$. It follows that σ is totally terminating. □

In fact, σ is locally confluent, as a Knuth-Bendix test shows, whether as the subsystem of Figure 1 or Figure 3, so it is convergent; but we shall not be interested in this property here.

3 Reduction Under Safe Contexts Terminates

We translate each $\lambda\sigma$-term with Church-style typing into a family of λ-terms, typed à la Church as well. Let $\top$ be a given arbitrary type. We assume that we have infinitely many variables of each type.

With each term variable $x_{\tau_1,\ldots,\tau_n,\cdot\vdash\tau}$ of $\lambda\sigma$, we associate a variable $\hat{x}_{\tau_1\to\ldots\to\tau_n\to\top\to\tau}$ of the λ-calculus of the indicated type. Similarly, we map each stack variable $X_{\tau_1,\ldots,\tau_n,\cdot\vdash\tau'_1,\ldots,\tau'_{n'},\cdot}$ to a list $\hat{X}_{1\ \tau_1\to\ldots\to\tau_n\to\top\to\tau'_1} :: \ldots :: \hat{X}_{n'\ \tau_1\to\ldots\to\tau_n\to\top\to\tau'_{n'}} :: \hat{X}_{n'+1\ \tau_1\to\ldots\to\tau_n\to\top\to\top}$ of λ-variables of the indicated types. (The cons operator $::$ is just a meta-linguistic way of forming lists; it is assumed to be right-associative.) We do not assume the mapping to be injective in any sense.

Our translation maps every $\lambda\sigma$-term u of type $\tau_1,\ldots,\tau_n,\cdot\vdash\tau$ to a function $[\![u]\!]$ taking a list of $n+1$ λ-terms $t_1 :: \ldots :: t_n :: t_{n+1}$ of respective types $\tau_1, \ldots, \tau_n$ and $\top$, and returns a λ-term $[\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1})$ of type τ. Similarly, our translation maps every $\lambda\sigma$-term u of type $\tau_1,\ldots,\tau_n,\cdot\vdash\tau'_1,\ldots,\tau'_{n'},\cdot$ to a function $[\![u]\!]$ taking a list of $n+1$ λ-terms $t_1 :: \ldots :: t_n :: t_{n+1}$ of respective types $\tau_1, \ldots, \tau_n$ and $\top$, and returns a list of $n'+1$ λ-terms of respective types $\tau'_1, \ldots, \tau'_{n'}$ and $\top$. The translation is given in Figure 4.

$$
\begin{array}{ll}
[\![x_{\tau_1,\ldots,\tau_n,\cdot\vdash\tau}]\!](t_1 :: \ldots :: t_n :: t_{n+1}) & = \hat{x}_{\tau_1\to\ldots\to\tau_n\to\top\to\tau}\, t_1 \ldots t_n t_{n+1} \\
[\![X_{\tau_1,\ldots,\tau_n,\cdot\vdash\tau'_1,\ldots,\tau'_{n'},\cdot}]\!](t_1 :: \ldots :: t_n :: t_{n+1}) & = (\hat{X}_{1\ \tau_1\to\ldots\to\tau_n\to\top\to\tau'_1}\, t_1 \ldots t_n t_{n+1}) \\
 & \quad :: \ldots \\
 & \quad :: (\hat{X}_{n'\ \tau_1\to\ldots\to\tau_n\to\top\to\tau'_{n'}}\, t_1 \ldots t_n t_{n+1}) \\
 & \quad :: (\hat{X}_{n'+1\ \tau_1\to\ldots\to\tau_n\to\top\to\top}\, t_1 \ldots t_n t_{n+1}) \\
[\![\lambda u]\!](s) & = \lambda z_\tau \cdot [\![u]\!](z :: s) \quad \text{where } u : \tau, \Gamma \vdash \tau' \\
[\![uv]\!](s) & = ([\![u]\!](s))([\![v]\!](s)) \\
[\![u[v]]\!](s) & = [\![u]\!]([\![v]\!](s)) \\
[\![1_{\tau,\Gamma\vdash\tau}]\!](t :: s) & = t \\
[\![u \circ v]\!](s) & = [\![u]\!]([\![v]\!](s)) \\
[\![id_{\Gamma\vdash\Gamma}]\!](s) & = s \\
[\![u \cdot v]\!](s) & = [\![u]\!](s) :: [\![v]\!](s) \\
[\![\uparrow_{\tau,\Gamma\vdash\Gamma}]\!](t :: s) & = s \\
[\![\Uparrow u]\!](t :: s) & = t :: [\![u]\!](s)
\end{array}
$$

Figure 4: Translation to the simply-typed λ-calculus

The β rule $(\lambda x \cdot t)t' \to t[t'/x]$ defines a rewrite relation on λ-terms that we again write $\longrightarrow$. Recall that the simply-typed λ-calculus terminates [Kri92]. We write $\longrightarrow^+$, resp. $\longrightarrow^*$, its (resp. reflexive) transitive closure. These notions are extended to lists: $t_1 :: \ldots :: t_n :: t_{n+1} \longrightarrow^* t'_1 :: \ldots :: t'_n :: t'_{n+1}$ if and only if $t_i \longrightarrow^* t'_i$ for every i, $1 \leq i \leq n+1$; and $t_1 :: \ldots :: t_n :: t_{n+1} \longrightarrow^+ t'_1 :: \ldots :: t'_n :: t'_{n+1}$ if in addition $t_i \longrightarrow^+ t'_i$ for some i, $1 \leq i \leq n+1$.

