Bounded Natural Functors with Covariance and Contravariance

Andreas Lochbihler and Joshua Schneider

June 16, 2019

Abstract

Bounded natural functors (BNFs) provide a modular framework for the construction of (co)datatypes in higher-order logic. Their functorial operations, the mapper and relator, are restricted to a subset of the parameters, namely those where recursion can take place. For certain applications, such as free theorems, data refinement, quotients, and generalised rewriting, it is desirable that these operations do not ignore the other parameters. In this article, we formalise the generalisation $\mathrm{BNF}_{\mathrm{CC}}$ [2] that extends the mapper and relator to covariant and contravariant parameters. We show that (i) $\mathrm{BNF}_{\mathrm{CC}}$s are closed under functor composition and least and greatest fixpoints, (ii) subtypes inherit the $\mathrm{BNF}_{\mathrm{CC}}$ structure under conditions that generalise those for the BNF case, and (iii) $\mathrm{BNF}_{\mathrm{CC}}$s preserve quotients under mild conditions. These proofs are carried out for abstract $\mathrm{BNF}_{\mathrm{CC}}$s similar to the AFP entry BNF Operations [1]. In addition, we apply the $\mathrm{BNF}_{\mathrm{CC}}$ theory to several concrete functors.

For an informal description of the abstract proofs see [2].

Contents

1 Preliminaries

theory *Preliminaries* **imports**
Main
begin

alias *Grp* = *BNF-Def.Grp*
alias *vimage2p* = *BNF-Def.vimage2p*

lemma *Domainp-conversep*: *Domainp* R^{-1-1} = *Rangep R*
by *auto*

lemma *Grp-apply*: *Grp A f x y* $\longleftrightarrow$ $y = f\ x \wedge x \in A$
by (*simp add*: *Grp-def*)

lemma *conversep-Grp-id*: (*Grp A id*)$^{-1-1}$ = *Grp A id*
by (*auto simp add*: *fun-eq-iff Grp-apply*)

lemma *eq-onp-compp-Grp*: *eq-onp P OO Grp A f* = *Grp* (*Collect P* $\cap$ *A*) *f*
by (*auto simp add*: *fun-eq-iff eq-onp-def elim*: *GrpE intro*: *GrpI*)

consts *relcompp-witness* :: ('*a* $\Rightarrow$ '*b* $\Rightarrow$ *bool*) $\Rightarrow$ ('*b* $\Rightarrow$ '*c* $\Rightarrow$ *bool*) $\Rightarrow$ '*a* $\times$ '*c* $\Rightarrow$ '*b*

specification (*relcompp-witness*)
relcompp-witness1: (*A OO B*) (*fst xy*) (*snd xy*) $\Longrightarrow$ *A* (*fst xy*) (*relcompp-witness A B xy*)
relcompp-witness2: (*A OO B*) (*fst xy*) (*snd xy*) $\Longrightarrow$ *B* (*relcompp-witness A B xy*) (*snd xy*)
apply(*fold all-conj-distrib*)
apply(*rule choice allI*)+
apply(*auto*)
done

lemmas *relcompp-witness*[*of* - - (*x*, *y*) **for** *x y*, *simplified*] = *relcompp-witness1 relcompp-witness2*

hide-fact (**open**) *relcompp-witness1 relcompp-witness2*

lemma *relcompp-witness-eq* [*simp*]: *relcompp-witness* (=) (=) (*x*, *x*) = *x*
using *relcompp-witness*(*1*)[*of* (=) (=) *x x*] **by** (*simp add*: *eq-OO*)

lemma *Quotient-equiv-abs1*: $[\![$ *Quotient R Abs Rep T*; *R x y* $]\!] \Longrightarrow$ *T x* (*Abs y*)
unfolding *Quotient-alt-def2* **by** *blast*

lemma *Quotient-equiv-abs2*: $[\![$ *Quotient R Abs Rep T*; *R x y* $]\!] \Longrightarrow$ *T y* (*Abs x*)
unfolding *Quotient-alt-def2* **by** *blast*

lemma *Quotient-rep-equiv1*: ⟦ *Quotient R Abs Rep T*; *T a b* ⟧ ⟹ *R a* (*Rep b*)
unfolding *Quotient-alt-def3* **by** *blast*

lemma *Quotient-rep-equiv2*: ⟦ *Quotient R Abs Rep T*; *T a b* ⟧ ⟹ *R* (*Rep b*) *a*
unfolding *Quotient-alt-def3* **by** *blast*

end

2 Axiomatisation

theory *Axiomatised-BNF-CC* **imports**
Preliminaries
HOL−Library.Rewrite
begin

unbundle *cardinal-syntax*

This theory axiomatises two $\mathrm{BNF_{CC}}$s, which will be used to demonstrate the closedness of $\mathrm{BNF_{CC}}$s under various operations.

2.1 First abstract $\mathrm{BNF_{CC}}$

2.1.1 Axioms and basic definitions

typedecl (*'l1*, *'l2*, *'l3*, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F*

F has each three live, co-, and contravariant parameters, and one fixed parameter.

consts
rel-F :: (*'l1* ⇒ *'l1'* ⇒ *bool*) ⇒ (*'l2* ⇒ *'l2'* ⇒ *bool*) ⇒ (*'l3* ⇒ *'l3'* ⇒ *bool*) ⇒
(*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co3* ⇒ *'co3'* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒
(*'contra3* ⇒ *'contra3'* ⇒ *bool*) ⇒
(*'l1*, *'l2*, *'l3*, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F* ⇒
(*'l1'*, *'l2'*, *'l3'*, *'co1'*, *'co2'*, *'co3'*, *'contra1'*, *'contra2'*, *'contra3'*, *'f*) *F* ⇒ *bool*
map-F :: (*'l1* ⇒ *'l1'*) ⇒ (*'l2* ⇒ *'l2'*) ⇒ (*'l3* ⇒ *'l3'*) ⇒
(*'co1* ⇒ *'co1'*) ⇒ (*'co2* ⇒ *'co2'*) ⇒ (*'co3* ⇒ *'co3'*) ⇒
(*'contra1'* ⇒ *'contra1*) ⇒ (*'contra2'* ⇒ *'contra2*) ⇒ (*'contra3'* ⇒ *'contra3*) ⇒
(*'l1*, *'l2*, *'l3*, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F* ⇒
(*'l1'*, *'l2'*, *'l3'*, *'co1'*, *'co2'*, *'co3'*, *'contra1'*, *'contra2'*, *'contra3'*, *'f*) *F*

axiomatization where
rel-F-mono [*mono*]:
⋀*L1 L1' L2 L2' L3 L3' Co1 Co1' Co2 Co2' Co3 Co3'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3'.
⟦ *L1* ≤ *L1'*; *L2* ≤ *L2'*; *L3* ≤ *L3'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Co3* ≤ *Co3'*;
Contra1' ≤ *Contra1*; *Contra2'* ≤ *Contra2*; *Contra3'* ≤ *Contra3* ⟧ ⟹
rel-F L1 L2 L3 Co1 Co2 Co3 Contra1 Contra2 Contra3 ≤
rel-F L1' L2' L3' Co1' Co2' Co3' Contra1' Contra2' Contra3' **and**

rel-F-eq: *rel-F* (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=) **and**
rel-F-conversep: $\bigwedge$*L1 L2 L3 Co1 Co2 Co3 Contra1 Contra2 Contra3*.
rel-F L1^{-1-1} *L2*$^{-1-1}$ *L3*$^{-1-1}$ *Co1*$^{-1-1}$ *Co2*$^{-1-1}$ *Co3*$^{-1-1}$ *Contra1*$^{-1-1}$ *Contra2*$^{-1-1}$ *Contra3*$^{-1-1}$ =
(*rel-F L1 L2 L3 Co1 Co2 Co3 Contra1 Contra2 Contra3*)$^{-1-1}$ **and**
map-F-id0: *map-F id id id id id id id id id* = *id* **and**
map-F-comp: $\bigwedge$*l1 l1' l2 l2' l3 l3' co1 co1' co2 co2' co3 co3'*
contra1 contra1' contra2 contra2' contra3 contra3'.
map-F l1 l2 l3 co1 co2 co3 contra1 contra2 contra3 ∘
map-F l1' l2' l3' co1' co2' co3' contra1' contra2' contra3' =
map-F (*l1* ∘ *l1'*) (*l2* ∘ *l2'*) (*l3* ∘ *l3'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*co3* ∘ *co3'*)
(*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*) (*contra3'* ∘ *contra3*) **and**
map-F-parametric:
$\bigwedge$*L1 L1' L2 L2' L3 L3' Co1 Co1' Co2 Co2' Co3 Co3'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3'.
rel-fun (*rel-fun L1 L1'*) (*rel-fun* (*rel-fun L2 L2'*) (*rel-fun* (*rel-fun L3 L3'*)
(*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*) (*rel-fun* (*rel-fun Co3 Co3'*)
(*rel-fun* (*rel-fun Contra1' Contra1*) (*rel-fun* (*rel-fun Contra2' Contra2*)
(*rel-fun* (*rel-fun Contra3' Contra3*)
(*rel-fun* (*rel-F L1 L2 L3 Co1 Co2 Co3 Contra1 Contra2 Contra3*)
(*rel-F L1' L2' L3' Co1' Co2' Co3' Contra1' Contra2' Contra3'*))))))))))
map-F map-F

definition *rel-F-pos-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'co3* ⇒ *'co3'* ⇒ *bool*) ⇒ (*'co3'* ⇒ *'co3''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'contra3* ⇒ *'contra3'* ⇒ *bool*) ⇒ (*'contra3'* ⇒ *'contra3''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'l3* × *'l3'* × *'l3''* × *'f*) *itself* ⇒ *bool*
where
rel-F-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' - ⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*)
(*L2* :: *'l2* ⇒ *'l2'* ⇒ *bool*) (*L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*)
(*L3* :: *'l3* ⇒ *'l3'* ⇒ *bool*) (*L3'* :: *'l3'* ⇒ *'l3''* ⇒ *bool*).
(*rel-F L1 L2 L3 Co1 Co2 Co3 Contra1 Contra2 Contra3* ::
(-, -, -, -, -, -, -, -, *'f*) *F* ⇒ -) *OO*
rel-F L1' L2' L3' Co1' Co2' Co3' Contra1' Contra2' Contra3' ≤
rel-F (*L1 OO L1'*) (*L2 OO L2'*) (*L3 OO L3'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*))

definition *rel-F-neg-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'co3* ⇒ *'co3'* ⇒ *bool*) ⇒ (*'co3'* ⇒ *'co3''* ⇒ *bool*) ⇒

$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('contra3 \Rightarrow 'contra3' \Rightarrow bool) \Rightarrow ('contra3' \Rightarrow 'contra3'' \Rightarrow bool) \Rightarrow$
$('l1 \times 'l1' \times 'l1'' \times 'l2 \times 'l2' \times 'l2'' \times 'l3 \times 'l3' \times 'l3'' \times 'f)$ *itself* $\Rightarrow$ *bool*
where
rel-F-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' - $\longleftrightarrow$
$(\forall (L1 :: 'l1 \Rightarrow 'l1' \Rightarrow bool)\ (L1' :: 'l1' \Rightarrow 'l1'' \Rightarrow bool)$
$(L2 :: 'l2 \Rightarrow 'l2' \Rightarrow bool)\ (L2' :: 'l2' \Rightarrow 'l2'' \Rightarrow bool)$
$(L3 :: 'l3 \Rightarrow 'l3' \Rightarrow bool)\ (L3' :: 'l3' \Rightarrow 'l3'' \Rightarrow bool).$
rel-F $(L1\ OO\ L1')\ (L2\ OO\ L2')\ (L3\ OO\ L3')\ (Co1\ OO\ Co1')\ (Co2\ OO\ Co2')$
$(Co3\ OO\ Co3')$
$(Contra1\ OO\ Contra1')\ (Contra2\ OO\ Contra2')\ (Contra3\ OO\ Contra3') \leq$
$(rel\text{-}F\ L1\ L2\ L3\ Co1\ Co2\ Co3\ Contra1\ Contra2\ Contra3 ::$
$(-, -, -, -, -, -, -, -, -, 'f)\ F \Rightarrow -)\ OO$
rel-F L1' L2' L3' Co1' Co2' Co3' Contra1' Contra2' Contra3')

axiomatization where
rel-F-pos-distr-cond-eq:
$\bigwedge tytok.$ *rel-F-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
and
rel-F-neg-distr-cond-eq:
$\bigwedge tytok.$ *rel-F-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*

Restrictions to live variables.

definition *rell-F L1 L2 L3 = rel-F L1 L2 L3* (=) (=) (=) (=) (=) (=)
definition *mapl-F l1 l2 l3 = map-F l1 l2 l3 id id id id id id*

typedecl $('co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)$ *Fbd*

consts
set1-F :: $('l1, 'l2, 'l3, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)\ F \Rightarrow 'l1$ *set*
set2-F :: $('l1, 'l2, 'l3, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)\ F \Rightarrow 'l2$ *set*
set3-F :: $('l1, 'l2, 'l3, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)\ F \Rightarrow 'l3$ *set*
bd-F :: $('co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)$ *Fbd rel*

axiomatization where
set1-F-map: $\bigwedge l1\ l2\ l3.$ *set1-F* $\circ$ *mapl-F l1 l2 l3 = image l1* $\circ$ *set1-F* **and**
set2-F-map: $\bigwedge l1\ l2\ l3.$ *set2-F* $\circ$ *mapl-F l1 l2 l3 = image l2* $\circ$ *set2-F* **and**
set3-F-map: $\bigwedge l1\ l2\ l3.$ *set3-F* $\circ$ *mapl-F l1 l2 l3 = image l3* $\circ$ *set3-F* **and**
bd-F-card-order: *card-order bd-F* **and**
bd-F-cinfinite: *cinfinite bd-F* **and**
set1-F-bound: $\bigwedge x$ :: $(-, -, -, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f)\ F.$
card-of $(set1\text{-}F\ x) \leq o$ $(bd\text{-}F :: ('co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3,$

'f) Fbd rel) **and**
set2-F-bound: $\bigwedge x$ :: (-, -, -, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f) F.
card-of (set2-F x) ≤o (bd-F :: ('co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f) Fbd rel) **and**
set3-F-bound: $\bigwedge x$ :: (-, -, -, 'co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f) F.
card-of (set3-F x) ≤o (bd-F :: ('co1, 'co2, 'co3, 'contra1, 'contra2, 'contra3, 'f) Fbd rel) **and**
mapl-F-cong: $\bigwedge$l1 l1' l2 l2' l3 l3' x.
⟦ $\bigwedge$z. z ∈ set1-F x ⟹ l1 z = l1' z; $\bigwedge$z. z ∈ set2-F x ⟹ l2 z = l2' z;
$\bigwedge$z. z ∈ set3-F x ⟹ l3 z = l3' z ⟧ ⟹
mapl-F l1 l2 l3 x = mapl-F l1' l2' l3' x **and**
rell-F-mono-strong: $\bigwedge$L1 L1' L2 L2' L3 L3' x y.
⟦ rell-F L1 L2 L3 x y;
$\bigwedge$a b. a ∈ set1-F x ⟹ b ∈ set1-F y ⟹ L1 a b ⟹ L1' a b;
$\bigwedge$a b. a ∈ set2-F x ⟹ b ∈ set2-F y ⟹ L2 a b ⟹ L2' a b;
$\bigwedge$a b. a ∈ set3-F x ⟹ b ∈ set3-F y ⟹ L3 a b ⟹ L3' a b ⟧ ⟹
rell-F L1' L2' L3' x y

2.1.2 Derived rules

lemmas rel-F-mono' = rel-F-mono[THEN predicate2D, rotated −1]

lemma rel-F-eq-refl: rel-F (=) (=) (=) (=) (=) (=) (=) (=) (=) x x
by (simp add: rel-F-eq)

lemma map-F-id: map-F id id id id id id id id id x = x
by (simp add: map-F-id0)

lemmas map-F-rel-cong = map-F-parametric[unfolded rel-fun-def, rule-format, rotated −1]

lemma rell-F-mono: ⟦ L1 ≤ L1'; L2 ≤ L2'; L3 ≤ L3' ⟧ ⟹ rell-F L1 L2 L3 ≤ rell-F L1' L2' L3'
unfolding rell-F-def **by** (rule rel-F-mono) (auto)

lemma mapl-F-id0: mapl-F id id id = id
unfolding mapl-F-def **using** map-F-id0 **.**

lemma mapl-F-id: mapl-F id id id x = x
by (simp add: mapl-F-id0)

lemma mapl-F-comp: mapl-F l1 l2 l3 ∘ mapl-F l1' l2' l3' = mapl-F (l1 ∘ l1') (l2 ∘ l2') (l3 ∘ l3')
unfolding mapl-F-def
apply (rule trans)
apply (rule map-F-comp)
apply (simp)
done

lemma *map-F-mapl-F*: *map-F l1 l2 l3 co1 co2 co3 contra1 contra2 contra3 x =*
map-F id id id co1 co2 co3 contra1 contra2 contra3 (*mapl-F l1 l2 l3 x*)
unfolding *mapl-F-def map-F-comp*[*THEN fun-cong*, *simplified*] **by** *simp*

lemma *mapl-F-map-F*: *mapl-F l1 l2 l3* (*map-F id id id co1 co2 co3 contra1 contra2 contra3 x*) =
map-F l1 l2 l3 co1 co2 co3 contra1 contra2 contra3 x
unfolding *mapl-F-def map-F-comp*[*THEN fun-cong*, *simplified*] **by** *simp*

Parametric mappers are unique:

lemma *rel-F-Grp-weak*: *rel-F* (*Grp UNIV l1*) (*Grp UNIV l2*) (*Grp UNIV l3*)
(*Grp UNIV co1*) (*Grp UNIV co2*) (*Grp UNIV co3*)
(*Grp UNIV contra1*)$^{-1-1}$ (*Grp UNIV contra2*)$^{-1-1}$ (*Grp UNIV contra3*)$^{-1-1}$
=
Grp UNIV (*map-F l1 l2 l3 co1 co2 co3 contra1 contra2 contra3*)
apply (*rule antisym*)
apply (*rule predicate2I*)
apply (*rule GrpI*)
apply (*rewrite* **in** - = ⌑ *map-F-id*[*symmetric*])
apply (*subst rel-F-eq*[*symmetric*])
apply (*erule map-F-rel-cong*; *simp add*: *Grp-def*)
apply (*rule UNIV-I*)
apply (*rule predicate2I*)
apply (*erule GrpE*)
apply (*drule sym*)
apply (*hypsubst*)
apply (*rewrite* **in** *rel-F* - - - - - - - - - ⌑ *map-F-id*[*symmetric*])
apply (*rule map-F-rel-cong*)
apply (*rule rel-F-eq-refl*)
apply (*simp-all add*: *Grp-def*)
done

lemmas
rel-F-pos-distr = *rel-F-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-F-neg-distr = *rel-F-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

lemma *rell-F-compp*:
rell-F (*L1 OO L1'*) (*L2 OO L2'*) (*L3 OO L3'*) = *rell-F L1 L2 L3 OO rell-F L1' L2' L3'*
unfolding *rell-F-def*
apply (*rule antisym*)
apply (*rule order-trans*[*rotated*])
apply (*rule rel-F-neg-distr*)
apply (*rule rel-F-neg-distr-cond-eq*)
apply (*simp add*: *eq-OO*)
apply (*rule order-trans*)
apply (*rule rel-F-pos-distr*)
apply (*rule rel-F-pos-distr-cond-eq*)
apply (*simp add*: *eq-OO*)

done

2.1.3 F is a BNF

lemma *rell-F-eq-onp*: *rell-F* (*eq-onp P1*) (*eq-onp P2*) (*eq-onp P3*) =
eq-onp (λ*x*. (∀ *z*∈*set1-F x*. *P1 z*) ∧ (∀ *z*∈*set2-F x*. *P2 z*) ∧ (∀ *z*∈*set3-F x*. *P3 z*))
(**is** *?rel-eq-onp* = *?eq-onp-pred*)
proof (*intro ext iffI*)
fix *x y*
assume *rel*: *?rel-eq-onp x y*
from *rel* **have** *rell-F* (=) (=) (=) *x y*
unfolding *rell-F-def* **by** (*auto elim*: *rel-F-mono′ simp add*: *eq-onp-def*)
then have *y* = *x* **unfolding** *rell-F-def rel-F-eq* **..**
let *?true* = λ-. *True* **and** *?label* = λ*P x*. *P x*
from *rel* **have** *rell-F* (=) (=) (=) (*mapl-F ?true ?true ?true x*)
(*mapl-F* (*?label P1*) (*?label P2*) (*?label P3*) *x*)
unfolding *rell-F-def mapl-F-def* ‹*y* = *x*›
by (*auto simp add*: *eq-onp-def elim*: *map-F-rel-cong*)
then have ∗: *mapl-F ?true ?true ?true x* = *mapl-F* (*?label P1*) (*?label P2*) (*?label P3*) *x*
unfolding *rell-F-def rel-F-eq* **.**
note ‹*y* = *x*›
moreover {
from ∗
have *set1-F* (*mapl-F ?true ?true ?true x*) = *set1-F* (*mapl-F* (*?label P1*) (*?label P2*) (*?label P3*) *x*)
by *simp*
then have *?true ‘ set1-F x* = *?label P1 ‘ set1-F x*
unfolding *set1-F-map*[*THEN fun-cong*, *simplified*] **.**
then have ∀ *z*∈*set1-F x*. *P1 z* **by** *auto*
}
moreover {
from ∗
have *set2-F* (*mapl-F ?true ?true ?true x*) = *set2-F* (*mapl-F* (*?label P1*) (*?label P2*) (*?label P3*) *x*)
by *simp*
then have *?true ‘ set2-F x* = *?label P2 ‘ set2-F x*
unfolding *set2-F-map*[*THEN fun-cong*, *simplified*] **.**
then have ∀ *z*∈*set2-F x*. *P2 z* **by** *auto*
}
moreover {
from ∗
have *set3-F* (*mapl-F ?true ?true ?true x*) = *set3-F* (*mapl-F* (*?label P1*) (*?label P2*) (*?label P3*) *x*)
by *simp*
then have *?true ‘ set3-F x* = *?label P3 ‘ set3-F x*
unfolding *set3-F-map*[*THEN fun-cong*, *simplified*] **.**
then have ∀ *z*∈*set3-F x*. *P3 z* **by** *auto*
}

ultimately show *?eq-onp-pred x y* **by** (*simp add*: *eq-onp-def*)
next
fix *x y*
assume *eq-onp*: *?eq-onp-pred x y*
then have *rell-F* (=) (=) (=) *x y*
by (*auto simp add*: *rell-F-def rel-F-eq eq-onp-def*)
then show *?rel-eq-onp x y*
using *eq-onp* **by** (*auto elim*!: *rell-F-mono-strong simp add*: *eq-onp-def*)
qed

lemma *rell-F-Grp*: *rell-F* (*Grp A1 f1*) (*Grp A2 f2*) (*Grp A3 f3*) =
Grp {*x*. *set1-F x* ⊆ *A1* ∧ *set2-F x* ⊆ *A2* ∧ *set3-F x* ⊆ *A3*} (*mapl-F f1 f2 f3*)
proof −
have *rell-F* (*Grp A1 f1*) (*Grp A2 f2*) (*Grp A3 f3*) = *rell-F* (*eq-onp* (λ*x*. *x* ∈ *A1*) *OO Grp UNIV f1*)
(*eq-onp* (λ*x*. *x* ∈ *A2*) *OO Grp UNIV f2*) (*eq-onp* (λ*x*. *x* ∈ *A3*) *OO Grp UNIV f3*)
by (*simp add*: *eq-onp-compp-Grp*)
also have ... = *rell-F* (*eq-onp* (λ*x*. *x* ∈ *A1*)) (*eq-onp* (λ*x*. *x* ∈ *A2*)) (*eq-onp* (λ*x*. *x* ∈ *A3*)) *OO*
rell-F (*Grp UNIV f1*) (*Grp UNIV f2*) (*Grp UNIV f3*)
using *rell-F-compp* **.**
also have ... = *eq-onp* (λ*x*. *set1-F x* ⊆ *A1* ∧ *set2-F x* ⊆ *A2* ∧ *set3-F x* ⊆ *A3*) *OO*
rell-F (*Grp UNIV f1*) (*Grp UNIV f2*) (*Grp UNIV f3*)
by (*simp add*: *rell-F-eq-onp subset-eq*)
also have ... = *eq-onp* (λ*x*. *set1-F x* ⊆ *A1* ∧ *set2-F x* ⊆ *A2* ∧ *set3-F x* ⊆ *A3*) *OO*
Grp UNIV (*mapl-F f1 f2 f3*)
unfolding *rell-F-def mapl-F-def*
rel-F-Grp-weak[*of* - - - *id id id id id id*, *folded eq-alt*, *simplified*]
..
also have ... = *Grp* {*x*. *set1-F x* ⊆ *A1* ∧ *set2-F x* ⊆ *A2* ∧ *set3-F x* ⊆ *A3*} (*mapl-F f1 f2 f3*)
by (*simp add*: *eq-onp-compp-Grp*)
finally show *?thesis* **.**
qed

lemma *rell-F-compp-Grp*: *rell-F L1 L2 L3* =
(*Grp* {*x*. *set1-F x* ⊆ {(*x*, *y*). *L1 x y*} ∧ *set2-F x* ⊆ {(*x*, *y*). *L2 x y*} ∧ *set3-F x* ⊆ {(*x*, *y*). *L3 x y*}}
(*mapl-F fst fst fst*))$^{-1-1}$ *OO*
Grp {*x*. *set1-F x* ⊆ {(*x*, *y*). *L1 x y*} ∧ *set2-F x* ⊆ {(*x*, *y*). *L2 x y*} ∧ *set3-F x* ⊆ {(*x*, *y*). *L3 x y*}}
(*mapl-F snd snd snd*)
apply (*unfold rell-F-Grp*[*symmetric*])
apply (*unfold rell-F-def*)
apply (*simp add*: *rel-F-conversep*[*symmetric*])
apply (*fold rell-F-def*)

apply (*simp add*: *rell-F-compp*[*symmetric*] *Grp-fst-snd*)
done

lemma *F-in-rell*: *rell-F L1 L2 L3* = (λ*x y*. ∃ *z*. (*set1-F z* ⊆ {(*x*, *y*). *L1 x y*} ∧
set2-F z ⊆ {(*x*, *y*). *L2 x y*} ∧ *set3-F z* ⊆ {(*x*, *y*). *L3 x y*}) ∧
mapl-F fst fst fst z = *x* ∧ *mapl-F snd snd snd z* = *y*)
using *rell-F-compp-Grp* **by** (*simp add*: *OO-Grp-alt*)

bnf ('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*contra1*, '*contra2*, '*contra3*, '*f*) *F*
map: *mapl-F*
sets: *set1-F set2-F set3-F*
bd: *bd-F* :: ('*co1*, '*co2*, '*co3*, '*contra1*, '*contra2*, '*contra3*, '*f*) *Fbd rel*
rel: *rell-F*
by (*fact mapl-F-id0 mapl-F-comp*[*symmetric*] *mapl-F-cong set1-F-map set2-F-map set3-F-map*
bd-F-card-order bd-F-cinfinite set1-F-bound set2-F-bound set3-F-bound
rell-F-compp[*symmetric*, *THEN eq-refl*] *F-in-rell*)+

2.1.4 Composition witness

consts
rel-F-witness :: ('*l1* ⇒ '*l1''* ⇒ *bool*) ⇒ ('*l2* ⇒ '*l2''* ⇒ *bool*) ⇒ ('*l3* ⇒ '*l3''* ⇒ *bool*) ⇒
('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*co3* ⇒ '*co3'* ⇒ *bool*) ⇒ ('*co3'* ⇒ '*co3''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*contra3* ⇒ '*contra3'* ⇒ *bool*) ⇒ ('*contra3'* ⇒ '*contra3''* ⇒ *bool*) ⇒
('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*contra1*, '*contra2*, '*contra3*, '*f*) *F* ×
('*l1''*, '*l2''*, '*l3''*, '*co1''*, '*co2''*, '*co3''*, '*contra1''*, '*contra2''*, '*contra3''*, '*f*) *F* ⇒
('*l1* × '*l1''*, '*l2* × '*l2''*, '*l3* × '*l3''*, '*co1'*, '*co2'*, '*co3'*, '*contra1'*, '*contra2'*, '*contra3'*,
'*f*) *F*

specification (*rel-F-witness*)
rel-F-witness1: ⋀*L1 L2 L3 Co1 Co1' Co2 Co2' Co3 Co3'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3'
(*tytok* :: ('*l1* × ('*l1* × '*l1''*) × '*l1''* × '*l2* × ('*l2* × '*l2''*) × '*l2''* ×
'*l3* × ('*l3* × '*l3''*) × '*l3''* × '*f*) *itself*)
(*x* :: ('*l1*, '*l2*, '*l3*, -, -, -, -, -, -, '*f*) *F*)
(*y* :: ('*l1''*, '*l2''*, '*l3''*, -, -, -, -, -, -, '*f*) *F*).
⟦ *rel-F-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' tytok;
rel-F L1 L2 L3 (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*)
x y ⟧ ⟹
rel-F (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L1 x y*) (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L2 x y*)
(λ*x* (*x'*, *y*). *x'* = *x* ∧ *L3 x y*) *Co1 Co2 Co3 Contra1 Contra2 Contra3 x*

(*rel-F-witness L1 L2 L3 Co1 Co1′ Co2 Co2′ Co3 Co3′*
Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′ (*x*, *y*))
rel-F-witness2:⋀*L1 L2 L3 Co1 Co1′ Co2 Co2′ Co3 Co3′*
Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′
(*tytok* :: (*'l1* × (*'l1* × *'l1″*) × *'l1″* × *'l2* × (*'l2* × *'l2″*) × *'l2″* ×
'l3 × (*'l3* × *'l3″*) × *'l3″* × *'f*) *itself*)
(*x* :: (*'l1*, *'l2*, *'l3*, -, -, -, -, -, -, *'f*) *F*)
(*y* :: (*'l1″*, *'l2″*, *'l3″*, -, -, -, -, -, -, *'f*) *F*).
⟦ *rel-F-neg-distr-cond Co1 Co1′ Co2 Co2′ Co3 Co3′*
Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′ tytok;
rel-F L1 L2 L3 (*Co1 OO Co1′*) (*Co2 OO Co2′*) (*Co3 OO Co3′*)
(*Contra1 OO Contra1′*) (*Contra2 OO Contra2′*) (*Contra3 OO Contra3′*)
x y ⟧ ⟹
rel-F (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L1 x y*) (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L2 x y*)
(λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L3 x y*) *Co1′ Co2′ Co3′ Contra1′ Contra2′ Contra3′*
(*rel-F-witness L1 L2 L3 Co1 Co1′ Co2 Co2′ Co3 Co3′*
Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′ (*x*, *y*)) *y*
apply(*rule exI*[**where** *x*=λ*L1 L2 L3 Co1 Co1′ Co2 Co2′ Co3 Co3′*
Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′ (*x*, *y*). *SOME z*.
rel-F (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L1 x y*) (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L2 x y*) (λ*x* (*x′*, *y*).
x′ = *x* ∧ *L3 x y*)
Co1 Co2 Co3 Contra1 Contra2 Contra3 x z ∧
rel-F (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L1 x y*) (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L2 x y*) (λ(*x*, *y′*) *y*.
y′ = *y* ∧ *L3 x y*)
Co1′ Co2′ Co3′ Contra1′ Contra2′ Contra3′ z y])
apply(*fold all-conj-distrib*)
apply(*rule allI*)+
apply(*fold imp-conjR*)
apply(*rule impI*)+
apply *clarify*
apply(*rule someI-ex*)
subgoal for *L1 L2 L3 Co1 Co1′ Co2 Co2′ Co3 Co3′ Contra1 Contra1′ Contra2 Contra2′ Contra3 Contra3′ x y*
apply(*drule rel-F-neg-distr*[**where**
?L1.0 = λ*x* (*x′*, *y*). *x′* = *x* ∧ *L1 x y* **and** *?L1′.0* = λ(*x*, *y*) *y′*. *y* = *y′* ∧
L1 x y′ **and**
?L2.0 = λ*x* (*x′*, *y*). *x′* = *x* ∧ *L2 x y* **and** *?L2′.0* = λ(*x*, *y*) *y′*. *y* = *y′* ∧
L2 x y′ **and**
?L3.0 = λ*x* (*x′*, *y*). *x′* = *x* ∧ *L3 x y* **and** *?L3′.0* = λ(*x*, *y*) *y′*. *y* = *y′* ∧
L3 x y′])
apply(*drule predicate2D*)
apply(*erule rel-F-mono*[*THEN predicate2D*, *rotated* −*1*]; *fastforce*)
apply(*erule relcomppE*)
apply(*rule exI conjI*)+
apply *assumption*+
done
done

