

Tree Examples

Jasmin Christian Blanchette Andrei Popescu
Dmitriy Traytel

July 8, 2014

Contents

1	Sum	1
2	Shuffle product	2
3	Coinduction Up-To Congruence	2
4	Proofs by Coinduction Up-To Congruence	5

1 Sum

definition *pls* :: *tree* \Rightarrow *tree* \Rightarrow *tree* **where**

pls xs ys = *dtor-corec-J* ($\lambda(xs, ys). (val\ xs + val\ ys, map\ Inr\ (zip\ (sub\ xs)\ (sub\ ys)))) (xs, ys)$)

lemma *val-pls[simp]*: *val* (*pls t u*) = *val t* + *val u*

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

lemma *sub-pls[simp]*: *sub* (*pls t u*) = *map* (*split pls*) (*zip* (*sub t*) (*sub u*))

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

lemma *pls-code[code]*: *pls t u* = *Node* (*val t* + *val u*) (*map* (*split pls*) (*zip* (*sub t*) (*sub u*)))

by (*metis J.ctor-dtor prod.collapse val-pls sub-pls*)

lemma *pls-uniform*: *pls t u* = *alg_{Q1}* (*t*, *u*)

unfolding *pls-def*

apply (*rule fun-cong[OF sym[OF J.dtor-corec-unique]]*)

unfolding *alg_{Q1}*

by (*simp add: map-pre-J-def BNF-Comp.id-bnf-comp-def fun-eq-iff convol-def Q1-def alg_{Q1}-def*)

2 Shuffle product

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

$prd\ t\ u = corecUU1\ (\lambda(t, u).\ GUARD1\ (val\ t * val\ u,$
 $\quad map\ (\lambda(t', u').\ PLS1\ (CONT1\ (t, u'), CONT1\ (t', u'))\ (zip\ (sub\ t)\ (sub\ u))))$
 (t, u)

lemma $val\text{-}prd[simp]: val\ (prd\ t\ u) = val\ t * val\ u$

unfolding $prd\text{-}def\ corecUU1$

by ($simp\ add: map\text{-}pre\text{-}J\text{-}def\ BNF\text{-}Comp.id\text{-}bnf\text{-}comp\text{-}def\ J.dtor\text{-}ctor\ eval1\text{-}leaf1'$)

lemma $sub\text{-}prd[simp]:$

$sub\ (prd\ t\ u) = map\ (\lambda(t', u').\ pls\ (prd\ t\ u')\ (prd\ t'\ u'))\ (zip\ (sub\ t)\ (sub\ u))$

apply ($subst\ prd\text{-}def$)

unfolding $corecUU1\ prod.case$

by ($simp\ add: map\text{-}pre\text{-}J\text{-}def\ BNF\text{-}Comp.id\text{-}bnf\text{-}comp\text{-}def\ J.dtor\text{-}ctor\ eval1\text{-}leaf1'$
 $eval1\text{-}op1\ alg\Lambda1\text{-}Inr\ o\text{-}eq\text{-}dest[OF\ Abs\text{-}\Sigma1\text{-}natural]\ pls\text{-}uniform\ prd\text{-}def\ split\text{-}beta$)

lemma $prd\text{-}code[code]: prd\ t\ u =$

$Node\ (val\ t * val\ u)\ (map\ (\lambda(t', u').\ pls\ (prd\ t\ u')\ (prd\ t'\ u'))\ (zip\ (sub\ t)\ (sub\ u)))$

by ($metis\ J.ctor\text{-}dctor\ prod.collapse\ val\text{-}prd\ sub\text{-}prd$)

lemma $prd\text{-}uniform: prd\ t\ u = alg\varrho2\ (t, u)$

unfolding $prd\text{-}def$

apply ($rule\ fun\text{-}cong[OF\ sym[OF\ corecUU1\text{-}unique]]$)

apply ($rule\ iffD1[OF\ dtor\text{-}J\text{-}o\text{-}inj]$)

unfolding $alg\varrho2$

apply ($simp\ add: map\text{-}pre\text{-}J\text{-}def\ BNF\text{-}Comp.id\text{-}bnf\text{-}comp\text{-}def\ fun\text{-}eq\text{-}iff\ J.dtor\text{-}ctor$
 $\varrho2\text{-}def\ Let\text{-}def\ convol\text{-}def\ eval2\text{-}op2\ eval1\text{-}op1\ eval1\text{-}leaf1'$
 $o\text{-}eq\text{-}dest[OF\ Abs\text{-}\Sigma1\text{-}natural]\ o\text{-}eq\text{-}dest[OF\ Abs\text{-}\Sigma2\text{-}natural]\ alg\Lambda2\text{-}Inl\ alg\varrho2\text{-}def$
 $split\text{-}beta$)

done

3 Coinduction Up-To Congruence

lemma $Node\text{-}uniform: Node\ x\ ts = eval0\ (gg0\ (x, map\ leaf0\ ts))$

by ($rule\ iffD1[OF\ J.dtor\text{-}inject]$)

