Stream Examples

Jasmin Christian Blanchette Andrei Popescu
Dmitriy Traytel

January 30, 2015

Contents

1 Sum

definition *pls* :: *stream* ⇒ *stream* ⇒ *stream* **where**
pls xs ys = *dtor-corec-J* (λ(*xs*, *ys*). (*head xs* + *head ys*, *Inr* (*tail xs*, *tail ys*))) (*xs*, *ys*)

lemma *head-pls*[*simp*]: *head* (*pls xs ys*) = *head xs* + *head ys*
unfolding *pls-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

lemma *tail-pls*[*simp*]: *tail* (*pls xs ys*) = *pls* (*tail xs*) (*tail ys*)

unfolding *pls-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

lemma *pls-code*[*code*]: *pls xs ys* = *SCons* (*head xs* + *head ys*) (*pls* (*tail xs*) (*tail ys*))
by (*metis J.ctor-dtor prod.collapse head-pls tail-pls*)

lemma *pls-uniform*: *pls xs ys* = *alg*ϱ*1* (*xs*, *ys*)
unfolding *pls-def*
apply (*rule fun-cong*[*OF sym*[*OF J.dtor-corec-unique*]])
unfolding *alg*ϱ*1*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff convol-def* ϱ*1-def alg*ϱ*1-def*)

2 Onetwo

definition *onetwo* :: *stream* **where**
onetwo = *corecUU0* (λ-. *GUARD0* (*1*, *SCONS0* (*2*, *CONT0* ()))) ()

lemma *onetwo-code*[*code*]: *onetwo* = *SCons 1* (*SCons 2 onetwo*)
apply (*subst onetwo-def*)
unfolding *corecUU0*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval0-leaf0′ o-eq-dest*[*OF eval0-gg0*] *o-eq-dest*[*OF gg0-natural*] *onetwo-def*)

definition *onetwo′* :: *stream* **where**
onetwo′ = *corecUU0* (λ-. *SCONS0* (*1*, *GUARD0* (*2*, *CONT0* ()))) ()

lemma *onetwo′-code*[*code*]: *onetwo′* = *SCons 1* (*SCons 2 onetwo′*)
apply (*subst onetwo′-def*)
unfolding *corecUU0*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval0-leaf0′ o-eq-dest*[*OF eval0-gg0*] *o-eq-dest*[*OF gg0-natural*] *onetwo′-def*)

definition *stutter* :: *stream* **where**
stutter = *corecUU1* (λ-. *SCONS1* (*1*, *GUARD1* (*1*, *PLS1* (*CONT1* (), *CONT1* ()))))) ()

lemma *stutter-code*[*code*]: *stutter* = *SCons 1* (*SCons 1* (*pls stutter stutter*))
apply (*subst stutter-def*)
unfolding *corecUU1 prod.case*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1′ eval1-*$\mathfrak{op}$*1 alg*Λ*1-Inr o-eq-dest*[*OF Abs-*Σ*1-natural*] *o-eq-dest*[*OF eval1-gg1*] *o-eq-dest*[*OF gg1-natural*] *pls-uniform stutter-def*)

3 Shuffle product

definition *prd* :: *stream* $\Rightarrow$ *stream* $\Rightarrow$ *stream* **where**

prd xs ys = corecUU1 (λ(*xs*, *ys*). *GUARD1* (*head xs* $*$ *head ys*,
PLS1 (*CONT1* (*xs*, *tail ys*), *CONT1* (*tail xs*, *ys*)))) (*xs*, *ys*)

lemma *head-prd*[*simp*]: *head* (*prd xs ys*) = *head xs* $*$ *head ys*
unfolding *prd-def corecUU1*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1'*)

lemma *tail-prd*[*simp*]: *tail* (*prd xs ys*) = *pls* (*prd xs* (*tail ys*)) (*prd* (*tail xs*) *ys*)
apply (*subst prd-def*)
unfolding *corecUU1 prod.case*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1'*
eval1-$\mathfrak{op}$*1 alg*Λ*1-Inr o-eq-dest*[*OF Abs-*Σ*1-natural*] *pls-uniform prd-def*)

lemma *prd-code*[*code*]: *prd xs ys* = *SCons* (*head xs* $*$ *head ys*) (*pls* (*prd xs* (*tail ys*)) (*prd* (*tail xs*) *ys*))
by (*metis J.ctor-dtor prod.collapse head-prd tail-prd*)

lemma *prd-uniform*: *prd xs ys* = *alg*ϱ*2* (*xs*, *ys*)
unfolding *prd-def*
apply (*rule fun-cong*[*OF sym*[*OF corecUU1-unique*]])
apply (*rule iffD1*[*OF dtor-J-o-inj*])
unfolding *alg*ϱ*2*
apply (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff J.dtor-ctor*
ϱ*2-def Let-def convol-def eval2-*$\mathfrak{op}$*2 eval1-*$\mathfrak{op}$*1 eval1-leaf1'*
o-eq-dest[*OF Abs-*Σ*1-natural*] *o-eq-dest*[*OF Abs-*Σ*2-natural*] *alg*Λ*2-Inl alg*ϱ*2-def*)
done

abbreviation *scale n s* $\equiv$ *prd* (*sconst n*) *s*

4 Exponentiation

definition *Exp* :: *stream* $\Rightarrow$ *stream* **where**
Exp = *corecUU2* (λ*xs*. *GUARD2* (*exp* (*head xs*), *PRD2* (*END2* (*tail xs*), *CONT2 xs*)))

lemma *head-Exp*[*simp*]: *head* (*Exp xs*) = *exp* (*head xs*)
unfolding *Exp-def corecUU2*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval2-leaf2'*)

