Confluence by Z in Agda for PLFA

Vincent van Oostrom
University of Sussex
vvo@sussex.ac.uk

Pen-and-paper confluence of $\lambda\beta$ (cf. Barendregt 84)

Definition (λ-term; Church 32)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

Pen-and-paper confluence of $\lambda\beta$

Definition (λ-term)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

λ-terms up to α-congruence induced by $\lambda x.M = \lambda y.M[x{:=}y]$, for y not in M

Pen-and-paper confluence of $\lambda\beta$

Definition (λ-term)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

λ-terms up to α-congruence induced by $\lambda x.M = \lambda y.M[x{:=}y]$, for y not in M

Definition (β-reduction)

$\rightarrow_\beta$ on λ-terms is compatible closure of β-scheme $(\lambda x.M)\,N = M[x{:=}N]$

Pen-and-paper confluence of $\lambda\beta$

Definition (λ-term)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

λ-terms up to α-congruence induced by $\lambda x.M = \lambda y.M[x{:=}y]$, for y not in M

Definition (β-reduction)

$\rightarrow_\beta$ on λ-terms is compatible closure of β-scheme $(\lambda x.M)\,N = M[x{:=}N]$

$M[x{:=}N]$ the capture-avoiding substitution of N for x in M

Pen-and-paper confluence of $\lambda\beta$

Definition (λ-term)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

λ-terms up to α-congruence induced by $\lambda x.M = \lambda y.M[x{:=}y]$, for y not in M

Definition (β-reduction)

$\rightarrow_\beta$ on λ-terms is compatible closure of β-scheme $(\lambda x.M)\,N = M[x{:=}N]$

$M[x{:=}N]$ the capture-avoiding substitution of N for x in M

Theorem (Church–Rosser 36)

$\rightarrow_\beta$ *has the Church–Rosser property*

Pen-and-paper confluence of $\lambda\beta$

Definition (λ-term)

A λ-term either is a variable x or an application $M\,N$ or a λ-abstraction $\lambda x.M$

λ-terms up to α-congruence induced by $\lambda x.M = \lambda y.M[x{:=}y]$, for y not in M

Definition (β-reduction)

$\to_\beta$ on λ-terms is compatible closure of β-scheme $(\lambda x.M)\,N = M[x{:=}N]$

$M[x{:=}N]$ the capture-avoiding substitution of N for x in M

Theorem (Church–Rosser)

$\to_\beta$ *has the Church–Rosser property*

$\iff$ $\to_\beta$ is confluent $\iff$ $\twoheadrightarrow_\beta$ has the diamond property

Z

Definition (Z-property of $\to$ for bullet-function $\bullet$ on objects)

$$
\begin{tikzcd}
a \arrow[r] & b \arrow[dl, dashed] \\
a^{\bullet} \arrow[r, dashed, two heads] & b^{\bullet}
\end{tikzcd}
$$

Z

Definition (Z-property of $\to$ for $\bullet$; Loader 98, Dehornoy & ⩔ 08)

for every step $a \to b$

(upperbound) $b \twoheadrightarrow a^{\bullet}$

(monotonic) $a^{\bullet} \twoheadrightarrow b^{\bullet}$

Z

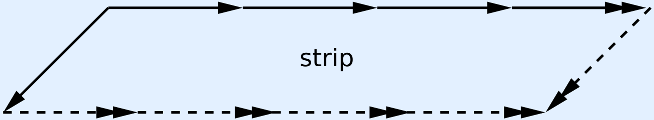
Definition (Z-property of $\rightarrow$ for $\bullet$)

$$\begin{tikzcd} a \arrow[r] & b \arrow[dl, dashed, "\text{upperbound}"] \\ a^{\bullet} \arrow[r, dashed, twoheadrightarrow, "\text{monotonic}"'] & b^{\bullet} \end{tikzcd}$$

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\implies$ strip $\implies$ confluence (Barendregt 84))

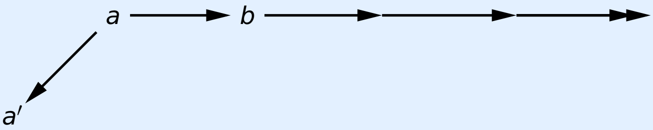


Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)

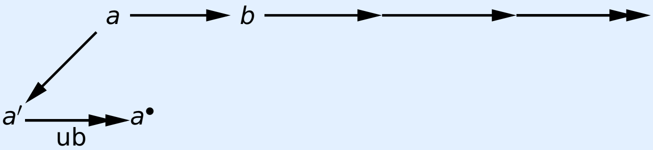


Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\implies$ strip $\implies$ confluence)



Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)

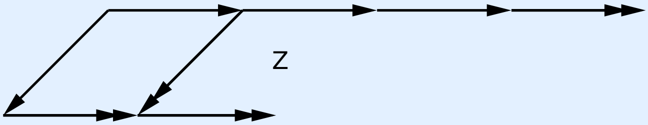
$$\begin{tikzcd}
 & a \arrow[r] \arrow[dl] & b \arrow[r] \arrow[dl, two heads] \arrow[d, phantom, "Z"] & {} \arrow[r] & {} \arrow[r, two heads] & {} \\
a' \arrow[r, two heads] & a^\bullet \arrow[r, two heads] & b^\bullet & & &
\end{tikzcd}$$

Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

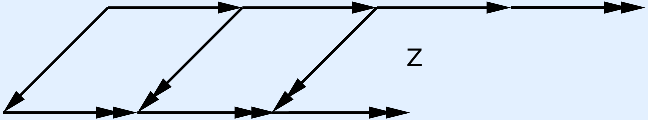
Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)



Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^{\bullet}$ (ub) and $a^{\bullet} \twoheadrightarrow b^{\bullet}$ (mon)

Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)

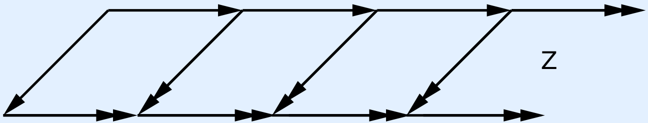


Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)

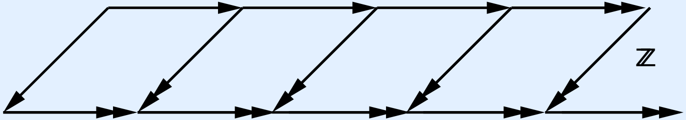


Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\Longrightarrow$ strip $\Longrightarrow$ confluence)

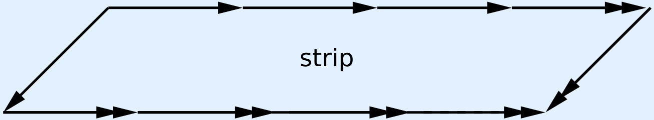


Z

Definition (Z-property of $\to$ for $\bullet$)

for every step $a \to b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z $\implies$ strip $\implies$ confluence)



Z for $\lambda\beta$

Theorem (Loader 98)

for every step $a \to_\beta b$, both $b \twoheadrightarrow_\beta a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow_\beta b^\bullet$ (mon), where

$$\begin{array}{rcll}
x^\bullet & := & x & \\
(\lambda x.M)^\bullet & := & \lambda x.M^\bullet & \\
((\lambda x.M)\,N)^\bullet & := & M^\bullet[x{:=}N^\bullet] & \\
(M\,N)^\bullet & := & M^\bullet\,N^\bullet & \text{otherwise (if } M = x \text{ or } M = P\,Q\text{)}
\end{array}$$

Z for $\lambda\beta$

Theorem (Loader 98)

for every step $a \to_\beta b$, both $b \twoheadrightarrow_\beta a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow_\beta b^\bullet$ (mon), where

$$
\begin{array}{rcll}
x^\bullet & := & x & \\
(\lambda x.M)^\bullet & := & \lambda x.M^\bullet & \\
((\lambda x.M)\, N)^\bullet & := & M^\bullet[x{:=}N^\bullet] & \\
(M\, N)^\bullet & := & M^\bullet\, N^\bullet & \quad\quad \textit{otherwise}
\end{array}
$$

Remark

full development map $\bullet$ contracting all β-redex-patterns in λ-term
(Church–Rosser 30s; Gross–Knuth, preprint 70s; Takahashi, Loader 90s)

Z for $\lambda\beta$ proof

$$\begin{tikzcd}
M \arrow[r, "\beta"'] & N \arrow[dl, dashed, "\beta"] \\
M^{\bullet} \arrow[r, dashed, twoheadrightarrow, "\beta"'] & N^{\bullet}
\end{tikzcd}$$

Z for $\lambda\beta$ proof

$$\begin{tikzcd}
M \arrow[r, "\beta"'] & N \arrow[dl, dashed, "\text{upperbound}"] \\
M^\bullet \arrow[r, dashed, twoheadrightarrow, "\text{monotonic}"'] & N^\bullet
\end{tikzcd}$$

Proof.

(ub) and (mon) by induction on $M \to_\beta N$

Z for $\lambda\beta$ proof

$$\begin{tikzcd}
(\lambda x.M)\,N \arrow[r, "\beta"'] & M[x{:=}N] \arrow[dl, dashed, "\text{upperbound}"] \\
M^\bullet[x{:=}N^\bullet] \arrow[r, dashed, two heads, "\text{monotonic}"'] & M[x{:=}N]^\bullet
\end{tikzcd}$$

Proof.

