

Operations on Bounded Natural Functors

Jasmin Christian Blanchette Andrei Popescu Dmitriy Traytel

November 24, 2021

Abstract

This entry formalizes the closure property of bounded natural functors (BNFs) under seven operations. These operations and the corresponding proofs constitute the core of Isabelle’s (co)datatype package. To be close to the implemented tactics, the proofs are deliberately formulated as detailed apply scripts. The (co)datatypes together with (co)induction principles and (co)recursors are byproducts of the fixpoint operations LFP and GFP. Composition of BNFs is subdivided into four simpler operations: Compose, Kill, Lift, and Permute. The N2M operation provides mutual (co)induction principles and (co)recursors for nested (co)datatypes.

Contents

1	Least Fixpoint (a.k.a. Datatype)	1
1.1	Algebra	2
1.2	Morphism	3
1.3	Bounds	6
1.4	Minimal Algebras	7
1.5	Initiality	18
1.6	Initial Algebras	18
1.7	The datatype	26
1.8	The Result as an BNF	35
2	Greatest Fixpoint (a.k.a. Codatatype)	51
2.1	Coalgebra	53
2.2	Type-coalgebra	53
2.3	Morphism	54
2.4	Bisimulations	57
2.5	The Tree Coalgebra	65
2.6	Quotient Coalgebra	106
2.7	Coinduction	115
2.8	The Result as an BNF	116
3	Normalized Composition of BNFs	144
4	Removing Live Variables	147
5	Adding New Live Variables	149
6	Changing the Order of Live Variables	151
7	Mutual View on Nested Datatypes	152
7.1	Nested Definition	152
7.2	Isomorphic Mutual Definition	153
7.3	Mutualization	153
7.3.1	Iterators	153
7.3.2	Recursors	154
7.3.3	Induction	154

8	Mutual View on Nested Coatypes	155
8.1	Nested definition	155
8.2	Isomorphic Mutual Definition	155
8.3	Mutualization	155
8.3.1	Coiterators	155
8.3.2	Corecursors	156
8.3.3	Coinduction	157

1 Least Fixpoint (a.k.a. Datatype)

unbundle *cardinal_syntax*

ML $\langle open\ Ctr_Sugar_Util \rangle$
notation *BNF_Def.convolve* ($\langle_ , _ \rangle$)

'b1 = ('a, 'b1, 'b2) F1
'b2 = ('a, 'b1, 'b2) F2

To build a witness scenario, let us assume

('a, 'b1, 'b2) F1 = 'a * 'b1 + 'a * 'b2
('a, 'b1, 'b2) F2 = unit + 'b1 * 'b2

declare $[[bnf_internals]]$

bnf-axiomatization (*F1set1: 'a, F1set2: 'b1, F1set3: 'b2*) *F1*
[wits: 'a \Rightarrow 'b1 \Rightarrow ('a, 'b1, 'b2) F1 'a \Rightarrow 'b2 \Rightarrow ('a, 'b1, 'b2) F1]
for map: *F1map rel: F1rel*

bnf-axiomatization (*F2set1: 'a, F2set2: 'b1, F2set3: 'b2*) *F2*
[wits: ('a, 'b1, 'b2) F2]
for map: *F2map rel: F2rel*

abbreviation *F1in :: 'a1 set \Rightarrow 'a2 set \Rightarrow 'a3 set \Rightarrow (('a1, 'a2, 'a3) F1) set* **where**
F1in A1 A2 A3 \equiv {x. F1set1 x \subseteq A1 \wedge F1set2 x \subseteq A2 \wedge F1set3 x \subseteq A3}

abbreviation *F2in :: 'a1 set \Rightarrow 'a2 set \Rightarrow 'a3 set \Rightarrow (('a1, 'a2, 'a3) F2) set* **where**
F2in A1 A2 A3 \equiv {x. F2set1 x \subseteq A1 \wedge F2set2 x \subseteq A2 \wedge F2set3 x \subseteq A3}

lemma *F1map_comp_id: F1map g1 g2 g3 (F1map id f2 f3 x) = F1map g1 (g2 o f2) (g3 o f3) x*

apply (*rule trans*)
apply (*rule F1.map_comp*)
unfolding *o_id*
apply (*rule refl*)
done

lemmas *F1in_mono23 = F1.in_mono[OF subset_refl]*

lemma *F1map_congL: $[\forall a \in F1set2\ x. f\ a = a; \forall a \in F1set3\ x. g\ a = a] \Rightarrow$*

F1map id f g x = x
apply (*rule trans*)
apply (*rule F1.map_cong0*)
apply (*rule refl*)
apply (*rule trans*)
apply (*erule bspec*)
apply *assumption*
apply (*rule sym*)
apply (*rule id_apply*)
apply (*rule trans*)
apply (*erule bspec*)
apply *assumption*
apply (*rule sym*)
apply (*rule id_apply*)
apply (*rule F1.map_id*)
done

lemma *F2map_comp_id*: $F2map\ g1\ g2\ g3\ (F2map\ id\ f2\ f3\ x) = F2map\ g1\ (g2\ o\ f2)\ (g3\ o\ f3)\ x$
apply (*rule trans*)
apply (*rule F2.map_comp*)
unfolding *o_id*
apply (*rule refl*)
done

lemmas *F2in_mono23* = *F2.in_mono*[*OF subset_refl*]

lemma *F2map_congL*: $\llbracket \forall a \in F2set2\ x.\ f\ a = a; \forall a \in F2set3\ x.\ g\ a = a \rrbracket \implies$
 $F2map\ id\ f\ g\ x = x$
apply (*rule trans*)
apply (*rule F2.map_cong0*)
apply (*rule refl*)
apply (*rule trans*)
apply (*erule bspec*)
apply *assumption*
apply (*rule sym*)
apply (*rule id_apply*)
apply (*rule trans*)
apply (*erule bspec*)
apply *assumption*
apply (*rule sym*)
apply (*rule id_apply*)
apply (*rule F2.map_id*)
done

1.1 Algebra

definition *alg where*

alg B1 B2 s1 s2 =
 $(\forall x \in F1in\ (UNIV :: 'a\ set)\ B1\ B2.\ s1\ x \in B1) \wedge (\forall y \in F2in\ (UNIV :: 'a\ set)\ B1\ B2.\ s2\ y \in B2)$

lemma *alg_F1set*: $\llbracket alg\ B1\ B2\ s1\ s2; F1set2\ x \subseteq B1; F1set3\ x \subseteq B2 \rrbracket \implies s1\ x \in B1$
apply (*tactic <dtac @{\context} @{\thm iffD1[OF alg_def]} 1>*)
apply (*erule conjE*)
apply (*erule bspec*)
apply (*rule CollectI*)
apply (*rule conjI[OF subset_UNIV]*)
apply (*erule conjI*)
apply *assumption*
done

lemma *alg_F2set*: $\llbracket alg\ B1\ B2\ s1\ s2; F2set2\ x \subseteq B1; F2set3\ x \subseteq B2 \rrbracket \implies s2\ x \in B2$
apply (*tactic <dtac @{\context} @{\thm iffD1[OF alg_def]} 1>*)
apply (*erule conjE*)
apply (*erule bspec*)
apply (*rule CollectI*)
apply (*rule conjI[OF subset_UNIV]*)
apply (*erule conjI*)
apply *assumption*
done

lemma *alg_not_empty*:

alg B1 B2 s1 s2 $\implies B1 \neq \{\} \wedge B2 \neq \{\}$
apply (*rule conjI*)
apply (*rule notI*)
apply (*tactic <hyp_subst_tac @{\context} 1>*)
apply (*frule alg_F1set*)

apply (*rule subset_emptyI*)

```

apply (erule F1.wit1 F1.wit2 F2.wit)

apply (rule subsetI)
apply (drule F1.wit1 F1.wit2 F2.wit)

apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (tactic ⟨FIRST' (map (fn thm => rtac @ {context} thm THEN' assume_tac @ {context}) @ {thms alg_F1set
alg_F2set}) 1⟩)

apply (rule subset_emptyI)
apply (erule F1.wit1 F1.wit2 F2.wit)

apply (rule subsetI)
apply (drule F1.wit1 F1.wit2 F2.wit)
apply (erule FalseE)

apply (erule emptyE)

apply (rule notI)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (drule alg_F2set)

apply (rule subsetI)
apply (rule FalseE)
apply (erule F1.wit1 F1.wit2 F2.wit)

apply (rule subset_emptyI)
apply (erule F1.wit1 F1.wit2 F2.wit)

apply (erule emptyE)
done

```

1.2 Morphism

definition mor where

$$\begin{aligned}
\text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g = & \\
& (((\forall a \in B1. f \ a \in B1') \wedge (\forall a \in B2. g \ a \in B2')) \wedge \\
& ((\forall z \in F1in \ (UNIV :: 'a \ set) \ B1 \ B2. f \ (s1 \ z) = s1' \ (F1map \ id \ f \ g \ z)) \wedge \\
& (\forall z \in F2in \ (UNIV :: 'a \ set) \ B1 \ B2. g \ (s2 \ z) = s2' \ (F2map \ id \ f \ g \ z))))
\end{aligned}$$

lemma morE1: $\llbracket \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g; z \in F1in \ UNIV \ B1 \ B2 \rrbracket$
 $\implies f \ (s1 \ z) = s1' \ (F1map \ id \ f \ g \ z)$
apply (tactic ⟨dtac @ {context} @ {thm iffD1[OF mor_def]} 1⟩)
apply (erule conjE)+
apply (erule bspec)
apply assumption
done

lemma morE2: $\llbracket \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g; z \in F2in \ UNIV \ B1 \ B2 \rrbracket$
 $\implies g \ (s2 \ z) = s2' \ (F2map \ id \ f \ g \ z)$
apply (tactic ⟨dtac @ {context} @ {thm iffD1[OF mor_def]} 1⟩)
apply (erule conjE)+
apply (erule bspec)
apply assumption
done

lemma mor_incl: $\llbracket B1 \subseteq B1'; B2 \subseteq B2' \rrbracket \implies \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1 \ s2 \ id \ id$
apply (tactic ⟨rtac @ {context} (@ {thm mor_def}) RS iffD2) 1⟩
apply (rule conjI)

apply (rule conjI)
apply (rule ballI)

apply (erule subsetD)
apply (erule ssubst_mem[OF id_apply])

apply (rule ballI)
apply (erule subsetD)
apply (erule ssubst_mem[OF id_apply])

apply (rule conjI)
apply (rule ballI)
apply (rule trans)
apply (rule id_apply)
apply (tactic <stac @ {context} @ {thm F1.map_id} 1 >)
apply (rule refl)

apply (rule ballI)
apply (rule trans)
apply (rule id_apply)
apply (tactic <stac @ {context} @ {thm F2.map_id} 1 >)
apply (rule refl)
done

lemma mor_comp:

$\llbracket \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g; \\ \text{mor } B1' \ B2' \ s1' \ s2' \ B1'' \ B2'' \ s1'' \ s2'' \ f' \ g' \rrbracket \implies \\ \text{mor } B1 \ B2 \ s1 \ s2 \ B1'' \ B2'' \ s1'' \ s2'' \ (f' \circ f) \ (g' \circ g)$
apply (tactic <dtac @ {context} (@ {thm mor_def} RS iffD1) 1 >)
apply (tactic <dtac @ {context} (@ {thm mor_def} RS iffD1) 1 >)
apply (tactic <rtac @ {context} (@ {thm mor_def} RS iffD2) 1 >)
apply (erule conjE)+
apply (rule conjI)

apply (rule conjI)
apply (rule ballI)
apply (erule ssubst_mem[OF o_apply])
apply (erule bspec)
apply (erule bspec)
apply assumption

apply (rule ballI)
apply (erule ssubst_mem[OF o_apply])
apply (erule bspec)
apply (erule bspec)
apply assumption

apply (rule conjI)
apply (rule ballI)
apply (rule trans[OF o_apply])
apply (rule trans)
apply (rule trans)
apply (erule bspec[rotated])
apply assumption
apply (erule arg_cong)
apply (erule CollectE conjE)+
apply (erule bspec)
apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule conjI)
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(2))
apply (rule image_subsetI)
apply (erule bspec)
apply (erule subsetD)

```

apply assumption
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(3))
apply (rule image_subsetI)
apply (erule bspec)
apply (erule subsetD)
apply assumption
apply (rule arg_cong[OF F1map_comp_id])

```

```

apply (rule ballI)
apply (rule trans[OF o_apply])
apply (rule trans)
apply (rule trans)
apply (drule bspec[rotated])
apply assumption
apply (erule arg_cong)
apply (erule CollectE conjE)+
apply (erule bspec)
apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule conjI)
apply (rule ord_eq_le_trans)
apply (rule F2.set_map(2))
apply (rule image_subsetI)
apply (erule bspec)
apply (erule subsetD)
apply assumption
apply (rule ord_eq_le_trans)
apply (rule F2.set_map(3))
apply (rule image_subsetI)
apply (erule bspec)
apply (erule subsetD)
apply assumption
apply (rule arg_cong[OF F2map_comp_id])
done

```

```

lemma mor_cong:  $\llbracket f' = f; g' = g; \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g \rrbracket \implies$ 
 $\text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f' \ g'$ 
apply (tactic <hyp_subst_tac @ {context} 1>)
apply assumption
done

```

```

lemma mor_str:
 $\text{mor } UNIV \ UNIV \ (F1map \ id \ s1 \ s2) \ (F2map \ id \ s1 \ s2) \ UNIV \ UNIV \ s1 \ s2 \ s1 \ s2$ 
apply (rule iffD2)
apply (rule mor_def)
apply (rule conjI)
apply (rule conjI)
apply (rule ballI)
apply (rule UNIV_I)
apply (rule ballI)
apply (rule UNIV_I)

```

```

apply (rule conjI)
apply (rule ballI)
apply (rule refl)
apply (rule ballI)
apply (rule refl)
done

```

1.3 Bounds

```

type-synonym  $bd\_type\_F1' = bd\_type\_F1 + (bd\_type\_F1, bd\_type\_F1, bd\_type\_F1) \ F1$ 

```

type-synonym $bd_type_F2' = bd_type_F2 + (bd_type_F2, bd_type_F2, bd_type_F2) F2$

type-synonym $SucFbd_type = ((bd_type_F1' + bd_type_F2') set)$

type-synonym $'a1 ASucFbd_type = (SucFbd_type \Rightarrow ('a1 + bool))$

abbreviation $F1bd' \equiv bd_F1 + c \mid UNIV :: (bd_type_F1, bd_type_F1, bd_type_F1) F1 set$

lemma $F1set1_bd_incr: \bigwedge x. |F1set1 x| \leq o F1bd'$

by (rule ordLeq_transitive[OF F1.set_bd(1) ordLeq_csum1[OF F1.bd_Card_order]])

lemma $F1set2_bd_incr: \bigwedge x. |F1set2 x| \leq o F1bd'$

by (rule ordLeq_transitive[OF F1.set_bd(2) ordLeq_csum1[OF F1.bd_Card_order]])

lemma $F1set3_bd_incr: \bigwedge x. |F1set3 x| \leq o F1bd'$

by (rule ordLeq_transitive[OF F1.set_bd(3) ordLeq_csum1[OF F1.bd_Card_order]])

lemmas $F1bd'_Card_order = Card_order_csum$

lemmas $F1bd'_Cinfinite = Cinfinite_csum1[OF F1.bd_Cinfinite]$

lemmas $F1bd'_Cnotzero = Cinfinite_Cnotzero[OF F1bd'_Cinfinite]$

lemmas $F1bd'_card_order = card_order_csum[OF F1.bd_card_order card_of_card_order_on]$

abbreviation $F2bd' \equiv bd_F2 + c \mid UNIV :: (bd_type_F2, bd_type_F2, bd_type_F2) F2 set$

lemma $F2set1_bd_incr: \bigwedge x. |F2set1 x| \leq o F2bd'$

by (rule ordLeq_transitive[OF F2.set_bd(1) ordLeq_csum1[OF F2.bd_Card_order]])

lemma $F2set2_bd_incr: \bigwedge x. |F2set2 x| \leq o F2bd'$

by (rule ordLeq_transitive[OF F2.set_bd(2) ordLeq_csum1[OF F2.bd_Card_order]])

lemma $F2set3_bd_incr: \bigwedge x. |F2set3 x| \leq o F2bd'$

by (rule ordLeq_transitive[OF F2.set_bd(3) ordLeq_csum1[OF F2.bd_Card_order]])

lemmas $F2bd'_Card_order = Card_order_csum$

lemmas $F2bd'_Cinfinite = Cinfinite_csum1[OF F2.bd_Cinfinite]$

lemmas $F2bd'_Cnotzero = Cinfinite_Cnotzero[OF F2bd'_Cinfinite]$

lemmas $F2bd'_card_order = card_order_csum[OF F2.bd_card_order card_of_card_order_on]$

abbreviation $SucFbd$ where $SucFbd \equiv cardSuc (F1bd' + c F2bd')$

abbreviation $ASucFbd$ where $ASucFbd \equiv (\mid UNIV \mid + c ctwo) \hat{c} SucFbd$

lemma $F1set1_bd: |F1set1 x| \leq o bd_F1 + c bd_F2$

apply (rule ordLeq_transitive)

apply (rule F1.set_bd(1))

apply (rule ordLeq_csum1)

apply (rule F1.bd_Card_order)

done

lemma $F1set2_bd: |F1set2 x| \leq o bd_F1 + c bd_F2$

apply (rule ordLeq_transitive)

apply (rule F1.set_bd(2))

apply (rule ordLeq_csum1)

apply (rule F1.bd_Card_order)

done

lemma $F1set3_bd: |F1set3 x| \leq o bd_F1 + c bd_F2$

apply (rule ordLeq_transitive)

apply (rule F1.set_bd(3))

apply (rule ordLeq_csum1)

apply (rule F1.bd_Card_order)

done

lemma $F2set1_bd: |F2set1 x| \leq o bd_F1 + c bd_F2$

apply (rule ordLeq_transitive)

apply (rule F2.set_bd(1))

apply (rule ordLeq_csum2)

apply (rule F2.bd_Card_order)

done

lemma $F2set2_bd: |F2set2 x| \leq o bd_F1 + c bd_F2$

apply (rule ordLeq_transitive)

apply (rule *F2.set_bd(2)*)
apply (rule *ordLeq_csum2*)
apply (rule *F2.bd_Card_order*)
done

lemma *F2set3_bd*: $|F2set3\ x| \leq o\ bd_F1 + c\ bd_F2$
apply (rule *ordLeq_transitive*)
apply (rule *F2.set_bd(3)*)
apply (rule *ordLeq_csum2*)
apply (rule *F2.bd_Card_order*)
done

lemmas *SucFbd_Card_order* = *cardSuc_Card_order*[*OF Card_order_csum*]
lemmas *SucFbd_Cinfinite* = *Cinfinite_cardSuc*[*OF Cinfinite_csum1* [*OF F1bd'_Cinfinite*]]
lemmas *SucFbd_Cnotzero* = *Cinfinite_Cnotzero*[*OF SucFbd_Cinfinite*]
lemmas *worel_SucFbd* = *Card_order_wo_rel*[*OF SucFbd_Card_order*]
lemmas *ASucFbd_Cinfinite* = *Cinfinite_cexp*[*OF ordLeq_csum2* [*OF Card_order_ctwo*] *SucFbd_Cinfinite*]

1.4 Minimal Algebras

abbreviation *min_G1* **where**
 $min_G1\ As1_As2\ i \equiv (\bigcup j \in underS\ SucFbd\ i.\ fst\ (As1_As2\ j))$

abbreviation *min_G2* **where**
 $min_G2\ As1_As2\ i \equiv (\bigcup j \in underS\ SucFbd\ i.\ snd\ (As1_As2\ j))$

abbreviation *min_H* **where**
 $min_H\ s1\ s2\ As1_As2\ i \equiv$
 $(min_G1\ As1_As2\ i \cup s1\ ' (F1in\ (UNIV :: 'a\ set)\ (min_G1\ As1_As2\ i)\ (min_G2\ As1_As2\ i)),$
 $min_G2\ As1_As2\ i \cup s2\ ' (F2in\ (UNIV :: 'a\ set)\ (min_G1\ As1_As2\ i)\ (min_G2\ As1_As2\ i)))$

abbreviation *min_algs* **where**
 $min_algs\ s1\ s2 \equiv wo_rel.worec\ SucFbd\ (min_H\ s1\ s2)$

definition *min_alg1* **where**
 $min_alg1\ s1\ s2 = (\bigcup i \in Field\ SucFbd.\ fst\ (min_algs\ s1\ s2\ i))$

definition *min_alg2* **where**
 $min_alg2\ s1\ s2 = (\bigcup i \in Field\ SucFbd.\ snd\ (min_algs\ s1\ s2\ i))$

lemma *min_algs*:
 $i \in Field\ SucFbd \implies min_algs\ s1\ s2\ i = min_H\ s1\ s2\ (min_algs\ s1\ s2)\ i$
apply (rule *fun_cong*[*OF wo_rel.worec_fixpoint*[*OF worel_SucFbd*]])
apply (rule *iffD2*)
apply (rule *meta_eq_to_obj_eq*)
apply (rule *wo_rel.adm_wo_def*[*OF worel_SucFbd*])
apply (rule *allI*)
apply (rule *impI*)

apply (rule *iffD2*)
apply (rule *prod.inject*)
apply (rule *conjI*)

apply (rule *arg_cong2*[*of _ _ _ _ (U)*])
apply (rule *SUP_cong*)
apply (rule *refl*)
apply (rule *bspec*)
apply *assumption*
apply (rule *arg_cong*)

apply (rule *image_cong*)
apply (rule *arg_cong2*[*of _ _ _ _ F1in UNIV*])
apply (rule *SUP_cong*)
apply (rule *refl*)


```

apply (drule bspec)
apply assumption
apply (erule arg_cong)
apply (rule SUP_cong)
apply (rule refl)
apply (drule bspec)
apply assumption
apply (erule arg_cong)
apply (rule refl)

apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule SUP_cong)
apply (rule refl)
apply (drule bspec)
apply assumption
apply (erule arg_cong)

apply (rule image_cong)
apply (rule arg_cong2[of _ _ _ _ F2in UNIV])
apply (rule SUP_cong)
apply (rule refl)
apply (drule bspec)
apply assumption
apply (erule arg_cong)
apply (rule SUP_cong)
apply (rule refl)
apply (drule bspec)
apply assumption
apply (erule arg_cong)
apply (rule refl)
done

corollary min_algs1:  $i \in \text{Field SucFbd} \implies \text{fst} (\text{min\_algs } s1 \ s2 \ i) =$ 
  min_G1 (min_algs s1 s2) i  $\cup$ 
  s1 ' (F1in UNIV (min_G1 (min_algs s1 s2) i) (min_G2 (min_algs s1 s2) i))
apply (rule trans)
apply (erule arg_cong[OF min_algs])
apply (rule fst_conv)
done

corollary min_algs2:  $i \in \text{Field SucFbd} \implies \text{snd} (\text{min\_algs } s1 \ s2 \ i) =$ 
  min_G2 (min_algs s1 s2) i  $\cup$ 
  s2 ' (F2in UNIV (min_G1 (min_algs s1 s2) i) (min_G2 (min_algs s1 s2) i))
apply (rule trans)
apply (erule arg_cong[OF min_algs])
apply (rule snd_conv)
done

lemma min_algs_mono1: relChain SucFbd (%i. fst (min_algs s1 s2 i))
apply (tactic <rtac @ {context} @ {thm iffD2[OF meta_eq_to_obj_eq[OF relChain_def]]} 1>)
apply (rule allI)+
apply (rule impI)
apply (rule case_split)
apply (rule xt1(3))
apply (rule min_algs1)
apply (erule FieldI2)
apply (rule subsetI)
apply (rule UnI1)
apply (rule UN_I)
apply (erule underS_I)
apply assumption
apply assumption
apply (rule equalityD1)

```

```

apply (drule notnotD)
apply (erule arg_cong)
done

```

```

lemma min_algs_mono2: relChain SucFbd (%i. snd (min_algs s1 s2 i))
apply (tactic <rtac @{context} @<{thm iffD2[OF meta_eq_to_obj_eq[OF relChain_def]]} 1>)
apply (rule allI)+
apply (rule impI)
apply (rule case_split)
apply (rule xt1(3))
apply (rule min_algs2)
apply (erule FieldI2)
apply (rule subsetI)
apply (rule UnI1)
apply (rule UN_I)
apply (erule underS_I)
apply assumption
apply assumption
apply (rule equalityD1)
apply (drule notnotD)
apply (erule arg_cong)
done

```

```

lemma SucFbd_limit: [[x1 ∈ Field SucFbd & x2 ∈ Field SucFbd]]
⇒ ∃ y ∈ Field SucFbd. (x1 ≠ y ∧ (x1, y) ∈ SucFbd) ∧ (x2 ≠ y ∧ (x2, y) ∈ SucFbd)
apply (erule conjE)+
apply (rule rev_mp)
apply (rule Cinfinitelimit_finite)
apply (rule finite.insertI)
apply (rule finite.insertI)
apply (rule finite.emptyI)
apply (erule insert_subsetI)
apply (erule insert_subsetI)
apply (rule empty_subsetI)
apply (rule SucFbd_Cinfinitelimit)
apply (rule impI)
apply (erule bexE)
apply (rule beXI)

apply (rule conjI)

apply (erule bspec)
apply (rule insertI1)

apply (erule bspec)
apply (rule insertI2)
apply (rule insertI1)
apply assumption
done

```

```

lemma alg_min_alg: alg (min_alg1 s1 s2) (min_alg2 s1 s2) s1 s2
apply (tactic <rtac @{context} @<{thm alg_def} RS iffD2) 1>)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE conjE)+

apply (rule bexE)
apply (rule cardSuc_UNION_Cinfinitelimit)
apply (rule Cinfinitelimit_csum1)
apply (rule F1bd'_Cinfinitelimit)
apply (rule min_algs_mono1)
apply (erule subset_trans[OF _ equalityD1[OF min_alg1_def]])
apply (rule ordLeq_transitive)

```

apply (rule *F1set2_bd_incr*)
apply (rule *ordLeq_csum1*)
apply (rule *F1bd'_Card_order*)

apply (rule *bexE*)
apply (rule *cardSuc_UNION_Cinfinite*)
apply (rule *Cinfinite_csum1*)
apply (rule *F1bd'_Cinfinite*)
apply (rule *min_algs_mono2*)
apply (erule *subset_trans*[*OF equalityD1*[*OF min_alg2_def*]])
apply (rule *ordLeq_transitive*)
apply (rule *F1set3_bd_incr*)
apply (rule *ordLeq_csum1*)
apply (rule *F1bd'_Card_order*)

apply (rule *bexE*)
apply (rule *SucFbd_limit*)
apply (erule *conjI*)
apply *assumption*
apply (rule *subsetD*[*OF equalityD2*[*OF min_alg1_def*]])
apply (rule *UN_I*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply *assumption*
apply (rule *subsetD*)
apply (rule *equalityD2*)
apply (rule *min_algs1*)
apply *assumption*
apply (rule *UnI2*)
apply (rule *image_eqI*)
apply (rule *refl*)
apply (rule *CollectI*)
apply (erule *asm_rl*)
apply (erule *thin_rl*)
apply (erule *thin_rl*)
apply (erule *conjE*)+

apply (rule *conjI*)
apply *assumption*

apply (rule *conjI*)
apply (erule *subset_trans*)
apply (rule *subsetI*)
apply (rule *UN_I*)
apply (erule *underS_I*)
apply *assumption*
apply *assumption*

apply (erule *subset_trans*)
apply (erule *UN_upper*[*OF underS_I*])
apply *assumption*

apply (rule *ballI*)
apply (erule *CollectE conjE*)+

apply (rule *bexE*)

```

apply (rule cardSuc_UNION_Cinfinite)
  apply (rule Cinfinite_csum1)
  apply (rule F1bd'_Cinfinite)
  apply (rule min_algs_mono1)

apply (erule subset_trans[OF equalityD1[OF min_alg1_def]])
apply (rule ordLeq_transitive)
apply (rule F2set2_bd_incr)
apply (rule ordLeq_csum2)
apply (rule F2bd'_Card_order)

apply (rule bezE)
apply (rule cardSuc_UNION_Cinfinite)
  apply (rule Cinfinite_csum1)
  apply (rule F1bd'_Cinfinite)
  apply (rule min_algs_mono2)

apply (erule subset_trans[OF equalityD1[OF min_alg2_def]])
apply (rule ordLeq_transitive)
apply (rule F2set3_bd_incr)
apply (rule ordLeq_csum2)
apply (rule F2bd'_Card_order)

apply (rule bezE)
apply (rule SucFbd_limit)
apply (erule conjI)
apply assumption
apply (rule subsetD[OF equalityD2[OF min_alg2_def]])
apply (rule UN_I)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply assumption
apply (rule subsetD)
apply (rule equalityD2)
apply (rule min_algs2)
apply assumption
apply (rule UnI2)
apply (rule image_eqI)
apply (rule refl)
apply (rule CollectI)
apply (rule conjI)
apply assumption

apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule conjE)+
apply (rule conjI)
apply (erule subset_trans)
apply (rule UN_upper)
apply (erule underS_I)
apply assumption

apply (erule subset_trans)
apply (rule UN_upper)
apply (erule underS_I)
apply assumption
done

```

lemmas *SucFbd_ASucFbd* = *ordLess_ordLeq_trans*[*OF*
ordLess_ctwo_cexp
cexp_mono1[*OF ordLeq_csum2*[*OF Card_order_ctwo*]],
OF SucFbd_Card_order SucFbd_Card_order]

lemma *card_of_min_algs*:
fixes *s1* :: ('a, 'b, 'c) *F1* \Rightarrow 'b **and** *s2* :: ('a, 'b, 'c) *F2* \Rightarrow 'c
shows *i* \in *Field SucFbd* \longrightarrow
(|fst (*min_algs s1 s2 i*)| \leq_o (*ASucFbd* :: 'a *ASucFbd_type rel*) \wedge |snd (*min_algs s1 s2 i*)| \leq_o (*ASucFbd* :: 'a
ASucFbd_type rel))
apply (*rule well_order_induct_imp*[of _ %i. (|fst (*min_algs s1 s2 i*)| \leq_o *ASucFbd* \wedge |snd (*min_algs s1 s2 i*)|
 \leq_o *ASucFbd*), *OF wored_SucFbd*])
apply (*rule impI*)
apply (*rule conjI*)
apply (*rule ordIso_ordLeq_trans*)
apply (*rule card_of_ordIso_subst*)
apply (*erule min_algs1*)
apply (*rule Un_Cinfinite_bound*)

apply (*rule UNION_Cinfinite_bound*)

apply (*rule ordLess_imp_ordLeq*)
apply (*rule ordLess_transitive*)
apply (*rule card_of_underS*)
apply (*rule SucFbd_Card_order*)
apply *assumption*
apply (*rule SucFbd_ASucFbd*)

apply (*rule ballI*)
apply (*erule allE*)
apply (*drule mp*)
apply (*erule underS_E*)
apply (*drule mp*)
apply (*erule underS_Field*)
apply (*erule conjE*)
apply *assumption*

apply (*rule ASucFbd_Cinfinite*)

apply (*rule ordLeq_transitive*)
apply (*rule card_of_image*)
apply (*rule ordLeq_transitive*)
apply (*rule F1.in_bd*)
apply (*rule ordLeq_transitive*)
apply (*rule cexp_mono1*)
apply (*rule csum_mono1*)
apply (*rule csum_mono2*)
apply (*rule csum_cinfinite_bound*)
apply (*rule UNION_Cinfinite_bound*)

apply (*rule ordLess_imp_ordLeq*)
apply (*rule ordLess_transitive*)
apply (*rule card_of_underS*)
apply (*rule SucFbd_Card_order*)
apply *assumption*
apply (*rule SucFbd_ASucFbd*)

apply (*rule ballI*)
apply (*erule allE*)
apply (*drule mp*)
apply (*erule underS_E*)
apply (*drule mp*)

```

    apply (erule underS_Field)
    apply (erule conjE)+
    apply assumption

    apply (rule ASucFbd_Cinfinite)

    apply (rule UNION_Cinfinite_bound)

    apply (rule ordLess_imp_ordLeq)
    apply (rule ordLess_transitive)
    apply (rule card_of_underS)
    apply (rule SucFbd_Card_order)
    apply assumption
    apply (rule SucFbd_ASucFbd)

    apply (rule ballI)
    apply (erule allE)
    apply (drule mp)
    apply (erule underS_E)
    apply (drule mp)
    apply (erule underS_Field)
    apply (erule conjE)+
    apply assumption

    apply (rule ASucFbd_Cinfinite)

    apply (rule card_of_Card_order)
    apply (rule card_of_Card_order)
    apply (rule ASucFbd_Cinfinite)

    apply (rule F1bd'_Card_order)
    apply (rule ordIso_ordLeq_trans)
    apply (rule cexp_cong1)
    apply (rule ordIso_transitive)
    apply (rule csum_cong1)
    apply (rule ordIso_transitive)
    apply (tactic ⟨BNF_Tactics.mk_rotate_eq_tac @ {context}
      (rtac @ {context} @ {thm ordIso_refl} THEN'
        FIRST' [rtac @ {context} @ {thm card_of_Card_order},
          rtac @ {context} @ {thm Card_order_csum},
          rtac @ {context} @ {thm Card_order_cexp}]
        @ {thm ordIso_transitive} @ {thm csum_assoc} @ {thm csum_com} @ {thm csum_cong}
        [1,2] [2,1] 1)⟩)
    apply (rule csum_absorb1)
    apply (rule ASucFbd_Cinfinite)

    apply (rule ordLeq_transitive)
    apply (rule ordLeq_csum1)
    apply (tactic ⟨FIRST' [rtac @ {context} @ {thm Card_order_csum}, rtac @ {context} @ {thm card_of_Card_order}]
1)⟩)
    apply (rule ordLeq_cexp1)
    apply (rule SucFbd_Cnotzero)
    apply (rule Card_order_csum)
    apply (rule csum_absorb1)
    apply (rule ASucFbd_Cinfinite)
    apply (rule ctwo_ordLeq_Cinfinite)
    apply (rule ASucFbd_Cinfinite)
    apply (rule F1bd'_Card_order)
    apply (rule ordIso_imp_ordLeq)
    apply (rule cexp_cprod_ordLeq)

    apply (rule Card_order_csum)
    apply (rule SucFbd_Cinfinite)

```

```

apply (rule F1bd'_Cnotzero)
apply (rule ordLeq_transitive)
apply (rule ordLeq_csum1)
apply (rule F1bd'_Card_order)
apply (rule cardSuc_ordLeq)
apply (rule Card_order_csum)

apply (rule ASucFbd_Cinfinite)

apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (erule min_algs2)
apply (rule Un_Cinfinite_bound)

apply (rule UNION_Cinfinite_bound)

apply (rule ordLess_imp_ordLeq)
apply (rule ordLess_transitive)
apply (rule card_of_underS)
apply (rule SucFbd_Card_order)
apply assumption
apply (rule SucFbd_ASucFbd)

apply (rule ballI)
apply (erule allE)
apply (drule mp)
apply (erule underS_E)
apply (drule mp)
apply (erule underS_Field)
apply (erule conjE)+
apply assumption

apply (rule ASucFbd_Cinfinite)

apply (rule ordLeq_transitive)
apply (rule card_of_image)
apply (rule ordLeq_transitive)
apply (rule F2.in_bd)
apply (rule ordLeq_transitive)
apply (rule cexp_mono1)
apply (rule csum_mono1)
apply (rule csum_mono2)
apply (rule csum_cinfinite_bound)
apply (rule UNION_Cinfinite_bound)

apply (rule ordLess_imp_ordLeq)
apply (rule ordLess_transitive)
apply (rule card_of_underS)
apply (rule SucFbd_Card_order)
apply assumption
apply (rule SucFbd_ASucFbd)

apply (rule ballI)
apply (erule allE)
apply (drule mp)
apply (erule underS_E)
apply (drule mp)
apply (erule underS_Field)
apply (erule conjE)+
apply assumption

apply (rule ASucFbd_Cinfinite)

```

```

apply (rule UNION_Cinfinite_bound)

apply (rule ordLess_imp_ordLeq)
apply (rule ordLess_transitive)
apply (rule card_of_underS)
  apply (rule SucFbd_Card_order)
apply assumption
apply (rule SucFbd_ASucFbd)

apply (rule ballI)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption

apply (rule ASucFbd_Cinfinite)

apply (rule card_of_Card_order)
apply (rule card_of_Card_order)
apply (rule ASucFbd_Cinfinite)

apply (rule F2bd'_Card_order)
apply (rule ordIso_ordLeq_trans)
apply (rule cexp_cong1)

apply (rule ordIso_transitive)
apply (rule csum_cong1)
apply (rule ordIso_transitive)
apply (tactic ⟨BNF_Tactics.mk_rotate_eq_tac @ {context}
  (rtac @ {context} @ {thm ordIso_refl} THEN'
    FIRST' [rtac @ {context} @ {thm card_of_Card_order},
      rtac @ {context} @ {thm Card_order_csum},
      rtac @ {context} @ {thm Card_order_cexp}]
    @ {thm ordIso_transitive} @ {thm csum_assoc} @ {thm csum_com} @ {thm csum_cong}
    [1,2] [2,1] 1)
apply (rule csum_absorb1)
apply (rule ASucFbd_Cinfinite)

apply (rule ordLeq_transitive)
apply (rule ordLeq_csum1)
apply (tactic ⟨FIRST' [rtac @ {context} @ {thm Card_order_csum}, rtac @ {context} @ {thm card_of_Card_order}]
1)
apply (rule ordLeq_cexp1)
apply (rule SucFbd_Cnotzero)
apply (rule Card_order_csum)

apply (rule csum_absorb1)
apply (rule ASucFbd_Cinfinite)
apply (rule ctwo_ordLeq_Cinfinite)
apply (rule ASucFbd_Cinfinite)
apply (rule F2bd'_Card_order)
apply (rule ordIso_imp_ordLeq)
apply (rule cexp_cprod_ordLeq)
  apply (rule Card_order_csum)
apply (rule SucFbd_Cinfinite)
apply (rule F2bd'_Cnotzero)
apply (rule ordLeq_transitive)
apply (rule ordLeq_csum2)
apply (rule F2bd'_Card_order)
apply (rule cardSuc_ordLeq)

```



```

apply (rule Card_order_csum)

apply (rule ASucFbd_Cinfinite)
done

lemma card_of_min_alg1:
  fixes s1 :: ('a, 'b, 'c) F1  $\Rightarrow$  'b and s2 :: ('a, 'b, 'c) F2  $\Rightarrow$  'c
  shows |min_alg1 s1 s2|  $\leq o$  (ASucFbd :: 'a ASucFbd_type rel)
  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_ordIso_subst[OF min_alg1_def])
  apply (rule UNION_Cinfinite_bound)

  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_Field_ordIso)
  apply (rule SucFbd_Card_order)
  apply (rule ordLess_imp_ordLeq)
  apply (rule SucFbd_ASucFbd)

  apply (rule ballI)
  apply (drule rev_mp)
  apply (rule card_of_min_algs)
  apply (erule conjE)+
  apply assumption
  apply (rule ASucFbd_Cinfinite)
done

lemma card_of_min_alg2:
  fixes s1 :: ('a, 'b, 'c) F1  $\Rightarrow$  'b and s2 :: ('a, 'b, 'c) F2  $\Rightarrow$  'c
  shows |min_alg2 s1 s2|  $\leq o$  (ASucFbd :: 'a ASucFbd_type rel)
  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_ordIso_subst[OF min_alg2_def])
  apply (rule UNION_Cinfinite_bound)

  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_Field_ordIso)
  apply (rule SucFbd_Card_order)
  apply (rule ordLess_imp_ordLeq)
  apply (rule SucFbd_ASucFbd)

  apply (rule ballI)
  apply (drule rev_mp)
  apply (rule card_of_min_algs)
  apply (erule conjE)+
  apply assumption
  apply (rule ASucFbd_Cinfinite)
done

lemma least_min_algs: alg B1 B2 s1 s2  $\implies$ 
  i  $\in$  Field SucFbd  $\longrightarrow$ 
  fst (min_algs s1 s2 i)  $\subseteq$  B1  $\wedge$  snd (min_algs s1 s2 i)  $\subseteq$  B2
  apply (rule well_order_induct_imp[of _ %i. (fst (min_algs s1 s2 i)  $\subseteq$  B1  $\wedge$  snd (min_algs s1 s2 i)  $\subseteq$  B2), OF
  worel_SucFbd])
  apply (rule impI)
  apply (rule conjI)
  apply (rule ord_eq_le_trans)
  apply (erule min_algs1)
  apply (rule Un_least)
  apply (rule UN_least)
  apply (erule allE)
  apply (drule mp)
  apply (erule underS_E)
  apply (drule mp)
  apply (erule underS_Field)

```

```

apply (erule conjE)+
apply assumption
apply (rule image_subsetI)
apply (erule CollectE conjE)+
apply (erule alg_F1set)

```

```

apply (erule subset_trans)
apply (rule UN_least)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption

```

```

apply (erule subset_trans)
apply (rule UN_least)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption

```

```

apply (rule ord_eq_le_trans)
apply (erule min_algs2)
apply (rule Un_least)
apply (rule UN_least)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption
apply (rule image_subsetI)
apply (erule CollectE conjE)+
apply (erule alg_F2set)

```

```

apply (erule subset_trans)
apply (rule UN_least)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption

```

```

apply (erule subset_trans)
apply (rule UN_least)
apply (erule allE)
apply (drule mp)
  apply (erule underS_E)
apply (drule mp)
  apply (erule underS_Field)
apply (erule conjE)+
apply assumption
done

```

lemma least_min_alg1: $\text{alg } B1 \ B2 \ s1 \ s2 \implies \text{min_alg1 } s1 \ s2 \subseteq B1$

```

apply (rule ord_eq_le_trans[OF min_alg1_def])
apply (rule UN_least)
apply (drule least_min_algs)
apply (drule mp)
  apply assumption
apply (erule conjE)+
apply assumption
done

```

```

lemma least_min_alg2: alg B1 B2 s1 s2  $\implies$  min_alg2 s1 s2  $\subseteq$  B2
apply (rule ord_eq_le_trans[OF min_alg2_def])
apply (rule UN_least)
apply (drule least_min_algs)
apply (drule mp)
  apply assumption
apply (erule conjE)+
apply assumption
done

```

```

lemma mor_incl_min_alg:
  alg B1 B2 s1 s2  $\implies$ 
  mor (min_alg1 s1 s2) (min_alg2 s1 s2) s1 s2 B1 B2 s1 s2 id id
apply (rule mor_incl)
apply (erule least_min_alg1)
apply (erule least_min_alg2)
done

```

1.5 Initiality

The following "happens" to be the type (for our particular construction) of the initial algebra carrier:

```

type-synonym 'a1 F1init_type = ('a1, 'a1 ASucFbd_type, 'a1 ASucFbd_type) F1  $\Rightarrow$  'a1 ASucFbd_type
type-synonym 'a1 F2init_type = ('a1, 'a1 ASucFbd_type, 'a1 ASucFbd_type) F2  $\Rightarrow$  'a1 ASucFbd_type

```

```

typedef 'a1 IIT =
  UNIV ::
  (('a1 ASucFbd_type set  $\times$  'a1 ASucFbd_type set)  $\times$  ('a1 F1init_type  $\times$  'a1 F2init_type)) set
by (rule exI) (rule UNIV_I)

```

1.6 Initial Algebras

```

abbreviation II :: 'a1 IIT set where
  II  $\equiv$  {Abs_IIT ((B1, B2), (s1, s2)) | B1 B2 s1 s2. alg B1 B2 s1 s2}

```

```

definition str_init1 where
  str_init1 (dummy :: 'a1)
  (y::('a1, 'a1 IIT  $\Rightarrow$  'a1 ASucFbd_type, 'a1 IIT  $\Rightarrow$  'a1 ASucFbd_type) F1)
  (i :: 'a1 IIT) =
  fst (snd (Rep_IIT i))
  (F1map id ( $\lambda$ f :: 'a1 IIT  $\Rightarrow$  'a1 ASucFbd_type. f i) ( $\lambda$ f. f i) y)

```

```

definition str_init2 where
  str_init2 (dummy :: 'a1) y (i :: 'a1 IIT) =
  snd (snd (Rep_IIT i)) (F2map id ( $\lambda$ f. f i) ( $\lambda$ f. f i) y)

```

```

abbreviation car_init1 where
  car_init1 dummy  $\equiv$  min_alg1 (str_init1 dummy) (str_init2 dummy)

```

```

abbreviation car_init2 where
  car_init2 dummy  $\equiv$  min_alg2 (str_init1 dummy) (str_init2 dummy)

```

```

lemma alg_select:
   $\forall i \in II.$  alg (fst (fst (Rep_IIT i))) (snd (fst (Rep_IIT i)))
  (fst (snd (Rep_IIT i))) (snd (snd (Rep_IIT i)))
apply (rule ballI)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @ {context} 1 >)
unfolding fst_conv snd_conv Abs_IIT_inverse[OF UNIV_I]

