Confluence

Matthew Doty

May 20, 2020

Contents

theory *DeBruijn*
imports *Main*
~~/src/HOL/Library/Lattice-Syntax
begin

sledgehammer-params [*smt-proofs* = *false*]

declare [[*syntax-ambiguity-warning* = *false*]]

1 Lambda Calculus Using de Bruijn Notation

1.1 Grammar

datatype dB =
$\quad Variable\ nat\ (\langle\text{-}\rangle\ [100]\ 100)$
$\mid Application\ dB\ dB$ (**infixl** $\cdot$ *200*)
$\mid Abstraction\ dB\ (\boldsymbol{\lambda})$

1.2 Substitution

primrec
$lift :: dB \Rightarrow nat \Rightarrow dB$
where
$\quad lift\ (\langle i\rangle)\ k = (if\ i < k\ then\ \langle i\rangle\ else\ \langle(i + 1)\rangle)$
$\mid lift\ (s \cdot t)\ k = lift\ s\ k \cdot lift\ t\ k$
$\mid lift\ (\boldsymbol{\lambda}\ s)\ k = \boldsymbol{\lambda}\ (lift\ s\ (k + 1))$

primrec
$subst :: dB \Rightarrow nat \Rightarrow dB \Rightarrow dB\ \ (\text{-}[\text{-}\ '\!\mapsto \text{-}]\ [300,\ 0,\ 0]\ 300)$
where
$\quad subst\text{-}Var: (\langle i\rangle)[k \mapsto s] =$
$\quad (if\ k < i\ then\ \langle(i - 1)\rangle\ else\ if\ i = k\ then\ s\ else\ \langle i\rangle)$
$\mid subst\text{-}App: (t \cdot u)[k \mapsto s] = t[k \mapsto s] \cdot u[k \mapsto s]$
$\mid subst\text{-}Abs: (\boldsymbol{\lambda}\ t)[k \mapsto s] = \boldsymbol{\lambda}\ (t\ [k + 1 \mapsto lift\ s\ 0])$

declare *subst-Var* [*simp del*]
declare *if-not-P* [*simp*] *not-less-eq* [*simp*]

1.3 Derived Substitution Rules

lemma *subst-eq* [*simp*]: $(\langle k\rangle)[k \mapsto u] = u$
by (*simp add*: *subst-Var*)

lemma *subst-gt* [*simp*]: $i < j \Longrightarrow (\langle j\rangle)[i \mapsto u] = \langle(j - 1)\rangle$
by (*simp add*: *subst-Var*)

lemma *subst-lt* [*simp*]: $j < i \Longrightarrow (\langle j\rangle)[i \mapsto u] = \langle j\rangle$
by (*simp add*: *subst-Var*)

lemma *lift-lift*:
$i < k + 1 \Longrightarrow \mathit{lift}\ (\mathit{lift}\ t\ i)\ (\mathit{Suc}\ k) = \mathit{lift}\ (\mathit{lift}\ t\ k)\ i$
by (*induct t arbitrary*: *i k*) *auto*

lemma *lift-subst* [*simp*]:
$j < i + 1 \Longrightarrow \mathit{lift}\ (t[j \mapsto s])\ i = (\mathit{lift}\ t\ (i + 1))[j \mapsto \mathit{lift}\ s\ i]$
by (*induct t arbitrary*: *i j s*)
(*simp-all add*: *diff-Suc subst-Var lift-lift split*: *nat.split*)

lemma *lift-subst-lt*:
$i < j + 1 \Longrightarrow \mathit{lift}\ (t[j \mapsto s])\ i = (\mathit{lift}\ t\ i)[j+1 \mapsto \mathit{lift}\ s\ i]$
by (*induct t arbitrary*: *i j s*) (*simp-all add*: *subst-Var lift-lift*)

lemma *subst-lift* [*simp*]:
$(\mathit{lift}\ t\ k)[k \mapsto s] = t$
by (*induct t arbitrary*: *k s*) *simp-all*

lemma *subst-subst*:
$i < j + 1 \Longrightarrow t[j+1 \mapsto \mathit{lift}\ v\ i][i \mapsto u[j \mapsto v]] = t[i \mapsto u][j \mapsto v]$
by (*induct t arbitrary*: *i j u v*)
(*simp-all*
add: *diff-Suc subst-Var lift-lift* [*symmetric*] *lift-subst-lt*
split: *nat.split*)