(ub) and (mon) by induction on $M \to_\beta N$ with base case β

Z for $\lambda\beta$ proof

$$
\begin{tikzcd}
(\lambda x.M)\,N \arrow[r, "\beta"'] & M[x{:=}N] \arrow[dl, dashed, "\text{extensive \& term rewrite system}"] \\
M^\bullet[x{:=}N^\bullet] \arrow[r, dashed, two heads, "\text{right-hand side}"'] & M[x{:=}N]^\bullet
\end{tikzcd}
$$

Proof.

(ub) and (mon) by induction on $M \to_\beta N$ with base case β, using:

(extensive) $M \twoheadrightarrow_\beta M^\bullet$

($\overline{\text{ctx}}$,$\overline{\text{sub}}$) if $M \twoheadrightarrow_\beta N$ and $P \twoheadrightarrow_\beta Q$, then $M[x{:=}P] \twoheadrightarrow_\beta N[y{:=}Q]$

(right-hand side) $M^\bullet[x{:=}N^\bullet] \twoheadrightarrow_\beta M[x{:=}N]^\bullet$

Z for $\lambda\beta$ proof

$$\begin{tikzcd}
(\lambda x.M)\,N \arrow[rr, "\beta"'] & & M[x{:=}N] \arrow[dl, dashed, "\text{extensive \& term rewrite system}"] \\
M^\bullet[x{:=}N^\bullet] \arrow[r, dashed, two heads, "\text{right-hand side}"'] & M[x{:=}N]^\bullet &
\end{tikzcd}$$

Proof.

(ub) and (mon) by induction on $M \to_\beta N$ with base case β, using:

(extensive) $M \twoheadrightarrow_\beta M^\bullet$

($\overline{\text{ctx}}$,$\overline{\text{sub}}$) if $M \twoheadrightarrow_\beta N$ and $P \twoheadrightarrow_\beta Q$, then $M[x{:=}P] \twoheadrightarrow_\beta N[y{:=}Q]$

(right-hand side) $M^\bullet[x{:=}N^\bullet] \twoheadrightarrow_\beta M[x{:=}N]^\bullet$

(ext),(rhs),($\overline{\text{ctx}}$) by induction on M; ($\overline{\text{sub}}$) by induction on $M \to_\beta N$ $\square$

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q] \;{}_{\beta}{\leftarrow}\; (\lambda y.(\lambda x.M)\,N)\,Q \to_{\beta} (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q]\ {}_{\beta}{\leftarrow}\ (\lambda y.(\lambda x.M)\,N)\,Q \to_{\beta} (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Proof.

$((\lambda x.M)\,N)[y{:=}Q] = (\lambda x.M[y{:=}Q])\,N[y{:=}Q] \to_{\beta}$
$M[y{:=}Q][x{:=}N[y{:=}Q]] =_{SL} M[x{:=}N][y{:=}Q]\ {}_{\beta}{\leftarrow}\ (\lambda y.M[x{:=}N])\,Q$ $\square$

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q]\ {}_{\beta}{\leftarrow}\ (\lambda y.(\lambda x.M)\,N)\,Q \to_\beta (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Proof.

$((\lambda x.M)\,N)[y{:=}Q] = (\lambda x.M[y{:=}Q])\,N[y{:=}Q] \to_\beta$
$M[y{:=}Q][x{:=}N[y{:=}Q]] =_{SL} M[x{:=}N][y{:=}Q]\ {}_{\beta}{\leftarrow}\ (\lambda y.M[x{:=}N])\,Q$ $\square$

Remark

closure of $\to_\beta$ under substitution ($\overline{\text{sub}}$) $\iff$ β-critical peak lemma $\iff$ SL

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q]\ {}_\beta\!\leftarrow (\lambda y.(\lambda x.M)\,N)\,Q \to_\beta (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Proof.

$((\lambda x.M)\,N)[y{:=}Q] = (\lambda x.M[y{:=}Q])\,N[y{:=}Q] \to_\beta$
$M[y{:=}Q][x{:=}N[y{:=}Q]] =_{SL} M[x{:=}N][y{:=}Q]\ {}_\beta\!\leftarrow (\lambda y.M[x{:=}N])\,Q$ $\square$

Remark

closure of $\to_\beta$ under substitution ($\overline{\text{sub}}$) $\iff$ β-critical peak lemma $\iff$ SL
proof of (rhs) uses substitution lemma

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q] \;{}_{\beta}{\leftarrow}\; (\lambda y.(\lambda x.M)\,N)\,Q \to_{\beta} (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Proof.