```

apply *assumption*
done

lemma *mor_select*:

```

[[i ∈ II;
  mor (fst (fst (Rep_IIT i))) (snd (fst (Rep_IIT i)))
    (fst (snd (Rep_IIT i))) (snd (snd (Rep_IIT i))) UNIV UNIV s1' s2' f g]] ==>
mor (car_init1 dummy) (car_init2 dummy) (str_init1 dummy) (str_init2 dummy) UNIV UNIV s1' s2' (f ∘ (λh.
h i)) (g ∘ (λh. h i))
apply (rule mor_cong)
  apply (rule sym)
  apply (rule o_id)
  apply (rule sym)
  apply (rule o_id)
apply (tactic <rtac @ {context} (Thm.permute_prem 0 1 @ {thm mor_comp}) 1 >)
apply (tactic <etac @ {context} (Thm.permute_prem 0 1 @ {thm mor_comp}) 1 >)
apply (tactic <rtac @ {context} (@ {thm mor_def} RS iffD2) 1 >)
apply (rule conjI)

  apply (rule conjI)
  apply (rule ballI)
  apply (erule bspec[rotated])
  apply (erule CollectE)
  apply assumption

  apply (rule ballI)
  apply (erule bspec[rotated])
  apply (erule CollectE)
  apply assumption

  apply (rule conjI)
  apply (rule ballI)
  apply (rule str_init1_def)

  apply (rule ballI)
  apply (rule str_init2_def)

apply (rule mor_incl_min_alg)

apply (erule thin_rl)+
apply (tactic <rtac @ {context} (@ {thm alg_def} RS iffD2) 1 >)
apply (rule conjI)
  apply (rule ballI)
  apply (erule CollectE conjE)+
  apply (rule CollectI)
  apply (rule ballI)
  apply (erule bspec[OF alg_select])
  apply (rule ssubst_mem[OF str_init1_def])
  apply (erule alg_F1set)

  apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(2))
  apply (rule subset_trans)
  apply (erule image_mono)
  apply (rule image_Collect_subsetI)
  apply (erule bspec)
  apply assumption

  apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(3))
  apply (rule subset_trans)
  apply (erule image_mono)
  apply (rule image_Collect_subsetI)

```

apply (*erule bspec*)
apply *assumption*

apply (*rule ballI*)
apply (*erule CollectE conjE*)
apply (*rule CollectI*)
apply (*rule ballI*)
apply (*frule bspec*[*OF alg_select*])
apply (*rule ssubst_mem*[*OF str_init2_def*])
apply (*erule alg_F2set*)

apply (*rule ord_eq_le_trans*)
apply (*rule F2.set_map*(2))
apply (*rule subset_trans*)
apply (*erule image_mono*)
apply (*rule image_Collect_subsetI*)
apply (*erule bspec*)
apply *assumption*

apply (*rule ord_eq_le_trans*)
apply (*rule F2.set_map*(3))
apply (*rule subset_trans*)
apply (*erule image_mono*)
apply (*rule image_Collect_subsetI*)
apply (*erule bspec*)
apply *assumption*
done

lemma *init_unique_mor*:

$\llbracket a1 \in \text{car_init1 dummy}; a2 \in \text{car_init2 dummy};$
 $\text{mor } (\text{car_init1 dummy}) (\text{car_init2 dummy}) (\text{str_init1 dummy}) (\text{str_init2 dummy}) B1 B2 s1 s2 f1 f2;$
 $\text{mor } (\text{car_init1 dummy}) (\text{car_init2 dummy}) (\text{str_init1 dummy}) (\text{str_init2 dummy}) B1 B2 s1 s2 g1 g2 \rrbracket \implies$
 $f1 a1 = g1 a1 \wedge f2 a2 = g2 a2$

apply (*rule conjI*)
apply (*erule prop_restrict*)
apply (*erule thin_rl*)
apply (*rule least_min_alg1*)
apply (*tactic <rtac @*{context} (*@*{thm alg_def} *RS iffD2*) *1*>)

apply (*rule conjI*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)
apply (*rule conjI*)
apply (*rule alg_F1set*[*OF alg_min_alg*])
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule trans*)
apply (*erule morE1*)
apply (*rule subsetD*)
apply (*rule F1in_mono23*)
apply (*rule Collect_restrict*)
apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply *assumption*
apply *assumption*

apply (*rule trans*)
apply (*rule arg_cong*[*OF F1.map_cong0*])
 apply (*rule refl*)
 apply (*erule prop_restrict*)
 apply *assumption*
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule sym*)
apply (*erule morE1*)
apply (*rule subsetD*)
 apply (*rule F1in_mono23*)
 apply (*rule Collect_restrict*)
 apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
 apply *assumption*
apply (*rule conjI*)
 apply *assumption*
apply *assumption*

apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F2set*[*OF alg_min_alg*])
 apply (*erule subset_trans*)
 apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule trans*)
apply (*erule morE2*)
apply (*rule subsetD*)
 apply (*rule F2in_mono23*)
 apply (*rule Collect_restrict*)
 apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
 apply *assumption*
apply (*rule conjI*)
 apply *assumption*
apply *assumption*

apply (*rule trans*)
apply (*rule arg_cong*[*OF F2.map_cong0*])
 apply (*rule refl*)
 apply (*erule prop_restrict*)
 apply *assumption*
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule sym*)
apply (*erule morE2*)
apply (*rule subsetD*)
 apply (*rule F2in_mono23*)
 apply (*rule Collect_restrict*)
 apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*

apply (*rule conjI*)
apply *assumption*
apply *assumption*

apply (*erule thin_rl*)
apply (*erule prop_restrict*)
apply (*rule least_min_alg2*)
apply (*tactic* <*rtac* @*{context}*> (@*{thm alg_def}* *RS iffD2*) 1 >)
apply (*rule conjI*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F1set*[*OF alg_min_alg*])
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule trans*)
apply (*erule morE1*)
apply (*rule subsetD*)
apply (*rule F1in_mono23*)
apply (*rule Collect_restrict*)
apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply *assumption*
apply *assumption*

apply (*rule trans*)
apply (*rule arg_cong*[*OF F1.map_cong0*])
apply (*rule refl*)
apply (*erule prop_restrict*)
apply *assumption*
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule sym*)
apply (*erule morE1*)
apply (*rule subsetD*)
apply (*rule F1in_mono23*)
apply (*rule Collect_restrict*)
apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply *assumption*
apply *assumption*

apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F2set*[*OF alg_min_alg*])
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule trans*)
apply (*erule morE2*)
apply (*rule subsetD*)
apply (*rule F2in_mono23*)
apply (*rule Collect_restrict*)
apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply *assumption*
apply *assumption*

apply (*rule trans*)
apply (*rule arg_cong[OF F2.map_cong0]*)
apply (*rule refl*)
apply (*erule prop_restrict*)
apply *assumption*
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule sym*)
apply (*erule morE2*)
apply (*rule subsetD*)
apply (*rule F2in_mono23*)
apply (*rule Collect_restrict*)
apply (*rule Collect_restrict*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply *assumption*
apply *assumption*
done

abbreviation *closed where*

closed dummy phi1 phi2 $\equiv ((\forall x \in F1in UNIV (car_init1 dummy) (car_init2 dummy)).$
 $(\forall z \in F1set2 x. phi1 z) \wedge (\forall z \in F1set3 x. phi2 z) \longrightarrow phi1 (str_init1 dummy x)) \wedge$
 $(\forall x \in F2in UNIV (car_init1 dummy) (car_init2 dummy)).$
 $(\forall z \in F2set2 x. phi1 z) \wedge (\forall z \in F2set3 x. phi2 z) \longrightarrow phi2 (str_init2 dummy x))$

lemma *init_induct: closed dummy phi1 phi2* \implies

$(\forall x \in car_init1 dummy. phi1 x) \wedge (\forall x \in car_init2 dummy. phi2 x)$

apply (*rule conjI*)
apply (*rule ballI*)
apply (*erule prop_restrict*)
apply (*rule least_min_alg1*)
apply (*tactic <rtac @ {context} (@ {thm alg_def} RS iffD2) 1 >*)

apply (*rule conjI*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F1set[OF alg_min_alg]*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule mp*)
apply (*erule bspec*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule conjI*)
apply (*rule ballI*)
apply (*erule prop_restrict*)
apply *assumption*
apply (*rule ballI*)
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F2set*[*OF alg_min_alg*])
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule mp*)
apply (*erule bspec*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply *assumption*
apply (*rule conjI*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)
apply (*erule subset_trans*)
apply (*rule Collect_restrict*)

apply (*rule conjI*)
apply (*rule ballI*)
apply (*erule prop_restrict*)
apply *assumption*
apply (*rule ballI*)
apply (*erule prop_restrict*)
apply *assumption*

apply (*rule ballI*)
apply (*erule prop_restrict*)
apply (*rule least_min_alg2*)
apply (*tactic* $\langle \text{rtac } @\{\text{context}\} (@\{\text{thm alg_def}\} \text{RS iffD2}) 1 \rangle$)

apply (*rule conjI*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*erule CollectE conjE*)+
apply (*rule conjI*)

apply (*rule alg_F1set*[*OF alg_min_alg*])

```

apply (erule subset_trans)
apply (rule Collect_restrict)
apply (erule subset_trans)
apply (rule Collect_restrict)

```

```

apply (rule mp)
apply (erule bspec)
apply (rule CollectI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (erule subset_trans)
apply (rule Collect_restrict)
apply (erule subset_trans)
apply (rule Collect_restrict)

```

```

apply (rule conjI)
apply (rule ballI)
apply (erule prop_restrict)
apply assumption
apply (rule ballI)
apply (erule prop_restrict)
apply assumption

```

```

apply (rule ballI)
apply (rule CollectI)
apply (erule CollectE conjE)+
apply (rule conjI)

```

```

apply (rule alg_F2set[OF alg_min_alg])
apply (erule subset_trans)
apply (rule Collect_restrict)
apply (erule subset_trans)
apply (rule Collect_restrict)

```

```

apply (rule mp)
apply (erule bspec)
apply (rule CollectI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (erule subset_trans)
apply (rule Collect_restrict)
apply (erule subset_trans)
apply (rule Collect_restrict)

```

```

apply (rule conjI)
apply (rule ballI)
apply (erule prop_restrict)
apply assumption
apply (rule ballI)
apply (erule prop_restrict)
apply assumption
done

```

1.7 The datatype

```

typedef (overloaded) 'a1 IF1 = car_init1 (undefined :: 'a1)
apply (rule iffD2)
apply (rule ex_in_conv)
apply (rule conjunct1)
apply (rule alg_not_empty)
apply (rule alg_min_alg)

```

```

done

typedef (overloaded) 'a1 IF2 = car_init2 (undefined :: 'a1)
  apply (rule iffD2)
  apply (rule ex_in_conv)
  apply (rule conjunct2)
  apply (rule alg_not_empty)
  apply (rule alg_min_alg)
done

definition ctor1 where ctor1 = Abs_IF1 o str_init1 undefined o F1map id Rep_IF1 Rep_IF2
definition ctor2 where ctor2 = Abs_IF2 o str_init2 undefined o F2map id Rep_IF1 Rep_IF2

lemma mor_Rep_IF:
  mor (UNIV :: 'a IF1 set) (UNIV :: 'a IF2 set) ctor1 ctor2
  (car_init1 undefined) (car_init2 undefined) (str_init1 undefined) (str_init2 undefined) Rep_IF1 Rep_IF2
  unfolding mor_def ctor1_def ctor2_def o_apply
  apply (rule conjI)
  apply (rule conjI)
  apply (rule ballI)
  apply (rule Rep_IF1)
  apply (rule ballI)
  apply (rule Rep_IF2)

  apply (rule conjI)
  apply (rule ballI)
  apply (rule Abs_IF1_inverse)
  apply (rule alg_F1set[OF alg_min_alg])
  apply (rule ord_eq_le_trans[OF F1.set_map(2)])
  apply (rule image_subsetI)
  apply (rule Rep_IF1)
  apply (rule ord_eq_le_trans[OF F1.set_map(3)])
  apply (rule image_subsetI)
  apply (rule Rep_IF2)

  apply (rule ballI)
  apply (rule Abs_IF2_inverse)
  apply (rule alg_F2set[OF alg_min_alg])
  apply (rule ord_eq_le_trans[OF F2.set_map(2)])
  apply (rule image_subsetI)
  apply (rule Rep_IF1)
  apply (rule ord_eq_le_trans[OF F2.set_map(3)])
  apply (rule image_subsetI)
  apply (rule Rep_IF2)
done

lemma mor_Abs_IF:
  mor (car_init1 undefined) (car_init2 undefined)
  (str_init1 undefined) (str_init2 undefined) UNIV UNIV ctor1 ctor2 Abs_IF1 Abs_IF2
  unfolding mor_def ctor1_def ctor2_def o_apply
  apply (rule conjI)
  apply (rule conjI)
  apply (rule ballI)
  apply (rule UNIV_I)
  apply (rule ballI)
  apply (rule UNIV_I)

  apply (rule conjI)
  apply (rule ballI)
  apply (erule CollectE conjE)+
  apply (rule sym[OF arg_cong[OF trans[OF F1map_comp_id F1map_congL]]])
  apply (rule ballI[OF trans[OF o_apply]])
  apply (erule Abs_IF1_inverse[OF subsetD])

```

```

apply assumption
apply (rule ballI[OF trans[OF o_apply]])
apply (erule Abs_IF2_inverse[OF subsetD])
apply assumption

apply (rule ballI)
apply (erule CollectE conjE)+
apply (rule sym[OF arg_cong[OF trans[OF F2map_comp_id F2map_congL]]])
apply (rule ballI[OF trans[OF o_apply]])
apply (erule Abs_IF1_inverse[OF subsetD])
apply assumption
apply (rule ballI[OF trans[OF o_apply]])
apply (erule Abs_IF2_inverse[OF subsetD])
apply assumption
done

```

lemma *copy*:

```

[[alg B1 B2 s1 s2; bij_betw f B1' B1; bij_betw g B2' B2]] ==>
  ∃ f' g'. alg B1' B2' f' g' ∧ mor B1' B2' f' g' B1 B2 s1 s2 f g
apply (rule exI)+
apply (rule conjI)
apply (tactic <rtac @{context} (@{thm alg_def} RS iffD2) 1>)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE conjE)+
apply (rule subsetD)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on[OF bij_betw_the_inv_into])
apply (rule imageI)
apply (erule alg_F1set)
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(2))
apply (rule subset_trans)
apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(3))
apply (rule subset_trans)
apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)

apply (rule ballI)
apply (erule CollectE conjE)+
apply (rule subsetD)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on[OF bij_betw_the_inv_into])
apply (rule imageI)
apply (erule alg_F2set)
apply (rule ord_eq_le_trans)
apply (rule F2.set_map(2))
apply (rule subset_trans)
apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)
apply (rule ord_eq_le_trans)
apply (rule F2.set_map(3))
apply (rule subset_trans)
apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)

```

```

apply (tactic <rtac @{{context}} (@{thm mor_def} RS iffD2) 1>)
apply (rule conjI)
apply (rule conjI)
  apply (erule bij_betwE)
apply (erule bij_betwE)

```

```

apply (rule conjI)
apply (rule ballI)
apply (erule CollectE conjE)+
apply (erule f_the_inv_into_f_bij_betw)
apply (erule alg_F1set)
  apply (rule ord_eq_le_trans)
    apply (rule F1.set_map(2))
apply (rule subset_trans)
  apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(3))
apply (rule subset_trans)
  apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)

```

```

apply (rule ballI)
apply (erule CollectE conjE)+
apply (erule f_the_inv_into_f_bij_betw)
apply (erule alg_F2set)
  apply (rule ord_eq_le_trans)
    apply (rule F2.set_map(2))
apply (rule subset_trans)
  apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)
apply (rule ord_eq_le_trans)
apply (rule F2.set_map(3))
apply (rule subset_trans)
  apply (erule image_mono)
apply (rule equalityD1)
apply (erule bij_betw_imp_surj_on)
done

```

lemma *init_ex_mor*:

```

 $\exists f g. mor\ UNIV\ UNIV\ ctor1\ ctor2\ UNIV\ UNIV\ s1\ s2\ f\ g$ 
apply (insert ex_bij_betw[OF card_of_min_alg1, of s1 s2]
  ex_bij_betw[OF card_of_min_alg2, of s1 s2])
apply (erule exE)+
apply (rule rev_mp)
  apply (rule copy[OF alg_min_alg])
    apply assumption
  apply assumption
apply (rule impI)
apply (erule exE conjE)+

```

```

apply (rule exI)+
apply (rule mor_comp)
  apply (rule mor_Rep_IF)
apply (rule mor_select)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply assumption

```

```

unfolding fst_conv snd_conv Abs_IIT_inverse[OF UNIV_I]
apply (erule mor_comp)
apply (rule mor_incl)
  apply (rule subset_UNIV)
apply (rule subset_UNIV)
done

```

Iteration

abbreviation fold where

```

fold s1 s2  $\equiv$  (SOME f. mor UNIV UNIV ctor1 ctor2 UNIV UNIV s1 s2 (fst f) (snd f))

```

definition fold1 where fold1 s1 s2 = fst (fold s1 s2)

definition fold2 where fold2 s1 s2 = snd (fold s1 s2)

lemma mor_fold:

```

mor UNIV UNIV ctor1 ctor2 UNIV UNIV s1 s2 (fold1 s1 s2) (fold2 s1 s2)

```

```

unfolding fold1_def fold2_def

```

```

apply (rule rev_mp)

```

```

  apply (rule init_ex_mor)

```

```

apply (rule impI)

```

```

apply (erule exE)

```

```

apply (erule exE)

```

```

apply (rule someI[of %f :: ('a IF1  $\Rightarrow$  'b)  $\times$  ('a IF2  $\Rightarrow$  'c)].

```

```

mor UNIV UNIV ctor1 ctor2 UNIV UNIV s1 s2 (fst f) (snd f)])

```

```

apply (erule mor_cong[OF fst_conv snd_conv])

```

```

done

```

ML \langle

```

val fold1 = rule_by_tactic @{context}

```

```

  (rtac @{context} CollectI 1 THEN BNF_Util.CONJ_WRAP (K (rtac @{context} @{thm subset_UNIV} 1)) (1
upto 3))

```

```

  @{thm morE1[OF mor_fold]}

```

```

val fold2 = rule_by_tactic @{context}

```

```

  (rtac @{context} CollectI 1 THEN BNF_Util.CONJ_WRAP (K (rtac @{context} @{thm subset_UNIV} 1)) (1
upto 3))

```

```

  @{thm morE2[OF mor_fold]}

```

\rangle

theorem fold1:

```

(fold1 s1 s2) (ctor1 x) = s1 (F1map id (fold1 s1 s2) (fold2 s1 s2) x)

```

```

apply (rule morE1)

```

```

  apply (rule mor_fold)

```

```

apply (rule CollectI)

```

```

apply (rule conjI)

```

```

  apply (rule subset_UNIV)

```

```

apply (rule conjI)

```

```

  apply (rule subset_UNIV)

```

```

apply (rule subset_UNIV)

```

```

done

```

theorem fold2:

```

(fold2 s1 s2) (ctor2 x) = s2 (F2map id (fold1 s1 s2) (fold2 s1 s2) x)

```

```

apply (rule morE2)

```

```

  apply (rule mor_fold)

```

```

apply (rule CollectI)

```

```

apply (rule conjI)

```

```

  apply (rule subset_UNIV)

```

```

apply (rule conjI)

```

```

  apply (rule subset_UNIV)

```

```

apply (rule subset_UNIV)

```

```

done

```

lemma *mor_UNIV*: *mor UNIV UNIV s1 s2 UNIV UNIV s1' s2' f g* \longleftrightarrow

f o s1 = s1' o F1map id f g \wedge *g o s2 = s2' o F2map id f g*

```

apply (rule iffI)
apply (rule conjI)
apply (rule ext)
apply (rule trans)
apply (rule o_apply)
apply (rule trans)
apply (erule morE1)
apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule subset_UNIV)
apply (rule sym[OF o_apply])

```

```

apply (rule ext)
apply (rule trans)
apply (rule o_apply)
apply (rule trans)
apply (erule morE2)
apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule conjI)
apply (rule subset_UNIV)
apply (rule subset_UNIV)
apply (rule sym[OF o_apply])

```

```

apply (tactic <rtac @ {context} (@ {thm mor_def} RS iffD2) 1 >)

```

```

apply (rule conjI)
apply (rule conjI)
apply (rule ballI)
apply (rule UNIV_I)
apply (rule ballI)
apply (rule UNIV_I)
apply (erule conjE)
apply (drule iffD1[OF fun_eq_iff])
apply (drule iffD1[OF fun_eq_iff])
apply (rule conjI)
apply (rule ballI)
apply (erule allE)+
apply (rule trans)
apply (erule trans[OF sym[OF o_apply]])
apply (rule o_apply)
apply (rule ballI)
apply (erule allE)+
apply (rule trans)
apply (erule trans[OF sym[OF o_apply]])
apply (rule o_apply)
done

```

lemma *fold_unique_mor*: *mor UNIV UNIV ctor1 ctor2 UNIV UNIV s1 s2 f g* \implies

f = fold1 s1 s2 \wedge *g = fold2 s1 s2*

```

apply (rule conjI)
apply (rule surj_fun_eq)
apply (rule type_definition.Abs_image[OF type_definition_IF1])
apply (rule ballI)
apply (rule conjunct1)
apply (rule init_unique_mor)
apply assumption
apply (rule Rep_IF2)

```

```

apply (rule mor_comp)
  apply (rule mor_Abs_IF)
apply assumption
apply (rule mor_comp)
  apply (rule mor_Abs_IF)
apply (rule mor_fold)

apply (rule surj_fun_eq)
  apply (rule type_definition.Abs_image[OF type_definition_IF2])
apply (rule ballI)
apply (rule conjunct2)
apply (rule init_unique_mor)
  apply (rule Rep_IF1)
  apply assumption
apply (rule mor_comp)
  apply (rule mor_Abs_IF)
apply assumption
apply (rule mor_comp)
  apply (rule mor_Abs_IF)
apply (rule mor_fold)
done

lemmas fold_unique = fold_unique_mor[OF iffD2[OF mor_UNIV], OF conjI]

lemmas fold1_ctor = sym[OF conjunct1[OF fold_unique_mor[OF mor_incl[OF subset_UNIV subset_UNIV]]]]
lemmas fold2_ctor = sym[OF conjunct2[OF fold_unique_mor[OF mor_incl[OF subset_UNIV subset_UNIV]]]]

Case distinction

lemmas ctor1_o_fold1 =
  trans[OF conjunct1[OF fold_unique_mor[OF mor_comp[OF mor_fold mor_str]]] fold1_ctor]
lemmas ctor2_o_fold2 =
  trans[OF conjunct2[OF fold_unique_mor[OF mor_comp[OF mor_fold mor_str]]] fold2_ctor]

definition dtor1 = fold1 (F1map id ctor1 ctor2) (F2map id ctor1 ctor2)
definition dtor2 = fold2 (F1map id ctor1 ctor2) (F2map id ctor1 ctor2)

ML <Local_Defs.fold @ {context} @ {thms dtor1_def} @ {thm ctor1_o_fold1}>
ML <Local_Defs.fold @ {context} @ {thms dtor2_def} @ {thm ctor2_o_fold2}>

lemma ctor1_o_dtor1: ctor1 o dtor1 = id
  unfolding dtor1_def
  apply (rule ctor1_o_fold1)
done

lemma ctor2_o_dtor2: ctor2 o dtor2 = id
  unfolding dtor2_def
  apply (rule ctor2_o_fold2)
done

lemma dtor1_o_ctor1: dtor1 o ctor1 = id
  apply (rule ext)
  apply (rule trans[OF o_apply])
  apply (rule trans[OF fun_cong[OF dtor1_def]])
  apply (rule trans[OF fold1])
  apply (rule trans[OF F1map_comp_id])
  apply (rule trans[OF F1map_congL])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF ctor1_o_fold1] id_apply])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF ctor2_o_fold2] id_apply])
  apply (rule sym[OF id_apply])
done

```



```

lemma dtor2_o_ctor2: dtor2 o ctor2 = id
  apply (rule ext)
  apply (rule trans[OF o_apply])
  apply (rule trans[OF fun_cong[OF dtor2_def]])
  apply (rule trans[OF fold2])
  apply (rule trans[OF F2map_comp_id])
  apply (rule trans[OF F2map_congL])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF ctor1_o_fold1] id_apply])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF ctor2_o_fold2] id_apply])
  apply (rule sym[OF id_apply])
done

```

```

lemmas dtor1_ctor1 = pointfree_idE[OF dtor1_o_ctor1]
lemmas dtor2_ctor2 = pointfree_idE[OF dtor2_o_ctor2]
lemmas ctor1_dtor1 = pointfree_idE[OF ctor1_o_dtor1]
lemmas ctor2_dtor2 = pointfree_idE[OF ctor2_o_dtor2]

```

```

lemmas bij_dtor1 = o_bij[OF ctor1_o_dtor1 dtor1_o_ctor1]
lemmas inj_dtor1 = bij_is_inj[OF bij_dtor1]
lemmas surj_dtor1 = bij_is_surj[OF bij_dtor1]
lemmas dtor1_nchotomy = surjD[OF surj_dtor1]
lemmas dtor1_diff = inj_eq[OF inj_dtor1]
lemmas dtor1_cases = exE[OF dtor1_nchotomy]
lemmas bij_dtor2 = o_bij[OF ctor2_o_dtor2 dtor2_o_ctor2]
lemmas inj_dtor2 = bij_is_inj[OF bij_dtor2]
lemmas surj_dtor2 = bij_is_surj[OF bij_dtor2]
lemmas dtor2_nchotomy = surjD[OF surj_dtor2]
lemmas dtor2_diff = inj_eq[OF inj_dtor2]
lemmas dtor2_cases = exE[OF dtor2_nchotomy]

```

```

lemmas bij_ctor1 = o_bij[OF dtor1_o_ctor1 ctor1_o_dtor1]
lemmas inj_ctor1 = bij_is_inj[OF bij_ctor1]
lemmas surj_ctor1 = bij_is_surj[OF bij_ctor1]
lemmas ctor1_nchotomy = surjD[OF surj_ctor1]
lemmas ctor1_diff = inj_eq[OF inj_ctor1]
lemmas ctor1_cases = exE[OF ctor1_nchotomy]
lemmas bij_ctor2 = o_bij[OF dtor2_o_ctor2 ctor2_o_dtor2]
lemmas inj_ctor2 = bij_is_inj[OF bij_ctor2]
lemmas surj_ctor2 = bij_is_surj[OF bij_ctor2]
lemmas ctor2_nchotomy = surjD[OF surj_ctor2]
lemmas ctor2_diff = inj_eq[OF inj_ctor2]
lemmas ctor2_cases = exE[OF ctor2_nchotomy]

```

Primitive recursion

definition *rec1* **where**

```

rec1 s1 s2 = snd o fold1 (<ctor1 o F1map id fst fst, s1>) (<ctor2 o F2map id fst fst, s2>)

```

definition *rec2* **where**

```

rec2 s1 s2 = snd o fold2 (<ctor1 o F1map id fst fst, s1>) (<ctor2 o F2map id fst fst, s2>)

```

```

lemma fold1_o_ctor1: fold1 s1 s2 o ctor1 = s1 o F1map id (fold1 s1 s2) (fold2 s1 s2)
  by (tactic <rtac @<context> (BNF_Tactics.mk_pointfree2 @<context> @<thm fold1>) 1>)
lemma fold2_o_ctor2: fold2 s1 s2 o ctor2 = s2 o F2map id (fold1 s1 s2) (fold2 s1 s2)
  by (tactic <rtac @<context> (BNF_Tactics.mk_pointfree2 @<context> @<thm fold2>) 1>)

```

lemmas *fst_rec1_pair* =

```

trans[OF conjunct1[OF fold_unique[OF
  trans[OF o_assoc[symmetric] trans[OF arg_cong2[of _ _ _ _ (o), OF refl
    trans[OF fold1_o_ctor1 convol_o]]], OF trans[OF fst_convol]]]
  trans[OF o_assoc[symmetric] trans[OF arg_cong2[of _ _ _ _ (o), OF refl
    trans[OF fold2_o_ctor2 convol_o]]], OF trans[OF fst_convol]]]]]
fold1_ctor, unfolded F1.map_comp0[of id, unfolded id_o] F2.map_comp0[of id, unfolded id_o] o_assoc,

```

OF refl refl
lemmas *fst_rec2_pair* =
trans[*OF conjunct2*[*OF fold_unique*[*OF*
trans[*OF o_assoc*[*symmetric*] *trans*[*OF arg_cong2*[*of _ _ _ _ (o)*, *OF refl*
trans[*OF fold1_o_ctor1 convol_o*]], *OF trans*[*OF fst_convol*]]
trans[*OF o_assoc*[*symmetric*] *trans*[*OF arg_cong2*[*of _ _ _ _ (o)*, *OF refl*
trans[*OF fold2_o_ctor2 convol_o*]], *OF trans*[*OF fst_convol*]]]]
fold2_ctor, *unfolded F1.map_comp0*[*of id*, *unfolded id_o*] *F2.map_comp0*[*of id*, *unfolded id_o*] *o_assoc*,
OF refl refl]

theorem *rec1*: *rec1 s1 s2 (ctor1 x) = s1 (F1map id (<id, rec1 s1 s2>) (<id, rec2 s1 s2>) x)*
unfolding *rec1_def rec2_def o_apply fold1 snd_convol'*
convol_expand_snd[*OF fst_rec1_pair*] *convol_expand_snd*[*OF fst_rec2_pair*] ..

theorem *rec2*: *rec2 s1 s2 (ctor2 x) = s2 (F2map id (<id, rec1 s1 s2>) (<id, rec2 s1 s2>) x)*
unfolding *rec1_def rec2_def o_apply fold2 snd_convol'*
convol_expand_snd[*OF fst_rec1_pair*] *convol_expand_snd*[*OF fst_rec2_pair*] ..

lemma *rec_unique*:

f o ctor1 = s1 o F1map id <id, f> <id, g> \implies

g o ctor2 = s2 o F2map id <id, f> <id, g> $\implies f = rec1 s1 s2 \wedge g = rec2 s1 s2$

unfolding *rec1_def rec2_def convol_expand_snd'*[*OF fst_rec1_pair*] *convol_expand_snd'*[*OF fst_rec2_pair*]

apply (*rule fold_unique*)

apply (*unfold convol_o id_o o_id F1.map_comp0*[*symmetric*] *F2.map_comp0*[*symmetric*]

F1.map_id0 F2.map_id0 o_assoc[*symmetric*] *fst_convol*)

apply (*erule arg_cong2*[*of _ _ _ _ BNF_Def.convol, OF refl*])

apply (*erule arg_cong2*[*of _ _ _ _ BNF_Def.convol, OF refl*])

done

Induction

theorem *ctor_induct*:

$\llbracket \bigwedge x. (\bigwedge a. a \in F1set2\ x \implies \text{phi1}\ a) \implies (\bigwedge a. a \in F1set3\ x \implies \text{phi2}\ a) \implies \text{phi1}\ (\text{ctor1}\ x);$
 $\bigwedge x. (\bigwedge a. a \in F2set2\ x \implies \text{phi1}\ a) \implies (\bigwedge a. a \in F2set3\ x \implies \text{phi2}\ a) \implies \text{phi2}\ (\text{ctor2}\ x) \rrbracket \implies$
 $\text{phi1}\ a \wedge \text{phi2}\ b$

apply (*rule mp*)

apply (*rule impI*)

apply (*erule conjE*)

apply (*rule conjI*)

apply (*rule iffD1*[*OF arg_cong*[*OF Rep_IF1_inverse*]])

apply (*erule bspec*[*OF _ Rep_IF1*])

apply (*rule iffD1*[*OF arg_cong*[*OF Rep_IF2_inverse*]])

apply (*erule bspec*[*OF _ Rep_IF2*])

apply (*rule init_induct*)

apply (*rule conjI*)

apply (*drule asm_rl*)

apply (*erule thin_rl*)

apply (*rule ballI*)

apply (*rule impI*)

apply (*rule iffD2*[*OF arg_cong*[*OF morE1*[*OF mor_Abs_IF*]]])

apply *assumption*

apply (*erule CollectE conjE*)+

apply (*drule meta_spec*)

apply (*drule meta_mp*)

apply (*rule iffD1*[*OF arg_cong*[*OF Rep_IF1_inverse*]])

apply (*erule bspec*)

apply (*drule rev_subsetD*)

apply (*rule equalityD1*)

apply (*rule F1.set_map*(2))

apply (*erule imageE*)

apply (*tactic <hyp_subst_tac @{context} 1>*)

```

apply (rule ssubst_mem[OF Abs_IF1_inverse])
apply (erule subsetD)
apply assumption
apply assumption

```

```

apply (drule meta_mp)
apply (rule iffD1[OF arg_cong[OF Rep_IF2_inverse]])
apply (erule bspec)
apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule F1.set_map(3))
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @{context} 1⟩)
apply (rule ssubst_mem[OF Abs_IF2_inverse])
apply (erule subsetD)
apply assumption
apply assumption

```

```

apply assumption

```

```

apply (erule thin_rl)
apply (drule asm_rl)
apply (rule ballI)
apply (rule impI)
apply (rule iffD2[OF arg_cong[OF morE2[OF mor_Abs_IF]])]
apply assumption
apply (erule CollectE conjE)+
apply (drule meta_spec)
apply (drule meta_mp)
apply (rule iffD1[OF arg_cong[OF Rep_IF1_inverse]])
apply (erule bspec)
apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule F2.set_map(2))
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @{context} 1⟩)
apply (rule ssubst_mem[OF Abs_IF1_inverse])
apply (erule subsetD)
apply assumption
apply assumption

```

```

apply (drule meta_mp)
apply (rule iffD1[OF arg_cong[OF Rep_IF2_inverse]])
apply (erule bspec)
apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule F2.set_map(3))
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @{context} 1⟩)
apply (rule ssubst_mem[OF Abs_IF2_inverse])
apply (erule subsetD)
apply assumption
apply assumption

```

```

apply assumption
done

```

theorem *ctor_induct2*:

$$\begin{aligned}
& \llbracket \bigwedge x y. (\bigwedge a b. a \in F1set2\ x \implies b \in F1set2\ y \implies \mathit{phi1}\ a\ b) \implies \\
& \quad (\bigwedge a b. a \in F1set3\ x \implies b \in F1set3\ y \implies \mathit{phi2}\ a\ b) \implies \mathit{phi1}\ (\mathit{ctor1}\ x)\ (\mathit{ctor1}\ y); \\
& \bigwedge x y. (\bigwedge a b. a \in F2set2\ x \implies b \in F2set2\ y \implies \mathit{phi1}\ a\ b) \implies \\
& \quad (\bigwedge a b. a \in F2set3\ x \implies b \in F2set3\ y \implies \mathit{phi2}\ a\ b) \implies \mathit{phi2}\ (\mathit{ctor2}\ x)\ (\mathit{ctor2}\ y) \rrbracket \implies \\
& \mathit{phi1}\ a1\ b1 \wedge \mathit{phi2}\ a2\ b2
\end{aligned}$$

```

apply (rule rev_mp)
apply (rule ctor_induct[of %a1. (∀ x. phi1 a1 x) %a2. (∀ y. phi2 a2 y) a1 a2])
apply (rule allI[OF conjunct1[OF ctor_induct[OF asm_rl TrueI]])
apply (drule meta_spec2)
apply (erule thin_rl)
apply (tactic ⟨(dtac @{context} @{thm meta_mp} THEN_ALL_NEW Goal.norm_hhf_tac @{context}) 1⟩)
apply (drule meta_spec)+
apply (erule meta_mp[OF spec])
apply assumption
apply (drule meta_mp)
apply (drule meta_spec)+
apply (erule meta_mp[OF spec])
apply assumption
apply assumption

apply (rule allI[OF conjunct2[OF ctor_induct[OF TrueI asm_rl]])
apply (erule thin_rl)
apply (drule meta_spec2)
apply (drule meta_mp)
apply (drule meta_spec)+
apply (erule meta_mp[OF spec])
apply assumption
apply (erule meta_mp)
apply (drule meta_spec)+
apply (erule meta_mp[OF spec])
apply assumption

apply (rule impI)
apply (erule conjE allE)+
apply (rule conjI)
apply assumption
apply assumption
done

```

1.8 The Result as an BNF

The map operator

abbreviation *IF1map* **where** $IF1map\ f \equiv fold1\ (ctor1\ o\ (F1map\ f\ id\ id))\ (ctor2\ o\ (F2map\ f\ id\ id))$

abbreviation *IF2map* **where** $IF2map\ f \equiv fold2\ (ctor1\ o\ (F1map\ f\ id\ id))\ (ctor2\ o\ (F2map\ f\ id\ id))$

theorem *IF1map*:

$(IF1map\ f)\ o\ ctor1 = ctor1\ o\ (F1map\ f\ (IF1map\ f)\ (IF2map\ f))$

```

apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF fold1])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF F1map_comp_id]])
apply (rule trans[OF arg_cong[OF F1.map_cong0]])
apply (rule refl)
apply (rule trans[OF o_apply])
apply (rule id_apply)
apply (rule trans[OF o_apply])
apply (rule id_apply)
apply (rule sym[OF o_apply])
done

```

theorem *IF2map*:

$(IF2map\ f)\ o\ ctor2 = ctor2\ o\ (F2map\ f\ (IF1map\ f)\ (IF2map\ f))$

```

apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF fold2])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF F2map_comp_id]])

```

```

apply (rule trans[OF arg_cong[OF F2.map_cong0]])
  apply (rule refl)
  apply (rule trans[OF o_apply])
  apply (rule id_apply)
  apply (rule trans[OF o_apply])
  apply (rule id_apply)
apply (rule sym[OF o_apply])
done

```

```

lemmas IF1map_simps = o_eq_dest[OF IF1map]
lemmas IF2map_simps = o_eq_dest[OF IF2map]

```

lemma *IFmap_unique*:

```

[[u o ctor1 = ctor1 o F1map f u v; v o ctor2 = ctor2 o F2map f u v]] ==>
  u = IF1map f ^& v = IF2map f
apply (rule fold_unique)
unfolding o_assoc[symmetric] F1.map_comp0[symmetric] F2.map_comp0[symmetric] id_o o_id
apply assumption
apply assumption
done

```

theorem *IF1map_id*: $IF1map\ id = id$

```

apply (rule sym)
apply (rule conjunct1[OF IFmap_unique])
apply (rule trans[OF id_o])
apply (rule trans[OF sym[OF o_id]])
apply (rule arg_cong[OF sym[OF F1.map_id0]])
apply (rule trans[OF id_o])
apply (rule trans[OF sym[OF o_id]])
apply (rule arg_cong[OF sym[OF F2.map_id0]])
done

```

theorem *IF2map_id*: $IF2map\ id = id$

```

apply (rule sym)
apply (rule conjunct2[OF IFmap_unique])
apply (rule trans[OF id_o])
apply (rule trans[OF sym[OF o_id]])
apply (rule arg_cong[OF sym[OF F1.map_id0]])
apply (rule trans[OF id_o])
apply (rule trans[OF sym[OF o_id]])
apply (rule arg_cong[OF sym[OF F2.map_id0]])
done

```

theorem *IF1map_comp*: $IF1map\ (g\ o\ f) = IF1map\ g\ o\ IF1map\ f$

```

apply (rule sym)
apply (rule conjunct1[OF IFmap_unique])
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF IF1map_simps]])
apply (rule trans[OF IF1map_simps])
apply (rule trans[OF arg_cong[OF F1.map_comp]])
apply (rule sym[OF o_apply])
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF IF2map_simps]])
apply (rule trans[OF IF2map_simps])
apply (rule trans[OF arg_cong[OF F2.map_comp]])
apply (rule sym[OF o_apply])
done

```

theorem *IF2map_comp*: $IF2map\ (g\ o\ f) = IF2map\ g\ o\ IF2map\ f$

```

apply (rule sym)
apply (tactic ⟨rtac @ {context} (Thm.permute_premis 0 1 @ {thm conjunct2[OF IFmap_unique]}) 1⟩)
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF IF2map_simps]])
apply (rule trans[OF IF2map_simps])
apply (rule trans[OF arg_cong[OF F2.map_comp]])
apply (rule sym[OF o_apply])
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF o_apply])
apply (rule trans[OF arg_cong[OF IF1map_simps]])
apply (rule trans[OF IF1map_simps])
apply (rule trans[OF arg_cong[OF F1.map_comp]])
apply (rule sym[OF o_apply])
done

```

The bound

abbreviation *IFbd* **where** $IFbd \equiv bd_F1 + c\ bd_F2$

theorem *IFbd_card_order*: *card_order IFbd*

```

apply (rule card_order_csum)
apply (rule F1.bd_card_order)
apply (rule F2.bd_card_order)
done

```

lemma *IFbd_Cinfinite*: *Cinfinite IFbd*

```

apply (rule Cinfinite_csum1)
apply (rule F1.bd_Cinfinite)
done

```

lemmas *IFbd_cinfinite = conjunct1[OF IFbd_Cinfinite]*

The set operator

abbreviation *IF1col* **where** $IF1col \equiv (\lambda X. F1set1\ X \cup (\bigcup (F1set2\ X) \cup \bigcup (F1set3\ X)))$

abbreviation *IF2col* **where** $IF2col \equiv (\lambda X. F2set1\ X \cup (\bigcup (F2set2\ X) \cup \bigcup (F2set3\ X)))$

abbreviation *IF1set* **where** $IF1set \equiv fold1\ IF1col\ IF2col$

abbreviation *IF2set* **where** $IF2set \equiv fold2\ IF1col\ IF2col$

abbreviation *IF1in* **where** $IF1in\ A \equiv \{x. IF1set\ x \subseteq A\}$

abbreviation *IF2in* **where** $IF2in\ A \equiv \{x. IF2set\ x \subseteq A\}$

lemma *IF1set*: $IF1set\ o\ ctor1 = IF1col\ o\ (F1map\ id\ IF1set\ IF2set)$

```

apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF fold1])
apply (rule sym[OF o_apply])
done

```

lemma *IF2set*: $IF2set\ o\ ctor2 = IF2col\ o\ (F2map\ id\ IF1set\ IF2set)$

```

apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF fold2])
apply (rule sym[OF o_apply])
done

```

theorem *IF1set_simps*:

$IF1set\ (ctor1\ x) = F1set1\ x \cup ((\bigcup a \in F1set2\ x. IF1set\ a) \cup (\bigcup a \in F1set3\ x. IF2set\ a))$

```

apply (rule trans[OF o_eq_dest[OF IF1set]])

```

```

apply (rule arg_cong2[of _ _ _ (∪)])

```

```

apply (rule trans[OF F1.set_map(1) trans[OF fun_cong[OF image_id] id_apply]])

```

```

apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule arg_cong[OF F1.set_map(2)])
apply (rule arg_cong[OF F1.set_map(3)])
done

```

theorem *IF2set_simps*:

```

 $IF2set (ctor2\ x) = F2set1\ x \cup ((\bigcup a \in F2set2\ x.\ IF1set\ a) \cup (\bigcup a \in F2set3\ x.\ IF2set\ a))$ 
apply (rule trans[OF o_eq_dest[OF IF2set]])
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule trans[OF F2.set_map(1) trans[OF fun_cong[OF image_id] id_apply]])
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule arg_cong[OF F2.set_map(2)])
apply (rule arg_cong[OF F2.set_map(3)])
done

```

lemmas *F1set1_IF1set = xt1(3)[OF IF1set_simps Un_upper1]*

lemmas *F1set2_IF1set = subset_trans[OF UN_upper subset_trans[OF Un_upper1 xt1(3)[OF IF1set_simps Un_upper2]]]*

lemmas *F1set3_IF1set = subset_trans[OF UN_upper subset_trans[OF Un_upper2 xt1(3)[OF IF1set_simps Un_upper2]]]*

lemmas *F2set1_IF2set = xt1(3)[OF IF2set_simps Un_upper1]*

lemmas *F2set2_IF2set = subset_trans[OF UN_upper subset_trans[OF Un_upper1 xt1(3)[OF IF2set_simps Un_upper2]]]*

lemmas *F2set3_IF2set = subset_trans[OF UN_upper subset_trans[OF Un_upper2 xt1(3)[OF IF2set_simps Un_upper2]]]*

The BNF conditions for IF

lemma *IFset_natural*:

```

 $f' (IF1set\ x) = IF1set (IF1map\ f\ x) \wedge f' (IF2set\ y) = IF2set (IF2map\ f\ y)$ 
apply (rule ctor_induct[of _ _ x y])

```

```

apply (rule trans)
apply (rule image_cong)
apply (rule IF1set_simps)
apply (rule refl)
apply (rule sym)
apply (rule trans[OF arg_cong[of _ _ IF1set, OF IF1map_simps] trans[OF IF1set_simps]])

```

```

apply (rule sym)
apply (rule trans)
apply (rule image_UN)
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule sym)
apply (rule F1.set_map(1))