lemma *tail-Exp*[*simp*]: *tail* (*Exp xs*) = *prd* (*tail xs*) (*Exp xs*)
apply (*subst Exp-def*)
unfolding *corecUU2*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval2-leaf2'*
eval2-$\mathfrak{op}$*2 alg*Λ*2-Inr o-eq-dest*[*OF Abs-*Σ*2-natural*] *prd-uniform Exp-def*)

lemma *Exp-code*[*code*]: *Exp xs* = *SCons* (*exp* (*head xs*)) (*prd* (*tail xs*) (*Exp xs*))
by (*metis J.ctor-dtor prod.collapse head-Exp tail-Exp*)

lemma *Exp-uniform*: *Exp xs* = *alg*ϱ*3* (*K3.I xs*)

unfolding *Exp-def*
apply (*rule fun-cong*[*OF sym*[*OF corecUU2-unique*]])
apply (*rule iffD1*[*OF dtor-J-o-inj*])
unfolding *algϱ3 o-def*[*symmetric*] *o-assoc*
apply (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff J.dtor-ctor ϱ3-def Let-def convol-def eval3-*$\mathfrak{op}$*3 eval2-*$\mathfrak{op}$*2 eval2-leaf2′ eval3-leaf3′ o-eq-dest*[*OF Abs-Σ2-natural*] *o-eq-dest*[*OF Abs-Σ3-natural*] *algΛ3-Inl algϱ3-def*)
done

5 Supremum

definition *sup* :: *stream fset* ⇒ *stream* **where**
sup = *dtor-corec-J* (*λF*. (*fMax* (*head* |‘| *F*), *Inr* (*tail* |‘| *F*)))

lemma *head-sup*[*simp*]: *head* (*sup F*) = *fMax* (*head* |‘| *F*)
unfolding *sup-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

lemma *tail-sup*[*simp*]: *tail* (*sup F*) = *sup* (*tail* |‘| *F*)
unfolding *sup-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

lemma *sup-code*[*code*]: *sup F* = *SCons* (*fMax* (*head* |‘| *F*)) (*sup* (*tail* |‘| *F*))
by (*metis J.ctor-dtor prod.collapse head-sup tail-sup*)

lemma *sup-uniform*: *sup F* = *algϱ4 F*
unfolding *sup-def*
apply (*rule fun-cong*[*OF sym*[*OF J.dtor-corec-unique*]])
unfolding *algϱ4*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff convol-def ϱ4-def algϱ4-def o-def*)

6 Skewed product

definition *prd′* :: *stream* ⇒ *stream* ⇒ *stream* **where**
prd′ xs ys = *corecUU5* (*λ*(*xs*, *ys*). *GUARD5* (*head xs* ∗ *head ys*,
PRD5 (*CONT5* (*xs*, *tail ys*), *PLS5* (*END5* (*tail xs*), *END5 ys*)))) (*xs*, *ys*)

lemma *prd′-uniform*: *prd′ xs ys* = *algϱ5* (*xs*, *ys*)
unfolding *prd′-def*
apply (*rule fun-cong*[*OF sym*[*OF corecUU5-unique*]])
apply (*rule iffD1*[*OF dtor-J-o-inj*])
unfolding *algϱ5*
apply (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff J.dtor-ctor ϱ5-def Let-def convol-def eval5-*$\mathfrak{op}$*5 eval4-*$\mathfrak{op}$*4 eval3-*$\mathfrak{op}$*3 eval2-*$\mathfrak{op}$*2 eval1-*$\mathfrak{op}$*1 eval5-leaf5′*
o-eq-dest[*OF Abs-Σ1-natural*] *o-eq-dest*[*OF Abs-Σ2-natural*] *o-eq-dest*[*OF Abs-Σ3-natural*]
o-eq-dest[*OF Abs-Σ4-natural*] *o-eq-dest*[*OF Abs-Σ5-natural*] *algΛ5-Inl algϱ5-def*)

done

lemma *head-prd′*[*simp*]: *head* (*prd′ xs ys*) = *head xs* ∗ *head ys*
unfolding *prd′-def corecUU5*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval5-leaf5′*)

lemma *tail-prd′*[*simp*]: *tail* (*prd′ xs ys*) = *prd′* (*prd′ xs* (*tail ys*)) (*pls* (*tail xs*) *ys*)
apply (*subst prd′-def*, *subst* (*2*) *prd′-uniform*)
unfolding *corecUU5 prod.case*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor*
*eval5-*op*5 eval4-*op*4 eval3-*op*3 eval2-*op*2 eval1-*op*1 eval5-leaf5′*
*alg*Λ*5-Inr alg*Λ*5-Inl alg*Λ*4-Inl alg*Λ*3-Inl alg*Λ*2-Inl alg*Λ*1-Inr*
o-eq-dest[*OF Abs-*Σ*5-natural*] *o-eq-dest*[*OF Abs-*Σ*4-natural*]
o-eq-dest[*OF Abs-*Σ*3-natural*] *o-eq-dest*[*OF Abs-*Σ*2-natural*] *o-eq-dest*[*OF Abs-*Σ*1-natural*]
pls-uniform prd′-def)

lemma *prd′-code*[*code*]:
prd′ xs ys = *SCons* (*head xs* ∗ *head ys*) (*prd′* (*prd′ xs* (*tail ys*)) (*pls* (*tail xs*) *ys*))
by (*metis J.ctor-dtor prod.collapse head-prd′ tail-prd′*)