$((\lambda x.M)\,N)[y{:=}Q] = (\lambda x.M[y{:=}Q])\,N[y{:=}Q] \to_{\beta}$
$M[y{:=}Q][x{:=}N[y{:=}Q]] =_{SL} M[x{:=}N][y{:=}Q] \;{}_{\beta}{\leftarrow}\; (\lambda y.M[x{:=}N])\,Q$ $\square$

Remark

closure of $\to_{\beta}$ under substitution ($\overline{\text{sub}}$) $\iff$ β-critical peak lemma $\iff$ SL
proof of (rhs) uses substitution lemma
β-redexes do have overlap (redex-patterns do not)

Substitution lemma

Lemma (β-critical peak)

$((\lambda x.M)\,N)[y{:=}Q] \;{}_{\beta}\!\leftarrow (\lambda y.(\lambda x.M)\,N)\,Q \to_\beta (\lambda y.M[x{:=}N])\,Q$ *is single-step joinable*

Proof.

$((\lambda x.M)\,N)[y{:=}Q] = (\lambda x.M[y{:=}Q])\,N[y{:=}Q] \to_\beta$
$M[y{:=}Q][x{:=}N[y{:=}Q]] =_{SL} M[x{:=}N][y{:=}Q] \;{}_{\beta}\!\leftarrow (\lambda y.M[x{:=}N])\,Q$ $\square$

Remark

closure of $\to_\beta$ under substitution ($\overline{\text{sub}}$) $\iff$ β-critical peak lemma $\iff$ SL
proof of (rhs) uses substitution lemma
β-redexes do have overlap; SL needed to have term rewrite system

Formalisation of confluence by Z for $\lambda\beta$

Motivation

confluence of $\lambda\beta$-calculus PL-litmus test for proof assistants
(inductive λ-terms and β-steps, binding, substitution, modulo α)

Formalisation of confluence by Z for $\lambda\beta$

Motivation

confluence of $\lambda\beta$-calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

Formalisation of confluence by Z for $\lambda\beta$ in Agda

Motivation

confluence of $\lambda\beta$-calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda; had wanted to learn some Agda for some time
(done in 2021; learned at IWC 2023 of Andrea Laretto's MSc thesis)

Formalisation of confluence by Z in Agda based on PLFA

Motivation

confluence of $\lambda\beta$-calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda

design decision: adapt extant PLFA proof (by triangle property; Takahashi 95)
(Programming Language Foundations in Agda by Wadler, Kokke, Siek 20
adaptation allowed reuse of inductive λ-terms, β-steps and SL
reuse good software engineering and useful since absolute Agda beginner)

Formalisation of confluence by Z in Agda based on PLFA

Motivation

confluence of $\lambda\beta$-calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda

design decision: adapt extant PLFA proof (by triangle property

also means have to stick with design decisions of PLFA

λ-terms in PLFA

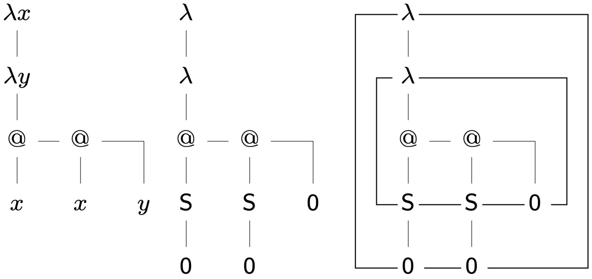
Definition (Nameless λ-term; de Bruijn 72)

PLFA design decision: scoped nameless λ-terms
(avoids α-renaming at the expense of re-indexing)

λ-terms in PLFA

Definition (Nameless λ-term)

PLFA design decision: scoped nameless λ-terms



$\underline{2}$: named $\lambda x.\lambda y.x\,(x\,y)$, nameless $\lambda\lambda \mathsf{S}0\,((\mathsf{S}0)\,0)$, scoped $0 \vdash \lambda\lambda \mathsf{S}0\,((\mathsf{S}0)\,0)$

λ-terms in PLFA

Definition (Nameless λ-term)

PLFA design decision: scoped nameless λ-terms

Definition (Scoped λ-term; & van der Looij & Zwitserlood 04??)