```

```

apply (rule trans)
apply (rule image_UN)
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule trans)
apply (rule image_UN)
apply (rule trans)
apply (rule SUP_cong)
apply (rule refl)
apply (tactic <Goal.assume_rule_tac @{context} 1>)
apply (rule sym)
apply (rule trans)
apply (rule SUP_cong)
apply (rule F1.set_map(2))
apply (rule refl)
apply (rule UN_simps(10))

```

```

apply (rule trans)
apply (rule image_UN)
apply (rule trans)
apply (rule SUP_cong)
apply (rule refl)

```

```

apply (tactic ⟨Goal.assume_rule_tac @{context} 1⟩)
apply (rule sym)
apply (rule trans)
apply (rule SUP_cong)
  apply (rule F1.set_map(3))
apply (rule refl)
apply (rule UN_simps(10))

```

```

apply (rule trans)
apply (rule image_cong)
  apply (rule IF2set_simps)
apply (rule refl)
apply (rule sym)
apply (rule trans[OF arg_cong[of _ _ IF2set, OF IF2map_simps] trans[OF IF2set_simps]])

```

```

apply (rule sym)
apply (rule trans)
apply (rule image_Un)
apply (rule arg_cong2[of _ _ _ _ (∪)])
apply (rule sym)
apply (rule F2.set_map(1))

```

```

apply (rule trans)
apply (rule image_Un)
apply (rule arg_cong2[of _ _ _ _ (∪)])

```

```

apply (rule trans)
apply (rule image_UN)
apply (rule trans)
apply (rule SUP_cong)
  apply (rule refl)
apply (tactic ⟨Goal.assume_rule_tac @{context} 1⟩)
apply (rule sym)
apply (rule trans)
apply (rule SUP_cong)
  apply (rule F2.set_map(2))
apply (rule refl)
apply (rule UN_simps(10))

```

```

apply (rule trans)
apply (rule image_UN)
apply (rule trans)
apply (rule SUP_cong)
  apply (rule refl)
apply (tactic ⟨Goal.assume_rule_tac @{context} 1⟩)
apply (rule sym)
apply (rule trans)
apply (rule SUP_cong)
  apply (rule F2.set_map(3))
apply (rule refl)
apply (rule UN_simps(10))
done

```

theorem *IF1set_natural*: $IF1set \circ (IF1map f) = image f \circ IF1set$

```

apply (rule ext)
apply (rule trans)
  apply (rule o_apply)
apply (rule sym)
apply (rule trans)
  apply (rule o_apply)
apply (rule conjunct1)
apply (rule IFset_natural)

```


done

theorem *IF2set_natural*: $IF2set\ o\ (IF2map\ f) = image\ f\ o\ IF2set$

```
apply (rule ext)
apply (rule trans)
  apply (rule o_apply)
apply (rule sym)
apply (rule trans)
  apply (rule o_apply)
apply (rule conjunct2)
apply (rule IFset_natural)
done
```

lemma *IFmap_cong*:

```
(( $\forall a \in IF1set\ x. f\ a = g\ a$ )  $\longrightarrow IF1map\ f\ x = IF1map\ g\ x$ )  $\wedge$ 
(( $\forall a \in IF2set\ y. f\ a = g\ a$ )  $\longrightarrow IF2map\ f\ y = IF2map\ g\ y$ )
apply (rule ctor_induct[of _ _ x y])
```

```
apply (rule impI)
apply (rule trans)
  apply (rule IF1map_simps)
apply (rule trans)
  apply (rule arg_cong[OF F1.map_cong0])
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (rule F1set1_IF1set)
  apply (rule mp)
  apply (tactic <Goal.assume_rule_tac @{context} 1>)
  apply (rule ballI)
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (erule F1set2_IF1set)
  apply (rule mp)
  apply (tactic <Goal.assume_rule_tac @{context} 1>)
  apply (rule ballI)
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (erule F1set3_IF1set)
  apply (rule sym)
  apply (rule IF1map_simps)
```

```
apply (rule impI)
apply (rule trans)
  apply (rule IF2map_simps)
apply (rule trans)
  apply (rule arg_cong[OF F2.map_cong0])
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (rule F2set1_IF2set)
  apply (rule mp)
  apply (tactic <Goal.assume_rule_tac @{context} 1>)
  apply (rule ballI)
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (erule F2set2_IF2set)
  apply (rule mp)
  apply (tactic <Goal.assume_rule_tac @{context} 1>)
  apply (rule ballI)
  apply (erule bspec)
  apply (erule rev_subsetD)
  apply (erule F2set3_IF2set)
  apply (rule sym)
  apply (rule IF2map_simps)
```

done

theorem *IF1map_cong*:

$(\bigwedge a. a \in IF1set\ x \implies f\ a = g\ a) \implies IF1map\ f\ x = IF1map\ g\ x$
apply (rule mp)
apply (rule conjunct1)
apply (rule IFmap_cong)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
done

theorem *IF2map_cong*:

$(\bigwedge a. a \in IF2set\ x \implies f\ a = g\ a) \implies IF2map\ f\ x = IF2map\ g\ x$
apply (rule mp)
apply (rule conjunct2)
apply (rule IFmap_cong)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
done

lemma *IFset_bd*:

$|IF1set\ (x :: 'a\ IF1)| \leq o\ IFbd \wedge |IF2set\ (y :: 'a\ IF2)| \leq o\ IFbd$
apply (rule ctor_induct[of _ _ x y])

apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule IF1set_simps)
apply (rule Un_Cinfinite_bound)
apply (rule F1set1_bd)
apply (rule Un_Cinfinite_bound)
apply (rule UNION_Cinfinite_bound)
apply (rule F1set2_bd)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
apply (rule IFbd_Cinfinite)
apply (rule UNION_Cinfinite_bound)
apply (rule F1set3_bd)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
apply (rule IFbd_Cinfinite)
apply (rule IFbd_Cinfinite)
apply (rule IFbd_Cinfinite)

apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule IF2set_simps)
apply (rule Un_Cinfinite_bound)
apply (rule F2set1_bd)
apply (rule Un_Cinfinite_bound)
apply (rule UNION_Cinfinite_bound)
apply (rule F2set2_bd)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
apply (rule IFbd_Cinfinite)
apply (rule UNION_Cinfinite_bound)
apply (rule F2set3_bd)
apply (rule ballI)
apply (tactic <Goal.assume_rule_tac @ {context} 1 >)
apply (rule IFbd_Cinfinite)
apply (rule IFbd_Cinfinite)
apply (rule IFbd_Cinfinite)
done

lemmas *IF1set_bd* = *conjunct1*[*OF IFset_bd*]

lemmas *IF2set_bd* = *conjunct2*[*OF IFset_bd*]

definition *IF1rel* **where**

IF1rel *R* =
(*BNF_Def.Grp* (*IF1in* (*Collect* (*case_prod* *R*))) (*IF1map fst*))^{^--1} *OO*
(*BNF_Def.Grp* (*IF1in* (*Collect* (*case_prod* *R*))) (*IF1map snd*))

definition *IF2rel* **where**

IF2rel *R* =
(*BNF_Def.Grp* (*IF2in* (*Collect* (*case_prod* *R*))) (*IF2map fst*))^{^--1} *OO*
(*BNF_Def.Grp* (*IF2in* (*Collect* (*case_prod* *R*))) (*IF2map snd*))

lemma *in_IF1rel*:

IF1rel *R* *x y* \longleftrightarrow ($\exists z. z \in$ *IF1in* (*Collect* (*case_prod* *R*)) \wedge *IF1map fst* *z* = *x* \wedge *IF1map snd* *z* = *y*)

unfolding *IF1rel_def* **by** (*rule predicate2_eqD*[*OF OO_Grp_alt*])

lemma *in_IF2rel*:

IF2rel *R* *x y* \longleftrightarrow ($\exists z. z \in$ *IF2in* (*Collect* (*case_prod* *R*)) \wedge *IF2map fst* *z* = *x* \wedge *IF2map snd* *z* = *y*)

unfolding *IF2rel_def* **by** (*rule predicate2_eqD*[*OF OO_Grp_alt*])

lemma *IF1rel_IF1rel*: *IF1rel* *R* (*ctor1* *a*) (*ctor1* *b*) \longleftrightarrow *F1rel* *R* (*IF1rel* *R*) (*IF2rel* *R*) *a* *b*

apply (*rule iffI*)

apply (*tactic* \langle *dtac* $\@$ {*context*} $\@$ {*thm in_IF1rel*[*THEN iffD1*]} \rangle *1*)⁺

apply (*erule exE conjE CollectE*)⁺

apply (*rule iffD2*)

apply (*rule F1.in_rel*)

apply (*rule exI*)

apply (*rule conjI*)

apply (*rule CollectI*)

apply (*rule conjI*)

apply (*rule ord_eq_le_trans*)

apply (*rule F1.set_map*(*1*))

apply (*rule ord_eq_le_trans*)

apply (*rule trans*[*OF fun_cong*[*OF image_id*] *id_apply*])

apply (*rule subset_trans*)

apply (*rule F1set1_IF1set*)

apply (*erule ord_eq_le_trans*[*OF arg_cong*[*OF ctor1_dtor1*]])

apply (*rule conjI*)

apply (*rule ord_eq_le_trans*)

apply (*rule F1.set_map*(*2*))

apply (*rule image_subsetI*)

apply (*rule CollectI*)

apply (*rule case_prodI*)

apply (*rule iffD2*)

apply (*rule in_IF1rel*)

apply (*rule exI*)

apply (*rule conjI*)

apply (*rule CollectI*)

apply (*erule subset_trans*[*OF F1set2_IF1set*])

apply (*erule ord_eq_le_trans*[*OF arg_cong*[*OF ctor1_dtor1*]])

apply (*rule conjI*)

apply (*rule refl*)

apply (*rule refl*)

apply (*rule ord_eq_le_trans*)

apply (*rule F1.set_map*(*3*))

apply (*rule image_subsetI*)

apply (*rule CollectI*)

apply (*rule case_prodI*)

apply (*rule iffD2*)

apply (*rule in_IF2rel*)

```

apply (rule exI)
apply (rule conjI)
  apply (rule CollectI)
  apply (rule subset_trans)
    apply (rule F1set3_IF1set)
    apply assumption
  apply (erule ord_eq_le_trans[OF arg_cong[OF ctor1_dtor1]])
apply (rule conjI)
  apply (rule refl)
apply (rule refl)
apply (rule conjI)

```

```

apply (rule trans)
  apply (rule F1.map_comp)
apply (rule trans)
  apply (rule F1.map_cong0)
    apply (rule fun_cong[OF o_id])
    apply (rule trans)
      apply (rule o_apply)
      apply (rule fst_conv)
apply (rule trans)
      apply (rule o_apply)
      apply (rule fst_conv)
apply (rule iffD1[OF ctor1_diff])
apply (rule trans)
  apply (rule sym)
  apply (rule IF1map_simps)
apply (erule trans[OF arg_cong[OF ctor1_dtor1]])

```

```

apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans)
    apply (rule o_apply)
    apply (rule snd_conv)
apply (rule trans)
    apply (rule o_apply)
    apply (rule snd_conv)
apply (rule iffD1[OF ctor1_diff])
apply (rule trans)
  apply (rule sym)
  apply (rule IF1map_simps)
apply (erule trans[OF arg_cong[OF ctor1_dtor1]])

```

```

apply (tactic <dtac @{context} (@{thm F1.in_rel[THEN iffD1]}) 1>)
apply (erule exE conjE CollectE)+
apply (rule iffD2)
  apply (rule in_IF1rel)
apply (rule exI)
apply (rule conjI)
  apply (rule CollectI)
  apply (rule ord_eq_le_trans)
  apply (rule IF1set_simps)
apply (rule Un_least)
  apply (rule ord_eq_le_trans)
    apply (rule box_equals[OF _ refl])
    apply (rule F1.set_map(1))
    apply (rule trans[OF fun_cong[OF image_id] id_apply])
  apply assumption
apply (rule Un_least)

```

```

apply (rule ord_eq_le_trans)
apply (rule SUP_cong[OF _ refl])
apply (rule F1.set_map(2))
apply (rule UN_least)
apply (drule rev_subsetD)
apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF1rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply (erule CollectD)

```

```

apply (rule ord_eq_le_trans)
apply (rule SUP_cong[OF _ refl])
apply (rule F1.set_map(3))
apply (rule UN_least)
apply (drule rev_subsetD)
apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply (erule CollectD)

```

```

apply (rule conjI)
apply (rule trans)
apply (rule IF1map_simps)
apply (rule iffD2[OF ctor1_diff])
apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
apply (rule fun_cong[OF o_id])
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF1rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply assumption

```

```

apply (rule trans)
apply (rule IF1map_simps)

```

```

apply (rule iffD2[OF ctor1_diff])
apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule fun_cong[OF o_id])
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
  apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF1rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply assumption
done

```

lemma IF2rel_F2rel: $IF2rel\ R\ (ctor2\ a)\ (ctor2\ b) \longleftrightarrow F2rel\ R\ (IF1rel\ R)\ (IF2rel\ R)\ a\ b$

```

apply (rule iffI)
apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)+
apply (erule exE conjE CollectE)+
apply (rule iffD2)
  apply (rule F2.in_rel)
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule conjI)
  apply (rule ord_eq_le_trans)
    apply (rule F2.set_map(1))
apply (rule ord_eq_le_trans)
  apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply (rule subset_trans)
apply (rule F2set1_IF2set)
apply (erule ord_eq_le_trans[OF arg_cong[OF ctor2_dtor2]])

apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(2))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule iffD2)
  apply (rule in_IF1rel)
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule subset_trans)
  apply (rule F2set2_IF2set)
  apply assumption
apply (erule ord_eq_le_trans[OF arg_cong[OF ctor2_dtor2]])
apply (rule conjI)

```

```

apply (rule refl)
apply (rule refl)

apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(3))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule iffD2)
  apply (rule in_IF2rel)
apply (rule exI)
apply (rule conjI)
  apply (rule CollectI)
  apply (rule subset_trans)
  apply (rule F2set3_IF2set)
  apply assumption
apply (erule ord_eq_le_trans[OF arg_cong[OF ctor2_dtor2]])
apply (rule conjI)
  apply (rule refl)
apply (rule refl)
apply (rule conjI)

apply (rule trans)
apply (rule F2.map_comp)
apply (rule trans)
  apply (rule F2.map_cong0)
    apply (rule fun_cong[OF o_id])
    apply (rule trans)
    apply (rule o_apply)
  apply (rule fst_conv)
apply (rule trans)
  apply (rule o_apply)
apply (rule fst_conv)
apply (rule iffD1[OF ctor2_diff])
apply (rule trans)
  apply (rule sym)
  apply (rule IF2map_simps)
apply (erule trans[OF arg_cong[OF ctor2_dtor2]])

apply (rule trans)
apply (rule F2.map_comp)
apply (rule trans)
  apply (rule F2.map_cong0)
    apply (rule fun_cong[OF o_id])
    apply (rule trans)
    apply (rule o_apply)
  apply (rule snd_conv)
apply (rule trans)
  apply (rule o_apply)
apply (rule snd_conv)
apply (rule iffD1[OF ctor2_diff])
apply (rule trans)
  apply (rule sym)
  apply (rule IF2map_simps)
apply (erule trans[OF arg_cong[OF ctor2_dtor2]])

apply (tactic <dtac @{context}> (@{thm F2.in_rel[THEN iffD1]}) 1)
apply (erule exE conjE CollectE)+
apply (rule iffD2)
  apply (rule in_IF2rel)
apply (rule exI)
apply (rule conjI)

```

```

apply (rule CollectI)
apply (rule ord_eq_le_trans)
apply (rule IF2set_simps)
apply (rule Un_least)
apply (rule ord_eq_le_trans)
  apply (rule trans)
    apply (rule trans)
      apply (rule arg_cong[OF dtor2_ctor2])
      apply (rule F2.set_map(1))
      apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption
apply (rule Un_least)
apply (rule ord_eq_le_trans)
  apply (rule trans[OF arg_cong[OF dtor2_ctor2]])
  apply (rule arg_cong[OF F2.set_map(2)])
apply (rule UN_least)
apply (drule rev_subsetD)
  apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (tactic <dtac @ {context} (@ {thm in_IF1rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+
apply (erule CollectD)

```

```

apply (rule ord_eq_le_trans)
apply (rule trans[OF arg_cong[OF dtor2_ctor2]])
apply (rule arg_cong[OF F2.set_map(3)])
apply (rule UN_least)
apply (drule rev_subsetD)
  apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @ {context} (@ {thm in_IF2rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule exE conjE)+
apply (erule CollectD)

```

```

apply (rule conjI)
apply (rule trans)
  apply (rule arg_cong[OF dtor2_ctor2])
apply (rule trans)
  apply (rule IF2map_simps)
apply (rule iffD2)
  apply (rule ctor2_diff)
apply (rule trans)
  apply (rule F2.map_comp)
apply (rule trans)
  apply (rule F2.map_cong0)
    apply (rule fun_cong[OF o_id])
    apply (rule trans[OF o_apply])
    apply (drule rev_subsetD)
  apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (tactic <dtac @ {context} (@ {thm in_IF1rel[THEN iffD1]}) 1>)
apply (drule someI_ex)
apply (erule conjE)+

```



```

  apply assumption
  apply (rule trans[OF o_apply])
  apply (drule rev_subsetD)
  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply assumption

```

```

apply (rule trans)
  apply (rule arg_cong[OF ctor2_ctor2])
apply (rule trans)
  apply (rule IF2map_simps)
apply (rule iffD2)
  apply (rule ctor2_diff)
apply (rule trans)
  apply (rule F2.map_comp)
apply (rule trans)
  apply (rule F2.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans[OF o_apply])
  apply (drule rev_subsetD)
  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (tactic <dtac @{context} (@{thm in_IF1rel[THEN iffD1]}) 1>)
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply (rule trans[OF o_apply])
  apply (drule rev_subsetD)
  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (tactic <dtac @{context} (@{thm in_IF2rel[THEN iffD1]}) 1>)
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply assumption
done

```

lemma *Irel_induct*:

```

assumes IH1:  $\forall x y. F1rel\ P1\ P2\ P3\ x\ y \longrightarrow P2\ (ctor1\ x)\ (ctor1\ y)$ 
  and IH2:  $\forall x y. F2rel\ P1\ P2\ P3\ x\ y \longrightarrow P3\ (ctor2\ x)\ (ctor2\ y)$ 
shows  $IF1rel\ P1 \leq P2 \wedge IF2rel\ P1 \leq P3$ 
unfolding le_fun_def le_bool_def all_simps(1,2)[symmetric]
apply (rule allI)+
apply (rule ctor_induct2)
apply (rule impI)
apply (drule iffD1[OF IF1rel_F1rel])
apply (rule mp[OF spec2[OF IH1]])
apply (erule F1.rel_mono_strong0)
  apply (rule ballI[OF ballI[OF imp_refl]])
apply (drule asm_rl)
apply (erule thin_rl)
apply (rule ballI[OF ballI])
apply assumption

```

```

apply (erule thin_rl)
apply (drule asm_rl)
apply (rule ballI[OF ballI])
apply assumption

```

```

apply (rule impI)
apply (drule iffD1[OF IF2rel_F2rel])
apply (rule mp[OF spec2[OF IH2]])
apply (erule F2.rel_mono_strong0)
  apply (rule ballI[OF ballI[OF imp_refl]])
  apply (drule asm_rl)
  apply (erule thin_rl)
  apply (rule ballI[OF ballI])
  apply assumption
apply (erule thin_rl)
apply (drule asm_rl)
apply (rule ballI[OF ballI])
apply assumption
done

```

lemma le_IFrel_Comp:

$((IF1rel\ R\ OO\ IF1rel\ S)\ x1\ y1 \longrightarrow IF1rel\ (R\ OO\ S)\ x1\ y1) \wedge$
 $((IF2rel\ R\ OO\ IF2rel\ S)\ x2\ y2 \longrightarrow IF2rel\ (R\ OO\ S)\ x2\ y2)$

```

apply (rule ctor_induct2[of _ _ x1 y1 x2 y2])
apply (rule impI)
apply (erule nchotomy_relcomppE[OF ctor1_nchotomy])
apply (drule iffD1[OF IF1rel_F1rel])
apply (drule iffD1[OF IF1rel_F1rel])
apply (rule iffD2[OF IF1rel_F1rel])
apply (rule F1.rel_mono_strong0)
  apply (rule iffD2[OF predicate2_eqD[OF F1.rel_compp]])
  apply (rule relcomppI)
  apply assumption
  apply assumption
  apply (rule ballI impI)+
  apply assumption
  apply (rule ballI)+
  apply assumption
  apply (rule ballI)+
  apply assumption

```

```

apply (rule impI)
apply (erule nchotomy_relcomppE[OF ctor2_nchotomy])
apply (drule iffD1[OF IF2rel_F2rel])
apply (drule iffD1[OF IF2rel_F2rel])
apply (rule iffD2[OF IF2rel_F2rel])
apply (rule F2.rel_mono_strong0)
  apply (rule iffD2[OF predicate2_eqD[OF F2.rel_compp]])
  apply (rule relcomppI)
  apply assumption
  apply assumption
  apply (rule ballI impI)+
  apply assumption
  apply (rule ballI)+
  apply assumption
  apply (rule ballI)+
  apply assumption
done

```

lemma le_IF1rel_Comp: $IF1rel\ R1\ OO\ IF1rel\ R2 \leq IF1rel\ (R1\ OO\ R2)$
by (rule predicate2I) (erule mp[OF conjunct1[OF le_IFrel_Comp]])

lemma le_IF2rel_Comp: $IF2rel\ R1\ OO\ IF2rel\ R2 \leq IF2rel\ (R1\ OO\ R2)$

```

by (rule predicate2I) (erule mp[OF conjunct2[OF le_IFrel_Comp]])

context includes lifting_syntax
begin

lemma fold_transfer:
  ((F1rel R S T ==> S) ==> (F2rel R S T ==> T) ==> IF1rel R ==> S) fold1 fold1 ^
  ((F1rel R S T ==> S) ==> (F2rel R S T ==> T) ==> IF2rel R ==> T) fold2 fold2
  unfolding rel_fun_def_butlast_all_conj_distrib[symmetric] imp_conjR[symmetric]
  unfolding rel_fun_iff_leq_vimage2p
  apply (rule allI impI)+
  apply (rule Irel_induct)
  apply (rule allI impI vimage2pI)+
  apply (unfold fold1 fold2) [1]
  apply (erule predicate2D_vimage2p)
  apply (rule rel_funD[OF rel_funD[OF rel_funD[OF rel_funD[OF F1.map_transfer]]]])
    apply (rule id_transfer)
    apply (rule vimage2p_rel_fun)
    apply (rule vimage2p_rel_fun)
  apply assumption

  apply (rule allI impI vimage2pI)+
  apply (unfold fold1 fold2) [1]
  apply (erule predicate2D_vimage2p)
  apply (rule rel_funD[OF rel_funD[OF rel_funD[OF rel_funD[OF F2.map_transfer]]]])
    apply (rule id_transfer)
    apply (rule vimage2p_rel_fun)
    apply (rule vimage2p_rel_fun)
  apply assumption
done

end

definition IF1wit x = ctor1 (wit2_F1 x (ctor2 wit_F2))
definition IF2wit = ctor2 wit_F2

lemma IF1wit: x ∈ IF1set (IF1wit y) ⇒ x = y
  unfolding IF1wit_def
  by (elim UnE F1.wit2[elim_format] F2.wit[elim_format] UN_E FalseE |
    rule refl | hypsubst | assumption | unfold IF1set_simps IF2set_simps)+

lemma IF2wit: x ∈ IF2set IF2wit ⇒ False
  unfolding IF2wit_def
  by (elim UnE F2.wit[elim_format] UN_E FalseE |
    rule refl | hypsubst | assumption | unfold IF2set_simps)+

ML <
  BNF_FP_Util.mk_xtor_co_iter_o_map_thms BNF_Util.Least_FP false 1 @ {thm fold_unique}
  @ {thms IF1map IF2map} (map (BNF_Tactics.mk_pointfree2 @ {context}) @ {thms fold1 fold2})
  @ {thms F1.map_comp0[symmetric] F2.map_comp0[symmetric]} @ {thms F1.map_cong0 F2.map_cong0}
  >

ML <
  BNF_FP_Util.mk_xtor_co_iter_o_map_thms BNF_Util.Least_FP true 1 @ {thm rec_unique}
  @ {thms IF1map IF2map} (map (BNF_Tactics.mk_pointfree2 @ {context}) @ {thms rec1 rec2})
  @ {thms F1.map_comp0[symmetric] F2.map_comp0[symmetric]} @ {thms F1.map_cong0 F2.map_cong0}
  >

bnf 'a IF1
  map: IF1map
  sets: IF1set
  bd: IFbd

```

```

wits: IF1wit
rel: IF1rel
  apply –
  apply (rule IF1map_id)
  apply (rule IF1map_comp)
  apply (erule IF1map_cong)
  apply (rule IF1set_natural)
  apply (rule IFbd_card_order)
  apply (rule IFbd_cinfinite)
  apply (rule IF1set_bd)
  apply (rule le_IF1rel_Comp)
  apply (rule IF1rel_def[unfolded OO_Grp_alt mem_Collect_eq])
  apply (erule IF1wit)
done

```

```

bnf 'a IF2
map: IF2map
sets: IF2set
bd: IFbd
wits: IF2wit
rel: IF2rel
  apply –
  apply (rule IF2map_id)
  apply (rule IF2map_comp)
  apply (erule IF2map_cong)
  apply (rule IF2set_natural)
  apply (rule IFbd_card_order)
  apply (rule IFbd_cinfinite)
  apply (rule IF2set_bd)
  apply (rule le_IF2rel_Comp)
  apply (rule IF2rel_def[unfolded OO_Grp_alt mem_Collect_eq])
  apply (erule IF2wit)
done

```

2 Greatest Fixpoint (a.k.a. Codatatype)

unbundle *cardinal_syntax*

```

'b1 = ('a, 'b1, 'b2) F1
'b2 = ('a, 'b1, 'b2) F2

```

To build a witness scenario, let us assume

```

('a, 'b1, 'b2) F1 = 'a * 'b1 + 'a * 'b2
('a, 'b1, 'b2) F2 = unit + 'b1 * 'b2

```

ML *<open Ctr_Sugar_Util>*

declare *[[bnf_internals]]*

```

bnf-axiomatization (F1set1: 'a, F1set2: 'b1, F1set3: 'b2) F1
[wits: 'a ⇒ 'b1 ⇒ ('a, 'b1, 'b2) F1 'a ⇒ 'b2 ⇒ ('a, 'b1, 'b2) F1]
for map: F1map rel: F1rel

```

```

bnf-axiomatization (F2set1: 'a, F2set2: 'b1, F2set3: 'b2) F2

```

```

[wits: ('a, 'b1, 'b2) F2]
for map: F2map rel: F2rel

```

lemma *F1rel_cong*: $\llbracket R1 = S1; R2 = S2; R3 = S3 \rrbracket \Longrightarrow F1rel\ R1\ R2\ R3 = F1rel\ S1\ S2\ S3$
by *hypsubst rule*

lemma *F2rel_cong*: $\llbracket R1 = S1; R2 = S2; R3 = S3 \rrbracket \Longrightarrow F2rel\ R1\ R2\ R3 = F2rel\ S1\ S2\ S3$
by *hypsubst rule*

abbreviation *F1in* :: 'a1 set ⇒ 'a2 set ⇒ 'a3 set ⇒ ((('a1, 'a2, 'a3) F1) set **where**

$F1in\ A1\ A2\ A3 \equiv \{x. F1set1\ x \subseteq A1 \wedge F1set2\ x \subseteq A2 \wedge F1set3\ x \subseteq A3\}$
abbreviation $F2in :: 'a1\ set \Rightarrow 'a2\ set \Rightarrow 'a3\ set \Rightarrow (('a1, 'a2, 'a3)\ F2)\ set\ where$
 $F2in\ A1\ A2\ A3 \equiv \{x. F2set1\ x \subseteq A1 \wedge F2set2\ x \subseteq A2 \wedge F2set3\ x \subseteq A3\}$

lemma $F1map_comp_id: F1map\ g1\ g2\ g3\ (F1map\ id\ f2\ f3\ x) = F1map\ g1\ (g2\ o\ f2)\ (g3\ o\ f3)\ x$
apply (rule trans)
apply (rule $F1.map_comp$)
unfolding o_id
apply (rule refl)
done

lemmas $F1in_mono23 = F1.in_mono[OF\ subset_refl]$
lemmas $F1in_mono23' = subsetD[OF\ F1in_mono23]$

lemma $F1map_congL: [\forall a \in F1set2\ x. f\ a = a; \forall a \in F1set3\ x. g\ a = a] \Longrightarrow$
 $F1map\ id\ f\ g\ x = x$
apply (rule trans)
apply (rule $F1.map_cong0$)
apply (rule refl)
apply (rule trans)
apply (erule bspec)
apply assumption
apply (rule sym)
apply (rule id_apply)
apply (rule trans)
apply (erule bspec)
apply assumption
apply (rule sym)
apply (rule id_apply)
apply (rule $F1.map_id$)
done

lemma $F2map_comp_id: F2map\ g1\ g2\ g3\ (F2map\ id\ f2\ f3\ x) = F2map\ g1\ (g2\ o\ f2)\ (g3\ o\ f3)\ x$
apply (rule trans)
apply (rule $F2.map_comp$)
unfolding o_id
apply (rule refl)
done

lemmas $F2in_mono23 = F2.in_mono[OF\ subset_refl]$
lemmas $F2in_mono23' = subsetD[OF\ F2in_mono23]$

lemma $F2map_congL: [\forall a \in F2set2\ x. f\ a = a; \forall a \in F2set3\ x. g\ a = a] \Longrightarrow$
 $F2map\ id\ f\ g\ x = x$
apply (rule trans)
apply (rule $F2.map_cong0$)
apply (rule refl)
apply (rule trans)
apply (erule bspec)
apply assumption
apply (rule sym)
apply (rule id_apply)
apply (rule trans)
apply (erule bspec)
apply assumption
apply (rule sym)
apply (rule id_apply)
apply (rule $F2.map_id$)
done

2.1 Coalgebra

definition $coalg\ where$
 $coalg\ B1\ B2\ s1\ s2 =$

$(\forall a \in B1. s1\ a \in F1in\ (UNIV\ ::\ 'a\ set)\ B1\ B2) \wedge (\forall a \in B2. s2\ a \in F2in\ (UNIV\ ::\ 'a\ set)\ B1\ B2))$

lemmas *coalg_F1in* = *bspec*[*OF conjunct1*[*OF iffD1*[*OF coalg_def*]]]
lemmas *coalg_F2in* = *bspec*[*OF conjunct2*[*OF iffD1*[*OF coalg_def*]]]

lemma *coalg_F1set2*:
 $\llbracket coalg\ B1\ B2\ s1\ s2; a \in B1 \rrbracket \implies F1set2\ (s1\ a) \subseteq B1$
apply (*tactic* <*dtac* @*{context}* @*{thm iffD1*[*OF coalg_def*]] 1>)
apply (*erule conjE*)
apply (*drule bspec*[*rotated*])
apply *assumption*
apply (*erule CollectE conjE*)
apply *assumption*
done

lemma *coalg_F1set3*:
 $\llbracket coalg\ B1\ B2\ s1\ s2; a \in B1 \rrbracket \implies F1set3\ (s1\ a) \subseteq B2$
apply (*tactic* <*dtac* @*{context}* @*{thm iffD1*[*OF coalg_def*]] 1>)
apply (*erule conjE*)
apply (*drule bspec*[*rotated*])
apply *assumption*
apply (*erule CollectE conjE*)
apply *assumption*
done

lemma *coalg_F2set2*:
 $\llbracket coalg\ B1\ B2\ s1\ s2; a \in B2 \rrbracket \implies F2set2\ (s2\ a) \subseteq B1$
apply (*tactic* <*dtac* @*{context}* @*{thm iffD1*[*OF coalg_def*]] 1>)
apply (*erule conjE*)
apply (*drule bspec*[*rotated*])
apply *assumption*
apply (*erule CollectE conjE*)
apply *assumption*
done

lemma *coalg_F2set3*:
 $\llbracket coalg\ B1\ B2\ s1\ s2; a \in B2 \rrbracket \implies F2set3\ (s2\ a) \subseteq B2$
apply (*tactic* <*dtac* @*{context}* @*{thm iffD1*[*OF coalg_def*]] 1>)
apply (*erule conjE*)
apply (*drule bspec*[*rotated*])
apply *assumption*
apply (*erule CollectE conjE*)
apply *assumption*
done

2.2 Type-coalgebra

abbreviation *tcoalg* *s1* *s2* $\equiv coalg\ UNIV\ UNIV\ s1\ s2$

lemma *tcoalg*: *tcoalg* *s1* *s2*
apply (*tactic* <*rtac* @*{context}* (@*{thm coalg_def}* *RS iffD2*) 1>)
apply (*rule conjI*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply (*rule subset_UNIV*)
apply (*rule conjI*)
apply (*rule subset_UNIV*)
apply (*rule subset_UNIV*)
apply (*rule ballI*)
apply (*rule CollectI*)
apply (*rule conjI*)
apply (*rule subset_UNIV*)
apply (*rule conjI*)

```

apply (rule subset_UNIV)
apply (rule subset_UNIV)
done

```

2.3 Morphism

definition *mor* where

```

mor B1 B2 s1 s2 B1' B2' s1' s2' f g =
  ((( $\forall a \in B1. f a \in B1'$ )  $\wedge$  ( $\forall a \in B2. g a \in B2'$ ))  $\wedge$ 
  (( $\forall z \in B1. F1map (id :: 'a \Rightarrow 'a) f g (s1 z) = s1' (f z)$ )  $\wedge$ 
  ( $\forall z \in B2. F2map (id :: 'a \Rightarrow 'a) f g (s2 z) = s2' (g z)$ )))

```

lemma *mor_image1*: $mor\ B1\ B2\ s1\ s2\ B1'\ B2'\ s1'\ s2'\ f\ g \implies f\ 'B1 \subseteq B1'$

```

apply (tactic <dtac @ {context} @ {thm iffD1[OF mor_def]} 1 >)
apply (erule conjE)+
apply (rule image_subsetI)
apply (erule bspec)
apply assumption
done

```

lemma *mor_image2*: $mor\ B1\ B2\ s1\ s2\ B1'\ B2'\ s1'\ s2'\ f\ g \implies g\ 'B2 \subseteq B2'$

```

apply (tactic <dtac @ {context} @ {thm iffD1[OF mor_def]} 1 >)
apply (erule conjE)+
apply (rule image_subsetI)
apply (erule bspec)
apply assumption
done

```

lemmas *mor_image1'* = subsetD[OF *mor_image1 imageI*]

lemmas *mor_image2'* = subsetD[OF *mor_image2 imageI*]

lemma *morE1*: $\llbracket mor\ B1\ B2\ s1\ s2\ B1'\ B2'\ s1'\ s2'\ f\ g; z \in B1 \rrbracket$

```

 $\implies F1map\ id\ f\ g\ (s1\ z) = s1'\ (f\ z)$ 
apply (tactic <dtac @ {context} @ {thm iffD1[OF mor_def]} 1 >)
apply (erule conjE)+
apply (erule bspec)
apply assumption
done

```

lemma *morE2*: $\llbracket mor\ B1\ B2\ s1\ s2\ B1'\ B2'\ s1'\ s2'\ f\ g; z \in B2 \rrbracket$

```

 $\implies F2map\ id\ f\ g\ (s2\ z) = s2'\ (g\ z)$ 
apply (tactic <dtac @ {context} @ {thm iffD1[OF mor_def]} 1 >)
apply (erule conjE)+
apply (erule bspec)
apply assumption
done

```

lemma *mor_incl*: $\llbracket B1 \subseteq B1'; B2 \subseteq B2' \rrbracket \implies mor\ B1\ B2\ s1\ s2\ B1'\ B2'\ s1\ s2\ id\ id$

```

apply (tactic <rtac @ {context} (@ {thm mor_def} RS iffD2) 1 >)
apply (rule conjI)
apply (rule conjI)
apply (rule ballI)
apply (rule ssubst_mem[OF id_apply])
apply (erule subsetD)
apply assumption

```

```

apply (rule ballI)
apply (rule ssubst_mem[OF id_apply])
apply (erule subsetD)
apply assumption

```

```

apply (rule conjI)
apply (rule ballI)
apply (rule trans[OF F1.map_id])

```

```

apply (rule sym)
apply (rule arg_cong[OF id_apply])
apply (rule ballI)
apply (rule trans[OF F2.map_id])
apply (rule sym)
apply (rule arg_cong[OF id_apply])
done

```

lemmas mor_id = mor_incl[OF subset_refl subset_refl]

lemma mor_comp:

```


$$\llbracket \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g; \\ \text{mor } B1' \ B2' \ s1' \ s2' \ B1'' \ B2'' \ s1'' \ s2'' \ f' \ g' \rrbracket \implies \\ \text{mor } B1 \ B2 \ s1 \ s2 \ B1'' \ B2'' \ s1'' \ s2'' \ (f' \circ f) \ (g' \circ g)$$

apply (tactic <rtac @\{context\} (@\{thm mor_def\} RS iffD2) 1>)
apply (rule conjI)

```

```

apply (rule conjI)
apply (rule ballI)
apply (rule ssubst_mem[OF o_apply])
apply (erule mor_image1')
apply (erule mor_image1')
apply assumption

```

```

apply (rule ballI)
apply (rule ssubst_mem[OF o_apply])
apply (erule mor_image2')
apply (erule mor_image2')
apply assumption

```

```

apply (rule conjI)
apply (rule ballI)
apply (tactic <stac @\{context\} @\{thm o_apply\} 1>)
apply (rule trans)
apply (rule sym[OF F1map_comp_id])
apply (rule trans)
apply (erule arg_cong[OF morE1])
apply assumption
apply (erule morE1)
apply (erule mor_image1')
apply assumption

```

```

apply (rule ballI)
apply (tactic <stac @\{context\} @\{thm o_apply\} 1>)
apply (rule trans)
apply (rule sym[OF F2map_comp_id])
apply (rule trans)
apply (erule arg_cong[OF morE2])
apply assumption
apply (erule morE2)
apply (erule mor_image2')
apply assumption
done

```

lemma mor_cong: $\llbracket f' = f; g' = g; \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f \ g \rrbracket \implies$
 $\text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f' \ g'$
apply (tactic <hyp_subst_tac @\{context\} 1>)
apply assumption
done

lemma mor_UNIV: $\text{mor } UNIV \ UNIV \ s1 \ s2 \ UNIV \ UNIV \ s1' \ s2' \ f1 \ f2 \longleftrightarrow$
 $F1map \ id \ f1 \ f2 \ o \ s1 = s1' \ o \ f1 \wedge F2map \ id \ f1 \ f2 \ o \ s2 = s2' \ o \ f2$
apply (rule iffI)


```

apply (rule conjI)
apply (rule ext)
apply (rule trans)
  apply (rule trans)
    apply (rule o_apply)
  apply (erule morE1)
  apply (rule UNIV_I)
apply (rule sym[OF o_apply])
apply (rule ext)
apply (rule trans)
apply (rule trans)
  apply (rule o_apply)
apply (erule morE2)
apply (rule UNIV_I)
apply (rule sym[OF o_apply])

```

```

apply (tactic <rtac @{context}> (@{thm mor_def} RS iffD2) 1)
apply (rule conjI)
apply (rule conjI)
  apply (rule ballI)
  apply (rule UNIV_I)
apply (rule ballI)
apply (rule UNIV_I)
apply (rule conjI)
apply (tactic <dtac @{context}> (BNF_Util.mk_conjunctN 2 1) 1)
apply (rule ballI)
apply (erule o_eq_dest)
apply (tactic <dtac @{context}> (BNF_Util.mk_conjunctN 2 2) 1)
apply (rule ballI)
apply (erule o_eq_dest)
done

```

lemma mor_str:

```

mor UNIV UNIV s1 s2 UNIV UNIV (F1map id s1 s2) (F2map id s1 s2) s1 s2
apply (rule iffD2)
apply (rule mor_UNIV)
apply (rule conjI)
  apply (rule refl)
apply (rule refl)
done

```

lemma mor_case_sum:

```

mor UNIV UNIV s1 s2 UNIV UNIV (case_sum (F1map id Inl Inl o s1) s1') (case_sum (F2map id Inl Inl o s2)
s2') Inl Inl
apply (tactic <rtac @{context}> (@{thm mor_UNIV} RS iffD2) 1)
apply (rule conjI)
apply (rule sym)
  apply (rule case_sum_o_inj(1))
apply (rule sym)
apply (rule case_sum_o_inj(1))
done

```

2.4 Bisimulations

definition bis where

```

bis B1 B2 s1 s2 B1' B2' s1' s2' R1 R2 =
((R1 ⊆ B1 × B1' ∧ R2 ⊆ B2 × B2') ∧
(∀ b1 b1'. (b1, b1') ∈ R1 →
(∃ z ∈ F1in UNIV R1 R2.
F1map id fst fst z = s1 b1 ∧ F1map id snd snd z = s1' b1')) ∧
(∀ b2 b2'. (b2, b2') ∈ R2 →
(∃ z ∈ F2in UNIV R1 R2.
F2map id fst fst z = s2 b2 ∧ F2map id snd snd z = s2' b2'))))

```

lemma *bis_cong*: $\llbracket \text{bis } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ R1 \ R2; \ R1' = R1; \ R2' = R2 \rrbracket \implies$
 $\text{bis } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ R1' \ R2'$
apply (tactic <hyp_subst_tac @{context} 1>)
apply assumption
done

lemma *bis_Frel*:
 $\text{bis } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ R1 \ R2 \longleftrightarrow$
 $(R1 \subseteq B1 \times B1' \wedge R2 \subseteq B2 \times B2') \wedge$
 $((\forall b1 \ b1'. (b1, b1') \in R1 \longrightarrow F1rel (=) (in_rel \ R1) (in_rel \ R2) (s1 \ b1) (s1' \ b1')) \wedge$
 $(\forall b2 \ b2'. (b2, b2') \in R2 \longrightarrow F2rel (=) (in_rel \ R1) (in_rel \ R2) (s2 \ b2) (s2' \ b2'))))$
apply (rule trans[OF bis_def])
apply (rule iffI)
apply (erule conjE)
apply (erule conjI)

apply (rule conjI)
apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule allE)+
apply (erule impE)
apply assumption
apply (erule bexE)
apply (erule conjE CollectE)+
apply (rule iffD2[OF F1.in_rel])
apply (rule exI)
apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans)
apply (rule trans)
apply (rule F1.map_comp)
apply (rule F1.map_cong0)
apply (rule fst_diag_id)
apply (rule fun_cong[OF o_id])
apply (rule fun_cong[OF o_id])
apply assumption

apply (rule trans)
apply (rule trans)
apply (rule F1.map_comp)
apply (rule F1.map_cong0)
apply (rule snd_diag_id)
apply (rule fun_cong[OF o_id])
apply (rule fun_cong[OF o_id])
apply assumption

apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans)
apply (rule F1.set_map(1))
apply (rule subset_trans)
apply (erule image_mono)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule refl)
apply (rule conjI)
apply (rule ord_eq_le_trans)
apply (rule trans)
apply (rule F1.set_map(2))
apply (rule trans[OF fun_cong[OF image_id] id_apply])

```

apply (erule Collect_case_prod_in_rel_leI)
apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule F1.set_map(3))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply (erule Collect_case_prod_in_rel_leI)

apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule allE)+
apply (erule impE)
apply assumption
apply (erule bexE)
apply (erule conjE CollectE)+
apply (rule iffD2[OF F2.in_rel])
apply (rule exI)
apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans)
  apply (rule trans)
    apply (rule F2.map_comp)
apply (rule F2.map_cong0)
  apply (rule fst_diag_id)
  apply (rule fun_cong[OF o_id])
apply (rule fun_cong[OF o_id])
apply assumption

apply (rule trans)
apply (rule trans)
  apply (rule F2.map_comp)
apply (rule F2.map_cong0)
  apply (rule snd_diag_id)
apply (rule fun_cong[OF o_id])
apply (rule fun_cong[OF o_id])
apply assumption

apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(1))
apply (rule subset_trans)
  apply (erule image_mono)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule refl)
apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule trans)
    apply (rule F2.set_map(2))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply (erule Collect_case_prod_in_rel_leI)
apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule F2.set_map(3))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply (erule Collect_case_prod_in_rel_leI)

apply (erule conjE)
apply (erule conjI)

```

```

apply (rule conjI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (erule allE)
apply (erule allE)
apply (erule impE)
apply assumption
apply (drule iffD1[OF F1.in_rel])
apply (erule exE conjE CollectE Collect_case_prod_in_rel_leE)+

```

```

apply (rule bexI)
apply (rule conjI)
apply (rule trans)
  apply (rule F1.map_comp)
apply (tactic <stac @{context} @{thm id_o} 1>)
apply (tactic <stac @{context} @{thm o_id} 1>)
apply (tactic <stac @{context} @{thm o_id} 1>)
apply assumption