($simp\ add: map\text{-}pre\text{-}J\text{-}def\ BNF\text{-}Comp.id\text{-}bnf\text{-}comp\text{-}def\ J.dtor\text{-}ctor\ o\text{-}eq\text{-}dest[OF\ eval0\text{-}gg0]\ eval0\text{-}leaf0$)

lemma $genCngdd0\text{-}Node: \llbracket x1 = x2; list\text{-}all2\ (genCngdd0\ R)\ ts1\ ts2 \rrbracket \implies$

$genCngdd0\ R\ (Node\ x1\ ts1)\ (Node\ x2\ ts2)$

unfolding $Node\text{-}uniform$

apply ($rule\ genCngdd0\text{-}eval0$)

apply ($rule\ rel\text{-}funD[OF\ gg0\text{-}transfer]$)

unfolding $rel\text{-}pre\text{-}J\text{-}def\ BNF\text{-}Comp.id\text{-}bnf\text{-}comp\text{-}def\ vimage2p\text{-}def$

apply ($rule\ rel\text{-}funD[OF\ rel\text{-}funD[OF\ Pair\text{-}transfer], rotated]$)

apply ($erule\ rel\text{-}funD[OF\ rel\text{-}funD[OF\ map\text{-}transfer], rotated]$)

```

apply (rule leaf0-transfer)
apply assumption
done

lemma genCngdd0-genCngdd1:  $\text{genCngdd0 } R \text{ } xs \text{ } ys \implies \text{genCngdd1 } R \text{ } xs \text{ } ys$ 
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-Node:  $\llbracket x1 = x2; \text{list-all2 } (\text{genCngdd1 } R) \text{ } ts1 \text{ } ts2 \rrbracket \implies$ 
 $\text{genCngdd1 } R \text{ } (\text{Node } x1 \text{ } ts1) \text{ } (\text{Node } x2 \text{ } ts2)$ 
apply (subst I1.idem-CI[symmetric])
apply (rule genCngdd0-genCngdd1)
apply (rule genCngdd0-Node)
apply (auto intro: predicate2D[OF list.rel-mono])
done

lemma genCngdd1-pls:  $\llbracket \text{genCngdd1 } R \text{ } xs1 \text{ } xs2; \text{genCngdd1 } R \text{ } ys1 \text{ } ys2 \rrbracket \implies$ 
 $\text{genCngdd1 } R \text{ } (\text{pls } xs1 \text{ } ys1) \text{ } (\text{pls } xs2 \text{ } ys2)$ 
unfolding pls-uniform alg01-def o-apply
apply (rule genCngdd1-eval1)
apply (rule rel-funD[OF K1-as- $\Sigma\Sigma 1$ -transfer])
apply simp
done

lemma genCngdd1-genCngdd2:  $\text{genCngdd1 } R \text{ } xs \text{ } ys \implies \text{genCngdd2 } R \text{ } xs \text{ } ys$ 
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-Node:  $\llbracket x1 = x2; \text{list-all2 } (\text{genCngdd2 } R) \text{ } ts1 \text{ } ts2 \rrbracket \implies$ 
 $\text{genCngdd2 } R \text{ } (\text{Node } x1 \text{ } ts1) \text{ } (\text{Node } x2 \text{ } ts2)$ 
apply (subst I2.idem-CI[symmetric])
apply (rule genCngdd1-genCngdd2)
apply (rule genCngdd1-Node)
apply (auto intro: predicate2D[OF list.rel-mono])
done

```

```

lemma genCngdd2-pls:  $\llbracket \text{genCngdd2 } R \text{ } xs1 \text{ } xs2; \text{genCngdd2 } R \text{ } ys1 \text{ } ys2 \rrbracket \implies$ 
  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:  $\llbracket \text{genCngdd2 } R \text{ } xs1 \text{ } xs2; \text{genCngdd2 } R \text{ } ys1 \text{ } ys2 \rrbracket \implies$ 
  genCngdd2 R (prd xs1 ys1) (prd xs2 ys2)
unfolding prd-uniform algQ2-def o-apply
apply (rule genCngdd2-eval2)
apply (rule rel-funD[OF K2-as-ΣΣ2-transfer])
apply simp
done