1.4 Small-Step Beta Reduction

inductive *beta* :: $dB \Rightarrow dB \Rightarrow bool$ (**infixl** $\rightarrow_\beta$ *50*)
where
beta [*simp*, *intro*!]: $\boldsymbol{\lambda}\ s \cdot t \rightarrow_\beta s[0 \mapsto t]$
| *appL* [*simp*, *intro*!]: $s \rightarrow_\beta t \Longrightarrow s \cdot u \rightarrow_\beta t \cdot u$
| *appR* [*simp*, *intro*!]: $s \rightarrow_\beta t \Longrightarrow u \cdot s \rightarrow_\beta u \cdot t$
| *abs* [*simp*, *intro*!]: $s \rightarrow_\beta t \Longrightarrow \boldsymbol{\lambda}\ s \rightarrow_\beta \boldsymbol{\lambda}\ t$

inductive-cases *beta-cases* [*elim*!]:
$\mathit{Var}\ i \rightarrow_\beta t$
$\boldsymbol{\lambda}\ r \rightarrow_\beta s$
$s \cdot t \rightarrow_\beta u$

1.5 Derived Small-Step Beta Reduction Rules

theorem *lift-preserves-beta* [*simp*]:
$r \rightarrow_\beta s \Longrightarrow \mathit{lift}\ r\ i \rightarrow_\beta \mathit{lift}\ s\ i$
by (*induct arbitrary*: *i set*: *beta*) *auto*

theorem *subst-preserves-beta* [*simp*, *intro*!]:
$r \rightarrow_\beta s \Longrightarrow r[i \mapsto t] \rightarrow_\beta s[i \mapsto t]$

by (*induct arbitrary*: *t i set*: *beta*)
(*simp-all add*: *subst-subst* [*symmetric*])

1.6 Transitive Beta Reduction

abbreviation
beta-reds :: $dB \Rightarrow dB \Rightarrow bool$ (**infixl** $\rightarrow_\beta{}^*$ *50*) **where**
$s \rightarrow_\beta{}^* t == beta^{**}\ s\ t$

1.7 Transitive Beta Reduction Rules

lemma *rtrancl-beta-Abs* [*intro*!]:
$s \rightarrow_\beta{}^* s' \Longrightarrow \boldsymbol{\lambda}\ s \rightarrow_\beta{}^* \boldsymbol{\lambda}\ s'$
by (*induct set*: *rtranclp*) (*blast intro*: *rtranclp.rtrancl-into-rtrancl*)+

lemma *rtrancl-beta-AppL*:
$s \rightarrow_\beta{}^* s' \Longrightarrow s \cdot t \rightarrow_\beta{}^* s' \cdot t$
by (*induct set*: *rtranclp*) (*blast intro*: *rtranclp.rtrancl-into-rtrancl*)+

lemma *rtrancl-beta-AppR*:
$t \rightarrow_\beta{}^* t' \Longrightarrow s \cdot t \rightarrow_\beta{}^* s \cdot t'$
by (*induct set*: *rtranclp*) (*blast intro*: *rtranclp.rtrancl-into-rtrancl*)+

lemma *rtrancl-beta-App* [*intro*]:
$s \rightarrow_\beta{}^* s' \Longrightarrow t \rightarrow_\beta{}^* t' \Longrightarrow s \cdot t \rightarrow_\beta{}^* s' \cdot t'$
by (*blast intro*!: *rtrancl-beta-AppL rtrancl-beta-AppR intro*: *rtranclp-trans*)