2.2 Second abstract $\mathbf{BNF_{CC}}$

2.2.1 Axioms and basic definitions

typedecl $('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G$

G has each two live, co, and contravariant parameters, and one fixed parameter.

consts
rel-G :: $('l1 \Rightarrow 'l1' \Rightarrow bool) \Rightarrow ('l2 \Rightarrow 'l2' \Rightarrow bool) \Rightarrow$
$('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow$
$('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G \Rightarrow$
$('l1', 'l2', 'co1', 'co2', 'contra1', 'contra2', 'f)\ G \Rightarrow bool$
map-G :: $('l1 \Rightarrow 'l1') \Rightarrow ('l2 \Rightarrow 'l2') \Rightarrow$
$('co1 \Rightarrow 'co1') \Rightarrow ('co2 \Rightarrow 'co2') \Rightarrow$
$('contra1' \Rightarrow 'contra1) \Rightarrow ('contra2' \Rightarrow 'contra2) \Rightarrow$
$('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G \Rightarrow$
$('l1', 'l2', 'co1', 'co2', 'contra1', 'contra2', 'f)\ G$

axiomatization where
rel-G-mono [*mono*]:
$\bigwedge L1\ L1'\ L2\ L2'\ Co1\ Co1'\ Co2\ Co2'\ Contra1\ Contra1'\ Contra2\ Contra2'.$
$[\![\ L1 \le L1';\ L2 \le L2';\ Co1 \le Co1';\ Co2 \le Co2';\ Contra1' \le Contra1;$
$Contra2' \le Contra2\ ]\!] \Longrightarrow$
$rel\text{-}G\ L1\ L2\ Co1\ Co2\ Contra1\ Contra2 \le rel\text{-}G\ L1'\ L2'\ Co1'\ Co2'\ Contra1'\ Contra2'$ **and**
rel-G-eq: $rel\text{-}G\ (=)\ (=)\ (=)\ (=)\ (=)\ (=) = (=)$ **and**
rel-G-conversep: $\bigwedge L1\ L2\ Co1\ Co2\ Contra1\ Contra2.$
$rel\text{-}G\ L1^{-1-1}\ L2^{-1-1}\ Co1^{-1-1}\ Co2^{-1-1}\ Contra1^{-1-1}\ Contra2^{-1-1} = (rel\text{-}G\ L1\ L2\ Co1\ Co2\ Contra1\ Contra2)^{-1-1}$ **and**
map-G-id0: $map\text{-}G\ id\ id\ id\ id\ id\ id = id$ **and**
map-G-comp: $\bigwedge l1\ l1'\ l2\ l2'\ co1\ co1'\ co2\ co2'\ contra1\ contra1'\ contra2\ contra2'.$
$map\text{-}G\ l1\ l2\ co1\ co2\ contra1\ contra2 \circ map\text{-}G\ l1'\ l2'\ co1'\ co2'\ contra1'\ contra2' =$
$map\text{-}G\ (l1 \circ l1')\ (l2 \circ l2')\ (co1 \circ co1')\ (co2 \circ co2')$
$(contra1' \circ contra1)\ (contra2' \circ contra2)$ **and**
map-G-parametric:
$\bigwedge L1\ L1'\ L2\ L2'\ Co1\ Co1'\ Co2\ Co2'\ Contra1\ Contra1'\ Contra2\ Contra2'.$
$rel\text{-}fun\ (rel\text{-}fun\ L1\ L1')\ (rel\text{-}fun\ (rel\text{-}fun\ L2\ L2')$
$(rel\text{-}fun\ (rel\text{-}fun\ Co1\ Co1')\ (rel\text{-}fun\ (rel\text{-}fun\ Co2\ Co2')$
$(rel\text{-}fun\ (rel\text{-}fun\ Contra1'\ Contra1)\ (rel\text{-}fun\ (rel\text{-}fun\ Contra2'\ Contra2)$
$(rel\text{-}fun\ (rel\text{-}G\ L1\ L2\ Co1\ Co2\ Contra1\ Contra2)$
$(rel\text{-}G\ L1'\ L2'\ Co1'\ Co2'\ Contra1'\ Contra2')))))))$
$map\text{-}G\ map\text{-}G$

definition *rel-G-pos-distr-cond* :: $('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co1' \Rightarrow 'co1'' \Rightarrow bool) \Rightarrow$
$('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow ('co2' \Rightarrow 'co2'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$

(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself* ⇒ *bool* **where**
rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*)
(*L2* :: *'l2* ⇒ *'l2'* ⇒ *bool*) (*L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*).
(*rel-G L1 L2 Co1 Co2 Contra1 Contra2* :: (-, -, -, -, -, -, *'f*) *G* ⇒ -) *OO*
rel-G L1' L2' Co1' Co2' Contra1' Contra2' ≤
rel-G (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*))

definition *rel-G-neg-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself* ⇒ *bool* **where**
rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*)
(*L2* :: *'l2* ⇒ *'l2'* ⇒ *bool*) (*L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*).
rel-G (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) ≤
(*rel-G L1 L2 Co1 Co2 Contra1 Contra2* :: (-, -, -, -, -, -, *'f*) *G* ⇒ -) *OO*
rel-G L1' L2' Co1' Co2' Contra1' Contra2')

axiomatization where
rel-G-pos-distr-cond-eq:
⋀*tytok*. *rel-G-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok* **and**
rel-G-neg-distr-cond-eq:
⋀*tytok*. *rel-G-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*

Restrictions to live variables.

definition *rell-G L1 L2* = *rel-G L1 L2* (=) (=) (=) (=)
definition *mapl-G l1 l2* = *map-G l1 l2 id id id id*

typedecl (*'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *Gbd*

consts
set1-G :: (*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G* ⇒ *'l1 set*
set2-G :: (*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G* ⇒ *'l2 set*
bd-G :: (*'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *Gbd rel*
wit-G :: *'l2* ⇒ (*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G*
— non-emptiness witness for least fixpoint

axiomatization where
set1-G-map: ⋀*l1 l2*. *set1-G* ∘ *mapl-G l1 l2* = *image l1* ∘ *set1-G* **and**
set2-G-map: ⋀*l1 l2*. *set2-G* ∘ *mapl-G l1 l2* = *image l2* ∘ *set2-G* **and**
bd-G-card-order: *card-order bd-G* **and**

bd-G-cinfinite: *cinfinite bd-G* **and**
set1-G-bound: $\bigwedge x$ :: (-, -, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G*.
card-of (*set1-G x*) $\leq o$ (*bd-G* :: (*'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *Gbd rel*) **and**
set2-G-bound: $\bigwedge x$ :: (-, -, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G*.
card-of (*set2-G x*) $\leq o$ (*bd-G* :: (*'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *Gbd rel*) **and**
mapl-G-cong: $\bigwedge$*l1 l1' l2 l2' l3 l3' x*.
$[\![\bigwedge z.\ z \in$ *set1-G x* $\Longrightarrow$ *l1 z* = *l1' z*; $\bigwedge z.\ z \in$ *set2-G x* $\Longrightarrow$ *l2 z* = *l2' z* $]\!] \Longrightarrow$
mapl-G l1 l2 x = *mapl-G l1' l2' x* **and**
rell-G-mono-strong: $\bigwedge$*L1 L1' L2 L2' x y*.
$[\![$ *rell-G L1 L2 x y*;
$\bigwedge a\ b.\ a \in$ *set1-G x* $\Longrightarrow b \in$ *set1-G y* $\Longrightarrow$ *L1 a b* $\Longrightarrow$ *L1' a b*;
$\bigwedge a\ b.\ a \in$ *set2-G x* $\Longrightarrow b \in$ *set2-G y* $\Longrightarrow$ *L2 a b* $\Longrightarrow$ *L2' a b* $]\!] \Longrightarrow$
rell-G L1' L2' x y **and**
wit-G-set1: $\bigwedge$*l2 x*. *x* $\in$ *set1-G* (*wit-G l2*) $\Longrightarrow$ *False* **and**
wit-G-set2: $\bigwedge$*l2 x*. *x* $\in$ *set2-G* (*wit-G l2*) $\Longrightarrow$ *x* = *l2*

2.2.2 Derived rules

lemmas *rel-G-mono'* = *rel-G-mono*[*THEN predicate2D*, *rotated* -1]

lemma *rel-G-eq-refl*: *rel-G* (=) (=) (=) (=) (=) (=) *x x*
by (*simp add*: *rel-G-eq*)

lemma *map-G-id*: *map-G id id id id id id x* = *x*
by (*simp add*: *map-G-id0*)

lemmas *map-G-rel-cong* = *map-G-parametric*[*unfolded rel-fun-def*, *rule-format*, *rotated* -1]

lemma *rell-G-mono*: $[\![$ *L1* $\leq$ *L1'*; *L2* $\leq$ *L2'* $]\!] \Longrightarrow$ *rell-G L1 L2* $\leq$ *rell-G L1' L2'*
unfolding *rell-G-def* **by** (*rule rel-G-mono*) (*auto*)

lemma *mapl-G-id0*: *mapl-G id id* = *id*
unfolding *mapl-G-def* **using** *map-G-id0* **.**

lemma *mapl-G-id*: *mapl-G id id x* = *x*
by (*simp add*: *mapl-G-id0*)

lemma *mapl-G-comp*: *mapl-G l1 l2* $\circ$ *mapl-G l1' l2'* = *mapl-G* (*l1* $\circ$ *l1'*) (*l2* $\circ$ *l2'*)
unfolding *mapl-G-def*
apply (*rule trans*)
apply (*rule map-G-comp*)
apply (*simp*)
done

lemma *map-G-mapl-G*:
map-G l1 l2 co1 co2 contra1 contra2 x = *map-G id id co1 co2 contra1 contra2* (*mapl-G l1 l2 x*)

unfolding *mapl-G-def map-G-comp*[*THEN fun-cong*, *simplified*] **by** *simp*

lemma *mapl-G-map-G*:
mapl-G l1 l2 (*map-G id id co1 co2 contra1 contra2 x*) = *map-G l1 l2 co1 co2 contra1 contra2 x*
unfolding *mapl-G-def map-G-comp*[*THEN fun-cong*, *simplified*] **by** *simp*

Parametric mappers are unique:

lemma *rel-G-Grp-weak*: *rel-G* (*Grp UNIV l1*) (*Grp UNIV l2*) (*Grp UNIV co1*) (*Grp UNIV co2*)
(*Grp UNIV contra1*)$^{-1-1}$ (*Grp UNIV contra2*)$^{-1-1}$ = *Grp UNIV* (*map-G l1 l2 co1 co2 contra1 contra2*)
apply (*rule antisym*)
apply (*rule predicate2I*)
apply (*rule GrpI*)
apply (*rewrite* **in** - = ⌑ *map-G-id*[*symmetric*])
apply (*subst rel-G-eq*[*symmetric*])
apply (*erule map-G-rel-cong*; *simp add*: *Grp-def*)
apply (*rule UNIV-I*)
apply (*rule predicate2I*)
apply (*erule GrpE*)
apply (*drule sym*)
apply (*hypsubst*)
apply (*rewrite* **in** *rel-G* - - - - - - ⌑ *map-G-id*[*symmetric*])
apply (*rule map-G-rel-cong*)
apply (*rule rel-G-eq-refl*)
apply (*simp-all add*: *Grp-def*)
done

lemmas
rel-G-pos-distr = *rel-G-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-G-neg-distr = *rel-G-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

lemma *rell-G-compp*:
rell-G (*L1 OO L1'*) (*L2 OO L2'*) = *rell-G L1 L2 OO rell-G L1' L2'*
unfolding *rell-G-def*
apply (*rule antisym*)
apply (*rule order-trans*[*rotated*])
apply (*rule rel-G-neg-distr*)
apply (*rule rel-G-neg-distr-cond-eq*)
apply (*simp add*: *eq-OO*)
apply (*rule order-trans*)
apply (*rule rel-G-pos-distr*)
apply (*rule rel-G-pos-distr-cond-eq*)
apply (*simp add*: *eq-OO*)
done

2.2.3 G is a BNF

lemma *rell-G-eq-onp*:
rell-G (*eq-onp P1*) (*eq-onp P2*) = *eq-onp* (λ*x*. (∀ *z*∈*set1-G x*. *P1 z*) ∧ (∀ *z*∈*set2-G x*. *P2 z*))
(**is** *?rel-eq-onp* = *?eq-onp-pred*)
proof (*intro ext iffI*)
fix *x y*
assume *rel*: *?rel-eq-onp x y*
from *rel* **have** *rell-G* (=) (=) *x y*
unfolding *rell-G-def* **by** (*auto elim*: *rel-G-mono′ simp add*: *eq-onp-def*)
then have *y* = *x* **unfolding** *rell-G-def rel-G-eq* **..**
let *?true* = λ-. *True* **and** *?label* = λ*P x*. *P x*
from *rel* **have** *rell-G* (=) (=) (*mapl-G ?true ?true x*) (*mapl-G* (*?label P1*) (*?label P2*) *x*)
unfolding *rell-G-def mapl-G-def* ‹*y* = *x*›
by (*auto simp add*: *eq-onp-def elim*: *map-G-rel-cong*)
then have ∗: *mapl-G ?true ?true x* = *mapl-G* (*?label P1*) (*?label P2*) *x*
unfolding *rell-G-def rel-G-eq* **.**
note ‹*y* = *x*›
moreover {
from ∗
have *set1-G* (*mapl-G ?true ?true x*) = *set1-G* (*mapl-G* (*?label P1*) (*?label P2*) *x*)
by *simp*
then have *?true* ‘ *set1-G x* = *?label P1* ‘ *set1-G x*
unfolding *set1-G-map*[*THEN fun-cong*, *simplified*] **.**
then have ∀ *z*∈*set1-G x*. *P1 z* **by** *auto*
}
moreover {
from ∗
have *set2-G* (*mapl-G ?true ?true x*) = *set2-G* (*mapl-G* (*?label P1*) (*?label P2*) *x*)
by *simp*
then have *?true* ‘ *set2-G x* = *?label P2* ‘ *set2-G x*
unfolding *set2-G-map*[*THEN fun-cong*, *simplified*] **.**
then have ∀ *z*∈*set2-G x*. *P2 z* **by** *auto*
}
ultimately show *?eq-onp-pred x y* **by** (*simp add*: *eq-onp-def*)
next
fix *x y*
assume *eq-onp*: *?eq-onp-pred x y*
then have *rell-G* (=) (=) *x y*
by (*auto simp add*: *rell-G-def rel-G-eq eq-onp-def*)
then show *?rel-eq-onp x y*
using *eq-onp* **by** (*auto elim*!: *rell-G-mono-strong simp add*: *eq-onp-def*)
qed

lemma *rell-G-Grp*:
rell-G (*Grp A1 f1*) (*Grp A2 f2*) = *Grp* {*x*. *set1-G x* ⊆ *A1* ∧ *set2-G x* ⊆ *A2*}

(*mapl-G f1 f2*)
proof −
have *rell-G* (*Grp A1 f1*) (*Grp A2 f2*) = *rell-G* (*eq-onp* (λ*x*. *x* ∈ *A1*) *OO Grp UNIV f1*)
(*eq-onp* (λ*x*. *x* ∈ *A2*) *OO Grp UNIV f2*)
by (*simp add*: *eq-onp-compp-Grp*)
also have ... = *rell-G* (*eq-onp* (λ*x*. *x* ∈ *A1*)) (*eq-onp* (λ*x*. *x* ∈ *A2*)) *OO*
rell-G (*Grp UNIV f1*) (*Grp UNIV f2*)
using *rell-G-compp* **.**
also have ... = *eq-onp* (λ*x*. *set1-G x* ⊆ *A1* ∧ *set2-G x* ⊆ *A2*) *OO*
rell-G (*Grp UNIV f1*) (*Grp UNIV f2*)
by (*simp add*: *rell-G-eq-onp subset-eq*)
also have ... = *eq-onp* (λ*x*. *set1-G x* ⊆ *A1* ∧ *set2-G x* ⊆ *A2*) *OO Grp UNIV* (*mapl-G f1 f2*)
unfolding *rell-G-def mapl-G-def*
rel-G-Grp-weak[*of - - id id id id*, *folded eq-alt*, *simplified*]
..
also have ... = *Grp* {*x*. *set1-G x* ⊆ *A1* ∧ *set2-G x* ⊆ *A2*} (*mapl-G f1 f2*)
by (*simp add*: *eq-onp-compp-Grp*)
finally show *?thesis* **.**
qed

lemma *rell-G-compp-Grp*: *rell-G L1 L2* =
(*Grp* {*x*. *set1-G x* ⊆ {(*x*, *y*). *L1 x y*} ∧ *set2-G x* ⊆ {(*x*, *y*). *L2 x y*}} (*mapl-G fst fst*))$^{-1-1}$ *OO*
Grp {*x*. *set1-G x* ⊆ {(*x*, *y*). *L1 x y*} ∧ *set2-G x* ⊆ {(*x*, *y*). *L2 x y*}} (*mapl-G snd snd*)
apply (*unfold rell-G-Grp*[*symmetric*])
apply (*unfold rell-G-def*)
apply (*simp add*: *rel-G-conversep*[*symmetric*])
apply (*fold rell-G-def*)
apply (*simp add*: *rell-G-compp*[*symmetric*] *Grp-fst-snd*)
done

lemma *G-in-rell*: *rell-G L1 L2* = (λ*x y*. ∃ *z*. (*set1-G z* ⊆ {(*x*, *y*). *L1 x y*} ∧
set2-G z ⊆ {(*x*, *y*). *L2 x y*}) ∧ *mapl-G fst fst z* = *x* ∧ *mapl-G snd snd z* = *y*)
using *rell-G-compp-Grp* **by** (*simp add*: *OO-Grp-alt*)

bnf ('*l1*, '*l2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*f*) *G*
map: *mapl-G*
sets: *set1-G set2-G*
bd: *bd-G* :: ('*co1*, '*co2*, '*contra1*, '*contra2*, '*f*) *Gbd rel*
wits: *wit-G*
rel: *rell-G*
by (*fact mapl-G-id0 mapl-G-comp*[*symmetric*] *mapl-G-cong set1-G-map set2-G-map*
bd-G-card-order bd-G-cinfinite set1-G-bound set2-G-bound rell-G-compp[*symmetric*, *THEN eq-refl*]
G-in-rell wit-G-set1 wit-G-set2)+

2.2.4 Composition witness

consts
rel-G-witness :: $('l1 \Rightarrow 'l1'' \Rightarrow bool) \Rightarrow ('l2 \Rightarrow 'l2'' \Rightarrow bool) \Rightarrow$
$('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co1' \Rightarrow 'co1'' \Rightarrow bool) \Rightarrow$
$('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow ('co2' \Rightarrow 'co2'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G \times$
$('l1'', 'l2'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f)\ G \Rightarrow$
$('l1 \times 'l1'', 'l2 \times 'l2'', 'co1', 'co2', 'contra1', 'contra2', 'f)\ G$

specification (*rel-G-witness*)
rel-G-witness1: $\bigwedge$*L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'*
(*tytok* :: $('l1 \times ('l1 \times 'l1'') \times 'l1'' \times 'l2 \times ('l2 \times 'l2'') \times 'l2'' \times 'f)$ *itself*)
(x :: $('l1, 'l2, -, -, -, -, 'f)\ G$) ($y$ :: $('l1'', 'l2'', -, -, -, -, 'f)\ G$).
⟦ *rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*;
rel-G L1 L2 (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) $x\ y$ ⟧ ⟹
rel-G $(\lambda x\ (x', y).\ x' = x \wedge L1\ x\ y)$ $(\lambda x\ (x', y).\ x' = x \wedge L2\ x\ y)$ *Co1 Co2 Contra1 Contra2 x*
(*rel-G-witness L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (x, y))
rel-G-witness2:$\bigwedge$*L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'*
(*tytok* :: $('l1 \times ('l1 \times 'l1'') \times 'l1'' \times 'l2 \times ('l2 \times 'l2'') \times 'l2'' \times 'f)$ *itself*)
(x :: $('l1, 'l2, -, -, -, -, 'f)\ G$) ($y$ :: $('l1'', 'l2'', -, -, -, -, 'f)\ G$).
⟦ *rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*;
rel-G L1 L2 (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) $x\ y$ ⟧ ⟹
rel-G $(\lambda(x, y')\ y.\ y' = y \wedge L1\ x\ y)$ $(\lambda(x, y')\ y.\ y' = y \wedge L2\ x\ y)$ *Co1' Co2' Contra1' Contra2'*
(*rel-G-witness L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (x, y)) y
apply(*rule exI*[**where** x=λ*L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (x, y). *SOME z*.
rel-G $(\lambda x\ (x', y).\ x' = x \wedge L1\ x\ y)$ $(\lambda x\ (x', y).\ x' = x \wedge L2\ x\ y)$ *Co1 Co2 Contra1 Contra2 x z* ∧
rel-G $(\lambda(x, y')\ y.\ y' = y \wedge L1\ x\ y)$ $(\lambda(x, y')\ y.\ y' = y \wedge L2\ x\ y)$ *Co1' Co2' Contra1' Contra2' z y*])
apply(*fold all-conj-distrib*)
apply(*rule allI*)+
apply(*fold imp-conjR*)
apply(*rule impI*)+
apply *clarify*
apply(*rule someI-ex*)
subgoal for *L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' x y*
apply(*drule rel-G-neg-distr*[**where**
?L1.0 = $\lambda x\ (x', y).\ x' = x \wedge L1\ x\ y$ **and** *?L1'.0* = $\lambda(x, y)\ y'.\ y = y' \wedge$

L1 x y' **and**
?L2.0 = λ*x* (*x'*, *y*). *x'* = *x* ∧ *L2 x y* **and** *?L2'.0* = λ(*x*, *y*) *y'*. *y* = *y'* ∧ *L2 x y'*])
apply(*drule predicate2D*)
apply(*erule rel-G-mono*[*THEN predicate2D*, *rotated* −1]; *fastforce*)
apply(*erule relcomppE*)
apply(*rule exI conjI*)+
apply *assumption*+
done
done

end

3 Simple operations: demotion, merging, composition

theory *Composition* **imports**
Axiomatised-BNF-CC
begin

We illustrate the composition of $\mathrm{BNF}_{\mathrm{CC}}$s with one example for each kind of parameters (live/co-/contravariant/fixed). We do not show demotion and merging in isolation, as the examples for composition use these operations, too.

3.1 Composition in a live position

type-synonym
('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGl* =
(('*l1*, '*l2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*f1*) *G*,
'*l1*, '*l3*, '*co1*, '*co3*, '*co4*, '*contra1*, '*contra3*, '*contra4*, '*f2*) *F*

The type variables '*l1*, '*co1* and '*contra1* have each been merged.

definition *rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4* =
rel-F (*rel-G L1 L2 Co1 Co2 Contra1 Contra2*) *L1 L3 Co1 Co3 Co4 Contra1 Contra3 Contra4*

definition *map-FGl l1 l2 l3 co1 co2 co3 co4 contra1 contra2 contra3 contra4* =
map-F (*map-G l1 l2 co1 co2 contra1 contra2*) *l1 l3 co1 co3 co4 contra1 contra3 contra4*

lemma *rel-FGl-mono*:
⟦ *L1* ≤ *L1'*; *L2* ≤ *L2'*; *L3* ≤ *L3'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Co3* ≤ *Co3'*; *Co4* ≤ *Co4'*;
Contra1' ≤ *Contra1*; *Contra2'* ≤ *Contra2*; *Contra3'* ≤ *Contra3*; *Contra4'* ≤ *Contra4* ⟧ ⟹

rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4 $\leq$
rel-FGl L1' L2' L3' Co1' Co2' Co3' Co4' Contra1' Contra2' Contra3' Contra4'
unfolding *rel-FGl-def*
apply (*rule rel-F-mono*)
apply (*rule rel-G-mono*)
apply (*assumption*)+
done

lemma *rel-FGl-eq*: *rel-FGl* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)
unfolding *rel-FGl-def* **by** (*simp add*: *rel-F-eq rel-G-eq*)

lemma *rel-FGl-conversep*:
rel-FGl $L1^{-1-1}$ $L2^{-1-1}$ $L3^{-1-1}$ $Co1^{-1-1}$ $Co2^{-1-1}$ $Co3^{-1-1}$ $Co4^{-1-1}$ $Contra1^{-1-1}$ $Contra2^{-1-1}$ $Contra3^{-1-1}$ $Contra4^{-1-1}$ =
(*rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4*)$^{-1-1}$
unfolding *rel-FGl-def* **by** (*simp add*: *rel-F-conversep rel-G-conversep*)

lemma *map-FGl-id0*: *map-FGl id id id id id id id id id id id* = *id*
unfolding *map-FGl-def* **by** (*simp add*: *map-F-id0 map-G-id0*)

lemma *map-FGl-comp*: *map-FGl l1 l2 l3 co1 co2 co3 co4 contra1 contra2 contra3 contra4* $\circ$
map-FGl l1' l2' l3' co1' co2' co3' co4' contra1' contra2' contra3' contra4' =
map-FGl (*l1* $\circ$ *l1'*) (*l2* $\circ$ *l2'*) (*l3* $\circ$ *l3'*) (*co1* $\circ$ *co1'*) (*co2* $\circ$ *co2'*) (*co3* $\circ$ *co3'*) (*co4* $\circ$ *co4'*)
(*contra1'* $\circ$ *contra1*) (*contra2'* $\circ$ *contra2*) (*contra3'* $\circ$ *contra3*) (*contra4'* $\circ$ *contra4*)
unfolding *map-FGl-def* **by** (*simp add*: *map-F-comp map-G-comp*)

lemma *map-FGl-parametric*:
rel-fun (*rel-fun L1 L1'*) (*rel-fun* (*rel-fun L2 L2'*) (*rel-fun* (*rel-fun L3 L3'*)
(*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*)
(*rel-fun* (*rel-fun Co3 Co3'*) (*rel-fun* (*rel-fun Co4 Co4'*)
(*rel-fun* (*rel-fun Contra1' Contra1*) (*rel-fun* (*rel-fun Contra2' Contra2*)
(*rel-fun* (*rel-fun Contra3' Contra3*) (*rel-fun* (*rel-fun Contra4' Contra4*)
(*rel-fun* (*rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4*)
(*rel-FGl L1' L2' L3' Co1' Co2' Co3' Co4' Contra1' Contra2' Contra3' Contra4'*))))))))))
map-FGl map-FGl
unfolding *rel-FGl-def map-FGl-def*
apply (*intro rel-funI*)
apply (*elim map-F-rel-cong map-G-rel-cong*)
apply (*erule* (*2*) *rel-funE*)+
done

definition *rel-FGl-pos-distr-cond* :: ('*co1* $\Rightarrow$ '*co1'* $\Rightarrow$ *bool*) $\Rightarrow$ ('*co1'* $\Rightarrow$ '*co1''* $\Rightarrow$ *bool*) $\Rightarrow$
('*co2* $\Rightarrow$ '*co2'* $\Rightarrow$ *bool*) $\Rightarrow$ ('*co2'* $\Rightarrow$ '*co2''* $\Rightarrow$ *bool*) $\Rightarrow$
('*co3* $\Rightarrow$ '*co3'* $\Rightarrow$ *bool*) $\Rightarrow$ ('*co3'* $\Rightarrow$ '*co3''* $\Rightarrow$ *bool*) $\Rightarrow$

('co4 ⇒ 'co4' ⇒ bool) ⇒ ('co4' ⇒ 'co4'' ⇒ bool) ⇒
('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
('contra3 ⇒ 'contra3' ⇒ bool) ⇒ ('contra3' ⇒ 'contra3'' ⇒ bool) ⇒
('contra4 ⇒ 'contra4' ⇒ bool) ⇒ ('contra4' ⇒ 'contra4'' ⇒ bool) ⇒
('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' × 'l3 × 'l3' × 'l3'' × 'f1 × 'f2) itself
⇒ bool
where
rel-FGl-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' - ⟷
(∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool)
(L2 :: 'l2 ⇒ 'l2' ⇒ bool) (L2' :: 'l2' ⇒ 'l2'' ⇒ bool)
(L3 :: 'l3 ⇒ 'l3' ⇒ bool) (L3' :: 'l3' ⇒ 'l3'' ⇒ bool).
(rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4 ::
(-, -, -, -, -, -, -, -, -, -, -, 'f1, 'f2) FGl ⇒ -) OO
rel-FGl L1' L2' L3' Co1' Co2' Co3' Co4' Contra1' Contra2' Contra3'
Contra4' ≤
rel-FGl (L1 OO L1') (L2 OO L2') (L3 OO L3') (Co1 OO Co1') (Co2 OO
Co2') (Co3 OO Co3') (Co4 OO Co4')
(Contra1 OO Contra1') (Contra2 OO Contra2') (Contra3 OO Contra3')
(Contra4 OO Contra4'))

definition rel-FGl-neg-distr-cond :: ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒
bool) ⇒
('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
('co3 ⇒ 'co3' ⇒ bool) ⇒ ('co3' ⇒ 'co3'' ⇒ bool) ⇒
('co4 ⇒ 'co4' ⇒ bool) ⇒ ('co4' ⇒ 'co4'' ⇒ bool) ⇒
('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
('contra3 ⇒ 'contra3' ⇒ bool) ⇒ ('contra3' ⇒ 'contra3'' ⇒ bool) ⇒
('contra4 ⇒ 'contra4' ⇒ bool) ⇒ ('contra4' ⇒ 'contra4'' ⇒ bool) ⇒
('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' × 'l3 × 'l3' × 'l3'' × 'f1 × 'f2) itself
⇒ bool
where
rel-FGl-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' - ⟷
(∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool)
(L2 :: 'l2 ⇒ 'l2' ⇒ bool) (L2' :: 'l2' ⇒ 'l2'' ⇒ bool)
(L3 :: 'l3 ⇒ 'l3' ⇒ bool) (L3' :: 'l3' ⇒ 'l3'' ⇒ bool).
rel-FGl (L1 OO L1') (L2 OO L2') (L3 OO L3')
(Co1 OO Co1') (Co2 OO Co2') (Co3 OO Co3') (Co4 OO Co4')
(Contra1 OO Contra1') (Contra2 OO Contra2') (Contra3 OO Contra3')
(Contra4 OO Contra4') ≤
(rel-FGl L1 L2 L3 Co1 Co2 Co3 Co4 Contra1 Contra2 Contra3 Contra4 ::
(-, -, -, -, -, -, -, -, -, -, -, 'f1, 'f2) FGl ⇒ -) OO
rel-FGl L1' L2' L3' Co1' Co2' Co3' Co4' Contra1' Contra2' Contra3'
Contra4')