$i \vdash t$ is nameless λ-term t in scope i

(think of i as binding-stack with t closed within it; i is upperbound on indices in t)

λ-terms in PLFA

Definition (Nameless λ-term)

PLFA design decision: scoped nameless λ-terms

Definition (Scoped λ-term)

$i \vdash t$ is nameless λ-term t in scope i (bottom–up) inductively derivable by:

$$\frac{\mathsf{S}i \vdash 0}{\quad}\,0 \qquad \frac{\mathsf{S}i \vdash \mathsf{S}t}{i \vdash t}\,\mathsf{S} \qquad \frac{i \vdash \lambda t}{\mathsf{S}i \vdash t}\,\lambda \qquad \frac{i \vdash t_1 t_2}{i \vdash t_1 \quad i \vdash t_2}\,@$$

λ-terms in PLFA

Definition (Nameless λ-term)

PLFA design decision: scoped nameless λ-terms

Definition (Scoped λ-term)

$i \vdash t$ is nameless λ-term t in scope i

$$\frac{\mathsf{S}i \vdash \mathsf{0}}{}\,\mathsf{0} \qquad \frac{\mathsf{S}i \vdash \mathsf{S}t}{i \vdash t}\,\mathsf{S} \qquad \frac{i \vdash \lambda t}{\mathsf{S}i \vdash t}\,\lambda \qquad \frac{i \vdash t_1 t_2}{i \vdash t_1 \quad i \vdash t_2}\,@$$

Remark

these are generalised nameless λ-terms (Bird & Paterson 99; Hendriks & 03)
PLFA only allows S on 0 and on other Ss; nameless λ-terms; no ho-signature

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, \mathsf{S}i \mapsto i$ (substitute s for the free 0s in t; decrement other indices)

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

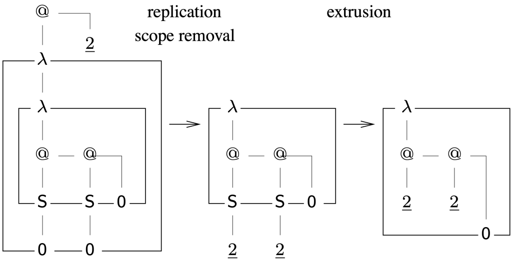
PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
$\to_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

$\longrightarrow_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$

$\longrightarrow_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$



$$0 \vdash \underline{2}\,\underline{2} \longrightarrow_\beta 0 \vdash (\lambda S0\,((S0)\,0))[\underline{2}] = 0 \vdash \lambda S\underline{2}\,((S\underline{2})\,0) = 0 \vdash \lambda \underline{2}\,(\underline{2}\,0)$$

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
$\to_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

($\overline{ctx}$,$\overline{sub}$) *closure of reduction under contexts,substitutions*

(substitution lemma) *for single substitution via parallel substitution*

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, \mathsf{S}i \mapsto i$
$\to_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

($\overline{ctx}$,$\overline{sub}$) closure of reduction under contexts,substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

($\overline{ctx}$) is called congruence in PLFA (wrong; compatible)

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$

$\to_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

($\overline{ctx}$,$\overline{sub}$) closure of reduction under contexts,substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

($\overline{ctx}$) is called congruence in PLFA; ($\overline{sub}$) missing from PLFA (50 loc)

$\rightarrow_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
$\rightarrow_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

($\overline{ctx}$,$\overline{sub}$) closure of reduction under contexts,substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

($\overline{ctx}$) is called congruence in PLFA; ($\overline{sub}$) missing from PLFA; (substitution lemma) is called commutation in PLFA (wrong; self-distributivity / associativity)

$\to_\beta$-steps in PLFA

Definition (Nameless β-reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
$\to_\beta$ on nameless λ-terms is compatible closure of β-scheme $i \vdash (\lambda t)\, s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

$(\overline{ctx},\overline{sub})$ *closure of reduction under contexts,substitutions*

(substitution lemma) for single substitution via parallel substitution

Remark

$(\overline{ctx})$ is called congruence in PLFA; $(\overline{sub})$ missing from PLFA; (substitution lemma) is called commutation in PLFA; terms / steps ad hoc (no signature)

Basic rewriting and full development map • for PLFA

$\text{app-cong} : \forall\{\Gamma\}\ \{K\ L\ M\ N : \Gamma \vdash \star\} \to K \longrightarrow\!\!\!\twoheadrightarrow L \to M \longrightarrow\!\!\!\twoheadrightarrow N \to K \cdot M \longrightarrow\!\!\!\twoheadrightarrow L \cdot N$

$\text{rew-rew} : \forall\{\Gamma\}\ \{M\ N : \Gamma\ ,\star \vdash \star\ \}\ \{K\ L : \Gamma \vdash \star\ \}$

