Bounded Natural Functors with Covariance and Contravariance

Andreas Lochbihler and Joshua Schneider

June 24, 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 **4**

2 Axiomatisation **5**
2.1 First abstract $\mathrm{BNF}_{\mathrm{CC}}$ 5
2.1.1 Axioms and basic definitions 5
2.1.2 Derived rules 8
2.1.3 F is a BNF 9
2.1.4 Composition witness 10
2.2 Second abstract $\mathrm{BNF}_{\mathrm{CC}}$ 11
2.2.1 Axioms and basic definitions 11
2.2.2 Derived rules 13
2.2.3 G is a BNF 14
2.2.4 Composition witness 15

3 Simple operations: demotion, merging, composition **15**
3.1 Composition in a live position 16
3.2 Composition in a covariant position 21
3.3 Composition in a contravariant position 25
3.4 Composition in a fixed position 30

4 Least and greatest fixpoints **33**
4.1 Least fixpoint 33
4.1.1 $\mathrm{BNF}_{\mathrm{CC}}$ structure 33
4.1.2 Parametricity laws 38
4.2 Greatest fixpoints 38
4.2.1 $\mathrm{BNF}_{\mathrm{CC}}$ structure 38
4.2.2 Parametricity laws 42

5 Subtypes **43**
5.1 $\mathrm{BNF}_{\mathrm{CC}}$ structure 43
5.2 Closedness under zippings 47
5.3 Subtypes of BNFs without co- and contravariance 49

6 Quotient preservation **51**

7 Concrete $\mathrm{BNF}_{\mathrm{CC}}$s **51**
7.1 Function space 51
7.2 Covariant powerset 53
7.3 Bounded sets 53
7.4 Contravariant powerset (sets as predicates) 54
7.5 Filter 56
7.6 Almost-everywhere equal sequences 58

8 Example: deterministic discrete system **59**
8.1 Definition and generalised mapper and relator 59
8.2 Evenness of partial sums 60
8.3 Composition 60
8.4 Graph traversal: refinement and quotients 61
8.5 Generalised rewriting 63

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*
⟨*proof*⟩

lemma *Grp-apply*: *Grp A f x y* ⟷ *y* = *f x* ∧ *x* ∈ *A*
⟨*proof*⟩

lemma *conversep-Grp-id*: (*Grp A id*)$^{-1-1}$ = *Grp A id*
⟨*proof*⟩

lemma *eq-onp-compp-Grp*: *eq-onp P OO Grp A f* = *Grp* (*Collect P* ∩ *A*) *f*
⟨*proof*⟩

consts *relcompp-witness* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*b* ⇒ '*c* ⇒ *bool*) ⇒ '*a* × '*c* ⇒ '*b*

specification (*relcompp-witness*)
relcompp-witness1: (*A OO B*) (*fst xy*) (*snd xy*) ⟹ *A* (*fst xy*) (*relcompp-witness A B xy*)
relcompp-witness2: (*A OO B*) (*fst xy*) (*snd xy*) ⟹ *B* (*relcompp-witness A B xy*) (*snd xy*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *Quotient-equiv-abs1*: ⟦ *Quotient R Abs Rep T*; *R x y* ⟧ ⟹ *T x* (*Abs y*)
⟨*proof*⟩

lemma *Quotient-equiv-abs2*: ⟦ *Quotient R Abs Rep T*; *R x y* ⟧ ⟹ *T y* (*Abs x*)
⟨*proof*⟩

lemma *Quotient-rep-equiv1*: ⟦ *Quotient R Abs Rep T*; *T a b* ⟧ ⟹ *R a* (*Rep b*)
⟨*proof*⟩

lemma *Quotient-rep-equiv2*: $[\![$ *Quotient R Abs Rep T*; *T a b* $]\!] \Longrightarrow R\ (Rep\ b)\ a$
⟨*proof*⟩

