

# The Z Property

Bertram Felgenhauer, Julian Nagele, Vincent van Oostrom, Christian Sternagel\*

February 6, 2026

## Abstract

We formalize the Z property introduced by Dehornoy and van Oostrom [1]. First we show that for any abstract rewrite system, Z implies confluence. Then we give two examples of proofs using Z: confluence of lambda-calculus with respect to beta-reduction and confluence of combinatory logic.

## Contents

<b>1</b>	<b>The Z property</b>	<b>1</b>
<b>2</b>	<b>Lambda Calculus has the Church-Rosser property</b>	<b>2</b>
2.1	Ad-hoc methods for nominal-functions over lambda terms . . .	2
2.2	Substitutions . . . . .	3
<b>3</b>	<b>Combinatory Logic has the Church-Rosser property</b>	<b>7</b>

## 1 The Z property

```
theory Z
imports Abstract-Rewriting.Abstract-Rewriting
begin

locale z-property =
  fixes bullet :: 'a ⇒ 'a (⟦•⟧ [1000] 1000)
  and R :: 'a rel
  assumes Z: (a, b) ∈ R ⇒ (b, a•) ∈ R* ∧ (a•, b•) ∈ R*
begin

lemma monotonicity:
  assumes (a, b) ∈ R*
  shows (a•, b•) ∈ R*
```

---

\*This work was partially supported by FWF (Austrian Science Fund) projects P27502 and P27528.

*<proof>*

**lemma** *semi-confluence*:

**shows**  $(R^{-1} \circ R^*) \subseteq R^\downarrow$

*<proof>*

**lemma** *CR*:  $CR\ R$

*<proof>*

**definition**  $R_d = \{(a, b). (a, b) \in R^* \wedge (b, a^\bullet) \in R^*\}$

**end**

**locale** *angle-property* =

**fixes** *bullet* ::  $'a \Rightarrow 'a \langle \cdot^\bullet \rangle [1000] 1000$

**and**  $R :: 'a\ rel$

**and**  $R_d :: 'a\ rel$

**assumes** *intermediate*:  $R \subseteq R_d\ R_d \subseteq R^*$

**and** *angle*:  $(a, b) \in R_d \implies (b, a^\bullet) \in R_d$

**sublocale** *angle-property*  $\subseteq$  *z-property*

*<proof>*

**sublocale** *z-property*  $\subseteq$  *angle-property bullet R z-property.R\_d bullet R*

*<proof>*

**end**

## 2 Lambda Calculus has the Church-Rosser property

**theory** *Lambda-Z*

**imports**

*Nominal2.Nominal2*

*HOL-Eisbach.Eisbach*

*Z*

**begin**

**atom-decl** *name*

**nominal-datatype** *term* =

*Var name*

| *App term term*

| *Abs x::name t::term binds x in t*

### 2.1 Ad-hoc methods for nominal-functions over lambda terms

*<ML>*

**method** *without-alpha-lst* **methods** *m* =  
 (match *termI* in *H* [simproc del: *alpha-lst*]: -  $\Rightarrow$   $\langle m \rangle$ )

**method** *Abs-lst* =  
 (match **premises** in  
 atom  $?x \# c$  and *P* [thin]: [[atom -]]*lst.* - = [[atom -]]*lst.* - for *c* :: 'a::fs  $\Rightarrow$   
 $\langle$ rule *Abs-lst1-fcb2'* [where *c* = *c*, OF *P*] $\rangle$   
 | *P* [thin]: [[atom -]]*lst.* - = [[atom -]]*lst.* -  $\Rightarrow$   $\langle$ rule *Abs-lst1-fcb2'* [where *c* = (),  
 OF *P*] $\rangle$ )

**method** *pat-comp-aux* =  
 (match **premises** in  
*x* = (- :: *term*)  $\Longrightarrow$  - for *x*  $\Rightarrow$   $\langle$ rule *term.strong-exhaust* [where *y* = *x* and *c* =  
*x*] $\rangle$   
 | *x* = (Var -, -)  $\Longrightarrow$  - for *x* :: - :: *fs*  $\Rightarrow$   
 $\langle$ rule *term.strong-exhaust* [where *y* = *fst x* and *c* = *x*] $\rangle$   
 | *x* = (-, Var -)  $\Longrightarrow$  - for *x* :: - :: *fs*  $\Rightarrow$   
 $\langle$ rule *term.strong-exhaust* [where *y* = *snd x* and *c* = *x*] $\rangle$   
 | *x* = (-, -, Var -)  $\Longrightarrow$  - for *x* :: - :: *fs*  $\Rightarrow$   
 $\langle$ rule *term.strong-exhaust* [where *y* = *snd (snd x)* and *c* = *x*] $\rangle$ )