```

```

apply (rule trans)
apply (rule F1.map_comp)
apply (tactic <stac @{context} @{thm id_o} 1>)
apply (tactic <stac @{context} @{thm o_id} 1>)
apply (tactic <stac @{context} @{thm o_id} 1>)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule Collect_case_prodD)
  apply (erule subsetD)
  apply assumption
apply (rule refl)
apply (rule refl)
apply assumption

```

```

apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)

```

```

apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule trans)
  apply (rule F1.set_map(2))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption

```

```

apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule F1.set_map(3))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption

```

```

apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (erule allE)
apply (erule allE)
apply (erule impE)
apply assumption
apply (drule iffD1[OF F2.in_rel])
apply (erule exE conjE CollectE Collect_case_prod_in_rel_leE)+

```

```

apply (rule beqI)
apply (rule conjI)
apply (rule trans)
  apply (rule F2.map_comp)
apply (tactic ‹stac @{{context}} @{{thm id_o}} 1›)
apply (tactic ‹stac @{{context}} @{{thm o_id}} 1›)
apply (tactic ‹stac @{{context}} @{{thm o_id}} 1›)
apply assumption

apply (rule trans)
apply (rule F2.map_comp)
apply (tactic ‹stac @{{context}} @{{thm id_o}} 1›)
apply (tactic ‹stac @{{context}} @{{thm o_id}} 1›)
apply (tactic ‹stac @{{context}} @{{thm o_id}} 1›)
apply (rule trans)
apply (rule F2.map_cong0)
  apply (rule Collect_case_prodD)
  apply (erule subsetD)
  apply assumption
  apply (rule refl)
apply (rule refl)
apply assumption

apply (rule CollectI)
apply (rule conjI)
apply (rule subset_UNIV)

apply (rule conjI)
apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule F2.set_map(2))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption

apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule F2.set_map(3))
apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption
done

lemma bis_converse:
  bis B1 B2 s1 s2 B1' B2' s1' s2' R1 R2  $\implies$ 
  bis B1' B2' s1' s2' B1 B2 s1 s2 (R1 $\hat{=}$ -1) (R2 $\hat{=}$ -1)
apply (tactic ‹rtac @{{context}} (@{{thm bis_Frel}} RS iffD2) 1›)
apply (tactic ‹dtac @{{context}} (@{{thm bis_Frel}} RS iffD1) 1›)
apply (erule conjE)+
apply (rule conjI)

apply (rule conjI)
apply (rule iffD1[OF converse_subset_swap])
apply (erule subset_trans)
apply (rule equalityD2)
apply (rule converse_Times)

apply (rule iffD1[OF converse_subset_swap])
apply (erule subset_trans)
apply (rule equalityD2)
apply (rule converse_Times)

apply (rule conjI)

```

```

apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (rule predicate2D[OF eq_refl[OF F1rel_cong]])
  apply (rule conversep_eq)
  apply (rule conversep_in_rel)
apply (rule conversep_in_rel)
apply (rule predicate2D[OF eq_refl[OF sym[OF F1.rel_conversep]]])
apply (erule allE)+
apply (rule conversepI)
apply (erule mp)
apply (erule converseD)

```

```

apply (rule allI)
apply (rule allI)
apply (rule impI)
apply (rule predicate2D[OF eq_refl[OF F2rel_cong]])
  apply (rule conversep_eq)
  apply (rule conversep_in_rel)
apply (rule conversep_in_rel)
apply (rule predicate2D[OF eq_refl[OF sym[OF F2.rel_conversep]]])
apply (erule allE)+
apply (rule conversepI)
apply (erule mp)
apply (erule converseD)
done

```

lemma bis_Comp:

```

[[bis B1 B2 s1 s2 B1' B2' s1' s2' P1 P2;
  bis B1' B2' s1' s2' B1'' B2'' s1'' s2'' Q1 Q2]] ==>
  bis B1 B2 s1 s2 B1'' B2'' s1'' s2'' (P1 O Q1) (P2 O Q2)
apply (tactic <rtac @ {context} (@ {thm bis_Frel[THEN iffD2]}) 1>)
apply (tactic <dtac @ {context} (@ {thm bis_Frel[THEN iffD1]}) 1>)+
apply (erule conjE)+
apply (rule conjI)
apply (rule conjI)
  apply (erule relcomp_subset_Sigma)
  apply assumption
apply (erule relcomp_subset_Sigma)
apply assumption

apply (rule conjI)
apply (rule allI)+
apply (rule impI)
apply (rule predicate2D[OF eq_refl[OF F1rel_cong]])
  apply (rule eq_OO)
  apply (rule relcompp_in_rel)
apply (rule relcompp_in_rel)
apply (rule predicate2D[OF eq_refl[OF sym[OF F1.rel_compp]]])
apply (erule relcompE)
apply (tactic <dtac @ {context} (@ {thm prod.inject[THEN iffD1]}) 1>)
apply (erule conjE)
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (erule allE)+
apply (rule relcomppI)
  apply (erule mp)
  apply assumption
apply (erule mp)
apply assumption

```

```

apply (rule allI)+
apply (rule impI)
apply (rule predicate2D[OF eq_refl[OF F2rel_cong]])

```

```

  apply (rule eq_OO)
  apply (rule relcompp_in_rel)
  apply (rule relcompp_in_rel)
  apply (rule predicate2D[OF eq_refl[OF sym[OF F2.rel_compp]]])
  apply (erule relcompE)
  apply (tactic <dtac @{context} (@{thm prod.inject[THEN iffD1]}) 1>)
  apply (erule conjE)
  apply (tactic <hyp_subst_tac @{context} 1>)
  apply (erule allE)+
  apply (rule relcomppI)
  apply (erule mp)
  apply assumption
  apply (erule mp)
  apply assumption
done

```

lemma *bis_Gr*: $\llbracket \text{coalg } B1 \ B2 \ s1 \ s2; \text{mor } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ f1 \ f2 \rrbracket \implies$
 $\text{bis } B1 \ B2 \ s1 \ s2 \ B1' \ B2' \ s1' \ s2' \ (\text{BNF_Def.Gr } B1 \ f1) \ (\text{BNF_Def.Gr } B2 \ f2)$
unfolding *bis_Frel eq_alt in_rel_Gr F1.rel_Grp F2.rel_Grp*
 apply (rule conjI)
 apply (rule conjI)
 apply (rule iffD2[OF Gr_incl])
 apply (erule mor_image1)
 apply (rule iffD2[OF Gr_incl])
 apply (erule mor_image2)

```

  apply (rule conjI)
  apply (rule allI)
  apply (rule allI)
  apply (rule impI)
  apply (rule GrpI)
  apply (erule trans[OF morE1])
  apply (erule GrD1)
  apply (erule arg_cong[OF GrD2])
  apply (erule coalg_F1in)
  apply (erule GrD1)

```

```

  apply (rule allI)
  apply (rule allI)
  apply (rule impI)
  apply (rule GrpI)
  apply (erule trans[OF morE2])
  apply (erule GrD1)
  apply (erule arg_cong[OF GrD2])
  apply (erule coalg_F2in)
  apply (erule GrD1)
done

```

lemmas *bis_image2* = *bis_cong*[*OF bis_Comp*[*OF bis_converse*[*OF bis_Gr*] *bis_Gr*] *image2_Gr image2_Gr*]
lemmas *bis_diag* = *bis_cong*[*OF bis_Gr*[*OF _ mor_id*] *Id_on_Gr Id_on_Gr*]

lemma *bis_Union*: $\forall i \in I. \text{bis } B1 \ B2 \ s1 \ s2 \ B1 \ B2 \ s1 \ s2 \ (R1i \ i) \ (R2i \ i) \implies$
 $\text{bis } B1 \ B2 \ s1 \ s2 \ B1 \ B2 \ s1 \ s2 \ (\bigcup_{i \in I. R1i \ i}) \ (\bigcup_{i \in I. R2i \ i})$
unfolding *bis_def*
 apply (rule conjI)
 apply (rule conjI)
 apply (rule UN_least)
 apply (drule bspec)
 apply assumption
 apply (drule conjunct1)
 apply (tactic <etac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
 apply (rule UN_least)
 apply (drule bspec)

```

apply assumption
apply (drule conjunct1)
apply (tactic <etac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)

```

```

apply (rule conjI)
apply (rule allI)+
apply (rule impI)
apply (erule UN_E)
apply (drule bspec)
apply assumption
apply (drule conjunct2)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule allE)+
apply (drule mp)
apply assumption
apply (erule bexE)
apply (rule bexE)
apply assumption
apply (rule F1in_mono23')
apply (erule SUP_upper2[OF _ subset_refl])
apply (erule SUP_upper2[OF _ subset_refl])
apply assumption

```

```

apply (rule allI)+
apply (rule impI)
apply (erule UN_E)
apply (drule bspec)
apply assumption
apply (drule conjunct2)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule allE)+
apply (drule mp)
apply assumption
apply (erule bexE)
apply (rule bexE)
apply assumption
apply (rule F2in_mono23')
apply (erule SUP_upper2[OF _ subset_refl])
apply (erule SUP_upper2[OF _ subset_refl])
apply assumption
done

```

abbreviation *sbis* *B1 B2 s1 s2 R1 R2* \equiv *bis* *B1 B2 s1 s2 B1 B2 s1 s2 R1 R2*

definition *lsbis1* **where** *lsbis1* *B1 B2 s1 s2* =
 $(\bigcup R \in \{(R1, R2) \mid R1 R2 . sbis B1 B2 s1 s2 R1 R2\}. fst R)$

definition *lsbis2* **where** *lsbis2* *B1 B2 s1 s2* =
 $(\bigcup R \in \{(R1, R2) \mid R1 R2 . sbis B1 B2 s1 s2 R1 R2\}. snd R)$

lemma *sbis_lsbis*:

```

sbis B1 B2 s1 s2 (lsbis1 B1 B2 s1 s2) (lsbis2 B1 B2 s1 s2)
apply (tactic <rtac @{context} (Thm.permute_premis 0 1 @{thm bis_cong}) 1>)
apply (rule lsbis1_def)
apply (rule lsbis2_def)
apply (rule bis_Union)
apply (rule ballI)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule bis_cong)
apply (rule fst_conv)

```



```

apply (rule snd_conv)
done

lemmas lsbis1_incl = conjunct1[OF conjunct1[OF iffD1[OF bis_def]], OF sbis_lsbis]
lemmas lsbis2_incl = conjunct2[OF conjunct1[OF iffD1[OF bis_def]], OF sbis_lsbis]
lemmas lsbisE1 =
  mp[OF spec[OF spec[OF conjunct1[OF conjunct2[OF iffD1[OF bis_def]], OF sbis_lsbis]]]]
lemmas lsbisE2 =
  mp[OF spec[OF spec[OF conjunct2[OF conjunct2[OF iffD1[OF bis_def]], OF sbis_lsbis]]]]

lemma incl_lsbis1: sbis B1 B2 s1 s2 R1 R2  $\implies$  R1  $\subseteq$  lsbis1 B1 B2 s1 s2
apply (rule xt1(3))
apply (rule lsbis1_def)
apply (rule SUP_upper2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply assumption
apply (rule equalityD2)
apply (rule fst_conv)
done

lemma incl_lsbis2: sbis B1 B2 s1 s2 R1 R2  $\implies$  R2  $\subseteq$  lsbis2 B1 B2 s1 s2
apply (rule xt1(3))
apply (rule lsbis2_def)
apply (rule SUP_upper2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply assumption
apply (rule equalityD2)
apply (rule snd_conv)
done

lemma equiv_lsbis1: coalg B1 B2 s1 s2  $\implies$  equiv B1 (lsbis1 B1 B2 s1 s2)
apply (rule iffD2[OF equiv_def])

apply (rule conjI)
apply (rule iffD2[OF refl_on_def])
apply (rule conjI)
apply (rule lsbis1_incl)
apply (rule ballI)
apply (rule subsetD)
apply (rule incl_lsbis1)
apply (rule bis_diag)
apply assumption
apply (erule Id_onI)

apply (rule conjI)
apply (rule iffD2[OF sym_def])
apply (rule allI)+
apply (rule impI)
apply (rule subsetD)
apply (rule incl_lsbis1)
apply (rule bis_converse)
apply (rule sbis_lsbis)
apply (erule converseI)

apply (rule iffD2[OF trans_def])
apply (rule allI)+
apply (rule impI)+

```

```

apply (rule subsetD)
apply (rule incl_lsbis1)
apply (rule bis_Comp)
  apply (rule sbis_lsbis)
apply (rule sbis_lsbis)
apply (erule relcompI)
apply assumption
done

```

```

lemma equiv_lsbis2: coalg B1 B2 s1 s2  $\implies$  equiv B2 (lsbis2 B1 B2 s1 s2)
unfolding equiv_def refl_on_def sym_def trans_def
apply (rule conjI)

```

```

  apply (rule conjI)
    apply (rule lsbis2_incl)
  apply (rule ballI)
apply (rule subsetD)
  apply (rule incl_lsbis2)
  apply (rule bis_diag)
  apply assumption
apply (erule Id_onI)

```

```

apply (rule conjI)

```

```

  apply (rule allI)+
apply (rule impI)
apply (rule subsetD)
  apply (rule incl_lsbis2)
  apply (rule bis_converse)
  apply (rule sbis_lsbis)
apply (erule converseI)

```

```

apply (rule allI)+
apply (rule impI)+
apply (rule subsetD)
  apply (rule incl_lsbis2)
  apply (rule bis_Comp)
  apply (rule sbis_lsbis)
apply (rule sbis_lsbis)
apply (erule relcompI)
apply assumption
done

```

2.5 The Tree Coalgebra

```

typedef bd_type_F = UNIV :: (bd_type_F1 + bd_type_F2) set
apply (rule exI) apply (rule UNIV_I)
done

```

```

type-synonym 'a carrier = ((bd_type_F + bd_type_F) list set  $\times$ 
  ((bd_type_F + bd_type_F) list  $\Rightarrow$  ('a, bd_type_F, bd_type_F) F1 + ('a, bd_type_F, bd_type_F) F2))

```

```

abbreviation bd_F  $\equiv$  dir_image (bd_F1 +c bd_F2) Abs_bd_type_F
lemmas bd_F = dir_image[OF Abs_bd_type_F inject[OF UNIV_I UNIV_I] Card_order_csum]
lemmas bd_F_Cinfinite = Cinfinite_cong[OF bd_F Cinfinite_csum1[OF F1.bd_Cinfinite]]
lemmas bd_F_Card_order = Card_order_ordIso[OF Card_order_csum ordIso_symmetric[OF bd_F]]
lemma bd_F_card_order: card_order bd_F
apply (rule card_order_dir_image)
apply (rule bijI')
  apply (rule Abs_bd_type_F inject[OF UNIV_I UNIV_I])
apply (rule Abs_bd_type_F_cases)
apply (erule exI)
apply (rule card_order_csum)
apply (rule F1.bd_card_order)

```

apply (rule *F2.bd_card_order*)
done

lemmas *F1set1_bd'* = *ordLeq_transitive*[*OF F1.set_bd(1) ordLeq_ordIso_trans*[*OF ordLeq_csum1*[*OF F1.bd_Card_order*]
bd_F]]

lemmas *F1set2_bd'* = *ordLeq_transitive*[*OF F1.set_bd(2) ordLeq_ordIso_trans*[*OF ordLeq_csum1*[*OF F1.bd_Card_order*]
bd_F]]

lemmas *F1set3_bd'* = *ordLeq_transitive*[*OF F1.set_bd(3) ordLeq_ordIso_trans*[*OF ordLeq_csum1*[*OF F1.bd_Card_order*]
bd_F]]

lemmas *F2set1_bd'* = *ordLeq_transitive*[*OF F2.set_bd(1) ordLeq_ordIso_trans*[*OF ordLeq_csum2*[*OF F2.bd_Card_order*]
bd_F]]

lemmas *F2set2_bd'* = *ordLeq_transitive*[*OF F2.set_bd(2) ordLeq_ordIso_trans*[*OF ordLeq_csum2*[*OF F2.bd_Card_order*]
bd_F]]

lemmas *F2set3_bd'* = *ordLeq_transitive*[*OF F2.set_bd(3) ordLeq_ordIso_trans*[*OF ordLeq_csum2*[*OF F2.bd_Card_order*]
bd_F]]

abbreviation *Succ1 Kl kl* $\equiv \{k1. \text{Inl } k1 \in \text{BNF_Greatest_Fixpoint.Succ } Kl \text{ } kl\}$

abbreviation *Succ2 Kl kl* $\equiv \{k2. \text{Inr } k2 \in \text{BNF_Greatest_Fixpoint.Succ } Kl \text{ } kl\}$

definition *isNode1* **where**

isNode1 Kl lab kl = $(\exists x1. \text{lab } kl = \text{Inl } x1 \wedge \text{F1set2 } x1 = \text{Succ1 } Kl \text{ } kl \wedge \text{F1set3 } x1 = \text{Succ2 } Kl \text{ } kl)$

definition *isNode2* **where**

isNode2 Kl lab kl = $(\exists x2. \text{lab } kl = \text{Inr } x2 \wedge \text{F2set2 } x2 = \text{Succ1 } Kl \text{ } kl \wedge \text{F2set3 } x2 = \text{Succ2 } Kl \text{ } kl)$

abbreviation *isTree* **where**

isTree Kl lab $\equiv (\square \in Kl \wedge$

$(\forall kl \in Kl. (\forall k1 \in \text{Succ1 } Kl \text{ } kl. \text{isNode1 } Kl \text{ } lab \text{ } (kl \text{ } @ \text{ } [\text{Inl } k1])) \wedge$

$(\forall k2 \in \text{Succ2 } Kl \text{ } kl. \text{isNode2 } Kl \text{ } lab \text{ } (kl \text{ } @ \text{ } [\text{Inr } k2])))$

definition *carT1* **where**

carT1 = $\{(Kl :: (\text{bd_type_F} + \text{bd_type_F}) \text{ list set}, lab) \mid Kl \text{ } lab. \text{isTree } Kl \text{ } lab \wedge \text{isNode1 } Kl \text{ } lab \ \square\}$

definition *carT2* **where**

carT2 = $\{(Kl :: (\text{bd_type_F} + \text{bd_type_F}) \text{ list set}, lab) \mid Kl \text{ } lab. \text{isTree } Kl \text{ } lab \wedge \text{isNode2 } Kl \text{ } lab \ \square\}$

definition *strT1* **where**

strT1 = $(\text{case_prod } (\%Kl \text{ } lab. \text{case_sum } (\text{F1map } id$
 $(\lambda k1. (\text{BNF_Greatest_Fixpoint.Shift } Kl \text{ } (\text{Inl } k1), \text{BNF_Greatest_Fixpoint.shift } lab \text{ } (\text{Inl } k1)))$
 $(\lambda k2. (\text{BNF_Greatest_Fixpoint.Shift } Kl \text{ } (\text{Inr } k2), \text{BNF_Greatest_Fixpoint.shift } lab \text{ } (\text{Inr } k2)))) \text{ undefined } (lab$
 $\square))$

definition *strT2* **where**

strT2 = $(\text{case_prod } (\%Kl \text{ } lab. \text{case_sum } \text{undefined } (\text{F2map } id$
 $(\lambda k1. (\text{BNF_Greatest_Fixpoint.Shift } Kl \text{ } (\text{Inl } k1), \text{BNF_Greatest_Fixpoint.shift } lab \text{ } (\text{Inl } k1)))$
 $(\lambda k2. (\text{BNF_Greatest_Fixpoint.Shift } Kl \text{ } (\text{Inr } k2), \text{BNF_Greatest_Fixpoint.shift } lab \text{ } (\text{Inr } k2)))) (lab \ \square))$

lemma *coalg_T*: *coalg carT1 carT2 strT1 strT2*

unfolding *coalg_def carT1_def carT2_def isNode1_def isNode2_def*

apply (rule *conjI*)

apply (rule *ballI*)

apply (erule *CollectE exE conjE*)⁺

apply (tactic $\langle \text{hyp_subst_tac } @ \{ \text{context} \} \ 1 \rangle$)

apply (rule *ssubst_mem*[*OF trans*[*OF trans*[*OF fun_cong*[*OF strT1_def*] *prod.case*]]])

apply (erule *trans*[*OF arg_cong*])

apply (rule *sum.case*(1))

apply (rule *CollectI*)

apply (rule *conjI*)

apply (rule *subset_UNIV*)

apply (rule *conjI*)

apply (rule *ord_eq_le_trans*[*OF F1.set_map*(2)])

apply (rule *image_subsetI*)

```

apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (rule conjI)
apply (rule conjI)
  apply (erule empty_Shift)
  apply (drule rev_subsetD)
    apply (erule equalityD1)
    apply (erule CollectD)
apply (rule ballI)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
  apply (erule thin_rl)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule bspec)
  apply (rule CollectI)
  apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
  apply assumption
apply (rule conjI)
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
  apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
  apply (erule thin_rl)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
  apply (rule CollectI)
  apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
  apply assumption
apply (rule conjI)
apply (erule trans)

```

```

apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

apply (drule bspec)
  apply assumption
apply (tactic ‹dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1›)
apply (drule bspec)
  apply (erule subsetD[OF equalityD1])
  apply assumption
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (erule trans[OF arg_cong[OF sym[OF append_Nil]]])
apply (rule conjI)
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
  apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Nil]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Nil]])

apply (rule ord_eq_le_trans[OF F1.set_map(3)])
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (rule conjI)
apply (rule conjI)
  apply (erule empty_Shift)
  apply (drule rev_subsetD)
  apply (erule equalityD1)
apply (erule CollectD)
apply (rule ballI)
apply (rule conjI)
  apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
  apply (erule thin_rl)
  apply assumption
apply (tactic ‹dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1›)
apply (drule bspec)

```

```

apply (rule CollectI)
apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
apply (rule trans[OF fun_cong[OF shift_def]])
apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
apply assumption
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
apply (erule thin_rl)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
apply (rule CollectI)
apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
apply (rule trans[OF fun_cong[OF shift_def]])
apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
apply assumption
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (drule bspec)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
apply (erule subsetD[OF equalityD1])
apply assumption

```

```

apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (erule trans[OF arg_cong[OF sym[OF append_Nil]]])
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Nil]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Nil]])

apply (rule ballI)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @ {context} 1 >)
apply (rule ssubst_mem[OF trans[OF fun_cong[OF strT2_def] prod.case]])
apply (rule ssubst_mem)
  apply (rule trans)
    apply (erule arg_cong)
    apply (rule sum.case(2))
apply (rule CollectI)
apply (rule conjI)
  apply (rule subset_UNIV)
apply (rule conjI)
apply (rule ord_eq_le_trans[OF F2.set_map(2)])
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (rule conjI)
apply (rule conjI)
  apply (erule empty_Shift)
  apply (drule rev_subsetD)
  apply (erule equalityD1)
apply (erule CollectD)
apply (rule ballI)
apply (rule conjI)
  apply (rule ballI)
  apply (erule CollectE)
  apply (drule ShiftD)
  apply (drule bspec)
  apply (erule thin_rl)
  apply assumption
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1 >)
apply (drule bspec)
apply (rule CollectI)
apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])

```

```

apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
apply assumption
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
apply (erule thin_rl)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
  apply (rule CollectI)
  apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
  apply assumption
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (drule bspec)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule bspec)
  apply (erule subsetD[OF equalityD1])
  apply assumption
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (erule trans[OF arg_cong[OF sym[OF append_Nil]]])
apply (rule conjI)

```



```

apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Nil]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Nil]])

apply (rule ord_eq_le_trans[OF F2.set_map(3)])
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (rule conjI)
apply (rule conjI)
  apply (erule empty_Shift)
apply (drule rev_subsetD)
  apply (erule equalityD1)
apply (erule CollectD)
apply (rule ballI)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
  apply (erule thin_rl)
apply assumption
apply (tactic {dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1})
apply (drule bspec)
  apply (rule CollectI)
  apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
apply (erule exE conjE)+
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
  apply assumption
apply (rule conjI)
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])
apply (erule trans)
apply (rule Collect_cong)
apply (rule eqset_imp_iff)
apply (rule sym)
apply (rule trans)
  apply (rule Succ_Shift)
apply (rule arg_cong[OF sym[OF append_Cons]])

apply (rule ballI)

```

```

apply (erule CollectE)
apply (drule ShiftD)
apply (drule bspec)
  apply (erule thin_rl)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
  apply (rule CollectI)
  apply (erule subsetD[OF equalityD1[OF Succ_Shift]])
  apply (erule exE conjE)+
  apply (rule exI)
  apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (rule trans[OF arg_cong[OF sym[OF append_Cons]]])
  apply assumption
  apply (rule conjI)
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
    apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Cons]])
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
    apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Cons]])

```

```

apply (drule bspec)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule bspec)
  apply (erule subsetD[OF equalityD1])
  apply assumption
  apply (erule exE conjE)+
  apply (rule exI)
  apply (rule conjI)
  apply (rule trans[OF fun_cong[OF shift_def]])
  apply (erule trans[OF arg_cong[OF sym[OF append_Nil]]])
  apply (rule conjI)
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
    apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Nil]])
  apply (erule trans)
  apply (rule Collect_cong)
  apply (rule eqset_imp_iff)
  apply (rule sym)
  apply (rule trans)
    apply (rule Succ_Shift)
  apply (rule arg_cong[OF sym[OF append_Nil]])
done

```

abbreviation *tobd_F12* **where** *tobd_F12 s1 x* \equiv *toCard (F1set2 (s1 x)) bd_F*
abbreviation *tobd_F13* **where** *tobd_F13 s1 x* \equiv *toCard (F1set3 (s1 x)) bd_F*
abbreviation *tobd_F22* **where** *tobd_F22 s2 x* \equiv *toCard (F2set2 (s2 x)) bd_F*
abbreviation *tobd_F23* **where** *tobd_F23 s2 x* \equiv *toCard (F2set3 (s2 x)) bd_F*

abbreviation *frombd_F12* **where** *frombd_F12 s1 x* \equiv *fromCard (F1set2 (s1 x)) bd_F*
abbreviation *frombd_F13* **where** *frombd_F13 s1 x* \equiv *fromCard (F1set3 (s1 x)) bd_F*
abbreviation *frombd_F22* **where** *frombd_F22 s2 x* \equiv *fromCard (F2set2 (s2 x)) bd_F*
abbreviation *frombd_F23* **where** *frombd_F23 s2 x* \equiv *fromCard (F2set3 (s2 x)) bd_F*

lemmas *tobd_F12_inj* = *toCard_inj[OF F1set2_bd' bd_F_Card_order]*
lemmas *tobd_F13_inj* = *toCard_inj[OF F1set3_bd' bd_F_Card_order]*
lemmas *tobd_F22_inj* = *toCard_inj[OF F2set2_bd' bd_F_Card_order]*
lemmas *tobd_F23_inj* = *toCard_inj[OF F2set3_bd' bd_F_Card_order]*
lemmas *frombd_F12_tobd_F12* = *fromCard_toCard[OF F1set2_bd' bd_F_Card_order]*
lemmas *frombd_F13_tobd_F13* = *fromCard_toCard[OF F1set3_bd' bd_F_Card_order]*
lemmas *frombd_F22_tobd_F22* = *fromCard_toCard[OF F2set2_bd' bd_F_Card_order]*
lemmas *frombd_F23_tobd_F23* = *fromCard_toCard[OF F2set3_bd' bd_F_Card_order]*

definition *Lev* **where**

Lev s1 s2 = *rec_nat (%a. {}, %b. {})*
(%n rec.
(%a1.
 $\{Inl (tobd_F12\ s1\ a1\ b1) \# kl \mid b1\ kl.\ b1 \in F1set2\ (s1\ a1) \wedge kl \in fst\ rec\ b1\} \cup$
 $\{Inr (tobd_F13\ s1\ a1\ b2) \# kl \mid b2\ kl.\ b2 \in F1set3\ (s1\ a1) \wedge kl \in snd\ rec\ b2\},$
%a2.
 $\{Inl (tobd_F22\ s2\ a2\ b1) \# kl \mid b1\ kl.\ b1 \in F2set2\ (s2\ a2) \wedge kl \in fst\ rec\ b1\} \cup$
 $\{Inr (tobd_F23\ s2\ a2\ b2) \# kl \mid b2\ kl.\ b2 \in F2set3\ (s2\ a2) \wedge kl \in snd\ rec\ b2\})$)

abbreviation *Lev1* **where** *Lev1 s1 s2 n* \equiv *fst (Lev s1 s2 n)*

abbreviation *Lev2* **where** *Lev2 s1 s2 n* \equiv *snd (Lev s1 s2 n)*

lemmas *Lev1_0* = *fun_cong[OF fstI[OF rec_nat_0_imp[OF Lev_def]]]*
lemmas *Lev2_0* = *fun_cong[OF sndI[OF rec_nat_0_imp[OF Lev_def]]]*
lemmas *Lev1_Suc* = *fun_cong[OF fstI[OF rec_nat_Suc_imp[OF Lev_def]]]*
lemmas *Lev2_Suc* = *fun_cong[OF sndI[OF rec_nat_Suc_imp[OF Lev_def]]]*

definition *rv* **where**

rv s1 s2 = *rec_list (%b1. Inl b1, %b2. Inr b2)*
(%k kl rec.
 $(\%b1.\ case_sum\ (\%k1.\ fst\ rec\ (frombd_F12\ s1\ b1\ k1))\ (\%k2.\ snd\ rec\ (frombd_F13\ s1\ b1\ k2))\ k,$
 $\%b2.\ case_sum\ (\%k1.\ fst\ rec\ (frombd_F22\ s2\ b2\ k1))\ (\%k2.\ snd\ rec\ (frombd_F23\ s2\ b2\ k2))\ k)$)

abbreviation *rv1* **where** *rv1 s1 s2 kl* \equiv *fst (rv s1 s2 kl)*

abbreviation *rv2* **where** *rv2 s1 s2 kl* \equiv *snd (rv s1 s2 kl)*

lemmas *rv1_Nil* = *fun_cong[OF fstI[OF rec_list_Nil_imp[OF rv_def]]]*
lemmas *rv2_Nil* = *fun_cong[OF sndI[OF rec_list_Nil_imp[OF rv_def]]]*
lemmas *rv1_Cons* = *fun_cong[OF fstI[OF rec_list_Cons_imp[OF rv_def]]]*
lemmas *rv2_Cons* = *fun_cong[OF sndI[OF rec_list_Cons_imp[OF rv_def]]]*

abbreviation *Lab1* *s1 s2 b1 kl* \equiv

$(case_sum\ (\%k.\ Inl\ (F1map\ id\ (tobd_F12\ s1\ k)\ (tobd_F13\ s1\ k)\ (s1\ k)))$
 $(\%k.\ Inr\ (F2map\ id\ (tobd_F22\ s2\ k)\ (tobd_F23\ s2\ k)\ (s2\ k)))\ (rv1\ s1\ s2\ kl\ b1))$

abbreviation *Lab2* *s1 s2 b2 kl* \equiv

$(case_sum\ (\%k.\ Inl\ (F1map\ id\ (tobd_F12\ s1\ k)\ (tobd_F13\ s1\ k)\ (s1\ k)))$
 $(\%k.\ Inr\ (F2map\ id\ (tobd_F22\ s2\ k)\ (tobd_F23\ s2\ k)\ (s2\ k)))\ (rv2\ s1\ s2\ kl\ b2))$

definition *beh1* *s1 s2 a* = $(\bigcup n.\ Lev1\ s1\ s2\ n\ a,\ Lab1\ s1\ s2\ a)$

definition *beh2* *s1 s2 a* = $(\bigcup n.\ Lev2\ s1\ s2\ n\ a,\ Lab2\ s1\ s2\ a)$

lemma *length_Lev*:

$\forall kl\ b1\ b2.\ (kl \in Lev1\ s1\ s2\ n\ b1 \longrightarrow length\ kl = n) \wedge$

```

      (kl ∈ Lev2 s1 s2 n b2 → length kl = n)
apply (rule nat_induct)
apply (rule allI)+
apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_0]])
apply (erule singletonE)
apply (erule ssubst)
apply (rule list.size(3))

apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev2_0]])
apply (erule singletonE)
apply (erule ssubst)
apply (rule list.size(3))

apply (rule allI)+
apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule trans)
  apply (rule length_Cons)
apply (rule arg_cong[of _ _ Suc])
apply (erule allE)+
apply (tactic ⟨dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1⟩)
apply (erule mp)
apply assumption

apply (erule CollectE exE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule trans)
  apply (rule length_Cons)
apply (rule arg_cong[of _ _ Suc])
apply (erule allE)+
apply (tactic ⟨dtac @ {context} (BNF_Util.mk_conjunctN 2 2) 1⟩)
apply (erule mp)
apply assumption

apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule trans)
  apply (rule length_Cons)
apply (rule arg_cong[of _ _ Suc])
apply (erule allE)+
apply (tactic ⟨dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1⟩)
apply (erule mp)
apply assumption

apply (erule CollectE exE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule trans)
  apply (rule length_Cons)
apply (rule arg_cong[of _ _ Suc])
apply (erule allE)+
apply (tactic ⟨dtac @ {context} (BNF_Util.mk_conjunctN 2 2) 1⟩)
apply (erule mp)

```

apply *assumption*
done

lemmas *length_Lev1* = *mp*[*OF conjunct1*[*OF spec*[*OF spec* [*OF spec*[*OF length_Lev*]]]]]
lemmas *length_Lev2* = *mp*[*OF conjunct2*[*OF spec*[*OF spec* [*OF spec*[*OF length_Lev*]]]]]

lemma *length_Lev1'*: $kl \in Lev1\ s1\ s2\ n\ a \implies kl \in Lev1\ s1\ s2\ (length\ kl)\ a$
apply (*frule length_Lev1*)
apply (*erule ssubst*)
apply *assumption*
done

lemma *length_Lev2'*: $kl \in Lev2\ s1\ s2\ n\ a \implies kl \in Lev2\ s1\ s2\ (length\ kl)\ a$
apply (*frule length_Lev2*)
apply (*erule ssubst*)
apply *assumption*
done

lemma *rv_last*:

$\forall k\ b1\ b2.$

$((\exists b1'.\ rv1\ s1\ s2\ (kl\ @\ [Inl\ k])\ b1 = Inl\ b1') \wedge$
 $(\exists b1'.\ rv1\ s1\ s2\ (kl\ @\ [Inr\ k])\ b1 = Inr\ b1')) \wedge$
 $((\exists b2'.\ rv2\ s1\ s2\ (kl\ @\ [Inl\ k])\ b2 = Inl\ b2') \wedge$
 $(\exists b2'.\ rv2\ s1\ s2\ (kl\ @\ [Inr\ k])\ b2 = Inr\ b2'))$

apply (*rule list.induct*[*of _ kl*])

apply (*rule allI*)⁺

apply (*rule conjI*)

apply (*rule conjI*)

apply (*rule exI*)

apply (*rule trans*[*OF arg_cong*[*OF append_Nil*]])

apply (*rule trans*[*OF rv1_Cons*])

apply (*rule trans*[*OF arg_cong*[*OF sum.case(1)*]])

apply (*rule rv1_Nil*)

apply (*rule exI*)

apply (*rule trans*[*OF arg_cong*[*OF append_Nil*]])

apply (*rule trans*[*OF rv1_Cons*])

apply (*rule trans*[*OF arg_cong*[*OF sum.case(2)*]])

apply (*rule rv2_Nil*)

apply (*rule conjI*)

apply (*rule exI*)

apply (*rule trans*[*OF arg_cong*[*OF append_Nil*]])

apply (*rule trans*[*OF rv2_Cons*])

apply (*rule trans*[*OF arg_cong*[*OF sum.case(1)*]])

apply (*rule rv1_Nil*)

apply (*rule exI*)

apply (*rule trans*[*OF arg_cong*[*OF append_Nil*]])

apply (*rule trans*[*OF rv2_Cons*])

apply (*rule trans*[*OF arg_cong*[*OF sum.case(2)*]])

apply (*rule rv2_Nil*)

apply (*rule allI*)⁺

apply (*rule sum.exhaust*)

apply (*rule conjI*)

apply (*erule allE*)⁺

apply (*tactic* <*dtac* @{*context*} (*BNF_Util.mk_conjunctN* 2 1) 1 >)

apply (*rule conjI*)

apply (*tactic* <*dtac* @{*context*} (*BNF_Util.mk_conjunctN* 2 1) 1 >)

apply (*erule exE*)

apply (*rule exI*)

apply (*rule trans*[*OF arg_cong*[*OF append_Cons*]])

apply (*rule trans*[*OF rv1_Cons*])

apply (*erule trans*[*OF arg_cong*[*OF sum.case_cong_weak*]])

apply (*rule trans*[*OF arg_cong*[*OF sum.case(1)*]])

```

apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv1_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply assumption

apply (erule allE) +
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (rule conjI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv2_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv2_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply assumption

apply (rule conjI)
apply (erule allE) +
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule conjI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv1_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv1_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply assumption

apply (erule allE) +
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule conjI)
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv2_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)

```

```

apply (erule exE)
apply (rule exI)
apply (rule trans[OF arg_cong[OF append_Cons]])
apply (rule trans[OF rv2_Cons])
apply (erule trans[OF arg_cong[OF sum.case_cong_weak]])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply assumption
done

```

```

lemmas rv_last' = spec[OF spec[OF spec[OF rv_last]]]
lemmas rv1_Inl_last = conjunct1[OF conjunct1[OF rv_last']]
lemmas rv1_Inr_last = conjunct2[OF conjunct1[OF rv_last']]
lemmas rv2_Inl_last = conjunct1[OF conjunct2[OF rv_last']]
lemmas rv2_Inr_last = conjunct2[OF conjunct2[OF rv_last']]

```

lemma Fset_Lev:

```

 $\forall kl\ b1\ b2\ b1'\ b2'\ b1''\ b2''.$ 
(kl  $\in$  Lev1 s1 s2 n b1  $\longrightarrow$ 
  ((rv1 s1 s2 kl b1 = Inl b1'  $\longrightarrow$ 
    (b1''  $\in$  F1set2 (s1 b1')  $\longrightarrow$ 
      kl @ [Inl (tobd_F12 s1 b1' b1'')]  $\in$  Lev1 s1 s2 (Suc n) b1)  $\wedge$ 
      (b2''  $\in$  F1set3 (s1 b1')  $\longrightarrow$ 
        kl @ [Inr (tobd_F13 s1 b1' b2'')]  $\in$  Lev1 s1 s2 (Suc n) b1))  $\wedge$ 
    (rv1 s1 s2 kl b1 = Inr b2'  $\longrightarrow$ 
      (b1''  $\in$  F2set2 (s2 b2')  $\longrightarrow$ 
        kl @ [Inl (tobd_F22 s2 b2' b1'')]  $\in$  Lev1 s1 s2 (Suc n) b1)  $\wedge$ 
        (b2''  $\in$  F2set3 (s2 b2')  $\longrightarrow$ 
          kl @ [Inr (tobd_F23 s2 b2' b2'')]  $\in$  Lev1 s1 s2 (Suc n) b1))))  $\wedge$ 
  (kl  $\in$  Lev2 s1 s2 n b2  $\longrightarrow$ 
    ((rv2 s1 s2 kl b2 = Inl b1'  $\longrightarrow$ 
      (b1''  $\in$  F1set2 (s1 b1')  $\longrightarrow$ 
        kl @ [Inl (tobd_F12 s1 b1' b1'')]  $\in$  Lev2 s1 s2 (Suc n) b2)  $\wedge$ 
        (b2''  $\in$  F1set3 (s1 b1')  $\longrightarrow$ 
          kl @ [Inr (tobd_F13 s1 b1' b2'')]  $\in$  Lev2 s1 s2 (Suc n) b2))  $\wedge$ 
      (rv2 s1 s2 kl b2 = Inr b2'  $\longrightarrow$ 
        (b1''  $\in$  F2set2 (s2 b2')  $\longrightarrow$ 
          kl @ [Inl (tobd_F22 s2 b2' b1'')]  $\in$  Lev2 s1 s2 (Suc n) b2)  $\wedge$ 
          (b2''  $\in$  F2set3 (s2 b2')  $\longrightarrow$ 
            kl @ [Inr (tobd_F23 s2 b2' b2'')]  $\in$  Lev2 s1 s2 (Suc n) b2))))
apply (rule nat_induct[of _ n])

```

```

apply (rule allI)+
apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_0]])
apply (erule singletonE)
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule rv1_Nil)
apply (drule Inl_inject)
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (rule conjI)
apply (rule impI)
apply (rule ssubst_mem[OF append_Nil])
apply (rule subsetD[OF equalityD2])
apply (rule Lev1_Suc)
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)

```

```

apply (erule conjI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_0)
  apply (rule singletonI)
apply (rule impI)
apply (rule ssubst_mem[OF append_Nil])
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_0)
apply (rule singletonI)

apply (rule impI)
apply (drule trans[OF sym])
  apply (rule rv1_Nil)
apply (erule notE[OF Inr_not_Inl])

apply (rule impI)
apply (drule rev_subsetD[OF _ equalityD1])
  apply (rule Lev2_0)
apply (erule singletonE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule conjI)
  apply (rule impI)
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (erule notE[OF Inl_not_Inr])

apply (rule impI)
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (drule Inr_inject)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (tactic ⟨stac @ {context} @ {thm append_Nil} 1⟩)+
apply (rule conjI)
  apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_0)
apply (rule singletonI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (rule subsetD[OF equalityD2])

```



```

apply (rule Lev2_0)
apply (rule singletonI)

```

```

apply (rule allI)+
apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @>{context} 1>)
apply (rule conjI)
apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (rule conjI)
apply assumption
apply (drule sym[OF trans[OF sym]])
  apply (rule trans[OF rv1_Cons])
  apply (rule trans[OF sum.case(1)])
  apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (rule conjI)
apply assumption
apply (drule sym[OF trans[OF sym]])
  apply (rule trans[OF rv1_Cons])
  apply (rule trans[OF sum.case(1)])
  apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 2) 1>)

```

apply (*erule mp*)
apply *assumption*

apply (*rule impI*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule subsetD*[*OF equalityD2*])
 apply (*rule Lev1_Suc*)
apply (*rule ssubst_mem*[*OF append_Cons*])
apply (*rule UnI1*)
apply (*rule CollectI*)
apply (*rule exI*)+
apply (*rule conjI*)
 apply (*rule refl*)
apply (*rule conjI*)
 apply *assumption*
apply (*drule sym*[*OF trans*[*OF sym*]])
 apply (*rule trans*[*OF rv1_Cons*])
 apply (*rule trans*[*OF sum.case(1)*])
 apply (*erule arg_cong*[*OF frombd_F12_tobd_F12*])
apply (*erule alle*)+
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 1*) 1)
apply (*drule mp*)
 apply *assumption*
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 2*) 1)
apply (*drule mp*)
 apply *assumption*
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 1*) 1)
apply (*erule mp*)
apply *assumption*

apply (*rule impI*)
apply (*rule subsetD*[*OF equalityD2*])
 apply (*rule Lev1_Suc*)
apply (*rule ssubst_mem*[*OF append_Cons*])
apply (*rule UnI1*)
apply (*rule CollectI*)
apply (*rule exI*)+
apply (*rule conjI*)
 apply (*rule refl*)
apply (*rule conjI*)
 apply *assumption*
apply (*drule sym*[*OF trans*[*OF sym*]])
 apply (*rule trans*[*OF rv1_Cons*])
 apply (*rule trans*[*OF sum.case(1)*])
 apply (*erule arg_cong*[*OF frombd_F12_tobd_F12*])
apply (*erule alle*)+
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 1*) 1)
apply (*drule mp*)
 apply *assumption*
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 2*) 1)
apply (*drule mp*)
 apply *assumption*
apply (*tactic <dtac* @*{context}* (*BNF_Util.mk_conjunctN 2 2*) 1)
apply (*erule mp*)
apply *assumption*

apply (*erule CollectE exE conjE*)+
apply (*tactic <hyp_subst_tac* @*{context}* 1)
apply (*tactic <stac* @*{context}* @*{thm rv1_Cons}* 1)
apply (*tactic <stac* @*{context}* @*{thm sum.case(2)}* 1)
apply (*tactic <stac* @*{context}* @*{thm frombd_F13_tobd_F13}* 1)
apply *assumption*

```

apply (rule conjI)
apply (rule impI)
apply (rule conjI)
  apply (rule impI)
    apply (rule subsetD[OF equalityD2])
      apply (rule Lev1_Suc)
        apply (rule ssubst_mem[OF append_Cons])
          apply (rule UnI2)
            apply (rule CollectI)
              apply (rule exI)+
                apply (rule conjI)
                  apply (rule refl)
                    apply (erule conjI)
                      apply (erule allE)+
                        apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
                          apply (drule mp)
                            apply assumption
                              apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
                                apply (drule mp)
                                  apply assumption
                                    apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
                                      apply (erule mp)
                                        apply assumption

