# A Proof of Weak Termination of Typed $\lambda\sigma$-Calculi

Jean Goubault-Larrecq

G.I.E. Dyade et projet Coq, I.N.R.I.A.

**Abstract.** We show that reducing any simply-typed $\lambda\sigma$-term (resp. $\lambda\sigma_{\Uparrow}$) by applying the rules in $\sigma$ (resp. $\sigma_{\Uparrow}$) eagerly always terminates, by a translation to the simply-typed $\lambda$-calculus. This holds even with term and substitution meta-variables. In fact, every reduction terminates provided that $(\beta)$-redexes are only contracted under so-called safe contexts; and in $\sigma$, resp. $\sigma_{\Uparrow}$-normal forms, all contexts around terms of sort $T$ are safe. The result is then extended to second-order type systems.

## 1 Introduction

The simply-typed $\lambda\sigma$-calculus does not terminate strongly [8, 9], but it terminates in the weak sense: every typed term has a normal form. We present a proof of this; in fact, we prove the widely believed claim that every reduction where $\sigma$ steps are applied eagerly is finite, as a consequence of a more general theorem stating that every reduction where $(\beta)$-contraction only occurs under so-called *safe contexts* terminates. For the sake of generality, we prove this result even in the presence of term and substitution metavariables, and for a general calculus that encompasses both the original $\lambda\sigma$-calculus [1] and the $\lambda\mathcal{E}$-calculus of [5].

Related works on this subject include [12], which proves the analogue of $\sigma$-eager termination for a left-linear cousin of $\lambda\sigma$ named $\lambda_{\phi}$, using reducibility techniques —the same technique applies to $\lambda\sigma$ as well— and [3] which shows that several related calculi, among which the simply-typed $\lambda\zeta$ and $\lambda\upsilon$-calculi [10, 7], are strongly normalizing in the simply-typed case (see also [2] for $\lambda\upsilon$).

Plan: We recapitulate some basic definitions in Section 2 and prove a few preliminaries in Section 3. In Section 4, we prove our main theorem: every reduction where $(\beta)$-contraction occurs only under so-called *safe* contexts always terminates. This result is applied in Section 5 to show that all normalization strategies that apply $\sigma$ (resp. $\sigma_{\Uparrow}$) eagerly in the $\lambda\sigma$-calculus (resp. in $\lambda\sigma_{\Uparrow}$, a.k.a. $\lambda\mathcal{E}$) terminate. The method is easily carried over to Ríos' $\lambda\tau$-calculus to show a similar result in Section 6, and is extended to second-order type systems in Section 7.

## 2 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:

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

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; $\sigma$ denotes the subsystem without $(\beta)$. It is a slight generalization, in the sense that it can simulate every reduction, of both the $\lambda\sigma$-calculus of [1] and of the $\lambda\sigma_{\Uparrow}$-calculus of [5] (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 $\eta$ in their names; and $\sigma_{\Uparrow}$ denotes $\lambda\sigma_{\Uparrow}$ minus $(\beta)$.

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

**Fig. 1.** The $\lambda\sigma_{\Uparrow}$-calculus

The purpose of this paper is in particular to show that the simply-typed $\lambda\sigma$-calculus terminates, when restricted to *$\sigma$-eager rewrites*, i.e. rewrite sequences where $(\beta)$-redexes are only contracted inside $\sigma$-normal terms. In particular, because $\sigma$ terminates (see Lemma 1), all $\sigma$-eager rewrites are of the form:

$$u \xrightarrow{\sigma}{}^* u_1 \xrightarrow{(\beta)} u_1' \xrightarrow{\sigma}{}^* u_2 \xrightarrow{(\beta)} u_2' \xrightarrow{\sigma}{}^* \dots \xrightarrow{\sigma}{}^* u_k \xrightarrow{(\beta)} u_k' \xrightarrow{\sigma}{}^* \dots$$

where $u_1$, $u_2$, ..., $u_k$, ... are $\sigma$-normal. We shall also show 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 $\tau$ is a formula and $\Gamma$ is a finite list of formulas $\tau_1, \dots, \tau_n, \cdot$, where $\cdot$ marks the end of the list; formulas $\tau$ are either atoms $\alpha$, $\beta$, ..., or implications (arrow types) $\tau_1 \to \tau_2$. Stack types are sequents $\Gamma \vdash \Delta$, with $\Delta$ 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 $\lambda$-terms in Section 4; otherwise, we would need to translate typing derivations of $\lambda\sigma$-terms to typing derivations of $\lambda$-terms, which is not that different, but is much less readable.)

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

$$\frac{u : \Gamma \vdash \tau \quad v : \Gamma \vdash \Delta}{u \cdot v : \Gamma \vdash \tau, \Delta}\,(,I) \qquad \frac{}{1 : \tau, \Gamma \vdash \tau}\,(,E_1) \qquad \frac{}{\uparrow : \tau, \Gamma \vdash \Gamma}\,(,E_2)$$