**method** *pat-comp* = (*pat-comp-aux*; force *simp*: *fresh-star-def fresh-Pair-elim*)

**method** *freshness* **uses** *fresh* =  
 (match **conclusion** in  
 - # -  $\Rightarrow$   $\langle$ *simp add*: *fresh-Unit fresh-Pair fresh* $\rangle$   
 | - #\* -  $\Rightarrow$   $\langle$ *simp add*: *fresh-star-def fresh-Unit fresh-Pair fresh* $\rangle$ )

**method** *solve-eqvt-at* =  
 (*simp add*: *eqvt-at-def*; *simp add*: *perm-supp-eq fresh-star-Pair*)+

**method** *nf* **uses** *fresh* = *without-alpha-lst*  $\langle$   
*eqvt-graph-aux*, rule *TrueI*, *pat-comp*, *auto*, *Abs-lst*,  
*auto simp*: *Abs-fresh-iff pure-fresh perm-supp-eq*,  
*freshness fresh*: *fresh* $\rangle$ +,  
*solve-eqvt-at?*

## 2.2 Substitutions

**nominal-function** *subst*

**where**

*subst x s* (Var *y*) = (if *x* = *y* then *s* else Var *y*)  
 | *subst x s* (App *t u*) = App (*subst x s t*) (*subst x s u*)  
 | atom *y* # (*x*, *s*)  $\Longrightarrow$  *subst x s* (Abs *y t*) = Abs *y* (*subst x s t*)  
 $\langle$ proof $\rangle$

**nominal-termination** (*eqvt*)  $\langle$ proof $\rangle$

**lemma** *fresh-subst*:

$atom\ z \# s \implies z = y \vee atom\ z \# t \implies atom\ z \# subst\ y\ s\ t$   
 ⟨proof⟩

**lemma** *fresh-subst-id* [simp]:  
 $atom\ x \# t \implies subst\ x\ s\ t = t$   
 ⟨proof⟩

The substitution lemma.

**lemma** *subst-subst*:  
**assumes**  $x \neq y$  **and**  $atom\ x \# u$   
**shows**  $subst\ y\ u\ (subst\ x\ s\ t) = subst\ x\ (subst\ y\ u\ s)\ (subst\ y\ u\ t)$   
 ⟨proof⟩

**inductive-set** *Beta* ( $\langle \{-\rightarrow_\beta\} \rangle$ )

**where**

$root: atom\ x \# t \implies (App\ (Abs\ x\ s)\ t, subst\ x\ t\ s) \in \{-\rightarrow_\beta\}$   
 $| Appl: (s, t) \in \{-\rightarrow_\beta\} \implies (App\ s\ u, App\ t\ u) \in \{-\rightarrow_\beta\}$   
 $| Appr: (s, t) \in \{-\rightarrow_\beta\} \implies (App\ u\ s, App\ u\ t) \in \{-\rightarrow_\beta\}$   
 $| Abs: (s, t) \in \{-\rightarrow_\beta\} \implies (Abs\ x\ s, Abs\ x\ t) \in \{-\rightarrow_\beta\}$

**abbreviation** *beta* ( $\langle (-/\rightarrow_\beta -) \rangle$  [56, 56] 55)

**where**

$s \rightarrow_\beta t \equiv (s, t) \in \{-\rightarrow_\beta\}$

**equivariance** *Betap*

**lemmas** *Beta-eqvt* = *Betap.eqvt* [to-set]

**nominal-inductive** *Betap*

**avoids** *Abs: x*

$| root: x$

⟨proof⟩

**lemmas** *Beta-strong-induct* = *Betap.strong-induct* [to-set]

**abbreviation** *betas* (**infix**  $\langle -\rightarrow_\beta^* \rangle$  50)

**where**

$s \rightarrow_\beta^* t \equiv (s, t) \in \{-\rightarrow_\beta\}^*$

**nominal-function** *app-beta* ::  $term \Rightarrow term \Rightarrow term$

**where**

$atom\ x \# u \implies app\ beta\ (Abs\ x\ s')\ u = subst\ x\ u\ s'$   
 $| app\ beta\ (Var\ x)\ u = App\ (Var\ x)\ u$   
 $| app\ beta\ (App\ s\ t)\ u = App\ (App\ s\ t)\ u$   
 ⟨proof⟩