7 Coinduction Up-To Congruence

lemma *SCons-uniform*: *SCons x s* = *eval0* (*gg0* (*x*, *leaf0 s*))
by (*rule iffD1*[*OF J.dtor-inject*])
(*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor o-eq-dest*[*OF eval0-gg0*] *eval0-leaf0′*)

lemma *genCngdd0-SCons*: ⟦*x1* = *x2*; *genCngdd0 R xs1 xs2*⟧ ⟹
genCngdd0 R (*SCons x1 xs1*) (*SCons x2 xs2*)
unfolding *SCons-uniform*
apply (*rule genCngdd0-eval0*)
apply (*rule rel-funD*[*OF gg0-transfer*])
unfolding *rel-pre-J-def BNF-Comp.id-bnf-comp-def vimage2p-def*
apply (*rule rel-funD*[*OF rel-funD*[*OF Pair-transfer*], *rotated*])
apply (*erule rel-funD*[*OF leaf0-transfer*])
apply *assumption*
done

lemma *genCngdd0-genCngdd1*: *genCngdd0 R xs ys* ⟹ *genCngdd1 R xs ys*
unfolding *genCngdd0-def cngdd0-def cptdd0-def genCngdd1-def cngdd1-def cptdd1-def eval1-embL1*[*symmetric*]
apply (*intro allI impI*)
apply (*erule conjE*)+
apply (*drule spec*)
apply (*erule mp conjI*)+
apply (*erule rel-funD*[*OF rel-funD*[*OF comp-transfer*]])
apply (*rule embL1-transfer*)
done

lemma *genCngdd1-SCons*: $[\![x1 = x2;\ genCngdd1\ R\ xs1\ xs2]\!] \Longrightarrow$
genCngdd1 R (*SCons x1 xs1*) (*SCons x2 xs2*)
apply (*subst I1.idem-Cl*[*symmetric*])
apply (*rule genCngdd0-genCngdd1*)
apply (*rule genCngdd0-SCons*)
apply *auto*
done

lemma *genCngdd1-pls*: $[\![genCngdd1\ R\ xs1\ xs2;\ genCngdd1\ R\ ys1\ ys2]\!] \Longrightarrow$
genCngdd1 R (*pls xs1 ys1*) (*pls xs2 ys2*)
unfolding *pls-uniform alg*ϱ*1-def o-apply*
apply (*rule genCngdd1-eval1*)
apply (*rule rel-funD*[*OF K1-as-*$\Sigma\Sigma$*1-transfer*])
apply *simp*
done

lemma *genCngdd1-genCngdd2*: *genCngdd1 R xs ys* $\Longrightarrow$ *genCngdd2 R xs ys*
unfolding *genCngdd1-def cngdd1-def cptdd1-def genCngdd2-def cngdd2-def cptdd2-def eval2-embL2*[*symmetric*]
apply (*intro allI impI*)
apply (*erule conjE*)+
apply (*drule spec*)
apply (*erule mp conjI*)+
apply (*erule rel-funD*[*OF rel-funD*[*OF comp-transfer*]])
apply (*rule embL2-transfer*)
done

lemma *genCngdd2-SCons*: $[\![x1 = x2;\ genCngdd2\ R\ xs1\ xs2]\!] \Longrightarrow$
genCngdd2 R (*SCons x1 xs1*) (*SCons x2 xs2*)
apply (*subst I2.idem-Cl*[*symmetric*])
apply (*rule genCngdd1-genCngdd2*)
apply (*rule genCngdd1-SCons*)
apply *auto*
done

lemma *genCngdd2-pls*: $[\![genCngdd2\ R\ xs1\ xs2;\ genCngdd2\ R\ ys1\ ys2]\!] \Longrightarrow$
genCngdd2 R (*pls xs1 ys1*) (*pls xs2 ys2*)
apply (*subst I2.idem-Cl*[*symmetric*])
apply (*rule genCngdd1-genCngdd2*)
apply (*rule genCngdd1-pls*)
apply *auto*
done

lemma *genCngdd2-prd*: $[\![genCngdd2\ R\ xs1\ xs2;\ genCngdd2\ R\ ys1\ ys2]\!] \Longrightarrow$
genCngdd2 R (*prd xs1 ys1*) (*prd xs2 ys2*)
unfolding *prd-uniform alg*ϱ*2-def o-apply*
apply (*rule genCngdd2-eval2*)
apply (*rule rel-funD*[*OF K2-as-*$\Sigma\Sigma$*2-transfer*])

apply *simp*
done

lemma *genCngdd2-genCngdd3*: *genCngdd2 R xs ys* $\Longrightarrow$ *genCngdd3 R xs ys*
unfolding *genCngdd2-def cngdd2-def cptdd2-def genCngdd3-def cngdd3-def cptdd3-def eval3-embL3*[*symmetric*]
apply (*intro allI impI*)
apply (*erule conjE*)+
apply (*drule spec*)
apply (*erule mp conjI*)+
apply (*erule rel-funD*[*OF rel-funD*[*OF comp-transfer*]])
apply (*rule embL3-transfer*)
done

lemma *genCngdd3-SCons*: $[\![$*x1* = *x2*; *genCngdd3 R xs1 xs2*$]\!]$ $\Longrightarrow$
genCngdd3 R (*SCons x1 xs1*) (*SCons x2 xs2*)
apply (*subst I3.idem-Cl*[*symmetric*])
apply (*rule genCngdd2-genCngdd3*)
apply (*rule genCngdd2-SCons*)
apply *auto*
done