theorem *rtancl-lift-preserves-beta*:
$r \rightarrow_\beta{}^* s \Longrightarrow lift\ r\ i \rightarrow_\beta{}^* lift\ s\ i$
by (*induct rule*: *rtranclp.induct*,
blast,
simp add: *rtranclp.rtrancl-into-rtrancl*)

theorem *rtrancl-subst-preserves-beta*:
$r \rightarrow_\beta{}^* s \Longrightarrow r[i \mapsto t] \rightarrow_\beta{}^* s[i \mapsto t]$
by (*induct rule*: *rtranclp.induct*,
blast,
meson rtranclp.simps subst-preserves-beta)

theorem *rtancl-subst-preserves-beta-inner*:
$r \rightarrow_\beta{}^* s \Longrightarrow t[i \mapsto r] \rightarrow_\beta{}^* t[i \mapsto s]$
proof −
{
fix *r s t*
have $r \rightarrow_\beta s \Longrightarrow t[i \mapsto r] \rightarrow_\beta{}^* t[i \mapsto s]$
by (*induct t arbitrary*: *r s i*,
simp add: *subst-Var r-into-rtranclp*,
simp add: *rtrancl-beta-App*,
simp add: *rtrancl-beta-Abs*)
} **note** † = *this*

show $r \to_\beta{}^* \ s \Longrightarrow t[i \mapsto r] \to_\beta{}^* \ t[i \mapsto s]$
by (*induct rule*: *rtranclp.induct*,
blast,
metis † *rtranclp.rtrancl-into-rtrancl rtranclp-idemp*)
qed

1.8 Confluence Principles

1.8.1 Definitions

definition *square* **where**
$square\ R\ S\ T\ U = (\forall x\ y.\ R\ x\ y \longrightarrow (\forall z.\ S\ x\ z \longrightarrow (\exists u.\ T\ y\ u \wedge U\ z\ u)))$

definition *commute* **where**
$commute\ R\ S = square\ R\ S\ S\ R$

definition *diamond* **where**
$diamond\ R = commute\ R\ R$

abbreviation *confluent* **where**
$confluent\ R \equiv diamond\ (R^{**})$

abbreviation *weakly-confluent* **where**
$weakly\text{-}confluent\ R \equiv square\ R\ R\ (R^{**})\ (R^{**})$

no-notation *converse* ((-$^{-1}$) [*1000*] *999*)
notation *conversep* ((-$^{-1}$) [*1000*] *1000*)

no-notation *rtrancl* ((-*) [*1000*] *1000*)
notation *rtranclp* ((-*) [*1000*] *1000*)

definition
Church-Rosser :: ($'a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$Church\text{-}Rosser\ R =$
$(\forall p\ q.\ (R \sqcup R^{-1})^{**}\ p\ q \longleftrightarrow (\exists x.\ R^{**}\ p\ x \wedge R^{**}\ q\ x))$

1.8.2 Church-Rosser Properties

lemma *common-reduction-to-equiv*:
assumes
$R^*\ x\ z$
$R^*\ y\ z$
shows $(R \sqcup R^{-1})^*\ x\ y$
proof −
have $(R \sqcup R^{-1})^*\ x\ z$
by (*metis* ‹$R^*\ x\ z$› *mono-rtranclp sup2I1*)
moreover have $(R \sqcup R^{-1})^{**}\ z\ y$
by (*metis* (*no-types*, *lifting*)
‹$R^{**}\ y\ z$›
conversep-conversep

mono-rtranclp
rtranclp-converseD
sup2CI)
ultimately show *?thesis*
by *auto*
qed

lemma *Church-Rosser-alt-def*:
Church-Rosser $R =$
$(\forall x\ y.\ (R \sqcup R^{-1})^{**}\ x\ y \longrightarrow (\exists z.\ R^{**}\ x\ z \wedge R^{**}\ y\ z))$
unfolding *Church-Rosser-def*
by (*rule iffI*, *blast*, *meson common-reduction-to-equiv*)