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

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

**Fig. 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.

| | | | |
|---|---|---|---|
| $(\beta)$ | $(\lambda u)v \to u[v \cdot id_{\Gamma\vdash\Gamma}]$ | $(\eta \Uparrow)$ | $(\Uparrow w)_{\tau,\Gamma\vdash\tau,\Delta} \to 1_{\tau,\Gamma\vdash\tau} \cdot (w \circ \uparrow_{\tau,\Gamma\vdash\Gamma})$ |
| | where $v : \Gamma \vdash \tau$ | $(\eta\cdot)$ | $1_{\tau,\Gamma\vdash\tau} \cdot \uparrow_{\tau,\Gamma\vdash\Gamma} \to id_{\tau,\Gamma\vdash\tau,\Gamma}$ |
| $([id])$ | $u[id_{\Gamma\vdash\Gamma}] \to u$ | $(\eta \cdot \circ)$ | $(1_{\tau,\Gamma\vdash\tau}[u]) \cdot (\uparrow_{\tau,\Gamma\vdash\Gamma} \circ u) \to u$ |
| $(\circ id)$ | $u \circ id_{\Gamma\vdash\Gamma} \to u$ | $(1 \Uparrow)$ | $1_{\tau,\Delta\vdash\tau}[(\Uparrow u)_{\tau,\Gamma\vdash\tau,\Delta}] \to 1_{\tau,\Gamma\vdash\tau}$ |
| $(id\circ)$ | $id_{\Gamma\vdash\Gamma} \circ u \to u$ | $(1 \Uparrow \circ)$ | $1_{\tau,\Delta\vdash\tau}[(\Uparrow u)_{\tau,\Gamma\vdash\tau,\Delta} \circ v] \to 1_{\tau,\Gamma\vdash\tau}[v]$ |
| $([])$ | $(u[v])[w] \to u[v \circ w]$ | $(\uparrow\Uparrow)$ | $\uparrow_{\tau,\Delta\vdash\Delta} \circ (\Uparrow u)_{\tau,\Gamma\vdash\tau,\Delta} \to u \circ \uparrow_{\tau,\Gamma\vdash\Gamma}$ |
| $(\circ)$ | $(u \circ v) \circ w \to u \circ (v \circ w)$ | $(\uparrow\Uparrow \circ)$ | $\uparrow_{\tau,\Delta\vdash\Delta} \circ ((\Uparrow u)_{\tau,\Gamma\vdash\tau,\Delta} \circ v) \to u \circ (\uparrow_{\tau,\Gamma\vdash\Gamma} \circ v)$ |
| $(1)$ | $1_{\Gamma\vdash\tau}[u \cdot v] \to u$ | $(\Uparrow\Uparrow)$ | $(\Uparrow u)_{\tau,\Xi\vdash\tau,\Delta} \circ (\Uparrow v)_{\tau,\Gamma\vdash\tau,\Xi}$ |
| $(\uparrow)$ | $\uparrow_{\Gamma\vdash\Delta} \circ (u \cdot v) \to v$ | | $\to (\Uparrow (u \circ v))_{\tau,\Gamma\vdash\tau,\Delta}$ |
| $(\cdot)$ | $(u \cdot v) \circ w \to (u[w]) \cdot (v \circ w)$ | $(\Uparrow\Uparrow \circ)$ | $(\Uparrow u)_{\tau,\Xi\vdash\tau,\Delta} \circ ((\Uparrow v)_{\tau,\Gamma\vdash\tau,\Xi} \circ w)$ |
| $(\lambda)$ | $(\lambda u)[w] \to \lambda(u[(\Uparrow w)_{\tau,\Gamma\vdash\tau,\Delta}])$ | | $\to (\Uparrow (u \circ v))_{\tau,\Gamma\vdash\tau,\Delta} \circ w$ |
| | where $w : \Gamma \vdash \Delta,\ u : \tau, \Delta \vdash \tau'$ | $(\Uparrow \cdot)$ | $(\Uparrow u)_{\tau,\Gamma\vdash\tau,\Delta} \circ (v \cdot w) \to v \cdot (u \circ w)$ |
| $(app)$ | $(uv)[w] \to (u[w])(v[w])$ | $(\Uparrow id)$ | $(\Uparrow id_{\Gamma\vdash\Gamma})_{\tau,\Gamma\vdash\tau,\Gamma} \to id_{\tau,\Gamma\vdash\tau,\Gamma}$ |

**Fig. 3.** The typed reduction rules

## 3 Other Preliminaries

**Lemma 1.** *$\sigma$ is a terminating rewrite system.*

**Proof:** We use Zantema's distribution elimination technique [14]. (Refer to this paper for notations and concepts; our $\sigma$ is slightly more complicated than the $\sigma$ that Zantema considers, but the argument is similar.) We actually show that $\sigma$ is totally terminating, i.e. that the reduction relation for $\sigma$ can actually be extended to a *total* well-founded quasi-ordering on terms. Total termination implies *simple* termination, which is the fact that $\sigma$ plus all rules of the form $f(\ldots, u, \ldots) \to u$, for every function symbol $f$, terminates.

Now consider the following rewrite system $\sigma'$, which is obtained from $\sigma$ 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 $(d \Uparrow)$:

| | | | |
|---|---|---|---|
| $(d \Uparrow)$ | $\Uparrow u \circ (v \cdot w) \to (\Uparrow u \circ v) \cdot (\Uparrow u \circ w)$ | $(\eta \Uparrow)$ | $\Uparrow w \to 1 \cdot (w \circ \uparrow)$ |
| $(\circ id)$ | $u \circ (1 \cdot \uparrow) \to u$ | $(\eta \cdot \circ)$ | $(1 \circ u) \cdot (\uparrow \circ u) \to u$ |
| $(id \circ)$ | $(1 \cdot \uparrow) \circ u \to u$ | $(1 \Uparrow)$ | $1 \circ \Uparrow u \to 1$ |
| | | $(1 \Uparrow \circ)$ | $1 \circ (\Uparrow u \circ v) \to 1 \circ v$ |
| $(\circ)$ | $(u \circ v) \circ w \to u \circ (v \circ w)$ | $(\uparrow\Uparrow)$ | $\uparrow \circ \Uparrow u \to u \circ \uparrow$ |
| $(1)$ | $1 \circ (u \cdot v) \to u$ | $(\uparrow\Uparrow \circ)$ | $\uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v)$ |
| $(\uparrow)$ | $\uparrow \circ (u \cdot v) \to v$ | $(\Uparrow\Uparrow)$ | $\Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v)$ |
| $(\cdot)$ | $(u \cdot v) \circ w \to (u \circ w) \cdot (v \circ w)$ | $(\Uparrow\Uparrow \circ)$ | $\Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w$ |
| $(\lambda)$ | $(\lambda u) \circ w \to \lambda(u \circ \Uparrow w)$ | $(\Uparrow \cdot)$ | $\Uparrow u \circ (v \cdot w) \to v \cdot (u \circ w)$ |
| | | $(\Uparrow id)$ | $\Uparrow (1 \cdot \uparrow) \to (1 \cdot \uparrow)$ |