We define the quasi-ordering $\succeq$ and the strict ordering $\succ$ on typed $\lambda\sigma$-terms by: $u \succeq v$ if and only if u and v have the same type and $[\![u]\!](s) \longrightarrow^* [\![v]\!](s)$ for every list s of λ-terms of the right types. (Observe that $\succ$ is an ordering, in particular it is irreflexive precisely because there are λ-terms of any given type, namely variables.) Similarly, $u \succ v$ if and only if u and v have the same type and $[\![u]\!](s) \longrightarrow^+ [\![v]\!](s)$ for every list s of λ-terms of the right types. We also let $u \approx v$ if and only if $[\![u]\!](s) = [\![v]\!](s)$ for every list s of λ-terms of the right types.

The interpretation is monotonic in the following sense:

Lemma 2 *If $s \longrightarrow^* s'$, then $[\![u]\!](s) \longrightarrow^* [\![u]\!](s')$.*

Proof: We claim that for any terms $t_1, \ldots, t_n, t_{n+1}$ of the right types, for any fresh variables $x_1, \ldots, x_n, x_{n+1}$ of the right types:

$$[\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1}) = [\![u]\!](x_1 :: \ldots :: x_n :: x_{n+1})[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}]$$

where substitution is extended componentwise when u is a stack. The lemma follows immediately from the claim, while the claim is proved by structural induction on u.

If $u = v[w]$, then $[\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1}) = [\![v]\!](t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1})$, where $t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1} = [\![w]\!](t_1 :: \ldots :: t_n :: t_{n+1})$. By induction hypothesis, $t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1} = [\![w]\!](x_1 :: \ldots :: x_n :: x_{n+1})[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}]$. Let: (1) $t''_1 :: \ldots :: t''_{n'} :: t''_{n'+1}$ be $[\![w]\!](x_1 :: \ldots :: x_n :: x_{n+1})$, then for each i', $1 \leq i' \leq n'+1$: (2) $t'_{i'} = t''_{i'}[t_1/x_1, \ldots, t_n/x_n]$.

Then:

$$\begin{aligned}
&[\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1})\\
&= [\![v]\!](t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1})\\
&= [\![v]\!](x'_1 :: \ldots :: x'_{n'} :: x'_{n'+1})[t'_1/x'_1, \ldots, t'_{n'}/x'_{n'}, t'_{n'+1}/x'_{n'+1}] \quad \text{(by induction hypothesis again)}\\
&= [\![v]\!](x'_1 :: \ldots :: x'_{n'} :: x'_{n'+1})\\
&\qquad [t''_1[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}]/x'_1, \ldots, t''_{n'+1}[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}]/x'_{n'}] \quad \text{(by (2))}\\
&= [\![v]\!](x'_1 :: \ldots :: x'_{n'} :: x'_{n'+1})[t''_1/x'_1, \ldots, t''_{n'+1}/x'_{n'+1}][t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}]\\
&= [\![v]\!](t''_1 :: \ldots :: t''_{n'} :: t''_{n'+1})[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}] \quad \text{(by induction hypothesis again)}\\
&= [\![u]\!](x_1 :: \ldots :: x_n :: x_{n+1})[t_1/x_1, \ldots, t_n/x_n, t_{n+1}/x_{n+1}] \quad \text{(by (1))}
\end{aligned}$$

If $u = v \circ w$, then the argument is identical, while all other cases are trivial. □

Recall that a *context* $\mathcal{C}$ is any term with a distinguished occurrence _, which we call the *hole*. For any term t, $\mathcal{C}\{t\}$ denotes $\mathcal{C}$ with the hole replaced by t. Similarly if t is itself a context, in which case we get a new context.

The interpretation is then also monotonic in the following sense:

Lemma 3 *If $u \succeq v$, then $\mathcal{C}\{u\} \succeq \mathcal{C}\{v\}$ for any context $\mathcal{C}$. If $u \approx v$, then $\mathcal{C}\{u\} \approx \mathcal{C}\{v\}$ for any context $\mathcal{C}$.*

Proof: The case of $\approx$ is trivial, because our definition of $[\![\_]\!]$ is compositional. That is, for any context $\mathcal{C}$, there is a functional $[\![\mathcal{C}]\!]$ such that, for every u, $[\![\mathcal{C}\{u\}]\!] = [\![\mathcal{C}]\!]([\![u]\!])$: this is an easy structural induction on $\mathcal{C}$.

