The Z Property

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

March 17, 2025

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

1 The Z property

theory *Z*
imports *Abstract−Rewriting.Abstract-Rewriting*
begin

locale *z-property* =
  fixes *bullet* :: $'a \Rightarrow 'a$ ($\langle$-$^\bullet\rangle$ [*1000*] *1000*)
  and R :: $'a$ *rel*
  assumes *Z*: $(a, b) \in R \Longrightarrow (b, a^\bullet) \in R^* \wedge (a^\bullet, b^\bullet) \in R^*$
begin

lemma *monotonicity*:
  assumes $(a, b) \in R^*$
  shows $(a^\bullet, b^\bullet) \in R^*$

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

⟨*proof*⟩

lemma *semi-confluence*:
  shows $(R^{-1}\ O\ R^*) \subseteq R^{\downarrow}$
⟨*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* :: $'a \Rightarrow 'a$ ($\langle\text{-}^\bullet\rangle$ [*1000*] *1000*)
  and *R* :: *'a rel*
  and R_d :: *'a rel*
  assumes *intermediate*: $R \subseteq R_d\ R_d \subseteq R^*$
  and *angle*: $(a,\ b) \in R_d \Longrightarrow (b,\ a^\bullet) \in R_d$

sublocale *angle-property* ⊆ *z-property*
⟨*proof*⟩

sublocale *z-property* ⊆ *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*]: - ⇒ ‹*m*›)

method *Abs-lst* =
(*match* **premises in**
atom ?x ♯ *c* **and** *P* [*thin*]: [[*atom* -]]*lst*. - = [[*atom* -]]*lst*. - **for** *c* :: *'a*::*fs* ⇒
‹*rule Abs-lst1-fcb2'* [*where c* = *c*, *OF P*]›
| *P* [*thin*]: [[*atom* -]]*lst*. - = [[*atom* -]]*lst*. - ⇒ ‹*rule Abs-lst1-fcb2'* [*where c* = (), *OF P*]›)

method *pat-comp-aux* =
(*match* **premises in**
x = (- :: *term*) ⟹ - **for** *x* ⇒ ‹*rule term.strong-exhaust* [*where y* = *x and c* = *x*]›
| *x* = (*Var* -, -) ⟹ - **for** *x* :: - :: *fs* ⇒
‹*rule term.strong-exhaust* [*where y* = *fst x and c* = *x*]›
| *x* = (-, *Var* -) ⟹ - **for** *x* :: - :: *fs* ⇒
‹*rule term.strong-exhaust* [*where y* = *snd x and c* = *x*]›
| *x* = (-, -, *Var* -) ⟹ - **for** *x* :: - :: *fs* ⇒
‹*rule term.strong-exhaust* [*where y* = *snd* (*snd x*) *and c* = *x*]›)

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

method *freshness* **uses** *fresh* =
(*match* **conclusion in**
- ♯ - ⇒ ‹*simp add*: *fresh-Unit fresh-Pair fresh*›
| - ♯* - ⇒ ‹*simp add*: *fresh-star-def fresh-Unit fresh-Pair fresh*›)

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

method *nf* **uses** *fresh* = *without-alpha-lst* ‹
eqvt-graph-aux, *rule TrueI*, *pat-comp*, *auto*, *Abs-lst*,
auto simp: *Abs-fresh-iff pure-fresh perm-supp-eq*,
(*freshness fresh*: *fresh*)+,
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*) ⟹ *subst x s* (*Abs y t*) = *Abs y* (*subst x s t*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *fresh-subst*:

atom z ♯ *s* ⟹ *z* = *y* ∨ *atom z* ♯ *t* ⟹ *atom z* ♯ *subst y s t*
⟨*proof*⟩

lemma *fresh-subst-id* [*simp*]:
atom x ♯ *t* ⟹ *subst x s t* = *t*
⟨*proof*⟩

The substitution lemma.

lemma *subst-subst*:
assumes *x* ≠ *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* (‹{→$_\beta$}›)
where
root: *atom x* ♯ *t* ⟹ (*App* (*Abs x s*) *t*, *subst x t s*) ∈ {→$_\beta$}
| *Appl*: (*s*, *t*) ∈ {→$_\beta$} ⟹ (*App s u*, *App t u*) ∈ {→$_\beta$}
| *Appr*: (*s*, *t*) ∈ {→$_\beta$} ⟹ (*App u s*, *App u t*) ∈ {→$_\beta$}
| *Abs*: (*s*, *t*) ∈ {→$_\beta$} ⟹ (*Abs x s*, *Abs x t*) ∈ {→$_\beta$}

abbreviation *beta* (‹(-/ →$_\beta$ -)› [*56*, *56*] *55*)
where
s →$_\beta$ *t* ≡ (*s*, *t*) ∈ {→$_\beta$}