Sufficient conditions for subdistributivity over relation composition.

lemma *rel-FGl-pos-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-F* :: ((*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f1*) *G* ×
(*'l1'*, *'l2'*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'f1*) *G* ×
(*'l1''*, *'l2''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f1*) *G* ×
'l1 × *'l1'* × *'l1''* × *'l3* × *'l3'* × *'l3''* × *'f2*) *itself*
and *tytok-G* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f1*) *itself*
and *tytok-FGl* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'l3* × *'l3'* × *'l3''* × *'f1* × *'f2*) *itself*
assumes *rel-F-pos-distr-cond Co1 Co1' Co3 Co3' Co4 Co4' Contra1 Contra1' Contra3 Contra3' Contra4 Contra4' tytok-F*
and *rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-G*
shows *rel-FGl-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' tytok-FGl*
unfolding *rel-FGl-pos-distr-cond-def rel-FGl-def*
apply (*intro allI*)
apply (*rule order-trans*)
apply (*rule rel-F-pos-distr*)
apply (*rule assms(1)*)
apply (*rule rel-F-mono*)
apply (*rule rel-G-pos-distr*)
apply (*rule assms(2)*)
apply (*rule order-refl*)+
done

lemma *rel-FGl-neg-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-F* :: ((*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f1*) *G* ×
(*'l1'*, *'l2'*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'f1*) *G* ×
(*'l1''*, *'l2''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f1*) *G* ×
'l1 × *'l1'* × *'l1''* × *'l3* × *'l3'* × *'l3''* × *'f2*) *itself*
and *tytok-G* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f1*) *itself*
and *tytok-FGl* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'l3* × *'l3'* × *'l3''* × *'f1* × *'f2*) *itself*
assumes *rel-F-neg-distr-cond Co1 Co1' Co3 Co3' Co4 Co4' Contra1 Contra1' Contra3 Contra3' Contra4 Contra4' tytok-F*
and *rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-G*

shows *rel-FGl-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' tytok-FGl*
unfolding *rel-FGl-neg-distr-cond-def rel-FGl-def*
apply (*intro allI*)
apply (*rule order-trans*[*rotated*])
apply (*rule rel-F-neg-distr*)
apply (*rule assms*(*1*))
apply (*rule rel-F-mono*)
apply (*rule rel-G-neg-distr*)
apply (*rule assms*(*2*))
apply (*rule order-refl*)+
done

lemma *rel-FGl-pos-distr-cond-eq*:
fixes *tytok* :: ('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* × '*l3* × '*l3'* × '*l3''* × '*f1* × '*f2*) *itself*
shows *rel-FGl-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
by (*rule rel-FGl-pos-distr-imp rel-F-pos-distr-cond-eq rel-G-pos-distr-cond-eq*)+

lemma *rel-FGl-neg-distr-cond-eq*:
fixes *tytok* :: ('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* × '*l3* × '*l3'* × '*l3''* × '*f1* × '*f2*) *itself*
shows *rel-FGl-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
by (*rule rel-FGl-neg-distr-imp rel-F-neg-distr-cond-eq rel-G-neg-distr-cond-eq*)+

definition *rell-FGl L1 L2 L3* = *rel-FGl L1 L2 L3* (=) (=) (=) (=) (=) (=) (=) (=)
definition *mapl-FGl l1 l2 l3* = *map-FGl l1 l2 l3 id id id id id id id id*

type-synonym ('*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGlbd* =
('*co1*, '*co3*, '*co4*, '*contra1*, '*contra3*, '*contra4*, '*f2*) *Fbd* ×
('*co1*, '*co2*, '*contra1*, '*contra2*, '*f1*) *Gbd* +
('*co1*, '*co3*, '*co4*, '*contra1*, '*contra3*, '*contra4*, '*f2*) *Fbd*

definition *set1-FGl* :: ('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGl* ⇒ '*l1 set* **where**
set1-FGl x = (⋃ *y*∈*set1-F x*. *set1-G y*) ∪ *set2-F x*

definition *set2-FGl* :: ('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGl* ⇒ '*l2 set* **where**
set2-FGl x = (⋃ *y*∈*set1-F x*. *set2-G y*)

definition *set3-FGl* :: ('*l1*, '*l2*, '*l3*, '*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGl* ⇒ '*l3 set* **where**
set3-FGl x = *set3-F x*

definition
bd-FGl :: (*'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGlbd rel*
where *bd-FGl* = *bd-F* ∗*c* *bd-G* +*c* *bd-F*

lemma *set1-FGl-map*: *set1-FGl* ∘ *mapl-FGl l1 l2 l3* = *image l1* ∘ *set1-FGl*
by (*simp add*: *fun-eq-iff set1-FGl-def mapl-FGl-def map-FGl-def*
mapl-F-def[*symmetric*] *mapl-G-def*[*symmetric*]
set1-F-map[*THEN fun-cong*, *simplified*] *set2-F-map*[*THEN fun-cong*, *simplified*]
set1-G-map[*THEN fun-cong*, *simplified*]
image-Un image-UN)

lemma *set2-FGl-map*: *set2-FGl* ∘ *mapl-FGl l1 l2 l3* = *image l2* ∘ *set2-FGl*
by (*simp add*: *fun-eq-iff set2-FGl-def mapl-FGl-def map-FGl-def*
mapl-F-def[*symmetric*] *mapl-G-def*[*symmetric*]
set1-F-map[*THEN fun-cong*, *simplified*] *set2-G-map*[*THEN fun-cong*, *simplified*] *image-UN*)

lemma *set3-FGl-map*: *set3-FGl* ∘ *mapl-FGl l1 l2 l3* = *image l3* ∘ *set3-FGl*
by (*simp add*: *fun-eq-iff set3-FGl-def mapl-FGl-def map-FGl-def*
mapl-F-def[*symmetric*] *mapl-G-def*[*symmetric*] *set3-F-map*[*THEN fun-cong*, *simplified*])

lemma *bd-FGl-card-order*: *card-order bd-FGl*
unfolding *bd-FGl-def* **using** *bd-F-card-order bd-G-card-order*
by (*intro card-order-csum card-order-cprod*)

lemma *bd-FGl-cinfinite*: *cinfinite bd-FGl*
unfolding *bd-FGl-def* **using** *bd-F-cinfinite bd-G-cinfinite*
by (*intro cinfinite-csum disjI2*)

lemma
fixes *x* :: (-, -, -, *'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGl*
shows *set1-FGl-bound*: *card-of* (*set1-FGl x*) ≤*o*
(*bd-FGl* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGlbd rel*)
and *set2-FGl-bound*: *card-of* (*set2-FGl x*) ≤*o*
(*bd-FGl* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGlbd rel*)
and *set3-FGl-bound*: *card-of* (*set3-FGl x*) ≤*o*
(*bd-FGl* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGlbd rel*)
unfolding *set1-FGl-def set2-FGl-def set3-FGl-def bd-FGl-def*
apply (*simp*)
apply (*rule ordLeq-transitive*)
apply (*rule Un-csum*)
apply (*rule csum-mono*)

apply (*rule comp-single-set-bd*[**where** *fset*=*set1-G* **and** *gset*=*set1-F*, *rotated*])
apply (*rule set1-G-bound*)
apply (*rule set1-F-bound*)
apply (*rule card-order-on-Card-order*[*THEN conjunct2*, *OF bd-G-card-order*])
apply (*rule set2-F-bound*)
apply (*rule ordLeq-transitive*)
apply (*rule comp-single-set-bd*[**where** *fset*=*set2-G* **and** *gset*=*set1-F*, *rotated*])
apply (*rule set2-G-bound*)
apply (*rule set1-F-bound*)
apply (*rule card-order-on-Card-order*[*THEN conjunct2*, *OF bd-G-card-order*])
apply (*rule ordLeq-csum1*)
apply (*rule Card-order-cprod*)
apply (*rule ordLeq-transitive*)
apply (*rule set3-F-bound*)
apply (*rule ordLeq-csum2*)
apply (*rule card-order-on-Card-order*[*THEN conjunct2*, *OF bd-F-card-order*])
done

lemma *mapl-FGl-cong*:
assumes $\bigwedge z.\ z \in$ *set1-FGl x* $\Longrightarrow$ *l1 z* = *l1' z* **and** $\bigwedge z.\ z \in$ *set2-FGl x* $\Longrightarrow$ *l2 z* = *l2' z*
and $\bigwedge z.\ z \in$ *set3-FGl x* $\Longrightarrow$ *l3 z* = *l3' z*
shows *mapl-FGl l1 l2 l3 x* = *mapl-FGl l1' l2' l3' x*
unfolding *mapl-FGl-def map-FGl-def mapl-G-def*[*symmetric*] *mapl-F-def*[*symmetric*]
by (*auto 0 5 intro*: *mapl-F-cong mapl-G-cong assms simp add*: *set1-FGl-def set2-FGl-def set3-FGl-def*)

lemma *rell-FGl-mono-strong*:
assumes *rell-FGl L1 L2 L3 x y*
and $\bigwedge a\ b.\ a \in$ *set1-FGl x* $\Longrightarrow$ *b* $\in$ *set1-FGl y* $\Longrightarrow$ *L1 a b* $\Longrightarrow$ *L1' a b*
and $\bigwedge a\ b.\ a \in$ *set2-FGl x* $\Longrightarrow$ *b* $\in$ *set2-FGl y* $\Longrightarrow$ *L2 a b* $\Longrightarrow$ *L2' a b*
and $\bigwedge a\ b.\ a \in$ *set3-FGl x* $\Longrightarrow$ *b* $\in$ *set3-FGl y* $\Longrightarrow$ *L3 a b* $\Longrightarrow$ *L3' a b*
shows *rell-FGl L1' L2' L3' x y*
using *assms*(*1*) **unfolding** *rell-FGl-def rel-FGl-def rell-G-def*[*symmetric*] *rell-F-def*[*symmetric*]
by (*auto 0 5 intro*: *rell-F-mono-strong rell-G-mono-strong assms*(*2*−*4*)
simp add: *set1-FGl-def set2-FGl-def set3-FGl-def*)

3.2 Composition in a covariant position

type-synonym
('*l1*, '*co1*, '*co2*, '*co3*, '*co4*, '*co5*, '*co6*, '*contra1*, '*contra2*, '*contra3*, '*contra4*, '*f1*, '*f2*) *FGco* =
('*l1*, '*co1*, '*co5*, ('*co1*, '*co2*, '*co3*, '*co4*, '*contra1*, '*contra2*, '*f1*) *G*, '*co3*, '*co6*, '*contra1*, '*contra3*, '*contra4*, '*f2*) *F*

The type variables *'co1*, *'co3* and *'contra1* have each been merged.

definition *rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4* =
rel-F L1 Co1 Co5 (*rel-G Co1 Co2 Co3 Co4 Contra1 Contra2*) *Co3 Co6 Contra1*

Contra3 Contra4

definition *map-FGco l1 co1 co2 co3 co4 co5 co6 contra1 contra2 contra3 contra4* =
map-F l1 co1 co5 (*map-G co1 co2 co3 co4 contra1 contra2*) *co3 co6 contra1 contra3 contra4*

lemma *rel-FGco-mono*:
⟦ *L1* ≤ *L1'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Co3* ≤ *Co3'*; *Co4* ≤ *Co4'*; *Co5* ≤ *Co5'*; *Co6* ≤ *Co6'*;
Contra1' ≤ *Contra1*; *Contra2'* ≤ *Contra2*; *Contra3'* ≤ *Contra3*; *Contra4'* ≤ *Contra4* ⟧ ⟹
rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4 ≤
rel-FGco L1' Co1' Co2' Co3' Co4' Co5' Co6' Contra1' Contra2' Contra3' Contra4'
unfolding *rel-FGco-def*
apply (*rule rel-F-mono*)
apply (*assumption*)+
apply (*rule rel-G-mono*)
apply (*assumption*)+
done

lemma *rel-FGco-eq*: *rel-FGco* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)
unfolding *rel-FGco-def* **by** (*simp add*: *rel-F-eq rel-G-eq*)

lemma *rel-FGco-conversep*:
rel-FGco L1^{-1-1} *Co1*$^{-1-1}$ *Co2*$^{-1-1}$ *Co3*$^{-1-1}$ *Co4*$^{-1-1}$ *Co5*$^{-1-1}$ *Co6*$^{-1-1}$ *Contra1*$^{-1-1}$ *Contra2*$^{-1-1}$ *Contra3*$^{-1-1}$ *Contra4*$^{-1-1}$ =
(*rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4*)$^{-1-1}$
unfolding *rel-FGco-def* **by** (*simp add*: *rel-F-conversep rel-G-conversep*)

lemma *map-FGco-id0*: *map-FGco id id id id id id id id id id* = *id*
unfolding *map-FGco-def* **by** (*simp add*: *map-F-id0 map-G-id0*)

lemma *map-FGco-comp*: *map-FGco l1 co1 co2 co3 co4 co5 co6 contra1 contra2 contra3 contra4* ∘
map-FGco l1' co1' co2' co3' co4' co5' co6' contra1' contra2' contra3' contra4' =
map-FGco (*l1* ∘ *l1'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*co3* ∘ *co3'*) (*co4* ∘ *co4'*) (*co5* ∘ *co5'*) (*co6* ∘ *co6'*)
(*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*) (*contra3'* ∘ *contra3*) (*contra4'* ∘ *contra4*)
unfolding *map-FGco-def* **by** (*simp add*: *map-F-comp map-G-comp*)

lemma *map-FGco-parametric*:
rel-fun (*rel-fun L1 L1'*) (*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*)
(*rel-fun* (*rel-fun Co3 Co3'*) (*rel-fun* (*rel-fun Co4 Co4'*)
(*rel-fun* (*rel-fun Co5 Co5'*) (*rel-fun* (*rel-fun Co6 Co6'*)

(*rel-fun* (*rel-fun Contra1' Contra1*) (*rel-fun* (*rel-fun Contra2' Contra2*)
(*rel-fun* (*rel-fun Contra3' Contra3*) (*rel-fun* (*rel-fun Contra4' Contra4*)
(*rel-fun* (*rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4*)
(*rel-FGco L1' Co1' Co2' Co3' Co4' Co5' Co6' Contra1' Contra2' Contra3' Contra4'*)))))))))))
map-FGco map-FGco
unfolding *rel-FGco-def map-FGco-def*
apply (*intro rel-funI*)
apply (*elim map-F-rel-cong map-G-rel-cong*)
apply (*erule* (*2*) *rel-funE*)+
done

definition *rel-FGco-pos-distr-cond* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*co3* ⇒ '*co3'* ⇒ *bool*) ⇒ ('*co3'* ⇒ '*co3''* ⇒ *bool*) ⇒
('*co4* ⇒ '*co4'* ⇒ *bool*) ⇒ ('*co4'* ⇒ '*co4''* ⇒ *bool*) ⇒
('*co5* ⇒ '*co5'* ⇒ *bool*) ⇒ ('*co5'* ⇒ '*co5''* ⇒ *bool*) ⇒
('*co6* ⇒ '*co6'* ⇒ *bool*) ⇒ ('*co6'* ⇒ '*co6''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*contra3* ⇒ '*contra3'* ⇒ *bool*) ⇒ ('*contra3'* ⇒ '*contra3''* ⇒ *bool*) ⇒
('*contra4* ⇒ '*contra4'* ⇒ *bool*) ⇒ ('*contra4'* ⇒ '*contra4''* ⇒ *bool*) ⇒
('*l1* × '*l1'* × '*l1''* × '*f1* × '*f2*) *itself* ⇒ *bool* **where**
rel-FGco-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Co6 Co6'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' - ⟷
(∀(*L1* :: '*l1* ⇒ '*l1'* ⇒ *bool*) (*L1'* :: '*l1'* ⇒ '*l1''* ⇒ *bool*).
(*rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4* ::
(-, -, -, -, -, -, -, -, -, -, -, '*f1*, '*f2*) *FGco* ⇒ -) *OO*
rel-FGco L1' Co1' Co2' Co3' Co4' Co5' Co6' Contra1' Contra2' Contra3' Contra4' ≤
rel-FGco (*L1 OO L1'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*)
(*Co4 OO Co4'*) (*Co5 OO Co5'*) (*Co6 OO Co6'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*) (*Contra4 OO Contra4'*))

definition *rel-FGco-neg-distr-cond* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*co3* ⇒ '*co3'* ⇒ *bool*) ⇒ ('*co3'* ⇒ '*co3''* ⇒ *bool*) ⇒
('*co4* ⇒ '*co4'* ⇒ *bool*) ⇒ ('*co4'* ⇒ '*co4''* ⇒ *bool*) ⇒
('*co5* ⇒ '*co5'* ⇒ *bool*) ⇒ ('*co5'* ⇒ '*co5''* ⇒ *bool*) ⇒
('*co6* ⇒ '*co6'* ⇒ *bool*) ⇒ ('*co6'* ⇒ '*co6''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*contra3* ⇒ '*contra3'* ⇒ *bool*) ⇒ ('*contra3'* ⇒ '*contra3''* ⇒ *bool*) ⇒
('*contra4* ⇒ '*contra4'* ⇒ *bool*) ⇒ ('*contra4'* ⇒ '*contra4''* ⇒ *bool*) ⇒

(*'l1 × 'l1' × 'l1'' × 'f1 × 'f2*) *itself* ⇒ *bool* **where**
rel-FGco-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Co6 Co6'
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' - ⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*).
rel-FGco (*L1 OO L1'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*)
(*Co4 OO Co4'*) (*Co5 OO Co5'*) (*Co6 OO Co6'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*) (*Contra4 OO Contra4'*) ≤
(*rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4* ::
(-, -, -, -, -, -, -, -, -, -, -, *'f1*, *'f2*) *FGco* ⇒ -) *OO*
rel-FGco L1' Co1' Co2' Co3' Co4' Co5' Co6' Contra1' Contra2' Contra3' Contra4')

Sufficient conditions for subdistributivity over relation composition.

lemma *rel-FGco-pos-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Co5* :: *'co5* ⇒ *'co5'* ⇒ *bool* **and** *Co5'* :: *'co5'* ⇒ *'co5''* ⇒ *bool*
and *tytok-F* :: (*'l1 × 'l1' × 'l1'' × 'co1 × 'co1' × 'co1'' × 'co5 × 'co5' × 'co5'' ×*
'f2) *itself*
and *tytok-G* :: (*'co1 × 'co1' × 'co1'' × 'co2 × 'co2' × 'co2'' × 'f1*) *itself*
and *tytok-FGco* :: (*'l1 × 'l1' × 'l1'' × 'f1 × 'f2*) *itself*
assumes *rel-F-pos-distr-cond*
(*rel-G Co1 Co2 Co3 Co4 Contra1 Contra2* :: (-, -, -, -, -, -, *'f1*) *G* ⇒ -)
(*rel-G Co1' Co2' Co3' Co4' Contra1' Contra2'*) *Co3 Co3' Co6 Co6'*
Contra1 Contra1' Contra3 Contra3' Contra4 Contra4' tytok-F
and *rel-G-pos-distr-cond Co3 Co3' Co4 Co4' Contra1 Contra1' Contra2 Contra2' tytok-G*
shows *rel-FGco-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Co6 Co6'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' tytok-FGco
unfolding *rel-FGco-pos-distr-cond-def rel-FGco-def*
apply (*intro allI*)
apply (*rule order-trans*)
apply (*rule rel-F-pos-distr*)
apply (*rule assms*(*1*))
apply (*rule rel-F-mono*)
apply (*rule order-refl*)+
apply (*rule rel-G-pos-distr*)
apply (*rule assms*(*2*))
apply (*rule order-refl*)+
done

lemma *rel-FGco-neg-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Co5* :: *'co5* ⇒ *'co5'* ⇒ *bool* **and** *Co5'* :: *'co5'* ⇒ *'co5''* ⇒ *bool*

and *tytok-F* :: (*'l1* × *'l1'* × *'l1''* × *'co1* × *'co1'* × *'co1''* × *'co5* × *'co5'* × *'co5''* × *'f2*) *itself*
and *tytok-G* :: (*'co1* × *'co1'* × *'co1''* × *'co2* × *'co2'* × *'co2''* × *'f1*) *itself*
and *tytok-FGco* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
assumes *rel-F-neg-distr-cond*
(*rel-G Co1 Co2 Co3 Co4 Contra1 Contra2* :: (-, -, -, -, -, -, *'f1*) *G* ⇒ -)
(*rel-G Co1' Co2' Co3' Co4' Contra1' Contra2'*) *Co3 Co3' Co6 Co6' Contra1 Contra1' Contra3 Contra3' Contra4 Contra4' tytok-F*
and *rel-G-neg-distr-cond Co3 Co3' Co4 Co4' Contra1 Contra1' Contra2 Contra2' tytok-G*
shows *rel-FGco-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Co6 Co6' Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' tytok-FGco*
unfolding *rel-FGco-neg-distr-cond-def rel-FGco-def*
apply (*intro allI*)
apply (*rule order-trans*[*rotated*])
apply (*rule rel-F-neg-distr*)
apply (*rule assms*(*1*))
apply (*rule rel-F-mono*)
apply (*rule order-refl*)+
apply (*rule rel-G-neg-distr*)
apply (*rule assms*(*2*))
apply (*rule order-refl*)+
done

lemma *rel-FGco-pos-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGco-pos-distr-cond* (=) *tytok*
apply (*rule rel-FGco-pos-distr-imp*)
apply (*simp add*: *rel-G-eq*)
apply (*rule rel-F-pos-distr-cond-eq rel-G-pos-distr-cond-eq*)+
done

lemma *rel-FGco-neg-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGco-neg-distr-cond* (=) *tytok*
apply (*rule rel-FGco-neg-distr-imp*)
apply (*simp add*: *rel-G-eq*)
apply (*rule rel-F-neg-distr-cond-eq rel-G-neg-distr-cond-eq*)+
done

definition *rell-FGco L1* = *rel-FGco L1* (=)
definition *mapl-FGco l1* = *map-FGco l1 id id id id id id id id id id*

type-synonym (*'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'co6*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGcobd* =
((*'co1*, *'co2*, *'co3*, *'co4*, *'contra1*, *'contra2*, *'f1*) *G*,
'co3, *'co6*, *'contra1*, *'contra3*, *'contra4*, *'f2*) *Fbd*

definition *set1-FGco* :: (*'l1*, *'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'co6*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGco* ⇒ *'l1 set* **where**
set1-FGco x = *set1-F x*

definition *bd-FGco* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'co6*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGcobd rel* **where**
bd-FGco = *bd-F*

lemma *set1-FGco-map*: *set1-FGco* ∘ *mapl-FGco l1* = *image l1* ∘ *set1-FGco*
by (*simp add*: *fun-eq-iff set1-FGco-def mapl-FGco-def map-FGco-def*
mapl-F-def[*symmetric*] *mapl-G-def*[*symmetric*] *mapl-G-id0*
set1-F-map[*THEN fun-cong*, *simplified*])

lemma *bd-FGco-card-order*: *card-order bd-FGco*
unfolding *bd-FGco-def* **using** *bd-F-card-order* **.**

lemma *bd-FGco-cinfinite*: *cinfinite bd-FGco*
unfolding *bd-FGco-def* **using** *bd-F-cinfinite* **.**

lemma *set1-FGco-bound*:
fixes *x* :: (-, *'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'co6*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGco*
shows *card-of* (*set1-FGco x*) ≤*o* (*bd-FGco* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'co6*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'f1*, *'f2*) *FGcobd rel*)
unfolding *set1-FGco-def bd-FGco-def* **using** *set1-F-bound* **.**

lemma *mapl-FGco-cong*:
assumes ⋀*z*. *z* ∈ *set1-FGco x* ⟹ *l1 z* = *l1' z*
shows *mapl-FGco l1 x* = *mapl-FGco l1' x*
unfolding *mapl-FGco-def map-FGco-def mapl-G-def*[*symmetric*] *mapl-F-def*[*symmetric*]
mapl-G-id0
by (*auto 0 3 intro*: *mapl-F-cong assms simp add*: *set1-FGco-def*)

lemma *rell-FGco-mono-strong*:
assumes *rell-FGco L1 x y*
and ⋀*a b*. *a* ∈ *set1-FGco x* ⟹ *b* ∈ *set1-FGco y* ⟹ *L1 a b* ⟹ *L1' a b*
shows *rell-FGco L1' x y*
using *assms*(*1*) **unfolding** *rell-FGco-def rel-FGco-def rel-G-eq rell-F-def*[*symmetric*]
by (*auto 0 3 intro*: *rell-F-mono-strong assms*(*2*) *simp add*: *set1-FGco-def*)

3.3 Composition in a contravariant position

type-synonym
(*'l1*, *'co1*, *'co2*, *'co3*, *'co4*, *'co5*, *'contra1*,

'contra2, *'contra3*, *'contra4*, *'contra5*, *'f1*, *'f2*) *FGcontra* =
(*'l1*, *'co1*, *'co3*, *'co1*, *'co4*, *'co5*, (*'contra1*, *'contra2*, *'contra3*, *'contra4*, *'co1*, *'co2*, *'f1*) *G*,
'contra1, *'contra5*, *'f2*) *F*

The type variables *'co1* and *'contra1* have each been merged.

definition *rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5* =
rel-F L1 Co1 Co3 Co1 Co4 Co5 (*rel-G Contra1 Contra2 Contra3 Contra4 Co1 Co2*) *Contra1 Contra5*

definition *map-FGcontra l1 co1 co2 co3 co4 co5 contra1 contra2 contra3 contra4 contra5* =
map-F l1 co1 co3 co1 co4 co5 (*map-G contra1 contra2 contra3 contra4 co1 co2*) *contra1 contra5*

lemma *rel-FGcontra-mono*:
⟦ *L1* ≤ *L1'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Co3* ≤ *Co3'*; *Co4* ≤ *Co4'*; *Co5* ≤ *Co5'*;
Contra1' ≤ *Contra1*; *Contra2'* ≤ *Contra2*; *Contra3'* ≤ *Contra3*;
Contra4' ≤ *Contra4*; *Contra5'* ≤ *Contra5* ⟧ ⟹
rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5 ≤
rel-FGcontra L1' Co1' Co2' Co3' Co4' Co5' Contra1' Contra2' Contra3' Contra4' Contra5'
unfolding *rel-FGcontra-def*
apply (*rule rel-F-mono*)
apply (*assumption*)+
apply (*rule rel-G-mono*)
apply (*assumption*)+
done

lemma *rel-FGcontra-eq*: *rel-FGcontra* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)
unfolding *rel-FGcontra-def* **by** (*simp add*: *rel-F-eq rel-G-eq*)

lemma *rel-FGcontra-conversep*:
rel-FGcontra L1^{-1-1} *Co1*$^{-1-1}$ *Co2*$^{-1-1}$ *Co3*$^{-1-1}$ *Co4*$^{-1-1}$ *Co5*$^{-1-1}$ *Contra1*$^{-1-1}$ *Contra2*$^{-1-1}$ *Contra3*$^{-1-1}$ *Contra4*$^{-1-1}$ *Contra5*$^{-1-1}$ =
(*rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5*)$^{-1-1}$
unfolding *rel-FGcontra-def* **by** (*simp add*: *rel-F-conversep rel-G-conversep*)

lemma *map-FGcontra-id0*: *map-FGcontra id id id id id id id id id id id* = *id*
unfolding *map-FGcontra-def* **by** (*simp add*: *map-F-id0 map-G-id0*)

lemma *map-FGcontra-comp*:
map-FGcontra l1 co1 co2 co3 co4 co5 contra1 contra2 contra3 contra4 contra5 ∘
map-FGcontra l1' co1' co2' co3' co4' co5' contra1' contra2' contra3' contra4' contra5' =

map-FGcontra (*l1* ∘ *l1'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*co3* ∘ *co3'*) (*co4* ∘ *co4'*) (*co5* ∘ *co5'*)
(*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*) (*contra3'* ∘ *contra3*)
(*contra4'* ∘ *contra4*) (*contra5'* ∘ *contra5*)
unfolding *map-FGcontra-def* **by** (*simp add*: *map-F-comp map-G-comp*)

lemma *map-FGcontra-parametric*:
rel-fun (*rel-fun L1 L1'*) (*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*)
(*rel-fun* (*rel-fun Co3 Co3'*) (*rel-fun* (*rel-fun Co4 Co4'*) (*rel-fun* (*rel-fun Co5 Co5'*)
(*rel-fun* (*rel-fun Contra1' Contra1*) (*rel-fun* (*rel-fun Contra2' Contra2*)
(*rel-fun* (*rel-fun Contra3' Contra3*) (*rel-fun* (*rel-fun Contra4' Contra4*)
(*rel-fun* (*rel-fun Contra5' Contra5*)
(*rel-fun* (*rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5*)
(*rel-FGcontra L1' Co1' Co2' Co3' Co4' Co5' Contra1' Contra2' Contra3' Contra4' Contra5'*))))))))))
map-FGcontra map-FGcontra
unfolding *rel-FGcontra-def map-FGcontra-def*
apply (*intro rel-funI*)
apply (*elim map-F-rel-cong map-G-rel-cong*)
apply (*erule* (2) *rel-funE*)+
done

definition *rel-FGcontra-pos-distr-cond* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*co3* ⇒ '*co3'* ⇒ *bool*) ⇒ ('*co3'* ⇒ '*co3''* ⇒ *bool*) ⇒
('*co4* ⇒ '*co4'* ⇒ *bool*) ⇒ ('*co4'* ⇒ '*co4''* ⇒ *bool*) ⇒
('*co5* ⇒ '*co5'* ⇒ *bool*) ⇒ ('*co5'* ⇒ '*co5''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*contra3* ⇒ '*contra3'* ⇒ *bool*) ⇒ ('*contra3'* ⇒ '*contra3''* ⇒ *bool*) ⇒
('*contra4* ⇒ '*contra4'* ⇒ *bool*) ⇒ ('*contra4'* ⇒ '*contra4''* ⇒ *bool*) ⇒
('*contra5* ⇒ '*contra5'* ⇒ *bool*) ⇒ ('*contra5'* ⇒ '*contra5''* ⇒ *bool*) ⇒
('*l1* × '*l1'* × '*l1''* × '*f1* × '*f2*) *itself* ⇒ *bool* **where**
rel-FGcontra-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' Contra5 Contra5' - ⟷
(∀(*L1* :: '*l1* ⇒ '*l1'* ⇒ *bool*) (*L1'* :: '*l1'* ⇒ '*l1''* ⇒ *bool*).
(*rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5* ::
(-, -, -, -, -, -, -, -, -, -, '*f1*, '*f2*) *FGcontra* ⇒ -) *OO*
rel-FGcontra L1' Co1' Co2' Co3' Co4' Co5' Contra1' Contra2' Contra3' Contra4' Contra5' ≤
rel-FGcontra (*L1 OO L1'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*) (*Co4 OO Co4'*) (*Co5 OO Co5'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*)
(*Contra4 OO Contra4'*) (*Contra5 OO Contra5'*))

definition *rel-FGcontra-neg-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'co3* ⇒ *'co3'* ⇒ *bool*) ⇒ (*'co3'* ⇒ *'co3''* ⇒ *bool*) ⇒
(*'co4* ⇒ *'co4'* ⇒ *bool*) ⇒ (*'co4'* ⇒ *'co4''* ⇒ *bool*) ⇒
(*'co5* ⇒ *'co5'* ⇒ *bool*) ⇒ (*'co5'* ⇒ *'co5''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'contra3* ⇒ *'contra3'* ⇒ *bool*) ⇒ (*'contra3'* ⇒ *'contra3''* ⇒ *bool*) ⇒
(*'contra4* ⇒ *'contra4'* ⇒ *bool*) ⇒ (*'contra4'* ⇒ *'contra4''* ⇒ *bool*) ⇒
(*'contra5* ⇒ *'contra5'* ⇒ *bool*) ⇒ (*'contra5'* ⇒ *'contra5''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself* ⇒ *bool* **where**
rel-FGcontra-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5' Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' Contra5 Contra5' - ⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*).
rel-FGcontra (*L1 OO L1'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Co3 OO Co3'*) (*Co4 OO Co4'*) (*Co5 OO Co5'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) (*Contra3 OO Contra3'*)
(*Contra4 OO Contra4'*) (*Contra5 OO Contra5'*) ≤
(*rel-FGcontra L1 Co1 Co2 Co3 Co4 Co5 Contra1 Contra2 Contra3 Contra4 Contra5* ::
(-, -, -, -, -, -, -, -, -, -, -, *'f1*, *'f2*) *FGcontra* ⇒ -) *OO*
rel-FGcontra L1' Co1' Co2' Co3' Co4' Co5' Contra1' Contra2' Contra3' Contra4' Contra5')