If $\sigma'$ is totally terminating, then it is clear that $\sigma$ is totally terminating as well. Let indeed $>$ be any total well-founded quasi-ordering extending the rewrite relation of $\sigma'$. This induces a total well-founded quasi-ordering on $\lambda\sigma$-terms that extends all the rules in $\sigma$ 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 $\sigma$.

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, $\sigma'$ is totally terminating if and only if $\sigma'$ 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, $\sigma'$ is totally terminating if and only if the following system $\sigma''$ is totally terminating:

| | | | |
|---|---|---|---|
| $(d \Uparrow)$ | $\Uparrow u \circ (v \cdot w) \to (\Uparrow u \circ v) \cdot (\Uparrow u \circ w)$ | $(\eta \Uparrow)$ | $\Uparrow w \to 1 \cdot (w \circ \uparrow)$ |
| $(\circ)$ | $(u \circ v) \circ w \to u \circ (v \circ w)$ | $(\uparrow\Uparrow)$ | $\uparrow \circ \Uparrow u \to u \circ \uparrow$ |
| | | $(\uparrow\Uparrow \circ)$ | $\uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v)$ |
| | | $(\Uparrow\Uparrow)$ | $\Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v)$ |
| $(\cdot)$ | $(u \cdot v) \circ w \to (u \circ w) \cdot (v \circ w)$ | $(\Uparrow\Uparrow \circ)$ | $\Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w$ |
| $(\lambda)$ | $(\lambda u) \circ w \to \lambda(u \circ \Uparrow w)$ | | |

Now, in $\sigma''$ 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 [14] then states that the rewrite system $\sigma'''$ obtained from $\sigma''$ 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 $\sigma''$ is. Here is $\sigma'''$:

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

$\sigma'''$ is then proved totally terminating by using two polynomial interpretations $P_1$ and $P_2$, ordered lexicographically, as in [5]; we let:

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

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 (i.e., $(\lambda)$), $P_1(l) \geq P_1(r)$ and $P_2(l) > P_2(r)$, while in the case of $(\lambda)$ $P_1(l) > P_1(r)$. So $\sigma$ is totally terminating. $\Box$

## 4 Reduction Under Safe Contexts Terminates

We translate each $\lambda\sigma$-term with Church-style typing into a family of $\lambda$-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 constant[1] $\hat{x}_{\tau_1\to\ldots\to\tau_n\to\top\to\tau}$ of the $\lambda$-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 $\lambda$-constants of the indicated types. (The cons operator :: is just a way of forming lists and does not belong to the $\lambda$-calculus; :: is right-associative.) The mapping need not be injective.

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 $n+1$ $\lambda$-terms $t_1 :: \ldots :: t_n :: t_{n+1}$ of respective types $\tau_1, \ldots, \tau_n$ and $\top$, and returns a $\lambda$-term $[\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1})$ of type $\tau$. 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 $n+1$ $\lambda$-terms $t_1 :: \ldots :: t_n :: t_{n+1}$ of respective types $\tau_1, \ldots, \tau_n$ and $\top$, and returns a list of $n'+1$ $\lambda$-terms of respective types $\tau'_1, \ldots, \tau'_{n'}$ and $\top$. The translation is given in Figure 4.

[1] We follow a suggestion of César Muñoz here in using constants instead of variables.

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

**Fig. 4.** Translation to the simply-typed $\lambda$-calculus

The $\beta$ rule $(\lambda x \cdot t)t' \to t[t'/x]$ defines a rewrite relation on $\lambda$-terms that we again write $\longrightarrow$. Recall that the simply-typed $\lambda$-calculus terminates [6]. 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 $\llbracket u \rrbracket(s) \longrightarrow^* \llbracket v \rrbracket(s)$ for every list $s$ of $\lambda$-terms of the right types. (Observe that $\succ$ is an ordering, in particular it is irreflexive precisely because there are $\lambda$-terms of any given type, namely variables.) Similarly, $u \succ v$ if and only if $u$ and $v$ have the same type and $\llbracket u \rrbracket(s) \longrightarrow^+ \llbracket v \rrbracket(s)$ for every list $s$ of $\lambda$-terms of the right types. We also let $u \approx v$ if and only if $\llbracket u \rrbracket(s) = \llbracket v \rrbracket(s)$ for every list $s$ of $\lambda$-terms of the right types.