lemma *genCngdd3-pls*: $[\![$*genCngdd3 R xs1 xs2*; *genCngdd3 R ys1 ys2*$]\!]$ $\Longrightarrow$
genCngdd3 R (*pls xs1 ys1*) (*pls xs2 ys2*)
apply (*subst I3.idem-Cl*[*symmetric*])
apply (*rule genCngdd2-genCngdd3*)
apply (*rule genCngdd2-pls*)
apply *auto*
done

lemma *genCngdd3-prd*: $[\![$*genCngdd3 R xs1 xs2*; *genCngdd3 R ys1 ys2*$]\!]$ $\Longrightarrow$
genCngdd3 R (*prd xs1 ys1*) (*prd xs2 ys2*)
apply (*subst I3.idem-Cl*[*symmetric*])
apply (*rule genCngdd2-genCngdd3*)
apply (*rule genCngdd2-prd*)
apply *auto*
done

lemma *genCngdd3-Exp*: *genCngdd3 R xs ys* $\Longrightarrow$
genCngdd3 R (*Exp xs*) (*Exp ys*)
unfolding *Exp-uniform alg*ϱ*3-def o-apply*
apply (*rule genCngdd3-eval3*)
apply (*rule rel-funD*[*OF K3-as-*$\Sigma\Sigma$*3-transfer*])
apply *simp*
done

lemma *genCngdd3-genCngdd4*: *genCngdd3 R xs ys* $\Longrightarrow$ *genCngdd4 R xs ys*
unfolding *genCngdd3-def cngdd3-def cptdd3-def genCngdd4-def cngdd4-def cptdd4-def eval4-embL4*[*symmetric*]

apply (*intro allI impI*)
apply (*erule conjE*)+
apply (*drule spec*)
apply (*erule mp conjI*)+
apply (*erule rel-funD*[*OF rel-funD*[*OF comp-transfer*]])
apply (*rule embL4-transfer*)
done

lemma *genCngdd4-SCons*: $[\![x1 = x2; \textit{genCngdd4 } R \textit{ xs1 xs2}]\!] \Longrightarrow$
genCngdd4 R (*SCons x1 xs1*) (*SCons x2 xs2*)
apply (*subst I4.idem-Cl*[*symmetric*])
apply (*rule genCngdd3-genCngdd4*)
apply (*rule genCngdd3-SCons*)
apply *auto*
done

lemma *genCngdd4-pls*: $[\![\textit{genCngdd4 } R \textit{ xs1 xs2}; \textit{genCngdd4 } R \textit{ ys1 ys2}]\!] \Longrightarrow$
genCngdd4 R (*pls xs1 ys1*) (*pls xs2 ys2*)
apply (*subst I4.idem-Cl*[*symmetric*])
apply (*rule genCngdd3-genCngdd4*)
apply (*rule genCngdd3-pls*)
apply *auto*
done

lemma *genCngdd4-prd*: $[\![\textit{genCngdd4 } R \textit{ xs1 xs2}; \textit{genCngdd4 } R \textit{ ys1 ys2}]\!] \Longrightarrow$
genCngdd4 R (*prd xs1 ys1*) (*prd xs2 ys2*)
apply (*subst I4.idem-Cl*[*symmetric*])
apply (*rule genCngdd3-genCngdd4*)
apply (*rule genCngdd3-prd*)
apply *auto*
done

lemma *genCngdd4-Exp*: *genCngdd4 R xs ys* $\Longrightarrow$
genCngdd4 R (*Exp xs*) (*Exp ys*)
apply (*subst I4.idem-Cl*[*symmetric*])
apply (*rule genCngdd3-genCngdd4*)
apply (*rule genCngdd3-Exp*)
apply *auto*
done

lemma *genCngdd4-sup*: *rel-fset* (*genCngdd4 R*) *xs ys* $\Longrightarrow$
genCngdd4 R (*sup xs*) (*sup ys*)
unfolding *sup-uniform alg*ϱ*4-def o-apply*
apply (*rule genCngdd4-eval4*)
apply (*rule rel-funD*[*OF K4-as-*$\Sigma\Sigma$*4-transfer*])
apply *simp*
done

lemma *genCngdd4-genCngdd5*: *genCngdd4 R xs ys* $\Longrightarrow$ *genCngdd5 R xs ys*

unfolding *genCngdd4-def cngdd4-def cptdd4-def genCngdd5-def cngdd5-def cptdd5-def eval5-embL5*[*symmetric*]
apply (*intro allI impI*)
apply (*erule conjE*)+
apply (*drule spec*)
apply (*erule mp conjI*)+
apply (*erule rel-funD*[*OF rel-funD*[*OF comp-transfer*]])
apply (*rule embL5-transfer*)
done

lemma *genCngdd5-SCons*: $[\![x1 = x2;\ genCngdd5\ R\ xs1\ xs2]\!] \Longrightarrow$
genCngdd5 R (*SCons x1 xs1*) (*SCons x2 xs2*)
apply (*subst I5.idem-Cl*[*symmetric*])
apply (*rule genCngdd4-genCngdd5*)
apply (*rule genCngdd4-SCons*)
apply *auto*
done

lemma *genCngdd5-pls*: $[\![genCngdd5\ R\ xs1\ xs2;\ genCngdd5\ R\ ys1\ ys2]\!] \Longrightarrow$
genCngdd5 R (*pls xs1 ys1*) (*pls xs2 ys2*)
apply (*subst I5.idem-Cl*[*symmetric*])
apply (*rule genCngdd4-genCngdd5*)
apply (*rule genCngdd4-pls*)
apply *auto*
done