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* $]\!] \Longrightarrow$
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: ⋀*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: ⋀*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:
⋀*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-fun 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* ⇒ '*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-neg-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 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'*) ≤
(*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')

axiomatization where
rel-F-pos-distr-cond-eq:
⋀*tytok*. *rel-F-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
and
rel-F-neg-distr-cond-eq:
⋀*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* ⇒ *'l1 set*
set2-F :: (*'l1*, *'l2*, *'l3*, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F* ⇒ *'l2 set*
set3-F :: (*'l1*, *'l2*, *'l3*, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F* ⇒ *'l3 set*
bd-F :: (*'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *Fbd rel*

axiomatization where
set1-F-map: ⋀*l1 l2 l3*. *set1-F* ∘ *mapl-F l1 l2 l3* = *image l1* ∘ *set1-F* **and**
set2-F-map: ⋀*l1 l2 l3*. *set2-F* ∘ *mapl-F l1 l2 l3* = *image l2* ∘ *set2-F* **and**
set3-F-map: ⋀*l1 l2 l3*. *set3-F* ∘ *mapl-F l1 l2 l3* = *image l3* ∘ *set3-F* **and**
bd-F-card-order: *card-order bd-F* **and**
bd-F-cinfinite: *cinfinite bd-F* **and**
set1-F-bound: ⋀*x* :: (-, -, -, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F*.
card-of (*set1-F x*) ≤*o* (*bd-F* :: (*'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *Fbd rel*) **and**
set2-F-bound: ⋀*x* :: (-, -, -, *'co1*, *'co2*, *'co3*, *'contra1*, *'contra2*, *'contra3*, *'f*) *F*.

card-of (*set2-F x*) $\leq o$ (*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*) $\leq o$ (*bd-F* :: ('*co1*, '*co2*, '*co3*, '*contra1*, '*contra2*, '*contra3*, '*f*) *Fbd rel*) **and**
mapl-F-cong: $\bigwedge$*l1 l1' l2 l2' l3 l3' x*.
$[\![\bigwedge z.\ z \in$ *set1-F x* $\Longrightarrow$ *l1 z* = *l1' z*; $\bigwedge z.\ z \in$ *set2-F x* $\Longrightarrow$ *l2 z* = *l2' z*;
$\bigwedge z.\ z \in$ *set3-F x* $\Longrightarrow$ *l3 z* = *l3' z* $]\!] \Longrightarrow$
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*;
$\bigwedge a\ b.\ a \in$ *set1-F x* $\Longrightarrow$ *b* $\in$ *set1-F y* $\Longrightarrow$ *L1 a b* $\Longrightarrow$ *L1' a b*;
$\bigwedge a\ b.\ a \in$ *set2-F x* $\Longrightarrow$ *b* $\in$ *set2-F y* $\Longrightarrow$ *L2 a b* $\Longrightarrow$ *L2' a b*;
$\bigwedge a\ b.\ a \in$ *set3-F x* $\Longrightarrow$ *b* $\in$ *set3-F y* $\Longrightarrow$ *L3 a b* $\Longrightarrow$ *L3' a b* $]\!] \Longrightarrow$
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*
⟨*proof*⟩

lemma *map-F-id*: *map-F id id id id id id id id id x* = *x*
⟨*proof*⟩

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

lemma *rell-F-mono*: $[\![$ *L1* $\leq$ *L1'*; *L2* $\leq$ *L2'*; *L3* $\leq$ *L3'* $]\!] \Longrightarrow$ *rell-F L1 L2 L3* $\leq$ *rell-F L1' L2' L3'*
⟨*proof*⟩

lemma *mapl-F-id0*: *mapl-F id id id* = *id*
⟨*proof*⟩

lemma *mapl-F-id*: *mapl-F id id id x* = *x*
⟨*proof*⟩

lemma *mapl-F-comp*: *mapl-F l1 l2 l3* $\circ$ *mapl-F l1' l2' l3'* = *mapl-F* (*l1* $\circ$ *l1'*) (*l2* $\circ$ *l2'*) (*l3* $\circ$ *l3'*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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
⟨*proof*⟩

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*)
⟨*proof*⟩

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'*
⟨*proof*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