We prove the first claim by structural induction on the context $\mathcal{C}$ as well. The base case, when $\mathcal{C}$ is the empty context _, is clear. Otherwise, the induction case reduces to the elementary cases that whenever $u \succeq v$, then $f(\ldots, u, \ldots) \succeq f(\ldots, v, \ldots)$ for each function symbol f in the language of $\lambda\sigma$ and each argument position.

For example, if $f = \lambda$: $[\![\lambda u]\!](s) = \lambda z \cdot [\![u]\!](z :: s) \longrightarrow^* \lambda z \cdot [\![v]\!](z :: s)$ (since $u \succeq v$) $= [\![\lambda v]\!](s)$. Since s is arbitrary, $\lambda u \succeq \lambda v$.

The cases when f is the application symbol or the $\cdot$ pair operator (whatever the argument position), or the $\Uparrow$ lift operator are equally easy.

When f is the _[_] operator, we have two cases. At argument position 1, we must show that $u \succeq v$ entails $u[w] \succeq v[w]$. But this is trivial, since $[\![u[w]]\!](s) = [\![u]\!]([\![w]\!](s)) \longrightarrow^* [\![v]\!]([\![w]\!](s))$ (by assumption) $= [\![v[w]]\!](s)$. At argument position 2, we must show that $u \succeq v$ entails $w[u] \succeq w[v]$. But $[\![w[u]]\!](s) = [\![w]\!]([\![u]\!](s)) \longrightarrow^* [\![w]\!]([\![v]\!](s))$ (since $u \succeq v$ and by Lemma 2) $= [\![w[v]]\!](s)$. The case of $\circ$ is entirely analogous. □

Observe that the interpretation is not strictly monotonic, in that $u \succ v$ does not imply $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$, only $\mathcal{C}\{u\} \succeq \mathcal{C}\{v\}$: consider the context $\mathcal{C} = \uparrow \circ (\_ \cdot w)$ for some w. This leads to the following definition:

Definition 1 *A context $\mathcal{C}$ is called* safe *if and only if $u \succ v$ implies $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$ for every terms u and v of the same type such that $\mathcal{C}\{u\}$ and $\mathcal{C}\{v\}$ are typable.*

We shall see in Section 4 that there are enough safe contexts to all typed $\lambda\sigma$-terms to be reduced to normal form by reductions under safe contexts.

Lemma 4 *For each rule $l \to r$ except (β) in Figure 3, $l \approx r$. For rule (β), $l \succ r$.*

Proof: We omit all type information unless strictly necessary.

Rule (β):

$$\begin{aligned} &[\![l]\!](s) \\ &= ([\![\lambda u]\!](s))([\![v]\!](s)) \\ &= (\lambda z \cdot [\![u]\!](z :: s))([\![v]\!](s)) \\ &\longrightarrow ([\![u]\!](z :: s))[[\![v]\!](s)/z] \\ &= [\![u]\!]([\![v]\!](s) :: s) \end{aligned}$$

by Lemma 3, while $[\![r]\!](s) = [\![u]\!]([\![v \cdot id]\!](s)) = [\![u]\!]([\![v]\!](s) :: s)$.

The cases of rules $([id])$, $(\circ id)$ and $(id\circ)$ are obvious: $[\![l]\!](s) = [\![r]\!](s) = [\![u]\!](s)$ in all three cases.

Rule $([])$: $[\![l]\!](s) = [\![u[v]]\!]([\![w]\!](s)) = [\![u]\!]([\![v]\!]([\![w]\!](s))) = [\![u]\!]([\![v \circ w]\!](s)) = [\![u[v \circ w]]\!](s)$, and similarly for rule $(\circ)$.

Rule (1): $[\![l]\!](s) = [\![1]\!]([\![u \cdot v]\!](s)) = [\![1]\!]([\![u]\!](s) :: [\![v]\!](s)) = [\![u]\!](s) = [\![r]\!](s)$.

Rule $(\uparrow)$: $[\![l]\!](s) = [\![\uparrow]\!]([\![u \cdot v]\!](s)) = [\![\uparrow]\!]([\![u]\!](s) :: [\![v]\!](s)) = [\![v]\!](s) = [\![r]\!](s)$.

Rule $(\cdot)$: $[\![l]\!](s) = [\![u \cdot v]\!]([\![w]\!](s)) = [\![u]\!]([\![w]\!](s)) :: [\![v]\!]([\![w]\!](s)) = [\![u[w]]\!](s) :: [\![v \circ w]\!](s) = [\![(u[w]) \cdot (v \circ w)]\!](s) = [\![r]\!](s)$.

Rule (λ): $[\![l]\!](s) = [\![\lambda u]\!]([\![w]\!](s)) = \lambda z_\tau \cdot [\![u]\!](z :: [\![w]\!](s))$, while $[\![r]\!](s) = \lambda z_\tau \cdot [\![u[\Uparrow w]]\!](z :: s) = \lambda z_\tau \cdot [\![u]\!]([\![\Uparrow w]\!](z :: s)) = \lambda z_\tau \cdot [\![u]\!](z :: [\![w]\!](s))$.

Rule (app): similar argument as for rule $(\cdot)$.