lemma *common-ancestor-to-equiv*:
assumes
$R^{**}\ x\ y$
$R^{**}\ x\ z$
shows $(R \sqcup R^{-1})^{**}\ y\ z$
proof −
have $(R \sqcup R^{-1})^{**}\ y\ x$
by (*meson*
‹$R^{**}\ x\ y$›
common-reduction-to-equiv
rtranclp.rtrancl-refl)
moreover have $(R \sqcup R^{-1})^{**}\ x\ z$
by (*metis* ‹$R^{**}\ x\ z$› *mono-rtranclp sup2I1*)
ultimately show *?thesis* **by** *auto*
qed

lemma *Church-Rosser-confluent*: *confluent* $R \longleftrightarrow$ *Church-Rosser* R
unfolding *square-def commute-def diamond-def Church-Rosser-alt-def*
proof (*rule iffI*; (*rule allI* | *rule impI*)+)
fix $x\ y\ z$
assume
$\forall x\ y.\ (R \sqcup R^{-1})^{**}\ x\ y \longrightarrow (\exists z.\ R^{**}\ x\ z \wedge R^{**}\ y\ z)$
$R^{**}\ x\ y$
$R^{**}\ x\ z$
thus $\exists u.\ R^{**}\ y\ u \wedge R^{**}\ z\ u$
by (*meson common-ancestor-to-equiv*)
next
fix $x\ y$
assume $(R \sqcup R^{-1})^{**}\ x\ y$
$\forall x\ y.\ R^{**}\ x\ y \longrightarrow (\forall z.\ R^{**}\ x\ z \longrightarrow (\exists u.\ R^{**}\ y\ u \wedge R^{**}\ z\ u))$
thus $\exists z.\ R^{**}\ x\ z \wedge R^{**}\ y\ z$
by (*induct rule*: *rtranclp.induct*)
(*auto*, *meson r-into-rtranclp rtranclp-trans*)
qed

1.8.3 Primitive Properties

lemma *square-sym*: *square* $R\ S\ T\ U \Longrightarrow$ *square* $S\ R\ U\ T$
  by (*metis* (*mono-tags*, *hide-lams*) *square-def*)

lemma *square-subset*:
  assumes *square* $R\ S\ T\ U$
  and $T \leq T'$
  shows *square* $R\ S\ T'\ U$
    using *assms*
    unfolding *square-def*
    by (*meson predicate2D-conj*)

lemma *square-reflcl*:
  assumes *square* $R\ S\ T\ (R^{==})$
  and $S \leq T$
  shows *square* $(R^{==})\ S\ T\ (R^{==})$
    using *assms*
    unfolding *square-def*
    by *blast*

lemma *square-rtrancl*:
  assumes *square* $R\ S\ S\ T$
  shows *square* $(R^{**})\ S\ S\ (T^{**})$
  unfolding *square-def*
proof (*intro strip*, *erule rtranclp-induct*)
  fix $x\ y\ z$
  assume $S\ x\ z$
  thus $\exists\, u.\ S\ x\ u \wedge T^{**}\ z\ u$
    by *blast*
next
  fix $x\ y\ z\ y'\ z'$
  assume $S\ x\ z$
    $R^{**}\ x\ y'$
    $R\ y'\ z'$
    $\exists\, u.\ S\ y'\ u \wedge T^{**}\ z\ u$
  thus $\exists\, u'.\ S\ z'\ u' \wedge T^{**}\ z\ u'$
    using ‹*square* $R\ S\ S\ T$›
    unfolding *square-def*
    by (*meson rtranclp.rtrancl-into-rtrancl*)
qed

lemma *square-rtrancl-reflcl-commute*:
  square $R\ S\ (S^{**})\ (R^{==}) \Longrightarrow$ *commute* $(R^{**})\ (S^{**})$
  unfolding *commute-def*
  by (*metis*
    predicate2I
    r-into-rtranclp
    rtranclp-idemp
    rtranclp-reflclp