Sufficient conditions for subdistributivity over relation composition.

lemma *rel-FGcontra-pos-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co3* :: *'co3* ⇒ *'co3'* ⇒ *bool* **and** *Co3'* :: *'co3'* ⇒ *'co3''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-F* :: (*'l1* × *'l1'* × *'l1''* × *'co1* × *'co1'* × *'co1''* × *'co3* × *'co3'* × *'co3''* ×
'f2) *itself*
and *tytok-G* :: (*'contra1* × *'contra1'* × *'contra1''* × *'contra2* × *'contra2'* × *'contra2''* ×
'f1) *itself*
and *tytok-FGcontra* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
assumes *rel-F-pos-distr-cond Co1 Co1' Co4 Co4' Co5 Co5'*
(*rel-G Contra1 Contra2 Contra3 Contra4 Co1 Co2* :: (-, -, -, -, -, -, *'f1*) *G* ⇒ -)
(*rel-G Contra1' Contra2' Contra3' Contra4' Co1' Co2'*)
Contra1 Contra1' Contra5 Contra5' tytok-F
and *rel-G-neg-distr-cond Contra3 Contra3' Contra4 Contra4' Co1 Co1' Co2 Co2' tytok-G*

shows *rel-FGcontra-pos-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' Contra5 Contra5'
tytok-FGcontra
unfolding *rel-FGcontra-pos-distr-cond-def rel-FGcontra-def*
apply (*intro allI*)
apply (*rule order-trans*)
apply (*rule rel-F-pos-distr*)
apply (*rule assms(1)*)
apply (*rule rel-F-mono*)
apply (*rule order-refl*)+
apply (*rule rel-G-neg-distr*)
apply (*rule assms(2)*)
apply (*rule order-refl*)+
done

lemma *rel-FGcontra-neg-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co3* :: *'co3* ⇒ *'co3'* ⇒ *bool* **and** *Co3'* :: *'co3'* ⇒ *'co3''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-F* :: (*'l1* × *'l1'* × *'l1''* × *'co1* × *'co1'* × *'co1''* × *'co3* × *'co3'* × *'co3''* ×
'f2) *itself*
and *tytok-G* :: (*'contra1* × *'contra1'* × *'contra1''* × *'contra2* × *'contra2'* × *'contra2''* ×
'f1) *itself*
and *tytok-FGcontra* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
assumes *rel-F-neg-distr-cond Co1 Co1' Co4 Co4' Co5 Co5'*
(*rel-G Contra1 Contra2 Contra3 Contra4 Co1 Co2* :: (-, -, -, -, -, -, *'f1*) *G* ⇒ -)
(*rel-G Contra1' Contra2' Contra3' Contra4' Co1' Co2'*)
Contra1 Contra1' Contra5 Contra5' tytok-F
and *rel-G-pos-distr-cond Contra3 Contra3' Contra4 Contra4' Co1 Co1' Co2 Co2' tytok-G*
shows *rel-FGcontra-neg-distr-cond Co1 Co1' Co2 Co2' Co3 Co3' Co4 Co4' Co5 Co5'*
Contra1 Contra1' Contra2 Contra2' Contra3 Contra3' Contra4 Contra4' Contra5 Contra5' tytok-FGcontra
unfolding *rel-FGcontra-neg-distr-cond-def rel-FGcontra-def*
apply (*intro allI*)
apply (*rule order-trans*[*rotated*])
apply (*rule rel-F-neg-distr*)
apply (*rule assms(1)*)
apply (*rule rel-F-mono*)
apply (*rule order-refl*)+

```
   apply (rule rel-G-pos-distr)
   apply (rule assms(2))
  apply (rule order-refl)+
 done

lemma rel-FGcontra-pos-distr-cond-eq:
 fixes tytok :: ('l1 × 'l1' × 'l1'' × 'f1 × 'f2) itself
 shows rel-FGcontra-pos-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) (=) (=)
   (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) tytok
 apply (rule rel-FGcontra-pos-distr-imp)
  apply (simp add: rel-G-eq)
  apply (rule rel-F-pos-distr-cond-eq rel-G-neg-distr-cond-eq)+
 done

lemma rel-FGcontra-neg-distr-cond-eq:
 fixes tytok :: ('l1 × 'l1' × 'l1'' × 'f1 × 'f2) itself
 shows rel-FGcontra-neg-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) (=) (=)
   (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) tytok
 apply (rule rel-FGcontra-neg-distr-imp)
  apply (simp add: rel-G-eq)
  apply (rule rel-F-neg-distr-cond-eq rel-G-pos-distr-cond-eq)+
 done

definition rell-FGcontra L1 = rel-FGcontra L1 (=) (=) (=) (=) (=) (=) (=) (=)
(=) (=)
definition mapl-FGcontra l1 = map-FGcontra l1 id id id id id id id id id

type-synonym ('co1, 'co2, 'co3, 'co4, 'co5, 'contra1, 'contra2, 'contra3, 'contra4,
'contra5,
   'f1, 'f2) FGcontrabd =
 ('co1, 'co4, 'co5, ('contra1, 'contra2, 'contra3, 'contra4, 'co1, 'co2, 'f1) G,
   'contra1, 'contra5, 'f2) Fbd

definition set1-FGcontra :: ('l1, 'co1, 'co2, 'co3, 'co4, 'co5,
    'contra1, 'contra2, 'contra3, 'contra4, 'contra5, 'f1, 'f2) FGcontra ⇒ 'l1 set
where
 set1-FGcontra x = set1-F x

definition bd-FGcontra :: ('co1, 'co2, 'co3, 'co4, 'co5,
  'contra1, 'contra2, 'contra3, 'contra4, 'contra5, 'f1, 'f2) FGcontrabd rel where
 bd-FGcontra = bd-F

lemma set1-FGcontra-map: set1-FGcontra ∘ mapl-FGcontra l1 = image l1 ∘ set1-FGcontra
 by (simp add: fun-eq-iff set1-FGcontra-def mapl-FGcontra-def map-FGcontra-def
     mapl-F-def[symmetric] mapl-G-def[symmetric] mapl-G-id0
     set1-F-map[THEN fun-cong, simplified])

lemma bd-FGcontra-card-order: card-order bd-FGcontra
 unfolding bd-FGcontra-def using bd-F-card-order .
```

lemma *bd-FGcontra-cinfinite*: *cinfinite bd-FGcontra*
unfolding *bd-FGcontra-def* **using** *bd-F-cinfinite* **.**

lemma *set1-FGcontra-bound*:
fixes *x* :: (-, *'co1*, *'co2*, *'co3*, *'co4*, *'co5*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'contra5*, *'f1*, *'f2*) *FGcontra*
shows *card-of* (*set1-FGcontra x*) ≤*o* (*bd-FGcontra* :: (*'co1*, *'co2*, *'co3*, *'co4*, *'co5*,
'contra1, *'contra2*, *'contra3*, *'contra4*, *'contra5*, *'f1*, *'f2*) *FGcontrabd rel*)
unfolding *set1-FGcontra-def bd-FGcontra-def* **using** *set1-F-bound* **.**

lemma *mapl-FGcontra-contrang*:
assumes ⋀*z*. *z* ∈ *set1-FGcontra x* ⟹ *l1 z* = *l1' z*
shows *mapl-FGcontra l1 x* = *mapl-FGcontra l1' x*
unfolding *mapl-FGcontra-def map-FGcontra-def mapl-G-def*[*symmetric*] *mapl-F-def*[*symmetric*] *mapl-G-id0*
by (*auto 0 3 intro*: *mapl-F-cong assms simp add*: *set1-FGcontra-def*)

lemma *rell-FGcontra-mono-strong*:
assumes *rell-FGcontra L1 x y*
and ⋀*a b*. *a* ∈ *set1-FGcontra x* ⟹ *b* ∈ *set1-FGcontra y* ⟹ *L1 a b* ⟹ *L1' a b*
shows *rell-FGcontra L1' x y*
using *assms*(*1*) **unfolding** *rell-FGcontra-def rel-FGcontra-def rel-G-eq rell-F-def*[*symmetric*]
by (*auto 0 3 intro*: *rell-F-mono-strong assms*(*2*) *simp add*: *set1-FGcontra-def*)

3.4 Composition in a fixed position

type-synonym (*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f1*, *'f2*, *'f3*, *'f4*, *'f5*, *'f6*, *'f7*) *FGf* =
(*'l1*, *'l2*, *'f2*, *'co1*, *'co2*, *'f4*, *'contra1*, *'contra2*, *'f6*, (*'f1*, *'f2*, *'f3*, *'f4*, *'f5*, *'f6*, *'f7*) *G*) *F*

The type variables *'f2*, *'f4* and *'f6* have each been merged.

definition *rel-FGf L1 L2 Co1 Co2 Contra1 Contra2* =
rel-F L1 L2 (=) *Co1 Co2* (=) *Contra1 Contra2* (=)

definition *map-FGf l1 l2 co1 co2 contra1 contra2* = *map-F l1 l2 id co1 co2 id contra1 contra2 id*

lemma *rel-FGf-mono*:
⟦ *L1* ≤ *L1'*; *L2* ≤ *L2'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Contra1'* ≤ *Contra1*; *Contra2'* ≤ *Contra2* ⟧ ⟹
rel-FGf L1 L2 Co1 Co2 Contra1 Contra2 ≤ *rel-FGf L1' L2' Co1' Co2' Contra1' Contra2'*
unfolding *rel-FGf-def* **by** (*rule rel-F-mono*) (*auto*)

lemma *rel-FGf-eq*: *rel-FGf* (=) (=) (=) (=) (=) (=) = (=)

unfolding *rel-FGf-def* **by** (*simp add*: *rel-F-eq*)

lemma *rel-FGf-conversep*:
rel-FGf L1^{-1-1} *L2*$^{-1-1}$ *Co1*$^{-1-1}$ *Co2*$^{-1-1}$ *Contra1*$^{-1-1}$ *Contra2*$^{-1-1}$ = (*rel-FGf L1 L2 Co1 Co2 Contra1 Contra2*)$^{-1-1}$
unfolding *rel-FGf-def* **by** (*simp add*: *rel-F-conversep*[*symmetric*])

lemma *map-FGf-id0*: *map-FGf id id id id id id* = *id*
unfolding *map-FGf-def* **by** (*simp add*: *map-F-id0*)

lemma *map-FGf-comp*: *map-FGf l1 l2 co1 co2 contra1 contra2* ∘
map-FGf l1' l2' co1' co2' contra1' contra2' =
map-FGf (*l1* ∘ *l1'*) (*l2* ∘ *l2'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*)
unfolding *map-FGf-def* **by** (*simp add*: *map-F-comp*)

lemma *map-FGf-parametric*:
rel-fun (*rel-fun L1 L1'*) (*rel-fun* (*rel-fun L2 L2'*)
(*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*)
(*rel-fun* (*rel-fun Contra1' Contra1*) (*rel-fun* (*rel-fun Contra2' Contra2*)
(*rel-fun* (*rel-FGf L1 L2 Co1 Co2 Contra1 Contra2*)
(*rel-FGf L1' L2' Co1' Co2' Contra1' Contra2'*)))))))
map-FGf map-FGf
unfolding *rel-FGf-def map-FGf-def*
apply (*intro rel-funI*)
apply (*elim map-F-rel-cong*)
apply (*simp-all*)
apply (*erule* (2) *rel-funE*)+
done

definition *rel-FGf-pos-distr-cond* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* ×
'*f1* × '*f2* × '*f3* × '*f4* × '*f5* × '*f6* × '*f7*) *itself* ⇒ *bool* **where**
rel-FGf-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' - ⟷
(∀(*L1* :: '*l1* ⇒ '*l1'* ⇒ *bool*) (*L1'* :: '*l1'* ⇒ '*l1''* ⇒ *bool*)
(*L2* :: '*l2* ⇒ '*l2'* ⇒ *bool*) (*L2'* :: '*l2'* ⇒ '*l2''* ⇒ *bool*).
(*rel-FGf L1 L2 Co1 Co2 Contra1 Contra2* ::
(-, -, -, -, -, -, '*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *FGf* ⇒ -) *OO*
rel-FGf L1' L2' Co1' Co2' Contra1' Contra2' ≤
rel-FGf (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*))

definition *rel-FGf-neg-distr-cond* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒

```
    ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
    ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
    ('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' ×
      'f1 × 'f2 × 'f3 × 'f4 × 'f5 × 'f6 × 'f7) itself ⇒ bool where
  rel-FGf-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'
- ⟷
  (∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool)
    (L2 :: 'l2 ⇒ 'l2' ⇒ bool) (L2' :: 'l2' ⇒ 'l2'' ⇒ bool).
    rel-FGf (L1 OO L1') (L2 OO L2') (Co1 OO Co1') (Co2 OO Co2')
      (Contra1 OO Contra1') (Contra2 OO Contra2') ≤
    (rel-FGf L1 L2 Co1 Co2 Contra1 Contra2 ::
      (-, -, -, -, -, -,'f1, 'f2, 'f3, 'f4, 'f5, 'f6, 'f7) FGf ⇒ -) OO
      rel-FGf L1' L2' Co1' Co2' Contra1' Contra2')
```

Sufficient conditions for subdistributivity over relation composition.

```
lemma rel-FGf-pos-distr-imp:
  fixes tytok-F :: ('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' × 'f2 × 'f2 × 'f2 ×
      ('f1, 'f2, 'f3, 'f4, 'f5, 'f6, 'f7) G) itself
    and tytok-FGf :: ('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' ×
      'f1 × 'f2 × 'f3 × 'f4 × 'f5 × 'f6 × 'f7) itself
  assumes rel-F-pos-distr-cond Co1 Co1' Co2 Co2' ((=) :: 'f4 ⇒ -) ((=) :: 'f4 ⇒
-)
      Contra1 Contra1' Contra2 Contra2' ((=) :: 'f6 ⇒ -) ((=) :: 'f6 ⇒ -) tytok-F
   shows rel-FGf-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-FGf
  unfolding rel-FGf-pos-distr-cond-def rel-FGf-def
  apply (intro allI)
  apply (rule order-trans)
   apply (rule rel-F-pos-distr)
   apply (rule assms(1))
  apply (rule rel-F-mono)
        apply (simp-all add: eq-OO)
  done

lemma rel-FGf-neg-distr-imp:
  fixes tytok-F :: ('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' × 'f2 × 'f2 × 'f2 ×
      ('f1, 'f2, 'f3, 'f4, 'f5, 'f6, 'f7) G) itself
    and tytok-FGf :: ('l1 × 'l1' × 'l1'' × 'l2 × 'l2' × 'l2'' ×
      'f1 × 'f2 × 'f3 × 'f4 × 'f5 × 'f6 × 'f7) itself
  assumes rel-F-neg-distr-cond Co1 Co1' Co2 Co2' ((=) :: 'f4 ⇒ -) ((=) :: 'f4 ⇒
-)
      Contra1 Contra1' Contra2 Contra2' ((=) :: 'f6 ⇒ -) ((=) :: 'f6 ⇒ -) tytok-F
   shows rel-FGf-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-FGf
  unfolding rel-FGf-neg-distr-cond-def rel-FGf-def
  apply (intro allI)
  apply (rule order-trans[rotated])
   apply (rule rel-F-neg-distr)
```

apply (*rule assms*(*1*))
apply (*rule rel-F-mono*)
apply (*simp-all add*: *eq-OO*)
done

lemma *rel-FGf-pos-distr-cond-eq*:
fixes *tytok* :: ('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* ×
'*f1* × '*f2* × '*f3* × '*f4* × '*f5* × '*f6* × '*f7*) *itself*
shows *rel-FGf-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
by (*intro rel-FGf-pos-distr-imp rel-F-pos-distr-cond-eq*)

lemma *rel-FGf-neg-distr-cond-eq*:
fixes *tytok* :: ('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* ×
'*f1* × '*f2* × '*f3* × '*f4* × '*f5* × '*f6* × '*f7*) *itself*
shows *rel-FGf-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
by (*intro rel-FGf-neg-distr-imp rel-F-neg-distr-cond-eq*)

definition *rell-FGf L1 L2* = *rel-FGf L1 L2* (=) (=) (=) (=)
definition *mapl-FGf l1 l2* = *map-FGf l1 l2 id id id id*

type-synonym ('*co1*, '*co2*, '*contra1*, '*contra2*, '*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *FGfbd*
=
('*co1*, '*co2*, '*f4*, '*contra1*, '*contra2*, '*f6*, ('*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *G*) *Fbd*

definition *set1-FGf* :: ('*l1*, '*l2*, '*co1*, '*co2*, '*contra1*, '*contra2*,
'*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *FGf* ⇒ '*l1 set* **where**
set1-FGf x = *set1-F x*

definition *set2-FGf* :: ('*l1*, '*l2*, '*co1*, '*co2*, '*contra1*, '*contra2*,
'*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *FGf* ⇒ '*l2 set* **where**
set2-FGf x = *set2-F x*

definition *bd-FGf* :: ('*co1*, '*co2*, '*contra1*, '*contra2*, '*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*)
FGfbd rel
where *bd-FGf* = *bd-F*

lemma *set1-FGf-map*: *set1-FGf* ∘ *mapl-FGf l1 l2* = *image l1* ∘ *set1-FGf*
by (*simp add*: *fun-eq-iff set1-FGf-def mapl-FGf-def map-FGf-def mapl-F-def* [*symmetric*]
set1-F-map[*THEN fun-cong*, *simplified*])

lemma *bd-FGf-card-order*: *card-order bd-FGf*
unfolding *bd-FGf-def* **using** *bd-F-card-order* .

lemma *bd-FGf-cinfinite*: *cinfinite bd-FGf*
unfolding *bd-FGf-def* **using** *bd-F-cinfinite* .

lemma
fixes *x* :: (-, -, '*co1*, '*co2*, '*contra1*, '*contra2*, '*f1*, '*f2*, '*f3*, '*f4*, '*f5*, '*f6*, '*f7*) *FGf*
shows *set1-FGf-bound*: *card-of* (*set1-FGf x*) ≤*o* (*bd-FGf* :: ('*co1*, '*co2*, '*contra1*,

'contra2,
'f1, *'f2*, *'f3*, *'f4*, *'f5*, *'f6*, *'f7*) *FGfbd rel*)
and *set2-FGf-bound*: *card-of* (*set2-FGf x*) $\leq o$ (*bd-FGf* :: (*'co1*, *'co2*, *'contra1*, *'contra2*,
'f1, *'f2*, *'f3*, *'f4*, *'f5*, *'f6*, *'f7*) *FGfbd rel*)
unfolding *set1-FGf-def set2-FGf-def bd-FGf-def* **by** (*rule set1-F-bound set2-F-bound*)+

lemma *mapl-FGf-cong*:
assumes $\bigwedge z.\ z \in$ *set1-FGf x* $\Longrightarrow$ *l1 z* = *l1' z* **and** $\bigwedge z.\ z \in$ *set2-FGf x* $\Longrightarrow$ *l2 z* = *l2' z*
shows *mapl-FGf l1 l2 x* = *mapl-FGf l1' l2' x*
unfolding *mapl-FGf-def map-FGf-def mapl-F-def*[*symmetric*]
by (*auto 0 3 intro*: *mapl-F-cong assms simp add*: *set1-FGf-def set2-FGf-def*)

lemma *rell-FGf-mono-strong*:
assumes *rell-FGf L1 L2 x y*
and $\bigwedge a\ b.\ a \in$ *set1-FGf x* $\Longrightarrow$ *b* $\in$ *set1-FGf y* $\Longrightarrow$ *L1 a b* $\Longrightarrow$ *L1' a b*
and $\bigwedge a\ b.\ a \in$ *set2-FGf x* $\Longrightarrow$ *b* $\in$ *set2-FGf y* $\Longrightarrow$ *L2 a b* $\Longrightarrow$ *L2' a b*
shows *rell-FGf L1' L2' x y*
using *assms*(*1*) **unfolding** *rell-FGf-def rel-FGf-def rell-F-def*[*symmetric*]
by (*auto 0 3 intro*: *rell-F-mono-strong assms*(*2*−*3*) *simp add*: *set1-FGf-def set2-FGf-def*)

end

4 Least and greatest fixpoints

theory *Fixpoints* **imports**
Axiomatised-BNF-CC
begin

4.1 Least fixpoint

4.1.1 $\mathbf{BNF_{CC}}$ structure

context notes [[*typedef-overloaded*, *bnf-internals*]]
begin

datatype (*set-T*: *'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *T* =
C-T (*D-T*: ((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *T*, *'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G*)
for
map: *mapl-T*
rel: *rell-T*

end

inductive *rel-T* :: (*'l1* ⇒ *'l1'* ⇒ *bool*) ⇒
(*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒

```
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra2 ⇒ 'contra2' ⇒ bool) ⇒
    ('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ⇒
    ('l1', 'co1', 'co2', 'contra1', 'contra2', 'f) T ⇒ bool
  for L1 Co1 Co2 Contra1 Contra2 where
  rel-T L1 Co1 Co2 Contra1 Contra2 (C-T x) (C-T y)
    if rel-G (rel-T L1 Co1 Co2 Contra1 Contra2) L1 Co1 Co2 Contra1 Contra2 x
y

primrec map-T :: ('l1 ⇒ 'l1') ⇒ ('co1 ⇒ 'co1') ⇒ ('co2 ⇒ 'co2') ⇒
    ('contra1' ⇒ 'contra1) ⇒ ('contra2' ⇒ 'contra2) ⇒
    ('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ⇒
    ('l1', 'co1', 'co2', 'contra1', 'contra2', 'f) T where
  map-T l1 co1 co2 contra1 contra2 (C-T x) =
    C-T (map-G id id co1 co2 contra1 contra2 (mapl-G (map-T l1 co1 co2 contra1
contra2) l1 x))
```

The mapper and relator generated by the datatype package coincide with our generalised definitions restricted to live arguments.

```
lemma rell-T-alt-def: rell-T L1 = rel-T L1 (=) (=) (=) (=)
  apply (intro ext iffI)
   apply (erule T.rel-induct)
   apply (unfold rell-G-def)
   apply (erule rel-T.intros)
  apply (erule rel-T.induct)
  apply (rule T.rel-intros)
  apply (unfold rell-G-def)
  apply (erule rel-G-mono')
      apply (auto)
  done

lemma mapl-T-alt-def: mapl-T l1 = map-T l1 id id id id
  supply id-apply[simp del]
  apply (rule ext)
  subgoal for x
    apply (induction x)
    apply (simp add: mapl-G-def map-G-comp[THEN fun-cong, simplified])
    apply (fold mapl-G-def)
    apply (erule mapl-G-cong)
    apply (rule refl)
    done
  done

lemma rel-T-mono [mono]:
  ⟦ L1 ≤ L1'; Co1 ≤ Co1'; Co2 ≤ Co2'; Contra1' ≤ Contra1; Contra2' ≤ Contra2
⟧ ⟹
  rel-T L1 Co1 Co2 Contra1 Contra2 ≤ rel-T L1' Co1' Co2' Contra1' Contra2'
  apply (rule predicate2I)
  apply (erule rel-T.induct)
  apply (rule rel-T.intros)
```

apply (*erule rel-G-mono'*)
apply (*auto*)
done

lemma *rel-T-eq*: *rel-T* (=) (=) (=) (=) (=) = (=)
unfolding *rell-T-alt-def*[*symmetric*] *T.rel-eq* **..**

lemma *rel-T-conversep*:
rel-T L1^{-1-1} *Co1*$^{-1-1}$ *Co2*$^{-1-1}$ *Contra1*$^{-1-1}$ *Contra2*$^{-1-1}$ = (*rel-T L1 Co1 Co2 Contra1 Contra2*)$^{-1-1}$
apply (*intro ext iffI*)
apply (*simp*)
apply (*erule rel-T.induct*)
apply (*rule rel-T.intros*)
apply (*rewrite conversep-iff*[*symmetric*])
apply (*fold rel-G-conversep*)
apply (*erule rel-G-mono'*; *blast*)
apply (*simp*)
apply (*erule rel-T.induct*)
apply (*rule rel-T.intros*)
apply (*rewrite conversep-iff*[*symmetric*])
apply (*unfold rel-G-conversep*[*symmetric*] *conversep-conversep*)
apply (*erule rel-G-mono'*; *blast*)
done

lemma *map-T-id0*: *map-T id id id id id* = *id*
unfolding *mapl-T-alt-def*[*symmetric*] *T.map-id0* **..**

lemma *map-T-id*: *map-T id id id id id x* = *x*
by (*simp add*: *map-T-id0*)

lemma *map-T-comp*: *map-T l1 co1 co2 contra1 contra2* ∘ *map-T l1' co1' co2' contra1' contra2'* =
map-T (*l1* ∘ *l1'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*)
apply (*rule ext*)
subgoal for *x*
apply (*induction x*)
apply (*simp add*: *mapl-G-def map-G-comp*[*THEN fun-cong*, *simplified*])
apply (*fold comp-def*)
apply (*subst* (*1 2*) *map-G-mapl-G*)
apply (*rule arg-cong*[**where** *f*=*map-G - - - - - -*])
apply (*rule mapl-G-cong*)
apply (*simp-all*)
done
done

lemma *map-T-parametric*: *rel-fun* (*rel-fun L1 L1'*)
(*rel-fun* (*rel-fun Co1 Co1'*) (*rel-fun* (*rel-fun Co2 Co2'*)

```
  (rel-fun (rel-fun Contra1' Contra1) (rel-fun (rel-fun Contra2' Contra2)
  (rel-fun (rel-T L1 Co1 Co2 Contra1 Contra2) (rel-T L1' Co1' Co2' Contra1'
Contra2'))))))
  map-T map-T
  apply (intro rel-funI)
  apply (erule rel-T.induct)
  apply (simp)
  apply (rule rel-T.intros)
  apply (fold map-G-mapl-G)
  apply (erule map-G-rel-cong)
      apply (blast elim: rel-funE)+
  done

definition rel-T-pos-distr-cond :: ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒
bool) ⇒
    ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
    ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
    ('l1 × 'l1' × 'l1'' × 'f) itself ⇒ bool where
  rel-T-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
⟷
  (∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool).
    (rel-T L1 Co1 Co2 Contra1 Contra2 :: (-, -, -, -, -, 'f) T ⇒ -) OO
      rel-T L1' Co1' Co2' Contra1' Contra2' ≤
     rel-T (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1')
(Contra2 OO Contra2'))

definition rel-T-neg-distr-cond :: ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒
bool) ⇒
    ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
    ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
    ('l1 × 'l1' × 'l1'' × 'f) itself ⇒ bool where
  rel-T-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
⟷
  (∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool).
     rel-T (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1')
(Contra2 OO Contra2') ≤
    (rel-T L1 Co1 Co2 Contra1 Contra2 :: (-, -, -, -, -, 'f) T ⇒ -) OO
      rel-T L1' Co1' Co2' Contra1' Contra2')
```