Rule $(\eta \Uparrow)$: $[\![l]\!](t :: s) = t :: [\![w]\!](s)$, while $[\![r]\!](t :: s) = [\![1]\!](t :: s) :: [\![w\circ \uparrow]\!](t :: s) = t :: [\![w\circ \uparrow]\!](t :: s) = t :: [\![w]\!]([\![\uparrow]\!](t :: s)) = t :: [\![w]\!](s)$.

Rule $(\eta\cdot)$: $[\![l]\!](t :: s) = [\![1]\!](t :: s) :: [\![\uparrow]\!](t :: s) = t :: s = [\![r]\!](t :: s)$; and $t :: s$ is the most general form of argument to $[\![l]\!]$ by typing, so $l \approx r$.

Rule $(\eta \circ)$: $l = (1[u]) \cdot (\uparrow \circ u) \approx (1 \cdot \uparrow) \circ u$ (by case $([])$) $\approx id \circ u$ (by case $(\eta\cdot)$) $\approx u = r$ (by case $(id\circ)$). Observe that we have used Lemma 3 (the $\approx$ part) implicitly throughout. All the other rules are dealt with similarly:

Rule $(1 \Uparrow)$: $l = 1[\Uparrow u] \approx 1[1 \cdot (u\circ \uparrow)]$ (by $(\eta \Uparrow)$) $\approx 1 = r$ (by (1)).

Rule $(1 \Uparrow \circ)$: $l = 1[\Uparrow u \circ v] \approx 1[(1 \cdot (u\circ \uparrow)) \circ v]$ (by $(\eta \Uparrow)$) $\approx 1[(1[v]) \cdot ((u\circ \uparrow) \circ v)]$ (by $(\cdot)$) $\approx 1[v] = r$ (by (1)).

Rule $(\uparrow\Uparrow)$: $l = \uparrow \circ \Uparrow u \approx \uparrow \circ(1 \cdot (u\circ \uparrow))$ (by $(\eta \Uparrow)$) $\approx u\circ \uparrow = r$ (by $(\uparrow)$).

Rule $(\uparrow\Uparrow \circ)$: $l = \uparrow \circ(\Uparrow u \circ v) \approx \uparrow \circ((1 \cdot (u\circ \uparrow)) \circ v)$ (by $(\eta \Uparrow)$) $\approx \uparrow \circ((1[v]) \cdot ((u\circ \uparrow) \circ v))$ (by $(\cdot)$) $\approx (u\circ \uparrow) \circ v$ (by $(\uparrow)$) $\approx u \circ (\uparrow \circ v) = r$ (by $(\circ)$).

Rule $(\Uparrow\Uparrow)$: $l = \Uparrow u\circ \Uparrow v \approx (1 \cdot (u\circ \uparrow))\circ \Uparrow v$ (by $(\eta \Uparrow)$) $\approx (1[\Uparrow v]) \cdot ((u\circ \uparrow)\circ \Uparrow v)$ (by $(\cdot)$) $\approx 1 \cdot ((u\circ \uparrow)\circ \Uparrow v)$ (by $(1 \Uparrow)$) $\approx 1 \cdot (u \circ (\uparrow \circ \Uparrow v))$ (by $(\circ)$) $\approx 1 \cdot (u \circ (v\circ \uparrow))$ (by $(\uparrow\Uparrow)$) $\approx 1 \cdot ((u \circ v)\circ \uparrow)$ (by $(\circ)$ used in the opposite direction) $\approx \Uparrow (u \circ v) = r$ (by $(\eta \Uparrow)$ used in the opposite direction).

Rule $(\Uparrow\Uparrow \circ)$: $l = \Uparrow u \circ (\Uparrow v \circ w) \approx (\Uparrow u\circ \Uparrow v) \circ w$ (by $(\circ)$ used in the opposite direction) $\approx \Uparrow (u \circ v) \circ w = r$ (by $(\Uparrow\Uparrow)$).

Rule $(\Uparrow \cdot)$: $l = \Uparrow u \circ (v \cdot w) \approx (1 \cdot (u\circ \uparrow)) \circ (v \cdot w)$ (by $(\eta \Uparrow)$) $\approx (1[v \cdot w]) \cdot ((u\circ \uparrow) \circ (v \cdot w))$ (by $(\cdot)$) $\approx v \cdot ((u\circ \uparrow) \circ (v \cdot w))$ (by (1)) $\approx v \cdot (u \circ (\uparrow \circ(v \cdot w)))$ (by $(\circ)$) $\approx v \cdot (u \circ w) = r$ (by $(\uparrow)$).