square-reflcl
square-rtrancl
square-sym)

lemma *commute-sym*: *commute R S* $\Longrightarrow$ *commute S R*
unfolding *commute-def square-def*
by *blast*

lemma *commute-rtrancl*: *commute R S* $\Longrightarrow$ *commute* (R^{**}) (S^{**})
unfolding *commute-def*
by (*blast intro*: *square-rtrancl square-sym*)

lemma *commute-Un*:
commute R T $\Longrightarrow$ *commute S T* $\Longrightarrow$ *commute* $(R \sqcup S)$ *T*
unfolding *commute-def square-def*
by (*metis sup2CI sup2E*)

lemma *diamond-Un*:
assumes *diamond R*
and *diamond S*
and *commute R S*
shows *diamond* (*sup R S*)
using *assms*
unfolding *diamond-def*
by (*blast intro*: *commute-Un commute-sym*)

lemma *diamond-confluent*: *diamond R* $\Longrightarrow$ *confluent R*
unfolding *diamond-def*
by (*simp add*: *commute-rtrancl*)

lemma *square-reflcl-confluent*:
square R R $(R^{==})$ $(R^{==})$ $\Longrightarrow$ *confluent R*
unfolding *diamond-def commute-def*
by (*metis*
inf-sup-ord(*3*)
rtranclp-reflclp
square-reflcl
square-rtrancl
square-sym)

lemma *confluent-Un*:
assumes *confluent R*
and *confluent S*
and *commute* (R^{**}) (S^{**})
shows *confluent* $(R \sqcup S)$
using *assms*
by (*metis diamond-Un diamond-confluent rtranclp-sup-rtranclp*)

lemma *diamond-to-confluence*:

```
  assumes diamond R
  and T ≤ R
  and R ≤ T**
  shows confluent T
    using assms diamond-confluent rtranclp-subset
    by fastforce

lemma basic-diamond-to-confluence:
  assumes diamond R
  and T ≤ R
  and R ≤ T**
  shows confluent R
    using assms diamond-confluent rtranclp-subset
    by fastforce

theorem newman:
  assumes wfP R⁻¹
  and weakly-confluent R
  shows confluent R
proof −
  {
    fix a b c
    have R** a b ⟹ R** a c ⟹ ∃ d. R** b d ∧ R** c d
      using ‹wfP R⁻¹›
    proof (induct arbitrary: b c)
      case (less x b c)
      have R** x c by fact
      have R** x b by fact
      thus ?case
      proof (rule converse-rtranclpE)
        assume x = b
        thus ?thesis
          using ‹R** x c› by blast
      next
        fix y
        assume R x y
        and R** y b
        from ‹R** x c› show ∃ d. R** b d ∧ R** c d
        proof (rule converse-rtranclpE)
          assume x = c
          thus ?thesis
            using ‹R** x b› by blast
        next
          fix z
          assume R** z c and R x z
          from this obtain u where R** y u and R** z u
            using ‹weakly-confluent R›
            unfolding square-def
            using ‹R x y› by blast
```

from *this* **obtain** v **where** $R^{**}\ b\ v$ **and** $R^{**}\ u\ v$
  by (*meson conversep.intros less.hyps* ‹$R\ x\ y$› ‹$R^{**}\ y\ b$›)
from *this* **obtain** w **where** $R^{**}\ v\ w$ **and** $R^{**}\ c\ w$
  by (*meson*
    ‹$R\ x\ z$›
    ‹$R^{**}\ z\ c$›
    ‹$R^{**}\ z\ u$›
    conversep.intros
    less.hyps
    rtranclp-trans)
thus *?thesis*
  by (*meson* ‹$R^{**}\ b\ v$› *rtranclp-trans*)
qed
qed
qed
}
thus *?thesis*
  unfolding *diamond-def commute-def square-def*
  by *auto*
qed

2 Confluence Of Beta Reduction

Here we present a proof of the confluence confluence of $\rightarrow_\beta$. This proof is attributed to William Tait and Per Martin-Löf. The technique has been described in [1]