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-G L1 L2 Co1 Co2 Contra1 Contra2 $\le$ *rel-G L1' L2' Co1' Co2' Contra1' Contra2'* **and**
rel-G-eq: *rel-G* (=) (=) (=) (=) (=) (=) = (=) **and**
rel-G-conversep: $\bigwedge L1\ L2\ Co1\ Co2\ Contra1\ Contra2.$
rel-G $L1^{-1-1}$ $L2^{-1-1}$ $Co1^{-1-1}$ $Co2^{-1-1}$ $Contra1^{-1-1}$ $Contra2^{-1-1}$ = (*rel-G L1 L2 Co1 Co2 Contra1 Contra2*)$^{-1-1}$ **and**
map-G-id0: *map-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-G l1 l2 co1 co2 contra1 contra2 $\circ$ *map-G l1' l2' co1' co2' contra1' contra2'* =
map-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-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-G L1 L2 Co1 Co2 Contra1 Contra2*)
(*rel-G L1' L2' Co1' Co2' Contra1' Contra2'*)))))))
map-G map-G

definition *rel-G-pos-distr-cond* :: $('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co1' \Rightarrow 'co1'' \Rightarrow bool) \Rightarrow$
$('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow ('co2' \Rightarrow 'co2'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('l1 \times 'l1' \times 'l1'' \times 'l2 \times 'l2' \times 'l2'' \times 'f)\ itself \Rightarrow bool$ **where**
rel-G-pos-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
$\longleftrightarrow$
$(\forall (L1 :: 'l1 \Rightarrow 'l1' \Rightarrow bool)\ (L1' :: 'l1' \Rightarrow 'l1'' \Rightarrow bool)$
$(L2 :: 'l2 \Rightarrow 'l2' \Rightarrow bool)\ (L2' :: 'l2' \Rightarrow 'l2'' \Rightarrow bool).$
$(rel\text{-}G\ L1\ L2\ Co1\ Co2\ Contra1\ Contra2 :: (-, -, -, -, -, -, 'f)\ G \Rightarrow -)\ OO$
$rel\text{-}G\ L1'\ L2'\ Co1'\ Co2'\ Contra1'\ Contra2' \leq$
$rel\text{-}G\ (L1\ OO\ L1')\ (L2\ OO\ L2')\ (Co1\ OO\ Co1')\ (Co2\ OO\ Co2')$
$(Contra1\ OO\ Contra1')\ (Contra2\ OO\ Contra2'))$

definition *rel-G-neg-distr-cond* :: $('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co1' \Rightarrow 'co1'' \Rightarrow bool) \Rightarrow$
$('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow ('co2' \Rightarrow 'co2'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('l1 \times 'l1' \times 'l1'' \times 'l2 \times 'l2' \times 'l2'' \times 'f)\ itself \Rightarrow bool$ **where**
rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' -
$\longleftrightarrow$
$(\forall (L1 :: 'l1 \Rightarrow 'l1' \Rightarrow bool)\ (L1' :: 'l1' \Rightarrow 'l1'' \Rightarrow bool)$
$(L2 :: 'l2 \Rightarrow 'l2' \Rightarrow bool)\ (L2' :: 'l2' \Rightarrow 'l2'' \Rightarrow bool).$
$rel\text{-}G\ (L1\ OO\ L1')\ (L2\ OO\ L2')\ (Co1\ OO\ Co1')\ (Co2\ OO\ Co2')$
$(Contra1\ OO\ Contra1')\ (Contra2\ OO\ Contra2') \leq$
$(rel\text{-}G\ L1\ L2\ Co1\ Co2\ Contra1\ Contra2 :: (-, -, -, -, -, -, 'f)\ G \Rightarrow -)\ OO$
$rel\text{-}G\ L1'\ L2'\ Co1'\ Co2'\ Contra1'\ Contra2')$

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

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 :: $('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G \Rightarrow 'l1\ set$
set2-G :: $('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G \Rightarrow 'l2\ set$
bd-G :: $('co1, 'co2, 'contra1, 'contra2, 'f)\ Gbd\ rel$
wit-G :: $'l2 \Rightarrow ('l1, 'l2, 'co1, 'co2, 'contra1, 'contra2, 'f)\ G$
— non-emptiness witness for least fixpoint