```

```

lemma tree-coinduct[case-names Eq-tree, case-conclusion Eq-tree val sub]:
  assumes  $R \text{ } s \text{ } s' \wedge s \text{ } s' \implies \text{val } s = \text{val } s' \wedge \text{list-all2 } R \text{ } (sub \text{ } s) \text{ } (sub \text{ } s')$ 
  shows  $s = s'$ 
using assms(1) proof (rule mp[OF J.dtor-coinduct, rotated], safe)
  fix a b
  assume  $R \text{ } a \text{ } b$ 
  from assms(2)[OF this] show  $F\text{-rel } R \text{ } (dtor\text{-}J \text{ } a) \text{ } (dtor\text{-}J \text{ } b)$ 
  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 tree-coinduct0[case-names Eq-tree, case-conclusion Eq-tree val sub]:
  assumes  $R \text{ } s \text{ } s' \wedge s \text{ } s' \implies \text{val } s = \text{val } s' \wedge \text{list-all2 } (genCngdd0 \text{ } R) \text{ } (sub$ 
  s) (sub s')
  shows  $s = s'$ 
using assms(1) proof (rule mp[OF coinductionU-genCngdd0, rotated], safe)
  fix a b
  assume  $R \text{ } a \text{ } b$ 
  from assms(2)[OF this] show  $F\text{-rel } (genCngdd0 \text{ } R) \text{ } (dtor\text{-}J \text{ } a) \text{ } (dtor\text{-}J \text{ } b)$ 
  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 tree-coinduct1[case-names Eq-tree, case-conclusion Eq-tree val sub]:
  assumes  $R \text{ } s \text{ } s' \wedge s \text{ } s' \implies \text{val } s = \text{val } s' \wedge \text{list-all2 } (genCngdd1 \text{ } R) \text{ } (sub$ 
  s) (sub s')
  shows  $s = s'$ 
using assms(1) proof (rule mp[OF coinductionU-genCngdd1, rotated], safe)
  fix a b
  assume  $R \text{ } a \text{ } b$ 
  from assms(2)[OF this] show  $F\text{-rel } (genCngdd1 \text{ } R) \text{ } (dtor\text{-}J \text{ } a) \text{ } (dtor\text{-}J \text{ } b)$ 
  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 tree-coinduct2[case-names Eq-tree, case-conclusion Eq-tree val sub]:
  assumes  $R\ s\ s' \wedge s\ s' \implies \text{val } s = \text{val } s' \wedge \text{list-all2 } (\text{genCngdd2 } R) (\text{sub } s) (\text{sub } s')$ 
  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  $F\text{-rel } (\text{genCngdd2 } R) (\text{dtor-J } a) (\text{dtor-J } b)$ 
  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

```

4 Proofs by Coinduction Up-To Congruence

```

lemma pls-assoc: pls (pls t u) zs = pls t (pls u zs)
  by (coinduction arbitrary: t u zs rule: tree-coinduct) (force simp: list-all2-iff in-set-zip)

```

```

lemma pls-commute: pls t u = pls u t
  by (coinduction arbitrary: t u rule: tree-coinduct) (force simp: list-all2-iff in-set-zip)

```

```

lemma pls-commute-assoc: pls t (pls u zs) = pls u (pls t zs)
  by (metis pls-assoc pls-commute)

```

```

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

```

```

lemma prd-commute: prd t u = prd u t
proof (coinduction arbitrary: t u rule: tree-coinduct1)
  case Eq-tree
  have ?sub unfolding sub-prd
  by (subst pls-commute) (fastforce simp: list-all2-iff in-set-zip intro!: genCngdd1-pls)
  then show ?case by simp
qed

```

```

lemma prd-distribL: prd xs (pls ys zs) = pls (prd xs ys) (prd xs zs)
proof (coinduction arbitrary: xs ys zs rule: tree-coinduct1)
  case Eq-tree
  have  $\bigwedge a\ b\ c\ d. \text{pls } (\text{pls } a\ b) (\text{pls } c\ d) = \text{pls } (\text{pls } a\ c) (\text{pls } b\ d)$  by (metis pls-assoc pls-commute)
  then have ?sub by (fastforce simp: list-all2-iff in-set-zip 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: tree-coinduct1)
  case Eq-tree

```

have $\bigwedge a\ b\ c\ d. \text{pls} (\text{pls}\ a\ b) (\text{pls}\ c\ d) = \text{pls} (\text{pls}\ a\ c) (\text{pls}\ b\ d)$ **by** (*metis pls-assoc pls-commute*)
then have $?sub$ **by** (*fastforce simp: list-all2-iff in-set-zip intro!: genCngdd1-pls*)
then show $?case$ **by** (*simp add: algebra-simps*)
qed

lemma *prd-assoc*: $\text{prd} (\text{prd}\ xs\ ys)\ zs = \text{prd}\ xs\ (\text{prd}\ ys\ zs)$
proof (*coinduction arbitrary: xs ys zs rule: tree-coinduct1*)
case *Eq-tree*
have $?sub$ **unfolding** *sub-prd zip-map1 zip-map2 list.map-comp*
by (*fastforce simp: list-all2-iff in-set-zip pls-ac-simps prd-distribL prd-distribR intro!: genCngdd1-pls*)
then show $?case$ **by** *simp*
qed

lemma *prd-commute-assoc*: $\text{prd}\ xs\ (\text{prd}\ ys\ zs) = \text{prd}\ ys\ (\text{prd}\ xs\ zs)$
by (*metis prd-assoc prd-commute*)

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