lemma *genCngdd5-prd*: $[\![genCngdd5\ R\ xs1\ xs2;\ genCngdd5\ R\ ys1\ ys2]\!] \Longrightarrow$
genCngdd5 R (*prd xs1 ys1*) (*prd xs2 ys2*)
apply (*subst I5.idem-Cl*[*symmetric*])
apply (*rule genCngdd4-genCngdd5*)
apply (*rule genCngdd4-prd*)
apply *auto*
done

lemma *genCngdd5-Exp*: *genCngdd5 R xs ys* $\Longrightarrow$
genCngdd5 R (*Exp xs*) (*Exp ys*)
apply (*subst I5.idem-Cl*[*symmetric*])
apply (*rule genCngdd4-genCngdd5*)
apply (*rule genCngdd4-Exp*)
apply *auto*
done

lemma *genCngdd5-sup*: *rel-fset* (*genCngdd5 R*) *xs ys* $\Longrightarrow$
genCngdd5 R (*sup xs*) (*sup ys*)
apply (*subst I5.idem-Cl*[*symmetric*])
apply (*rule genCngdd4-genCngdd5*)
apply (*rule genCngdd4-sup*)
apply (*auto intro*: *predicate2D*[*OF fset.rel-mono*])
done

lemma *genCngdd5-prd'*: $[\![$*genCngdd5 R xs1 xs2*; *genCngdd5 R ys1 ys2*$]\!] \Longrightarrow$
  genCngdd5 R (*prd' xs1 ys1*) (*prd' xs2 ys2*)
  unfolding *prd'-uniform alg*ϱ*5-def o-apply*
  apply (*rule genCngdd5-eval5*)
  apply (*rule rel-funD*[*OF K5-as-*$\Sigma\Sigma$*5-transfer*])
  apply *simp*
  done

lemma *stream-coinduct*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes *R s s'* $\bigwedge$*s s'*. *R s s'* $\Longrightarrow$ *head s* = *head s'* $\wedge$ *R* (*tail s*) (*tail s'*)
  shows *s* = *s'*
using *assms*(*1*) **proof** (*rule mp*[*OF J.dtor-coinduct*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel R* (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct0*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes *R s s'* $\bigwedge$*s s'*. *R s s'* $\Longrightarrow$ *head s* = *head s'* $\wedge$ *genCngdd0 R* (*tail s*) (*tail s'*)
  shows *s* = *s'*
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd0*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd0 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct1*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes *R s s'* $\bigwedge$*s s'*. *R s s'* $\Longrightarrow$ *head s* = *head s'* $\wedge$ *genCngdd1 R* (*tail s*) (*tail s'*)
  shows *s* = *s'*
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd1*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd1 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct2*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes *R s s'* $\bigwedge$*s s'*. *R s s'* $\Longrightarrow$ *head s* = *head s'* $\wedge$ *genCngdd2 R* (*tail s*) (*tail*

s')
  shows $s = s'$
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd2*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd2 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct3*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes $R\ s\ s'$ $\bigwedge s\ s'.\ R\ s\ s' \Longrightarrow head\ s = head\ s' \wedge genCngdd3\ R\ (tail\ s)\ (tail\ s')$
  shows $s = s'$
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd3*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd3 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct4*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes $R\ s\ s'$ $\bigwedge s\ s'.\ R\ s\ s' \Longrightarrow head\ s = head\ s' \wedge genCngdd4\ R\ (tail\ s)\ (tail\ s')$
  shows $s = s'$
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd4*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd4 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

lemma *stream-coinduct5*[*case-names Eq-stream*, *case-conclusion Eq-stream head tail*]:
  assumes $R\ s\ s'$ $\bigwedge s\ s'.\ R\ s\ s' \Longrightarrow head\ s = head\ s' \wedge genCngdd5\ R\ (tail\ s)\ (tail\ s')$
  shows $s = s'$
using *assms*(*1*) **proof** (*rule mp*[*OF coinductionU-genCngdd5*, *rotated*], *safe*)
  fix *a b*
  assume *R a b*
  from *assms*(*2*)[*OF this*] **show** *F-rel* (*genCngdd5 R*) (*dtor-J a*) (*dtor-J b*)
    by (*cases dtor-J a dtor-J b rule*: *prod.exhaust*[*case-product prod.exhaust*])
      (*auto simp*: *rel-pre-J-def vimage2p-def BNF-Comp.id-bnf-comp-def*)
qed

8 Proofs by Coinduction Up-To Congruence

lemma *pls-commute*: *pls xs ys* = *pls ys xs*
by (*coinduction arbitrary*: *xs ys rule*: *stream-coinduct*) *auto*

lemma *prd-commute*: *prd xs ys* = *prd ys xs*
proof (*coinduction arbitrary*: *xs ys rule*: *stream-coinduct1*)
case *Eq-stream*
then show *?case* **unfolding** *tail-prd*
by (*subst pls-commute*) (*auto intro*: *genCngdd1-pls*)
qed

lemma *pls-assoc*: *pls* (*pls xs ys*) *zs* = *pls xs* (*pls ys zs*)
by (*coinduction arbitrary*: *xs ys zs rule*: *stream-coinduct*) *auto*

lemma *pls-commute-assoc*: *pls xs* (*pls ys zs*) = *pls ys* (*pls xs zs*)
by (*metis pls-assoc pls-commute*)

lemmas *pls-ac-simps* = *pls-assoc pls-commute pls-commute-assoc*

lemma *onetwo* = *onetwo′*
by (*coinduction rule*: *stream-coinduct0*)
(*auto simp*: *arg-cong*[*OF onetwo-code*, *of head*] *arg-cong*[*OF onetwo′-code*, *of head*] *J.dtor-ctor*
arg-cong[*OF onetwo-code*, *of tail*] *arg-cong*[*OF onetwo′-code*, *of tail*] *intro*: *genCngdd0-SCons*)