```

```

apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
    apply (rule ssubst_mem[OF append_Cons])
      apply (rule UnI2)
        apply (rule CollectI)
          apply (rule exI)+
            apply (rule conjI)
              apply (rule refl)
                apply (erule conjI)
                  apply (erule allE)+
                    apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
                      apply (drule mp)
                        apply assumption
                          apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
                            apply (drule mp)
                              apply assumption
                                apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
                                  apply (erule mp)
                                    apply assumption

```

```

apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
    apply (rule ssubst_mem[OF append_Cons])
      apply (rule UnI2)
        apply (rule CollectI)
          apply (rule exI)+
            apply (rule conjI)
              apply (rule refl)
                apply (erule conjI)
                  apply (erule allE)+
                    apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
                      apply (drule mp)
                        apply assumption
                          apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
                            apply (drule mp)

```

```

apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule subsetD[OF equalityD2])
apply (rule Lev1_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (drule rev_subsetD[OF _ equalityD1])
apply (rule Lev2_Suc)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{context} 1>)
apply (tactic <stac @{context} @{thm rv2_Cons} 1>)
apply (tactic <stac @{context} @{thm sum.case(1)} 1>)
apply (tactic <stac @{context} @{thm frombd_F22_tobd_F22} 1>)
apply assumption
apply (rule conjI)
apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

apply (rule impI)

```

```

apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply assumption

```

```

apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @>{context} 1>)
apply (tactic <stac @>{context} @>{thm rv2_Cons} 1>)
apply (tactic <stac @>{context} @>{thm sum.case(2)} 1>)
apply (tactic <stac @>{context} @>{thm frombd_F23_tobd_F23} 1>)
apply assumption
apply (rule conjI)
apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @>{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)

```

```

apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply assumption

```

```

apply (rule impI)
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule ssubst_mem[OF append_Cons])
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply assumption
done

```

```

lemmas Fset_Lev' = spec[OF spec[OF spec[OF spec[OF spec[OF spec[OF spec[OF Fset_Lev]]]]]]]
lemmas F1set2_Lev1 = mp[OF conjunct1[OF mp[OF conjunct1[OF mp[OF conjunct1[OF Fset_Lev]]]]]]
lemmas F1set2_Lev2 = mp[OF conjunct1[OF mp[OF conjunct1[OF mp[OF conjunct2[OF Fset_Lev]]]]]]
lemmas F2set2_Lev1 = mp[OF conjunct1[OF mp[OF conjunct2[OF mp[OF conjunct1[OF Fset_Lev]]]]]]
lemmas F2set2_Lev2 = mp[OF conjunct1[OF mp[OF conjunct2[OF mp[OF conjunct2[OF Fset_Lev]]]]]]
lemmas F1set3_Lev1 = mp[OF conjunct2[OF mp[OF conjunct1[OF mp[OF conjunct1[OF Fset_Lev]]]]]]
lemmas F1set3_Lev2 = mp[OF conjunct2[OF mp[OF conjunct1[OF mp[OF conjunct2[OF Fset_Lev]]]]]]
lemmas F2set3_Lev1 = mp[OF conjunct2[OF mp[OF conjunct2[OF mp[OF conjunct1[OF Fset_Lev]]]]]]
lemmas F2set3_Lev2 = mp[OF conjunct2[OF mp[OF conjunct2[OF mp[OF conjunct2[OF Fset_Lev]]]]]]

```

lemma Fset_image_Lev:

```

 $\forall kl\ k\ b1\ b2\ b1'\ b2'.$ 
  (kl  $\in$  Lev1 s1 s2 n b1  $\longrightarrow$ 
    (kl @ [Inl k]  $\in$  Lev1 s1 s2 (Suc n) b1  $\longrightarrow$ 
      (rv1 s1 s2 kl b1 = Inl b1'  $\longrightarrow$  k  $\in$  tobd_F12 s1 b1' ' F1set2 (s1 b1'))  $\wedge$ 
      (rv1 s1 s2 kl b1 = Inr b2'  $\longrightarrow$  k  $\in$  tobd_F22 s2 b2' ' F2set2 (s2 b2'))))  $\wedge$ 
    (kl @ [Inr k]  $\in$  Lev1 s1 s2 (Suc n) b1  $\longrightarrow$ 
      (rv1 s1 s2 kl b1 = Inl b1'  $\longrightarrow$  k  $\in$  tobd_F13 s1 b1' ' F1set3 (s1 b1'))  $\wedge$ 
      (rv1 s1 s2 kl b1 = Inr b2'  $\longrightarrow$  k  $\in$  tobd_F23 s2 b2' ' F2set3 (s2 b2'))))  $\wedge$ 
    (kl  $\in$  Lev2 s1 s2 n b2  $\longrightarrow$ 
      (kl @ [Inl k]  $\in$  Lev2 s1 s2 (Suc n) b2  $\longrightarrow$ 
        (rv2 s1 s2 kl b2 = Inl b1'  $\longrightarrow$  k  $\in$  tobd_F12 s1 b1' ' F1set2 (s1 b1'))  $\wedge$ 
        (rv2 s1 s2 kl b2 = Inr b2'  $\longrightarrow$  k  $\in$  tobd_F22 s2 b2' ' F2set2 (s2 b2'))))  $\wedge$ 
      (kl @ [Inr k]  $\in$  Lev2 s1 s2 (Suc n) b2  $\longrightarrow$ 
        (rv2 s1 s2 kl b2 = Inl b1'  $\longrightarrow$  k  $\in$  tobd_F13 s1 b1' ' F1set3 (s1 b1'))  $\wedge$ 
        (rv2 s1 s2 kl b2 = Inr b2'  $\longrightarrow$  k  $\in$  tobd_F23 s2 b2' ' F2set3 (s2 b2'))))
  )
apply (rule nat_induct[of _ n])

```

```

apply (rule allI)+

```

```

apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_0]])
apply (erule singletonE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule conjI)
apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule rv1_Nil)
apply (drule ssubst_mem[OF sym[OF append_Nil]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (drule Inl_inject)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule imageI)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])
apply (rule impI)
apply (drule trans[OF sym])
apply (rule rv1_Nil)
apply (erule notE[OF Inr_not_Inl])

```

```

apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Nil]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (drule trans[OF sym])
apply (rule rv1_Nil)
apply (drule Inl_inject)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule imageI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule rv1_Nil)
apply (erule notE[OF Inr_not_Inl])

```

```

apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev2_0]])
apply (erule singletonE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule conjI)
apply (rule impI)
apply (rule conjI)

```



```

apply (rule impI)
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (erule notE[OF Inl_not_Inr])
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Nil]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (drule Inr_inject)
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule imageI)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

```

```

apply (rule impI)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (erule notE[OF Inl_not_Inr])
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Nil]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (drule trans[OF sym])
  apply (rule rv2_Nil)
apply (drule Inr_inject)
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule imageI)

```

```

apply (rule allI)+
apply (rule conjI)
apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+

```

```

apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <dtac @ {context}
(Thm.permute_premis 0 2 (@ {thm tobd_F12_inj} RS iffD1)) 1)
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @ {context} 1)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 1) 1)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context} (BNF_Util.mk_conjunctN 2 2) 1)
apply (erule mp)
apply (erule sym)

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <dtac @ {context}
(Thm.permute_premis 0 2 @ {thm tobd_F12_inj[THEN iffD1]}) 1)
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @ {context} 1)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])

```

```

apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F12_tobd_F12])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{\context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <dtac @{\context}
(Thm.permute_premis 0 2 @{\thm tobd_F13_inj[THEN iffD1]}) 1>)
apply assumption
apply assumption
apply (tactic <hyp_subst_tac @{\context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F13_tobd_F13])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)

```

```

apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F13_tobd_F13])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev1_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <dtac @{\context}
(Thm.permute_premis 0 2 @{\thm tobd_F13_inj[THEN iffD1]}) 1>)
apply assumption
apply assumption
apply (tactic <hyp_subst_tac @{\context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F13_tobd_F13])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv1_Cons])

```

```

apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F13_tobd_F13])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)

```

```

apply (rule impI)
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{\context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <dtac @{\context}
(Thm.permute_premis 0 2 @{\thm_tobd_F22_inj[THEN iffD1]}) 1>)
apply assumption
apply assumption
apply (tactic <hyp_subst_tac @{\context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F22_tobd_F22])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

```

```

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F22_tobd_F22])
apply (erule allE)+
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption
apply (tactic <dtac @{\context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
apply assumption

```

```

apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

apply (rule impI)
apply (drule subst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <dtac @{context}
(Thm.permute_premis 0 2 @{thm tobd_F22_inj[THEN iffD1]}) 1>)
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @{context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
  apply (rule trans[OF rv2_Cons])
  apply (rule trans[OF arg_cong[OF sum.case(1)]])
  apply (erule arg_cong[OF frombd_F22_tobd_F22])
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
  apply (rule trans[OF rv2_Cons])
  apply (rule trans[OF arg_cong[OF sum.case(1)]])
  apply (erule arg_cong[OF frombd_F22_tobd_F22])
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

apply (erule CollectE exE conjE)+
apply (tactic <hyp_subst_tac @{context} 1>)
apply (rule conjI)

```

```

apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <dtac @ {context}>)
(Thm.permute_premis 0 2 @ {thm tobd_F23_inj[THEN iffD1]}) 1 >
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @ {context}> 1 >)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
  apply (rule trans[OF rv2_Cons])
  apply (rule trans[OF arg_cong[OF sum.case(2)]])
  apply (erule arg_cong[OF frombd_F23_tobd_F23])
apply (erule allE)+
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 2) 1 >)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 1) 1 >)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 1) 1 >)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F23_tobd_F23])
apply (erule allE)+
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 2) 1 >)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 1) 1 >)
apply (drule mp)
  apply assumption
apply (tactic <dtac @ {context}> (BNF_Util.mk_conjunctN 2 2) 1 >)
apply (erule mp)
apply (erule sym)

apply (rule impI)
apply (drule ssubst_mem[OF sym[OF append_Cons]])
apply (drule subsetD[OF equalityD1[OF Lev2_Suc]])
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])

apply (erule CollectE exE conjE)+
apply (drule list.inject[THEN iffD1])
apply (erule conjE)

```

```

apply (drule Inr_inject)
apply (tactic <dtac @{context}
(Thm.permute_premis 0 2 @{thm tobd_F23_inj[THEN iffD1]}) 1>)
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @{context} 1>)
apply (rule conjI)
apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F23_tobd_F23])
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (erule sym)

```

```

apply (rule impI)
apply (drule trans[OF sym])
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F23_tobd_F23])
apply (erule allE)+
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (drule mp)
  apply assumption
apply (tactic <dtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (erule sym)
done

```

```

lemmas Fset_image_Lev' =
  spec[OF spec[OF spec[OF spec[OF spec[OF spec[OF Fset_image_Lev]]]]]]]
lemmas F1set2_image_Lev1 =
  mp[OF conjunct1[OF mp[OF conjunct1[OF mp[OF conjunct1[OF Fset_image_Lev']]]]]]
lemmas F1set2_image_Lev2 =
  mp[OF conjunct1[OF mp[OF conjunct1[OF mp[OF conjunct2[OF Fset_image_Lev']]]]]]
lemmas F1set3_image_Lev1 =
  mp[OF conjunct1[OF mp[OF conjunct2[OF mp[OF conjunct1[OF Fset_image_Lev']]]]]]
lemmas F1set3_image_Lev2 =
  mp[OF conjunct1[OF mp[OF conjunct2[OF mp[OF conjunct2[OF Fset_image_Lev']]]]]]
lemmas F2set2_image_Lev1 =
  mp[OF conjunct2[OF mp[OF conjunct1[OF mp[OF conjunct1[OF Fset_image_Lev']]]]]]
lemmas F2set2_image_Lev2 =
  mp[OF conjunct2[OF mp[OF conjunct1[OF mp[OF conjunct2[OF Fset_image_Lev']]]]]]
lemmas F2set3_image_Lev1 =
  mp[OF conjunct2[OF mp[OF conjunct2[OF mp[OF conjunct1[OF Fset_image_Lev']]]]]]
lemmas F2set3_image_Lev2 =
  mp[OF conjunct2[OF mp[OF conjunct2[OF mp[OF conjunct2[OF Fset_image_Lev']]]]]]

```

```

lemma mor_beh:
  mor UNIV UNIV s1 s2 carT1 carT2 strT1 strT2 (beh1 s1 s2) (beh2 s1 s2)
apply (rule mor_cong)
  apply (rule ext[OF beh1_def])

```



```

apply (rule ext[OF beh2_def])
apply (tactic ⟨rtac @{{context}} (@{{thm mor_def}} RS iffD2) 1⟩)
apply (rule conjI)
apply (rule conjI)
apply (rule ballI)
apply (rule subsetD[OF equalityD2[OF carT1_def]])
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (rule conjI)
apply (rule conjI)
apply (rule UN_I)
apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule Lev1_0)
apply (rule singletonI)

apply (rule ballI)
apply (erule UN_E)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule rev_mp[OF rv1_Inl_last_impI])
apply (erule exE)
apply (rule iffD2[OF isNode1_def])
apply (rule exI)
apply (rule conjI)
apply (erule trans[OF sum.case_cong_weak])
apply (rule sum.case(1))

apply (rule conjI)
apply (rule trans[OF F1.set_map(2)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (erule F1set2_Lev1)
apply assumption
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F1set2_image_Lev1)
apply assumption
apply (drule length_Lev1)
apply (tactic ⟨hyp_subst_tac @{{context}} 1⟩)
apply (drule length_Lev1')
apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]])
apply assumption

apply (rule trans[OF F1.set_map(3)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (erule F1set3_Lev1)

```

```

apply assumption
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F1set3_image_Lev1)
  apply assumption
  apply (drule length_Lev1)
  apply (tactic <hyp_subst_tac @{context} 1>)
  apply (drule length_Lev1 ^)
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

apply (rule ballI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule rev_mp[OF rv1_Inr_last_impI])
apply (erule exE)
apply (rule iffD2[OF isNode2_def])
apply (erule exI)
apply (rule conjI)
apply (erule trans[OF sum.case_cong_weak])
apply (rule sum.case(2))

apply (rule conjI)
apply (rule trans[OF F2.set_map(2)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
  apply (rule UN_I[OF UNIV_I])
  apply (erule F2set2_Lev1)
  apply assumption
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F2set2_image_Lev1)
  apply assumption
  apply (drule length_Lev1)
  apply (tactic <hyp_subst_tac @{context} 1>)
  apply (drule length_Lev1 ^)
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

apply (rule trans[OF F2.set_map(3)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (erule F2set3_Lev1)
  apply assumption
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)

```

```

apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F2set3_image_Lev1)
  apply assumption
  apply (drule length_Lev1)
  apply (tactic <hyp_subst_tac @ {context} 1>)
  apply (drule length_Lev1')
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

```

```

apply (rule iffD2[OF isNode1_def])
apply (rule exI)
apply (rule conjI)
  apply (rule trans[OF sum.case_cong_weak])
    apply (rule rv1_Nil)
  apply (rule sum.case(1))

```

```

apply (rule conjI)
apply (rule trans[OF F1.set_map(2)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (rule F1set2_Lev1)
  apply (rule subsetD[OF equalityD2])
    apply (rule Lev1_0)
    apply (rule singletonI)
    apply (rule rv1_Nil)
  apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule F1set2_image_Lev1)
  apply (rule subsetD[OF equalityD2[OF Lev1_0]])
  apply (rule singletonI)
apply (drule length_Lev1')
apply (erule subsetD[OF equalityD1[OF arg_cong[OF
  trans[OF length_append_singleton arg_cong[of __ Suc, OF list.size(3)]]]])
apply (rule rv1_Nil)

```

```

apply (rule trans[OF F1.set_map(3)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (rule F1set3_Lev1)
  apply (rule subsetD[OF equalityD2])
    apply (rule Lev1_0)
    apply (rule singletonI)
    apply (rule rv1_Nil)
  apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule F1set3_image_Lev1)
  apply (rule subsetD[OF equalityD2[OF Lev1_0]])
  apply (rule singletonI)
apply (drule length_Lev1')
apply (erule subsetD[OF equalityD1[OF arg_cong[OF
  trans[OF length_append_singleton arg_cong[of __ Suc, OF list.size(3)]]]])
apply (rule rv1_Nil)

```

```

apply (rule ballI)
apply (rule subsetD[OF equalityD2[OF carT2_def]])
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (rule conjI)
apply (rule conjI)
apply (rule UN_I)
  apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule Lev2_0)
apply (rule singletonI)

apply (rule ballI)
apply (erule UN_E)
apply (rule conjI)
apply (rule ballI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule rev_mp[OF rv2_Inl_last_impI])
apply (erule exE)
apply (rule iffD2[OF isNode1_def])
apply (rule exI)
apply (rule conjI)
apply (erule trans[OF sum.case_cong_weak])
apply (rule sum.case(1))

apply (rule conjI)
apply (rule trans[OF F1.set_map(2)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
  apply (rule UN_I[OF UNIV_I])
  apply (erule F1set2_Lev2)
  apply assumption
  apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F1set2_image_Lev2)
  apply assumption
  apply (drule length_Lev2)
  apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
  apply (drule length_Lev2')
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]])]
apply assumption

apply (rule trans[OF F1.set_map(3)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (erule F1set3_Lev2)
  apply assumption
  apply assumption
apply (rule subsetI)

```

```

apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F1set3_image_Lev2)
  apply assumption
  apply (drule length_Lev2)
  apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
  apply (drule length_Lev2')
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

```

```

apply (rule ballI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule rev_mp[OF rv2_Inr_last_impI])
apply (erule exE)
apply (rule iffD2[OF isNode2_def])
apply (rule exI)
apply (rule conjI)
  apply (erule trans[OF sum.case_cong_weak])
  apply (rule sum.case(2))

```

```

apply (rule conjI)
apply (rule trans[OF F2.set_map(2)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
  apply (rule UN_I[OF UNIV_I])
  apply (erule F2set2_Lev2)
  apply assumption
  apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F2set2_image_Lev2)
  apply assumption
  apply (drule length_Lev2)
  apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
  apply (drule length_Lev2')
  apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

```

```

apply (rule trans[OF F2.set_map(3)])
apply (rule equalityI)
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule SuccI)
  apply (rule UN_I[OF UNIV_I])
  apply (erule F2set3_Lev2)
  apply assumption
  apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule F2set3_image_Lev2)

```

```

apply assumption
apply (drule length_Lev2)
apply (tactic  $\langle$ hyp_subst_tac  $\@$ {context} 1 $\rangle$ )
apply (drule length_Lev2^)
apply (erule subsetD[OF equalityD1[OF arg_cong[OF length_append_singleton]]])
apply assumption

```

```

apply (rule iffD2[OF isNode2_def])
apply (rule exI)
apply (rule conjI)
apply (rule trans[OF sum.case_cong_weak])
apply (rule rv2_Nil)
apply (rule sum.case(2))

```

```

apply (rule conjI)
apply (rule trans[OF F2.set_map(2)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (rule F2set2_Lev2)
apply (rule subsetD[OF equalityD2[OF Lev2_0]])
apply (rule singletonI)
apply (rule rv2_Nil)
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule F2set2_image_Lev2)
apply (rule subsetD[OF equalityD2[OF Lev2_0]])
apply (rule singletonI)
apply (drule length_Lev2^)
apply (erule subsetD[OF equalityD1[OF arg_cong[OF
trans[OF length_append_singleton arg_cong[of  $\_ \_$  Suc, OF list.size(3)]]]]])
apply (rule rv2_Nil)

```

```

apply (rule trans[OF F2.set_map(3)])
apply (rule equalityI)
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule SuccI)
apply (rule UN_I[OF UNIV_I])
apply (rule F2set3_Lev2)
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_0)
apply (rule singletonI)
apply (rule rv2_Nil)
apply assumption
apply (rule subsetI)
apply (erule CollectE SuccD[elim_format] UN_E)+
apply (rule F2set3_image_Lev2)
apply (rule subsetD[OF equalityD2[OF Lev2_0]])
apply (rule singletonI)
apply (drule length_Lev2^)
apply (erule subsetD[OF equalityD1[OF arg_cong[OF
trans[OF length_append_singleton arg_cong[of  $\_ \_$  Suc, OF list.size(3)]]]]])
apply (rule rv2_Nil)

```

```

apply (rule conjI)
apply (rule ballI)
apply (rule sym)

```

```

apply (rule trans)
apply (rule trans[OF fun_cong[OF strT1_def] prod.case])
apply (tactic ‹CONVERSION (Conv.top_conv
      (K (Conv.try_conv (Conv.rewr_conv (@{thm rv1_Nil} RS eq_reflection)))) @>{context} 1›)
apply (rule trans[OF sum.case_cong_weak])
apply (rule sum.case(1))
apply (rule trans[OF sum.case(1)])
apply (rule trans[OF F1map_comp_id])
apply (rule F1.map_cong0[OF refl])
apply (rule trans)
  apply (rule o_apply)
apply (rule iffD2)
apply (rule prod.inject)
apply (rule conjI)
apply (rule trans)
  apply (rule Shift_def)

```

```

apply (rule equalityI)
apply (rule subsetI)
apply (erule thin_rl)
apply (erule CollectE UN_E)+
apply (erule length_Lev1)
apply (erule asm_rl)
apply (erule thin_rl)
apply (erule rev_subsetD[OF equalityD1])
  apply (rule trans[OF arg_cong[OF length_Cons]])
  apply (rule Lev1_Suc)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic ‹dtac @>{context} @>{thm list.inject[THEN iffD1]} 1›)
apply (erule conjE)
apply (erule Inl_inject)
apply (tactic ‹dtac @>{context}
(Thm.permute_prem0 2 @>{thm tobd_F12_inj[THEN iffD1]}) 1›)
  apply assumption
  apply assumption
  apply (tactic ‹hyp_subst_tac @>{context} 1›)
  apply (erule UN_I[OF UNIV_I])
apply (erule CollectE exE conjE)+
apply (tactic ‹dtac @>{context} @>{thm list.inject[THEN iffD1]} 1›)
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

```

```

apply (rule UN_least)
apply (rule subsetI)
apply (rule CollectI)
apply (rule UN_I[OF UNIV_I])
apply (rule subsetD[OF equalityD2])
  apply (rule Lev1_Suc)
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
  apply (rule refl)
apply (erule conjI)
apply assumption

```

```

apply (rule trans)
  apply (rule shift_def)
apply (rule iffD2)
apply (rule fun_eq_iff)

```

```

apply (rule allI)
apply (tactic <CONVERSION (Conv.top_conv
  (K (Conv.try_conv (Conv.reur_conv (@{thm rv1_Cons} RS eq_reflection)))) @context}) 1>)
apply (rule sum.case_cong_weak)
apply (rule trans[OF sum.case(1)])
apply (drule frombd_F12_tobd_F12)
apply (erule arg_cong)

apply (rule trans)
apply (rule o_apply)
apply (rule iffD2)
apply (rule prod.inject)
apply (rule conjI)
apply (rule trans)
apply (rule Shift_def)

apply (rule equalityI)
apply (rule subsetI)
apply (erule thin_rl)
apply (erule CollectE UN_E)+
apply (drule length_Lev1 ^)
apply (drule asm_rl)
apply (erule thin_rl)
apply (drule rev_subsetD[OF equalityD1])
apply (rule trans[OF arg_cong[OF length_Cons]])
apply (rule Lev1_Suc)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <dtac @context @thm list.inject[THEN iffD1] 1>)
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])
apply (erule CollectE exE conjE)+
apply (tactic <dtac @context @thm list.inject[THEN iffD1] 1>)
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <dtac @context
  (Thm.permute_premis 0 2 @thm tobd_F13_inj[THEN iffD1]) 1>)
apply assumption
apply assumption
apply (tactic <hyp_subst_tac @context 1>)
apply (erule UN_I[OF UNIV_I])

apply (rule UN_least)
apply (rule subsetI)
apply (rule CollectI)
apply (rule UN_I[OF UNIV_I])
apply (rule subsetD[OF equalityD2])
apply (rule Lev1_Suc)
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply assumption

apply (rule trans)
apply (rule shift_def)
apply (rule iffD2)
apply (rule fun_eq_iff)
apply (rule allI)

```



```

apply (rule sum.case_cong_weak)
apply (rule trans[OF rv1_Cons])
apply (rule trans[OF sum.case(2)])
apply (erule arg_cong[OF frombd_F13_tobd_F13])

```

```

apply (rule ballI)
apply (rule sym)
apply (rule trans)
  apply (rule trans[OF fun_cong[OF strT2_def] prod.case])
apply (rule trans[OF sum.case_cong_weak[OF trans[OF sum.case_cong_weak]]])
  apply (rule rv2_Nil)
  apply (rule sum.case(2))
apply (rule trans[OF sum.case(2)])
apply (rule trans[OF F2map_comp_id])
apply (rule F2.map_cong0[OF refl])
apply (rule trans)
  apply (rule o_apply)
apply (rule iffD2)
  apply (rule prod.inject)
apply (rule conjI)
apply (rule trans)
  apply (rule Shift_def)

```

```

apply (rule equalityI)
apply (rule subsetI)
apply (erule thin_rl)
apply (erule CollectE UN_E)+
apply (drule length_Lev2')
apply (drule asm_rl)
apply (erule thin_rl)
apply (drule rev_subsetD[OF _ equalityD1])
apply (rule trans[OF arg_cong[OF length_Cons]])
apply (rule Lev2_Suc)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <dtac @ {context} @ {thm list.inject [THEN iffD1]} 1>)
apply (erule conjE)
apply (drule Inl_inject)
apply (tactic <dtac @ {context}
(Thm.permute_premis 0 2 @ {thm tobdd_F22_inj [THEN iffD1]}) 1>)
  apply assumption
  apply assumption
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (erule UN_I[OF UNIV_I])
apply (erule CollectE exE conjE)+
apply (tactic <dtac @ {context} @ {thm list.inject [THEN iffD1]} 1>)
apply (erule conjE)
apply (erule notE[OF Inl_not_Inr])

```

```

apply (rule UN_least)
apply (rule subsetI)
apply (rule CollectI)
apply (rule UN_I[OF UNIV_I])
apply (rule subsetD[OF equalityD2])
  apply (rule Lev2_Suc)
apply (rule UnI1)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)

```

```

apply (erule conjI)
apply assumption

```

```

apply (rule trans)
apply (rule shift_def)
apply (rule iffD2)
apply (rule fun_eq_iff)
apply (rule allI)
apply (rule sum.case_cong_weak)
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(1)]])
apply (erule arg_cong[OF frombd_F22_tobd_F22])

```

```

apply (rule trans)
apply (rule o_apply)
apply (rule iffD2)
apply (rule prod.inject)
apply (rule conjI)
apply (rule trans)
apply (rule Shift_def)

```

```

apply (rule equalityI)
apply (rule subsetI)
apply (erule thin_rl)
apply (erule CollectE UN_E)+
apply (drule length_Lev2')
apply (drule asm_rl)
apply (erule thin_rl)
apply (drule rev_subsetD[OF equalityD1])
apply (rule trans[OF arg_cong[OF length_Cons]])
apply (rule Lev2_Suc)
apply (erule UnE)
apply (erule CollectE exE conjE)+
apply (tactic <dtac @{context} @{thm list.inject[THEN iffD1]} 1>)
apply (erule conjE)
apply (erule notE[OF Inr_not_Inl])
apply (erule CollectE exE conjE)+
apply (tactic <dtac @{context} @{thm list.inject[THEN iffD1]} 1>)
apply (erule conjE)
apply (drule Inr_inject)
apply (tactic <dtac @{context}
(Thm.permute_premis 0 2 @{thm tobd_F23_inj[THEN iffD1]}) 1>)
apply assumption
apply assumption
apply (tactic <hyp_subst_tac @{context} 1>)
apply (erule UN_I[OF UNIV_I])

```

```

apply (rule UN_least)
apply (rule subsetI)
apply (rule CollectI)
apply (rule UN_I[OF UNIV_I])
apply (rule subsetD[OF equalityD2])
apply (rule Lev2_Suc)
apply (rule UnI2)
apply (rule CollectI)
apply (rule exI)+
apply (rule conjI)
apply (rule refl)
apply (erule conjI)
apply assumption

```

```

apply (rule trans)
apply (rule shift_def)
apply (rule iffD2)
apply (rule fun_eq_iff)
apply (rule allI)

apply (rule sum.case_cong_weak)
apply (rule trans[OF rv2_Cons])
apply (rule trans[OF arg_cong[OF sum.case(2)]])
apply (erule arg_cong[OF frombd_F23_tobd_F23])
done

```

2.6 Quotient Coalgebra

```

abbreviation car_final1 where
  car_final1  $\equiv$  carT1 // (lsbis1 carT1 carT2 strT1 strT2)
abbreviation car_final2 where
  car_final2  $\equiv$  carT2 // (lsbis2 carT1 carT2 strT1 strT2)
abbreviation str_final1 where
  str_final1  $\equiv$  univ (F1map id
    (Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2))
    (Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2)) o strT1)
abbreviation str_final2 where
  str_final2  $\equiv$  univ (F2map id
    (Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2))
    (Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2)) o strT2)

```

```

lemma congruent_strQ1: congruent (lsbis1 carT1 carT2 strT1 strT2 :: 'a carrier rel)
  (F1map id (Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2 :: 'a carrier rel))
    (Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2 :: 'a carrier rel)) o strT1)
apply (rule congruentI)
apply (drule lsbisE1)
apply (erule bexE conjE CollectE)+
apply (rule trans[OF o_apply])
apply (erule trans[OF arg_cong[OF sym]])
apply (rule trans[OF F1map_comp_id])
apply (rule trans[OF F1.map_cong0])
  apply (rule refl)
  apply (rule equiv_proj)
  apply (rule equiv_lsbis1)
  apply (rule coalg_T)
  apply (erule subsetD)
  apply assumption
apply (rule equiv_proj)
apply (rule equiv_lsbis2)
apply (rule coalg_T)
apply (erule subsetD)
apply assumption
apply (rule sym)
apply (rule trans[OF o_apply])
apply (erule trans[OF arg_cong[OF sym]])
apply (rule F1map_comp_id)
done

```

```

lemma congruent_strQ2: congruent (lsbis2 carT1 carT2 strT1 strT2 :: 'a carrier rel)
  (F2map id (Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2 :: 'a carrier rel))
    (Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2 :: 'a carrier rel)) o strT2)
apply (rule congruentI)
apply (drule lsbisE2)
apply (erule bexE conjE CollectE)+
apply (rule trans[OF o_apply])
apply (erule trans[OF arg_cong[OF sym]])
apply (rule trans[OF F2map_comp_id])

```

```

apply (rule trans[OF F2.map_cong0])
  apply (rule refl)
  apply (rule equiv_proj)
  apply (rule equiv_lsbis1)
  apply (rule coalg_T)
  apply (erule subsetD)
  apply assumption
apply (rule equiv_proj)
  apply (rule equiv_lsbis2)
  apply (rule coalg_T)
  apply (erule subsetD)
  apply assumption
apply (rule sym)
apply (rule trans[OF o_apply])
apply (erule trans[OF arg_cong[OF sym]])
apply (rule F2map_comp_id)
done

```

lemma coalg_final:

```

coalg car_final1 car_final2 str_final1 str_final2
apply (tactic <rtac @_{context} (@{thm coalg_def} RS iffD2) 1>)
apply (rule conjI)
  apply (rule univ_preserves)
    apply (rule equiv_lsbis1)
    apply (rule coalg_T)
  apply (rule congruent_strQ1)
apply (rule ballI)
apply (rule ssubst_mem)
  apply (rule o_apply)
apply (rule CollectI)
apply (rule conjI)
  apply (rule subset_UNIV)
apply (rule conjI)
  apply (rule ord_eq_le_trans[OF F1.set_map(2)])
  apply (rule image_subsetI)
  apply (rule iffD2)
    apply (rule proj_in_iff)
    apply (rule equiv_lsbis1[OF coalg_T])
  apply (erule rev_subsetD)
  apply (erule coalg_F1set2[OF coalg_T])
apply (rule ord_eq_le_trans[OF F1.set_map(3)])
apply (rule image_subsetI)
apply (rule iffD2)
  apply (rule proj_in_iff)
  apply (rule equiv_lsbis2[OF coalg_T])
apply (erule rev_subsetD)
apply (erule coalg_F1set3[OF coalg_T])

apply (rule univ_preserves)
  apply (rule equiv_lsbis2)
  apply (rule coalg_T)
apply (rule congruent_strQ2)
apply (rule ballI)
apply (tactic <stac @_{context} @_{thm o_apply} 1>)
apply (rule CollectI)
apply (rule conjI)
  apply (rule subset_UNIV)
apply (rule conjI)
  apply (rule ord_eq_le_trans[OF F2.set_map(2)])
  apply (rule image_subsetI)
  apply (rule iffD2)
    apply (rule proj_in_iff)
    apply (rule equiv_lsbis1[OF coalg_T])

```

```

apply (erule rev_subsetD)
apply (erule coalg_F2set2[OF coalg_T])
apply (rule ord_eq_le_trans[OF F2.set_map(3)])
apply (rule image_subsetI)
apply (rule iffD2)
apply (rule proj_in_iff)
apply (rule equiv_lsbis2[OF coalg_T])
apply (erule rev_subsetD)
apply (erule coalg_F2set3[OF coalg_T])
done

```

lemma *mor_T_final*:

```

mor carT1 carT2 strT1 strT2 car_final1 car_final2 str_final1 str_final2
(Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2))
(Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2))
apply (tactic ⟨rtac @_{context} (@{thm mor_def} RS iffD2) 1⟩)
apply (rule conjI)
apply (rule conjI)
apply (rule ballI)
apply (rule iffD2)
apply (rule proj_in_iff)
apply (rule equiv_lsbis1[OF coalg_T])
apply assumption
apply (rule ballI)
apply (rule iffD2)
apply (rule proj_in_iff)
apply (rule equiv_lsbis2[OF coalg_T])
apply assumption

```

```

apply (rule conjI)
apply (rule ballI)
apply (rule sym)
apply (rule trans)
apply (rule univ_commute)
apply (rule equiv_lsbis1[OF coalg_T])
apply (rule congruent_strQ1)
apply assumption
apply (rule o_apply)

```

```

apply (rule ballI)
apply (rule sym)
apply (rule trans)
apply (rule univ_commute)
apply (rule equiv_lsbis2[OF coalg_T])
apply (rule congruent_strQ2)
apply assumption
apply (rule o_apply)
done

```

```

lemmas mor_final = mor_comp[OF mor_beh mor_T_final]
lemmas in_car_final1 = mor_image1'[OF mor_final UNIV_I]
lemmas in_car_final2 = mor_image2'[OF mor_final UNIV_I]

```

```

typedef (overloaded) 'a JF1 = car_final1 :: 'a carrier set set
by (rule exI) (rule in_car_final1)

```

```

typedef (overloaded) 'a JF2 = car_final2 :: 'a carrier set set
by (rule exI) (rule in_car_final2)

```

definition *dtor1* **where**

dtor1 $x = F1map\ id\ Abs_JF1\ Abs_JF2\ (str_final1\ (Rep_JF1\ x))$

definition *dtor2* **where**

dtor2 $x = F2map\ id\ Abs_JF1\ Abs_JF2\ (str_final2\ (Rep_JF2\ x))$

lemma *mor_Rep_JF*: *mor UNIV UNIV dtor1 dtor2*

car_final1 car_final2 str_final1 str_final2

Rep_JF1 Rep_JF2

unfolding *mor_def dtor1_def dtor2_def*

apply (*rule conjI*)

apply (*rule conjI*)

apply (*rule ballI*)

apply (*rule Rep_JF1*)

apply (*rule ballI*)

apply (*rule Rep_JF2*)

apply (*rule conjI*)

apply (*rule ballI*)

apply (*rule trans[OF F1map_comp_id]*)

apply (*rule F1map_congL*)

apply (*rule ballI*)

apply (*rule trans[OF o_apply]*)

apply (*rule Abs_JF1_inverse*)

apply (*erule rev_subsetD*)

apply (*rule coalg_F1set2*)

apply (*rule coalg_final*)

apply (*rule Rep_JF1*)

apply (*rule ballI*)

apply (*rule trans[OF o_apply]*)

apply (*rule Abs_JF2_inverse*)

apply (*erule rev_subsetD*)

apply (*rule coalg_F1set3*)

apply (*rule coalg_final*)

apply (*rule Rep_JF1*)

apply (*rule ballI*)

apply (*rule trans[OF F2map_comp_id]*)

apply (*rule F2map_congL*)

apply (*rule ballI*)

apply (*rule trans[OF o_apply]*)

apply (*rule Abs_JF1_inverse*)

apply (*erule rev_subsetD*)

apply (*rule coalg_F2set2*)

apply (*rule coalg_final*)

apply (*rule Rep_JF2*)

apply (*rule ballI*)

apply (*rule trans[OF o_apply]*)

apply (*rule Abs_JF2_inverse*)

apply (*erule rev_subsetD*)

apply (*rule coalg_F2set3*)

apply (*rule coalg_final*)

apply (*rule Rep_JF2*)

done

lemma *mor_Abs_JF*: *mor car_final1 car_final2 str_final1 str_final2*

UNIV UNIV dtor1 dtor2 Abs_JF1 Abs_JF2

unfolding *mor_def dtor1_def dtor2_def*

apply (*rule conjI*)

apply (*rule conjI*)

apply (*rule ballI*)

apply (*rule UNIV_I*)

apply (*rule ballI*)

apply (*rule UNIV_I*)

```

apply (rule conjI)
apply (rule ballI)
apply (erule sym[OF arg_cong[OF Abs_JF1_inverse]])
apply (rule ballI)
apply (erule sym[OF arg_cong[OF Abs_JF2_inverse]])
done

```

definition *unfold1* **where**

```

unfold1 s1 s2 x =
  Abs_JF1 ((Equiv_Relations.proj (lsbis1 carT1 carT2 strT1 strT2) o beh1 s1 s2) x)

```

definition *unfold2* **where**

```

unfold2 s1 s2 x =
  Abs_JF2 ((Equiv_Relations.proj (lsbis2 carT1 carT2 strT1 strT2) o beh2 s1 s2) x)

```

lemma *mor_unfold*:

```

mor UNIV UNIV s1 s2 UNIV UNIV dtor1 dtor2 (unfold1 s1 s2) (unfold2 s1 s2)
apply (rule iffD2)
apply (rule mor_UNIV)
apply (rule conjI)
apply (rule ext)
apply (rule sym[OF trans[OF o_apply]])
apply (rule trans[OF dtor1_def])
apply (rule trans[OF arg_cong[OF unfold1_def]])
apply (rule trans[OF arg_cong[OF Abs_JF1_inverse[OF in_car_final1]])
apply (rule trans[OF arg_cong[OF sym[OF morE1[OF mor_final UNIV_I]])])
apply (rule trans[OF F1map_comp_id])
apply (rule sym[OF trans[OF o_apply]])
apply (rule F1.map_cong0)
apply (rule refl)
apply (rule trans[OF unfold1_def])
apply (rule sym[OF o_apply])
apply (rule trans[OF unfold2_def])
apply (rule sym[OF o_apply])

```

```

apply (rule ext)
apply (rule sym[OF trans[OF o_apply]])
apply (rule trans[OF dtor2_def])
apply (rule trans[OF arg_cong[OF unfold2_def]])
apply (rule trans[OF arg_cong[OF Abs_JF2_inverse[OF in_car_final2]])
apply (rule trans[OF arg_cong[OF sym[OF morE2[OF mor_final UNIV_I]])])
apply (rule trans[OF F2map_comp_id])
apply (rule sym[OF trans[OF o_apply]])
apply (rule F2.map_cong0)
apply (rule refl)
apply (rule trans[OF unfold1_def])
apply (rule sym[OF o_apply])
apply (rule trans[OF unfold2_def])
apply (rule sym[OF o_apply])
done

```

lemmas *unfold1* = sym[OF morE1[OF mor_unfold UNIV_I]]

lemmas *unfold2* = sym[OF morE2[OF mor_unfold UNIV_I]]

lemma *JF_cind*: $\text{sbis UNIV UNIV dtor1 dtor2 R1 R2} \implies R1 \subseteq \text{Id} \wedge R2 \subseteq \text{Id}$

```

apply (rule rev_mp)
apply (tactic <forward_tac @ {context} @ {thms bis_def[THEN iffD1]} 1>)
apply (erule conjE)+
apply (rule bis_cong)
apply (rule bis_Comp)
apply (rule bis_converse)
apply (rule bis_Gr)
apply (rule tcoalg)

```

```

    apply (rule mor_Rep_JF)
    apply (rule bis_Comp)
    apply assumption
    apply (rule bis_Gr)
    apply (rule tcoalg)
    apply (rule mor_Rep_JF)
    apply (erule relImage_Gr)
    apply (erule relImage_Gr)

apply (rule impI)
apply (rule rev_mp)
apply (rule bis_cong)
  apply (rule bis_Comp)
  apply (rule bis_Gr)
  apply (rule coalg_T)
  apply (rule mor_T_final)
  apply (rule bis_Comp)
  apply (rule sbis_lsbis)
  apply (rule bis_converse)
  apply (rule bis_Gr)
  apply (rule coalg_T)
  apply (rule mor_T_final)
  apply (rule relInvImage_Gr[OF lsbis1_incl])
  apply (rule relInvImage_Gr[OF lsbis2_incl])

apply (rule impI)
apply (rule conjI)
  apply (rule subset_trans)
  apply (rule relInvImage_UNIV_relImage)
  apply (rule subset_trans)
  apply (rule relInvImage_mono)
  apply (rule subset_trans)
  apply (erule incl_lsbis1)
  apply (rule ord_eq_le_trans)
  apply (rule sym[OF relImage_relInvImage])
  apply (rule xt1(3))
  apply (rule Sigma_cong)
  apply (rule proj_image)
  apply (rule proj_image)
  apply (rule lsbis1_incl)
  apply (rule subset_trans)
  apply (rule relImage_mono)
  apply (rule incl_lsbis1)
  apply assumption
  apply (rule relImage_proj)
  apply (rule equiv_lsbis1[OF coalg_T])
  apply (rule relInvImage_Id_on)
  apply (rule Rep_JF1_inject)

apply (rule subset_trans)
  apply (rule relInvImage_UNIV_relImage)
  apply (rule subset_trans)
  apply (rule relInvImage_mono)
  apply (rule subset_trans)
  apply (erule incl_lsbis2)
  apply (rule ord_eq_le_trans)
  apply (rule sym[OF relImage_relInvImage])
  apply (rule xt1(3))
  apply (rule Sigma_cong)
  apply (rule proj_image)
  apply (rule proj_image)
  apply (rule lsbis2_incl)
  apply (rule subset_trans)

```



```

  apply (rule relImage_mono)
  apply (rule incl_lsbis2)
  apply assumption
  apply (rule relImage_proj)
  apply (rule equiv_lsbis2[OF coalg_T])
  apply (rule relInvImage_Id_on)
  apply (rule Rep_JF2_inject)
done

lemmas JF_cind1 = conjunct1[OF JF_cind]
lemmas JF_cind2 = conjunct2[OF JF_cind]

lemma unfold_unique_mor:
  mor UNIV UNIV s1 s2 UNIV UNIV dtor1 dtor2 f1 f2  $\implies$ 
  f1 = unfold1 s1 s2  $\wedge$  f2 = unfold2 s1 s2
  apply (rule conjI)
  apply (rule ext)
  apply (erule IdD[OF subsetD[OF JF_cind1[OF bis_image2[OF tcoalg _ tcoalg]]]])
  apply (rule mor_comp[OF mor_final mor_Abs_JF])
  apply (rule image2_eqI)
  apply (rule refl)
  apply (rule trans[OF arg_cong[OF unfold1_def]])
  apply (rule sym[OF o_apply])
  apply (rule UNIV_I)

  apply (rule ext)
  apply (erule IdD[OF subsetD[OF JF_cind2[OF bis_image2[OF tcoalg _ tcoalg]]]])
  apply (rule mor_comp[OF mor_final mor_Abs_JF])
  apply (rule image2_eqI)
  apply (rule refl)
  apply (rule trans[OF arg_cong[OF unfold2_def]])
  apply (rule sym[OF o_apply])
  apply (rule UNIV_I)
done

lemmas unfold_unique = unfold_unique_mor[OF iffD2[OF mor_UNIV], OF conjI]
lemmas unfold1_dtor = sym[OF conjunct1[OF unfold_unique_mor[OF mor_id]]]
lemmas unfold2_dtor = sym[OF conjunct2[OF unfold_unique_mor[OF mor_id]]]

lemmas unfold1_o_dtor1 =
  trans[OF conjunct1[OF unfold_unique_mor[OF mor_comp[OF mor_str mor_unfold]]] unfold1_dtor]
lemmas unfold2_o_dtor2 =
  trans[OF conjunct2[OF unfold_unique_mor[OF mor_comp[OF mor_str mor_unfold]]] unfold2_dtor]

definition ctor1 where ctor1 = unfold1 (F1map id dtor1 dtor2) (F2map id dtor1 dtor2)
definition ctor2 where ctor2 = unfold2 (F1map id dtor1 dtor2) (F2map id dtor1 dtor2)

lemma ctor1_o_dtor1:
  ctor1 o dtor1 = id
  unfolding ctor1_def
  apply (rule unfold1_o_dtor1)
done

lemma ctor2_o_dtor2:
  ctor2 o dtor2 = id
  unfolding ctor2_def
  apply (rule unfold2_o_dtor2)
done

lemma dtor1_o_ctor1:
  dtor1 o ctor1 = id
  unfolding ctor1_def