Rule $(\Uparrow id)$: $l = \Uparrow id \approx 1 \cdot (id\circ \uparrow)$ (by $(\eta \Uparrow)$) $\approx 1\cdot \uparrow$ (by $(id\circ)$) $\approx id = r$ (by $(\eta\cdot)$). $\Box$

It follows:

Theorem 5 (Main Theorem) *Every reduction in the typed $\lambda\sigma$-calculus, where every contracted (β)-redex occurs under a safe context, is finite.*

Proof: Any step in such a reduction is either a (β)-contraction $\mathcal{C}\{(\lambda u)v\} \longrightarrow \mathcal{C}\{u[v \cdot id]\}$, then $\mathcal{C}\{(\lambda u)v\} \succ \mathcal{C}\{u[v \cdot id]\}$ by Lemma 4 and the fact that $\mathcal{C}$ is safe; or a σ-contraction $u \longrightarrow_\sigma v$, where $u \approx v$ by Lemma 4 and Lemma 3. Therefore each rewrite step is strictly decreasing in the lexicographic product of $\succ$ and $\longrightarrow_\sigma^+$, which is well-founded since the typed λ-calculus terminates (for $\succ$) and σ terminates (by Lemma 1). □

4 Some Safe Contexts

It remains to produce some non-trivial safe contexts:

Definition 2 *Let the* syntactically safe contexts *be defined as those in* $\mathcal{S}_T \cup \mathcal{S}_S$, *where:*

$$
\begin{array}{lll}
\mathcal{S}_T & ::= & \_ \mid \lambda \mathcal{S}_T \mid \mathcal{S}_T T \mid T \mathcal{S}_T \mid \mathcal{S}_T[S] \mid \mathcal{V}_T[\mathcal{S}_S] \mid 1[\mathcal{S}'_S] \\
\mathcal{S}_S & ::= & \mathcal{S}'_S \mid \mathcal{S}_T \cdot S \mid T \cdot \mathcal{S}_S \mid \Uparrow \mathcal{S}_S \mid \Uparrow S \circ \mathcal{S}'_S \mid \mathcal{S}_S \circ S \\
\mathcal{S}'_S & ::= & \mathcal{V}_S \circ \mathcal{S}_S \mid \uparrow \circ \mathcal{S}'_S \mid \mathcal{S}'_S \circ S
\end{array}
$$

Before we show Lemma 7, we need the following technical lemma. Let tl be defined by $tl(t :: s) = s$, and let tl^n be defined by: $tl^0(s) = s$, $tl^{n+1} = tl \circ tl^n$.

Lemma 6 *For every context $\mathcal{C}'$ in $\mathcal{S}'_S$, there is a stack variable X, a proper subcontext $\mathcal{C}$ of $\mathcal{C}'$ in $\mathcal{S}_S$ and an integer $m \in \mathbb{N}$ such that, for every term u such that $\mathcal{C}'\{u\}$ is well-typed, for every s' of the right type, for some s, $[\![\mathcal{C}'\{u\}]\!](s') = tl^m([\![X]\!]([\![\mathcal{C}\{u\}]\!](s)))$.*

Proof: By structural induction on $\mathcal{C}'$. This is obvious if $\mathcal{C}'$ has the form $X \circ \mathcal{C}$, with $\mathcal{C} \in \mathcal{S}_S$: $m = 0$ and $s = s'$.

If $\mathcal{C}' = \uparrow \circ \mathcal{C}''$ with $\mathcal{C}'' \in \mathcal{S}'_S$, then for every u such that $\mathcal{C}'\{u\}$ is well-typed, in particular $\mathcal{C}''\{u\}$ is well-typed, so by induction hypothesis $[\![\mathcal{C}''\{u\}]\!](s') = tl^m([\![X]\!]([\![\mathcal{C}\{u\}]\!](s)))$ for some $m \geq 0$ and some s. Then $[\![\mathcal{C}'\{u\}]\!](s') = [\![\uparrow]\!]([\![\mathcal{C}''\{u\}]\!](s')) = tl([\![\mathcal{C}''\{u\}]\!](s')) = tl^{m+1}([\![X]\!]([\![\mathcal{C}\{u\}]\!](s)))$.

If $\mathcal{C}' = \mathcal{C}'' \circ w$, where $\mathcal{C}'' \in \mathcal{S}'_S$, then for every u such that $\mathcal{C}'\{u\}$ is well-typed, in particular $\mathcal{C}''\{u\}$ is well-typed, so by induction hypothesis, for every s'_1 of the right type, $[\![\mathcal{C}''\{u\}]\!](s'_1) = tl^m([\![X]\!]([\![\mathcal{C}\{u\}]\!](s_1)))$ for some $m \geq 0$ and some s_1. Then $[\![\mathcal{C}'\{u\}]\!](s') = [\![\mathcal{C}''\{u\}]\!]([\![w]\!](s'))$; letting s'_1 be $[\![w]\!](s')$, $[\![\mathcal{C}'\{u\}]\!](s') = [\![\mathcal{C}''\{u\}]\!](s'_1) = tl^m([\![X]\!]([\![\mathcal{C}\{u\}]\!](s_1)))$. □

Lemma 7 *Every syntactically safe context is safe.*

Proof: We have to show that if $u \succ v$, then $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$ for any syntactically safe context $\mathcal{C}$. This is by structural induction on $\mathcal{C}$, using Definition 1. The base case, when $\mathcal{C}$ is the empty context $\_$, is clear. Otherwise:

If $\mathcal{C} = \lambda \mathcal{C}_1$, where $\mathcal{C}_1 \in \mathcal{S}_T$, then $[\![\mathcal{C}\{u\}]\!](s) = \lambda z \cdot [\![\mathcal{C}_1\{u\}]\!](z :: s) \longrightarrow^+ \lambda z \cdot [\![\mathcal{C}_1\{v\}]\!](z :: s) = [\![\mathcal{C}\{v\}]\!](s)$ by induction hypothesis. Similarly when $\mathcal{C} = \mathcal{C}_1 w$ or $\mathcal{C} = w\mathcal{C}_1$, or $\mathcal{C} = \mathcal{C}_1 \cdot w$ for some term w and some $\mathcal{C}_1 \in \mathcal{S}_T$, or $\mathcal{C} = \Uparrow \mathcal{C}_1$ or $\mathcal{C} = w \cdot \mathcal{C}_1$ for some term w and some $\mathcal{C}_1 \in \mathcal{S}_S$.

If $\mathcal{C} = \mathcal{C}_1[w]$ for some $\mathcal{C}_1 \in \mathcal{S}_T$ and some stack w, then $[\![\mathcal{C}\{u\}]\!](s) = [\![\mathcal{C}_1\{u\}]\!]([\![w]\!](s)) \longrightarrow^+ [\![\mathcal{C}_1\{v\}]\!]([\![w]\!](s)) = [\![\mathcal{C}\{v\}]\!](s)$ by induction hypothesis. Similarly when $\mathcal{C} = \mathcal{C}_1 \circ w$ for some $\mathcal{C}_1$ in $\mathcal{S}_S$ and some stack w.

If $\mathcal{C} = x[\mathcal{C}_1]$ where x is an term variable and $\mathcal{C}_1 \in \mathcal{S}_S$, then $[\![\mathcal{C}\{u\}]\!](s) = \hat{x} t_1 \ldots t_n t_{n+1}$ where $t_1 :: \ldots :: t_n :: t_{n+1} = [\![\mathcal{C}_1\{u\}]\!](s) \longrightarrow^+ [\![\mathcal{C}_1\{v\}]\!](s)$ by induction hypothesis. Let $(t'_1, \ldots, t'_n, t'_{n+1})$ be $[\![\mathcal{C}_1\{v\}]\!](s)$: this means that $t_i \longrightarrow^* t'_i$ for every i, $1 \leq i \leq n+1$, and that $t_i \longrightarrow^+ t'_i$ for some i. So $\hat{x} t_1 \ldots t_n t_{n+1} \longrightarrow^+ \hat{x} t'_1 \ldots t'_n t'_{n+1} = [\![\mathcal{C}\{v\}]\!](s)$.

It remains to deal with the cases when $\mathcal{C}$ is in $\mathcal{S}'_S$, and when $\mathcal{C}$ is of the form $1[\mathcal{C}']$ or $\Uparrow w \circ \mathcal{C}'$ with $\mathcal{C}' \in \mathcal{S}'_S$.

Consider first the case where $\mathcal{C}$ is some context $\mathcal{C}'$ in $\mathcal{S}'_S$. By Lemma 6, there is a stack variable X, a proper subcontext $\mathcal{C}_1$ in $\mathcal{S}_S$ and an integer $m \geq 0$ such that for every u of the right type, for every s' of the right type, there is a λ-term s such that $[\![\mathcal{C}'\{u\}]\!](s') = tl^m([\![X]\!]([\![\mathcal{C}_1\{u\}]\!](s)))$. Let X be $X_{\Gamma \vdash \tau_1, \ldots, \tau_p}$, where by typing $p \geq m$. Let also $t_1 :: \ldots :: t_n :: t_{n+1}$ be $[\![\mathcal{C}_1\{u\}]\!](s)$, and $t'_1 :: \ldots :: t'_n :: t'_{n+1}$ be $[\![\mathcal{C}_1\{v\}]\!](s)$. Then:

$$
\begin{aligned}
&[\![\mathcal{C}'\{u\}]\!](s') \\
&= tl^m \left((\hat{X}_1 t_1 \ldots t_n t_{n+1}) :: \ldots :: (\hat{X}_m t_1 \ldots t_n t_{n+1}) :: (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: \ldots :: (\hat{X}_{p+1} t_1 \ldots t_n t_{n+1})\right) \\
&= (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: \ldots :: (\hat{X}_{p+1} t_1 \ldots t_n t_{n+1}) \\
&\longrightarrow^+ (\hat{X}_{m+1} t'_1 \ldots t'_n t'_{n+1}) :: \ldots :: (\hat{X}_{p+1} t'_1 \ldots t'_n t'_{n+1}) \quad \text{(by induction hypothesis)} \\
&= tl^m \left((\hat{X}_1 t'_1 \ldots t'_n t'_{n+1}) :: \ldots :: (\hat{X}_m t'_1 \ldots t'_n t'_{n+1}) :: (\hat{X}_{m+1} t'_1 \ldots t'_n t'_{n+1}) :: \ldots :: (\hat{X}_{p+1} t'_1 \ldots t'_n t'_{n+1})\right) \\
&= [\![\mathcal{C}'\{v\}]\!](s')
\end{aligned}
$$

so $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$.