The interpretation is monotonic in the following sense:

**Lemma 2.** *If $s \longrightarrow^* s'$, then $\llbracket u \rrbracket(s) \longrightarrow^* \llbracket u \rrbracket(s')$.*

**Proof:** Follows from the next Lemma. □

**Lemma 3.** *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:*

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

*where substitution is extended componentwise when $u$ is a stack.*

**Proof:** By structural induction on $u$.

If $u = v[w]$, then $\llbracket u \rrbracket(t_1 :: \ldots :: t_n :: t_{n+1}) = \llbracket v \rrbracket(t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1})$, where $t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1} = \llbracket w \rrbracket(t_1 :: \ldots :: t_n :: t_{n+1})$. 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:

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

If $u = v \circ w$, then the argument is identical; 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 4.** *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 $u[w] \succeq v[w]$: indeed, $[\![u[w]]\!](s) = [\![u]\!]([\![w]\!](s)) \longrightarrow^* [\![v]\!]([\![w]\!](s))$ (by assumption) $= [\![v[w]]\!](s)$. At argument position 2, we must show that $u \succeq v$ entails $w[u] \succeq w[v]$; and indeed $[\![w[u]]\!](s) = [\![w]\!]([\![u]\!](s)) \longrightarrow^* [\![w]\!]([\![v]\!](s))$ (since $u \succeq v$ and by Lemma 2) $= [\![w[v]]\!](s)$. Similarly when $f$ is $\circ$. □

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 e.g. 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 5 that there are enough safe contexts for all typed $\lambda\sigma$-terms to be reduced to normal form by reductions under safe contexts.

**Lemma 5.** *For each rule $l \to r$ except $(\beta)$ in Figure 3, $l \approx r$. For rule $(\beta)$, $l \succ r$.*

**Proof:** We omit all type information for clarity, unless strictly necessary.

Rule $(\beta)$:

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

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 $(\lambda)$: $[\![l]\!](s) = [\![\lambda u]\!]([\![w]\!](s)) = \lambda z_\tau \cdot [\![u]\!](z :: [\![w]\!](s))$, 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 $(\eta \cdot \circ)$: $l = (1[u]) \cdot (\uparrow \circ u) \approx (1 \cdot \uparrow) \circ u$ (by case $([])$) $\approx id \circ u$ (by case $(\eta\cdot)$) $\approx u = r$ (by case $(id\circ)$). Observe that we have used Lemma 4 (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 ($\eta\cdot$)). □

It follows:

**Theorem 1 (Main Theorem)** *Every reduction in the typed $\lambda\sigma$-calculus, where every contracted ($\beta$)-redex occurs under a safe context, is finite.*

**Proof:** Any step in such a reduction is either a ($\beta$)-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 5 and the fact that $\mathcal{C}$ is safe; or a $\sigma$-contraction $u \longrightarrow_\sigma v$, where $u \approx v$ by Lemma 5 and Lemma 4. Therefore each rewrite step is strictly decreasing in the lexicographic product of $\succ$ and $\longrightarrow_\sigma^+$, which is well-founded since the typed $\lambda$-calculus terminates (for $\succ$) and $\sigma$ terminates (by Lemma 1). □

## 5 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:*

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

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 a 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 $\hat{x} t_1 \ldots t_n t_{n+1} \longrightarrow^+ \hat{x} t'_1 \ldots t'_n t'_{n+1} = [\![\mathcal{C}\{v\}]\!](s)$.

If $\mathcal{C} = \Uparrow w_1 \circ (\mathcal{C}_1 \cdot w_2)$ with $\mathcal{C}_1 \in \mathcal{S}_T$ and where $w_1$ and $w_2$ are stacks, then $[\![\mathcal{C}\{u\}]\!](s) = [\![\Uparrow w_1]\!]([\![\mathcal{C}_1\{u\} \cdot w_2]\!](s)) = [\![\Uparrow w_1]\!]([\![\mathcal{C}_1\{u\}]\!](s) :: [\![w_2]\!](s)) = [\![\mathcal{C}_1\{u\}]\!](s) :: [\![w_1]\!]([\![w_2]\!](s)) \longrightarrow^+ [\![\mathcal{C}_1\{v\}]\!](s) :: [\![w_1]\!]([\![w_2]\!](s))$ (by induction hypothesis) $= [\![\mathcal{C}\{v\}]\!](s)$.

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 $\lambda$-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:

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

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:

$$\begin{array}{l}
[\![\mathcal{C}\{u\}]\!](s') \\
= (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{u\}]\!](s'))) \\
\longrightarrow^* (\hat{X}_{m+1} t_1 \ldots t_n t_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{v\}]\!](s'))) \quad \text{(by Lemma 4)} \\
\longrightarrow^+ (\hat{X}_{m+1} t'_1 \ldots t'_n t'_{n+1}) :: [\![w]\!](tl([\![\mathcal{C}'\{v\}]\!](s'))) \quad \text{(by induction hypothesis)} \\
= [\![\mathcal{C}\{v\}]\!](s')
\end{array}$$

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

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

**Proof:** Recall that the $\sigma$-normal forms ([13], 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 $\sigma$ 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. □

Notice however that there are many syntactically safe contexts which cannot occur in $\sigma$-normal terms, for example $\_[w]$ for any stack $w$, or $(\_ \cdot w_1) \circ w_2$ for instance. In particular, the following Corollary is really only a special case of the fact that rewrites where we contract only $(\beta)$-redexes under syntactically safe redexes terminate (itself a special case of Theorem 1).

**Corollary 2** *In the simply-typed $\lambda\sigma$-calculus, every $\sigma$-eager rewrite is finite.*

**Proof:** In every $\sigma$-eager rewrite, every $(\beta)$ is performed under a syntactically safe context by Lemma 8, which is safe by Lemma 7, so Theorem 1 applies. □

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

**Lemma 9.** *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 [13], 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 $\lambda v$, 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 ($\lambda$), ($app$) 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 $C_t$, $C_s$, $C'_s$ are clearly sublanguages of $\mathcal{S}_T$, $\mathcal{S}_S$ and $\mathcal{S}'_S$ respectively. □

**Corollary 3** *In the simply-typed $\lambda\sigma_{\Uparrow}$-calculus, every $\sigma_{\Uparrow}$-eager rewrite is finite.*

**Proof:** In every $\sigma_{\Uparrow}$-eager rewrite, every ($\beta$) is performed under a syntactically safe context by Lemma 9, which is safe by Lemma 7, so Theorem 1 applies. □

## 6 The $\boldsymbol{\lambda\tau}$-Calculus

Ríos has introduced in [13] another $\lambda$-calculus with explicit substitutions, the $\lambda\tau$-*calculus*, which is a left-linear alternative to $\lambda\sigma$. Its language (extended with term and stack variables) is the same as for $\lambda\sigma$, except that we replace the binary operator $\cdot$ by a postfix unary operator $/$:

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

Intuitively, $u/$ denotes $u \cdot id$. The (untyped) reduction rules are given in Figure 5. We call $\tau$ the set of all rules in $\lambda\tau$ except $(\beta_\tau)$.