$\quad \to M \longrightarrow\!\!\!\twoheadrightarrow N$

$\quad \to K \longrightarrow\!\!\!\twoheadrightarrow L$

$\quad$ --------------

$\quad \to M\ [\ K\ ] \longrightarrow\!\!\!\twoheadrightarrow N\ [\ L\ ]$

Remark

$\{\ldots\}$ indicates implicit argument; Γ is scope; $\star$ is singleton type of λ-terms
app-cong function taking reductions $K \twoheadrightarrow_\beta L$ and $K \twoheadrightarrow_\beta L$ yielding $K\,M \twoheadrightarrow_\beta L\,N$
rew-rew same but yielding closure under contexts,substitutions

Basic rewriting and full development map • for PLFA

app-cong : $\forall\{\Gamma\}\ \{K\ L\ M\ N : \Gamma \vdash \star\} \to K \longrightarrow\!\!\!\!\rightarrow L \to M \longrightarrow\!\!\!\!\rightarrow N \to K \cdot M \longrightarrow\!\!\!\!\rightarrow L \cdot N$

rew-rew : $\forall\{\Gamma\}\ \{M\ N : \Gamma\ ,\ \star \vdash \star\ \}\ \{K\ L : \Gamma \vdash \star\ \}$

$\to M \longrightarrow\!\!\!\!\rightarrow N$

$\to K \longrightarrow\!\!\!\!\rightarrow L$

$\to M\ [\ K\ ] \longrightarrow\!\!\!\!\rightarrow N\ [\ L\ ]$

$\_\bullet : \forall\ \{\Gamma\ A\} \to \Gamma \vdash A \to \Gamma \vdash A$

$(\text{`}\ x)\bullet\ =\ \text{`}\ x$

$(\lambda\ M)\bullet\ =\ \lambda\ (M\ \bullet)$

$((\lambda\ M) \cdot N)\bullet\ =\ M\ \bullet\ [\ N\ \bullet\ ]$

$(M \cdot N)\bullet\ =\ (M\ \bullet) \cdot (N\ \bullet)$

prime indicates index (as λ-term)

(Extensive) $M \twoheadrightarrow M^{\bullet}$ for PLFA

extensive : ∀ {Γ A} → (M : Γ ⊢ A) → M —↠ M •
extensive (‘ _) = _ ∎
extensive (ƛ M) = abs-cong (extensive M)
extensive ((ƛ M) · N) = _ —→⟨ β ⟩ rew-rew (extensive M) (extensive N)
extensive (‘ _ · N) = appR-cong (extensive N)
extensive (L · M · N) = app-cong (extensive (L · M)) (extensive N)

(Extensive) $M \twoheadrightarrow M^{\bullet}$ for PLFA

$$
\begin{aligned}
&\mathsf{extensive} : \forall \{\Gamma\ A\} \to (M : \Gamma \vdash A) \to M \longrightarrow\!\!\!\!\twoheadrightarrow M\ \bullet \\
&\mathsf{extensive}\ (`\ \_) = \_\ \blacksquare \\
&\mathsf{extensive}\ (\lambda\!\!\lambda\ M) = \mathsf{abs\text{-}cong}\ (\mathsf{extensive}\ M) \\
&\mathsf{extensive}\ ((\lambda\!\!\lambda\ M) \cdot N) = \_ \longrightarrow\langle\ \beta\ \rangle\ \mathsf{rew\text{-}rew}\ (\mathsf{extensive}\ M)\ (\mathsf{extensive}\ N) \\
&\mathsf{extensive}\ (`\ \_ \cdot N) = \mathsf{appR\text{-}cong}\ (\mathsf{extensive}\ N) \\
&\mathsf{extensive}\ (L \cdot M \cdot N) = \mathsf{app\text{-}cong}\ (\mathsf{extensive}\ (L \cdot M))\ (\mathsf{extensive}\ N)
\end{aligned}
$$

Remark

recursion on scoped nameless $M : \Gamma \vdash A$ ($\blacksquare$ is empty reduction)
otherwise only compatibility (wrongly named congruence in PLFA)

(Upperbound) $N \twoheadrightarrow M^{\bullet}$ if $M \rightarrow_{\beta} N$ for PLFA

```
upperbound : ∀ {Γ} → {M N : Γ ⊢ ★}
  → M —→ N
    ------
  → N —↠ M •
upperbound {_} {ƛ _} (ζ ϕ) = abs-cong (upperbound ϕ)
upperbound {_} {(‘ _) · _} {_} (ξ₂ ϕ) = appR-cong (upperbound ϕ)
upperbound {_} {(ƛ _) · M} {((ƛ _) · M)} (ξ₁ (ζ ϕ)) = _ —→⟨ β ⟩ rew-rew (upperbound ϕ) (extensive M)
upperbound {_} {(ƛ L) · _} {.((ƛ L) · _)} (ξ₂ ϕ) = _ —→⟨ β ⟩ rew-rew (extensive L) (upperbound ϕ)
upperbound {_} {(ƛ L) · M} {.(subst (subst-zero M) L)} β = rew-rew (extensive L) (extensive M)
upperbound {_} {_ · _ · M} {.(_ · M)} (ξ₁ ϕ) = app-cong (upperbound ϕ) (extensive M)
upperbound {_} {K · L · _} {.(_ · _ · _)} (ξ₂ ϕ) = app-cong (extensive (K · L)) (upperbound ϕ)
```