If $\mathcal{C} = 1[\mathcal{C}']$, then taking the same notation as above:

$$
\begin{aligned}
&[\![\mathcal{C}\{u\}]\!](s') \\
&= \hat{X}_{m+1} t_1 \ldots t_n t_{n+1} \\
&\longrightarrow^+ \hat{X}_{m+1} t'_1 \ldots t'_n t'_{n+1} \quad \text{(by induction hypothesis)} \\
&= [\![\mathcal{C}\{v\}]\!](s')
\end{aligned}
$$

(observe that now $p > m$ by typing) so again $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$.

And if $\mathcal{C} = \Uparrow w \circ \mathcal{C}'$, then:

$$
\begin{aligned}
&[\![\mathcal{C}\{u\}]\!](s') \\
&= (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{u\}]\!](s')) \\
&\longrightarrow^* (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{v\}]\!](s')) \quad \text{(by Lemma 3)} \\
&\longrightarrow^+ (\hat{X}_{m+1} t'_1 \ldots t'_n t'_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{v\}]\!](s')) \quad \text{(by induction hypothesis)} \\
&= [\![\mathcal{C}\{v\}]\!](s')
\end{aligned}
$$

so $\mathcal{C}\{u\} \succ \mathcal{C}\{v\}$. $\square$

Lemma 8 *Let $\mathcal{C}\{u\}$ be a σ-normal simply-typed term, where $u \in T$. Then $\mathcal{C}$ is a syntactically safe context.*

Proof: Recall that the σ-normal forms ([Río93], p.76) are all elements of the languages described by the following grammar:

$$
\begin{array}{ll}
\text{In } T\text{:} & t ::= 1 \mid \lambda t \mid tt \mid \mathcal{V}_T \mid \mathcal{V}_T[s] \mid 1[s'] \\
\text{In } S\text{:} & s ::= s' \mid id \mid t \cdot s \\
& s' ::= \uparrow \mid \mathcal{V}_S \mid \mathcal{V}_S \circ s \mid \uparrow \circ s'
\end{array}
$$

(Although our σ is different from Ríos', it is easy to see that it has exactly the same normal forms.) Therefore the contexts having a hole accepting terms of T are elements of the languages defined by the grammar:

$$
\begin{array}{ll}
\text{In } T\text{:} & C_t ::= \_ \mid \lambda C_t \mid C_t t \mid t C_t \mid C_t[s] \mid \mathcal{V}_T[C_s] \mid 1[C'_s] \\
\text{In } S\text{:} & C_s ::= C'_s \mid C_t \cdot s \mid t \cdot C_s \\
& C'_s ::= \mathcal{V}_S \circ C_s \mid \uparrow \circ C'_s
\end{array}
$$

which clearly defines sublanguages of $\mathcal{S}_T$, $\mathcal{S}_S$ and $\mathcal{S}'_S$ respectively, proving the claim. $\square$

Corollary 9 *In the simply-typed $\lambda\sigma$-calculus, every σ-eager rewrite is finite.*

Proof: In every σ-eager rewrite, every (β) is performed under a syntactically safe context by Lemma 8. This context is safe by Lemma 7, so Theorem 5 applies. $\square$

We can do the same thing with the $\lambda\sigma_\Uparrow$-calculus and $\sigma_\Uparrow$-eager rewrites:

Lemma 10 *Let $\mathcal{C}\{u\}$ be a $\sigma_\Uparrow$-normal simply-typed term, where $u \in T$. Then $\mathcal{C}$ is a syntactically safe context.*

Proof: An easy argument, similar to those found in [Río93], shows that the $\sigma_{\Uparrow}$-normal forms are all in the languages described by the following grammar:

$$\begin{array}{ll} \text{In } T\text{:} & t ::= 1 \mid \lambda t \mid tt \mid \mathcal{V}_T \mid \mathcal{V}_T[s] \mid 1[s'] \\ \text{In } S\text{:} & s ::= s' \mid id \mid t \cdot s \mid \Uparrow s \mid \Uparrow s \circ s' \\ & s' ::= \uparrow \mid \mathcal{V}_S \mid \mathcal{V}_S \circ s \mid \uparrow \circ s' \end{array}$$

Indeed, we show by structural induction on the $\sigma_{\Uparrow}$-normal term u that: (1) if u is in T, then u is in language t, (2) if u is a stack, then u is in language s, (3) if u is a stack and $1[u]$ is $\sigma_{\Uparrow}$-normal, then u is in language s', (4) if u is a stack and $\uparrow \circ u$ is $\sigma_{\Uparrow}$-normal, then u is in language s', and (5) if u is a stack and $\Uparrow v \circ u$ is $\sigma_{\Uparrow}$-normal, then u is in language s'.