2.1 Parallel Reduction

inductive *par-beta* :: $dB \Rightarrow dB \Rightarrow bool$ (**infixl** $\Rrightarrow_\beta$ *50*)
where
  var [*simp*, *intro*!]: $(\langle i \rangle) \Rrightarrow_\beta (\langle i \rangle)$
| *abs* [*simp*, *intro*!]: $s \Rrightarrow_\beta t \Longrightarrow \boldsymbol{\lambda}\, s \Rrightarrow_\beta \boldsymbol{\lambda}\, t$
| *app* [*simp*, *intro*!]: $[\![\ s \Rrightarrow_\beta s'; t \Rrightarrow_\beta t'\ ]\!] \Longrightarrow s \cdot t \Rrightarrow_\beta s' \cdot t'$
| *beta* [*simp*, *intro*!]: $[\![\ s \Rrightarrow_\beta s'; t \Rrightarrow_\beta t'\ ]\!] \Longrightarrow (\boldsymbol{\lambda}\, s) \cdot t \Rrightarrow_\beta s'[0 \mapsto t']$

inductive-cases *par-beta-cases* [*elim*!]:
$(\langle i \rangle) \Rrightarrow_\beta t$
$\boldsymbol{\lambda}\, s \Rrightarrow_\beta \boldsymbol{\lambda}\, t$
$(\boldsymbol{\lambda}\, s) \cdot t \Rrightarrow_\beta u$
$s \cdot t \Rrightarrow_\beta u$
$\boldsymbol{\lambda}\ s \Rrightarrow_\beta\ t$

2.2 Properties of Parallel Reduction

lemma *par-beta-varL* [*simp*]:
  $((\langle i \rangle) \Rrightarrow_\beta t) = (t = (\langle i \rangle))$
  by *blast*

lemma *par-beta-refl* [*simp*]: $t \Rrightarrow_\beta t$
by (*induct t*) *simp-all*

lemma *par-beta-lift* [*simp*]:
$t \Rrightarrow_\beta t' \Longrightarrow$ *lift t n* $\Rrightarrow_\beta$ *lift t' n*
by (*induct t arbitrary*: *t' n*) *fastforce*+

lemma *par-beta-subst*:
$s \Rrightarrow_\beta s' \Longrightarrow t \Rrightarrow_\beta t' \Longrightarrow t[n \mapsto s] \Rrightarrow_\beta t'[n \mapsto s']$
apply (*induct t arbitrary*: *s s' t' n*)
apply (*simp add*: *subst-Var*)
apply (*erule par-beta-cases*)
apply (*simp add*: *subst-subst* [*symmetric*]
| *fastforce intro*!: *par-beta-lift*)+
done

2.3 Parallel Reduction is Intermediate To Small-Step and Transitive Beta Reduction

lemma *beta-subset-par-beta*: $(\rightarrow_\beta) \leq (\Rrightarrow_\beta)$
by (*rule predicate2I*, *erule beta.induct*) *simp-all*

lemma *beta-implies-par-beta*: $s \rightarrow_\beta t \Longrightarrow s \Rrightarrow_\beta t$
using *beta-subset-par-beta* **by** *blast*

lemma *par-beta-subset-beta*: $(\Rrightarrow_\beta) \leq (\rightarrow_\beta{}^*)$
by (*rule predicate2I*, *erule par-beta.induct*)
(*blast*, *blast*, *blast*,
blast del: *rtranclp.rtrancl-refl intro*: *rtranclp.rtrancl-into-rtrancl*)

lemma *par-beta-implies-transitive-beta*: $s \Rrightarrow_\beta t \Longrightarrow s \rightarrow_\beta{}^* t$
using *par-beta-subset-beta* **by** *blast*