(Upperbound) $N \twoheadrightarrow M^\bullet$ if $M \rightarrow_\beta N$ for PLFA

```
upperbound : ∀ {Γ} → {M N : Γ ⊢ ★}
  → M —→ N
    -----
  → N —↠ M •
upperbound {_} {ƛ _} (ζ ϕ) = abs-cong (upperbound ϕ)
upperbound {_} {(‘ _) · _} {_} (ξ₂ ϕ) = appR-cong (upperbound ϕ)
upperbound {_} {(ƛ _) · M} {((ƛ _) · M)} (ξ₁ (ζ ϕ)) = _ —→⟨ β ⟩ rew-rew (upperbound ϕ) (extensive M)
upperbound {_} {(ƛ L) · _} {.((ƛ L) · _)} (ξ₂ ϕ) = _ —→⟨ β ⟩ rew-rew (extensive L) (upperbound ϕ)
upperbound {_} {(ƛ L) · M} {.(subst (subst-zero M) L)} β = rew-rew (extensive L) (extensive M)
upperbound {_} {_ · _ · M} {.(_ · M)} (ξ₁ ϕ) = app-cong (upperbound ϕ) (extensive M)
upperbound {_} {K · L · _} {.(_ · _ · _)} (ξ₂ ϕ) = app-cong (extensive (K · L)) (upperbound ϕ)
```

Remark

recursion on $M \rightarrow_\beta N$ (ζ,ξ_1,ξ_2 traditional names of compatibility clauses)
otherwise only (extensive) and compatibility

(Right-hand side) $(M^\bullet)^{\sigma^\bullet} \twoheadrightarrow (M^\sigma)^\bullet$ for PLFA

```
rhss : ∀{Γ Δ} (M : Γ ⊢ ★) {σ τ : Subst Γ Δ} → ((x : Γ ∋ ★) → τ x ≡ σ x •)
    --------------------
  → subst τ (M •) —↠ (subst σ M)•
rhss (` x) eq rewrite (eq x) = _ ∎
rhss (ƛ M) eq = abs-cong (rhss M (exts-bullet eq))
rhss ((` x) · M) {σ} eq rewrite (eq x) = —↠-trans
  (appR-cong (rhss M eq)) (app-bullet (σ x) (subst σ M)) where
  {- auxiliary rhs/monotonicity lemma for application -}
    app-bullet : ∀{Γ} (L M : Γ ⊢ ★) → L • · M • —↠ (L · M)•
    app-bullet (` _) _ = _ ∎
    app-bullet (ƛ _) _ = (_ —→⟨ β ⟩ _ ∎)
    app-bullet (_ · _) _ = _ ∎
rhss ((ƛ L) · M) {τ = τ} eq rewrite (sym (subst-commute {N = L •} {M •} {τ})) =
  rew-rew (rhss L (exts-bullet eq)) (rhss M eq)
rhss (K · L · M) eq = app-cong (rhss (K · L) eq) (rhss M eq)
```

(Right-hand side) $(M^{\bullet})^{\sigma^{\bullet}} \twoheadrightarrow (M^{\sigma})^{\bullet}$ for PLFA

```
rhss : ∀{Γ Δ} (M : Γ ⊢ ★) {σ τ : Subst Γ Δ} → ((x : Γ ∋ ★) → τ x ≡ σ x •)
    --------------------
  → subst τ (M •) —↠ (subst σ M)•
rhss (‘ x) eq rewrite (eq x) = _ ∎
rhss (ƛ M) eq = abs-cong (rhss M (exts-bullet eq))
rhss ((‘ x) · M) {σ} eq rewrite (eq x) = —↠-trans
  (appR-cong (rhss M eq)) (app-bullet (σ x) (subst σ M)) where
  {- auxiliary rhs/monotonicity lemma for application -}
    app-bullet : ∀{Γ} (L M : Γ ⊢ ★) → L • · M • —↠ (L · M)•
    app-bullet (‘ _) _ = _ ∎
    app-bullet (ƛ _) _ = (_ —→⟨ β ⟩ _ ∎)
    app-bullet (_ · _) _ = _ ∎
rhss ((ƛ L) · M) {τ = τ} eq rewrite (sym (subst-commute {N = L •} {M •} {τ})) =
  rew-rew (rhss L (exts-bullet eq)) (rhss M eq)
rhss (K · L · M) eq = app-cong (rhss (K · L) eq) (rhss M eq)
```