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** ‹→$_\beta$*›* *50*)
where
s →$_\beta$* *t* ≡ (*s*, *t*) ∈ {→$_\beta$}*

nominal-function *app-beta* :: *term* ⇒ *term* ⇒ *term*
where
atom x ♯ *u* ⟹ *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* ⇒ *term* (‹-•› [*1000*] *1000*)
where
(*Var x*)• = *Var x*

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

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

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

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

lemma *self*:
  $t \rightarrow_\beta{}^*\ t^\bullet$
⟨*proof*⟩

lemma *fresh-atom-bullet*:
  $atom\ (x::name) \sharp t \Longrightarrow atom\ x \sharp t^\bullet$
⟨*proof*⟩

lemma *subst-Beta*:
  assumes $t \rightarrow_\beta t'$
  shows $subst\ x\ s\ t \rightarrow_\beta subst\ x\ s\ t'$
⟨*proof*⟩

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

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

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

⟨*proof*⟩

lemma *Abs-BetaD*:
assumes $Abs\ x\ s \to_\beta t$
shows $\exists u.\ t = Abs\ x\ u \wedge s \to_\beta u$
⟨*proof*⟩

lemma *Abs-BetaE*:
assumes $Abs\ x\ s \to_\beta t$
obtains u **where** $t = Abs\ x\ u$ **and** $s \to_\beta u$
⟨*proof*⟩

lemma *Abs-BetasE*:
assumes $Abs\ x\ s \to_\beta{}^*\ t$
obtains u **where** $t = Abs\ x\ u$ **and** $s \to_\beta{}^*\ u$
⟨*proof*⟩

lemma *bullet-App*:
$(App\ s^\bullet\ t^\bullet, (App\ s\ t)^\bullet) \in \{\to_\beta\}^=$
⟨*proof*⟩

lemma *rhs*:
$subst\ x\ s^\bullet\ t^\bullet \to_\beta{}^*\ (subst\ x\ s\ t)^\bullet$
⟨*proof*⟩

lemma *Betas-fresh*:
fixes $x :: name$
assumes $s \to_\beta{}^*\ t$ **and** $atom\ x \sharp s$
shows $atom\ x \sharp t$
⟨*proof*⟩

lemma *Var-BetaD*:
assumes $Var\ x \to_\beta t$
shows *False*
⟨*proof*⟩

lemma *Var-BetasD*:
assumes $Var\ x \to_\beta{}^*\ t$
shows $t = Var\ x$
⟨*proof*⟩

lemma *app-beta-Betas*:
assumes $s \to_\beta{}^*\ s'$ **and** $t \to_\beta{}^*\ t'$
shows $app\text{-}beta\ s\ t \to_\beta{}^*\ app\text{-}beta\ s'\ t'$
⟨*proof*⟩

lemma *lambda-Z*:
assumes $s \to_\beta t$
shows $t \to_\beta{}^*\ s^\bullet \wedge s^\bullet \to_\beta{}^*\ t^\bullet$

⟨*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 $CL = S \mid K \mid I \mid App\ CL\ CL$ (‹‘ - -› [*999*, *999*] *999*)

inductive-set *red* :: *CL rel* **where**
  L: $(t, t') \in red \Longrightarrow (‘\ t\ u, ‘\ t'\ u) \in red$
| *R*: $(u, u') \in red \Longrightarrow (‘\ t\ u, ‘\ t\ u') \in red$
| *S*: $(‘\ ‘\ ‘\ S\ x\ y\ z, ‘\ ‘\ x\ z\ ‘\ y\ z) \in red$
| *K*: $(‘\ ‘\ K\ x\ y, x) \in red$
| *I*: $(‘\ I\ x, x) \in red$

lemma *App-mono*:
  $(t, t') \in red^* \Longrightarrow (u, u') \in red^* \Longrightarrow (‘\ t\ u, ‘\ t'\ u') \in red^*$
⟨*proof*⟩

fun *bullet-app* :: $CL \Rightarrow CL \Rightarrow CL$ **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* $t\ u) \in red^=$
⟨*proof*⟩

lemma *bullet-app-redsI*:
  $(s, ‘\ t\ u) \in red^* \Longrightarrow (s,$ *bullet-app* $t\ u) \in red^*$
⟨*proof*⟩

lemma *bullet-app-redL*:
  $(t, t') \in red \Longrightarrow ($*bullet-app* $t\ u,$ *bullet-app* $t'\ u) \in red^*$
⟨*proof*⟩

lemma *bullet-app-redR*:
  $(u, u') \in red \Longrightarrow ($*bullet-app* $t\ u,$ *bullet-app* $t\ u') \in red^*$
⟨*proof*⟩

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

fun *bullet* :: $CL \Rightarrow CL$ **where**
  $bullet\ (`\ t\ u) = bullet\text{-}app\ (bullet\ t)\ (bullet\ u)$
| $bullet\ t = t$

lemma *bullet-incremental*:
  $(t,\ bullet\ t) \in red^*$
⟨*proof*⟩

interpretation *CL*:*z-property bullet red*
⟨*proof*⟩

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.