We inherit the conditions for subdistributivity over relation composition via a composition witness, which is derived from a witness for the underlying functor G.

```
primrec rel-T-witness :: ('l1 ⇒ 'l1'' ⇒ bool) ⇒
    ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒ bool) ⇒
    ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
    ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
```

```
    ('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ⇒
    ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T ⇒
    ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T where
  rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' (C-T
x) Cy = C-T
  (mapl-G (λ((x, f), y). f y) id
    (rel-G-witness (λ(x, f) y. rel-T (λx (x', y). x' = x ∧ L1 x y) Co1 Co2 Contra1
Contra2 x (f y) ∧
       rel-T (λ(x, y') y. y' = y ∧ L1 x y) Co1' Co2' Contra1' Contra2' (f y) y)
    L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'
    (mapl-G (λx. (x, rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Con-
tra2 Contra2' x)) id x,
       D-T Cy)))

lemma rel-T-pos-distr-imp:
  fixes Co1 :: 'co1 ⇒ 'co1' ⇒ bool and Co1' :: 'co1' ⇒ 'co1'' ⇒ bool
    and Co2 :: 'co2 ⇒ 'co2' ⇒ bool and Co2' :: 'co2' ⇒ 'co2'' ⇒ bool
     and Contra1 :: 'contra1 ⇒ 'contra1' ⇒ bool and Contra1' :: 'contra1' ⇒
'contra1'' ⇒ bool
     and Contra2 :: 'contra2 ⇒ 'contra2' ⇒ bool and Contra2' :: 'contra2' ⇒
'contra2'' ⇒ bool
    and tytok-G :: (('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ×
      ('l1', 'co1', 'co2', 'contra1', 'contra2', 'f) T ×
     ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T × 'l1 × 'l1' × 'l1'' × 'f) itself
    and tytok-T :: ('l1 × 'l1' × 'l1'' × 'f) itself
  assumes rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-G
  shows rel-T-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Con-
tra2' tytok-T
  unfolding rel-T-pos-distr-cond-def
  apply (intro allI predicate2I)
  apply (erule relcomppE)
  subgoal premises prems for L1 L1' x z y
    using prems apply (induction arbitrary: z)
    apply (erule rel-T.cases)
    apply (simp)
    apply (rule rel-T.intros)
    apply (drule (1) rel-G-pos-distr[THEN predicate2D, OF assms relcomppI])
    apply (erule rel-G-mono'; blast)
    done
  done

lemma
  fixes L1 :: 'l1 ⇒ 'l1'' ⇒ bool
    and Co1 :: 'co1 ⇒ 'co1' ⇒ bool and Co1' :: 'co1' ⇒ 'co1'' ⇒ bool
    and Co2 :: 'co2 ⇒ 'co2' ⇒ bool and Co2' :: 'co2' ⇒ 'co2'' ⇒ bool
     and Contra1 :: 'contra1 ⇒ 'contra1' ⇒ bool and Contra1' :: 'contra1' ⇒
'contra1'' ⇒ bool
     and Contra2 :: 'contra2 ⇒ 'contra2' ⇒ bool and Contra2' :: 'contra2' ⇒
```

```
'contra2'' ⇒ bool
    and tytok-G :: ((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ×
        (('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T
         ⇒ ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) ×
       ((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ×
         (('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T
          ⇒ ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) ×
        ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T) ×
       ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T ×
       'l1 × ('l1 × 'l1'') × 'l1'' × 'f) itself
    and x :: (-, -, -, -, -, 'f) T
  assumes cond: rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-G
    and rel-OO: rel-T L1 (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1') (Contra2 OO Contra2') x y
  shows rel-T-witness1: rel-T (λx (x', y). x' = x ∧ L1 x y) Co1 Co2 Contra1 Contra2 x
      (rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' x y)
    and rel-T-witness2: rel-T (λ(x, y') y. y' = y ∧ L1 x y) Co1' Co2' Contra1' Contra2'
      (rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' x y) y
  using rel-OO apply (induction)
  subgoal premises prems for x y
  proof−
    have x-expansion: x = mapl-G fst id (mapl-G (λx.
      (x, rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' x)) id x)
      by (simp add: mapl-G-def map-G-comp[THEN fun-cong, simplified] map-G-id[unfolded id-def] comp-def)
    show ?thesis
      apply (simp)
      apply (rule rel-T.intros)
      apply (rewrite in rel-G - - - - - - ⌑ - x-expansion)
      apply (rewrite in rel-G - - - - - - - ⌑ mapl-G-def)
      apply (subst mapl-G-def)
      apply (rule map-G-rel-cong)
          apply (rule rel-G-witness1[OF cond])
          apply (rewrite in rel-G - - - - - - ⌑ - mapl-G-def)
          apply (rewrite in rel-G - - - - - - - ⌑ map-G-id[symmetric])
          apply (rule map-G-rel-cong[OF prems])
              apply (clarsimp)+
      done
  qed
  subgoal for x y
    apply (simp)
    apply (rule rel-T.intros)
    apply (rewrite in rel-G - - - - - - ⌑ - mapl-G-def)
```

```
  apply (rewrite in rel-G - - - - - - - ⌑ map-G-id[symmetric])
  apply (rule map-G-rel-cong)
      apply (rule rel-G-witness2[OF cond[unfolded rel-T-neg-distr-cond-def]])
      apply (rewrite in rel-G - - - - - - ⌑ - mapl-G-def)
      apply (rewrite in rel-G - - - - - - - ⌑ map-G-id[symmetric])
      apply (erule map-G-rel-cong)
          apply (clarsimp)+
  done
 done

lemma rel-T-neg-distr-imp:
 fixes Co1 :: 'co1 ⇒ 'co1' ⇒ bool and Co1' :: 'co1' ⇒ 'co1'' ⇒ bool
  and Co2 :: 'co2 ⇒ 'co2' ⇒ bool and Co2' :: 'co2' ⇒ 'co2'' ⇒ bool
   and Contra1 :: 'contra1 ⇒ 'contra1' ⇒ bool and Contra1' :: 'contra1' ⇒
'contra1'' ⇒ bool
   and Contra2 :: 'contra2 ⇒ 'contra2' ⇒ bool and Contra2' :: 'contra2' ⇒
'contra2'' ⇒ bool
  and tytok-G :: ((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ×
      (('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T
       ⇒ ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) ×
     ((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T ×
       (('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T
        ⇒ ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) ×
      ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T) ×
     ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T ×
     'l1 × ('l1 × 'l1'') × 'l1'' × 'f) itself
  and tytok-T :: ('l1 × 'l1' × 'l1'' × 'f) itself
 assumes rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-G
 shows rel-T-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Con-
tra2' tytok-T
 unfolding rel-T-neg-distr-cond-def
proof (intro allI predicate2I relcomppI)
 fix L1 :: 'l1 ⇒ 'l1' ⇒ bool and L1' :: 'l1' ⇒ 'l1'' ⇒ bool
  and x :: (-, -, -, -, -, 'f) T and y :: (-, -, -, -, -, 'f) T
 assume ∗: rel-T (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2')
  (Contra1 OO Contra1') (Contra2 OO Contra2') x y
 let ?z = map-T (relcompp-witness L1 L1') id id id id
   (rel-T-witness (L1 OO L1') Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' x y)
 show rel-T L1 Co1 Co2 Contra1 Contra2 x ?z
  apply(subst map-T-id[symmetric])
  apply(rule map-T-parametric[unfolded rel-fun-def, rule-format, rotated −1])
      apply(rule rel-T-witness1[OF assms ∗])
     apply(auto simp add: vimage2p-def del: relcomppE elim!: relcompp-witness)
  done
 show rel-T L1' Co1' Co2' Contra1' Contra2' ?z y
  apply(rewrite in rel-T - - - - - - ⌑ map-T-id[symmetric])
  apply(rule map-T-parametric[unfolded rel-fun-def, rule-format, rotated −1])
```

```
      apply(rule rel-T-witness2[OF assms *])
     apply(auto simp add: vimage2p-def del: relcomppE elim!: relcompp-witness)
   done
qed

lemma rel-T-pos-distr-cond-eq:
  ⋀tytok. rel-T-pos-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) tytok
  by (intro rel-T-pos-distr-imp rel-G-pos-distr-cond-eq)

lemma rel-T-neg-distr-cond-eq:
  ⋀tytok. rel-T-neg-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) tytok
  by (intro rel-T-neg-distr-imp rel-G-neg-distr-cond-eq)
```

The BNF axioms are proved by the datatype package.

```
thm T.set-map T.bd-card-order T.bd-cinfinite T.set-bd T.map-cong[OF refl]
  T.rel-mono-strong T.wit
```

4.1.2 Parametricity laws

```
context includes lifting-syntax begin

lemma C-T-parametric: (rel-G (rel-T L1 Co1 Co2 Contra1 Contra2) L1 Co1 Co2
Contra1 Contra2 ===>
  rel-T L1 Co1 Co2 Contra1 Contra2) C-T C-T
  by (fast elim: rel-T.intros)

lemma D-T-parametric: (rel-T L1 Co1 Co2 Contra1 Contra2 ===>
  rel-G (rel-T L1 Co1 Co2 Contra1 Contra2) L1 Co1 Co2 Contra1 Contra2) D-T
D-T
  by (fastforce elim: rel-T.cases)

lemma rec-T-parametric:
  ((rel-G (rel-prod (rel-T L1 Co1 Co2 Contra1 Contra2) A) L1 Co1 Co2 Contra1
Contra2 ===> A) ===>
  rel-T L1 Co1 Co2 Contra1 Contra2 ===> A) rec-T rec-T
  apply (intro rel-funI)
  subgoal premises prems for f g x y
    using prems(2) apply (induction)
    apply (simp)
    apply (rule prems(1)[THEN rel-funD])
    apply (unfold mapl-G-def)
    apply (erule map-G-rel-cong)
        apply (auto)
    done
  done

end
```

4.2 Greatest fixpoints

4.2.1 BNF$_{CC}$ structure

context notes [[*typedef-overloaded*, *bnf-internals*]]
begin

codatatype (*set-U*: *'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* =
  C-U (*D-U*: ((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U*, *'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *G*)
  for
    map: *mapl-U*
    rel: *rell-U*

end

coinductive *rel-U* :: (*'l1* ⇒ *'l1'* ⇒ *bool*) ⇒
    (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒
    (*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒
    (*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ⇒
    (*'l1'*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'f*) *U* ⇒ *bool*
  for *L1 Co1 Co2 Contra1 Contra2* **where**
  rel-U L1 Co1 Co2 Contra1 Contra2 x y
    if *rel-G* (*rel-U L1 Co1 Co2 Contra1 Contra2*) *L1 Co1 Co2 Contra1 Contra2* (*D-U x*) (*D-U y*)

primcorec *map-U* :: (*'l1* ⇒ *'l1'*) ⇒ (*'co1* ⇒ *'co1'*) ⇒ (*'co2* ⇒ *'co2'*) ⇒
    (*'contra1'* ⇒ *'contra1*) ⇒ (*'contra2'* ⇒ *'contra2*) ⇒
    (*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ⇒
    (*'l1'*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'f*) *U* **where**
  D-U (*map-U l1 co1 co2 contra1 contra2 x*) =
    mapl-G (*map-U l1 co1 co2 contra1 contra2*) *l1* (*map-G id id co1 co2 contra1 contra2* (*D-U x*))

lemma *rell-U-alt-def*: *rell-U L1* = *rel-U L1* (=) (=) (=) (=)
  apply (*intro ext iffI*)
   apply (*erule rel-U.coinduct*)
   apply (*erule U.rel-cases*)
   apply (*simp add*: *rell-G-def*)
   apply (*erule rel-G-mono'*; *blast*)
  apply (*erule U.rel-coinduct*)
  apply (*erule rel-U.cases*)
  apply (*simp add*: *rell-G-def*)
  done

lemma *mapl-U-alt-def*: *mapl-U l1* = *map-U l1 id id id id*
  supply *id-apply*[*simp del*]
  apply (*rule ext*)
  subgoal for *x*
    apply (*coinduction arbitrary*: *x*)

```
  apply (simp add: mapl-G-def map-G-comp[THEN fun-cong, simplified] U.map-sel)
   apply (unfold rell-G-def)
   apply (rule map-G-rel-cong[OF rel-G-eq-refl])
       apply (auto)
   done
 done

lemma rel-U-mono [mono]:
 ⟦ L1 ≤ L1'; Co1 ≤ Co1'; Co2 ≤ Co2'; Contra1' ≤ Contra1; Contra2' ≤ Contra2
⟧ ⟹
 rel-U L1 Co1 Co2 Contra1 Contra2 ≤ rel-U L1' Co1' Co2' Contra1' Contra2'
 apply (rule predicate2I)
 apply (erule rel-U.coinduct[of rel-U L1 Co1 Co2 Contra1 Contra2])
 apply (erule rel-U.cases)
 apply (simp)
 apply (erule rel-G-mono')
     apply (blast)+
 done

lemma rel-U-eq: rel-U (=) (=) (=) (=) (=) = (=)
 unfolding rell-U-alt-def[symmetric] U.rel-eq ..

lemma rel-U-conversep:
  rel-U L1^-1-1 Co1^-1-1 Co2^-1-1 Contra1^-1-1 Contra2^-1-1 = (rel-U L1 Co1
Co2 Contra1 Contra2)^-1-1
 apply (intro ext iffI)
  apply (simp)
  apply (erule rel-U.coinduct)
  apply (erule rel-U.cases)
  apply (simp del: conversep-iff)
  apply (rewrite conversep-iff[symmetric])
  apply (fold rel-G-conversep)
  apply (erule rel-G-mono'; blast)
 apply (erule rel-U.coinduct)
 apply (subst (asm) conversep-iff)
 apply (erule rel-U.cases)
 apply (simp del: conversep-iff)
 apply (rewrite conversep-iff[symmetric])
 apply (unfold rel-G-conversep[symmetric] conversep-conversep)
 apply (erule rel-G-mono'; blast)
 done

lemma map-U-id0: map-U id id id id id = id
 unfolding mapl-U-alt-def[symmetric] U.map-id0 ..

lemma map-U-id: map-U id id id id id x = x
 by (simp add: map-U-id0)

lemma map-U-comp: map-U l1 co1 co2 contra1 contra2 ∘ map-U l1' co1' co2'
```

```
contra1' contra2' =
  map-U (l1 ∘ l1') (co1 ∘ co1') (co2 ∘ co2') (contra1' ∘ contra1) (contra2' ∘
contra2)
 apply (rule ext)
 subgoal for x
  apply (coinduction arbitrary: x)
  apply (simp add: mapl-G-def map-G-comp[THEN fun-cong, simplified])
  apply (unfold rell-G-def)
  apply (rule map-G-rel-cong[OF rel-G-eq-refl])
     apply (auto)
  done
 done

lemma map-U-parametric: rel-fun (rel-fun L1 L1')
 (rel-fun (rel-fun Co1 Co1') (rel-fun (rel-fun Co2 Co2')
 (rel-fun (rel-fun Contra1' Contra1) (rel-fun (rel-fun Contra2' Contra2)
 (rel-fun (rel-U L1 Co1 Co2 Contra1 Contra2) (rel-U L1' Co1' Co2' Contra1'
Contra2'))))))
 map-U map-U
 apply (intro rel-funI)
 apply (coinduction)
 apply (simp add: mapl-G-def map-G-comp[THEN fun-cong, simplified])
 apply (erule rel-U.cases)
 apply (hypsubst)
 apply (erule map-G-rel-cong)
    apply (blast elim: rel-funE)+
 done

definition rel-U-pos-distr-cond :: ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒
bool) ⇒
   ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
   ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
   ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
   ('l1 × 'l1' × 'l1'' × 'f) itself ⇒ bool where
 rel-U-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
⟷
 (∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool).
   (rel-U L1 Co1 Co2 Contra1 Contra2 :: (-, -, -, -, -, 'f) U ⇒ -) OO
     rel-U L1' Co1' Co2' Contra1' Contra2' ≤
    rel-U (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1')
(Contra2 OO Contra2'))

definition rel-U-neg-distr-cond :: ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒
bool) ⇒
   ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
   ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
   ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
   ('l1 × 'l1' × 'l1'' × 'f) itself ⇒ bool where
 rel-U-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
```

```
⟷
  (∀(L1 :: 'l1 ⇒ 'l1' ⇒ bool) (L1' :: 'l1' ⇒ 'l1'' ⇒ bool).
    rel-U (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1')
(Contra2 OO Contra2') ≤
    (rel-U L1 Co1 Co2 Contra1 Contra2 :: (-, -, -, -, -, 'f) U ⇒ -) OO
      rel-U L1' Co1' Co2' Contra1' Contra2')

primcorec rel-U-witness :: ('l1 ⇒ 'l1'' ⇒ bool) ⇒
    ('co1 ⇒ 'co1' ⇒ bool) ⇒ ('co1' ⇒ 'co1'' ⇒ bool) ⇒
    ('co2 ⇒ 'co2' ⇒ bool) ⇒ ('co2' ⇒ 'co2'' ⇒ bool) ⇒
    ('contra1 ⇒ 'contra1' ⇒ bool) ⇒ ('contra1' ⇒ 'contra1'' ⇒ bool) ⇒
    ('contra2 ⇒ 'contra2' ⇒ bool) ⇒ ('contra2' ⇒ 'contra2'' ⇒ bool) ⇒
    ('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) U ×
    ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) U ⇒
    ('l1 × 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) U where
  D-U (rel-U-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'
xy) =
  mapl-G (rel-U-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Con-
tra2') id
    (rel-G-witness (rel-U L1 (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Con-
tra1') (Contra2 OO Contra2'))
    L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' (D-U (fst xy),
D-U (snd xy)))

lemma rel-U-pos-distr-imp:
  fixes Co1 :: 'co1 ⇒ 'co1' ⇒ bool and Co1' :: 'co1' ⇒ 'co1'' ⇒ bool
    and Co2 :: 'co2 ⇒ 'co2' ⇒ bool and Co2' :: 'co2' ⇒ 'co2'' ⇒ bool
    and Contra1 :: 'contra1 ⇒ 'contra1' ⇒ bool and Contra1' :: 'contra1' ⇒
'contra1'' ⇒ bool
    and Contra2 :: 'contra2 ⇒ 'contra2' ⇒ bool and Contra2' :: 'contra2' ⇒
'contra2'' ⇒ bool
    and tytok-G :: (('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) U ×
      ('l1', 'co1', 'co2', 'contra1', 'contra2', 'f) U ×
      ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) U × 'l1 × 'l1' × 'l1'' × 'f)
itself
    and tytok-T :: ('l1 × 'l1' × 'l1'' × 'f) itself
  assumes rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-G
  shows rel-U-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Con-
tra2' tytok-T
  unfolding rel-U-pos-distr-cond-def
  apply (intro allI predicate2I)
  apply (erule relcomppE)
  subgoal premises prems for L1 L1' x z y
    using prems apply (coinduction arbitrary: x y z)
    apply (simp)
    apply (rule rel-G-pos-distr[THEN predicate2D,
          OF assms relcomppI, THEN rel-G-mono'])
          apply (auto elim: rel-U.cases)
```

done
done

lemma *rel-U-witness1*:
fixes *L1* :: *'l1* ⇒ *'l1''* ⇒ *bool*
and *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-G* :: ((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ×
((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ×
(*'l1''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f*) *U*) ×
(*'l1''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f*) *U* ×
'l1 × (*'l1* × *'l1''*) × *'l1''* × *'f*) *itself*
and *x* :: (-, -, -, -, -, *'f*) *U*
assumes *cond*: *rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-G*
and *rel-OO*: *rel-U L1* (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) *x y*
shows *rel-U* (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L1 x y*) *Co1 Co2 Contra1 Contra2 x*
(*rel-U-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (*x*, *y*))
using *rel-OO* **apply** (*coinduction arbitrary*: *x y*)
apply (*erule rel-U.cases*)
apply (*clarsimp*)
apply (*rewrite* **in** *rel-G - - - - - - ⌑ - map-G-id*[*symmetric*])
apply (*subst mapl-G-def*)
apply (*rule map-G-rel-cong*)
apply (*erule rel-G-witness1*[*OF cond*])
apply (*auto*)
done

lemma *rel-U-witness2*:
fixes *L1* :: *'l1* ⇒ *'l1''* ⇒ *bool*
and *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-G* :: ((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ×
((*'l1*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *U* ×
(*'l1''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f*) *U*) ×
(*'l1''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f*) *U* ×
'l1 × (*'l1* × *'l1''*) × *'l1''* × *'f*) *itself*
and *x* :: (-, -, -, -, -, *'f*) *U*
assumes *cond*: *rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Con-*

```
tra2 Contra2' tytok-G
   and rel-OO: rel-U L1 (Co1 OO Co1') (Co2 OO Co2') (Contra1 OO Contra1')
(Contra2 OO Contra2') x y
  shows rel-U (λ(x, y') y. y' = y ∧ L1 x y) Co1' Co2' Contra1' Contra2'
      (rel-U-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'
(x, y)) y
  using rel-OO apply (coinduction arbitrary: x y)
  apply (erule rel-U.cases)
  apply (clarsimp)
  apply (rewrite in rel-G - - - - - - - ⌑ map-G-id[symmetric])
  apply (subst mapl-G-def)
  apply (rule map-G-rel-cong)
       apply (erule rel-G-witness2[OF cond])
      apply (auto)
  done

lemma rel-U-neg-distr-imp:
  fixes Co1 :: 'co1 ⇒ 'co1' ⇒ bool and Co1' :: 'co1' ⇒ 'co1'' ⇒ bool
    and Co2 :: 'co2 ⇒ 'co2' ⇒ bool and Co2' :: 'co2' ⇒ 'co2'' ⇒ bool
     and Contra1 :: 'contra1 ⇒ 'contra1' ⇒ bool and Contra1' :: 'contra1' ⇒
'contra1'' ⇒ bool
     and Contra2 :: 'contra2 ⇒ 'contra2' ⇒ bool and Contra2' :: 'contra2' ⇒
'contra2'' ⇒ bool
    and tytok-G :: (('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) U ×
         (('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) U ×
          ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) U) ×
         ('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) U ×
         'l1 × ('l1 × 'l1'') × 'l1'' × 'f) itself
    and tytok-T :: ('l1 × 'l1' × 'l1'' × 'f) itself
   assumes rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' tytok-G
  shows rel-U-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Con-
tra2' tytok-T
  unfolding rel-U-neg-distr-cond-def
proof (intro allI predicate2I relcomppI)
  fix L1 :: 'l1 ⇒ 'l1' ⇒ bool and L1' :: 'l1' ⇒ 'l1'' ⇒ bool
    and x :: (-, -, -, -, -, 'f) U and y :: (-, -, -, -, -, 'f) U
  assume ∗: rel-U (L1 OO L1') (Co1 OO Co1') (Co2 OO Co2')
    (Contra1 OO Contra1') (Contra2 OO Contra2') x y
  let ?z = map-U (relcompp-witness L1 L1') id id id id
     (rel-U-witness (L1 OO L1') Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2
Contra2' (x, y))
  show rel-U L1 Co1 Co2 Contra1 Contra2 x ?z
    apply(subst map-U-id[symmetric])
    apply(rule map-U-parametric[unfolded rel-fun-def, rule-format, rotated −1])
        apply(rule rel-U-witness1[OF assms ∗])
       apply(auto simp add: vimage2p-def del: relcomppE elim!: relcompp-witness)
    done
  show rel-U L1' Co1' Co2' Contra1' Contra2' ?z y
```

```
    apply(rewrite in rel-U - - - - - - ⌑ map-U-id[symmetric])
    apply(rule map-U-parametric[unfolded rel-fun-def, rule-format, rotated −1])
       apply(rule rel-U-witness2[OF assms ∗])
      apply(auto simp add: vimage2p-def del: relcomppE elim!: relcompp-witness)
    done
qed

lemma rel-U-pos-distr-cond-eq:
  ⋀tytok. rel-U-pos-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) tytok
  by (intro rel-U-pos-distr-imp rel-G-pos-distr-cond-eq)

lemma rel-U-neg-distr-cond-eq:
  ⋀tytok. rel-U-neg-distr-cond (=) (=) (=) (=) (=) (=) (=) (=) tytok
  by (intro rel-U-neg-distr-imp rel-G-neg-distr-cond-eq)
```

The BNF axioms are proved by the datatype package.

```
thm U.set-map U.bd-card-order U.bd-cinfinite U.set-bd U.map-cong[OF refl]
  U.rel-mono-strong U.wit
```

4.2.2 Parametricity laws

```
context includes lifting-syntax begin

lemma C-U-parametric: (rel-G (rel-U L1 Co1 Co2 Contra1 Contra2) L1 Co1 Co2
Contra1 Contra2 ===>
  rel-U L1 Co1 Co2 Contra1 Contra2) C-U C-U
  by (fastforce intro: rel-U.intros)

lemma D-U-parametric: (rel-U L1 Co1 Co2 Contra1 Contra2 ===>
  rel-G (rel-U L1 Co1 Co2 Contra1 Contra2) L1 Co1 Co2 Contra1 Contra2) D-U
D-U
  by (fast elim: rel-U.cases)

lemma corec-U-parametric:
  ((A ===> rel-G (rel-sum (rel-U L1 Co1 Co2 Contra1 Contra2) A) L1 Co1 Co2
Contra1 Contra2) ===>
  A ===> rel-U L1 Co1 Co2 Contra1 Contra2) corec-U corec-U
  apply (intro rel-funI)
  subgoal premises prems for f g x y
    using prems(2) apply (coinduction arbitrary: x y)
    apply (simp)
    apply (unfold mapl-G-def)
    apply (rule map-G-rel-cong)
         apply (erule prems(1)[THEN rel-funD])
        apply (fastforce elim: rel-sum.cases)
       apply (simp-all)
    done
  done
```

end

end

5 Subtypes

theory *Subtypes* **imports**
Axiomatised-BNF-CC
HOL−Library.BNF-Axiomatization
begin

5.1 $\mathbf{BNF_{CC}}$ structure

consts P :: ('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) $G \Rightarrow$ *bool*

axiomatization where
P-map: $\bigwedge x\ l1\ l2\ co1\ co2\ contra1\ contra2.\ P\ x \Longrightarrow P$ (*map-G l1 l2 co1 co2 contra1 contra2 x*)
— $\{x.\ P\ x\}$ is closed under the mapper of G
and
ex-P: $\exists x.\ P\ x$ — $\{x.\ P\ x\}$ is non-empty

typedef (**overloaded**) ('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) S =
$\{x$ :: ('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) $G.\ P\ x\}$ **by** (*simp add*: *ex-P*)

The subtype S is isomorphic to the set $\{x.\ P\ x\}$.

context includes *lifting-syntax*
begin

definition *rel-S* :: ('*live1* ⇒ '*live1'* ⇒ *bool*) ⇒ ('*live2* ⇒ '*live2'* ⇒ *bool*) ⇒
('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒
('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) *S* ⇒
('*live1'*, '*live2'*, '*co1'*, '*co2'*, '*contra1'*, '*contra2'*, '*fixed*) *S* ⇒ *bool*
where
rel-S L1 L2 Co1 Co2 Contra1 Contra2 = *vimage2p Rep-S Rep-S* (*rel-G L1 L2 Co1 Co2 Contra1 Contra2*)

definition *map-S* :: ('*live1* ⇒ '*live1'*) ⇒ ('*live2* ⇒ '*live2'*) ⇒
('*co1* ⇒ '*co1'*) ⇒ ('*co2* ⇒ '*co2'*) ⇒
('*contra1'* ⇒ '*contra1*) ⇒ ('*contra2'* ⇒ '*contra2*) ⇒
('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) *S* ⇒
('*live1'*, '*live2'*, '*co1'*, '*co2'*, '*contra1'*, '*contra2'*, '*fixed*) *S*
where
map-S = (*id* −−−> *id* −−−> *id* −−−> *id* −−−> *id* −−−> *id* −−−> *Rep-S* −−−> *Abs-S*) *map-G*

lemma *rel-S-mono*:

⟦ *L1* ≤ *L1'*; *L2* ≤ *L2'*; *Co1* ≤ *Co1'*; *Co2* ≤ *Co2'*; *Contra1'* ≤ *Contra1*; *Contra2'* ≤ *Contra2* ⟧
⟹ *rel-S L1 L2 Co1 Co2 Contra1 Contra2* ≤ *rel-S L1' L2' Co1' Co2' Contra1' Contra2'*
unfolding *rel-S-def*
apply(*rule predicate2I*)
apply(*simp add*: *vimage2p-def*)
by(*erule rel-G-mono'*)

lemma *rel-S-eq*: *rel-S* (=) (=) (=) (=) (=) (=) = (=)
unfolding *rel-S-def* **by**(*clarsimp simp add*: *vimage2p-def fun-eq-iff rel-G-eq Rep-S-inject*)

lemma *rel-S-conversep*:
rel-S $L1^{-1-1}$ $L2^{-1-1}$ $Co1^{-1-1}$ $Co2^{-1-1}$ $Contra1^{-1-1}$ $Contra2^{-1-1}$ = (*rel-S L1 L2 Co1 Co2 Contra1 Contra2*)$^{-1-1}$
unfolding *rel-S-def* **apply**(*simp add*: *vimage2p-def*)
apply(*subst rel-G-conversep*)
apply(*simp add*: *map-fun-def fun-eq-iff*)
done

lemma *map-S-id0*: *map-S id id id id id id* = *id*
by(*simp add*: *map-S-def fun-eq-iff map-G-id Rep-S-inverse*)

lemma *map-S-id*: *map-S id id id id id id x* = *x*
by (*simp add*: *map-S-id0*)

lemma *map-S-comp*:
map-S l1 l2 co1 co2 contra1 contra2 ∘ *map-S l1' l2' co1' co2' contra1' contra2'* =
map-S (*l1* ∘ *l1'*) (*l2* ∘ *l2'*) (*co1* ∘ *co1'*) (*co2* ∘ *co2'*) (*contra1'* ∘ *contra1*) (*contra2'* ∘ *contra2*)
apply (*rule ext*)
apply (*simp add*: *map-S-def*)
apply (*subst Abs-S-inverse*)
subgoal for *x* **using** *Rep-S*[*of x*] **by**(*simp add*: *P-map*)
apply (*subst map-G-comp*[*THEN fun-cong*, *simplified*])
apply *simp*
done

lemma *map-S-parametric*:
((*L1* ===> *L1'*) ===> (*L2* ===> *L2'*) ===> (*Co1* ===> *Co1'*) ===> (*Co2* ===> *Co2'*) ===>
(*Contra1'* ===> *Contra1*) ===> (*Contra2'* ===> *Contra2*) ===>
rel-S L1 L2 Co1 Co2 Contra1 Contra2 ===> *rel-S L1' L2' Co1' Co2' Contra1' Contra2'*)
map-S map-S
apply(*rule rel-funI*)+
unfolding *rel-S-def map-S-def*
apply(*simp add*: *vimage2p-def*)

apply(*subst Abs-S-inverse*)
subgoal for ... *x* - **using** *Rep-S*[*of x*] **by**(*simp add*: *P-map*)
apply(*subst Abs-S-inverse*)
subgoal for ... *y* **using** *Rep-S*[*of y*] **by**(*simp add*: *P-map*)
by(*erule map-G-parametric*[*THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*,
THEN rel-funD, *THEN rel-funD*, *THEN rel-funD*, *rotated* −*1*])

lemmas *map-S-rel-cong* = *map-S-parametric*[*unfolded rel-fun-def*, *rule-format*, *rotated* −*1*]

end

definition *rel-S-pos-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself* ⇒ *bool* **where**
rel-S-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*)
(*L2* :: *'l2* ⇒ *'l2'* ⇒ *bool*) (*L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*).
(*rel-S L1 L2 Co1 Co2 Contra1 Contra2* :: (-, -, -, -, -, -, *'f*) *S* ⇒ -) *OO*
rel-S L1' L2' Co1' Co2' Contra1' Contra2' ≤
rel-S (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*))

definition *rel-S-neg-distr-cond* :: (*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself* ⇒ *bool* **where**
rel-S-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -⟷
(∀(*L1* :: *'l1* ⇒ *'l1'* ⇒ *bool*) (*L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*)
(*L2* :: *'l2* ⇒ *'l2'* ⇒ *bool*) (*L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*).
rel-S (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*)
(*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) ≤
(*rel-S L1 L2 Co1 Co2 Contra1 Contra2* :: (-, -, -, -, -, -, *'f*) *S* ⇒ -) *OO*
rel-S L1' L2' Co1' Co2' Contra1' Contra2')

axiomatization where
rel-S-neg-distr-cond-eq:
⋀*tytok*. *rel-S-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*

The subtype inherits the conditions for positive subdistributivity.