```

```

apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule trans[OF unfold1])
apply (rule trans[OF F1map_comp_id])
apply (rule trans[OF F1map_congL])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF unfold1_o_dtor1] id_apply])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF unfold2_o_dtor2] id_apply])
apply (rule sym[OF id_apply])
done

lemma dtor2_o_ctor2:
  dtor2 o ctor2 = id
  unfolding ctor2_def
  apply (rule ext)
  apply (rule trans[OF o_apply])
  apply (rule trans[OF unfold2])
  apply (rule trans[OF F2map_comp_id])
  apply (rule trans[OF F2map_congL])
    apply (rule ballI)
    apply (rule trans[OF fun_cong[OF unfold1_o_dtor1] id_apply])
  apply (rule ballI)
  apply (rule trans[OF fun_cong[OF unfold2_o_dtor2] id_apply])
  apply (rule sym[OF id_apply])
done

lemmas dtor1_ctor1 = pointfree_idE[OF dtor1_o_ctor1]
lemmas dtor2_ctor2 = pointfree_idE[OF dtor2_o_ctor2]
lemmas ctor1_dtor1 = pointfree_idE[OF ctor1_o_dtor1]
lemmas ctor2_dtor2 = pointfree_idE[OF ctor2_o_dtor2]

lemmas bij_dtor1 = o_bij[OF ctor1_o_dtor1 dtor1_o_ctor1]
lemmas inj_dtor1 = bij_is_inj[OF bij_dtor1]
lemmas surj_dtor1 = bij_is_surj[OF bij_dtor1]
lemmas dtor1_nchotomy = surjD[OF surj_dtor1]
lemmas dtor1_diff = inj_eq[OF inj_dtor1]
lemmas dtor1_cases = exE[OF dtor1_nchotomy]
lemmas bij_dtor2 = o_bij[OF ctor2_o_dtor2 dtor2_o_ctor2]
lemmas inj_dtor2 = bij_is_inj[OF bij_dtor2]
lemmas surj_dtor2 = bij_is_surj[OF bij_dtor2]
lemmas dtor2_nchotomy = surjD[OF surj_dtor2]
lemmas dtor2_diff = inj_eq[OF inj_dtor2]
lemmas dtor2_cases = exE[OF dtor2_nchotomy]

lemmas bij_ctor1 = o_bij[OF dtor1_o_ctor1 ctor1_o_dtor1]
lemmas inj_ctor1 = bij_is_inj[OF bij_ctor1]
lemmas surj_ctor1 = bij_is_surj[OF bij_ctor1]
lemmas ctor1_nchotomy = surjD[OF surj_ctor1]
lemmas ctor1_diff = inj_eq[OF inj_ctor1]
lemmas ctor1_cases = exE[OF ctor1_nchotomy]
lemmas bij_ctor2 = o_bij[OF dtor2_o_ctor2 ctor2_o_dtor2]
lemmas inj_ctor2 = bij_is_inj[OF bij_ctor2]
lemmas surj_ctor2 = bij_is_surj[OF bij_ctor2]
lemmas ctor2_nchotomy = surjD[OF surj_ctor2]
lemmas ctor2_diff = inj_eq[OF inj_ctor2]
lemmas ctor2_cases = exE[OF ctor2_nchotomy]

lemmas ctor1_unfold1 = iffD1[OF dtor1_diff trans[OF unfold1 sym[OF dtor1_ctor1]]]
lemmas ctor2_unfold2 = iffD1[OF dtor2_diff trans[OF unfold2 sym[OF dtor2_ctor2]]]

```

definition corec1 where corec1 s1 s2 =
 unfold1 (case_sum (F1map id Inl Inl o dtor1) s1)
 (case_sum (F2map id Inl Inl o dtor2) s2) o Inr

definition corec2 where corec2 s1 s2 =
 unfold2 (case_sum (F1map id Inl Inl o dtor1) s1)
 (case_sum (F2map id Inl Inl o dtor2) s2) o Inr

lemma dtor1_o_unfold1: dtor1 o unfold1 s1 s2 = F1map id (unfold1 s1 s2) (unfold2 s1 s2) o s1
 by (tactic <rtac @{\context} (BNF_Tactics.mk_pointfree2 @{\context} @{\thm unfold1}) 1>)

lemma dtor2_o_unfold2: dtor2 o unfold2 s1 s2 = F2map id (unfold1 s1 s2) (unfold2 s1 s2) o s2
 by (tactic <rtac @{\context} (BNF_Tactics.mk_pointfree2 @{\context} @{\thm unfold2}) 1>)

lemma corec1_Inl_sum:

unfold1 (case_sum (F1map id Inl Inl o dtor1) s1) (case_sum (F2map id Inl Inl o dtor2) s2) o Inl = id
apply (rule trans[OF conjunct1[OF unfold_unique] unfold1_dtor])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF F1.map_comp0[of id, unfolded id_o] refl]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF dtor1_o_unfold1 refl]])
apply (rule box_equals[OF _ o_assoc o_assoc])
apply (rule arg_cong2[of _ _ _ _ (o), OF refl case_sum_o_inj(1)])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF F2.map_comp0[of id, unfolded id_o] refl]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF dtor2_o_unfold2 refl]])
apply (rule box_equals[OF _ o_assoc o_assoc])
apply (rule arg_cong2[of _ _ _ _ (o), OF refl case_sum_o_inj(1)])
done

lemma corec2_Inl_sum:

unfold2 (case_sum (F1map id Inl Inl o dtor1) s1) (case_sum (F2map id Inl Inl o dtor2) s2) o Inl = id
apply (rule trans[OF conjunct2[OF unfold_unique] unfold2_dtor])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF F1.map_comp0[of id, unfolded id_o] refl]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF dtor1_o_unfold1 refl]])
apply (rule box_equals[OF _ o_assoc o_assoc])
apply (rule arg_cong2[of _ _ _ _ (o), OF refl case_sum_o_inj(1)])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF F2.map_comp0[of id, unfolded id_o] refl]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[of _ _ _ _ (o), OF dtor2_o_unfold2 refl]])
apply (rule box_equals[OF _ o_assoc o_assoc])
apply (rule arg_cong2[of _ _ _ _ (o), OF refl case_sum_o_inj(1)])
done

lemma case_sum_expand_Inr: f o Inl = g \implies case_sum g (f o Inr) = f
 by (auto split: sum.splits)

theorem corec1:

dtor1 (corec1 s1 s2 a) =
 F1map id (case_sum id (corec1 s1 s2)) (case_sum id (corec2 s1 s2)) (s1 a)
unfolding corec1_def corec2_def o_apply unfold1 sum.case
 case_sum_expand_Inr[OF corec1_Inl_sum] case_sum_expand_Inr[OF corec2_Inl_sum] ..

theorem corec2:

dtor2 (corec2 s1 s2 a) =
 F2map id (case_sum id (corec1 s1 s2)) (case_sum id (corec2 s1 s2)) (s2 a)
unfolding corec1_def corec2_def o_apply unfold2 sum.case
 case_sum_expand_Inr[OF corec1_Inl_sum] case_sum_expand_Inr[OF corec2_Inl_sum] ..

lemma corec_unique:

F1map id (case_sum id f1) (case_sum id f2) o s1 = dtor1 o f1 \implies
 F2map id (case_sum id f1) (case_sum id f2) o s2 = dtor2 o f2 \implies
 f1 = corec1 s1 s2 \wedge f2 = corec2 s1 s2
unfolding corec1_def corec2_def case_sum_expand_Inr'[OF corec1_Inl_sum] case_sum_expand_Inr'[OF corec2_Inl_sum]
apply (rule unfold_unique)

```

apply (unfold o_case_sum id_o o_id F1.map_comp0[symmetric] F2.map_comp0[symmetric]
  F1.map_id0 F2.map_id0 o_assoc case_sum_o_inj(1))
apply (erule arg_cong2[of _ _ _ _ case_sum, OF refl])
apply (erule arg_cong2[of _ _ _ _ case_sum, OF refl])
done

```

2.7 Coinduction

lemma *Frel_coind*:

```


$$\llbracket \forall a b. \text{phi1 } a \ b \longrightarrow F1rel \ (=) \ \text{phi1 } \text{phi2} \ (\text{dtor1 } a) \ (\text{dtor1 } b);$$


$$\forall a b. \text{phi2 } a \ b \longrightarrow F2rel \ (=) \ \text{phi1 } \text{phi2} \ (\text{dtor2 } a) \ (\text{dtor2 } b) \rrbracket \Longrightarrow$$

```

```


$$(\text{phi1 } a1 \ b1 \longrightarrow a1 = b1) \wedge (\text{phi2 } a2 \ b2 \longrightarrow a2 = b2)$$


```

```
apply (rule rev_mp)
```

```
apply (rule JF_cind)
```

```
apply (rule iffD2)
```

```
apply (rule bis_Frel)
```

```
apply (rule conjI)
```

```
apply (rule conjI)
```

```
apply (rule ord_le_eq_trans[OF subset_UNIV UNIV_Times_UNIV[THEN sym]])
```

```
apply (rule ord_le_eq_trans[OF subset_UNIV UNIV_Times_UNIV[THEN sym]])
```

```
apply (rule conjI)
```

```
apply (rule allI)+
```

```
apply (rule impI)
```

```
apply (erule allE)+
```

```
apply (rule predicate2D[OF eq_refl[OF F1rel_cong]])
```

```
apply (rule refl)
```

```
apply (rule in_rel_Collect_case_prod_eq[symmetric])
```

```
apply (rule in_rel_Collect_case_prod_eq[symmetric])
```

```
apply (erule mp)
```

```
apply (erule CollectE)
```

```
apply (erule case_prodD)
```

```
apply (rule allI)+
```

```
apply (rule impI)
```

```
apply (erule allE)+
```

```
apply (rule predicate2D[OF eq_refl[OF F2rel_cong]])
```

```
apply (rule refl)
```

```
apply (rule in_rel_Collect_case_prod_eq[symmetric])
```

```
apply (rule in_rel_Collect_case_prod_eq[symmetric])
```

```
apply (erule mp)
```

```
apply (erule CollectE)
```

```
apply (erule case_prodD)
```

```
apply (rule impI)
```

```
apply (erule conjE)+
```

```
apply (rule conjI)
```

```
apply (rule impI)
```

```
apply (rule IdD)
```

```
apply (erule subsetD)
```

```
apply (rule CollectI)
```

```
apply (erule case_prodI)
```

```
apply (rule impI)
```

```
apply (rule IdD)
```

```
apply (erule subsetD)
```

```
apply (rule CollectI)
```

```
apply (erule case_prodI)
```

```
done
```

2.8 The Result as an BNF

abbreviation *JF1map* **where**

$JF1map\ u \equiv unfold1\ (F1map\ u\ id\ id\ o\ dtor1)\ (F2map\ u\ id\ id\ o\ dtor2)$

abbreviation *JF2map* **where**

$JF2map\ u \equiv unfold2\ (F1map\ u\ id\ id\ o\ dtor1)\ (F2map\ u\ id\ id\ o\ dtor2)$

lemma *JF1map*: $dtor1\ o\ JF1map\ u = F1map\ u\ (JF1map\ u)\ (JF2map\ u)\ o\ dtor1$

```

apply (rule ext)
apply (rule sym[OF trans[OF o_apply]])
apply (rule sym[OF trans[OF o_apply]])
apply (rule trans[OF unfold1])
apply (rule box_equals[OF F1.map_comp _ F1.map_cong0, rotated])
  apply (rule fun_cong[OF id_o])
  apply (rule fun_cong[OF o_id])
  apply (rule fun_cong[OF o_id])
apply (rule sym[OF arg_cong[OF o_apply]])
done

```

lemma *JF2map*: $dtor2\ o\ JF2map\ u = F2map\ u\ (JF1map\ u)\ (JF2map\ u)\ o\ dtor2$

```

apply (rule ext)
apply (rule sym[OF trans[OF o_apply]])
apply (rule sym[OF trans[OF o_apply]])
apply (rule trans[OF unfold2])
apply (rule box_equals[OF F2.map_comp _ F2.map_cong0, rotated])
  apply (rule fun_cong[OF id_o])
  apply (rule fun_cong[OF o_id])
  apply (rule fun_cong[OF o_id])
apply (rule sym[OF arg_cong[OF o_apply]])
done

```

lemmas *JF1map_simps* = *o_eq_dest*[*OF JF1map*]

lemmas *JF2map_simps* = *o_eq_dest*[*OF JF2map*]

theorem *JF1map_id*: $JF1map\ id = id$

```

apply (rule trans)
apply (rule conjunct1)
apply (rule unfold_unique)
apply (rule sym[OF JF1map])
apply (rule sym[OF JF2map])
apply (rule unfold1_dtor)
done

```

theorem *JF2map_id*: $JF2map\ id = id$

```

apply (rule trans)
apply (rule conjunct2)
apply (rule unfold_unique)
apply (rule sym[OF JF1map])
apply (rule sym[OF JF2map])
apply (rule unfold2_dtor)
done

```

lemma *JFmap_unique*:

```

[[dtor1 o u = F1map f u v o dtor1; dtor2 o v = F2map f u v o dtor2]] ==>
  u = JF1map f ^& v = JF2map f
apply (rule unfold_unique)
unfolding o_assoc F1.map_comp0[symmetric] F2.map_comp0[symmetric] id_o o_id
apply (erule sym)
apply (erule sym)
done

```

theorem *JF1map_comp*: $JF1map\ (g\ o\ f) = JF1map\ g\ o\ JF1map\ f$

```

apply (rule sym)
apply (rule conjunct1)

```

```

apply (rule JFmap_unique)
apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong2[of _ _ _ (o), OF JF1map refl])
apply (rule trans[OF sym[OF o_assoc]])
apply (rule trans[OF arg_cong[OF JF1map]])
apply (rule trans[OF o_assoc])
apply (rule arg_cong2[of _ _ _ (o), OF sym[OF F1.map_comp0] refl])

```

```

apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong2[of _ _ _ (o), OF JF2map refl])
apply (rule trans[OF sym[OF o_assoc]])
apply (rule trans[OF arg_cong[OF JF2map]])
apply (rule trans[OF o_assoc])
apply (rule arg_cong2[of _ _ _ (o), OF sym[OF F2.map_comp0] refl])
done

```

theorem *JF2map_comp*: $JF2map (g \circ f) = JF2map g \circ JF2map f$

```

apply (rule sym)
apply (rule conjunct2)
apply (tactic <rtac @{context} (Thm.permute_prem 0 1 @{thm unfold_unique}) 1>)

```

```

apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong[OF sym[OF F2.map_comp0]]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[OF JF2map refl])
apply (rule trans[OF sym[OF o_assoc]])
apply (rule trans[OF arg_cong[OF JF2map]])
apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong2[OF sym[OF F2.map_comp0] refl])
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule sym)
apply (rule trans[OF o_apply])
apply (rule F2.map_comp0)
  apply (rule trans[OF o_apply])
  apply (rule id_apply)
  apply (rule trans[OF o_apply])
  apply (rule arg_cong[OF id_apply])
apply (rule trans[OF o_apply])
apply (rule arg_cong[OF id_apply])

```

```

apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong[OF sym[OF F1.map_comp0]]])
apply (rule sym[OF trans[OF o_assoc]])
apply (rule trans[OF arg_cong2[OF JF1map refl])
apply (rule trans[OF sym[OF o_assoc]])
apply (rule trans[OF arg_cong[OF JF1map]])
apply (rule trans[OF o_assoc])
apply (rule trans[OF arg_cong2[OF sym[OF F1.map_comp0] refl])
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule sym)
apply (rule trans[OF o_apply])
apply (rule F1.map_comp0)
  apply (rule trans[OF o_apply])
  apply (rule id_apply)
  apply (rule trans[OF o_apply])
  apply (rule arg_cong[OF id_apply])
apply (rule trans[OF o_apply])
apply (rule arg_cong[OF id_apply])
done

```

definition *JFcol* where

```
JFcol = rec_nat (%a. {}, %b. {})
(%n rec.
(%a. F1set1 (dtor1 a) ∪
((∪ a' ∈ F1set2 (dtor1 a). fst rec a') ∪
(∪ a' ∈ F1set3 (dtor1 a). snd rec a')),
%b. F2set1 (dtor2 b) ∪
((∪ b' ∈ F2set2 (dtor2 b). fst rec b') ∪
(∪ b' ∈ F2set3 (dtor2 b). snd rec b'))))
```

abbreviation *JF1col* where $JF1col\ n \equiv fst\ (JFcol\ n)$

abbreviation *JF2col* where $JF2col\ n \equiv snd\ (JFcol\ n)$

lemmas *JF1col_0* = fun_cong[OF fstI[OF rec_nat_0_imp[OF *JFcol_def*]]]

lemmas *JF2col_0* = fun_cong[OF sndI[OF rec_nat_0_imp[OF *JFcol_def*]]]

lemmas *JF1col_Suc* = fun_cong[OF fstI[OF rec_nat_Suc_imp[OF *JFcol_def*]]]

lemmas *JF2col_Suc* = fun_cong[OF sndI[OF rec_nat_Suc_imp[OF *JFcol_def*]]]

lemma *JFcol_minimal*:

```
[[∧ a. F1set1 (dtor1 a) ⊆ K1 a;
∧ b. F2set1 (dtor2 b) ⊆ K2 b;
∧ a a'. a' ∈ F1set2 (dtor1 a) ⇒ K1 a' ⊆ K1 a;
∧ a b'. b' ∈ F1set3 (dtor1 a) ⇒ K2 b' ⊆ K1 a;
∧ b a'. a' ∈ F2set2 (dtor2 b) ⇒ K1 a' ⊆ K2 b;
∧ b b'. b' ∈ F2set3 (dtor2 b) ⇒ K2 b' ⊆ K2 b] ⇒
∀ a b. JF1col n a ⊆ K1 a ∧ JF2col n b ⊆ K2 b
```

apply (rule nat_induct)

apply (rule allI)+

apply (rule conjI)

apply (rule ord_eq_le_trans)

apply (rule *JF1col_0*)

apply (rule empty_subsetI)

apply (rule ord_eq_le_trans)

apply (rule *JF2col_0*)

apply (rule empty_subsetI)

apply (rule allI)+

apply (rule conjI)

apply (rule ord_eq_le_trans)

apply (rule *JF1col_Suc*)

apply (rule Un_least)

apply assumption

apply (rule Un_least)

apply (rule UN_least)

apply (erule allE conjE)+

apply (rule subset_trans)

apply assumption

apply assumption

apply (rule UN_least)

apply (erule allE conjE)+

apply (rule subset_trans)

apply assumption

apply assumption

apply (rule ord_eq_le_trans)

apply (rule *JF2col_Suc*)

apply (rule Un_least)

apply assumption

apply (rule Un_least)

apply (rule UN_least)
apply (erule allE conjE)+
apply (rule subset_trans)
apply assumption
apply assumption

apply (rule UN_least)
apply (erule allE conjE)+
apply (rule subset_trans)
apply assumption
apply assumption
done

lemma JFset_minimal:

$$\llbracket \bigwedge a. F1set1 (dtor1 a) \subseteq K1 a;$$

$$\bigwedge b. F2set1 (dtor2 b) \subseteq K2 b;$$

$$\bigwedge a a'. a' \in F1set2 (dtor1 a) \implies K1 a' \subseteq K1 a;$$

$$\bigwedge a b'. b' \in F1set3 (dtor1 a) \implies K2 b' \subseteq K1 a;$$

$$\bigwedge b a'. a' \in F2set2 (dtor2 b) \implies K1 a' \subseteq K2 b;$$

$$\bigwedge b b'. b' \in F2set3 (dtor2 b) \implies K2 b' \subseteq K2 b \rrbracket \implies$$

$$(\bigcup n. JF1col n a) \subseteq K1 a \wedge (\bigcup n. JF2col n b) \subseteq K2 b$$

apply (rule conjI)
apply (rule UN_least)
apply (rule rev_mp)
apply (rule JFcol_minimal)
apply assumption
apply assumption
apply assumption
apply assumption
apply assumption
apply assumption
apply (rule impI)
apply (erule allE conjE)+
apply assumption

apply (rule UN_least)
apply (rule rev_mp)
apply (rule JFcol_minimal)
apply assumption
apply assumption
apply assumption
apply assumption
apply assumption
apply assumption
apply (rule impI)
apply (erule allE conjE)+
apply assumption
done

abbreviation JF1set **where** JF1set a $\equiv (\bigcup n. JF1col n a)$
abbreviation JF2set **where** JF2set a $\equiv (\bigcup n. JF2col n a)$

lemma F1set1_incl_JF1set:

$$F1set1 (dtor1 a) \subseteq JF1set a$$
apply (rule SUP_upper2)
apply (rule UNIV_I)
apply (rule ord_le_eq_trans)
apply (rule Un_upper1)
apply (rule sym)
apply (rule JF1col_Suc)
done

lemma F2set1_incl_JF2set:


```

F2set1 (dtor2 a) ⊆ JF2set a
apply (rule SUP_upper2)
apply (rule UNIV_I)
apply (rule ord_le_eq_trans)
apply (rule Un_upper1)
apply (rule sym)
apply (rule JF2col_Suc)
done

```

```

lemma F1set2_JF1set_incl_JF1set:
a' ∈ F1set2 (dtor1 a) ⇒ JF1set a' ⊆ JF1set a
apply (rule UN_least)
apply (rule subsetI)
apply (rule UN_I)
apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule JF1col_Suc)
apply (rule UnI2)
apply (tactic ‹rtac @{context} (BNF_Util.mk_UnIN 2 1) 1›)
apply (erule UN_I)
apply assumption
done

```

```

lemma F1set3_JF2set_incl_JF1set:
a' ∈ F1set3 (dtor1 a) ⇒ JF2set a' ⊆ JF1set a
apply (rule UN_least)
apply (rule subsetI)
apply (rule UN_I)
apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule JF1col_Suc)
apply (rule UnI2)
apply (tactic ‹rtac @{context} (BNF_Util.mk_UnIN 2 2) 1›)
apply (erule UN_I)
apply assumption
done

```

```

lemma F2set2_JF1set_incl_JF2set:
a' ∈ F2set2 (dtor2 a) ⇒ JF1set a' ⊆ JF2set a
apply (rule UN_least)
apply (rule subsetI)
apply (rule UN_I)
apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule JF2col_Suc)
apply (rule UnI2)
apply (tactic ‹rtac @{context} (BNF_Util.mk_UnIN 2 1) 1›)
apply (erule UN_I)
apply assumption
done

```

```

lemma F2set3_JF2set_incl_JF2set:
a' ∈ F2set3 (dtor2 a) ⇒ JF2set a' ⊆ JF2set a
apply (rule UN_least)
apply (rule subsetI)
apply (rule UN_I)
apply (rule UNIV_I)
apply (rule subsetD)
apply (rule equalityD2)
apply (rule JF2col_Suc)

```

```

apply (rule UnI2)
apply (tactic <rtac @{{context}} (BNF_Util.mk_UnIN 2 2) 1>)
apply (erule UN_I)
apply assumption
done

```

```

lemmas F1set1_JF1set = subsetD[OF F1set1_incl_JF1set]
lemmas F2set1_JF2set = subsetD[OF F2set1_incl_JF2set]
lemmas F1set2_JF1set_JF1set = subsetD[OF F1set2_JF1set_incl_JF1set]
lemmas F1set3_JF2set_JF1set = subsetD[OF F1set3_JF2set_incl_JF1set]
lemmas F2set2_JF1set_JF2set = subsetD[OF F2set2_JF1set_incl_JF2set]
lemmas F2set3_JF2set_JF2set = subsetD[OF F2set3_JF2set_incl_JF2set]

```

lemma JFset_le:

fixes a :: 'a JF1 **and** b :: 'a JF2

shows

$$JF1set\ a \subseteq F1set1\ (d\text{tor}1\ a) \cup (\bigcup (JF1set\ 'F1set2\ (d\text{tor}1\ a)) \cup \bigcup (JF2set\ 'F1set3\ (d\text{tor}1\ a))) \wedge$$

$$JF2set\ b \subseteq F2set1\ (d\text{tor}2\ b) \cup (\bigcup (JF1set\ 'F2set2\ (d\text{tor}2\ b)) \cup \bigcup (JF2set\ 'F2set3\ (d\text{tor}2\ b)))$$

```

apply (rule JFset_minimal)
  apply (rule Un_upper1)
  apply (rule Un_upper1)
  apply (rule subsetI)
  apply (rule UnI2)
  apply (tactic <rtac @{{context}} (BNF_Util.mk_UnIN 2 1) 1>)
  apply (erule UN_I)
  apply (erule UnE)
  apply (erule F1set1_JF1set)
  apply (erule UnE)+
  apply (erule UN_E)
  apply (erule F1set2_JF1set_JF1set)
  apply assumption
  apply (erule UN_E)
  apply (erule F1set3_JF2set_JF1set)
  apply assumption
  apply (rule subsetI)
  apply (rule UnI2)
  apply (tactic <rtac @{{context}} (BNF_Util.mk_UnIN 2 2) 1>)
  apply (erule UN_I)
  apply (erule UnE)
  apply (erule F2set1_JF2set)
  apply (erule UnE)+
  apply (erule UN_E)
  apply (erule F2set2_JF1set_JF2set)
  apply assumption
  apply (erule UN_E)
  apply (erule F2set3_JF2set_JF2set)
  apply assumption
  apply (rule subsetI)
  apply (rule UnI2)
  apply (tactic <rtac @{{context}} (BNF_Util.mk_UnIN 2 1) 1>)
  apply (erule UN_I)
  apply (erule UnE)+
  apply (erule F1set1_JF1set)
  apply (erule UnE)+
  apply (erule UN_E)
  apply (erule F1set2_JF1set_JF1set)
  apply assumption
  apply (erule UN_E)
  apply (erule F1set3_JF2set_JF1set)
  apply assumption
  apply (rule subsetI)
  apply (rule UnI2)
  apply (tactic <rtac @{{context}} (BNF_Util.mk_UnIN 2 2) 1>)

```

```

apply (erule UN_I)
apply (erule UnE)+
apply (erule F2set1_JF2set)
apply (erule UnE)+
apply (erule UN_E)
apply (erule F2set2_JF1set_JF2set)
apply assumption
apply (erule UN_E)
apply (erule F2set3_JF2set_JF2set)
apply assumption
done

```

theorem *JF1set_simps*:

```

JF1set a = F1set1 (dtor1 a) ∪
((∪ b ∈ F1set2 (dtor1 a). JF1set b) ∪
(∪ b ∈ F1set3 (dtor1 a). JF2set b))
apply (rule equalityI)
apply (rule conjunct1[OF JFset_le])
apply (rule Un_least)
apply (rule F1set1_incl_JF1set)
apply (rule Un_least)
apply (rule UN_least)
apply (erule F1set2_JF1set_incl_JF1set)
apply (rule UN_least)
apply (erule F1set3_JF2set_incl_JF1set)
done

```

theorem *JF2set_simps*:

```

JF2set a = F2set1 (dtor2 a) ∪
((∪ b ∈ F2set2 (dtor2 a). JF1set b) ∪
(∪ b ∈ F2set3 (dtor2 a). JF2set b))
apply (rule equalityI)
apply (rule conjunct2[OF JFset_le])
apply (rule Un_least)
apply (rule F2set1_incl_JF2set)
apply (rule Un_least)
apply (rule UN_least)
apply (erule F2set2_JF1set_incl_JF2set)
apply (rule UN_least)
apply (erule F2set3_JF2set_incl_JF2set)
done

```

lemma *JFcol_natural*:

```

 $\forall b1\ b2. u \text{ ' } (JF1col\ n\ b1) = JF1col\ n\ (JF1map\ u\ b1) \wedge$ 
 $u \text{ ' } (JF2col\ n\ b2) = JF2col\ n\ (JF2map\ u\ b2)$ 
apply (rule nat_induct)
apply (rule allI)+
unfolding JF1col_0 JF2col_0
apply (rule conjI)
apply (rule image_empty)
apply (rule image_empty)

apply (rule allI)+
apply (rule conjI)
apply (unfold JF1col_Suc JF1map_simps image_Un image_UN UN_simps(10)
  F1.set_map(1) F1.set_map(2) F1.set_map(3)) [1]
apply (rule arg_cong2[of _ _ _ (∪)])
apply (rule refl)
apply (rule arg_cong2[of _ _ _ (∪)])
apply (rule SUP_cong[OF refl])
apply (erule allE)+
apply (tactic ‹etac @{context} (BNF_Util.mk_conjunctN 2 1) 1›)
apply (rule SUP_cong[OF refl])

```

```

apply (erule allE)+
apply (tactic <etac @ {context} (BNF_Util.mk_conjunctN 2 2) 1>)

apply (unfold JF2col_Suc JF2map_simps image_Un image_UN UN_simps(10)
  F2.set_map(1) F2.set_map(2) F2.set_map(3)) [1]
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule refl)
apply (rule arg_cong2[of _ _ _ _ (U)])
apply (rule SUP_cong[OF refl])
apply (erule allE)+
apply (tactic <etac @ {context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (rule SUP_cong[OF refl])
apply (erule allE)+
apply (tactic <etac @ {context} (BNF_Util.mk_conjunctN 2 2) 1>)
done

theorem JF1set_natural: JF1set o (JF1map u) = image u o JF1set
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule sym)
apply (rule trans[OF o_apply])
apply (rule trans[OF image_UN])
apply (rule SUP_cong[OF refl])
apply (rule conjunct1)
apply (rule spec[OF spec[OF JFcol_natural]])
done

theorem JF2set_natural: JF2set o (JF2map u) = image u o JF2set
apply (rule ext)
apply (rule trans[OF o_apply])
apply (rule sym)
apply (rule trans[OF o_apply])
apply (rule trans[OF image_UN])
apply (rule SUP_cong[OF refl])
apply (rule conjunct2)
apply (rule spec[OF spec[OF JFcol_natural]])
done

theorem JFmap_cong0:
(( $\forall p \in JF1set a. u p = v p$ )  $\longrightarrow$   $JF1map u a = JF1map v a$ )  $\wedge$ 
(( $\forall p \in JF2set b. u p = v p$ )  $\longrightarrow$   $JF2map u b = JF2map v b$ )
apply (rule rev_mp)
apply (rule Frel_coind[of
  %b c.  $\exists a. a \in \{a. \forall p \in JF1set a. u p = v p\} \wedge b = JF1map u a \wedge c = JF1map v a$ 
  %b c.  $\exists a. a \in \{a. \forall p \in JF2set a. u p = v p\} \wedge b = JF2map u a \wedge c = JF2map v a$ ])
apply (intro allI impI iffD2[OF F1.in_rel])

apply (erule exE conjE)+
apply (tactic <hyp_subst_tac @ {context} 1>)
apply (rule exI)

apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans[OF F1.map_comp])
apply (rule sym)
apply (rule trans[OF JF1map_simps])
apply (rule F1.map_cong0)
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule sym[OF fun_cong[OF fst_conv[unfolded convol_def]]])
apply (rule sym[OF fun_cong[OF fst_conv[unfolded convol_def]]])

apply (rule trans[OF F1.map_comp])

```

```

apply (rule sym)
apply (rule trans[OF JF1map_simps])
apply (rule F1.map_cong0)
  apply (rule sym[OF trans[OF o_apply]])
  apply (rule trans[OF snd_conv])
  apply (erule CollectE)
  apply (erule bspec)
  apply (erule F1set1_JF1set)
apply (rule sym[OF fun_cong[OF snd_conv[unfolded convol_def]])]
apply (rule sym[OF fun_cong[OF snd_conv[unfolded convol_def]])]

```

```

apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(1))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule refl)

```

```

apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(2))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule ballI)
apply (erule CollectE)
apply (erule bspec)
apply (erule F1set2_JF1set_JF1set)
apply assumption
apply (rule conjI[OF refl refl])

```

```

apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(3))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule ballI)
apply (erule CollectE)
apply (erule bspec)
apply (erule F1set3_JF2set_JF1set)
apply assumption
apply (rule conjI[OF refl refl])

```

```

apply (intro allI impI iffD2[OF F2.in_rel])

```

```

apply (erule exE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule exI)

```

```

apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans[OF F2.map_comp])

```

```

apply (rule sym)
apply (rule trans[OF JF2map_simps])
apply (rule F2.map_cong0)
  apply (rule sym[OF trans[OF o_apply]])
  apply (rule fst_conv)
  apply (rule sym[OF fun_cong[OF fst_convол[unfolded_convол_def]])]
apply (rule sym[OF fun_cong[OF fst_convол[unfolded_convол_def]])]

apply (rule trans[OF F2.map_comp])
apply (rule sym)
apply (rule trans[OF JF2map_simps])
apply (rule F2.map_cong0)
  apply (rule sym[OF trans[OF o_apply]])
  apply (rule trans[OF snd_conv])
  apply (erule CollectE)
  apply (erule bspec)
  apply (erule F2set1_JF2set)
  apply (rule sym[OF fun_cong[OF snd_convол[unfolded_convол_def]])]
apply (rule sym[OF fun_cong[OF snd_convол[unfolded_convол_def]])]

apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(1))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule refl)

apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(2))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule exI)
apply (rule conjI)
  apply (rule CollectI)
  apply (rule ballI)
  apply (erule CollectE)
  apply (erule bspec)
  apply (erule F2set2_JF1set_JF2set)
apply assumption
apply (rule conjI[OF refl refl])

apply (rule ord_eq_le_trans)
apply (rule F2.set_map(3))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule ballI)
apply (erule CollectE)
apply (erule bspec)
apply (erule F2set3_JF2set_JF2set)
apply assumption
apply (rule conjI[OF refl refl])

apply (rule impI)

```

```

apply (rule conjI)
apply (rule impI)
apply (tactic <dtac @{{context}} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (erule mp)
apply (rule exI)
apply (rule conjI)
apply (erule CollectI)
apply (rule conjI)
apply (rule refl)
apply (rule refl)

```

```

apply (rule impI)
apply (tactic <dtac @{{context}} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (erule mp)
apply (rule exI)
apply (rule conjI)
apply (erule CollectI)
apply (rule conjI)
apply (rule refl)
apply (rule refl)
done

```

```

lemmas JF1map_cong0 = mp[OF conjunct1[OF JFmap_cong0]]
lemmas JF2map_cong0 = mp[OF conjunct2[OF JFmap_cong0]]

```

```

lemma JFcol_bd:  $\forall (j1 :: 'a JF1) (j2 :: 'a JF2). |JF1col\ n\ j1| \leq o\ bd\_F \wedge |JF2col\ n\ j2| \leq o\ bd\_F$ 

```

```

apply (rule nat_induct)
apply (rule allI)+
apply (rule conjI)
apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule JF1col_0)
apply (rule Card_order_empty)
apply (rule bd_F_Card_order)
apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule JF2col_0)
apply (rule Card_order_empty)
apply (rule bd_F_Card_order)

```

```

apply (rule allI)+
apply (rule conjI)
apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule JF1col_Suc)
apply (rule Un_Cinfinite_bound)
apply (rule F1set1_bd')
apply (rule Un_Cinfinite_bound)
apply (rule UNION_Cinfinite_bound)
apply (rule F1set2_bd')
apply (rule ballI)
apply (erule allE)+
apply (tactic <etac @{{context}} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (rule bd_F_Cinfinite)
apply (rule UNION_Cinfinite_bound)
apply (rule F1set3_bd')
apply (rule ballI)
apply (erule allE)+
apply (tactic <etac @{{context}} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule bd_F_Cinfinite)
apply (rule bd_F_Cinfinite)
apply (rule bd_F_Cinfinite)

```

```

apply (rule ordIso_ordLeq_trans)
apply (rule card_of_ordIso_subst)
apply (rule JF2col_Suc)
apply (rule Un_Cinfinite_bound)
  apply (rule F2set1_bd')
apply (rule Un_Cinfinite_bound)
  apply (rule UNION_Cinfinite_bound)
    apply (rule F2set2_bd')
    apply (rule ballI)
    apply (erule allE)+
    apply (tactic <etac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
  apply (rule bd_F_Cinfinite)
apply (rule UNION_Cinfinite_bound)
  apply (rule F2set3_bd')
  apply (rule ballI)
  apply (erule allE)+
  apply (tactic <etac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule bd_F_Cinfinite)
apply (rule bd_F_Cinfinite)
apply (rule bd_F_Cinfinite)
done

```

```

theorem JF1set_bd: |JF1set j| ≤o bd_F
apply (rule UNION_Cinfinite_bound)
  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_nat)
  apply (rule natLeq_ordLeq_cinfinite)
  apply (rule bd_F_Cinfinite)
apply (rule ballI)
apply (tactic <rtac @{context} (BNF_Util.mk_conjunctN 2 1) 1>)
apply (rule spec[OF spec[OF JFcol_bd]])
apply (rule bd_F_Cinfinite)
done

```

```

theorem JF2set_bd: |JF2set j| ≤o bd_F
apply (rule UNION_Cinfinite_bound)
  apply (rule ordIso_ordLeq_trans)
  apply (rule card_of_nat)
  apply (rule natLeq_ordLeq_cinfinite)
  apply (rule bd_F_Cinfinite)
apply (rule ballI)
apply (tactic <rtac @{context} (BNF_Util.mk_conjunctN 2 2) 1>)
apply (rule spec[OF spec[OF JFcol_bd]])
apply (rule bd_F_Cinfinite)
done

```

abbreviation JF2wit ≡ ctor2 wit_F2

```

theorem JF2wit: ∧x. x ∈ JF2set JF2wit ⇒ False
apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule JF2set_simps)
unfolding dtor2_ctor2
apply (erule UnE)
apply (erule F2.wit)
apply (erule UnE)
apply (erule UN_E)
apply (erule F2.wit)
apply (erule UN_E)
apply (erule F2.wit)
done

```


abbreviation $JF1wit \equiv (\%a. \text{ctor1 } (wit2_F1 \ a \ JF2wit))$

theorem $JF1wit: \bigwedge x. x \in JF1set \ (JF1wit \ a) \implies x = a$

```

apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule JF1set_simps)
unfolding dtor1_ctor1
apply (erule UnE)+
apply (erule F1.wit2)
apply (erule UnE)
apply (erule UN_E)
apply (drule F1.wit2)
apply (erule FalseE)
apply (erule UN_E)
apply (drule F1.wit2)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (drule rev_subsetD)
apply (rule equalityD1)
apply (rule JF2set_simps)
unfolding dtor2_ctor2
apply (erule UnE)+
apply (drule F2.wit)
apply (erule FalseE)
apply (erule UnE)
apply (erule UN_E)
apply (drule F2.wit)
apply (erule FalseE)
apply (erule UN_E)
apply (drule F2.wit)
apply (erule FalseE)
done

```

context

fixes $\phi1 :: 'a \Rightarrow 'a \ JF1 \Rightarrow \text{bool}$ **and** $\phi2 :: 'a \Rightarrow 'a \ JF2 \Rightarrow \text{bool}$

begin

lemmas $JFset_induct =$

```

JFset_minimal[of %b1. {a ∈ JF1set b1 . phi1 a b1} %b2. {a ∈ JF2set b2 . phi2 a b2}],
unfolded subset_Collect_iff[OF F1set1_incl_JF1set] subset_Collect_iff[OF F2set1_incl_JF2set]
subset_Collect_iff[OF subset_refl],
OF ballI ballI
subset_CollectI[OF F1set2_JF1set_incl_JF1set]
subset_CollectI[OF F1set3_JF2set_incl_JF1set]
subset_CollectI[OF F2set2_JF1set_incl_JF2set]
subset_CollectI[OF F2set3_JF2set_incl_JF2set]]

```

end

ML $\langle \text{rule_by_tactic } @\{context\} \ (ALLGOALS \ (TRY \ o \ etac \ @\{context\} \ \text{asm_rl})) \ @\{thm \ JFset_induct\} \rangle$

abbreviation $JF1in \ \text{where} \ JF1in \ B \equiv \{a. \ JF1set \ a \subseteq B\}$

abbreviation $JF2in \ \text{where} \ JF2in \ B \equiv \{a. \ JF2set \ a \subseteq B\}$

definition $JF1rel \ \text{where}$

```

JF1rel R = (BNF_Def.Grp (JF1in (Collect (case_prod R))) (JF1map fst)) ^--1 OO
  (BNF_Def.Grp (JF1in (Collect (case_prod R))) (JF1map snd))

```

definition *JF2rel* where

$$JF2rel\ R = (BNF_Def.Grp\ (JF2in\ (Collect\ (case_prod\ R)))\ (JF2map\ fst))\ \hat{-}\ -1\ OO \\ (BNF_Def.Grp\ (JF2in\ (Collect\ (case_prod\ R)))\ (JF2map\ snd))$$

lemma *in_JF1rel*:

$$JF1rel\ R\ x\ y \longleftrightarrow (\exists\ z.\ z \in JF1in\ (Collect\ (case_prod\ R)) \wedge JF1map\ fst\ z = x \wedge JF1map\ snd\ z = y)$$

by (rule predicate2_eqD[OF trans[OF JF1rel_def OO_Grp_alt]])

lemma *in_JF2rel*:

$$JF2rel\ R\ x\ y \longleftrightarrow (\exists\ z.\ z \in JF2in\ (Collect\ (case_prod\ R)) \wedge JF2map\ fst\ z = x \wedge JF2map\ snd\ z = y)$$

by (rule predicate2_eqD[OF trans[OF JF2rel_def OO_Grp_alt]])

lemma *J_rel_coind_ind*:

$$\llbracket \forall x\ y.\ R2\ x\ y \longrightarrow (f\ x\ y \in F1in\ (Collect\ (case_prod\ R1))\ (Collect\ (case_prod\ R2))\ (Collect\ (case_prod\ R3))) \wedge \\ F1map\ fst\ fst\ fst\ (f\ x\ y) = dtor1\ x \wedge \\ F1map\ snd\ snd\ snd\ (f\ x\ y) = dtor1\ y); \\ \forall x\ y.\ R3\ x\ y \longrightarrow (g\ x\ y \in F2in\ (Collect\ (case_prod\ R1))\ (Collect\ (case_prod\ R2))\ (Collect\ (case_prod\ R3))) \wedge \\ F2map\ fst\ fst\ fst\ (g\ x\ y) = dtor2\ x \wedge \\ F2map\ snd\ snd\ snd\ (g\ x\ y) = dtor2\ y \rrbracket \Longrightarrow$$

$$(\forall a \in JF1set\ z1.\ \forall x\ y.\ R2\ x\ y \wedge z1 = unfold1\ (case_prod\ f)\ (case_prod\ g)\ (x,\ y) \longrightarrow R1\ (fst\ a)\ (snd\ a)) \wedge \\ (\forall a \in JF2set\ z2.\ \forall x\ y.\ R3\ x\ y \wedge z2 = unfold2\ (case_prod\ f)\ (case_prod\ g)\ (x,\ y) \longrightarrow R1\ (fst\ a)\ (snd\ a))$$

apply (tactic <rtac @<context> (rule_by_tactic @<context> (ALLGOALS (TRY o etac @<context> asm_rl)) @<thm JFset_induct[of

$$\lambda a\ z1.\ \forall x\ y.\ R2\ x\ y \wedge z1 = unfold1\ (case_prod\ f)\ (case_prod\ g)\ (x,\ y) \longrightarrow R1\ (fst\ a)\ (snd\ a)$$

$$\lambda a\ z2.\ \forall x\ y.\ R3\ x\ y \wedge z2 = unfold2\ (case_prod\ f)\ (case_prod\ g)\ (x,\ y) \longrightarrow R1\ (fst\ a)\ (snd\ a)$$

$$z1\ z2\}])\ 1\ \rangle$$

apply (rule allI impI)+

apply (erule conjE)

apply (erule thin_rl)

apply (erule thin_rl)

apply (erule mp)

apply assumption

apply (erule CollectE conjE Collect_case_prodD[OF subsetD] rev_subsetD)+

apply hypsubst

unfolding unfold1 F1.set_map(1) prod.case_image_id id_apply

apply (rule subset_refl)

apply (rule allI impI)+

apply (erule conjE)

apply (erule thin_rl)

apply (erule thin_rl)

apply (erule mp)

apply assumption

apply (erule CollectE conjE Collect_case_prodD[OF subsetD] rev_subsetD)+

apply hypsubst

unfolding unfold2 F2.set_map(1) prod.case_image_id id_apply

apply (rule subset_refl)

apply (rule impI allI)+

apply (erule conjE)

apply (erule thin_rl)

apply (erule thin_rl)

apply (erule mp)

apply assumption

apply (erule CollectE conjE)+

apply (tactic <hyp_subst_tac @<context> 1>)

unfolding unfold1 F1.set_map(2) prod.case_image_id id_apply

apply (erule imageE)

apply (tactic <hyp_subst_tac @<context> 1>)

apply (erule allE mp)+

```

apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (rule impI allI)+
apply (erule conjE)
apply (drule spec2)
apply (erule thin_rl)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
unfolding unfold1 F1.set_map(3) prod.case image_id id_apply
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule allE mp)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (rule impI allI)+
apply (erule conjE)
apply (erule thin_rl)
apply (drule spec2)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
unfolding unfold2 F2.set_map(2) prod.case image_id id_apply
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule allE mp)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (rule impI allI)+
apply (erule conjE)
apply (erule thin_rl)
apply (drule spec2)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
unfolding unfold2 F2.set_map(3) prod.case image_id id_apply
apply (erule imageE)
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (erule allE mp)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])
done