**nominal-termination** (*eqvt*) ⟨proof⟩

**nominal-function** *bullet* ::  $term \Rightarrow term$  ( $\langle -\bullet \rangle$  [1000] 1000)

**where**

$(Var\ x)^\bullet = Var\ x$

$| (Abs\ x\ t)^\bullet = Abs\ x\ t^\bullet$   
 $| (App\ s\ t)^\bullet = app\text{-}beta\ s^\bullet\ t^\bullet$   
 $\langle proof \rangle$   
**nominal-termination** (*eqvt*)  $\langle proof \rangle$

**lemma** *app-beta-exhaust* [*case-names Redex no-Redex*]:  
**fixes**  $c :: 'a :: fs$   
**assumes**  $\bigwedge x\ s'.\ atom\ x \# c \implies s = Abs\ x\ s' \implies thesis$   
**and**  $(\bigwedge t.\ app\text{-}beta\ s\ t = App\ s\ t) \implies thesis$   
**shows** *thesis*  
 $\langle proof \rangle$

**lemma** *App-Betas*:  
**assumes**  $s \rightarrow_{\beta^*} t$  **and**  $u \rightarrow_{\beta^*} v$   
**shows**  $App\ s\ u \rightarrow_{\beta^*} App\ t\ v$   
 $\langle proof \rangle$

**lemma** *Abs-Betas*:  
**assumes**  $s \rightarrow_{\beta^*} t$   
**shows**  $Abs\ x\ s \rightarrow_{\beta^*} Abs\ x\ t$   
 $\langle proof \rangle$

**lemma** *self*:  
 $t \rightarrow_{\beta^*} t^\bullet$   
 $\langle proof \rangle$

**lemma** *fresh-atom-bullet*:  
 $atom\ (x::name) \# t \implies atom\ x \# t^\bullet$   
 $\langle proof \rangle$

**lemma** *subst-Beta*:  
**assumes**  $t \rightarrow_{\beta} t'$   
**shows**  $subst\ x\ s\ t \rightarrow_{\beta} subst\ x\ s\ t'$   
 $\langle proof \rangle$

**lemma** *Beta-in-subst*:  
**assumes**  $s \rightarrow_{\beta} s'$   
**shows**  $subst\ x\ s\ t \rightarrow_{\beta^*} subst\ x\ s'\ t$   
 $\langle proof \rangle$

**lemma** *subst-Betas*:  
**assumes**  $s \rightarrow_{\beta^*} s'$  **and**  $t \rightarrow_{\beta^*} t'$   
**shows**  $subst\ x\ s\ t \rightarrow_{\beta^*} subst\ x\ s'\ t'$   
 $\langle proof \rangle$

**lemma** *Beta-fresh*:  
**fixes**  $x :: name$   
**assumes**  $s \rightarrow_{\beta} t$  **and**  $atom\ x \# s$   
**shows**  $atom\ x \# t$

$\langle proof \rangle$

**lemma** *Abs-BetaD*:

**assumes**  $Abs\ x\ s \rightarrow_{\beta}\ t$

**shows**  $\exists u. t = Abs\ x\ u \wedge s \rightarrow_{\beta}\ u$

$\langle proof \rangle$

**lemma** *Abs-BetaE*:

**assumes**  $Abs\ x\ s \rightarrow_{\beta}\ t$

**obtains**  $u$  **where**  $t = Abs\ x\ u$  **and**  $s \rightarrow_{\beta}\ u$

$\langle proof \rangle$

**lemma** *Abs-BetasE*:

**assumes**  $Abs\ x\ s \rightarrow_{\beta^*}\ t$

**obtains**  $u$  **where**  $t = Abs\ x\ u$  **and**  $s \rightarrow_{\beta^*}\ u$

$\langle proof \rangle$

**lemma** *bullet-App*:

$(App\ s^{\bullet}\ t^{\bullet}, (App\ s\ t)^{\bullet}) \in \{\rightarrow_{\beta}\}^=$