lemma *rel-S-pos-distr-imp*:

fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-G* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself*
and *tytok-S* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself*
assumes *rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-G*
shows *rel-S-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-S*
unfolding *rel-S-pos-distr-cond-def rel-S-def*
apply(*simp add*: *vimage2p-def*)
apply(*intro allI predicate2I*)
apply(*clarsimp*)
apply(*rule rel-G-pos-distr*[*THEN predicate2D*])
apply(*rule assms*)
apply(*rule relcomppI*)
apply *simp*
apply *simp*
done

lemma *rel-S-pos-distr-cond-eq*:
⋀*tytok*. *rel-S-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
by (*intro rel-S-pos-distr-imp rel-G-pos-distr-cond-eq*)

lemmas
rel-S-pos-distr = *rel-S-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-S-neg-distr = *rel-S-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

The following composition witness depends only on the abstract condition *rel-S-neg-distr-cond*, without additional assumptions.

consts
rel-S-witness :: (*'l1* ⇒ *'l1''* ⇒ *bool*) ⇒ (*'l2* ⇒ *'l2''* ⇒ *bool*) ⇒
(*'co1* ⇒ *'co1'* ⇒ *bool*) ⇒ (*'co1'* ⇒ *'co1''* ⇒ *bool*) ⇒
(*'co2* ⇒ *'co2'* ⇒ *bool*) ⇒ (*'co2'* ⇒ *'co2''* ⇒ *bool*) ⇒
(*'contra1* ⇒ *'contra1'* ⇒ *bool*) ⇒ (*'contra1'* ⇒ *'contra1''* ⇒ *bool*) ⇒
(*'contra2* ⇒ *'contra2'* ⇒ *bool*) ⇒ (*'contra2'* ⇒ *'contra2''* ⇒ *bool*) ⇒
(*'l1*, *'l2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'f*) *S* ×
(*'l1''*, *'l2''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'f*) *S* ⇒
(*'l1* × *'l1''*, *'l2* × *'l2''*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'f*) *S*

specification (*rel-S-witness*)
rel-S-witness1: ⋀*L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'*
(*tytok* :: (*'l1* × (*'l1* × *'l1''*) × *'l1''* × *'l2* × (*'l2* × *'l2''*) × *'l2''* × *'f*) *itself*)
(*x* :: (*'l1*, *'l2*, -, -, -, -, *'f*) *S*) (*y* :: (*'l1''*, *'l2''*, -, -, -, -, *'f*) *S*).
⟦ *rel-S-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*;

rel-S L1 L2 (Co1 OO Co1′) (Co2 OO Co2′) (Contra1 OO Contra1′) (Contra2 OO Contra2′) x y ⟧ ⟹
rel-S (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L1 x y*) (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L2 x y*) *Co1 Co2 Contra1 Contra2 x*
(*rel-S-witness L1 L2 Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′* (*x*, *y*))
rel-S-witness2:⋀*L1 L2 Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′*
(*tytok* :: (*'l1* × (*'l1* × *'l1″*) × *'l1″* × *'l2* × (*'l2* × *'l2″*) × *'l2″* × *'f*) *itself*)
(*x* :: (*'l1*, *'l2*, -, -, -, -, *'f*) *S*) (*y* :: (*'l1″*, *'l2″*, -, -, -, -, *'f*) *S*).
⟦ *rel-S-neg-distr-cond Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′ tytok*;
rel-S L1 L2 (Co1 OO Co1′) (Co2 OO Co2′) (Contra1 OO Contra1′) (Contra2 OO Contra2′) x y ⟧ ⟹
rel-S (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L1 x y*) (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L2 x y*) *Co1′ Co2′ Contra1′ Contra2′*
(*rel-S-witness L1 L2 Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′* (*x*, *y*)) *y*
apply(*rule exI*[**where** *x*=λ*L1 L2 Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′* (*x*, *y*). *SOME z*.
rel-S (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L1 x y*) (λ*x* (*x′*, *y*). *x′* = *x* ∧ *L2 x y*) *Co1 Co2 Contra1 Contra2 x z* ∧
rel-S (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L1 x y*) (λ(*x*, *y′*) *y*. *y′* = *y* ∧ *L2 x y*) *Co1′ Co2′ Contra1′ Contra2′ z y*])
apply(*fold all-conj-distrib*)
apply(*rule allI*)+
apply(*fold imp-conjR*)
apply(*rule impI*)+
apply *clarify*
apply(*rule someI-ex*)
subgoal for *L1 L2 Co1 Co1′ Co2 Co2′ Contra1 Contra1′ Contra2 Contra2′ x y*
apply(*drule rel-S-neg-distr*[**where**
?L1.0 = λ*x* (*x′*, *y*). *x′* = *x* ∧ *L1 x y* **and** *?L1′.0* = λ(*x*, *y*) *y′*. *y* = *y′* ∧ *L1 x y′* **and**
?L2.0 = λ*x* (*x′*, *y*). *x′* = *x* ∧ *L2 x y* **and** *?L2′.0* = λ(*x*, *y*) *y′*. *y* = *y′* ∧ *L2 x y′*])
apply(*drule predicate2D*)
apply(*erule rel-S-mono*[*THEN predicate2D*, *rotated* −*1*]; *fastforce*)
apply(*erule relcomppE*)
apply(*rule exI conjI*)+
apply *assumption*+
done
done

definition *set1-S* :: (*'live1*, *'live2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'fixed*) *S* ⇒ *'live1 set*
where *set1-S* = *set1-G* ∘ *Rep-S*

definition *set2-S* :: (*'live1*, *'live2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'fixed*) *S* ⇒ *'live2 set*

where *set2-S = set2-G ∘ Rep-S*

lemma *rel-S-alt*:
rel-S L1 L2 (=) (=) (=) (=) x y ⟷ (∃ z. (set1-S z ⊆ {(x, y). L1 x y} ∧ set2-S z ⊆ {(x, y). L2 x y}) ∧ map-S fst fst id id id id z = x ∧ map-S snd snd id id id id z = y)
unfolding *set1-S-def set2-S-def o-def*
apply(*rule iffI*)
subgoal
apply(*subst (asm) (3 4 5 7) OO-eq*[*symmetric*])
apply(*rule exI*[**where** *x=rel-S-witness L1 L2 (=) (=) (=) (=) (=) (=) (=) (=) (x, y)*])
apply(*frule rel-S-witness1*[*OF rel-S-neg-distr-cond-eq*])
apply(*drule rel-S-witness2*[*OF rel-S-neg-distr-cond-eq*])
apply(*auto simp add*: *rel-S-def vimage2p-def rell-G-def*[*symmetric*])
apply(*drule (1) G.Domainp-rel*[*THEN eq-refl*, *THEN predicate1D*, *OF DomainPI*, *unfolded pred-G-def*, *THEN conjunct1*, *THEN bspec*, *of conversep - conversep -*, *unfolded G.rel-conversep Domainp-conversep*, *unfolded conversep-iff*])
apply(*simp add*: *Rangep.simps*)
apply(*drule (1) G.Domainp-rel*[*THEN eq-refl*, *THEN predicate1D*, *OF DomainPI*, *unfolded pred-G-def*, *THEN conjunct2*, *THEN bspec*, *of conversep - conversep -*, *unfolded G.rel-conversep Domainp-conversep*, *unfolded conversep-iff*])
apply(*simp add*: *Rangep.simps*)
apply(*rewrite* **in** - = ⌑ *map-S-id*[*symmetric*])
apply(*rule sym*)
apply(*subst rel-S-eq*[*symmetric*])
apply(*rule map-S-parametric*[*THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *rotated −1*])
apply(*simp add*: *rel-S-def vimage2p-def*)
apply(*subst rell-G-def*[*symmetric*])
apply *assumption*
apply(*simp-all add*: *rel-fun-def*)
apply(*rewrite* **in** - = ⌑ *map-S-id*[*symmetric*])
apply(*subst rel-S-eq*[*symmetric*])
apply(*rule map-S-parametric*[*THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *THEN rel-funD*, *rotated −1*])
apply(*simp add*: *rel-S-def vimage2p-def*)
apply(*subst rell-G-def*[*symmetric*])
apply *assumption*
apply(*simp-all add*: *rel-fun-def*)
done
subgoal
apply(*elim conjE exE*)
apply *hypsubst*

```
    apply(rule map-S-parametric[where ?L1.0=eq-onp (λ(x, y). L1 x y) and
          ?L2.0=eq-onp (λ(x, y). L2 x y), THEN rel-funD, THEN rel-funD, THEN
rel-funD,
         THEN rel-funD, THEN rel-funD, THEN rel-funD, THEN rel-funD, rotated
−1])
          apply(simp add: rel-S-def vimage2p-def)
          apply(subst rell-G-def[symmetric])
          apply(rule G.rel-refl-strong)
           apply(drule (1) subsetD)
           apply(simp add: eq-onp-def)
          apply(drule (1) subsetD)
          apply(simp add: eq-onp-def)
         apply(simp-all add: rel-fun-def eq-onp-def)
    done
  done

bnf ('live1, 'live2, 'co1, 'co2, 'contra1, 'contra2, 'fixed) S
  map: λl1 l2. map-S l1 l2 id id id id
  sets: set1-S    set2-S
  bd: bd-G :: ('co1, 'co2, 'contra1, 'contra2, 'fixed) Gbd rel
  rel: λL1 L2. rel-S L1 L2 (=) (=) (=) (=)
  subgoal by (rule map-S-id0)
  subgoal by (simp add: map-S-comp)
  subgoal
    apply(simp add: map-S-def set1-S-def set2-S-def)
    apply(rule arg-cong[where f=Abs-S])
    apply(fold mapl-G-def)
    apply(rule G.map-cong[OF refl])
     apply simp-all
    done
  subgoal
    apply(simp add: set1-S-def map-S-def fun-eq-iff)
    apply(subst Abs-S-inverse)
    subgoal for x using Rep-S[of x] by(simp add: P-map)
    apply(simp add: G.set-map mapl-G-def[symmetric])
    done
  subgoal
    apply(simp add: set2-S-def map-S-def fun-eq-iff)
    apply(subst Abs-S-inverse)
    subgoal for x using Rep-S[of x] by(simp add: P-map)
    apply(simp add: G.set-map mapl-G-def[symmetric])
    done
  subgoal by(rule bd-G-card-order)
  subgoal by(rule bd-G-cinfinite)
  subgoal
    apply (simp add: set1-S-def)
    apply (rule set1-G-bound)
    done
  subgoal
```

apply (*simp add*: *set2-S-def*)
apply (*rule set2-G-bound*)
done
subgoal
apply(*subst* (*23 24 25 27*) *eq-OO*[*symmetric*])
apply(*rule rel-S-pos-distr-cond-def*[*THEN iffD1*, *rule-format*])
apply(*rule rel-S-pos-distr-imp*)
apply(*rule rel-G-pos-distr-cond-eq*)
done
subgoal
apply(*rule ext*)+
apply(*rule rel-S-alt*)
done
done

5.2 Closedness under zippings

lemma *P-zip-closed*: — This is **lift-bnf**'s property that is too strong.
assumes *P* (*mapl-G fst fst z*) *P* (*mapl-G snd snd z*)
shows *P z*
oops

consts *rel-S-neg-distr-cond'* :: ('*co1* ⇒ '*co1'* ⇒ *bool*) ⇒ ('*co1'* ⇒ '*co1''* ⇒ *bool*) ⇒
('*co2* ⇒ '*co2'* ⇒ *bool*) ⇒ ('*co2'* ⇒ '*co2''* ⇒ *bool*) ⇒
('*contra1* ⇒ '*contra1'* ⇒ *bool*) ⇒ ('*contra1'* ⇒ '*contra1''* ⇒ *bool*) ⇒
('*contra2* ⇒ '*contra2'* ⇒ *bool*) ⇒ ('*contra2'* ⇒ '*contra2''* ⇒ *bool*) ⇒
('*l1* × '*l1'* × '*l1''* × '*l2* × '*l2'* × '*l2''* × '*f*) *itself* ⇒ *bool*

If the set $\{x.\ P\ x\}$ is closed under zippings for *rel-S-neg-distr-cond'*, we inherit the condition for negative subdistributivity from *G*.

axiomatization where
P-rel-G-zipping: ⋀(*L1* :: '*l1* ⇒ '*l1''* ⇒ *bool*) (*L2* :: '*l2* ⇒ '*l2''* ⇒ *bool*)
Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'
(*tytok* :: ('*l1* × ('*l1* × '*l1''*) × '*l1''* × '*l2* × ('*l2* × '*l2''*) × '*l2''* × '*f*) *itself*) *x y z*.
⟦ *P x*; *P y*;
rel-G L1 L2 (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) *x y*;
rel-G (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L1 x y*) (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L2 x y*) *Co1 Co2 Contra1 Contra2 x z*;
rel-G (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L1 x y*) (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L2 x y*) *Co1' Co2' Contra1' Contra2' z y*;
rel-S-neg-distr-cond' Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok ⟧
⟹ *P z*
and
rel-S-neg-distr-cond'-stronger: ⋀*Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*.

rel-S-neg-distr-cond' Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok $\Longrightarrow$
rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok
and
rel-S-neg-distr-cond'-eq:
$\bigwedge$*tytok*. *rel-S-neg-distr-cond'* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*

context includes *lifting-syntax*
begin

definition *rel-S-witness'* :: (*'live1* $\Rightarrow$ *'live1''* $\Rightarrow$ *bool*) $\Rightarrow$ (*'live2* $\Rightarrow$ *'live2''* $\Rightarrow$ *bool*) $\Rightarrow$
(*'co1* $\Rightarrow$ *'co1'* $\Rightarrow$ *bool*) $\Rightarrow$ (*'co1'* $\Rightarrow$ *'co1''* $\Rightarrow$ *bool*) $\Rightarrow$
(*'co2* $\Rightarrow$ *'co2'* $\Rightarrow$ *bool*) $\Rightarrow$ (*'co2'* $\Rightarrow$ *'co2''* $\Rightarrow$ *bool*) $\Rightarrow$
(*'contra1* $\Rightarrow$ *'contra1'* $\Rightarrow$ *bool*) $\Rightarrow$ (*'contra1'* $\Rightarrow$ *'contra1''* $\Rightarrow$ *bool*) $\Rightarrow$
(*'contra2* $\Rightarrow$ *'contra2'* $\Rightarrow$ *bool*) $\Rightarrow$ (*'contra2'* $\Rightarrow$ *'contra2''* $\Rightarrow$ *bool*) $\Rightarrow$
(*'live1*, *'live2*, *'co1*, *'co2*, *'contra1*, *'contra2*, *'fixed*) *S* $\times$
(*'live1''*, *'live2''*, *'co1''*, *'co2''*, *'contra1''*, *'contra2''*, *'fixed*) *S* $\Rightarrow$
(*'live1* $\times$ *'live1''*, *'live2* $\times$ *'live2''*, *'co1'*, *'co2'*, *'contra1'*, *'contra2'*, *'fixed*) *S*
where
rel-S-witness' = (*id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$
id $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *id* $-{-}{-}{>}$ *map-prod Rep-S Rep-S* $-{-}{-}{>}$ *Abs-S*) *rel-G-witness*

lemma *rel-S-witness'1*:
fixes *L1* :: *'l1* $\Rightarrow$ *'l1''* $\Rightarrow$ *bool* **and** *L2* :: *'l2* $\Rightarrow$ *'l2''* $\Rightarrow$ *bool*
and *Co1* :: *'co1* $\Rightarrow$ *'co1'* $\Rightarrow$ *bool* **and** *Co1'* :: *'co1'* $\Rightarrow$ *'co1''* $\Rightarrow$ *bool*
and *Co2* :: *'co2* $\Rightarrow$ *'co2'* $\Rightarrow$ *bool* **and** *Co2'* :: *'co2'* $\Rightarrow$ *'co2''* $\Rightarrow$ *bool*
and *Contra1* :: *'contra1* $\Rightarrow$ *'contra1'* $\Rightarrow$ *bool* **and** *Contra1'* :: *'contra1'* $\Rightarrow$ *'contra1''* $\Rightarrow$ *bool*
and *Contra2* :: *'contra2* $\Rightarrow$ *'contra2'* $\Rightarrow$ *bool* **and** *Contra2'* :: *'contra2'* $\Rightarrow$ *'contra2''* $\Rightarrow$ *bool*
and *tytok* :: (*'l1* $\times$ (*'l1* $\times$ *'l1''*) $\times$ *'l1''* $\times$ *'l2* $\times$ (*'l2* $\times$ *'l2''*) $\times$ *'l2''* $\times$ *'f*) *itself*
and *x* :: (-, -, -, -, -, -, *'f*) *S*
assumes *rel-S L1 L2* (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) *x y*
and *rel-S-neg-distr-cond' Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*
shows *rel-S* (*λx* (*x'*, *y*). *x'* = *x* $\wedge$ *L1 x y*) (*λx* (*x'*, *y*). *x'* = *x* $\wedge$ *L2 x y*) *Co1 Co2 Contra1 Contra2 x*
(*rel-S-witness' L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (*x*, *y*))
using *assms* **unfolding** *rel-S-def rel-S-witness'-def*
apply(*simp add*: *vimage2p-def*)
apply(*subst Abs-S-inverse*)
apply(*simp*)
apply(*rule P-rel-G-zipping*[*OF Rep-S*[*of x*, *simplified*] *Rep-S*[*of y*, *simplified*], *rotated*])

apply(*erule rel-G-witness1*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
apply(*erule rel-G-witness2*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
apply(*assumption*)
apply(*assumption*)
apply(*erule rel-G-witness1*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
done

lemma *rel-S-witness'2*:
fixes *L1* :: *'l1* ⇒ *'l1''* ⇒ *bool* **and** *L2* :: *'l2* ⇒ *'l2''* ⇒ *bool*
and *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*
and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok* :: (*'l1* × (*'l1* × *'l1''*) × *'l1''* × *'l2* × (*'l2* × *'l2''*) × *'l2''* × *'f*) *itself*
and *x* :: (-, -, -, -, -, -, *'f*) *S*
assumes *rel-S L1 L2* (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) *x y*
and *rel-S-neg-distr-cond' Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok*
shows *rel-S* (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L1 x y*) (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L2 x y*) *Co1' Co2' Contra1' Contra2'*
(*rel-S-witness' L1 L2 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (*x*, *y*)) *y*
using *assms* **unfolding** *rel-S-def rel-S-witness'-def*
apply(*simp add*: *vimage2p-def*)
apply(*subst Abs-S-inverse*)
apply(*simp*)
apply(*rule P-rel-G-zipping*[*OF Rep-S*[*of x*, *simplified*] *Rep-S*[*of y*, *simplified*], *rotated*])
apply(*erule rel-G-witness1*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
apply(*erule rel-G-witness2*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
apply(*assumption*)
apply(*assumption*)
apply(*erule rel-G-witness2*[*rotated*])
apply(*erule rel-S-neg-distr-cond'-stronger*)
done

lemma *rel-S-neg-distr-imp*:
fixes *Co1* :: *'co1* ⇒ *'co1'* ⇒ *bool* **and** *Co1'* :: *'co1'* ⇒ *'co1''* ⇒ *bool*
and *Co2* :: *'co2* ⇒ *'co2'* ⇒ *bool* **and** *Co2'* :: *'co2'* ⇒ *'co2''* ⇒ *bool*
and *Contra1* :: *'contra1* ⇒ *'contra1'* ⇒ *bool* **and** *Contra1'* :: *'contra1'* ⇒ *'contra1''* ⇒ *bool*

and *Contra2* :: *'contra2* ⇒ *'contra2'* ⇒ *bool* **and** *Contra2'* :: *'contra2'* ⇒ *'contra2''* ⇒ *bool*
and *tytok-S'* :: (*'l1* × (*'l1* × *'l1''*) × *'l1''* × *'l2* × (*'l2* × *'l2''*) × *'l2''* × *'f*) *itself*
and *tytok-S* :: (*'l1* × *'l1'* × *'l1''* × *'l2* × *'l2'* × *'l2''* × *'f*) *itself*
assumes *rel-S-neg-distr-cond' Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-S'*
shows *rel-S-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok-S*
unfolding *rel-S-neg-distr-cond-def*
proof (*intro allI predicate2I relcomppI*)
fix *L1* :: *'l1* ⇒ *'l1'* ⇒ *bool* **and** *L1'* :: *'l1'* ⇒ *'l1''* ⇒ *bool*
and *L2* :: *'l2* ⇒ *'l2'* ⇒ *bool* **and** *L2'* :: *'l2'* ⇒ *'l2''* ⇒ *bool*
and *x* :: (-, -, -, -, -, -, *'f*) *S* **and** *y* :: (-, -, -, -, -, -, *'f*) *S*
assume ∗: *rel-S* (*L1 OO L1'*) (*L2 OO L2'*) (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*Contra2 OO Contra2'*) *x y*
let *?z* = *map-S* (*relcompp-witness L1 L1'*) (*relcompp-witness L2 L2'*) *id id id id* (*rel-S-witness'* (*L1 OO L1'*) (*L2 OO L2'*) *Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2'* (*x*, *y*))
show *rel-S L1 L2 Co1 Co2 Contra1 Contra2 x ?z*
apply(*subst map-S-id*[*symmetric*])
apply(*rule map-S-rel-cong*)
apply(*rule rel-S-witness'1*[*OF* ∗ *assms*])
apply(*auto simp add*: *vimage2p-def del*: *relcomppE elim*!: *relcompp-witness*)
done
show *rel-S L1' L2' Co1' Co2' Contra1' Contra2' ?z y*
apply(*rewrite* **in** *rel-S - - - - - - -* ⌑ *map-S-id*[*symmetric*])
apply(*rule map-S-rel-cong*)
apply(*rule rel-S-witness'2*[*OF* ∗ *assms*])
apply(*auto simp add*: *vimage2p-def del*: *relcomppE elim*!: *relcompp-witness*)
done
qed

end

5.3 Subtypes of BNFs without co- and contravariance

If all variables are live, **lift-bnf**'s requirement *P-zip-closed* is equivalent to our closedness under zippings, and Popescu's weaker condition is equivalent to negative subdistributivity restricted to the subset.

bnf-axiomatization *'a H*

consts *Q* :: *'a H* ⇒ *bool*

axiomatization where
Q-map: ⋀*x l*. *Q x* ⟹ *Q* (*map-H l x*)

lemma *Q-rel-H-zipping*:
fixes *x* :: *'a H* **and** *y* :: *'c H* **and** *z* :: (*'a* × *'c*) *H*

assumes *Q-zip*: $\bigwedge z :: ('a \times 'c)\ H.\ [\![\ Q\ (map\text{-}H\ fst\ z);\ Q\ (map\text{-}H\ snd\ z)\ ]\!] \Longrightarrow Q\ z$
and $Q\ x$ **and** $Q\ y$ **and** *rel-H* $L\ x\ y$
and *related*: *rel-H* $(\lambda x\ (x', y).\ x' = x \wedge L\ x\ y)\ x\ z$ *rel-H* $(\lambda(x, y')\ y.\ y' = y \wedge L\ x\ y)\ z\ y$
shows $Q\ z$
proof −
have *map-H fst* $z = x$ **proof** −
from *related*(*1*) **have** *rel-H* $(=)\ x\ (map\text{-}H\ fst\ z)$
by (*auto simp add*: *H.rel-map elim*: *H.rel-mono-strong*)
then show *?thesis* **by** (*simp add*: *H.rel-eq*)
qed
moreover have *map-H snd* $z = y$ **proof** −
from *related*(*2*) **have** *rel-H* $(=)\ (map\text{-}H\ snd\ z)\ y$
by (*auto simp add*: *H.rel-map elim*: *H.rel-mono-strong*)
then show *?thesis* **by** (*simp add*: *H.rel-eq*)
qed
moreover note ‹$Q\ x$› ‹$Q\ y$›
ultimately show *?thesis* **using** *Q-zip* **by** *blast*
qed

lemma *Q-zip*:
fixes $z :: ('a \times 'c)\ H$
assumes *Q-rel-H-zipping*: $\bigwedge(L :: 'a \Rightarrow 'c \Rightarrow -)\ x\ y\ z.$
$[\![\ Q\ x;\ Q\ y;\ rel\text{-}H\ L\ x\ y;\ rel\text{-}H\ (\lambda x\ (x', y).\ x' = x \wedge L\ x\ y)\ x\ z;$
$rel\text{-}H\ (\lambda(x, y')\ y.\ y' = y \wedge L\ x\ y)\ z\ y\ ]\!] \Longrightarrow Q\ z$
and $Q\ (map\text{-}H\ fst\ z)$ **and** $Q\ (map\text{-}H\ snd\ z)$
shows $Q\ z$
proof −
let *?L* $= \lambda a\ (a', b).\ a' = a \wedge top\ a\ b$ **and** *?L'* $= \lambda(b, c')\ c.\ c' = c \wedge top\ b\ c$
have $*$: *rel-H ?L* $(map\text{-}H\ fst\ z)\ z$ *rel-H ?L'* $z\ (map\text{-}H\ snd\ z)$
by (*auto simp add*: *H.rel-map Grp-apply intro*!: *H.rel-refl-strong*)
then have *rel-H* $(?L\ OO\ ?L')\ (map\text{-}H\ fst\ z)\ (map\text{-}H\ snd\ z)$
by (*auto simp add*: *H.rel-compp*)
then have *rel-H top* $(map\text{-}H\ fst\ z)\ (map\text{-}H\ snd\ z)$
by (*simp add*: *relcompp-apply*[*abs-def*] *top-fun-def*)
with ‹$Q\ (map\text{-}H\ fst\ z)$› ‹$Q\ (map\text{-}H\ snd\ z)$› $*$ **show** $Q\ z$
using *Q-rel-H-zipping* **by** *blast*
qed

lemma *Q-neg-distr*:
fixes $x :: 'a\ H$ **and** $y :: 'c\ H$
assumes *Q-zip-weak*: $\bigwedge z :: ('a \times 'c)\ H.\ [\![\ Q\ (map\text{-}H\ fst\ z);\ Q\ (map\text{-}H\ snd\ z)\ ]\!] \Longrightarrow$
$\exists z'.\ Q\ z' \wedge set\text{-}H\ z' \subseteq set\text{-}H\ z \wedge map\text{-}H\ fst\ z' = map\text{-}H\ fst\ z \wedge map\text{-}H\ snd\ z' = map\text{-}H\ snd\ z$
and $Q\ x$ **and** $Q\ y$ **and** *related*: *rel-H* $(L\ OO\ L')\ x\ y$
shows $(rel\text{-}H\ L\ OO\ eq\text{-}onp\ Q\ OO\ rel\text{-}H\ L')\ x\ y$
proof −

from *related* **obtain** z **where**
$*$: $set\text{-}H\ z \subseteq \{(x,\ y).\ (L\ OO\ L')\ x\ y\}$ $map\text{-}H\ fst\ z = x$ $map\text{-}H\ snd\ z = y$
unfolding $H.rel\text{-}compp\text{-}Grp$ **by** (*blast elim*: *GrpE*)
with ‹$Q\ x$› ‹$Q\ y$› **have** $Q\ (map\text{-}H\ fst\ z)$ **and** $Q\ (map\text{-}H\ snd\ z)$ **by** *simp-all*
then obtain z' **where** $Q\ z'$ $set\text{-}H\ z' \subseteq set\text{-}H\ z$
$map\text{-}H\ fst\ z' = map\text{-}H\ fst\ z$ $map\text{-}H\ snd\ z' = map\text{-}H\ snd\ z$
using *Q-zip-weak* **by** *blast*
with $*$ **have** $**$: $set\text{-}H\ z' \subseteq \{(x,\ y).\ (L\ OO\ L')\ x\ y\}$ $x = map\text{-}H\ fst\ z'$ $y = map\text{-}H\ snd\ z'$
by *simp-all*
let $?z = map\text{-}H\ (relcompp\text{-}witness\ L\ L')\ z'$
from ‹$Q\ z'$› **have** $Q\ ?z$ **by** (*rule Q-map*)
moreover have $rel\text{-}H\ L\ x\ ?z$ $rel\text{-}H\ L'\ ?z\ y$
using $**$ **by** (*auto simp add*: $H.rel\text{-}map$ *intro*!: $H.rel\text{-}refl\text{-}strong$ $relcompp\text{-}witness[of\ L\ L'\ fst\ p$ $snd\ p$ **for** p, *simplified*])
ultimately show *?thesis* **unfolding** *eq-onp-def* **by** *blast*
qed

lemma *Q-zip-weak*:
fixes $z :: ('a \times 'c)\ H$
assumes *Q-neg-distr*: $\bigwedge(L :: 'a \Rightarrow ('a \times 'c) \Rightarrow \text{-})\ (L' :: ('a \times 'c) \Rightarrow 'c \Rightarrow bool)\ x\ y.$
$[\![\ Q\ x;\ Q\ y;\ rel\text{-}H\ (L\ OO\ L')\ x\ y\ ]\!] \Longrightarrow (rel\text{-}H\ L\ OO\ eq\text{-}onp\ Q\ OO\ rel\text{-}H\ L')\ x\ y$
and $Q\ (map\text{-}H\ fst\ z)$ **and** $Q\ (map\text{-}H\ snd\ z)$
obtains z' **where** $Q\ z'$ **and** $set\text{-}H\ z' \subseteq set\text{-}H\ z$
and $map\text{-}H\ fst\ z' = map\text{-}H\ fst\ z$ **and** $map\text{-}H\ snd\ z' = map\text{-}H\ snd\ z$
proof −
let $?L = (Grp\ (set\text{-}H\ z)\ fst)^{-1-1}$ **and** $?L' = Grp\ (set\text{-}H\ z)\ snd$
have $rel\text{-}H\ ?L\ (map\text{-}H\ fst\ z)\ z$ $rel\text{-}H\ ?L'\ z\ (map\text{-}H\ snd\ z)$
by (*auto simp add*: $H.rel\text{-}map$ *Grp-apply intro*!: $H.rel\text{-}refl\text{-}strong$)
then have $rel\text{-}H\ (?L\ OO\ ?L')\ (map\text{-}H\ fst\ z)\ (map\text{-}H\ snd\ z)$
by (*auto simp add*: $H.rel\text{-}compp$)
with ‹$Q\ (map\text{-}H\ fst\ z)$› ‹$Q\ (map\text{-}H\ snd\ z)$›
have $(rel\text{-}H\ ?L\ OO\ eq\text{-}onp\ Q\ OO\ rel\text{-}H\ ?L')\ (map\text{-}H\ fst\ z)\ (map\text{-}H\ snd\ z)$
by (*rule Q-neg-distr*)
then obtain z' **where** $Q\ z'$ $rel\text{-}H\ ?L\ (map\text{-}H\ fst\ z)\ z'$ $rel\text{-}H\ ?L'\ z'\ (map\text{-}H\ snd\ z)$
unfolding *eq-onp-def* **by** *blast*
then have $rel\text{-}H\ (\lambda a\ b.\ snd\ b = snd\ a \wedge a \in set\text{-}H\ z)\ z'\ z$
by (*simp add*: $H.rel\text{-}map\ Grp\text{-}apply$)
then have $rel\text{-}H\ (\lambda a\ b.\ a \in set\text{-}H\ z)\ z'\ z$
by (*auto elim*: $H.rel\text{-}mono\text{-}strong$)
then have $pred\text{-}H\ (Domainp\ (\lambda a\ (b :: ('a \times 'c)).\ a \in set\text{-}H\ z))\ z'$
by (*auto simp add*: $H.Domainp\text{-}rel[symmetric]\ Domainp\text{-}iff$)
then have $set\text{-}H\ z' \subseteq set\text{-}H\ z$
unfolding $H.axiom10\text{-}H$ **by** *auto*
moreover have $map\text{-}H\ fst\ z' = map\text{-}H\ fst\ z$
apply (*rule sym*)