```

lemma *J_rel_coind_coind1*:

$$\llbracket \forall x y. R2 \ x \ y \longrightarrow (f \ x \ y \in F1 \text{in } (\text{Collect } (\text{case_prod } R1)) (\text{Collect } (\text{case_prod } R2)) (\text{Collect } (\text{case_prod } R3))) \wedge$$

```

      F1map fst fst fst (f x y) = dtor1 x ∧
      F1map snd snd snd (f x y) = dtor1 y);
∀ x y. R3 x y → (g x y ∈ F2in (Collect (case_prod R1)) (Collect (case_prod R2)) (Collect (case_prod R3))) ∧
      F2map fst fst fst (g x y) = dtor2 x ∧
      F2map snd snd snd (g x y) = dtor2 y]] ==>
((∃ y. R2 x1 y ∧ x1' = JF1map fst (unfold1 (case_prod f) (case_prod g) (x1, y))) → x1' = x1) ∧
(∃ y. R3 x2 y ∧ x2' = JF2map fst (unfold2 (case_prod f) (case_prod g) (x2, y))) → x2' = x2)
apply (rule Frel_coind[of
  λx1' x1. ∃ y. R2 x1 y ∧ x1' = JF1map fst (unfold1 (case_prod f) (case_prod g) (x1, y))
  λx2' x2. ∃ y. R3 x2 y ∧ x2' = JF2map fst (unfold2 (case_prod f) (case_prod g) (x2, y))
  x1' x1
  x2' x2
])
apply (intro allI impI iffD2[OF F1.in_rel])

apply (erule exE conjE)+
apply (drule spec2)
apply (erule thin_rl)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule exI)
apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans[OF F1.map_comp])
apply (rule sym[OF trans[OF JF1map_simps]])
apply (rule trans[OF arg_cong[OF unfold1]])
apply (rule trans[OF F1.map_comp F1.map_cong0])
apply (rule trans[OF fun_cong[OF o_id]])
apply (rule sym[OF fun_cong[OF fst_diag fst]])
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule trans[OF F1.map_comp])
apply (rule trans[OF F1.map_cong0])
apply (rule fun_cong[OF snd_diag fst])
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (erule trans[OF arg_cong[OF prod.case]])

apply (unfold prod.case o_def fst_conv snd_conv) []
apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans[OF F1.set_map(1)])
apply (rule image_subsetI CollectI case_prodI)+
apply (rule refl)

apply (rule conjI)
apply (rule ord_eq_le_trans[OF F1.set_map(2)])
apply (rule image_subsetI CollectI case_prodI exI)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

apply (rule ord_eq_le_trans[OF F1.set_map(3)])
apply (rule image_subsetI CollectI case_prodI exI)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])

```

```

apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (intro allI impI iffD2[OF F2.in_rel])

```

```

apply (erule exE conjE)+
apply (erule thin_rl)
apply (drule spec2)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic <hyp_subst_tac @{context} 1 >)
apply (rule exI)
apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans[OF F2.map_comp])
apply (rule sym[OF trans[OF JF2map_simps]])
apply (rule trans[OF arg_cong[OF unfold2]])
apply (rule trans[OF F2.map_comp F2.map_cong0])
apply (rule fun_cong[OF trans[OF o_id fst_diag_fst[symmetric]]])
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule trans[OF F2.map_comp])
apply (rule trans[OF F2.map_cong0])
apply (rule fun_cong[OF snd_diag_fst])
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (erule trans[OF arg_cong[OF prod.case]])

```

```

apply (unfold prod.case o_def fst_conv snd_conv) []
apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans[OF F2.set_map(1)])
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule refl)

```

```

apply (rule conjI)
apply (rule ord_eq_le_trans[OF F2.set_map(2)])
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule exI)
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (rule ord_eq_le_trans[OF F2.set_map(3)])
apply (rule image_subsetI CollectI case_prodI exI)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])
done

```

lemma *J_rel_coind_coind2*:

$$\begin{aligned} & \llbracket \forall x y. R2 x y \longrightarrow (f x y \in F1in (Collect (case_prod R1)) (Collect (case_prod R2)) (Collect (case_prod R3))) \wedge \\ & \quad F1map fst fst fst (f x y) = dtor1 x \wedge \\ & \quad F1map snd snd snd (f x y) = dtor1 y); \\ & \forall x y. R3 x y \longrightarrow (g x y \in F2in (Collect (case_prod R1)) (Collect (case_prod R2)) (Collect (case_prod R3))) \wedge \\ & \quad F2map fst fst fst (g x y) = dtor2 x \wedge \\ & \quad F2map snd snd snd (g x y) = dtor2 y \rrbracket \implies \end{aligned}$$

$$((\exists x. R2 x y1 \wedge y1' = JF1map snd (unfold1 (case_prod f) (case_prod g) (x, y1))) \longrightarrow y1' = y1) \wedge$$

$$((\exists x. R3 x y2 \wedge y2' = JF2map snd (unfold2 (case_prod f) (case_prod g) (x, y2))) \longrightarrow y2' = y2)$$

apply (rule *Frel_coind*[of

$\lambda y1' y1. \exists x. R2 x y1 \wedge y1' = JF1map snd (unfold1 (case_prod f) (case_prod g) (x, y1))$

$\lambda y2' y2. \exists x. R3 x y2 \wedge y2' = JF2map snd (unfold2 (case_prod f) (case_prod g) (x, y2))$

$y1' y1$

$y2' y2$

])

apply (intro allI impI iffD2[OF *F1.in_rel*])

apply (erule *exE conjE*)+

apply (drule *spec2*)

apply (erule *thin_rl*)

apply (drule *mp*)

apply *assumption*

apply (erule *CollectE conjE*)+

apply (tactic <hyp_subst_tac @ {context} 1 >)

apply (rule *exI*)

apply (rule *conjI*[rotated])

apply (rule *conjI*)

apply (rule *trans*[OF *F1.map_comp*])

apply (rule *sym*[OF *trans*[OF *JF1map_simps*]])

apply (rule *trans*[OF *arg_cong*[OF *unfold1*]])

apply (rule *trans*[OF *F1.map_comp F1.map_cong0*])

apply (rule *trans*[OF *fun_cong*[OF *o_id*]])

apply (rule *sym*[OF *fun_cong*[OF *fst_diag_snd*]])

apply (rule *sym*[OF *trans*[OF *o_apply*]])

apply (rule *fst_conv*)

apply (rule *sym*[OF *trans*[OF *o_apply*]])

apply (rule *fst_conv*)

apply (rule *trans*[OF *F1.map_comp*])

apply (rule *trans*[OF *F1.map_cong0*])

apply (rule *fun_cong*[OF *snd_diag_snd*])

apply (rule *trans*[OF *o_apply*])

apply (rule *snd_conv*)

apply (rule *trans*[OF *o_apply*])

apply (rule *snd_conv*)

apply (erule *trans*[OF *arg_cong*[OF *prod.case*]])

apply (unfold *prod.case o_def fst_conv snd_conv*) []

apply (rule *CollectI*)

apply (rule *conjI*)

apply (rule *ord_eq_le_trans*[OF *F1.set_map(1)*])

apply (rule *image_subsetI CollectI case_prodI*)+

apply (rule *refl*)

apply (rule *conjI*)

apply (rule *ord_eq_le_trans*[OF *F1.set_map(2)*])

apply (rule *image_subsetI CollectI case_prodI exI*)+

apply (rule *conjI*)

apply (erule *Collect_case_prodD*[OF *subsetD*])

apply *assumption*

apply (rule *arg_cong*[OF *surjective_pairing*])

apply (rule *ord_eq_le_trans*[OF *F1.set_map(3)*])

apply (rule *image_subsetI CollectI case_prodI exI*)+

```

apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (intro allI impI iffD2[OF F2.in_rel])

```

```

apply (erule exE conjE)+
apply (erule thin_rl)
apply (drule spec2)
apply (drule mp)
apply assumption
apply (erule CollectE conjE)+
apply (tactic ⟨hyp_subst_tac @ {context} 1⟩)
apply (rule exI)
apply (rule conjI[rotated])
apply (rule conjI)
apply (rule trans[OF F2.map_comp])
apply (rule sym[OF trans[OF JF2map_simps]])
apply (rule trans[OF arg_cong[OF unfold2]])
apply (rule trans[OF F2.map_comp F2.map_cong0])
apply (rule trans[OF fun_cong[OF o_id]])
apply (rule sym[OF fun_cong[OF fst_diag_snd]])
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule sym[OF trans[OF o_apply]])
apply (rule fst_conv)
apply (rule trans[OF F2.map_comp])
apply (rule trans[OF F2.map_cong0])
apply (rule fun_cong[OF snd_diag_snd])
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (rule trans[OF o_apply])
apply (rule snd_conv)
apply (erule trans[OF arg_cong[OF prod.case]])

```

```

apply (unfold prod.case o_def fst_conv snd_conv) []
apply (rule CollectI)
apply (rule conjI)
apply (rule ord_eq_le_trans[OF F2.set_map(1)])
apply (rule image_subsetI CollectI case_prodI)+
apply (rule refl)

```

```

apply (rule conjI)
apply (rule ord_eq_le_trans[OF F2.set_map(2)])
apply (rule image_subsetI CollectI case_prodI exI)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])

```

```

apply (rule ord_eq_le_trans[OF F2.set_map(3)])
apply (rule image_subsetI CollectI case_prodI exI)+
apply (rule conjI)
apply (erule Collect_case_prodD[OF subsetD])
apply assumption
apply (rule arg_cong[OF surjective_pairing])
done

```

lemma *J_rel_coind*:

```

assumes CIH1:  $\forall x2\ y2. R2\ x2\ y2 \longrightarrow F1rel\ R1\ R2\ R3\ (dtor1\ x2)\ (dtor1\ y2)$ 

```

```

and    CIH2:  $\forall x3\ y3. R3\ x3\ y3 \longrightarrow F2rel\ R1\ R2\ R3\ (dior2\ x3)\ (dior2\ y3)$ 
shows   $R2 \leq JF1rel\ R1 \wedge R3 \leq JF2rel\ R1$ 
apply (insert CIH1 CIH2)
unfolding F1.in_rel F2.in_rel ex_simps(6)[symmetric] choice_iff
apply (erule exE)+
apply (rule conjI)

apply (rule predicate2I)
apply (rule iffD2[OF in_JF1rel])
apply (rule exI conjI)+
apply (rule CollectI)
apply (rule rev_mp[OF conjunct1[OF J_rel_coind_ind]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (rule subsetI CollectI iffD2[OF case_prod_beta])+
apply (drule bspec)
  apply assumption
apply (erule allE mp conjE)+
apply (erule conjI[OF _ refl])

apply (rule conjI)
apply (rule rev_mp[OF conjunct1[OF J_rel_coind_coind1]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (erule mp)
apply (rule exI)
apply (erule conjI[OF _ refl])

apply (rule rev_mp[OF conjunct1[OF J_rel_coind_coind2]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (erule mp)
apply (rule exI)
apply (erule conjI[OF _ refl])

apply (rule predicate2I)
apply (rule iffD2[OF in_JF2rel])
apply (rule exI conjI)+
apply (rule rev_mp[OF conjunct2[OF J_rel_coind_ind]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (rule CollectI)
apply (rule subsetI CollectI iffD2[OF case_prod_beta])+
apply (drule bspec)
  apply assumption
apply (erule allE mp conjE)+
apply (erule conjI[OF _ refl])

apply (rule conjI)

```



```

apply (rule rev_mp[OF conjunct2[OF J_rel_coind_coind1]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (erule mp)
apply (rule exI)
apply (erule conjI[OF _ refl])

```

```

apply (rule rev_mp[OF conjunct2[OF J_rel_coind_coind2]])
  apply assumption
  apply assumption
apply (erule thin_rl)
apply (erule thin_rl)
apply (rule impI)
apply (erule mp)
apply (rule exI)
apply (erule conjI[OF _ refl])
done

```

lemma *JF1rel_F1rel*: $JF1rel\ R\ a\ b \longleftrightarrow F1rel\ R\ (JF1rel\ R)\ (JF2rel\ R)\ (dtor1\ a)\ (dtor1\ b)$

```

apply (rule iffI)
  apply (drule iffD1[OF in_JF1rel])
  apply (erule exE conjE CollectE)+
  apply (rule iffD2[OF F1.in_rel])
  apply (rule exI)
  apply (rule conjI)
  apply (rule CollectI)
  apply (rule conjI)
  apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(1))
  apply (rule ord_eq_le_trans)
  apply (rule trans[OF fun_cong[OF image_id] id_apply])
  apply (erule subset_trans[OF F1set1_incl_JF1set])

apply (rule conjI)
  apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(2))
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule case_prodI)
  apply (rule iffD2[OF in_JF1rel])
  apply (rule exI)
  apply (rule conjI)
  apply (rule CollectI)
  apply (erule subset_trans[OF F1set2_JF1set_incl_JF1set])
  apply assumption
  apply (rule conjI)
  apply (rule refl)
  apply (rule refl)

apply (rule ord_eq_le_trans)
  apply (rule F1.set_map(3))
  apply (rule image_subsetI)
  apply (rule CollectI)
  apply (rule case_prodI)
  apply (rule iffD2[OF in_JF2rel])
  apply (rule exI)
  apply (rule conjI)
  apply (rule CollectI)
  apply (erule subset_trans[OF F1set3_JF2set_incl_JF1set])
  apply assumption

```

```

apply (rule conjI)
apply (rule refl)
apply (rule refl)
apply (rule conjI)

apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans)
  apply (rule o_apply)
  apply (rule fst_conv)
apply (rule trans)
  apply (rule o_apply)
  apply (rule fst_conv)
apply (rule trans)
apply (rule sym)
apply (rule JF1map_simps)
apply (rule iffD2[OF dtor1_diff])
apply assumption

apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans)
  apply (rule o_apply)
  apply (rule snd_conv)
apply (rule trans)
  apply (rule o_apply)
  apply (rule snd_conv)
apply (rule trans)
apply (rule sym)
apply (rule JF1map_simps)
apply (rule iffD2[OF dtor1_diff])
apply assumption

apply (drule iffD1[OF F1.in_rel])
apply (erule exE conjE CollectE)+
apply (rule iffD2[OF in_JF1rel])
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule ord_eq_le_trans)
apply (rule JF1set_simps)
apply (rule Un_least)
apply (rule ord_eq_le_trans)
  apply (rule box_equals)
  apply (rule F1.set_map(1))
  apply (rule arg_cong[OF sym[OF dtor1_ctor1]])
  apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption
apply (rule Un_least)
apply (rule ord_eq_le_trans)
apply (rule SUP_cong[OF _ refl])
apply (rule box_equals[OF _ _ refl])
  apply (rule F1.set_map(2))
  apply (rule arg_cong[OF sym[OF dtor1_ctor1]])
apply (rule UN_least)
apply (drule rev_subsetD)
apply (erule image_mono)

```

```

apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF1rel])
apply (drule someI_ex)
apply (erule conjE)+
apply (erule CollectE conjE)+
apply assumption

apply (rule ord_eq_le_trans)
apply (rule trans[OF arg_cong[OF dtor1_ctor1]])
apply (rule arg_cong[OF F1.set_map(3)])
apply (rule UN_least)
apply (drule rev_subsetD)
apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF2rel])
apply (drule someI_ex)
apply (erule exE conjE)+
apply (erule CollectD)

apply (rule conjI)

apply (rule iffD1[OF dtor1_diff])
apply (rule trans)
apply (rule JF1map_simps)
apply (rule box_equals)
apply (rule F1.map_comp)
apply (rule arg_cong[OF sym[OF dtor1_ctor1]])
apply (rule trans)
apply (rule F1.map_cong0)
apply (rule fun_cong[OF o_id])
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF1rel])
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF2rel])
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply assumption

apply (rule iffD1[OF dtor1_diff])
apply (rule trans)
apply (rule JF1map_simps)
apply (rule trans)
apply (rule arg_cong[OF dtor1_ctor1])

```

```

apply (rule trans)
apply (rule F1.map_comp)
apply (rule trans)
apply (rule F1.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
  apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF1rel])
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
  apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF2rel])
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply assumption
done

```

lemma *JF2rel_F2rel*: $JF2rel\ R\ a\ b \longleftrightarrow F2rel\ R\ (JF1rel\ R)\ (JF2rel\ R)\ (dior2\ a)\ (dior2\ b)$

```

apply (rule iffI)
apply (drule iffD1[OF in_JF2rel])
apply (erule exE conjE CollectE)+
apply (rule iffD2[OF F2.in_rel])
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule conjI)
  apply (rule ord_eq_le_trans)
    apply (rule F2.set_map(1))
  apply (rule ord_eq_le_trans)
    apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply (rule subset_trans)
  apply (rule F2set1_incl_JF2set)
apply assumption

apply (rule conjI)
apply (rule ord_eq_le_trans)
  apply (rule F2.set_map(2))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule iffD2[OF in_JF1rel])
apply (rule exI)
apply (rule conjI)
  apply (rule CollectI)
apply (rule subset_trans)
  apply (rule F2set2_JF1set_incl_JF2set)
apply assumption
apply assumption
apply (rule conjI)
  apply (rule refl)
apply (rule refl)

```

```

apply (rule ord_eq_le_trans)
apply (rule F2.set_map(3))
apply (rule image_subsetI)
apply (rule CollectI)
apply (rule case_prodI)
apply (rule iffD2[OF in_JF2rel])
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule subset_trans)
apply (rule F2set3_JF2set_incl_JF2set)
apply assumption
apply assumption
apply (rule conjI)
apply (rule refl)
apply (rule refl)
apply (rule conjI)

```

```

apply (rule trans)
apply (rule F2.map_comp)
apply (rule trans)
apply (rule F2.map_cong0)
apply (rule fun_cong[OF o_id])
apply (rule trans)
apply (rule o_apply)
apply (rule fst_conv)
apply (rule trans)
apply (rule o_apply)
apply (rule fst_conv)
apply (rule trans)
apply (rule sym)
apply (rule JF2map_simps)
apply (rule iffD2)
apply (rule dtor2_diff)
apply assumption

```

```

apply (rule trans)
apply (rule F2.map_comp)
apply (rule trans)
apply (rule F2.map_cong0)
apply (rule fun_cong[OF o_id])
apply (rule trans)
apply (rule o_apply)
apply (rule snd_conv)
apply (rule trans)
apply (rule o_apply)
apply (rule snd_conv)
apply (rule trans)
apply (rule sym)
apply (rule JF2map_simps)
apply (rule iffD2)
apply (rule dtor2_diff)
apply assumption

```

```

apply (drule iffD1[OF F2.in_rel])
apply (erule exE conjE CollectE)+
apply (rule iffD2[OF in_JF2rel])
apply (rule exI)
apply (rule conjI)
apply (rule CollectI)
apply (rule ord_eq_le_trans)
apply (rule JF2set_simps)
apply (rule Un_least)

```

```

apply (rule ord_eq_le_trans)
apply (rule trans)
  apply (rule trans)
    apply (rule arg_cong[OF dtor2_ctor2])
    apply (rule F2.set_map(1))
    apply (rule trans[OF fun_cong[OF image_id] id_apply])
apply assumption
apply (rule Un_least)
apply (rule ord_eq_le_trans)
  apply (rule trans[OF arg_cong[OF dtor2_ctor2]])
  apply (rule arg_cong[OF F2.set_map(2)])
apply (rule UN_least)
apply (drule rev_subsetD)
  apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF1rel])
apply (drule someI_ex)
apply (erule conjE)+
apply (erule CollectD)

apply (rule ord_eq_le_trans)
apply (rule trans[OF arg_cong[OF dtor2_ctor2]])
apply (rule arg_cong[OF F2.set_map(3)])
apply (rule UN_least)
apply (drule rev_subsetD)
  apply (erule image_mono)
apply (erule imageE)
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF2rel])
apply (drule someI_ex)
apply (erule exE conjE)+
apply (erule CollectD)

apply (rule conjI)

apply (rule iffD1)
  apply (rule dtor2_diff)
apply (rule trans)
  apply (rule JF2map_simps)
apply (rule trans)
apply (rule arg_cong[OF dtor2_ctor2])
apply (rule trans)
apply (rule F2.map_comp)
apply (rule trans)
apply (rule F2.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans[OF o_apply])
apply (drule rev_subsetD)
  apply assumption
apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
apply hypsubst
apply (drule iffD1[OF in_JF1rel])
apply (drule someI_ex)
apply (erule conjE)+
apply assumption
apply (rule trans[OF o_apply])
apply (drule rev_subsetD)

```

```

  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (drule iffD1[OF in_JF2rel])
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply assumption

```

```

apply (rule iffD1)
  apply (rule dtor2_diff)
  apply (rule trans)
  apply (rule JF2map_simps)
  apply (rule trans)
  apply (rule arg_cong[OF dtor2_ctor2])
  apply (rule trans)
  apply (rule F2.map_comp)
  apply (rule trans)
  apply (rule F2.map_cong0)
  apply (rule fun_cong[OF o_id])
  apply (rule trans[OF o_apply])
  apply (drule rev_subsetD)
  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (drule iffD1[OF in_JF1rel])
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply (rule trans[OF o_apply])
  apply (drule rev_subsetD)
  apply assumption
  apply (drule ssubst_mem[OF surjective_pairing[symmetric]])
  apply (erule CollectE case_prodE iffD1[OF prod.inject, elim_format] conjE)+
  apply hypsubst
  apply (drule iffD1[OF in_JF2rel])
  apply (drule someI_ex)
  apply (erule conjE)+
  apply assumption
  apply assumption
done

```

lemma *JFrel_Comp_le*:

```

  JF1rel R1 OO JF1rel R2 ≤ JF1rel (R1 OO R2) ∧ JF2rel R1 OO JF2rel R2 ≤ JF2rel (R1 OO R2)
  apply (rule J_rel_coind[OF allI[OF allI[OF impI]] allI[OF allI[OF impI]]])
  apply (rule predicate2D[OF eq_refl[OF sym[OF F1.rel_compp]]])
  apply (erule relcomppE)
  apply (rule relcomppI)
  apply (erule iffD1[OF JF1rel_F1rel])
  apply (erule iffD1[OF JF1rel_F1rel])
  apply (rule predicate2D[OF eq_refl[OF sym[OF F2.rel_compp]]])
  apply (erule relcomppE)
  apply (rule relcomppI)
  apply (erule iffD1[OF JF2rel_F2rel])
  apply (erule iffD1[OF JF2rel_F2rel])
done

```

context includes *lifting_syntax*
begin

lemma *unfold_transfer*:

```

((S ==> F1rel R S T) ==> (T ==> F2rel R S T) ==> S ==> JF1rel R) unfold1 unfold1 ∧
((S ==> F1rel R S T) ==> (T ==> F2rel R S T) ==> T ==> JF2rel R) unfold2 unfold2
unfolding rel_fun_def_butlast all_conj_distrib[symmetric] imp_conjR[symmetric]
unfolding rel_fun_iff_geq_image2p
apply (rule allI impI)+
apply (rule J_rel_coind)
apply (rule allI impI)+
apply (erule image2pE)
apply hypsubst
apply (unfold unfold1 unfold2) [1]
apply (rule rel_funD[OF rel_funD[OF rel_funD[OF rel_funD[OF F1.map_transfer]]]])
  apply (rule id_transfer)
  apply (rule rel_fun_image2p)
  apply (rule rel_fun_image2p)
apply (erule predicate2D)
apply (erule image2pI)

apply (rule allI impI)+
apply (erule image2pE)
apply hypsubst
apply (unfold unfold1 unfold2) [1]
apply (rule rel_funD[OF rel_funD[OF rel_funD[OF rel_funD[OF F2.map_transfer]]]])
  apply (rule id_transfer)
  apply (rule rel_fun_image2p)
  apply (rule rel_fun_image2p)
apply (erule predicate2D)
apply (erule image2pI)
done

```

end

```

ML ⟨
  BNF_FP_Util.mk_xtor_co_iter_o_map_thms BNF_Util.Greatest_FP false 1 @ {thm unfold_unique}
  @ {thms JF1map JF2map} (map (BNF_Tactics.mk_pointfree2 @ {context}) @ {thms unfold1 unfold2})
  @ {thms F1.map_comp0[symmetric] F2.map_comp0[symmetric]} @ {thms F1.map_cong0 F2.map_cong0}
  ⟩

```

```

ML ⟨
  BNF_FP_Util.mk_xtor_co_iter_o_map_thms BNF_Util.Greatest_FP true 1 @ {thm corec_unique}
  @ {thms JF1map JF2map} (map (BNF_Tactics.mk_pointfree2 @ {context}) @ {thms corec1 corec2})
  @ {thms F1.map_comp0[symmetric] F2.map_comp0[symmetric]} @ {thms F1.map_cong0 F2.map_cong0}
  ⟩

```

```

bnf 'a JF1
  map: JF1map
  sets: JF1set
  bd: bd_F
  wits: JF1wit
  rel: JF1rel
  apply –
  apply (rule JF1map_id)
  apply (rule JF1map_comp)
  apply (erule JF1map_cong0[OF ballI])
  apply (rule JF1set_natural)
  apply (rule bd_F_card_order)
  apply (rule conjunct1[OF bd_F_Cinfinite])
  apply (rule JF1set_bd)
  apply (rule conjunct1[OF JFrel_Comp_le])
  apply (rule JF1rel_def[unfolded OO_Grp_alt mem_Collect_eq])
apply (erule JF1wit)
done

```

```

bnf 'a JF2

```



```

map: JF2map
sets: JF2set
bd: bd_F
wits: JF2wit
rel: JF2rel
  apply -
  apply (rule JF2map_id)
  apply (rule JF2map_comp)
  apply (erule JF2map_cong0[OF ballI])
  apply (rule JF2set_natural)
  apply (rule bd_F_card_order)
  apply (rule conjunct1[OF bd_F_Cinfinite])
  apply (rule JF2set_bd)
  apply (rule conjunct2[OF JFrel_Comp_le])
  apply (rule JF2rel_def[unfolded OO_Grp_alt mem_Collect_eq])
  apply (erule JF2wit)
done

```

3 Normalized Composition of BNFs

Expected normal form: outer m-ary BNF is composed with m inner n-ary BNFs.

unbundle *cardinal_syntax*

```

declare [[bnf_internals]]
bnf-axiomatization (dead 'p1, F1set1: 'a1, F1set2: 'a2) F1
[wits: ('p1, 'a1, 'a2) F1]
for map: F1map rel: F1rel
bnf-axiomatization (dead 'p2, F2set1: 'a1, F2set2: 'a2) F2
[wits: 'a1  $\Rightarrow$  ('p2, 'a1, 'a2) F2 'a2  $\Rightarrow$  ('p2, 'a1, 'a2) F2]
for map: F2map rel: F2rel
bnf-axiomatization (dead 'p3, F3set1: 'a1, F3set2: 'a2) F3
[wits: 'a1  $\Rightarrow$  'a2  $\Rightarrow$  ('p3, 'a1, 'a2) F3]
for map: F3map rel: F3rel
bnf-axiomatization (dead 'p, Gset1: 'b1, Gset2: 'b2, Gset3: 'b3) G
[wits: 'b1  $\Rightarrow$  'b3  $\Rightarrow$  ('p, 'b1, 'b2, 'b3) G 'b2  $\Rightarrow$  'b3  $\Rightarrow$  ('p, 'b1, 'b2, 'b3) G]
for map: Gmap rel: Grel
type-synonym ('p1, 'p2, 'p3, 'p, 'a1, 'a2) H =
('p, ('p1, 'a1, 'a2) F1, ('p2, 'a1, 'a2) F2, ('p3, 'a1, 'a2) F3) G
type-synonym ('p1, 'p2, 'p3, 'p) Hbd_type =
('p1 bd_type_F1 + 'p2 bd_type_F2 + 'p3 bd_type_F3)  $\times$  'p bd_type_G

abbreviation F1in where F1in A1 A2  $\equiv$  {x. F1set1 x  $\subseteq$  A1  $\wedge$  F1set2 x  $\subseteq$  A2}
abbreviation F2in where F2in A1 A2  $\equiv$  {x. F2set1 x  $\subseteq$  A1  $\wedge$  F2set2 x  $\subseteq$  A2}
abbreviation F3in where F3in A1 A2  $\equiv$  {x. F3set1 x  $\subseteq$  A1  $\wedge$  F3set2 x  $\subseteq$  A2}
abbreviation Gin where Gin A1 A2 A3  $\equiv$  {x. Gset1 x  $\subseteq$  A1  $\wedge$  Gset2 x  $\subseteq$  A2  $\wedge$  Gset3 x  $\subseteq$  A3}

abbreviation Gset where
Gset  $\equiv$  BNF_Def.collect {Gset1, Gset2, Gset3}

abbreviation Hmap :: ('a1  $\Rightarrow$  'b1)  $\Rightarrow$  ('a2  $\Rightarrow$  'b2)  $\Rightarrow$ 
('p1, 'p2, 'p3, 'p, 'a1, 'a2) H  $\Rightarrow$  ('p1, 'p2, 'p3, 'p, 'b1, 'b2) H where
Hmap f g  $\equiv$  Gmap (F1map f g) (F2map f g) (F3map f g)

abbreviation Hset1 :: ('p1, 'p2, 'p3, 'p, 'a1, 'a2) H  $\Rightarrow$  'a1 set where
Hset1  $\equiv$  Union o Gset o Gmap F1set1 F2set1 F3set1

abbreviation Hset2 :: ('p1, 'p2, 'p3, 'p, 'a1, 'a2) H  $\Rightarrow$  'a2 set where
Hset2  $\equiv$  Union o Gset o Gmap F1set2 F2set2 F3set2

lemma Hset1_alt:

```

$Hset1 = Union\ o\ BNF_Def.collect\ \{image\ F1set1\ o\ Gset1,\ image\ F2set1\ o\ Gset2,\ image\ F3set1\ o\ Gset3\}$
by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_alt_tac\ @\{context\}\ @\{thm\ G.collect_set_map\}\rangle$)

lemma $Hset2_alt$:

$Hset2 = Union\ o\ BNF_Def.collect\ \{image\ F1set2\ o\ Gset1,\ image\ F2set2\ o\ Gset2,\ image\ F3set2\ o\ Gset3\}$
by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_alt_tac\ @\{context\}\ @\{thm\ G.collect_set_map\}\rangle$)

abbreviation Hbd **where**

$Hbd \equiv (bd_F1 +c\ bd_F2 +c\ bd_F3) *c\ bd_G$

theorem $Hmap_id$: $Hmap\ id\ id = id$

unfolding $G.map_id0\ F1.map_id0\ F2.map_id0\ F3.map_id0\ ..$

theorem $Hmap_comp$: $Hmap\ (f1\ o\ g1)\ (f2\ o\ g2) = Hmap\ f1\ f2\ o\ Hmap\ g1\ g2$

unfolding $G.map_comp0\ F1.map_comp0\ F2.map_comp0\ F3.map_comp0\ ..$

theorem $Hmap_cong$: $\llbracket \bigwedge z. z \in Hset1\ x \implies f1\ z = g1\ z; \bigwedge z. z \in Hset2\ x \implies f2\ z = g2\ z \rrbracket \implies$

$Hmap\ f1\ f2\ x = Hmap\ g1\ g2\ x$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_map_cong0_tac\ @\{context\}\$

$\rrbracket\ @\{thms\ Hset1_alt\ Hset2_alt\}\ @\{thm\ G.map_cong0\}\ @\{thms\ F1.map_cong0\ F2.map_cong0\ F3.map_cong0\}\rangle$)

theorem $Hset1_natural$: $Hset1\ o\ Hmap\ f1\ f2 = image\ f1\ o\ Hset1$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_map0_tac\ @\{context\}\ @\{thm\ refl\}\ @\{thm\ G.map_comp0\}\ @\{thm\ G.map_cong0\}$

$\ @\{thm\ G.collect_set_map\}\ @\{thms\ F1.set_map0(1)\ F2.set_map0(1)\ F3.set_map0(1)\}\rangle$)

theorem $Hset2_natural$: $Hset2\ o\ Hmap\ f1\ f2 = image\ f2\ o\ Hset2$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_map0_tac\ @\{context\}\ @\{thm\ refl\}\ @\{thm\ G.map_comp0\}\ @\{thm\ G.map_cong0\}$

$\ @\{thm\ G.collect_set_map\}\ @\{thms\ F1.set_map0(2)\ F2.set_map0(2)\ F3.set_map0(2)\}\rangle$)

theorem Hbd_card_order : $card_order\ Hbd$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_bd_card_order_tac\ @\{context\}\$

$\ @\{thms\ F1.bd_card_order\ F2.bd_card_order\ F3.bd_card_order\}\ @\{thm\ G.bd_card_order\}\rangle$)

theorem $Hbd_cinfinte$: $cinfinte\ Hbd$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_bd_cinfinte_tac\ @\{context\}\$

$\ @\{thm\ F1.bd_cinfinte\}\ @\{thm\ G.bd_cinfinte\}\rangle$)

theorem $Hset1_bd$: $|Hset1\ (x :: ('p1,\ 'p2,\ 'p3,\ 'p,\ 'a1,\ 'a2)\ H)| \leq o$

$(Hbd :: ('p1,\ 'p2,\ 'p3,\ 'p)\ Hbd_type\ rel)$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_bd_tac\ @\{context\}\ @\{thm\ refl\}\ NONE\ @\{thm\ Hset1_alt\}$

$\ @\{thms\ comp_single_set_bd[OF\ F1.bd_Card_order\ F1.set_bd(1)\ G.set_bd(1)]$

$\ comp_single_set_bd[OF\ F2.bd_Card_order\ F2.set_bd(1)\ G.set_bd(2)]$

$\ comp_single_set_bd[OF\ F3.bd_Card_order\ F3.set_bd(1)\ G.set_bd(3)]\}\rangle$)

theorem $Hset2_bd$: $|Hset2\ (x :: ('p1,\ 'p2,\ 'p3,\ 'p,\ 'a1,\ 'a2)\ H)| \leq o$

$(Hbd :: ('p1,\ 'p2,\ 'p3,\ 'p)\ Hbd_type\ rel)$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_set_bd_tac\ @\{context\}\ @\{thm\ refl\}\ NONE\ @\{thm\ Hset2_alt\}$

$\ @\{thms\ comp_single_set_bd[OF\ F1.bd_Card_order\ F1.set_bd(2)\ G.set_bd(1)]$

$\ comp_single_set_bd[OF\ F2.bd_Card_order\ F2.set_bd(2)\ G.set_bd(2)]$

$\ comp_single_set_bd[OF\ F3.bd_Card_order\ F3.set_bd(2)\ G.set_bd(3)]\}\rangle$)

abbreviation Hin **where** $Hin\ A1\ A2 \equiv \{x. Hset1\ x \subseteq A1 \wedge Hset2\ x \subseteq A2\}$

lemma Hin_alt : $Hin\ A1\ A2 = Gin\ (F1in\ A1\ A2)\ (F2in\ A1\ A2)\ (F3in\ A1\ A2)$

by (tactic $\langle BNF_Comp_Tactics.mk_comp_in_alt_tac\ @\{context\}\ @\{thms\ Hset1_alt\ Hset2_alt\}\rangle$)

definition $Hwit1$ **where** $Hwit1\ b\ c = wit1_G\ wit_F1\ (wit_F3\ b\ c)$

definition $Hwit21$ **where** $Hwit21\ b\ c = wit2_G\ (wit1_F2\ b)\ (wit_F3\ b\ c)$

definition $Hwit22$ **where** $Hwit22\ b\ c = wit2_G\ (wit2_F2\ c)\ (wit_F3\ b\ c)$

lemma $Hwit1$:

```

 $\bigwedge x. x \in \text{Hset1 } (\text{Hwit1 } b \ c) \implies x = b$ 
 $\bigwedge x. x \in \text{Hset2 } (\text{Hwit1 } b \ c) \implies x = c$ 
unfolding Hwit1_def
by (tactic  $\langle \text{BNF\_Comp\_Tactics.mk\_comp\_wit\_tac } @\{\text{context}\} \ [] \ @\{\text{thms } G.\text{wit1 } G.\text{wit2}\} \ @\{\text{thm } G.\text{collect\_set\_map}\} \ @\{\text{thms } F1.\text{wit } F2.\text{wit1 } F2.\text{wit2 } F3.\text{wit}\} \rangle$ )

```

lemma *Hwit21*:

```

 $\bigwedge x. x \in \text{Hset1 } (\text{Hwit21 } b \ c) \implies x = b$ 
 $\bigwedge x. x \in \text{Hset2 } (\text{Hwit21 } b \ c) \implies x = c$ 
unfolding Hwit21_def
by (tactic  $\langle \text{BNF\_Comp\_Tactics.mk\_comp\_wit\_tac } @\{\text{context}\} \ [] \ @\{\text{thms } G.\text{wit1 } G.\text{wit2}\} \ @\{\text{thm } G.\text{collect\_set\_map}\} \ @\{\text{thms } F1.\text{wit } F2.\text{wit1 } F2.\text{wit2 } F3.\text{wit}\} \rangle$ )

```

lemma *Hwit22*:

```

 $\bigwedge x. x \in \text{Hset1 } (\text{Hwit22 } b \ c) \implies x = b$ 
 $\bigwedge x. x \in \text{Hset2 } (\text{Hwit22 } b \ c) \implies x = c$ 
unfolding Hwit22_def
by (tactic  $\langle \text{BNF\_Comp\_Tactics.mk\_comp\_wit\_tac } @\{\text{context}\} \ [] \ @\{\text{thms } G.\text{wit1 } G.\text{wit2}\} \ @\{\text{thm } G.\text{collect\_set\_map}\} \ @\{\text{thms } F1.\text{wit } F2.\text{wit1 } F2.\text{wit2 } F3.\text{wit}\} \rangle$ )

```

lemma *Grel_cong*: $[[R1 = S1; R2 = S2; R3 = S3]] \implies \text{Grel } R1 \ R2 \ R3 = \text{Grel } S1 \ S2 \ S3$

by *hypsubst (rule refl)*

definition *Hrel where*

```

Hrel R1 R2 = (BNF_Def.Grp (Hin (Collect (case_prod R1)) (Collect (case_prod R2))) (Hmap fst fst))  $\hat{\ }^{-1}$  OO
  (BNF_Def.Grp (Hin (Collect (case_prod R1)) (Collect (case_prod R2))) (Hmap snd snd))

```

lemmas *Hrel_unfold* = $\text{trans}[OF \ \text{Hrel_def} \ \text{trans}[OF \ \text{OO_Grp_cong}[OF \ \text{Hin_alt}]$

```

   $\text{trans}[OF \ \text{arg\_cong2}[of \ \_ \ \_ \ \_ \ \text{relcompp}, \ OF \ \text{trans}[OF \ \text{arg\_cong}[of \ \_ \ \_ \ \text{conversep}, \ OF \ \text{sym}[OF \ G.\text{rel\_Grp}]]$ 
  G.rel_conversep[symmetric]] sym[OF G.rel_Grp]]
   $\text{trans}[OF \ G.\text{rel\_compp}[symmetric] Grel_cong[OF sym[OF F1.rel_compp_Grp] sym[OF F2.rel_compp_Grp]
  sym[OF F3.rel_compp_Grp]]]]]]$ 
```

bnf *H*: ('p1, 'p2, 'p3, 'p, 'a1, 'a2) *H*

map: *Hmap*

sets: *Hset1* *Hset2*

bd: *Hbd* :: ('p1, 'p2, 'p3, 'p) *Hbd_type* *rel*

rel: *Hrel*

```

  apply –
  apply (rule Hmap_id)
  apply (rule Hmap_comp)
  apply (erule Hmap_cong) apply assumption
  apply (rule Hset1_natural)
  apply (rule Hset2_natural)
  apply (rule Hbd_card_order)
  apply (rule Hbd_cinfinite)
  apply (rule Hset1_bd)
  apply (rule Hset2_bd)
  apply (unfold Hrel_unfold G.rel_compp[symmetric] F1.rel_compp[symmetric] F2.rel_compp[symmetric] F3.rel_compp[symmetric]
  eq_OO) [1] apply (rule order_refl)
  apply (rule Hrel_def[unfolded OO_Grp_alt mem_Collect_eq])
  done