Remark

recursion on scoped nameless $M : \Gamma \vdash \star$ (σ, τ are parallel substitutions)

(Monotonic) $M^\bullet \twoheadrightarrow_\beta N^\bullet$ if $M \rightarrow_\beta N$ for PLFA

```
monotonic : ∀{Γ} → {M N : Γ ⊢ ★}
  → M —→ N
    -------
  → M • —↠ N •
monotonic (ζ ϕ) = abs-cong (monotonic ϕ)
monotonic {_} {(‘ _) · _} {(‘ _) · _} (ξ₂ ϕ) = appR-cong (monotonic ϕ)
monotonic {Γ} {(ƛ M) · N} {.(subst (subst-zero N) M)} β = rhss M bullet-zero where
  {- bullet commutes with lifting terms to substitutions -}
  bullet-zero : (x : Γ , ★ ∋ ★) → subst-zero (N •) x ≡ subst-zero N x •
  bullet-zero Z = refl
  bullet-zero (S x) = refl
monotonic {_} {(ƛ _) · _} {(ƛ _) · _} (ξ₁ (ζ ϕ)) = rew-rew (monotonic ϕ) (_ ∎)
monotonic {_} {(ƛ M) · _} {.((ƛ M) · _)} (ξ₂ ϕ) = rew-rew (M • ∎) (monotonic ϕ)
monotonic {_} {_ · _ · _} {(ƛ _) · _} (ξ₁ ϕ) = —↠-trans (appL-cong (monotonic ϕ)) (_ —→⟨ β ⟩ _ ∎)
monotonic {_} {_ · _ · _} {(‘ _) · _} (ξ₁ ϕ) = appL-cong (monotonic ϕ)
monotonic {_} {_ · _ · _} {_ · _ · _} (ξ₁ ϕ) = appL-cong (monotonic ϕ)
monotonic {_} {_ · _ · _} {_ · _ · _} (ξ₂ ϕ) = appR-cong (monotonic ϕ)
```

Conclusions

- 4 key properties 65 loc

induction on scoped nameless λ-term		induction on derivations
(extensive)	$\Longrightarrow$	(upperbound)
(right-hand side)	$\Longrightarrow$	(monotonic)

- nameless = uninamed; scopes are stacks (Hendriks & 03; not lists)
- basic term rewrite theory of $\lambda\beta$ in PLFA is incomplete
  (no signature; does not show it's a ho-term rewrite system, no $\overline{\text{sub}}$)
- section on confluence of $\lambda\beta$ in PLFA is suboptimal
  (shorter proof via Z; the notes attributing are incorrect / improper)

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?

Remark

full superdevelopment map • contracting β-redex-patterns in inside–out sweep (Aczel 80s; van Raamsdonk 90s, Dehornoy & 00s)

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?

if $a \rightarrow_\beta b$ then $b \twoheadrightarrow_\beta a^\bullet \twoheadrightarrow_\beta b^\bullet$ (Z; Dehornoy & 08), where

$$
\begin{aligned}
x^\bullet &:= x \\
(\lambda x.M)^\bullet &:= \lambda x.M^\bullet \\
(M\,N)^\bullet &:= M'[x{:=}N^\bullet] && \text{if } M^\bullet = \lambda x.M' \\
&:= M^\bullet\,N^\bullet && \text{otherwise}
\end{aligned}
$$

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA? (for reindexing)

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?

by working with generalised scoped λ-terms (instead of separate indices)

Remark

generalised scoped λ-terms due to Bird & Paterson 99, Hendriks & 02 & van der Looij & Zwitserlood 04

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA? (only single substitution)

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?

  by working with single substitution at a given depth

Remark

analogous to Huet's 94 Coq formalisation (based on 6 axioms); cf. proceedings

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?
- avoid maximal scope extrusion? (work with minimal scope extrusion)

Remark

analogous to Hendriks & 03; cf. paper in proceedings

Future work

Questions

- single proof instantiating to full development,full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?
- avoid maximal scope extrusion?

Thanks to

Patrick Dehornoy, Christian Sternagel, Julian Nagele, Bertram Felgenhauer for Z
Andrea Laretto 24 for discussing confluence by Z in Agda (for PLFA; ongoing)