apply (*subst H.rel-eq*[*symmetric*])
apply (*subst H.rel-map*(*2*))
apply (*rule H.rel-mono-strong*)
apply (*fact* ‹*rel-H ?L* (*map-H fst z*) *z'*›)
apply (*simp add*: *Grp-apply*)
done
moreover have *map-H snd z'* = *map-H snd z*
apply (*subst H.rel-eq*[*symmetric*])
apply (*subst H.rel-map*(*1*))
apply (*rule H.rel-mono-strong*)
apply (*fact* ‹*rel-H ?L' z'* (*map-H snd z*)›)
apply (*simp add*: *Grp-apply*)
done
moreover note ‹*Q z'*›
ultimately show *thesis* **using** *that* **by** *blast*
qed

end

6 Quotient preservation

theory *Quotient-Preservation* **imports**
Axiomatised-BNF-CC
begin

lemma *G-Quotient*:
fixes *T-l1* :: *'l1* ⇒ *'l1'* ⇒ *bool* **and** *T-l2* :: *'l2* ⇒ *'l2'* ⇒ *bool*
and *tytok* :: (*'l1* × *'l1'* × *'l1* × *'l2* × *'l2'* × *'l2* × *'f*) *itself*
assumes *Quotient R-l1 Abs-l1 Rep-l1 T-l1* **and** *Quotient R-l2 Abs-l2 Rep-l2 T-l2*
and *Quotient R-co1 Abs-co1 Rep-co1 T-co1* **and** *Quotient R-co2 Abs-co2 Rep-co2 T-co2*
and *Quotient R-contra1 Abs-contra1 Rep-contra1 T-contra1*
and *Quotient R-contra2 Abs-contra2 Rep-contra2 T-contra2*
and *rel-G-pos-distr-cond T-co1 T-co1*$^{-1-1}$ *T-co2 T-co2*$^{-1-1}$ *T-contra1 T-contra1*$^{-1-1}$ *T-contra2 T-contra2*$^{-1-1}$
tytok
shows *Quotient* (*rel-G R-l1 R-l2 R-co1 R-co2 R-contra1 R-contra2*)
(*map-G Abs-l1 Abs-l2 Abs-co1 Abs-co2 Rep-contra1 Rep-contra2*)
(*map-G Rep-l1 Rep-l2 Rep-co1 Rep-co2 Abs-contra1 Abs-contra2*)
(*rel-G T-l1 T-l2 T-co1 T-co2 T-contra1 T-contra2* :: (-, -, -, -, -, -, *'f*) *G* ⇒ -)
unfolding *Quotient-alt-def5*
proof (*intro conjI*, *goal-cases*)
case *1*
have *rel-G T-l1 T-l2 T-co1 T-co2 T-contra1 T-contra2* ≤
rel-G (*Grp UNIV Abs-l1*) (*Grp UNIV Abs-l2*) (*Grp UNIV Abs-co1*) (*Grp UNIV Abs-co2*)
(*Grp UNIV Rep-contra1*)$^{-1-1}$ (*Grp UNIV Rep-contra2*)$^{-1-1}$
apply (*rule rel-G-mono*)
apply (*rule assms*(*1*−*4*)[*unfolded Quotient-alt-def5*, *THEN conjunct1*])+

apply (*rule assms*(5−6)[*unfolded Quotient-alt-def5*, *THEN conjunct2*, *THEN conjunct1*,
unfolded conversep-le-swap])+
done
also have ... = *Grp UNIV* (*map-G Abs-l1 Abs-l2 Abs-co1 Abs-co2 Rep-contra1 Rep-contra2*)
using *rel-G-Grp-weak* .
finally show *?case* .
next
case *2*
have *Grp UNIV* (*map-G Rep-l1 Rep-l2 Rep-co1 Rep-co2 Abs-contra1 Abs-contra2*) =
rel-G (*Grp UNIV Rep-l1*) (*Grp UNIV Rep-l2*) (*Grp UNIV Rep-co1*) (*Grp UNIV Rep-co2*)
(*Grp UNIV Abs-contra1*)$^{-1-1}$ (*Grp UNIV Abs-contra2*)$^{-1-1}$
using *rel-G-Grp-weak*[*symmetric*] .
also have ... ≤ *rel-G T-l1*$^{-1-1}$ *T-l2*$^{-1-1}$ *T-co1*$^{-1-1}$ *T-co2*$^{-1-1}$ *T-contra1*$^{-1-1}$ *T-contra2*$^{-1-1}$
apply (*rule rel-G-mono*)
apply (*rule assms*(1−4)[*unfolded Quotient-alt-def5*, *THEN conjunct2*, *THEN conjunct1*])+
apply (*simp-all*)
apply (*rule assms*(5−6)[*unfolded Quotient-alt-def5*, *THEN conjunct1*])+
done
finally show *?case* **by** (*simp add*: *rel-G-conversep*)
next
case *3*
show *?case*
apply (*rule antisym*)
apply (*rule predicate2I*)
apply (*rule relcomppI*)
apply (*subst map-G-id*[*symmetric*])
apply (*erule map-G-rel-cong*)
apply (*simp-all*)[*6*]
apply (*erule assms*(1−4)[*THEN Quotient-equiv-abs1*])+
apply (*erule assms*(5−6)[*THEN Quotient-rep-equiv1*])+
apply (*subst rel-G-conversep*[*symmetric*])
subgoal for *x y*
apply (*subgoal-tac map-G Abs-l1 Abs-l2 Abs-co1 Abs-co2 Rep-contra1 Rep-contra2 y* =
map-G Abs-l1 Abs-l2 Abs-co1 Abs-co2 Rep-contra1 Rep-contra2 x)
apply (*simp*)
apply (*subst* (*3*) *map-G-id*[*symmetric*])
apply (*erule map-G-rel-cong*)
apply (*simp-all*)
apply (*erule assms*(1−4)[*THEN Quotient-equiv-abs2*])+
apply (*erule assms*(5−6)[*THEN Quotient-rep-equiv2*])+
apply (*rule sym*)
apply (*subst rel-G-eq*[*symmetric*])

apply (*erule map-G-rel-cong*)
apply (*erule assms*(1−4)[*THEN Quotient-rel-abs*])+
apply (*simp*, *rule assms*(5−6)[*THEN Quotient-rep-reflp*])+
done
apply (*unfold rel-G-conversep*[*symmetric*]
assms(1−6)[*unfolded Quotient-alt-def5*, *THEN conjunct2*, *THEN conjunct2*])
apply (*rule rel-G-pos-distr*)
apply (*rule assms*(7))
done
qed

end

theory *Operation-Examples* **imports**
Composition
Fixpoints
Subtypes
Quotient-Preservation
begin

end

7 Concrete $\mathbf{BNF_{CC}s}$

theory *Concrete-Examples* **imports**
Preliminaries
HOL−Library.Rewrite
HOL−Library.Cardinality
begin

context includes *lifting-syntax*
begin

7.1 Function space

lemma *rel-fun-mono*: $(A ===> B) \leq (A' ===> B')$ **if** $A' \leq A \quad B \leq B'$
using *that* **by**(*auto simp add*: *rel-fun-def*)

lemma *rel-fun-eq*: $((=) ===> (=)) = (=)$ **by**(*fact fun.rel-eq*)

lemma *rel-fun-conversep*: $(A^{-1-1} ===> B^{-1-1}) = (A ===> B)^{-1-1}$ **by**(*auto simp add*: *rel-fun-def*)

lemma *map-fun-id0*: $(id ---> id) = id$ **by**(*fact map-fun.id*)

lemma *map-fun-comp*: $(f ---> g) \circ (f' ---> g') = ((f' \circ f) ---> (g \circ g'))$ **by**(*fact map-fun.comp*)

lemma *map-fun-parametric*: $((A \mathrel{===>} A') \mathrel{===>} (B \mathrel{===>} B') \mathrel{===>} (A' \mathrel{===>} B) \mathrel{===>} (A \mathrel{===>} B'))\ (-\!\!-\!\!->)\ (-\!\!-\!\!->)$
by(*fact map-fun-parametric*)

definition *rel-fun-pos-distr-cond* :: $('a \Rightarrow 'a' \Rightarrow bool) \Rightarrow ('a' \Rightarrow 'a'' \Rightarrow bool) \Rightarrow ('b \times 'b' \times 'b'')\ itself \Rightarrow bool$ **where**
rel-fun-pos-distr-cond A A' - $\longleftrightarrow$ $(\forall (B :: 'b \Rightarrow 'b' \Rightarrow bool)\ (B' :: 'b' \Rightarrow 'b'' \Rightarrow bool).$
$(A \mathrel{===>} B)\ OO\ (A' \mathrel{===>} B') \leq (A\ OO\ A') \mathrel{===>} (B\ OO\ B'))$

definition *rel-fun-neg-distr-cond* :: $('a \Rightarrow 'a' \Rightarrow bool) \Rightarrow ('a' \Rightarrow 'a'' \Rightarrow bool) \Rightarrow ('b \times 'b' \times 'b'')\ itself \Rightarrow bool$ **where**
rel-fun-neg-distr-cond A A' - $\longleftrightarrow$ $(\forall (B :: 'b \Rightarrow 'b' \Rightarrow bool)\ (B' :: 'b' \Rightarrow 'b'' \Rightarrow bool).$
$(A\ OO\ A') \mathrel{===>} (B\ OO\ B') \leq (A \mathrel{===>} B)\ OO\ (A' \mathrel{===>} B'))$

lemmas
rel-fun-pos-distr = *rel-fun-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-fun-neg-distr = *rel-fun-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

lemma *rel-fun-pos-distr-iff* [*simp*]: *rel-fun-pos-distr-cond A A' tytok* = *True*
unfolding *rel-fun-pos-distr-cond-def* **by** (*blast intro*!: *pos-fun-distr*)

lemma *rel-fun-neg-distr-imp*: ⟦ *left-unique A*; *right-total A*; *right-unique A'*; *left-total A'* ⟧ ⟹
rel-fun-neg-distr-cond A A' tytok
unfolding *rel-fun-neg-distr-cond-def* **by** (*fast elim*!: *neg-fun-distr1*[*THEN predicate2D*])

lemma *rel-fun-pos-distr-cond-eq*: *rel-fun-pos-distr-cond* (=) (=) *tytok*
by *simp*

lemma *rel-fun-neg-distr-cond-eq*: *rel-fun-neg-distr-cond* (=) (=) *tytok*
by (*blast intro*: *rel-fun-neg-distr-imp left-unique-eq right-unique-eq right-total-eq left-total-eq*)

thm *fun.set-map fun.map-cong0 fun.rel-mono-strong*

7.2 Covariant powerset

lemma *rel-set-mono*: $A \leq A' \Longrightarrow$ *rel-set A* $\leq$ *rel-set A'* **by**(*fact rel-set-mono*)

lemma *rel-set-eq*: *rel-set* (=) = (=) **by**(*fact rel-set-eq*)

lemma *rel-set-conversep*: *rel-set* A^{-1-1} = (*rel-set A*)$^{-1-1}$ **by**(*fact rel-set-conversep*)

lemma *map-set-id0*: *image id* = *id* **by**(*fact image-id*)

lemma *map-set-comp*: *image f* ∘ *image g* = *image* (*f* ∘ *g*) **by**(*simp add*: *fun-eq-iff*

image-image o-def)

lemma *map-set-parametric*: **includes** *lifting-syntax* **shows**
((A ===> B) ===> *rel-set* A ===> *rel-set* B) *image image*
by(*fact image-transfer*)

definition *rel-set-pos-distr-cond* :: ('a ⇒ 'a' ⇒ *bool*) ⇒ ('a' ⇒ 'a'' ⇒ *bool*) ⇒ *bool* **where**
rel-set-pos-distr-cond A $A' \longleftrightarrow$ *rel-set* A *OO rel-set* $A' \leq$ *rel-set* (A *OO* A')

definition *rel-set-neg-distr-cond* :: ('a ⇒ 'a' ⇒ *bool*) ⇒ ('a' ⇒ 'a'' ⇒ *bool*) ⇒ *bool* **where**
rel-set-neg-distr-cond A $A' \longleftrightarrow$ *rel-set* (A *OO* A') $\leq$ *rel-set* A *OO rel-set* A'

lemmas
rel-set-pos-distr = *rel-set-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-set-neg-distr = *rel-set-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

lemma *rel-set-pos-distr-iff* [*simp*]: *rel-set-pos-distr-cond* A A' = *True*
unfolding *rel-set-pos-distr-cond-def* **by**(*simp add*: *rel-set-OO*)

lemma *rel-set-neg-distr-iff* [*simp*]: *rel-set-neg-distr-cond* A A' = *True*
unfolding *rel-set-neg-distr-cond-def* **by**(*simp add*: *rel-set-OO*)

lemma *rel-set-pos-distr-eq*: *rel-set-pos-distr-cond* (=) (=)
by *simp*

lemma *rel-set-neg-distr-eq*: *rel-set-neg-distr-cond* (=) (=)
by *simp*

7.3 Bounded sets

We define bounded sets as a subtype, with an additional fixed parameter which controls the bound. Using the $\mathrm{BNF}_{\mathrm{CC}}$ structure on the covariant powerset functor, it suffices to show the preconditions for the closedness of $\mathrm{BNF}_{\mathrm{CC}}$ under subtypes.

typedef ('a, 'k) *bset* = {A :: 'a *set*. *finite* A ∧ *card* $A \leq$ *CARD*('k)}
proof
show {} ∈ *?bset* **by** *simp*
qed

setup-lifting *type-definition-bset*

lemma *bset-map-closed*:
fixes f A
defines $B \equiv$ *image* f A
assumes *finite* A ∧ *card* $A \leq$ *CARD*('k)
shows *finite* B ∧ *card* $B \leq$ *CARD*('k)

using *assms* **by**(*auto intro*: *card-image-le*[*THEN order-trans*])

lift-definition *map-bset* :: ('*a* ⇒ '*b*) ⇒ ('*a*, '*k*) *bset* ⇒ ('*b*, '*k*) *bset* **is** *image*
by(*fact bset-map-closed*)

lift-definition *rel-bset* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*a*, '*k*) *bset* ⇒ ('*b*, '*k*) *bset* ⇒ *bool*
is *rel-set* **.**

definition *neg-distr-cond-bset* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*b* ⇒ '*c* ⇒ *bool*) ⇒ '*k itself*
⇒ *bool* **where**
neg-distr-cond-bset C C' - ⟷ *rel-bset* (*C OO C'*) ≤ *rel-bset C OO* (*rel-bset C'*
:: (-, '*k*) *bset* ⇒ -)

lemma *right-unique-rel-set-lemma*:
assumes *right-unique R* **and** *rel-set R X Y*
obtains *f* **where** *Y* = *image f X* **and** ∀ *x*∈*X*. *R x* (*f x*)
proof
define *f* **where** *f x* = (*THE y*. *R x y*) **for** *x*
{ fix *x* **assume** *x* ∈ *X*
with ‹*rel-set R X Y*› ‹*right-unique R*› **have** *R x* (*f x*)
by (*simp add*: *right-unique-def rel-set-def f-def*) (*metis theI*)
with *assms* ‹*x* ∈ *X*› **have** *R x* (*f x*) *f x* ∈ *Y*
by (*fastforce simp add*: *right-unique-def rel-set-def*)+ **}**
moreover
have ∃ *x*∈*X*. *y* = *f x* **if** *y* ∈ *Y* **for** *y* **using** ‹*rel-set R X Y*› *that*
by(*auto simp add*: *f-def dest*!: *rel-setD2 dest*: *right-uniqueD*[*OF* ‹*right-unique R*›]
intro: *the-equality*[*symmetric*])
ultimately show ∀ *x*∈*X*. *R x* (*f x*) *Y* = *image f X*
by (*auto simp*: *inj-on-def image-iff*)
qed

lemma *left-unique-rel-set-lemma*:
assumes *left-unique R* **and** *rel-set R Y X*
obtains *f* **where** *Y* = *image f X* **and** ∀ *x*∈*X*. *R* (*f x*) *x*
proof
define *f* **where** *f x* = (*THE y*. *R y x*) **for** *x*
{ fix *x* **assume** *x* ∈ *X*
with ‹*rel-set R Y X*› ‹*left-unique R*› **have** *R* (*f x*) *x*
by (*simp add*: *left-unique-def rel-set-def f-def*) (*metis theI*)
with *assms* ‹*x* ∈ *X*› **have** *R* (*f x*) *x* *f x* ∈ *Y*
by (*fastforce simp add*: *left-unique-def rel-set-def*)+ **}**
moreover
have ∃ *x*∈*X*. *y* = *f x* **if** *y* ∈ *Y* **for** *y* **using** ‹*rel-set R Y X*› *that*
by(*auto simp add*: *f-def dest*!: *rel-setD1 dest*: *left-uniqueD*[*OF* ‹*left-unique R*›]
intro: *the-equality*[*symmetric*])
ultimately show ∀ *x*∈*X*. *R* (*f x*) *x* *Y* = *image f X*
by (*auto simp*: *inj-on-def image-iff*)
qed

lemma *neg-distr-cond-bset-right-unique*:
right-unique C $\Longrightarrow$ *neg-distr-cond-bset C D tytok*
unfolding *neg-distr-cond-bset-def*
apply(*rule predicate2I*)
apply *transfer*
apply(*auto 6 2 intro*: *card-image-le*[*THEN order-trans*] *elim*: *right-unique-rel-set-lemma*
simp add: *rel-set-OO*[*symmetric*])
done

lemma *neg-distr-cond-bset-left-unique*:
left-unique D $\Longrightarrow$ *neg-distr-cond-bset C D tytok*
unfolding *neg-distr-cond-bset-def*
apply(*rule predicate2I*)
apply *transfer*
apply(*auto 6 2 intro*: *card-image-le*[*THEN order-trans*] *elim*: *left-unique-rel-set-lemma*
simp add: *rel-set-OO*[*symmetric*])
done

lemma *neg-distr-cond-bset-eq*: *neg-distr-cond-bset* (=) (=) *tytok*
by (*intro neg-distr-cond-bset-right-unique right-unique-eq*)

7.4 Contravariant powerset (sets as predicates)

type-synonym *'a pred* = *'a* $\Rightarrow$ *bool*

definition *map-pred* :: (*'b* $\Rightarrow$ *'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'b pred* **where**
map-pred f = (*f* $---{>}$ *id*)

definition *rel-pred* :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'b pred* $\Rightarrow$ *bool* **where**
rel-pred R = (*R* $==={>}$ ($\longleftrightarrow$))

lemma *rel-pred-mono*: *A'* $\leq$ *A* $\Longrightarrow$ *rel-pred A* $\leq$ *rel-pred A'* **unfolding** *rel-pred-def*
by(*auto elim*!: *rel-fun-mono*)

lemma *rel-pred-eq*: *rel-pred* (=) = (=)
by(*simp add*: *rel-pred-def rel-fun-eq*)

lemma *rel-pred-conversep*: *rel-pred* A^{-1-1} = (*rel-pred A*)$^{-1-1}$
using *rel-fun-conversep*[*of* - (=)] **by** (*simp add*: *rel-pred-def*)

lemma *map-pred-id0*: *map-pred id* = *id*
by (*simp add*: *map-pred-def map-fun-id*)

lemma *map-pred-comp*: *map-pred f* $\circ$ *map-pred g* = *map-pred* (*g* $\circ$ *f*)
using *map-fun-comp*[**where** *g*=*id* **and** *g'*=*id*] **by** (*simp add*: *map-pred-def*)

lemma *map-pred-parametric*: ((*A'* $==={>}$ *A*) $==={>}$ *rel-pred A* $==={>}$ *rel-pred A'*) *map-pred map-pred*

by (*simp add*: *rel-fun-def rel-pred-def map-pred-def*)

definition *rel-pred-pos-distr-cond* :: ('*a* ⇒ '*a'* ⇒ *bool*) ⇒ ('*a'* ⇒ '*a''* ⇒ *bool*) ⇒ *bool* **where**
rel-pred-pos-distr-cond A B ⟷ *rel-pred A OO rel-pred B* ≤ *rel-pred* (*A OO B*)

definition *rel-pred-neg-distr-cond* :: ('*a* ⇒ '*a'* ⇒ *bool*) ⇒ ('*a'* ⇒ '*a''* ⇒ *bool*) ⇒ *bool* **where**
rel-pred-neg-distr-cond A B ⟷ *rel-pred* (*A OO B*) ≤ *rel-pred A OO rel-pred B*

lemmas
rel-pred-pos-distr = *rel-pred-pos-distr-cond-def*[*THEN iffD1*, *rule-format*] **and**
rel-pred-neg-distr = *rel-pred-neg-distr-cond-def*[*THEN iffD1*, *rule-format*]

lemma *rel-pred-pos-distr-iff* [*simp*]: *rel-pred-pos-distr-cond A B* = *True*
unfolding *rel-pred-pos-distr-cond-def* **by** (*auto simp add*: *rel-pred-def rel-fun-def*)

lemma *rel-pred-pos-distr-cond-eq*: *rel-pred-pos-distr-cond* (=) (=)
by *simp*

lemma *neg-fun-distr3*:
assumes *1*: *left-unique R* *right-total R*
and *2*: *right-unique S* *left-total S*
shows *rel-fun* (*R OO R'*) (*S OO S'*) ≤ *rel-fun R S OO rel-fun R' S'*
using *functional-converse-relation*[*OF 1*] *functional-relation*[*OF 2*]
unfolding *rel-fun-def OO-def*
apply *clarify*
apply (*subst all-comm*)
apply (*subst all-conj-distrib*[*symmetric*])
apply (*intro choice*)
by *metis*

As there are no live variables, we can get a weaker condition than if we derived it from (===>)'s condition!

lemma *rel-pred-neg-distr-imp*:
right-unique B ∧ *left-total B* ∨ *left-unique A* ∧ *right-total A* ⟹ *rel-pred-neg-distr-cond A B*
unfolding *rel-pred-neg-distr-cond-def rel-pred-def*
apply(*clarsimp simp add*: *vimage2p-def rel-pred-neg-distr-cond-def*)
apply(*rewrite* **in** *rel-fun* - ⌑ **in** *asm eq-OO*[*symmetric*])
apply(*elim disjE*)
apply(*drule neg-fun-distr2*[*THEN predicate2D*, *rotated* −*1*];
(*simp add*: *left-unique-eq right-unique-eq left-total-eq right-total-eq*)*?*)
apply(*drule neg-fun-distr3*[*THEN predicate2D*, *rotated* −*1*];
(*simp add*: *left-unique-eq right-unique-eq left-total-eq right-total-eq*)*?*)
done

lemma *rel-pred-neg-distr-cond-eq*: *rel-pred-neg-distr-cond* (=) (=)
by(*blast intro*: *rel-pred-neg-distr-imp left-unique-eq right-total-eq*)

lemma *left-unique-rel-pred*: *left-total A* $\Longrightarrow$ *left-unique* (*rel-pred A*)
unfolding *rel-pred-def* **by** (*erule left-unique-fun*) (*rule left-unique-eq*)

lemma *right-unique-rel-pred*: *right-total A* $\Longrightarrow$ *right-unique* (*rel-pred A*)
unfolding *rel-pred-def* **by** (*erule right-unique-fun*) (*rule right-unique-eq*)

lemma *left-total-rel-pred*: *left-unique A* $\Longrightarrow$ *left-total* (*rel-pred A*)
unfolding *rel-pred-def* **by** (*erule left-total-fun*) (*rule left-total-eq*)

lemma *right-total-rel-pred*: *right-unique A* $\Longrightarrow$ *right-total* (*rel-pred A*)
unfolding *rel-pred-def* **by** (*erule right-total-fun*) (*rule right-total-eq*)

end

7.5 Filter

Similarly to bounded sets, we exploit the definition of filters as a subtype in order to lift the $\mathrm{BNF}_{\mathrm{CC}}$ operations. Here we use that the *is-filter* predicate is closed under zippings.

lemma *map-filter-closed*:
includes *lifting-syntax*
assumes *is-filter F*
shows *is-filter* (((*f* −−−> *id*) −−−> *id*) *F*)
proof −
define *F′* **where** *F′* = *Abs-filter F*
have *is-filter* (((*f* −−−> *id*) −−−> *id*) (λ*P*. *eventually P F′*))
by (*rule is-filter.intro*)(*auto elim*!: *eventually-rev-mp simp add*: *map-fun-def o-def*)
then show *?thesis* **using** *assms* **by**(*simp add*: *F′-def eventually-Abs-filter*)
qed

definition *rel-pred2-neg-distr-cond* :: ('*a* ⇒ '*a′* ⇒ *bool*) ⇒ ('*a′* ⇒ '*a″* ⇒ *bool*) ⇒ *bool* **where**
rel-pred2-neg-distr-cond A B ⟷
rel-pred (*rel-pred* (*A OO B*)) ≤ *rel-pred* (*rel-pred A*) *OO rel-pred* (*rel-pred B*)

consts *rel-pred2-witness* :: ('*a* ⇒ '*a′* ⇒ *bool*) ⇒ ('*a′* ⇒ '*a″* ⇒ *bool*) ⇒
(('*a* ⇒ *bool*) ⇒ *bool*) × (('*a″* ⇒ *bool*) ⇒ *bool*) ⇒ ('*a′* ⇒ *bool*) ⇒ *bool*

specification (*rel-pred2-witness*)
rel-pred2-witness1: ⋀*K K′ x y*. ⟦ *rel-pred2-neg-distr-cond K K′*; *rel-pred* (*rel-pred* (*K OO K′*)) *x y* ⟧ ⟹
rel-pred (*rel-pred K*) *x* (*rel-pred2-witness K K′* (*x*, *y*))
rel-pred2-witness2: ⋀*K K′ x y*. ⟦ *rel-pred2-neg-distr-cond K K′*; *rel-pred* (*rel-pred* (*K OO K′*)) *x y* ⟧ ⟹
rel-pred (*rel-pred K′*) (*rel-pred2-witness K K′* (*x*, *y*)) *y*

apply (*rule exI*[*of* - λ*K K'* (*x*, *y*). *SOME z*. *rel-pred* (*rel-pred K*) *x z* ∧ *rel-pred* (*rel-pred K'*) *z y*])
apply (*fold all-conj-distrib*)
apply (*intro allI*)
apply (*fold imp-conjR*)
apply (*clarify*)
apply (*rule relcomppE*[*of rel-pred* (*rel-pred* -) *rel-pred* (*rel-pred* -), *rotated*])
apply (*rule someI*[**where** *P*=λ*z*. *rel-pred* (*rel-pred* -) - *z* ∧ *rel-pred* (*rel-pred* -) *z* -])
apply (*erule* (*1*) *conjI*)
apply (*auto simp add*: *rel-pred2-neg-distr-cond-def*)
done

lemmas *rel-pred2-witness* = *rel-pred2-witness1 rel-pred2-witness2*

context includes *lifting-syntax*
begin

definition *rel-filter-neg-distr-cond'* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*b* ⇒ '*c* ⇒ *bool*) ⇒ *bool* **where**
rel-filter-neg-distr-cond' C C' ⟷ *left-total C* ∧ *right-unique C* ∨ *right-total C'* ∧ *left-unique C'*

lemma *rel-filter-neg-distr-cond'-stronger*:
assumes *rel-filter-neg-distr-cond' C C'*
shows *rel-pred2-neg-distr-cond C C'*
unfolding *rel-pred2-neg-distr-cond-def*
proof −
have *rel-pred* (*rel-pred* (*C OO C'*)) ≤ *rel-pred* (*rel-pred C OO rel-pred C'*)
by (*auto intro*!: *rel-pred-mono rel-pred-pos-distr*)
also have ... ≤ *rel-pred* (*rel-pred C*) *OO rel-pred* (*rel-pred C'*)
apply (*rule rel-pred-neg-distr*)
apply (*rule rel-pred-neg-distr-imp*)
apply (*insert assms*[*unfolded rel-filter-neg-distr-cond'-def*])
by (*blast dest*: *left-unique-rel-pred right-total-rel-pred right-unique-rel-pred left-total-rel-pred*)
finally show *rel-pred* (*rel-pred* (*C OO C'*)) ≤ ... **.**
qed

lemma *rel-filter-neg-distr-cond'-eq*: *rel-filter-neg-distr-cond'* (=) (=)
unfolding *rel-filter-neg-distr-cond'-def* **by** (*simp add*: *left-total-eq right-unique-eq*)

lemma *is-filter-rel-witness*:
assumes *F*: *is-filter F* **and** *G*: *is-filter G*
and *FG*: *rel-pred* (*rel-pred* (*C OO C'*)) *F G*
and *cond*: *rel-filter-neg-distr-cond' C C'*
shows *is-filter* (*rel-pred2-witness C C'* (*F*, *G*))
proof
let *?C* = *rel-pred* (*rel-pred C*) **and** *?C'* = *rel-pred* (*rel-pred C'*)
let *?wit* = *rel-pred2-witness C C'* (*F*, *G*)

```
  have rel-pred2-neg-distr-cond C C'
    using cond by (rule rel-filter-neg-distr-cond'-stronger)
  with FG have wit1: ?C F ?wit and wit2: ?C' ?wit G
    by (rule rel-pred2-witness[rotated])+
  from wit1[unfolded rel-pred-def, THEN rel-funD, of λ-. True    λ-. True] F
  show ?wit (λ-. True) by (auto simp add: is-filter.True)

  fix P Q
  have ∗: (?wit P ⟶ ?wit Q ⟶ ?wit (λx. P x ∧ Q x)) ∧ (?wit P ⟶ (∀x. P x
⟶ Q x) ⟶ ?wit Q)
    using cond unfolding rel-filter-neg-distr-cond'-def
  proof(elim disjE conjE; use nothing in ‹intro conjI strip›)
    assume left-total C    right-unique C
    hence left-unique (C ===> (=))    right-total (C ===> (=))
      by(blast intro: left-unique-fun left-unique-eq right-total-fun right-total-eq)+
    from functional-converse-relation[OF this] obtain P' Q'
      where P' [transfer-rule]: (C ===> (=)) P' P and Q' [transfer-rule]: (C
===> (=)) Q' Q
      by fastforce
   have PQ: (C ===> (=)) (λx. P' x ∧ Q' x) (λx. P x ∧ Q x) by transfer-prover

    with wit1 P' Q' have P: ?wit P ⟷ F P' and Q: ?wit Q ⟷ F Q'
      and PQ: ?wit (λx. P x ∧ Q x) ⟷ F (λx. P' x ∧ Q' x)
      by(auto dest: rel-funD simp add: rel-pred-def)

    show ?wit (λx. P x ∧ Q x) if ?wit P    ?wit Q using that P Q PQ
      by(auto intro: is-filter.conj[OF F])

    assume ∀x. P x ⟶ Q x
      with P' Q' ‹left-total C› have ∀x. P' x ⟶ Q' x by(metis (full-types)
apply-rsp' left-total-def)
    then show ?wit Q if ?wit P using P Q that by(simp add: is-filter.mono[OF
F])
  next
    assume right-total C'    left-unique C'
    hence right-unique (C' ===> (=))    left-total (C' ===> (=))
      by(blast intro: right-unique-fun right-unique-eq left-total-fun left-total-eq)+
    from functional-relation[OF this] obtain P' Q'
      where P' [transfer-rule]: (C' ===> (=)) P P' and Q' [transfer-rule]: (C'
===> (=)) Q Q'
      by fastforce
   have PQ: (C' ===> (=)) (λx. P x ∧ Q x) (λx. P' x ∧ Q' x) by transfer-prover

    with wit2 P' Q' have P: ?wit P ⟷ G P' and Q: ?wit Q ⟷ G Q'
      and PQ: ?wit (λx. P x ∧ Q x) ⟷ G (λx. P' x ∧ Q' x)
      by(auto dest: rel-funD simp add: rel-pred-def)
    show ?wit (λx. P x ∧ Q x) if ?wit P    ?wit Q using that P Q PQ
      by(auto intro: is-filter.conj[OF G])
```

assume $\forall x.\ P\ x \longrightarrow Q\ x$
with $P'\ Q'$ ‹*right-total C'*› **have** $\forall x.\ P'\ x \longrightarrow Q'\ x$ **by**(*metis* (*full-types*) *apply-rsp' right-total-def*)
then show *?wit Q* **if** *?wit P* **using** *P Q that* **by**(*simp add*: *is-filter.mono*[*OF G*])
qed
show *?wit* $(\lambda x.\ P\ x \wedge Q\ x)$ **if** *P*: *?wit P* **and** *Q*: *?wit Q* **using** $*$ *that* **by** *simp*
show *?wit Q* **if** *P*: *?wit P* **and** *imp*: $\forall x.\ P\ x \longrightarrow Q\ x$ **using** $*$ *that* **by** *simp*
qed

end

The following example shows that filters do not satisfy **lift-bnf**'s condition.