First, observe that (2) implies (3): since $1[u]$ is $\sigma_{\Uparrow}$-normal, u cannot be id (by rule $([id])$), a cons $v \cdot w$ (rule (1) would be applicable) or a lift $\Uparrow v$ (rule $(1 \Uparrow)$ would be applicable), or of the form $\Uparrow v \circ w$ (rule $(1 \Uparrow \circ)$ would be applicable).

Similarly, (2) implies (4), because of rules $(\circ id)$, $(\uparrow)$, $(\uparrow\uparrow)$ and $(\uparrow\Uparrow \circ)$. And (2) implies (5) because of rules $(\circ id)$, $(\Uparrow \cdot)$, $(\Uparrow\Uparrow)$ and $(\Uparrow\Uparrow \circ)$.

We now show (2). Let u be a $\sigma_{\Uparrow}$-normal stack. If u is in $\mathcal{V}_S$ or is $\uparrow$, then u is clearly in s', hence in s. If $u = id$ or if u is a cons $v \cdot w$ or a lift $\Uparrow v$, then u is also in s. Finally, if u is a composition $v \circ w$, then in particular v is in s; but v cannot be id (rule $(id\circ)$ would apply), or a cons $v_1 \cdot v_2$ (rule $(\cdot)$ would apply), or a composition $v_1 \circ v_2$ (rule $(\circ)$ would apply); so v must be of the form $\Uparrow v_1$ (then u has the form $\Uparrow v_1 \circ w$ with $v_1 \in s$ and $w \in s$ by induction hypothesis; since (2) implies (5), w is actually in s', so u is in s), or $v = \uparrow$ (then $u = \uparrow \circ w$, where $w \in s$ by induction hypothesis; since (2) implies (4), w is actually in s', so u is in s', hence in s), or v is a variable X in $\mathcal{V}_S$ (then $u = X \circ w$ with $w \in s$ by induction hypothesis, so u is in s', hence in s).

We now show (1). Let u be a $\sigma_{\Uparrow}$-normal term in T. If u is a variable in $\mathcal{V}_T$, an abstraction λv, an application vw or 1, then u is in t. Finally, if u is of the form $v[w]$, then v must be a variable in $\mathcal{V}_T$ or be equal to 1: the other cases would be reducible by rules (λ), (app) or $([])$. Moreover, by induction hypothesis w is in s. If v is a variable x in $\mathcal{V}_T$, then $u = x[w]$ is in t. If $v = 1$, then $u = 1[w]$ where w is in s, and since (2) implies (3), w is actually in s'; so u is in t again.

It follows that the contexts having a hole accepting terms in T are in the languages defined by the grammar:

$$\begin{array}{ll} \text{In } T\text{:} & C_t ::= \_ \mid \lambda C_t \mid C_t t \mid t C_t \mid C_t[s] \mid \mathcal{V}_T[C_s] \mid 1[C'_s] \\ \text{In } S\text{:} & C_s ::= C'_s \mid C_t \cdot s \mid t \cdot C_s \mid \Uparrow C_s \mid \Uparrow C_s \circ s' \mid \Uparrow s \circ C'_s \\ & C'_s ::= \mathcal{V}_S \circ C_s \mid \uparrow \circ C'_s \end{array}$$

and it is easy to see that C_t, C_s, C'_s are respective sublanguages of $\mathcal{S}_T$, $\mathcal{S}_S$ and $\mathcal{S}'_S$. □

It follows:

Corollary 11 *In the simply-typed $\lambda\sigma_{\Uparrow}$-calculus, every $\sigma_{\Uparrow}$-eager rewrite is finite.*

Proof: In every $\sigma_{\Uparrow}$-eager rewrite, every (β) is performed under a syntactically safe context by Lemma 10. This context is safe by Lemma 7, so Theorem 5 applies. □

References

[ACCL90] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages*, pages 31–46, San Francisco, California, January 1990.

[GL96] Jean Goubault-Larrecq. On computational interpretations of the modal logic S4 II. The λ**ev**Q-calculus. Technical report, University of Karlsruhe, 1996. Available on `ftp://theory.doc.ic.ac.uk/theory/guests/GoubaultJ/`.

[GL97] Jean Goubault-Larrecq. On computational interpretations of the modal logic S4 IIIb. Confluence and conservativity of the λ**ev**Q_H-calculus. Technical report, Inria, 1997.

[HL89] Thérèse Hardin and Jean-Jacques Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, December 1989.

[Kri92] Jean-Louis Krivine. *Lambda-calcul, types et modèles.* Masson, 1992.

[Mel94] Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Proceedings of the CONFER workshop*, München, April 1994.

[Mel95] Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *2nd International Conference on Typed Lambda-Calculi and Applications (TLCA'95)*, pages 328–334, Edinburgh, UK, April 1995. Springer Verlag LNCS 902.

[Río93] Alejandro Ríos. *Contributions à l'étude des lambda-calculs avec substitutions explicites.* PhD thesis, École Normale Supérieure, December 1993.

[Zan94] Hans Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105,
78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399