lemma *prd-distribL*: *prd xs* (*pls ys zs*) = *pls* (*prd xs ys*) (*prd xs zs*)
proof (*coinduction arbitrary*: *xs ys zs rule*: *stream-coinduct1*)
case *Eq-stream*
have $\bigwedge$*a b c d*. *pls* (*pls a b*) (*pls c d*) = *pls* (*pls a c*) (*pls b d*) **by** (*metis pls-assoc pls-commute*)
then have *?tail* **by** (*auto intro*!: *genCngdd1-pls*)
then show *?case* **by** (*simp add*: *algebra-simps*)
qed

lemma *prd-distribR*: *prd* (*pls xs ys*) *zs* = *pls* (*prd xs zs*) (*prd ys zs*)
proof (*coinduction arbitrary*: *xs ys zs rule*: *stream-coinduct1*)
case *Eq-stream*
have $\bigwedge$*a b c d*. *pls* (*pls a b*) (*pls c d*) = *pls* (*pls a c*) (*pls b d*) **by** (*metis pls-assoc pls-commute*)
then have *?tail* **by** (*auto intro*!: *genCngdd1-pls*)
then show *?case* **by** (*simp add*: *algebra-simps*)
qed

lemma *prd-assoc*: *prd* (*prd xs ys*) *zs* = *prd xs* (*prd ys zs*)
proof (*coinduction arbitrary*: *xs ys zs rule*: *stream-coinduct1*)
case *Eq-stream*
have *?tail* **unfolding** *tail-prd pls-ac-simps prd-distribL prd-distribR* **by** (*auto*

intro!: *genCngdd1-pls*)
  then show *?case* **by** *simp*
qed

lemma *prd-commute-assoc*: *prd xs* (*prd ys zs*) = *prd ys* (*prd xs zs*)
  by (*metis prd-assoc prd-commute*)

lemmas *prd-ac-simps* = *prd-assoc prd-commute prd-commute-assoc*

lemma *sconst-0*[*simp*]: *same 0* = *sconst 0*
  by (*coinduction rule*: *stream-coinduct0*) *auto*

lemma *pls-sconst-0L*[*simp*]: *pls* (*sconst 0*) *s* = *s*
  by (*coinduction arbitrary*: *s rule*: *stream-coinduct*) *auto*

lemma *pls-sconst-0R*[*simp*]: *pls s* (*sconst 0*) = *s*
  by (*coinduction arbitrary*: *s rule*: *stream-coinduct*) *auto*

lemma *scale-0*[*simp*]: *scale 0 s* = *sconst 0*
  apply (*coinduction arbitrary*: *s rule*: *stream-coinduct1*)
  apply *simp*
  apply (*subst* (*5*) *pls-sconst-0L*[*of sconst 0*, *symmetric*])
  apply (*rule genCngdd1-pls*)
  apply *auto*
  done

lemma *scale-Suc*: *scale* (*Suc n*) *s* = *pls s* (*scale n s*)
  by (*coinduction arbitrary*: *s rule*: *stream-coinduct1*) *auto*

lemma *scale-add*: *scale* (*m* + *n*) *s* = *pls* (*scale m s*) (*scale n s*)
  by (*induct m*) (*auto simp*: *scale-Suc pls-assoc*)

lemma *scale-mult*: *scale* (*m* ∗ *n*) *s* = *scale m* (*scale n s*)
  by (*induct m*) (*auto simp*: *scale-Suc scale-add*)

lemma *sup-empty*: *sup* {||} = *sconst 0*
  by (*coinduction rule*: *stream-coinduct1*) (*auto simp*: *fMax-def*)

lemma *Exp-pls*: *Exp* (*pls xs ys*) = *prd* (*Exp xs*) (*Exp ys*)
  by (*coinduction arbitrary*: *xs ys rule*: *stream-coinduct2*)
    (*auto simp*: *exp-def power-add prd-distribR pls-commute prd-assoc prd-commute-assoc*[*of*
Exp x **for** *x*]
      intro!: *genCngdd2-pls genCngdd2-prd*)

9 Two Examples of Fibonacci streams

definition *fibA* :: *stream* **where**

fibA = *corecUU1* (*λxs*. *GUARD1* (*0*, *PLS1* (*SCONS1* (*1*, *CONT1 xs*), *CONT1 xs*))) ()

lemma *head-fibA*[*simp*]: *head fibA* = *0*
unfolding *fibA-def corecUU1*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1′*)

lemma *tail-fibA*[*simp*]: *tail fibA* = *pls* (*SCons 1 fibA*) *fibA*
apply (*subst fibA-def*)
unfolding *corecUU1*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1′ eval1-𝔬𝔭1 algΛ1-Inr o-eq-dest*[*OF Abs-Σ1-natural*] *o-eq-dest*[*OF gg1-natural*] *o-eq-dest*[*OF eval1-gg1*] *pls-uniform fibA-def*)

lemma *fibA-code*[*code*]: *fibA* = *SCons 0* (*pls* (*SCons 1 fibA*) *fibA*)
by (*metis J.ctor-dtor prod.collapse head-fibA tail-fibA*)

definition *fibB* :: *stream* **where**
fibB = *corecUU1* (*λxs*. *PLS1* (*GUARD1* (*0*, (*SCONS1* (*1*, *CONT1 xs*))), *GUARD1* (*0*, *CONT1 xs*))) ()