experiment begin
unbundle *lifting-syntax*

definition *raw-filtermap f* = ((*f* −−−> *id*) −−−> *id*)

lemma *raw-filtermap-apply*: *raw-filtermap f F* = $(\lambda P.\ F\ (\lambda x.\ P\ (f\ x)))$
unfolding *raw-filtermap-def*
by (*simp add*: *map-fun-def comp-def*)

lemma *filtermap f* = *Abs-filter* ∘ *raw-filtermap f* ∘ *Rep-filter*
unfolding *filtermap-def eventually-def*
by (*simp add*: *fun-eq-iff raw-filtermap-apply*)

definition *Z* **where**
Z = {{(*False*, *False*), (*False*, *True*)}, {(*False*, *False*), (*True*, *False*)},
{(*False*, *False*), (*False*, *True*), (*True*, *False*), (*True*, *True*)}}

abbreviation *Z'* ≡ $(\lambda P.$ *Collect P* ∈ *Z*$)$

lemma *is-filter* (*raw-filtermap fst Z'*)
unfolding *Z-def raw-filtermap-apply*
apply (*rule is-filter.intro*)
apply (*simp add*: *set-eq-iff*; *smt*)+
done

lemma *is-filter* (*raw-filtermap snd Z'*)
unfolding *Z-def raw-filtermap-apply*
apply (*rule is-filter.intro*)
apply (*simp add*: *set-eq-iff*; *smt*)+
done

lemma ¬ *is-filter Z'*
apply (*rule*)
apply (*drule is-filter.mono*[*of* - $\lambda x.\ x \in$ {(*False*, *False*), (*False*, *True*)}
$\lambda x.\ x \in$ {(*False*, *False*), (*False*, *True*), (*True*, *False*)}])
apply (*auto 3 0 simp add*: *Z-def*)

done

end

7.6 Almost-everywhere equal sequences

inductive *aeseq-eq* :: $(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$ **for** $f\ g$ **where**
aeseq-eq f g **if** *finite* $\{n.\ f\ n \neq g\ n\}$

lemma *equivp-aeseq-eq*: *equivp aeseq-eq*
proof(*rule equivpI*)
show *reflp aeseq-eq* **by**(*simp add*: *reflp-def aeseq-eq.simps*)
show *symp aeseq-eq* **by**(*simp add*: *symp-def aeseq-eq.simps eq-commute*)
have *finite* $\{n.\ f\ n \neq h\ n\}$ **if** *finite* $\{n.\ f\ n \neq g\ n\}$ *finite* $\{n.\ g\ n \neq h\ n\}$
for $f\ g\ h :: nat \Rightarrow 'b$
using *finite-subset*[*of* $\{n.\ f\ n \neq h\ n\}$ $\{n.\ f\ n \neq g\ n\} \cup \{n.\ g\ n \neq h\ n\}$] *that*
by(*fastforce intro*: *finite-subset*)
then show *transp aeseq-eq* **by**(*auto simp add*: *transp-def aeseq-eq.simps*)
qed

quotient-type $'a$ *aeseq* $=$ $nat \Rightarrow 'a$ / *aeseq-eq* **by**(*rule equivp-aeseq-eq*)

lift-definition *map-aeseq* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *aeseq* $\Rightarrow 'b$ *aeseq* **is** $(\circ)$
by(*auto simp add*: *aeseq-eq.simps elim*: *finite-subset*[*rotated*])

lemma *map-aeseq-id*: *map-aeseq id x* $= x$
by *transfer*(*simp add*: *equivp-reflp*[*OF equivp-aeseq-eq*])

lemma *map-aeseq-comp*: *map-aeseq f* (*map-aeseq g x*) $=$ *map-aeseq* $(f \circ g)$ *x*
by *transfer*(*simp add*: *o-assoc equivp-reflp*[*OF equivp-aeseq-eq*])

lift-definition *rel-aeseq* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a$ *aeseq* $\Rightarrow 'b$ *aeseq* $\Rightarrow bool$ **is**
$\lambda R\ f\ g.$ *finite* $\{n.\ \neg\ R\ (f\ n)\ (g\ n)\}$
proof(*unfold aeseq-eq.simps*)
show *finite* $\{n.\ \neg\ R\ (f\ n)\ (g\ n)\} \longleftrightarrow$ *finite* $\{n.\ \neg\ R\ (f'\ n)\ (g'\ n)\}$
if *ff'*: *finite* $\{n.\ f\ n \neq f'\ n\}$ **and** *gg'*: *finite* $\{n.\ g\ n \neq g'\ n\}$
for R **and** $f\ f' :: nat \Rightarrow 'a$ **and** $g\ g' :: nat \Rightarrow 'b$
proof(*rule iffI*)
assume *finite* $\{n.\ \neg\ R\ (f\ n)\ (g\ n)\}$
with *ff' gg'* **have** *finite* $(\{n.\ \neg\ R\ (f\ n)\ (g\ n)\} \cup \{n.\ f\ n \neq f'\ n\} \cup \{n.\ g\ n \neq g'\ n\})$ **by** *simp*
then show *finite* $\{n.\ \neg\ R\ (f'\ n)\ (g'\ n)\}$ **by**(*rule finite-subset*[*rotated*]) *auto*
next
assume *finite* $\{n.\ \neg\ R\ (f'\ n)\ (g'\ n)\}$
with *ff' gg'* **have** *finite* $(\{n.\ \neg\ R\ (f'\ n)\ (g'\ n)\} \cup \{n.\ f\ n \neq f'\ n\} \cup \{n.\ g\ n \neq g'\ n\})$ **by** *simp*
then show *finite* $\{n.\ \neg\ R\ (f\ n)\ (g\ n)\}$ **by**(*rule finite-subset*[*rotated*]) *auto*
qed
qed

lemma *rel-aeseq-mono*: $R \leq S \Longrightarrow$ *rel-aeseq* $R \leq$ *rel-aeseq* S
by(*rule predicate2I*; *transfer*; *auto intro*: *finite-subset*[*rotated*])

lemma *rel-aeseq-eq*: *rel-aeseq* $(=) = (=)$
by(*intro ext*; *transfer*; *simp add*: *aeseq-eq.simps*)

lemma *rel-aeseq-conversep*: *rel-aeseq* $R^{-1-1} =$ (*rel-aeseq* $R)^{-1-1}$
by(*simp add*: *fun-eq-iff*; *transfer*; *simp*)

lemma *map-aeseq-parametric*: **includes** *lifting-syntax* **shows**
$((A$ ===> $B)$ ===> *rel-aeseq* A ===> *rel-aeseq* $B)$ *map-aeseq map-aeseq*
by(*intro rel-funI*; *transfer*; *auto elim*: *finite-subset*[*rotated*] *dest*: *rel-funD*)

lemma *rel-aeseq-distr*: *rel-aeseq* $(R$ *OO* $S) =$ *rel-aeseq* R *OO rel-aeseq* S
apply(*intro ext*)
apply(*transfer fixing*: R S)
apply(*safe*)
subgoal for f h
apply(*rule relcomppI*[**where** b=λn. *SOME* z. R $(f$ $n)$ $z \wedge S$ z $(h$ $n)$])
apply(*auto elim*!: *finite-subset*[*rotated*] *intro*: *someI2*)
done
subgoal for f h g
apply(*rule finite-subset*[**where** B=$\{n. \neg R$ $(f$ $n)$ $(g$ $n)\} \cup \{n. \neg S$ $(g$ $n)$ $(h$ $n)\}$])
apply *auto*
done
done

end

8 Example: deterministic discrete system

theory *DDS* **imports**
Concrete-Examples
HOL−Library.Rewrite
HOL−Library.FSet
begin

unbundle *lifting-syntax*

8.1 Definition and generalised mapper and relator

codatatype $('a, 'b)$ *dds* = *DDS* (*run*: $'a \Rightarrow 'b \times ('a, 'b)$ *dds*)
for *map*: *map-dds'*
rel: *rel-dds'*

primcorec *map-dds* :: $('a' \Rightarrow 'a) \Rightarrow ('b \Rightarrow 'b') \Rightarrow ('a, 'b)$ *dds* $\Rightarrow ('a', 'b')$ *dds*
where

run (*map-dds f g S*) = (λ*a*. *map-prod g* (*map-dds f g*) (*run S* (*f a*)))

lemma *map-dds-id*: *map-dds id id S* = *S*
by(*coinduction arbitrary*: *S*)(*auto simp add*: *rel-fun-def prod.rel-map intro*: *prod.rel-refl-strong*)

lemma *map-dds-comp*: *map-dds f g* (*map-dds f′ g′ S*) = *map-dds* (*f′* ∘ *f*) (*g* ∘ *g′*)
S
by(*coinduction arbitrary*: *S*)(*auto simp add*: *rel-fun-def prod.rel-map intro*: *prod.rel-refl-strong*)

coinductive *rel-dds* :: ('*a* ⇒ '*a′* ⇒ *bool*) ⇒ ('*b* ⇒ '*b′* ⇒ *bool*) ⇒ ('*a*, '*b*) *dds* ⇒
('*a′*, '*b′*) *dds* ⇒ *bool*
for *A B* **where**
rel-dds A B S S′ **if** *rel-fun A* (*rel-prod B* (*rel-dds A B*)) (*run S*) (*run S′*)

lemma *rel-dds′-rel-dds*: *rel-dds′ B* = *rel-dds* (=) *B*
apply (*intro ext iffI*)
apply (*erule rel-dds.coinduct*)
apply (*erule dds.rel-cases*)
apply (*simp*)
apply (*erule rel-fun-mono*[*THEN predicate2D*, *OF order-refl*, *rotated* −*1*])
apply (*rule prod.rel-mono*[*OF order-refl*])
apply (*blast*)
apply (*erule dds.rel-coinduct*)
apply (*erule rel-dds.cases*)
apply (*simp*)
done

lemma *rel-dds-eq* [*relator-eq*]: *rel-dds* (=) (=) = (=)
apply(*rule ext iffI*)+
subgoal by(*erule dds.coinduct*)(*erule rel-dds.cases*; *simp*)
subgoal by(*erule rel-dds.coinduct*)(*auto simp add*: *rel-fun-def intro*!: *prod.rel-refl-strong*)
done

lemma *rel-dds-mono* [*relator-mono*]: *rel-dds A B* ≤ *rel-dds A′ B′* **if** *A′* ≤ *A* *B*
≤ *B′*
apply(*rule predicate2I*)
apply(*erule rel-dds.coinduct*)
apply(*erule rel-dds.cases*)
apply *simp*
apply(*erule BNF-Def.rel-fun-mono*)
apply(*auto intro*: *that*[*THEN predicate2D*])
done

lemma *rel-dds-conversep*: *rel-dds* A^{-1-1} B^{-1-1} = (*rel-dds A B*)$^{-1-1}$
apply(*intro ext iffI*; *simp*)
subgoal
apply(*erule rel-dds.coinduct*; *erule rel-dds.cases*; *simp del*: *conversep-iff*)
apply(*rewrite conversep-iff*[*symmetric*])
apply(*fold rel-fun-conversep prod.rel-conversep*)

```
    apply(erule BNF-Def.rel-fun-mono)
     apply(auto simp del: conversep-iff)
    done
  subgoal
   apply(erule rel-dds.coinduct; erule rel-dds.cases; clarsimp simp del: conversep-iff)
    apply(rewrite in asm conversep-iff[symmetric])
    apply(fold rel-fun-conversep prod.rel-conversep)
    apply(erule BNF-Def.rel-fun-mono)
     apply(auto simp del: conversep-iff)
    done
  done

lemma DDS-parametric [transfer-rule]:
  ((A ===> rel-prod B (rel-dds A B)) ===> rel-dds A B) DDS DDS
  by(auto intro!: rel-dds.intros)

lemma run-parametric [transfer-rule]:
  (rel-dds A B ===> A ===> rel-prod B (rel-dds A B)) run run
  by(auto elim: rel-dds.cases)

lemma corec-dds-parametric [transfer-rule]:
  ((S ===> A ===> rel-prod B (rel-sum (rel-dds A B) S)) ===> S ===>
rel-dds A B) corec-dds corec-dds
  apply(rule rel-funI)+
  subgoal premises prems for f g s s' using prems(2)
    apply(coinduction arbitrary: s s')
    apply simp
    apply(rule comp-transfer[THEN rel-funD, THEN rel-funD, rotated])
     apply(erule prems(1)[THEN rel-funD])
   apply(rule prod.map-transfer[THEN rel-funD, THEN rel-funD, OF id-transfer])
    apply(fastforce elim!: rel-sum.cases)
    done
  done

lemma map-dds-parametric [transfer-rule]:
  ((A' ===> A) ===> (B ===> B') ===> rel-dds A B ===> rel-dds A' B')
map-dds map-dds
  unfolding map-dds-def by transfer-prover

lemmas map-dds-rel-cong = map-dds-parametric[unfolded rel-fun-def, rule-format,
rotated -1]

lemma rel-dds-Grp:
  rel-dds (Grp UNIV f)^-1-1 (Grp UNIV g) = Grp UNIV (map-dds f g)
  apply(rule ext iffI)+
   apply(simp add: Grp-apply)
   apply(rule sym)
   apply(fold rel-dds-eq)
   apply(rewrite in rel-dds - - - ⌑ map-dds-id[symmetric])
```

```
  apply(erule map-dds-parametric[THEN rel-funD, THEN rel-funD, THEN rel-funD,
rotated −1];
      simp add: Grp-apply rel-fun-def)
  apply(erule GrpE; clarsimp)
  apply(rewrite in rel-dds - - ⌑ map-dds-id[symmetric])
  apply(rule map-dds-parametric[THEN rel-funD, THEN rel-funD, THEN rel-funD,
rotated −1])
    apply(subst rel-dds-eq; simp)
   apply(simp-all add: Grp-apply rel-fun-def)
  done

lemma rel-dds-pos-distr [relator-distr]:
  rel-dds A B OO rel-dds C D ≤ rel-dds (A OO C) (B OO D)
  apply (rule predicate2I)
  apply (erule relcomppE)
  subgoal for x y z
    apply (coinduction arbitrary: x y z)
    apply (simp)
    apply (rule rel-fun-mono[THEN predicate2D, OF order-refl,
          of rel-prod B (rel-dds A B) OO rel-prod D (rel-dds C D)])
     apply (rule order-trans)
      apply (rule prod.rel-compp[symmetric, THEN eq-refl])
     apply (rule prod.rel-mono[OF order-refl])
     apply (blast)
    apply (rule rel-fun-pos-distr[THEN predicate2D])
     apply (simp)
    apply (rule relcomppI)
     apply (auto elim: rel-dds.cases)
    done
  done

lemma Quotient-dds [quot-map]:
  assumes Quotient R1 Abs1 Rep1 T1 and Quotient R2 Abs2 Rep2 T2
   shows Quotient (rel-dds R1 R2) (map-dds Rep1 Abs2) (map-dds Abs1 Rep2)
(rel-dds T1 T2)
  unfolding Quotient-alt-def5
proof (intro conjI, goal-cases)
  case 1
  have rel-dds T1 T2 ≤ rel-dds (Grp UNIV Rep1)^-1-1 (Grp UNIV Abs2)
    apply (rule rel-dds-mono)
      apply (rule assms(1)[unfolded Quotient-alt-def5, THEN conjunct2, THEN
conjunct1,
          unfolded conversep-le-swap])
    apply (rule assms(2)[unfolded Quotient-alt-def5, THEN conjunct1])
    done
  also have ... = Grp UNIV (map-dds Rep1 Abs2) using rel-dds-Grp .
  finally show ?case .
next
  case 2
```

have *Grp UNIV* (*map-dds Abs1 Rep2*) = *rel-dds* (*Grp UNIV Abs1*)$^{-1-1}$ (*Grp UNIV Rep2*)
using *rel-dds-Grp*[*symmetric*] .
also have ... ≤ *rel-dds T1*$^{-1-1}$ *T2*$^{-1-1}$
apply (*rule rel-dds-mono*)
apply (*simp*)
apply (*rule assms*(*1*)[*unfolded Quotient-alt-def5*, *THEN conjunct1*])
apply (*rule assms*(*2*)[*unfolded Quotient-alt-def5*, *THEN conjunct2*, *THEN conjunct1*])
done
finally show *?case* **by** (*simp add*: *rel-dds-conversep*)
next
case *3*
show *?case*
apply (*rule antisym*)
apply (*rule predicate2I*)
apply (*rule relcomppI*)
apply (*subst map-dds-id*[*symmetric*])
apply (*erule map-dds-rel-cong*)
apply (*simp-all*)[*2*]
apply (*erule assms*(*1*)[*THEN Quotient-rep-equiv1*])
apply (*erule assms*(*2*)[*THEN Quotient-equiv-abs1*])
apply (*subst rel-dds-conversep*[*symmetric*])
subgoal for *x y*
apply (*subgoal-tac map-dds Rep1 Abs2 y* = *map-dds Rep1 Abs2 x*)
apply (*simp*)
apply (*subst* (*3*) *map-dds-id*[*symmetric*])
apply (*erule map-dds-rel-cong*)
apply (*simp-all*)
apply (*erule assms*(*1*)[*THEN Quotient-rep-equiv2*])
apply (*erule assms*(*2*)[*THEN Quotient-equiv-abs2*])
apply (*rule sym*)
apply (*subst rel-dds-eq*[*symmetric*])
apply (*erule map-dds-rel-cong*)
apply (*simp*, *rule assms*(*1*)[*THEN Quotient-rep-reflp*])
apply (*erule assms*(*2*)[*THEN Quotient-rel-abs*])
done
apply (*unfold rel-dds-conversep*[*symmetric*]
assms[*unfolded Quotient-alt-def5*, *THEN conjunct2*, *THEN conjunct2*])
apply (*rule rel-dds-pos-distr*)
done
qed

This is just the co-iterator.

primcorec *dds-of* :: ('*s* ⇒ '*a* ⇒ ('*b* × '*s*)) ⇒ '*s* ⇒ ('*a*, '*b*) *dds* **where**
run (*dds-of f s*) = *map-prod id* (*dds-of f*) ∘ *f s*

lemma *dds-of-parametric* [*transfer-rule*]:
((*S* ===> *A* ===> *rel-prod B S*) ===> *S* ===> *rel-dds A B*) *dds-of dds-of*

unfolding *dds-of-def* **by** *transfer-prover*

8.2 Evenness of partial sums

definition *even-psum* :: (*int*, *bool*) *dds* **where**
even-psum = *dds-of* (λ*psum n*. (*even* (*psum* + *n*), *psum* + *n*)) *0*

definition *even-psum-nat* :: (*nat*, *bool*) *dds* **where**
even-psum-nat = *map-dds int id even-psum*

8.3 Composition

primcorec *compose* :: ('*a*, '*b*) *dds* ⇒ ('*b*, '*c*) *dds* ⇒ ('*a*, '*c*) *dds* (**infixl** · *120*) **where**
run (*S1* · *S2*) = (λ*a*. *let* (*b*, *S1'*) = *run S1 a*; (*c*, *S2'*) = *run S2 b in* (*c*, *S1'* · *S2'*))

lemma *compose-parametric* [*transfer-rule*]:
(*rel-dds A B* ===> *rel-dds B C* ===> *rel-dds A C*) (·) (·)
unfolding *compose-def* **by** *transfer-prover*

For the following lemma, a direct proof by induction is easy as the inner functor of the *dds* codatatype is fairly simple.

lemma *map-dds f g S1* · *S2* = *map-dds f id* (*S1* · *map-dds g id S2*)
apply(*coinduction arbitrary*: *S1 S2*)
apply(*auto simp add*: *case-prod-map-prod rel-fun-def split*: *prod.split*)
done

However, we can also follow the systematic route via parametricity:

lemma *compose-map1*: *map-dds f g S1* · *S2* = *map-dds f id* (*S1* · *map-dds g id S2*)
for *S1* :: ('*a*, '*b*) *dds* **and** *S2* :: ('*b*, '*c*) *dds*
using *compose-parametric*[*of* (*Grp UNIV f*)$^{-1-1}$ *Grp UNIV g* *Grp UNIV id* :: '*c* ⇒ '*c* ⇒ *bool*]
apply(*rewrite* **in** - ===> *rel-dds* ⌑ - ===> - **in** *asm conversep-conversep*[*symmetric*])
apply(*rewrite* **in** - ===> *rel-dds* - ⌑ ===> - **in** *asm conversep-Grp-id*[*symmetric*])
apply(*simp only*: *rel-dds-conversep rel-dds-Grp*)
apply(*simp add*: *rel-fun-def Grp-apply*)
done

lemma *compose-map2*: *S1* · *map-dds f g S2* = *map-dds id g* (*map-dds id f S1* · *S2*)
for *S1* :: ('*a*, '*b*) *dds* **and** *S2* :: ('*b*, '*c*) *dds*
using *compose-parametric*[*of Grp UNIV id* :: '*a* ⇒ '*a* ⇒ *bool* (*Grp UNIV f*)$^{-1-1}$ *Grp UNIV g*]
apply(*rewrite* **in** *rel-dds* ⌑ - ===> - **in** *asm conversep-conversep*[*symmetric*])
apply(*rewrite* **in** - ===> - ===> *rel-dds* ⌑ - **in** *asm conversep-Grp-id*[*symmetric*])
apply(*simp only*: *rel-dds-conversep rel-dds-Grp*)
apply(*simp add*: *rel-fun-def Grp-apply*)

done

primcorec *parallel* :: ('*a*, '*b*) *dds* ⇒ ('*c*, '*d*) *dds* ⇒ ('*a* + '*c*, '*b* + '*d*) *dds* (**infixr** ‖ *130*) **where**
run (*S1* ‖ *S2*) = (λ*x*. *case x of*
Inl a ⇒ *let* (*b*, *S1'*) = *run S1 a in* (*Inl b*, *S1'* ‖ *S2*)
| *Inr c* ⇒ *let* (*d*, *S2'*) = *run S2 c in* (*Inr d*, *S1* ‖ *S2'*))

lemma *parallel-parametric* [*transfer-rule*]:
(*rel-dds A B* ===> *rel-dds C D* ===> *rel-dds* (*rel-sum A C*) (*rel-sum B D*)) (‖) (‖)
unfolding *parallel-def* **by** *transfer-prover*

lemma *map-parallel*:
map-dds f h S1 ‖ *map-dds g k S2* = *map-dds* (*map-sum f g*) (*map-sum h k*) (*S1* ‖ *S2*)
using *parallel-parametric*[**where** *A*=(*Grp UNIV f*)$^{-1-1}$ **and** *B*=*Grp UNIV h* **and**
C=(*Grp UNIV g*)$^{-1-1}$ **and** *D*=*Grp UNIV k*]
by(*simp add*: *sum.rel-conversep sum.rel-Grp rel-dds-Grp*)(*simp add*: *rel-fun-def Grp-apply*)

8.4 Graph traversal: refinement and quotients

lemma *finite-Image*:
finite A ⟹ *finite* (*R* `` *A*) ⟷ (∀ *x*∈*A*. *finite* {*y*. (*x*, *y*) ∈ *R*})
by(*simp add*: *Image-def*)

context includes *fset.lifting* **begin**
lift-definition *fImage* :: ('*a* × '*b*) *fset* ⇒ '*a fset* ⇒ '*b fset* **is** *Image* **parametric** *Image-parametric*
by(*auto simp add*: *finite-Image intro*: *finite-subset*[*OF* - *finite-imageI*])

lemmas *fImage-iff* = *Image-iff*[*Transfer.transferred*]
lemmas *fImageI* [*intro*] = *ImageI*[*Transfer.transferred*]
lemmas *fImageE* [*elim*!] = *ImageE*[*Transfer.transferred*]
lemmas *rev-fImageI* = *rev-ImageI*[*Transfer.transferred*]
lemmas *fImage-mono* = *Image-mono*[*Transfer.transferred*]

lifting-update *fset.lifting*
lifting-forget *fset.lifting*
end

type-synonym '*a graph* = ('*a* × '*a*) *fset*

definition *traverse* :: '*a graph* ⇒ ('*a fset*, '*a fset*) *dds* **where**
traverse E = *dds-of* (λ*visited A*. ((*fImage E A*) |−| *visited*, *visited* |∪| *A*)) {||}

type-synonym '*a graph'* = ('*a* × '*a*) *list*

definition *traverse-impl* :: *'a graph'* ⇒ (*'a list*, *'a list*) *dds* **where**
traverse-impl E =
dds-of (λ*visited A*. (*map snd* [(*x*, *y*)←*E* . *x* ∈ *set A* ∧ *y* |∉| *visited*],
visited |∪| *fset-of-list A*)) {||}

definition *list-fset-rel* :: *'a list* ⇒ *'a fset* ⇒ *bool* **where**
list-fset-rel xs A ⟷ *fset-of-list xs* = *A*

lemma *traverse-refinement*: — This is the refinement lemma.
(*list-fset-rel* ===> *rel-dds list-fset-rel list-fset-rel*) *traverse-impl traverse*
unfolding *traverse-impl-def traverse-def*
apply(*rule rel-funI*)
apply(*rule dds-of-parametric*[**where** *S*=(=), *THEN rel-funD*, *THEN rel-funD*])
apply(*auto simp add*: *rel-fun-def list-fset-rel-def fset-of-list-elem intro*: *rev-fimage-eqI*)
done

lemma *fset-of-list-parametric* [*transfer-rule*]:
(*list-all2 A* ===> *rel-fset A*) *fset-of-list fset-of-list*
including *fset.lifting* **unfolding** *rel-fun-def*
by *transfer*(*rule list.set-transfer*[*unfolded rel-fun-def*])

lemma *traverse-impl-parametric* [*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A*
shows (*list-all2* (*rel-prod A A*) ===> *rel-dds* (*list-all2 A*) (*list-all2 A*)) *traverse-impl traverse-impl*
unfolding *traverse-impl-def* **by** *transfer-prover*

By constructing finite sets as a quotient of lists, we can synthesise an abstract version of *traverse-impl* automatically, together with a polymorphic refinement lemma.

quotient-type *'a fset'* = *'a list* / *vimage2p set set* (=)
by (*auto intro*: *equivpI reflpI sympI transpI simp add*: *vimage2p-def*)

lift-definition *traverse''* :: (*'a* × *'a*) *fset'* ⇒ (*'a fset'*, *'a fset'*) *dds*
is *traverse-impl* :: *'a graph'* ⇒ - **parametric** *traverse-impl-parametric*
unfolding *traverse-impl-def*
apply (*rule dds-of-parametric*[**where** *S*=(=), *THEN rel-funD*, *THEN rel-funD*])
apply (*auto simp add*: *rel-fun-def vimage2p-def fset-of-list-elem*)
done

8.5 Generalised rewriting

definition *accumulate* :: (*'a fset*, *'a fset*) *dds* **where**
accumulate = *dds-of* (λ*A X*. (*A* |∪| *X*, *A* |∪| *X*)) {||}

lemma *accumulate-mono*: *rel-dds* (|⊆|) (|⊆|) *accumulate accumulate*
unfolding *accumulate-def*
apply (*rule dds-of-parametric*[*THEN rel-funD*, *THEN rel-funD*, *of* (|⊆|)])

```
  apply (intro rel-funI rel-prod.intros)
   apply (erule (1) funion-mono)+
  apply (simp)
  done

lemma traverse-mono: ((|⊆|) ===> rel-dds (=) (|⊆|)) traverse traverse
  unfolding traverse-def
  apply (rule rel-funI)
  apply (rule dds-of-parametric[THEN rel-funD, THEN rel-funD, of (=)])
   apply (intro rel-funI rel-prod.intros)
    apply (simp)
    apply (rule fminus-mono)
     apply (erule fImage-mono)
     apply (simp-all)
  done

lemma
  assumes G |⊆| H
  shows rel-dds (=) (|⊆|) (traverse G · accumulate) (traverse H · accumulate)
  apply (rule compose-parametric[THEN rel-funD, THEN rel-funD])
   apply (rule traverse-mono[THEN rel-funD])
   apply (rule assms)
  apply (rule accumulate-mono)
  done

definition seen :: ('a fset, 'a fset) dds where
  seen = dds-of (λS X. (S |∩| X, S |∪| X)) {||}

lemma seen-mono: rel-dds (|⊆|) (|⊆|) seen seen
  unfolding seen-def
  apply (rule dds-of-parametric[THEN rel-funD, THEN rel-funD, of (|⊆|)])
   apply (intro rel-funI rel-prod.intros)
    apply (erule (1) finter-mono funion-mono)+
  apply (simp)
  done

lemma
  assumes G |⊆| H
  shows rel-dds (=) (|⊆|) (traverse G · seen) (traverse H · seen)
  apply (rule compose-parametric[THEN rel-funD, THEN rel-funD])
   apply (rule traverse-mono[THEN rel-funD])
   apply (rule assms)
  apply (rule seen-mono)
  done

end
```

References

[1] J. C. Blanchette, A. Popescu, and D. Traytel. Operations on bounded natural functors. *Archive of Formal Proofs*, Dec. 2017. http://isa-afp.org/entries/BNF_Operations.html, Formal proof development.

[2] A. Lochbihler and J. Schneider. Relational parametricity and quotient preservation for modular (co)datatypes. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving (ITP 2018)*, LNCS. Springer, 2018.