The typing rules are as for the $\lambda\sigma$-calculus of Section 2, except that we replace rule $(,I)$ by:

$$\frac{u : \Gamma \vdash \tau}{u/ : \Gamma \vdash \tau, \Gamma}\ (,I/)$$

| | | | | | |
|---|---|---|---|---|---|
| $(\beta_\tau)$ | $(\lambda u)v \to u[v/]$ | | | | |
| $([id])$ | $u[id] \to u$ | | | | |
| $(\circ id)$ | $u \circ id \to u$ | | | | |
| $(id\circ)$ | $id \circ u \to u$ | $(1 \Uparrow)$ | $1[\Uparrow u] \to 1$ | | |
| $([])$ | $(u[v])[w] \to u[v \circ w]$ | $(1 \Uparrow \circ)$ | $1[\Uparrow u \circ v] \to 1[v]$ | | |
| $(\circ)$ | $(u \circ v) \circ w \to u \circ (v \circ w)$ | $(\uparrow\Uparrow)$ | $\uparrow \circ \Uparrow u \to u \circ \uparrow$ | | |
| $(1/)$ | $1[u/] \to u$ | $(\uparrow\Uparrow \circ)$ | $\uparrow \circ (\Uparrow u \circ v) \to u \circ (\uparrow \circ v)$ | | |
| $(\uparrow /)$ | $\uparrow \circ u/ \to id$ | $(\Uparrow\Uparrow)$ | $\Uparrow u \circ \Uparrow v \to \Uparrow (u \circ v)$ | | |
| $(/)$ | $u/ \circ w \to \Uparrow w \circ (u[w])/$ | $(\Uparrow\Uparrow \circ)$ | $\Uparrow u \circ (\Uparrow v \circ w) \to \Uparrow (u \circ v) \circ w$ | | |
| $(\lambda)$ | $(\lambda u)[w] \to \lambda(u[\Uparrow w])$ | | | | |
| $(app)$ | $(uv)[w] \to (u[w])(v[w])$ | $(\Uparrow id)$ | $\Uparrow id \to id$ | | |

**Fig. 5.** The reduction rules of the $\lambda\tau$-calculus

Notice that the rules of the $\lambda\tau$-calculus form essentially a subset of those of the $\lambda\sigma$-calculus, except for three minor modifications (rules $(\beta_\tau)$, $(1/)$ and $(\uparrow /)$ instead of $(\beta)$, $(1)$ and $(\uparrow)$) and a major one (rule $(/)$, which cannot be simulated by any rule in $\sigma$). Again, we can get a calculus in Church style by annotating $1$, $\uparrow$, $id$ and $\Uparrow u$ with types agreeing with the typing rules. We then need to adapt the rules of Figure 3 by letting:

| | | | |
|---|---|---|---|
| $(\beta_\tau)$ | $(\lambda u)v \to u[v/]$ | $(1/)$ | $1[u/] \to u$ |
| $(/)$ | $u/ \circ w \to (\Uparrow w)_{\tau, \Gamma \vdash \tau, \Delta} \circ (u[w])/$ | $(\uparrow /)$ | $\uparrow [u/] \to id_{\Gamma \vdash \Gamma}$ |
| | where $u : \Delta \vdash \tau,\ w : \Gamma \vdash \Delta$ | | where $u : \Gamma \vdash \tau$ |

Because of the interaction between $(/)$, $(\circ)$ and $(\uparrow\Uparrow \circ)$, $\tau$ does not terminate, even in the typed case. (This solves a conjecture of [13] in the negative.) Indeed, let $u$ and $v$ be any terms of respective types $\tau', \Gamma \vdash \tau$ and $\Gamma \vdash \tau'$, and let $v_0 = v$, $v_1 = u[v/]$ (of respective types $\Gamma \vdash \tau'$ and $\Gamma \vdash \tau$), and for every $n \geq 0$, $v_{n+2} = v_n[\uparrow \circ v_{n+1}/]$ (of type $\Gamma \vdash \tau'$ if $n$ is even, $\Gamma \vdash \tau$ if $n$ is odd). Then $\uparrow \circ(v_n/ \circ (\uparrow \circ v_{n+1}/))$ (of type $\Gamma \vdash \Gamma$ for every $n$) rewrites in 4 steps to $\uparrow \circ(v_{n+1}/ \circ (\uparrow \circ v_{n+2}/))$, leading to an infinite reduction. Indeed:

$$\begin{array}{ll} & \uparrow \circ(v_n/ \circ (\uparrow \circ v_{n+1}/)) \\ \longrightarrow & \uparrow \circ(\Uparrow (\uparrow \circ v_{n+1}/) \circ v_n[\uparrow \circ v_{n+1}/]/) \text{ by } (/) \\ \longrightarrow & (\uparrow \circ v_{n+1}/) \circ (\uparrow \circ v_n[\uparrow \circ v_{n+1}/]/) \quad \text{by } (\uparrow\Uparrow \circ) \\ \longrightarrow & \uparrow \circ(v_{n+1}/ \circ (\uparrow \circ v_n[\uparrow \circ v_{n+1}/]/)) \quad \text{by } (\circ) \\ = & \uparrow \circ(v_{n+1}/ \circ (\uparrow \circ v_{n+2}/)) \end{array}$$

On the other hand, $\tau$ normalizes weakly (leftmost-innermost reductions terminate [13]). So define weak-$\tau$-eager rewrites as:

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

where $u_1, u_2, \ldots, u_k, \ldots$ are $\tau$-normal. Observe indeed that we cannot guarantee that $\tau$-eager rewrites have this form, unless we restrict ourselves to, say, leftmost-innermost $\tau$-rewrites.