lemma *fibB-code*[*code*]: *fibB* = *pls* (*SCons 0* (*SCons 1 fibB*)) (*SCons 0 fibB*)
apply (*subst fibB-def*)
unfolding *corecUU1*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval1-leaf1′ eval1-𝔬𝔭1 algΛ1-Inr o-eq-dest*[*OF Abs-Σ1-natural*] *o-eq-dest*[*OF gg1-natural*] *o-eq-dest*[*OF eval1-gg1*] *pls-uniform fibB-def*)

lemma *fibA* = *fibB*
proof (*coinduction rule*: *stream-coinduct1*)
case *Eq-stream*
have *?head* **by** (*subst fibB-code*) (*simp add*: *J.dtor-ctor*)
moreover
have *?tail* **by** (*subst* (*2*) *fibB-code*) (*auto simp add*: *J.dtor-ctor intro*: *genCngdd1-pls genCngdd1-SCons*)
ultimately show *?case* **..**
qed

10 Streams of Factorials

definition *facsA* = *corecUU2* (*λxs*. *PRD2* (*GUARD2* (*1*, *CONT2 xs*), *GUARD2* (*1*, *CONT2 xs*))) ()

lemma *facsA-code*[*code*]: *facsA* = *prd* (*SCons 1 facsA*) (*SCons 1 facsA*)
apply (*subst facsA-def*)
unfolding *corecUU2*

by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval2-leaf2′ eval2-𝔬𝔭2 algΛ2-Inr o-eq-dest*[*OF Abs-Σ2-natural*] *o-eq-dest*[*OF gg2-natural*] *o-eq-dest*[*OF eval2-gg2*] *prd-uniform facsA-def*)

definition *facsB* = *corecUU3* (*λxs*. *EXP3* (*K3.I* (*GUARD3* (*0*, *CONT3 xs*)))) ()

lemma *facsB-code*[*code*]: *facsB* = *Exp* (*SCons 0 facsB*)
apply (*subst facsB-def*)
unfolding *corecUU3*
by (*simp add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def J.dtor-ctor eval3-leaf3′ eval3-𝔬𝔭3 algΛ3-Inr o-eq-dest*[*OF Abs-Σ3-natural*] *o-eq-dest*[*OF gg3-natural*] *o-eq-dest*[*OF eval3-gg3*] *Exp-uniform facsB-def*)

lemma *head-facsB*[*simp*]: *head facsB* = *1*
by (*subst facsB-code*) (*simp add*: *J.dtor-ctor exp-def*)

lemma *tail-facsB*[*simp*]: *tail facsB* = *prd facsB facsB*
by (*subst facsB-code*, *subst tail-Exp*) (*simp add*: *J.dtor-ctor facsB-code*[*symmetric*])

lemma *facsA-facsB*: *SCons 1 facsA* = *facsB*
proof (*coinduction rule*: *stream-coinduct3*)
case *Eq-stream*
have *?head* **by** (*subst facsA-code*) (*simp add*: *J.dtor-ctor exp-def*)
moreover
have *?tail* **by** (*subst* (*2*) *facsA-code*) (*auto intro*!: *genCngdd3-prd simp*: *J.dtor-ctor*)
ultimately show *?case* **..**
qed

fun $facsC_{rec}$ **where**
$facsC_{rec}$ (*n*, *fn*, *i*) =
(*if i* = *0 then GUARD0* (*fn*, *CONT0* (*n* + *1*, *1*, *n* + *1*)) *else* $facsC_{rec}$ (*n*, *fn* ∗ *i*, *i* − *1*))

definition *facsC* = *corecUU0* $facsC_{rec}$ (*1*, *1*, *1*)

lemma $factsD_{rec}$*-code*:
corecUU0 $facsC_{rec}$ (*n*, *fn*, *i*) =
(*if i* = *0 then SCons fn* (*corecUU0* $facsC_{rec}$ (*n* + *1*, *1*, *n* + *1*))
else corecUU0 $facsC_{rec}$ (*n*, *fn* ∗ *i*, *i* − *1*))
by (*subst corecUU0*, *subst* $facsC_{rec}$*.simps*)
(*simp del*: $facsC_{rec}$*.simps add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def eval0-leaf0′ corecUU0*)

definition *fromN* = *dtor-corec-J* (*λn*. (*n*, *Inr* (*Suc n*)))

lemma *head-fromN*[*simp*]: *head* (*fromN n*) = *n*
unfolding *fromN-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

lemma *tail-fromN*[*simp*]: *tail* (*fromN n*) = *fromN* (*Suc n*)
unfolding *fromN-def J.dtor-corec map-pre-J-def BNF-Comp.id-bnf-comp-def* **by** *simp*

abbreviation *facsD n* ≡ *smap fact* (*fromN n*)

primrec *prds* **where**
prds 0 s = *s*
| *prds* (*Suc n*) *s* = *prd s* (*prds n s*)

lemma *head-prds*[*simp*]: *head* (*prds n s*) = *head s* ^ (*Suc n*)
by (*induct n*) *auto*

lemma *tail-prds-fac*[*simp*]: *tail* (*prds n facsB*) = *scale* (*Suc n*) (*prds* (*Suc n*) *facsB*)
by (*induct n*) (*auto simp*: *scale-Suc*, *auto simp*: *prd-distribL pls-ac-simps prd-ac-simps*)