$\langle proof \rangle$

**lemma** *rhs*:

$subst\ x\ s^{\bullet}\ t^{\bullet} \rightarrow_{\beta^*}\ (subst\ x\ s\ t)^{\bullet}$

$\langle proof \rangle$

**lemma** *Betas-fresh*:

**fixes**  $x :: name$

**assumes**  $s \rightarrow_{\beta^*}\ t$  **and**  $atom\ x \# s$

**shows**  $atom\ x \# t$

$\langle proof \rangle$

**lemma** *Var-BetaD*:

**assumes**  $Var\ x \rightarrow_{\beta}\ t$

**shows** *False*

$\langle proof \rangle$

**lemma** *Var-BetasD*:

**assumes**  $Var\ x \rightarrow_{\beta^*}\ t$

**shows**  $t = Var\ x$

$\langle proof \rangle$

**lemma** *app-beta-Betas*:

**assumes**  $s \rightarrow_{\beta^*}\ s'$  **and**  $t \rightarrow_{\beta^*}\ t'$

**shows**  $app\ beta\ s\ t \rightarrow_{\beta^*}\ app\ beta\ s'\ t'$

$\langle proof \rangle$

**lemma** *lambda-Z*:

**assumes**  $s \rightarrow_{\beta}\ t$

**shows**  $t \rightarrow_{\beta^*}\ s^{\bullet} \wedge s^{\bullet} \rightarrow_{\beta^*}\ t^{\bullet}$

*<proof>*

**interpretation** *lambda-z: z-property* *bullet Beta*

*<proof>*

**end**

### 3 Combinatory Logic has the Church-Rosser property

**theory** *CL-Z imports Z*

**begin**

**datatype** *CL = S | K | I | App CL CL* (*' - -> [999, 999] 999*)

**inductive-set** *red :: CL rel* **where**

*L: (t, t') ∈ red ⇒ (' t u, ' t' u) ∈ red*  
*| R: (u, u') ∈ red ⇒ (' t u, ' t u') ∈ red*  
*| S: (' ' S x y z, ' ' x z ' y z) ∈ red*  
*| K: (' ' K x y, x) ∈ red*  
*| I: (' I x, x) ∈ red*

**lemma** *App-mono:*

*(t, t') ∈ red\* ⇒ (u, u') ∈ red\* ⇒ (' t u, ' t' u') ∈ red\**  
*<proof>*

**fun** *bullet-app :: CL ⇒ CL ⇒ CL* **where**

*bullet-app (' ' S x y) z = ' ' x z ' y z*  
*| bullet-app (' K x) y = x*  
*| bullet-app I x = x*  
*| bullet-app t u = ' t u*

**lemma** *bullet-app-red:*

*(' t u, bullet-app t u) ∈ red=*  
*<proof>*

**lemma** *bullet-app-redsI:*

*(s, ' t u) ∈ red\* ⇒ (s, bullet-app t u) ∈ red\**  
*<proof>*

**lemma** *bullet-app-redL:*

*(t, t') ∈ red ⇒ (bullet-app t u, bullet-app t' u) ∈ red\**  
*<proof>*

**lemma** *bullet-app-redR:*

*(u, u') ∈ red ⇒ (bullet-app t u, bullet-app t u') ∈ red\**  
*<proof>*

**lemma** *bullet-app-mono*:  
 **assumes**  $(t, t') \in \text{red}^*$   $(u, u') \in \text{red}^*$  **shows**  $(\text{bullet-app } t \ u, \text{bullet-app } t' \ u') \in \text{red}^*$   
  $\langle \text{proof} \rangle$

**fun** *bullet* ::  $CL \Rightarrow CL$  **where**  
 *bullet* ('  $t \ u$ ) = *bullet-app* (*bullet*  $t$ ) (*bullet*  $u$ )  
 | *bullet*  $t$  =  $t$

**lemma** *bullet-incremental*:  
  $(t, \text{bullet } t) \in \text{red}^*$   
  $\langle \text{proof} \rangle$

**interpretation** *CL:z-property bullet red*  
  $\langle \text{proof} \rangle$

**lemmas** *CR-red = CL.CR*

**end**

## References

- [1] P. Dehornoy and V. v. Oostrom. Z, proving confluence by monotonic single-step upperbound functions. In *Logical Models of Reasoning and Computation (LMRC'2008)*, 2008.