Extend the interpretation of Figure 4 by:

$$[\![u/]\!](s) = [\![u]\!](s) :: s$$

The new interpretation is monotonic just as the old was (see Lemma 2, Lemma 3, Lemma 4), and Lemma 5 still holds:

**Lemma 10.** *For each rule $l \to r$ except $(\beta)$ in the simply-typed $\lambda\tau$-calculus, $l \approx r$. For rule $(\beta)$, $l \succ r$.*

**Proof:** We only examine the new rules, and do as though $\cdot$ was still in the language. Notice that $[\![u/]\!](s) = [\![u \cdot id]\!](s)$, so the case of all old rules, plus $(\beta_\tau)$, $(1/)$, $(\uparrow /)$, is immediate. It remains to examine rule $(/)$, for which $l = u/ \circ w$ and $r = \Uparrow w \circ (u \circ w)/$. But then $[\![l]\!](s) = [\![u/]\!]([\![w]\!](s)) = [\![u]\!]([\![w]\!](s)) :: [\![w]\!](s)$, and $[\![r]\!](s) = [\![\Uparrow w \circ (u \circ w)/]\!](s) = [\![\Uparrow w]\!]([\![(u \circ w)/]\!](s)) = [\![\Uparrow w]\!]([\![u \circ w]\!](s) :: s) = [\![u \circ w]\!](s) :: [\![w]\!](s) = [\![u]\!]([\![w]\!](s)) :: [\![w]\!](s)$, and those two terms are equal. □

Because $u/$ works essentially like an abbreviation of $u \cdot id$, as far as $[\![\_]\!]$ is concerned, it is easy to see that we can redefine the syntactically safe contexts in $\lambda\tau$ as those in $\mathcal{S}_T \cup \mathcal{S}_S$, where:

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

and we immediately get the analogue of Lemma 7: every syntactically safe context is safe. Moreover, the $\tau$-normal forms are described by the grammar:

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

As in Lemma 9, indeed, $s'$ is the class of $\tau$-normal stacks $w$ such that $1[w]$ is $\tau$-normal, or equivalently such that $\uparrow \circ w$ is $\tau$-normal, and $s''$ is that of $\tau$-normal stacks $w$ such that $\Uparrow u \circ w$ is $\tau$-normal for some $u$. Every context around a term of sort $T$ is then of the form:

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

and $C_t$, $C_s$, $C'_s$ are clearly sublanguages of $\mathcal{S}_T$, $\mathcal{S}_S$, $\mathcal{S}_{S'}$ respectively. It follows:

**Corollary 4** *In the simply-typed $\lambda\tau$-calculus, every weak-$\tau$-eager rewrite is finite.*

## 7 Second-Order Types

Our translation technique is not limited to simple type systems. We offer an illustration —but not full proofs, due to space considerations— on a variant of Girard's system $F_2$ (see [4]). System F presents a problem in that the non-left-linear rule $(\eta \cdot \circ)$ breaks subject reduction, and similarly in dependent type systems [11], so we use a variant $\lambda\sigma^\forall$ of $F_2$ simply to illustrate the technique in a setting where no such problem arises. Be aware that we *don't* use the more complicated second-order system of [1].

Formulas of $\lambda\sigma^\forall$ are now either atoms $\alpha$, $\beta$, ..., (and there are infinitely many of them) or implications $\tau_1 \rightarrow \tau_2$, or universal quantifications $\forall\alpha \cdot \tau$. Term and stack types are then augmented accordingly. Correspondingly, we add new terms $\Lambda\alpha \cdot u$ (type abstraction) and $u\tau$ (type application), so that:

$$\begin{array}{l} T^\forall ::= \mathcal{V}_T \mid \lambda T^\forall \mid T^\forall T^\forall \mid T^\forall[S^\forall] \mid 1 \mid \Lambda\alpha \cdot T^\forall \mid T^\forall \tau \\ S^\forall ::= \mathcal{V}_S \mid S^\forall \circ S^\forall \mid id \mid T^\forall \cdot S^\forall \mid\uparrow\mid\Uparrow S^\forall \end{array}$$

We consider two $\alpha$-equivalent terms as equal, where $\alpha$-equivalence is the smallest congruence such that:

$$\Lambda\alpha \cdot u = \Lambda\beta \cdot (u[\beta/\alpha])$$

We then add the following typing rules to those of Figure 2:

$$\frac{u : \Gamma \vdash \tau \qquad \alpha \text{ not free in } \Gamma}{\Lambda\alpha \cdot u : \Gamma \vdash \forall\alpha \cdot \tau}\,(\forall I) \qquad \frac{u : \Gamma \vdash \forall\alpha \cdot \tau}{u\tau' : \Gamma \vdash \tau[\tau'/\alpha]}\,(\forall E)$$