```

4 Removing Live Variables

unbundle *cardinal_syntax*

declare [[*bnf_internals*]]

bnf-axiomatization (*dead* 'p, *Fset1*: 'a1, *Fset2*: 'a2, *Fset3*: 'a3) *F* **for** *map*: *Fmap* *rel*: *Frel*

abbreviation *F1map* :: ('a2 \Rightarrow 'b2) \Rightarrow ('a3 \Rightarrow 'b3) \Rightarrow ('p, 'a1, 'a2, 'a3) *F* \Rightarrow ('p, 'a1, 'b2, 'b3) *F* **where**
F1map \equiv *Fmap id*

abbreviation *F2map* :: ('a3 \Rightarrow 'b3) \Rightarrow ('p, 'a1, 'a2, 'a3) *F* \Rightarrow ('p, 'a1, 'a2, 'b3) *F* **where**
F2map \equiv *Fmap id id*

abbreviation *F1set1* \equiv *Fset2*

abbreviation *F1set2* \equiv *Fset3*

abbreviation *F2set* \equiv *Fset3*

theorem *F1map_id*: *F1map id id = id*
by (*rule F.map_id0*)

theorem *F2map_id*: *F2map id = id*
by (*rule F.map_id0*)

theorem *F1map_comp*: *F1map (f1 o g1) (f2 o g2) = F1map f1 f2 o F1map g1 g2*
by (*unfold F.map_comp0[symmetric] o_id*) (*rule refl*)

theorem *F2map_comp*: *F2map (f o g) = F2map f o F2map g*
by (*unfold F.map_comp0[symmetric] o_id*) (*rule refl*)

theorem *F1map_cong*: $\llbracket \bigwedge z. z \in F1set1\ x \Longrightarrow f1\ z = g1\ z; \bigwedge z. z \in F1set2\ x \Longrightarrow f2\ z = g2\ z \rrbracket$
 $\Longrightarrow F1map\ f1\ f2\ x = F1map\ g1\ g2\ x$
apply (*rule F.map_cong0*)
apply (*rule refl*)
apply *assumption*
apply *assumption*
done

theorem *F2map_cong*: $\llbracket \bigwedge z. z \in F2set\ x \Longrightarrow f\ z = g\ z \rrbracket \Longrightarrow F2map\ f\ x = F2map\ g\ x$
apply (*rule F.map_cong0*)
apply (*rule refl*)
apply (*rule refl*)
apply *assumption*
done

theorem *F1set1_natural*: *F1set1 o F1map f1 f2 = image f1 o F1set1*
by (*rule F.set_map0(2)*)

theorem *F1set2_natural*: *F1set2 o F1map f1 f2 = image f2 o F1set2*
by (*rule F.set_map0(3)*)

theorem *F2set_natural*: *F2set o F2map f = image f o F2set*
by (*rule F.set_map0(3)*)

abbreviation *Fin* :: 'a1 *set* \Rightarrow 'a2 *set* \Rightarrow 'a3 *set* \Rightarrow (('p, 'a1, 'a2, 'a3) *F*) *set* **where**
Fin A1 A2 A3 \equiv {*x*. *Fset1* *x* \subseteq *A1* \wedge *Fset2* *x* \subseteq *A2* \wedge *Fset3* *x* \subseteq *A3*}

abbreviation *F1in* :: 'a2 *set* \Rightarrow 'a3 *set* \Rightarrow (('p, 'a1, 'a2, 'a3) *F*) *set* **where**
F1in A1 A2 \equiv {*x*. *F1set1* *x* \subseteq *A1* \wedge *F1set2* *x* \subseteq *A2*}

lemma *F1in_alt*: *F1in A2 A3 = Fin UNIV A2 A3*
by (*tactic* \langle *BNF_Comp_Tactics.kill_in_alt_tac* $\@$ {*context*} \rangle)

abbreviation *F2in* :: 'a3 *set* \Rightarrow (('p, 'a1, 'a2, 'a3) *F*) *set* **where**
F2in A \equiv {*x*. *F2set* *x* \subseteq *A*}

lemma *F2in_alt*: *F2in A3 = Fin UNIV UNIV A3*
by (*tactic* \langle *BNF_Comp_Tactics.kill_in_alt_tac* $\@$ {*context*} \rangle)

lemma *Frel_cong*: $\llbracket R1 = S1; R2 = S2; R3 = S3 \rrbracket \implies \text{Frel } R1 \ R2 \ R3 = \text{Frel } S1 \ S2 \ S3$
by *hypsubst* (rule *refl*)

definition *F1rel* **where**

$\text{F1rel } R1 \ R2 = (\text{BNF_Def.Grp } (\text{F1in } (\text{Collect } (\text{case_prod } R1))) (\text{Collect } (\text{case_prod } R2))) (\text{F1map } \text{fst } \text{fst})^{\hat{-}-1}$
 OO
 $(\text{BNF_Def.Grp } (\text{F1in } (\text{Collect } (\text{case_prod } R1))) (\text{Collect } (\text{case_prod } R2))) (\text{F1map } \text{snd } \text{snd}))$

lemmas *F1rel_unfold* = $\text{trans}[\text{OF } \text{F1rel_def } \text{trans}[\text{OF } \text{OO_Grp_cong}[\text{OF } \text{F1in_alt}]$
 $\text{trans}[\text{OF } \text{arg_cong2}[\text{of } _ _ _ _ \text{relcompp}, \text{OF } \text{trans}[\text{OF } \text{arg_cong}[\text{of } _ _ \text{conversep}, \text{OF } \text{sym}[\text{OF } \text{F.rel_Grp}]]$
 $\text{F.rel_conversep}[\text{symmetric}]] \text{sym}[\text{OF } \text{F.rel_Grp}]]$
 $\text{trans}[\text{OF } \text{F.rel_compp}[\text{symmetric}] \text{Frel_cong}[\text{OF } \text{trans}[\text{OF } \text{Grp_UNIV_id}[\text{OF } \text{refl}] \text{eq_alt}[\text{symmetric}]] \text{Grp_fst_snd}$
 $\text{Grp_fst_snd}]]]]]$

definition *F2rel* **where**

$\text{F2rel } R1 = (\text{BNF_Def.Grp } (\text{F2in } (\text{Collect } (\text{case_prod } R1))) (\text{F2map } \text{fst}))^{\hat{-}-1}$
 OO
 $(\text{BNF_Def.Grp } (\text{F2in } (\text{Collect } (\text{case_prod } R1))) (\text{F2map } \text{snd}))$

lemmas *F2rel_unfold* = $\text{trans}[\text{OF } \text{F2rel_def } \text{trans}[\text{OF } \text{OO_Grp_cong}[\text{OF } \text{F2in_alt}]$
 $\text{trans}[\text{OF } \text{arg_cong2}[\text{of } _ _ _ _ \text{relcompp}, \text{OF } \text{trans}[\text{OF } \text{arg_cong}[\text{of } _ _ \text{conversep}, \text{OF } \text{sym}[\text{OF } \text{F.rel_Grp}]]$
 $\text{F.rel_conversep}[\text{symmetric}]] \text{sym}[\text{OF } \text{F.rel_Grp}]]$
 $\text{trans}[\text{OF } \text{F.rel_compp}[\text{symmetric}] \text{Frel_cong}[\text{OF } \text{trans}[\text{OF } \text{Grp_UNIV_id}[\text{OF } \text{refl}] \text{eq_alt}[\text{symmetric}]] \text{trans}[\text{OF } \text{Grp_UNIV_id}[\text{OF } \text{refl}] \text{eq_alt}[\text{symmetric}]] \text{Grp_fst_snd}]]]]]$

bnf *F1*: ('p, 'a1, 'a2, 'a3) *F*

map: *F1map*

sets: *F1set1 F1set2*

bd: *bd_F* :: ('p *bd_type_F*) *rel*

rel: *F1rel*

apply –
apply (rule *F1map_id*)
apply (rule *F1map_comp*)
apply (erule *F1map_cong*) **apply** *assumption*
apply (rule *F1set1_natural*)
apply (rule *F1set2_natural*)
apply (rule *F.bd_card_order*)
apply (rule *F.bd_cinfinite*)
apply (rule *F.set_bd(2)*)
apply (rule *F.set_bd(3)*)
apply (unfold *F1rel_unfold* *F.rel_compp*[*symmetric*] *eq_OO*) [1] **apply** (rule *order_refl*)
apply (rule *F1rel_def*[*unfolded OO_Grp_alt mem_Collect_eq*])
done

bnf *F2*: ('p, 'a1, 'a2, 'a3) *F*

map: *F2map*

sets: *F2set*

bd: *bd_F* :: ('p *bd_type_F*) *rel*

rel: *F2rel*

apply –
apply (rule *F2map_id*)
apply (rule *F2map_comp*)
apply (erule *F2map_cong*)
apply (rule *F2set_natural*)
apply (rule *F.bd_card_order*)
apply (rule *F.bd_cinfinite*)
apply (rule *F.set_bd(3)*)
apply (unfold *F2rel_unfold* *F.rel_compp*[*symmetric*] *eq_OO*) [1] **apply** (rule *order_refl*)
apply (rule *F2rel_def*[*unfolded OO_Grp_alt mem_Collect_eq*])
done

5 Adding New Live Variables

unbundle *cardinal_syntax*

declare $[[bnf_internals]]$

bnf-axiomatization (*dead* 'p, *Fset1*: 'a1, *Fset2*: 'a2) *F*

[*wits*: 'a1 \Rightarrow 'a2 \Rightarrow ('p, 'a1, 'a2) *F*]

for *map*: *Fmap* *rel*: *Frel*

type-synonym ('p, 'a1, 'a2, 'a3, 'a4) *F'* = ('p, 'a3, 'a4) *F*

abbreviation *F'map* :: ('a1 \Rightarrow 'b1) \Rightarrow ('a2 \Rightarrow 'b2) \Rightarrow ('a3 \Rightarrow 'b3) \Rightarrow ('a4 \Rightarrow 'b4) \Rightarrow ('p, 'a1, 'a2, 'a3, 'a4) *F'*
 \Rightarrow ('p, 'b1, 'b2, 'b3, 'b4) *F'* **where**

F'map *f1* *f2* *f3* *f4* \equiv *Fmap* *f3* *f4*

abbreviation *F'set1* :: ('p, 'a1, 'a2, 'a3, 'a4) *F'* \Rightarrow 'a1 *set* **where**

F'set1 \equiv $\lambda_.$ {}

abbreviation *F'set2* :: ('p, 'a1, 'a2, 'a3, 'a4) *F'* \Rightarrow 'a2 *set* **where**

F'set2 \equiv $\lambda_.$ {}

abbreviation *F'set3* :: ('p, 'a1, 'a2, 'a3, 'a4) *F'* \Rightarrow 'a3 *set* **where**

F'set3 \equiv *Fset1*

abbreviation *F'set4* :: ('p, 'a1, 'a2, 'a3, 'a4) *F'* \Rightarrow 'a4 *set* **where**

F'set4 \equiv *Fset2*

abbreviation *F'bd* **where**

F'bd \equiv *bd_F*

theorem *F'map_id*: *F'map* *id* *id* *id* *id* = *id*

by (*rule* *F.map_id0*)

theorem *F'map_comp*:

F'map (*f1* *o* *g1*) (*f2* *o* *g2*) (*f3* *o* *g3*) (*f4* *o* *g4*) = *F'map* *f1* *f2* *f3* *f4* *o* *F'map* *g1* *g2* *g3* *g4*

by (*rule* *F.map_comp0*)

theorem *F'map_cong*:

$[[\bigwedge z. z \in F'set1\ x \implies f1\ z = g1\ z; \bigwedge z. z \in F'set2\ x \implies f2\ z = g2\ z;$

$\bigwedge z. z \in F'set3\ x \implies f3\ z = g3\ z; \bigwedge z. z \in F'set4\ x \implies f4\ z = g4\ z]]$

$\implies F'map\ f1\ f2\ f3\ f4\ x = F'map\ g1\ g2\ g3\ g4\ x$

apply (*tactic* \langle *BNF_Util.rtac* $\@$ {*context*} $\@$ {*thm* *F.map_cong0*} 1 *THEN REPEAT_DETERM_N* 2 (*assume_tac* $\@$ {*context*} 1) \rangle)

apply *assumption+*

done

theorem *F'set1_natural*: *F'set1* *o* *F'map* *f1* *f2* *f3* *f4* = *image* *f1* *o* *F'set1*

by (*tactic* \langle *BNF_Comp_Tactics.empty_natural_tac* $\@$ {*context*} \rangle)

theorem *F'set2_natural*: *F'set2* *o* *F'map* *f1* *f2* *f3* *f4* = *image* *f2* *o* *F'set2*

by (*tactic* \langle *BNF_Comp_Tactics.empty_natural_tac* $\@$ {*context*} \rangle)

theorem *F'set3_natural*: *F'set3* *o* *F'map* *f1* *f2* *f3* *f4* = *image* *f3* *o* *F'set3*

by (*rule* *F.set_map0*(1))

theorem *F'set4_natural*: *F'set4* *o* *F'map* *f1* *f2* *f3* *f4* = *image* *f4* *o* *F'set4*

by (*rule* *F.set_map0*(2))

theorem *F'bd_card_order*: *card_order* *bd_F*

by (*rule* *F.bd_card_order*)

theorem *F'bd_cinfinite*: *cinfinite* *bd_F*

by (*rule* *F.bd_cinfinite*)

theorem $F'set1_bd$: $|F'set1\ x| \leq_o F'bd$
by (tactic $\langle BNF_Comp_Tactics.mk_lift_set_bd_tac\ @\{context\}\ @\{thm\ F.bd_Card_order\}\rangle$)

theorem $F'set2_bd$: $|F'set2\ x| \leq_o F'bd$
by (tactic $\langle BNF_Comp_Tactics.mk_lift_set_bd_tac\ @\{context\}\ @\{thm\ F.bd_Card_order\}\rangle$)

theorem $F'set3_bd$: $|F'set3\ (x :: ('c, 'a, 'd)\ F)| \leq_o (F'bd :: 'c\ bd_type_F\ rel)$
by (rule $F.set_bd(1)$)

theorem $F'set4_bd$: $|F'set4\ (x :: ('c, 'a, 'd)\ F)| \leq_o (F'bd :: 'c\ bd_type_F\ rel)$
by (rule $F.set_bd(2)$)

abbreviation $F'in :: 'a1\ set \Rightarrow 'a2\ set \Rightarrow 'a3\ set \Rightarrow 'a4\ set \Rightarrow ((p, 'a1, 'a2, 'a3, 'a4)\ F)\ set$ **where**
 $F'in\ A1\ A2\ A3\ A4 \equiv \{x. F'set1\ x \subseteq A1 \wedge F'set2\ x \subseteq A2 \wedge F'set3\ x \subseteq A3 \wedge F'set4\ x \subseteq A4\}$

definition $F'rel$ **where**
 $F'rel\ R1\ R2\ R3\ R4 = (BNF_Def.Grp\ (F'in\ (Collect\ (case_prod\ R1))\ (Collect\ (case_prod\ R2))\ (Collect\ (case_prod\ R3))\ (Collect\ (case_prod\ R4))))\ (F'map\ fst\ fst\ fst\ fst)\ ^{-1}\ OO$
 $(BNF_Def.Grp\ (F'in\ (Collect\ (case_prod\ R1))\ (Collect\ (case_prod\ R2))\ (Collect\ (case_prod\ R3))\ (Collect\ (case_prod\ R4))))\ (F'map\ snd\ snd\ snd\ snd)$

lemmas $F'rel_unfold = trans[OF\ F'rel_def[unfolded\ eqTrueI[OF\ empty_subsetI]\ simp_thms(2)]]$
 $trans[OF\ OO_Grp_cong[OF\ refl]\ sym[OF\ F.rel_compp_Grp]]]$

bnf F' : $(p, 'a1, 'a2, 'a3, 'a4)\ F'$
map: $F'map$
sets: $F'set1\ F'set2\ F'set3\ F'set4$
bd: $F'bd :: p\ bd_type_F\ rel$
wits: wit_F
rel: $F'rel$
plugins **del**: $lifting\ transfer$
apply $-$
apply (rule $F'map_id$)
apply (rule $F'map_comp$)
apply (erule $F'map_cong$) **apply** $assumption+$
apply (rule $F'set1_natural$)
apply (rule $F'set2_natural$)
apply (rule $F'set3_natural$)
apply (rule $F'set4_natural$)
apply (rule $F'bd_card_order$)
apply (rule $F'bd_cinfinte$)
apply (rule $F'set1_bd$)
apply (rule $F'set2_bd$)
apply (rule $F'set3_bd$)
apply (rule $F'set4_bd$)
apply (unfold $F'rel_unfold\ F.rel_compp[symmetric]\ eq_OO$) [1] **apply** (rule $order_refl$)
apply (rule $F'rel_def[unfolded\ OO_Grp_alt\ mem_Collect_eq]$)
apply (erule $F.wit_emptyE$)
done

6 Changing the Order of Live Variables

unbundle $cardinal_syntax$

declare $[[bnf_internals]]$

bnf-axiomatization $(dead\ p, Fset1: 'a1, Fset2: 'a2, Fset3: 'a3)\ F$ **for** $map: Fmap\ rel: Frel$
type-synonym $(p, 'a1, 'a2, 'a3)\ F' = (p, 'a3, 'a1, 'a2)\ F$

abbreviation $Fin :: 'a1\ set \Rightarrow 'a2\ set \Rightarrow 'a3\ set \Rightarrow ((p, 'a1, 'a2, 'a3)\ F)\ set$ **where**
 $Fin\ A1\ A2\ A3 \equiv \{x. Fset1\ x \subseteq A1 \wedge Fset2\ x \subseteq A2 \wedge Fset3\ x \subseteq A3\}$

abbreviation $F'_{map} :: ('a1 \Rightarrow 'b1) \Rightarrow ('a2 \Rightarrow 'b2) \Rightarrow ('a3 \Rightarrow 'b3) \Rightarrow ('p, 'a1, 'a2, 'a3) F' \Rightarrow ('p, 'b1, 'b2, 'b3) F'$ **where**

$F'_{map} f g h \equiv F_{map} h f g$

abbreviation $F'_{set1} :: ('p, 'a1, 'a2, 'a3) F' \Rightarrow 'a1$ **set where**

$F'_{set1} \equiv F_{set2}$

abbreviation $F'_{set2} :: ('p, 'a1, 'a2, 'a3) F' \Rightarrow 'a2$ **set where**

$F'_{set2} \equiv F_{set3}$

abbreviation $F'_{set3} :: ('p, 'a1, 'a2, 'a3) F' \Rightarrow 'a3$ **set where**

$F'_{set3} \equiv F_{set1}$

abbreviation F'_{bd} **where**

$F'_{bd} \equiv bd_F$

theorem F'_{map_id} : $F'_{map} id id id = id$

by (rule $F.map_id0$)

theorem F'_{map_comp} : $F'_{map} (f1 o g1) (f2 o g2) (f3 o g3) = F'_{map} f1 f2 f3 o F'_{map} g1 g2 g3$

by (rule $F.map_comp0$)

theorem F'_{map_cong} : $\llbracket \bigwedge z. z \in F'_{set1} x \Longrightarrow f1 z = g1 z; \bigwedge z. z \in F'_{set2} x \Longrightarrow f2 z = g2 z; \bigwedge z. z \in F'_{set3} x \Longrightarrow f3 z = g3 z \rrbracket$

$\Longrightarrow F'_{map} f1 f2 f3 x = F'_{map} g1 g2 g3 x$

apply (rule $F.map_cong0$)

apply *assumption+*

done

theorem $F'_{set1_natural}$: $F'_{set1} o F'_{map} f1 f2 f3 = image f1 o F'_{set1}$

by (rule $F.set_map0(2)$)

theorem $F'_{set2_natural}$: $F'_{set2} o F'_{map} f1 f2 f3 = image f2 o F'_{set2}$

by (rule $F.set_map0(3)$)

theorem $F'_{set3_natural}$: $F'_{set3} o F'_{map} f1 f2 f3 = image f3 o F'_{set3}$

by (rule $F.set_map0(1)$)

theorem $F'_{bd_card_order}$: $card_order F'_{bd}$

by (rule $F.bd_card_order$)

theorem $F'_{bd_cinfinte}$: $cinfinte F'_{bd}$

by (rule $F.bd_cinfinte$)

theorem F'_{set1_bd} : $|F'_{set1} (x :: ('c, 'a, 'b, 'd) F)| \leq o (F'_{bd} :: 'c$ bd_type_F $rel)$

by (rule $F.set_bd(2)$)

theorem F'_{set2_bd} : $|F'_{set2} (x :: ('c, 'a, 'b, 'd) F)| \leq o (F'_{bd} :: 'c$ bd_type_F $rel)$

by (rule $F.set_bd(3)$)

theorem F'_{set3_bd} : $|F'_{set3} (x :: ('c, 'a, 'b, 'd) F)| \leq o (F'_{bd} :: 'c$ bd_type_F $rel)$

by (rule $F.set_bd(1)$)

abbreviation $F'_{in} :: 'a1$ **set** $\Rightarrow 'a2$ **set** $\Rightarrow 'a3$ **set** $\Rightarrow ((p, 'a1, 'a2, 'a3) F')$ **set where**

$F'_{in} A1 A2 A3 \equiv \{x. F'_{set1} x \subseteq A1 \wedge F'_{set2} x \subseteq A2 \wedge F'_{set3} x \subseteq A3\}$

lemma F'_{in_alt} : $F'_{in} A1 A2 A3 = Fin A3 A1 A2$

apply (rule $Collect_cong$)

by (tactic $\langle BNF_Tactics.mk_rotate_eq_tac @\{context\}$

$(BNF_Util.rtac @\{context\} @\{thm refl\}) @\{thm trans\} @\{thm conj_assoc\} @\{thm conj_commute\} @\{thm conj_cong\}$

$[1, 2, 3] [3, 1, 2] 1 \rangle$)

definition F'_{rel} **where**


```

F'rel R1 R2 R3 = (BNF_Def.Grp (F'in (Collect (case_prod R1)) (Collect (case_prod R2)) (Collect (case_prod R3)))) (F'map fst fst fst) ^--1 OO
(BNF_Def.Grp (F'in (Collect (case_prod R1)) (Collect (case_prod R2)) (Collect (case_prod R3)))
(F'map snd snd snd))

```

```

lemmas F'rel_unfold = trans[OF F'rel_def trans[OF OO_Grp_cong[OF F'in_alt] sym[OF F.rel_compp_Grp]]]

```

```

bnf F': ('p, 'a1, 'a2, 'a3) F'
  map: F'map
  sets: F'set1 F'set2 F'set3
  bd: F'bd :: 'p bd_type_F rel
  rel: F'rel
    apply -
      apply (rule F'map_id)
      apply (rule F'map_comp)
      apply (erule F'map_cong) apply assumption+
      apply (rule F'set1_natural)
      apply (rule F'set2_natural)
      apply (rule F'set3_natural)
      apply (rule F'bd_card_order)
      apply (rule F'bd_cinfinite)
      apply (rule F'set1_bd)
      apply (rule F'set2_bd)
      apply (rule F'set3_bd)
      apply (unfold F'rel_unfold F.rel_compp[symmetric] eq_OO) [1] apply (rule order_refl)
      apply (rule F'rel_def[unfolded OO_Grp_alt mem_Collect_eq])
    done

```

7 Mutual View on Nested Datatypes

```

notation BNF_Def.convolve (<_, _>)

```

```

declare [[bnf_internals]]

```

```

declare [[typedef_overloaded]]

```

```

bnf-axiomatization ('a, 'b) F0 [wits: 'a ⇒ ('a, 'b) F0]

```

```

bnf-axiomatization ('a, 'b) G0 [wits: 'a ⇒ ('a, 'b) G0]

```

7.1 Nested Definition

```

datatype 'a F = CF ('a, 'a F) F0

```

```

datatype 'a G = CG ('a, ('a G) F) G0

```

```

type-synonym ('b, 'c) F_pre_F = ('c, 'b) F0

```

```

type-synonym ('c, 'a) G_pre_G = ('a, 'c F) G0

```

```

term ctor_fold_F :: (('b, 'c) F_pre_F ⇒ 'b) ⇒ 'c F ⇒ 'b

```

```

term ctor_fold_G :: (('c, 'a) G_pre_G ⇒ 'c) ⇒ 'a G ⇒ 'c

```

```

term ctor_rec_F :: (('c F × 'b, 'c) F_pre_F ⇒ 'b) ⇒ 'c F ⇒ 'b

```

```

term ctor_rec_G :: (('a G × 'c, 'a) G_pre_G ⇒ 'c) ⇒ 'a G ⇒ 'c

```

```

thm F.ctor_rel_induct

```

```

thm G.ctor_rel_induct[unfolded rel_pre_G_def id_apply]

```

7.2 Isomorphic Mutual Definition

```

datatype 'a G_M = CG ('a, 'a G_M) G0

```

```

and 'a G_F_M = CF ('a G_M, 'a G_F_M) F0

```

```

type-synonym ('b, 'c) G_F_M_pre_G_F_M = ('c, 'b) F0

```

type-synonym ($'c, 'a$) $G_M_pre_G_M = ('a, 'c) G_O$

term $ctor_fold_G_M :: (('c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('c, 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G_M \Rightarrow 'b$

term $ctor_fold_GF_M :: (('c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('c, 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a GF_M \Rightarrow 'c$

term $ctor_rec_G_M :: (('a GF_M \times 'c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('a GF_M \times 'c, 'a G_M \times 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G_M \Rightarrow 'b$

term $ctor_rec_GF_M :: (('a GF_M \times 'c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('a GF_M \times 'c, 'a G_M \times 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a GF_M \Rightarrow 'c$

thm $G_M_GF_M.ctor_rel_induct[unfolded rel_pre_G_M_def rel_pre_GF_M_def]$

7.3 Mutualization

7.3.1 Iterators

definition $n2m_ctor_fold_G :: (('c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('c, 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G \Rightarrow 'b$

where $n2m_ctor_fold_G s1 s2 = ctor_fold_G (s1 o map_pre_G_M id (id :: unit \Rightarrow unit) (ctor_fold_F (s2 o BNF_Composition.id_bnf o BNF_Composition.id_bnf)) o BNF_Composition.id_bnf o BNF_Composition.id_bnf)$

definition $n2m_ctor_fold_G_F :: (('c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('c, 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G F \Rightarrow 'c$

where $n2m_ctor_fold_G_F s1 s2 = ctor_fold_F (s2 o map_pre_GF_M (id :: unit \Rightarrow unit) (n2m_ctor_fold_G s1 s2) id o BNF_Composition.id_bnf o BNF_Composition.id_bnf)$

lemma $G_ctor_o_fold: ctor_fold_G s o ctor_G = s o map_pre_G id (ctor_fold_G s)$

unfolding $fun_eq_iff o_apply G.ctor_fold$ **by** $simp$

lemma $F_ctor_o_fold: ctor_fold_F s o ctor_F = s o map_pre_F id (ctor_fold_F s)$

unfolding $fun_eq_iff o_apply F.ctor_fold$ **by** $simp$

lemma $G_ctor_o_rec: ctor_rec_G s o ctor_G = s o map_pre_G id (BNF_Def.convolve id (ctor_rec_G s))$

unfolding $fun_eq_iff o_apply G.ctor_rec$ **by** $simp$

lemma $F_ctor_o_rec: ctor_rec_F s o ctor_F = s o map_pre_F id (BNF_Def.convolve id (ctor_rec_F s))$

unfolding $fun_eq_iff o_apply F.ctor_rec$ **by** $simp$

lemma $n2m_ctor_fold_G:$

$n2m_ctor_fold_G s1 s2 o ctor_G = s1 o map_pre_G_M id (n2m_ctor_fold_G_F s1 s2) o BNF_Composition.id_bnf o BNF_Composition.id_bnf$

unfolding $n2m_ctor_fold_G_def n2m_ctor_fold_G_F_def$

$map_pre_G_def map_pre_F_def map_pre_G_M_def map_pre_GF_M_def$

$G_ctor_o_fold id_apply comp_id id_comp comp_assoc$

$rewriteL_comp_comp[OF type_copy_map_comp0_undo[OF BNF_Composition.type_definition_id_bnf_UNIV BNF_Composition.type_definition_id_bnf_UNIV BNF_Composition.type_definition_id_bnf_UNIV pre_G_M.map_comp0[unfolding map_pre_G_M_def]]]$

$F.ctor_fold_o_map$

$rewriteL_comp_comp[OF type_copy_Rep_o_Abs[OF BNF_Composition.type_definition_id_bnf_UNIV]] ..$

lemma $n2m_ctor_fold_G_F:$

$n2m_ctor_fold_G_F s1 s2 o ctor_F = s2 o map_pre_GF_M id (n2m_ctor_fold_G s1 s2) (n2m_ctor_fold_G_F s1 s2) o BNF_Composition.id_bnf o BNF_Composition.id_bnf$

unfolding $n2m_ctor_fold_G_F_def map_pre_F_def map_pre_G_M_def map_pre_GF_M_def$

$F_ctor_o_fold id_apply comp_id id_comp comp_assoc$

$rewriteL_comp_comp[OF F0.map_comp0[symmetric]]$

$rewriteL_comp_comp[OF type_copy_Rep_o_Abs[OF BNF_Composition.type_definition_id_bnf_UNIV]] ..$

7.3.2 Recursors

definition $n2m_ctor_rec_G ::$

$(('a G F \times 'c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow (('a G F \times 'c, 'a G \times 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G \Rightarrow 'b$

where $n2m_ctor_rec_G s1 s2 =$

$ctor_rec_G (s1 o$

$map_pre_G_M id (id :: unit \Rightarrow unit)$

$(BNF_Def.convolve (map_F fst) (ctor_rec_F (s2 o map_pre_GF_M (id :: unit \Rightarrow unit) id (map_prod (map_F$

$fst) id) o BNF_Composition.id_bnf o BNF_Composition.id_bnf))) o$

$BNF_Composition.id_bnf o BNF_Composition.id_bnf)$

definition $n2m_ctor_rec_G_F$::
 $((\lambda a G F \times 'c, 'a) G_M_pre_G_M \Rightarrow 'b) \Rightarrow ((\lambda a G F \times 'c, 'a G \times 'b) GF_M_pre_GF_M \Rightarrow 'c) \Rightarrow 'a G F \Rightarrow 'c$
where $n2m_ctor_rec_G_F s1 s2 = ctor_rec_F (s2 o map_pre_GF_M (id :: unit \Rightarrow unit)) (BNF_Def.convolve id (n2m_ctor_rec_G s1 s2)) id o BNF_Composition.id_bnf o BNF_Composition.id_bnf$

lemma $n2m_ctor_rec_G$:
 $n2m_ctor_rec_G s1 s2 o ctor_G = s1 o map_pre_G_M id id (BNF_Def.convolve id (n2m_ctor_rec_G_F s1 s2))$
 $o BNF_Composition.id_bnf o BNF_Composition.id_bnf$
unfolding $n2m_ctor_rec_G_def n2m_ctor_rec_G_F_def$
 $map_pre_G_def map_pre_F_def map_pre_G_M_def map_pre_GF_M_def$
 $G_ctor_o_rec$
 $id_apply comp_id id_comp comp_assoc map_prod.comp map_prod.id$
 $fst_convolve map_prod_o_convolve convolve_o$
 $rewriteL_comp_comp[OF G0.map_comp0[symmetric]]$
 $rewriteL_comp_comp[OF F0.map_comp0[symmetric]]$
 $F.map_comp0[symmetric] F.map_id0$
 $F.ctor_rec_o_map$
 $rewriteL_comp_comp[OF type_copy_Rep_o_Abs[OF BNF_Composition.type_definition_id_bnf_UNIV]] ..$

lemma $n2m_ctor_rec_G_F$:
 $n2m_ctor_rec_G_F s1 s2 o ctor_F = s2 o map_pre_GF_M id (BNF_Def.convolve id (n2m_ctor_rec_G s1 s2))$
 $(BNF_Def.convolve id (n2m_ctor_rec_G_F s1 s2)) o BNF_Composition.id_bnf o BNF_Composition.id_bnf$
unfolding $n2m_ctor_rec_G_F_def map_pre_F_def map_pre_G_M_def map_pre_GF_M_def$
 $F_ctor_o_rec id_apply comp_id id_comp comp_assoc$
 $rewriteL_comp_comp[OF F0.map_comp0[symmetric]]$
 $rewriteL_comp_comp[OF type_copy_Rep_o_Abs[OF BNF_Composition.type_definition_id_bnf_UNIV]] ..$

7.3.3 Induction

lemma $n2m_rel_induct_G_G_F$:
assumes $IH1: \forall x y. BNF_Def.vimage2p (BNF_Composition.id_bnf o BNF_Composition.id_bnf) (BNF_Composition.id_bnf o BNF_Composition.id_bnf) (rel_pre_G_M P R S) x y \longrightarrow R (ctor_G x) (ctor_G y)$
and $IH2: \forall x y. BNF_Def.vimage2p (BNF_Composition.id_bnf o BNF_Composition.id_bnf) (BNF_Composition.id_bnf o BNF_Composition.id_bnf) (rel_pre_GF_M P R S) x y \longrightarrow S (ctor_F x) (ctor_F y)$
shows $rel_G P \leq R \wedge rel_F (rel_G P) \leq S$
apply (rule context_conjI)
apply (rule G.ctor_rel_induct[unfolded rel_pre_G_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF IH1], OF vimage2p_mono[OF _ pre_G_M.rel_mono], unfolded vimage2p_def o_apply rel_pre_G_M_def type_definition.Abs_inverse[OF BNF_Composition.type_definition_id_bnf_UNIV UNIV_I]])
apply (rule order_refl)
apply (rule order_refl)
apply (rule F.ctor_rel_induct[unfolded rel_pre_F_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF IH2], unfolded vimage2p_def o_apply rel_pre_GF_M_def type_definition.Abs_inverse[OF BNF_Composition.type_definition_id_bnf_UNIV UNIV_I]])
apply (rule F.ctor_rel_induct[unfolded rel_pre_F_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF IH2], OF vimage2p_mono[OF _ pre_GF_M.rel_mono], unfolded vimage2p_def o_apply rel_pre_GF_M_def type_definition.Abs_inverse[OF BNF_Composition.type_definition_id_bnf_UNIV UNIV_I]])
apply (rule order_refl)
apply assumption
apply (rule order_refl)
done

lemmas $n2m_ctor_induct_G_G_F = spec[OF spec [OF$
 $n2m_rel_induct_G_G_F[of (=) BNF_Def.Grp (Collect R) id BNF_Def.Grp (Collect S) id for R S,$
 $unfolded G.rel_eq F.rel_eq eq_le_Grp_id_iff all_simps(1,2)[symmetric]]],$
 $unfolded eq_alt pre_G_M.rel_Grp pre_GF_M.rel_Grp pre_G_M.map_id0 pre_GF_M.map_id0,$
 $unfolded vimage2p_comp vimage2p_id comp_apply comp_id Grp_id_mono_subst$
 $type_copy_vimage2p_Grp_Rep[OF BNF_Composition.type_definition_id_bnf_UNIV]$
 $type_copy_Abs_o_Rep[OF BNF_Composition.type_definition_id_bnf_UNIV]$
 $eqTrueI[OF subset_UNIV] simp_thms(22)$
 $atomize_conjL[symmetric] atomize_all[symmetric] atomize_imp[symmetric],$
 $unfolded subset_iff mem_Collect_eq]$

8 Mutual View on Nested Coatypes

bnf-axiomatization (a, b) $coF0$
bnf-axiomatization (a, b) $coG0$

8.1 Nested definition

codatatype a $coF = CcoF (a, a$ $coF) coF0$
codatatype a $coG = CcoG (a, (a$ $coG) coF) coG0$

type-synonym (b, c) $coF_pre_coF = (c, b)$ $coF0$
type-synonym (c, a) $coG_pre_coG = (a, c$ $coF) coG0$

term $dtor_unfold_coF :: (b \Rightarrow (b, c)$ $coF_pre_coF) \Rightarrow b \Rightarrow c$ coF
term $dtor_unfold_coG :: (c \Rightarrow (c, a)$ $coG_pre_coG) \Rightarrow c \Rightarrow a$ coG
term $dtor_corec_coF :: (b \Rightarrow (c$ $coF + b, c) coF_pre_coF) \Rightarrow b \Rightarrow c$ coF
term $dtor_corec_coG :: (c \Rightarrow (a$ $coG + c, a) coG_pre_coG) \Rightarrow c \Rightarrow a$ coG
thm $coF.dtor_rel_coinduct$
thm $coG.dtor_rel_coinduct[unfolded rel_pre_coG_def id_apply]$

8.2 Isomorphic Mutual Definition

codatatype a $coG_M = CcoG (a, a$ $coGcoF_M) coG0$
and a $coGcoF_M = CcoF (a$ coG_M, a $coGcoF_M) coF0$

type-synonym (b, c) $coGcoF_M_pre_coGcoF_M = (c, b)$ $coF0$
type-synonym (c, a) $coG_M_pre_coG_M = (a, c)$ $coG0$

term $dtor_unfold_coG_M :: (b \Rightarrow (c, a)$ $coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (c, b)$ $coGcoF_M_pre_coGcoF_M) \Rightarrow b \Rightarrow a$ coG_M
term $dtor_unfold_coGcoF_M :: (b \Rightarrow (c, a)$ $coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (c, b)$ $coGcoF_M_pre_coGcoF_M) \Rightarrow c \Rightarrow a$ $coGcoF_M$
term $dtor_corec_coG_M :: (b \Rightarrow (a$ $coGcoF_M + c, a) coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (a$ $coGcoF_M + c, a$ $coG_M + b) coGcoF_M_pre_coGcoF_M) \Rightarrow b \Rightarrow a$ coG_M
term $dtor_corec_coGcoF_M :: (b \Rightarrow (a$ $coGcoF_M + c, a) coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (a$ $coGcoF_M + c, a$ $coG_M + b) coGcoF_M_pre_coGcoF_M) \Rightarrow c \Rightarrow a$ $coGcoF_M$
thm $coG_M_coGcoF_M.dtor_rel_coinduct[unfolded rel_pre_coG_M_def rel_pre_coGcoF_M_def]$

8.3 Mutualization

8.3.1 Coiterators

definition $n2m_dtor_unfold_coG :: (b \Rightarrow (c, a)$ $coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (c, b)$ $coGcoF_M_pre_coGcoF_M) \Rightarrow b \Rightarrow a$ coG

where $n2m_dtor_unfold_coG$ $s1$ $s2 = dtor_unfold_coG (BNF_Composition.id_bnf o BNF_Composition.id_bnf o map_pre_coG_M id (id :: unit \Rightarrow unit) (dtor_unfold_coF (BNF_Composition.id_bnf o BNF_Composition.id_bnf o s2)) o s1)$

definition $n2m_dtor_unfold_coG_coF :: (b \Rightarrow (c, a)$ $coG_M_pre_coG_M) \Rightarrow (c \Rightarrow (c, b)$ $coGcoF_M_pre_coGcoF_M) \Rightarrow c \Rightarrow a$ coG coF

where $n2m_dtor_unfold_coG_coF$ $s1$ $s2 = dtor_unfold_coF (BNF_Composition.id_bnf o BNF_Composition.id_bnf o map_pre_coGcoF_M (id :: unit \Rightarrow unit) (n2m_dtor_unfold_coG$ $s1$ $s2) id o s2)$

lemma $coG_dtor_o_unfold: dtor_coG o dtor_unfold_coG$ $s = map_pre_coG$ $id (dtor_unfold_coG$ $s) o s$
unfolding fun_eq_iff o_apply $coG.dtor_unfold$ **by** $simp$

lemma $coF_dtor_o_unfold: dtor_coF o dtor_unfold_coF$ $s = map_pre_coF$ $id (dtor_unfold_coF$ $s) o s$
unfolding fun_eq_iff o_apply $coF.dtor_unfold$ **by** $simp$

lemma $coG_dtor_o_corec: dtor_coG o dtor_corec_coG$ $s = map_pre_coG$ $id (case_sum$ $id (dtor_corec_coG$ $s)) o s$

unfolding fun_eq_iff o_apply $coG.dtor_corec$ **by** $simp$

lemma $coF_dtor_o_corec: dtor_coF o dtor_corec_coF$ $s = map_pre_coF$ $id (case_sum$ $id (dtor_corec_coF$ $s)) o s$

unfolding fun_eq_iff o_apply $coF.dtor_corec$ **by** $simp$

lemma $n2m_dctor_unfold_coG$:

$dctor_coG \circ n2m_dctor_unfold_coG \ s1 \ s2 = BNF_Composition.id_bnf \circ BNF_Composition.id_bnf \circ map_pre_coG_M$
 $id \ id \ (n2m_dctor_unfold_coG_coF \ s1 \ s2) \ o \ s1$

unfolding $n2m_dctor_unfold_coG_def \ n2m_dctor_unfold_coG_coF_def$
 $map_pre_coG_def \ map_pre_coF_def \ map_pre_coG_M_def \ map_pre_coGcoF_M_def$
 $coG_dctor_o_unfold \ id \ apply \ comp_id \ id \ comp \ comp_assoc$
 $rewriteL_comp_comp[OF \ type_copy_map_comp0_undo[OF \ BNF_Composition.type_definition_id_bnf_UNIV$
 $BNF_Composition.type_definition_id_bnf_UNIV \ BNF_Composition.type_definition_id_bnf_UNIV \ pre_coG_M.map_comp0[unfo$
 $map_pre_coG_M_def]]]$
 $coF.dctor_unfold_o_map$
 $rewriteL_comp_comp[OF \ type_copy_Rep_o_Abs[OF \ BNF_Composition.type_definition_id_bnf_UNIV]] \ ..$

lemma $n2m_dctor_unfold_coG_coF$:

$dctor_coF \circ n2m_dctor_unfold_coG_coF \ s1 \ s2 = BNF_Composition.id_bnf \circ BNF_Composition.id_bnf \circ map_pre_coGcoF_M$
 $id \ (n2m_dctor_unfold_coG \ s1 \ s2) \ (n2m_dctor_unfold_coG_coF \ s1 \ s2) \ o \ s2$

unfolding $n2m_dctor_unfold_coG_coF_def \ map_pre_coF_def \ map_pre_coG_M_def \ map_pre_coGcoF_M_def$
 $coF_dctor_o_unfold \ id \ apply \ comp_id \ id \ comp \ comp_assoc$
 $rewriteL_comp_comp[OF \ coF0.map_comp0[symmetric]]$
 $rewriteL_comp_comp[OF \ type_copy_Rep_o_Abs[OF \ BNF_Composition.type_definition_id_bnf_UNIV]] \ ..$

8.3.2 Corecursors

definition $n2m_dctor_corec_coG \ ::$

$('b \Rightarrow ('a \ coG \ coF + 'c, 'a) \ coG_M_pre_coG_M) \Rightarrow ('c \Rightarrow ('a \ coG \ coF + 'c, 'a \ coG + 'b) \ coGcoF_M_pre_coGcoF_M)$
 $\Rightarrow 'b \Rightarrow 'a \ coG$

where $n2m_dctor_corec_coG \ s1 \ s2 =$
 $dctor_corec_coG \ (BNF_Composition.id_bnf \ o \ BNF_Composition.id_bnf \ o$
 $map_pre_coG_M \ id \ (id \ :: \ unit \Rightarrow \ unit)$
 $(case_sum \ (map_coF \ Inl) \ (dctor_corec_coF \ (BNF_Composition.id_bnf \ o \ BNF_Composition.id_bnf \ o$
 $map_pre_coGcoF_M \ (id \ :: \ unit \Rightarrow \ unit) \ id \ (map_sum \ (map_coF \ Inl) \ id) \ o \ s2))) \ o$
 $s1)$

definition $n2m_dctor_corec_coG_coF \ ::$

$('b \Rightarrow ('a \ coG \ coF + 'c, 'a) \ coG_M_pre_coG_M) \Rightarrow ('c \Rightarrow ('a \ coG \ coF + 'c, 'a \ coG + 'b) \ coGcoF_M_pre_coGcoF_M)$
 $\Rightarrow 'c \Rightarrow 'a \ coG \ coF$

where $n2m_dctor_corec_coG_coF \ s1 \ s2 = dctor_corec_coF \ (BNF_Composition.id_bnf \ o \ BNF_Composition.id_bnf$
 $o \ map_pre_coGcoF_M \ (id \ :: \ unit \Rightarrow \ unit) \ (case_sum \ id \ (n2m_dctor_corec_coG \ s1 \ s2)) \ id \ o \ s2)$

lemma $n2m_dctor_corec_coG$:

$dctor_coG \circ n2m_dctor_corec_coG \ s1 \ s2 = BNF_Composition.id_bnf \circ BNF_Composition.id_bnf \circ map_pre_coG_M$
 $id \ id \ (case_sum \ id \ (n2m_dctor_corec_coG_coF \ s1 \ s2)) \ o \ s1$

unfolding $n2m_dctor_corec_coG_def \ n2m_dctor_corec_coG_coF_def$
 $map_pre_coG_def \ map_pre_coF_def \ map_pre_coG_M_def \ map_pre_coGcoF_M_def$
 $coG_dctor_o_corec$
 $id \ apply \ comp_id \ id \ comp \ comp_assoc[symmetric] \ map_sum.comp \ map_sum.id$
 $case_sum_o_inj(1) \ case_sum_o_map_sum \ o \ case_sum$
 $rewriteR_comp_comp[OF \ coG0.map_comp0[symmetric]]$
 $rewriteR_comp_comp[OF \ coF0.map_comp0[symmetric]]$
 $coF.map_comp0[symmetric] \ coF.map_id0$
 $coF.dctor_corec_o_map$
 $rewriteR_comp_comp[OF \ type_copy_Rep_o_Abs[OF \ BNF_Composition.type_definition_id_bnf_UNIV]] \ ..$

lemma $n2m_dctor_corec_coG_coF$:

$dctor_coF \circ n2m_dctor_corec_coG_coF \ s1 \ s2 = BNF_Composition.id_bnf \circ BNF_Composition.id_bnf \circ map_pre_coGcoF_M$
 $id \ (case_sum \ id \ (n2m_dctor_corec_coG \ s1 \ s2)) \ (case_sum \ id \ (n2m_dctor_corec_coG_coF \ s1 \ s2)) \ o \ s2$

unfolding $n2m_dctor_corec_coG_coF_def \ map_pre_coF_def \ map_pre_coG_M_def \ map_pre_coGcoF_M_def$
 $coF_dctor_o_corec \ id \ apply \ comp_id \ id \ comp \ comp_assoc$
 $rewriteL_comp_comp[OF \ coF0.map_comp0[symmetric]]$
 $rewriteL_comp_comp[OF \ type_copy_Rep_o_Abs[OF \ BNF_Composition.type_definition_id_bnf_UNIV]] \ ..$

8.3.3 Coinduction

lemma $n2m_rel_coinduct_coG_coG_coF$:

```

assumes CIH1:  $\forall x y. R x y \longrightarrow \text{BNF\_Def.vimage2p} (\text{BNF\_Composition.id\_bnf } o \text{ BNF\_Composition.id\_bnf})$ 
( $\text{BNF\_Composition.id\_bnf } o \text{ BNF\_Composition.id\_bnf}$ ) ( $\text{rel\_pre\_coG}_M P R S$ ) ( $\text{dctor\_coG } x$ ) ( $\text{dctor\_coG } y$ )
and CIH2:  $\forall x y. S x y \longrightarrow \text{BNF\_Def.vimage2p} (\text{BNF\_Composition.id\_bnf } o \text{ BNF\_Composition.id\_bnf})$ 
( $\text{BNF\_Composition.id\_bnf } o \text{ BNF\_Composition.id\_bnf}$ ) ( $\text{rel\_pre\_coGcoF}_M P R S$ ) ( $\text{dctor\_coF } x$ ) ( $\text{dctor\_coF } y$ )
shows  $R \leq \text{rel\_coG } P \wedge S \leq \text{rel\_coF} (\text{rel\_coG } P)$ 
apply (rule context_conjI)
apply (rule coG.dctor_rel_coinduct[unfolded rel_pre_coG_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF CIH1], THEN vimage2p_mono[OF _ pre_coG_M.rel_mono], unfolded vimage2p_def
o_apply rel_pre_coG_M_def type_definition.Abs_inverse[OF BNF_Composition.type_definition_id_bnf_UNIV_UNIV_I]])
apply (rule order_refl)
apply (rule order_refl)
apply (rule coF.dctor_rel_coinduct[unfolded rel_pre_coF_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF CIH2], unfolded vimage2p_def o_apply rel_pre_coGcoF_M_def type_definition.Abs_inverse[OF
BNF_Composition.type_definition_id_bnf_UNIV_UNIV_I]])

apply (rule coF.dctor_rel_coinduct[unfolded rel_pre_coF_def id_apply vimage2p_def o_apply])
apply (erule mp[OF spec2[OF CIH2], THEN vimage2p_mono[OF _ pre_coGcoF_M.rel_mono], unfolded vimage2p_def o_apply rel_pre_coGcoF_M_def type_definition.Abs_inverse[OF BNF_Composition.type_definition_id_bnf_UNIV
UNIV_I]])
apply (rule order_refl)
apply assumption
apply (rule order_refl)
done

lemmas n2m_ctor_induct_coG_coG_coF = spec[OF spec[OF spec[OF spec[OF
n2m_rel_coinduct_coG_coG_coF[of _ (=),
unfolded coG.rel_eq coF.rel_eq le_fun_def le_bool_def all_simps(1,2)[symmetric]]]]]]]]

```