lemma *facsD-facsB*: *facsD n* = *scale* (*fact n*) (*prds n facsB*)
proof (*coinduction arbitrary*: *n rule*: *stream-coinduct3*)
case *Eq-stream*
have *?head* **by** (*subst facsB-code*) (*simp add*: *J.dtor-ctor exp-def*)
moreover
have *?tail* **by** (*subst* (*2*) *facsB-code*) (*auto simp add*: *J.dtor-ctor facsB-code*[*symmetric*]
scale-mult[*symmetric*] *trans*[*OF mult.commute fact-Suc*[*symmetric*]]
simp del: *mult-Suc-right mult-Suc fact-Suc prds.simps*)
ultimately show *?case* **..**
qed

corollary *facsA* = *facsD 1*
unfolding *facsD-facsB facsA-facsB*[*symmetric*] **by** (*subst facsA-code*) (*simp add*: *scale-Suc*)

corollary *facsB* = *facsD 0*
unfolding *facsD-facsB* **by** (*simp add*: *scale-Suc*)

primrec *ffac* **where**
ffac fn 0 = *fn*
| *ffac fn* (*Suc i*) = *ffac* (*fn* ∗ *Suc i*) *i*

lemma *ffac-fact*: *ffac m n* = *m* ∗ *fact n*
by (*induct n arbitrary*: *m*) (*auto simp*: *algebra-simps*)

lemma *ffac-fact-Suc*: *ffac* (*Suc n*) *n* = *fact* (*Suc n*)
unfolding *ffac-fact fact-Suc* **..**

lemma *factsD*$_{rec}$*-facsD*: *corecUU0 facsC*$_{rec}$ (*n*, *fn*, *i*) = *SCons* (*ffac fn i*) (*facsD* (*n* + *1*))
proof (*coinduction arbitrary*: *n fn i rule*: *stream-coinduct*)
case *Eq-stream*

have *?head*
proof (*induct i arbitrary*: *fn*)
case *0* **then show** *?case* **by** (*subst factsD*$_{rec}$*-code*) (*simp add*: *J.dtor-ctor*)
next
case (*Suc i*) **then show** *?case* **by** (*subst factsD*$_{rec}$*-code*) *simp*
qed
moreover have *?tail*
proof (*induct i arbitrary*: *fn*)
case *0*
have *facsD* (*Suc n*) = *SCons* (*ffac* (*Suc n*) *n*) (*facsD* (*Suc* (*Suc n*)))
by (*coinduction rule*: *stream-coinduct0*) (*auto simp*: *J.dtor-ctor ffac-fact-Suc*)
then show *?case* **by** (*subst factsD*$_{rec}$*-code*) (*force simp*: *J.dtor-ctor*)
next
case (*Suc i*)
then show *?case* **by** (*subst factsD*$_{rec}$*-code*) *simp*
qed
ultimately show *?case* **by** *blast*
qed

lemma *facsC-facsD*: *facsC* = *facsD 1*
unfolding *facsC-def factsD*$_{rec}$*-facsD* **by** (*subst* (*2*) *smap-code*) *auto*

11 Mixed Recursion-Corecursion

function *primes*$_{rec}$ :: (*nat* ∗ *nat*) ⇒ (*stream* + *nat* ∗ *nat*) *ΣΣ0 F ΣΣ0* **where**
primes$_{rec}$ (*m*, *n*) =
(*if* (*m* = *0* ∧ *n* > *1*) ∨ *coprime m n then GUARD0* (*n*, *CONT0* (*m* ∗ *n*, *Suc n*))
else (*primes*$_{rec}$ (*m*, *Suc n*)))
by *pat-completeness auto*
termination
apply (*relation measure* (λ(*m*, *n*).
if n = *0 then 1 else if coprime m n then 0 else m* − *n mod m*))
apply (*auto simp*: *mod-Suc intro*: *Suc-lessI*)
apply (*metis One-nat-def coprime-Suc-nat gcd-nat.commute gcd-red-nat*)
apply (*metis diff-less-mono2 lessI mod-less-divisor*)
done

definition *primes* :: *nat* ⇒ *nat* ⇒ *stream* **where**
primes = *curry* (*corecUU0 primes*$_{rec}$)

lemma *primes-code*:
primes m n =
(*if* (*m* = *0* ∧ *n* > *1*) ∨ *coprime m n then SCons n* (*primes* (*m* ∗ *n*) (*Suc n*))
else primes m (*Suc n*))
unfolding *primes-def curry-def*
by (*subst corecUU0*, *subst primes*$_{rec}$*.simps*)

(*simp del*: $primes_{rec}$.*simps add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def eval0-leaf0' corecUU0*)

lemma *primes*: *primes 1 2* = *SCons 2* (*primes 2 3*)
by (*subst primes-code*) *auto*

fun $catalan_{rec}$ :: *nat* ⇒ (*stream* + *nat*) *ΣΣ1 F ΣΣ1* **where**
$catalan_{rec}$ *n* =
(*if n* > *0 then PLS1* ($catalan_{rec}$ (*n* − *1*), *GUARD1* (*0*, *CONT1* (*n*+*1*))) *else GUARD1* (*1*, *CONT1 1*))

definition *catalan* :: *nat* ⇒ *stream* **where**
catalan = *corecUU1* $catalan_{rec}$

lemma *catalan-code*:
catalan n =
(*if n* > *0 then pls* (*catalan* (*n* − *1*)) (*SCons 0* (*catalan* (*n* + *1*)))
else SCons 1 (*catalan 1*))
unfolding *catalan-def*
by (*subst corecUU1*, *subst* $catalan_{rec}$.*simps*)
(*simp del*: $catalan_{rec}$.*simps add*: *map-pre-J-def BNF-Comp.id-bnf-comp-def eval1-*`op`*1 eval1-leaf1' algΛ1-Inr o-eq-dest*[*OF Abs-Σ1-natural*] *corecUU1 pls-uniform*)