And we add the following reduction rules to those of Figure 3:

| | | |
|---|---|---|
| $(B)$ | $(\Lambda\alpha \cdot u)\tau \rightarrow u[\tau/\alpha]$ | |
| $(\Lambda)$ | $(\Lambda\alpha \cdot u)[v] \rightarrow \Lambda\alpha \cdot u[v]$ provided that $\alpha$ is not free in $v$ | $(App)$ $(u\tau)[v] \rightarrow (u[v])\tau$ |

These extra rules obey subject reduction; the only non-trivial case is for rule $(\Lambda)$, where the derivation for the left-hand side is:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1\\ u : \Gamma \vdash \tau\end{matrix}}{\Lambda\alpha \cdot u : \Gamma \vdash \forall\alpha \cdot \tau}\,(\forall I) \qquad v : \Delta \vdash \Gamma}{(\Lambda\alpha \cdot u)[v] : \Delta \vdash \forall\alpha \cdot \tau}\,(Cut_T)$$

A tedious but easy structural induction on derivations shows that, if $u : \Gamma \vdash \tau$, then $u\sigma : \Gamma\sigma \vdash \tau\sigma$ for any substitution $\sigma$ of formulas for atoms (this uses $\alpha$-conversion in the case of $(\Lambda)$). So there is a derivation $\pi_1'$ of $u[\beta/\alpha] : \Gamma \vdash \tau[\beta/\alpha]$ for any atom $\beta$; choose $\beta$ not free in $\Delta$, and produce:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1'\\ u[\beta/\alpha] : \Gamma \vdash \tau[\beta/\alpha]\end{matrix} \quad v : \Delta \vdash \Gamma}{u[\beta/\alpha][v] : \Delta \vdash \tau[\beta/\alpha]}\,(Cut_T)}{\Lambda\beta \cdot u[\beta/\alpha][v] : \Delta \vdash \forall\alpha \cdot \tau}\,(\forall I)$$

But this is really a derivation of the right-hand side $\Lambda\alpha \cdot u[v]$ since $\Lambda\beta \cdot u[\beta/\alpha][v] = \Lambda\beta \cdot u[v][\beta/\alpha]$ (because $\alpha$ is not free in $v$) $= \Lambda\alpha \cdot u[v]$ (by $\alpha$-renaming).

Let $\sigma^\forall$ denote $\sigma$ plus the rules $(\Lambda)$ and $(App)$, and let $\sigma_\Uparrow^\forall$ be $\sigma_\Uparrow$ plus $(\Lambda)$ and $(App)$. Lemma 1 extends to this case, i.e. $\sigma^\forall$ terminates (even in the untyped case). The proof is the same; we just need to extend the polynomial interpretations $P_1$ and $P_2$ to deal with the added rules: take e.g. $P_1(u\tau)$ and $P_1(\Lambda\alpha \cdot u)$ to be $u$, and $P_2(u\tau)$ and $P_2(\Lambda\alpha \cdot u)$ to be $u + 1$, then for every rule $l \to r$ in $\sigma^\forall$ except $(\lambda)$, $P_1(l) \geq P_1(r)$ and $P_2(l) > P_2(r)$, whereas for $(\lambda)$ $P_1(l) > P_1(r)$.

We extend the translation of Figure 4 so that it now maps second-order $\lambda\sigma$-terms to $\lambda$-terms typed in Girard's $\mathrm{F}_2$ (i.e., to $\lambda^\forall$-terms [4]). Recall that $\lambda^\forall$-terms are variables $x_\tau$, constants $c_\tau$, applications $uv$, abstractions $\lambda x_\tau \cdot u$, type applications $u\tau$ or type abstractions $\Lambda\alpha \cdot u$, where $u$, $v$ are $\lambda^\forall$-terms and $\tau$ is any type. The $\lambda^\forall$-notion of reduction is defined by the two rules:

$$\begin{aligned}(\lambda x \cdot u)v &\to u[v/x]\\ (\Lambda\alpha \cdot u)\tau &\to u[\tau/\alpha]\end{aligned}$$

and its typing rules are:

$$\dfrac{}{\Xi, x_\tau : \tau \vdash x : \tau} \qquad\qquad \dfrac{}{\Xi \vdash c_\tau : \tau}$$

$$\dfrac{\Xi, x_\tau : \tau \vdash u : \tau'}{\Xi \vdash \lambda x_\tau \cdot u : \tau \to \tau'} \qquad\qquad \dfrac{\Xi \vdash u : \tau \to \tau' \qquad \Xi \vdash v : \tau}{\Xi \vdash uv : \tau'}$$

$$\dfrac{\Xi \vdash u : \tau \qquad \alpha \text{ not free in } \Xi}{\Xi \vdash \Lambda\alpha \cdot u : \forall\alpha \cdot \tau} \qquad\qquad \dfrac{\Xi \vdash u : \forall\alpha \cdot \tau}{\Xi \vdash u\tau' : \tau[\tau'/\alpha]}$$

The typed $\lambda^\forall$-calculus then normalizes strongly (by standard reducibility arguments, see e.g. [4, 6]).