2.4 Confluence Theorems Parallel Reduction And Beta Reduction

lemma *diamond-par-beta*: *diamond* $(\Rrightarrow_\beta)$
apply (*unfold diamond-def commute-def square-def*)
apply (*rule impI* [*THEN allI* [*THEN allI*]])
apply (*erule par-beta.induct*)
apply (*blast intro*!: *par-beta-subst*)+
done

lemma *par-beta-confluent*: *confluent* $(\Rrightarrow_\beta)$
by (*simp add*: *diamond-confluent diamond-par-beta*)

lemma *beta-confluent*: *confluent* $(\rightarrow_\beta)$
using

```
  beta-subset-par-beta
  diamond-par-beta
  diamond-to-confluence
  par-beta-subset-beta
 by blast
```

3 Equational Theory

```
inductive beta-equiv :: dB ⇒ dB ⇒ bool (infixl ≈β 40) where
   refl [simp]: t ≈β t
 | subst [simp]: (λ s) · t ≈β s [ 0 ↦ t]
 | abs [simp]: s ≈β t ⟹ λ s ≈β λ t
 | symm: s ≈β t ⟹ t ≈β s
 | app [simp]: s ≈β t ⟹ p ≈β q ⟹ s · p ≈β t · q
 | trans [simp]: s ≈β t ⟹ t ≈β u ⟹ s ≈β u

lemma beta-equiv-alt-def:
 (p ≈β q) = ((→β) ⊔ (→β)⁻¹)** p q
proof (rule iffI)
 {
  fix s t
  have ((→β) ⊔ (→β)⁻¹)** s t ⟹ ((→β) ⊔ (→β)⁻¹)** (λ s) (λ t)
   by (induct set: rtranclp,
       blast,
       metis
        beta.abs
        conversep-iff
        rtranclp.rtrancl-into-rtrancl
        sup2CI
        sup2E)
 } note † = this
 {
  fix s t p q
  {
   fix s t p
   have ((→β) ⊔ (→β)⁻¹)** s t ⟹ ((→β) ⊔ (→β)⁻¹)** (s · p) (t · p)
    by (induct set: rtranclp)
      (blast intro: rtranclp.rtrancl-into-rtrancl)+
  }
  moreover
  {
   fix s t p
   have ((→β) ⊔ (→β)⁻¹)** s t ⟹ ((→β) ⊔ (→β)⁻¹)** (p · s) (p · t)
    by (induct set: rtranclp)
      (blast intro: rtranclp.rtrancl-into-rtrancl)+
  }
  ultimately have
   ((→β) ⊔ (→β)⁻¹)** s t ⟹
     ((→β) ⊔ (→β)⁻¹)** p q ⟹
```

$((\to_\beta) \sqcup (\to_\beta)^{-1})^{**}\ (s \cdot p)\ (t \cdot q)$
by (*meson rtranclp-trans*)
} **note** ‡ = *this*
show $p \approx_\beta q \Longrightarrow ((\to_\beta) \sqcup (\to_\beta)^{-1})^{**}\ p\ q$
by (*induct set*: *beta-equiv*,
blast+,
blast intro: †,
metis
converse-join
conversep-conversep
rtranclp-converseD
sup-commute,
blast intro: ‡,
auto)
next
show $((\to_\beta) \sqcup (\to_\beta)^{-1})^{**}\ p\ q \Longrightarrow p \approx_\beta q$
proof (*induct set*: *rtranclp*)
show $p \approx_\beta p$ **by** *simp*
next
fix $x\ y$
assume $p \approx_\beta x$
moreover
assume $((\to_\beta) \sqcup (\to_\beta)^{-1})\ x\ y$
hence $x \to_\beta y \vee y \to_\beta x$ **by** *auto*
moreover
{
fix $p\ q$
have $p \to_\beta q \Longrightarrow p \approx_\beta q$
by (*induct set*: *beta*, *auto*)
}
ultimately show $p \approx_\beta y$
by (*meson beta-equiv.symm beta-equiv.trans*)
qed
qed

end

References

[1] M. Takahashi. Parallel Reductions in λ-Calculus. *Information and Computation*, 118(1):120–127, Apr. 1995.