Our translation extends trivially to $\lambda\sigma^\forall$ by:

$$\begin{array}{ll} [\![\Lambda\alpha \cdot u]\!](s) & = \Lambda\alpha \cdot ([\![u]\!](s)) \\ [\![u\tau]\!](s) & = ([\![u]\!](s))\tau \end{array}$$

We then need to check:

**Lemma 11.** *If $u : \Gamma \vdash \tau$ is derivable in $\lambda\sigma^\forall$, where $\Gamma = \tau_1, \ldots, \tau_n, \cdot$ and $\Xi \vdash t_1 : \tau_1, \ldots, \Xi \vdash t_n : \tau_n$, $\Xi \vdash t_{n+1} : \top$ are derivable in $\lambda^\forall$, then $\Xi \vdash [\![u]\!](t_1 :: \ldots :: t_n :: t_{n+1}) : \tau$ is derivable in $\lambda^\forall$.*

**Proof:** By structural induction on $u$, we show this and simultaneously the similar result that if $u : \Gamma \vdash \tau'_1, \ldots, \tau'_{n'}, \cdot$, then $[\![u]\!](s)$, where $s = (t_1 :: \ldots :: t_n :: t_{n+1})$, is of the form $t'_1 :: \ldots :: t'_{n'} :: t'_{n'+1}$ with $\Xi \vdash t'_j : \tau'_j$ for every $j$, $1 \leq j \leq n+1$. The only non-trivial case is when $u$ is of the form $\Lambda\alpha \cdot v$, where by $\alpha$-renaming (of $u$) we may assume that $\alpha$ is not free in $\Xi$. Since $u : \Gamma \vdash \tau$ holds, we must have $\tau = \forall\alpha \cdot \tau'$ with $v : \Gamma \vdash \tau'$, so by induction hypothesis $\Xi \vdash [\![v]\!](s) : \tau'$, and beacuse $\alpha$ is not free in $\Xi$, we can derive $\Xi \vdash [\![u]\!](s) : \forall\alpha \cdot \tau'$. This works because the translation maps $\alpha$-equivalent terms in $\lambda\sigma^\forall$ to $\alpha$-equivalent terms in $\lambda^\forall$, which is established by an easy structural induction on $\lambda\sigma^\forall$-terms. $\Box$

It is then easy to check that monotonicity still holds (Lemmas 2 and 4), and that the analogue of Lemma 5 still holds, i.e. for every rule $l \to r$ of $\lambda\sigma^\forall$, $l \approx r$ if this rule is neither $(\beta)$ nor $(B)$, and $l \succ r$ for $(\beta)$ and $(B)$, where $\succ$ and $\approx$ are defined from the notion of reduction of $\lambda^\forall$ as it was defined in terms of the simply-typed notion of reduction in Section 4. It follows:

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

Now extend the syntactically safe contexts by: $\mathcal{S}_T ::= \ldots \mid \Lambda\alpha \cdot \mathcal{S}_T \mid \mathcal{S}_T\tau$. Every syntactically safe context in the extended sense is still safe, and $\sigma^\forall$-normal and $\sigma^\forall_\Uparrow$-normal typed terms of the form $\mathcal{C}\{u\}$ with $u \in T$ are such that $\mathcal{C}$ is syntactically safe. (The normal forms in $T$ are defined exactly as in Lemma 8, resp. Lemma 9, with two new alternatives: $\ldots \mid \Lambda\alpha \cdot t \mid t\tau$.) Then:

**Corollary 6** *In the typed $\lambda\sigma^\forall$, resp. $\lambda\sigma^\forall_\Uparrow$-calculus, every $\sigma^\forall$-eager, resp.$\sigma^\forall_\Uparrow$-eager rewrite is finite.*

Notice finally that the technique applies to all type systems such that (1) the corresponding fragment of $\lambda\sigma$ obeys subject reduction, (2) typed $\lambda\sigma$-terms translate to typed $\lambda$-terms in a corresponding type system, and (3) in the latter, typed $\lambda$-terms terminate. Although types of terms are unique in $\lambda\sigma$ and in $\lambda\sigma^\forall$, this is not in general needed for our method to work: just define an interpretation $[\![u]\!]_t$ indexed by the possible types $t$ of its argument $u$. This is in particular necessary for variants of system $F$; we expect the method to work without effort —but (1) needs to be obeyed, which is not the case for $\lambda\sigma$.

Finally, the technique applies to other $\lambda$-calculi with explicit substitutions, although it seems too elementary to show that the typed $\lambda\zeta$ [10] and $\lambda\upsilon$ [7] calculi are *strongly* normalizing [3]. In particular, our technique, as it stands, always enforces that $u[v][w]$ and $u[v \circ w]$ have the same interpretation, and this prevents us from carrying the technique over to these calculi. We believe that a slight change to the technique, and in particular in the translation of terms of the form $u[v]$, should be enough, and we leave it as future work.

## References


1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *17th PoPL*, pages 31–46, 1990.
2. D. Briaud. Higher-order unification as a typed narrowing. Technical Report 96-R-112, CRIN, 1996.
3. R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *12th LICS*, 1997.
4. J. H. Gallier. On Girard's "candidats de réductibilité". In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Series*, pages 123–203. Academic Press, 1990.
5. T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.
6. J.-L. Krivine. *Lambda-calcul, types et modèles.* Masson, 1992.
7. P. Lescanne and J. Rouyer-Degli. From $\lambda\sigma$ to $\lambda\upsilon$: a journey through calculi of explicit substitutions. In *21st PoPL*, 1994.
8. P.-A. Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *CONFER workshop*, 1994.
9. P.-A. Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *2nd TLCA*, pages 328–334. Springer Verlag LNCS 902, 1995.
10. C. A. Muñoz Hurtado. Confluence and preservation of strong normalization in an explicit substitutions calculus. In *11th LICS*, 1996.
11. C. A. Muñoz Hurtado. Dependent types with explicit substitutions: A meta-theoretical development. Submitted to this workshop, 1997.
12. C. A. Muñoz Hurtado. Meta-theoretical properties of $\lambda_\phi$: A left-linear variant of $\lambda_\sigma$. Technical Report RR-3107, Inria, 1997.
13. A. Ríos. *Contributions à l'étude des lambda-calculs avec substitutions explicites.* PhD thesis, École Normale Supérieure, 1993.
14. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *J. Symbolic Computation*, 17:23–50, 1994.


This article was processed using the LaTeX macro package with LLNCS style