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

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*
⟨*proof*⟩

lemma *map-G-id*: *map-G id id id id id id x* = *x*
⟨*proof*⟩

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

lemma *rell-G-mono*: $[\![L1 \leq L1';\ L2 \leq L2']\!] \Longrightarrow rell\text{-}G\ L1\ L2 \leq rell\text{-}G\ L1'\ L2'$
⟨*proof*⟩

lemma *mapl-G-id0*: *mapl-G id id* = *id*
⟨*proof*⟩

lemma *mapl-G-id*: *mapl-G id id x* = *x*
⟨*proof*⟩

lemma *mapl-G-comp*: *mapl-G l1 l2* ∘ *mapl-G l1' l2'* = *mapl-G* (*l1* ∘ *l1'*) (*l2* ∘ *l2'*)
⟨*proof*⟩

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*)

⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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'*
⟨*proof*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

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 ≤
rel-FGl L1' L2' L3' Co1' Co2' Co3' Co4' Contra1' Contra2' Contra3' Contra4'
⟨*proof*⟩

lemma *rel-FGl-eq*: *rel-FGl (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)*
⟨*proof*⟩

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)$^{-1-1}$
⟨*proof*⟩

lemma *map-FGl-id0*: *map-FGl id id id id id id id id id id id = id*
⟨*proof*⟩

lemma *map-FGl-comp*: *map-FGl l1 l2 l3 co1 co2 co3 co4 contra1 contra2 contra3 contra4 ∘*

map-FGl l1' l2' l3' co1' co2' co3' co4' contra1' contra2' contra3' contra4' =
map-FGl (l1 ∘ l1') (l2 ∘ l2') (l3 ∘ l3') (co1 ∘ co1') (co2 ∘ co2') (co3 ∘ co3') (co4 ∘ co4')
(contra1' ∘ contra1) (contra2' ∘ contra2) (contra3' ∘ contra3) (contra4' ∘ contra4)
⟨*proof*⟩

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-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
⟨*proof*⟩

definition *rel-FGl-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) ⇒
('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
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *set2-FGl-map*: *set2-FGl* ∘ *mapl-FGl l1 l2 l3* = *image l2* ∘ *set2-FGl*
⟨*proof*⟩

lemma *set3-FGl-map*: *set3-FGl* ∘ *mapl-FGl l1 l2 l3* = *image l3* ∘ *set3-FGl*
⟨*proof*⟩

lemma *bd-FGl-card-order*: *card-order bd-FGl*
⟨*proof*⟩

lemma *bd-FGl-cinfinite*: *cinfinite bd-FGl*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mapl-FGl-cong*:
assumes ⋀*z*. *z* ∈ *set1-FGl x* ⟹ *l1 z* = *l1′ z* **and** ⋀*z*. *z* ∈ *set2-FGl x* ⟹ *l2 z* = *l2′ z*
and ⋀*z*. *z* ∈ *set3-FGl x* ⟹ *l3 z* = *l3′ z*
shows *mapl-FGl l1 l2 l3 x* = *mapl-FGl l1′ l2′ l3′ x*
⟨*proof*⟩

lemma *rell-FGl-mono-strong*:
assumes *rell-FGl L1 L2 L3 x y*
and ⋀*a b*. *a* ∈ *set1-FGl x* ⟹ *b* ∈ *set1-FGl y* ⟹ *L1 a b* ⟹ *L1′ a b*

and $\bigwedge a\ b.\ a \in \mathit{set2\text{-}FGl}\ x \Longrightarrow b \in \mathit{set2\text{-}FGl}\ y \Longrightarrow \mathit{L2}\ a\ b \Longrightarrow \mathit{L2'}\ a\ b$
and $\bigwedge a\ b.\ a \in \mathit{set3\text{-}FGl}\ x \Longrightarrow b \in \mathit{set3\text{-}FGl}\ y \Longrightarrow \mathit{L3}\ a\ b \Longrightarrow \mathit{L3'}\ a\ b$
shows *rell-FGl L1' L2' L3' x y*
⟨*proof*⟩

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*:
⟦ $L1 \leq L1'$; $Co1 \leq Co1'$; $Co2 \leq Co2'$; $Co3 \leq Co3'$; $Co4 \leq Co4'$; $Co5 \leq Co5'$; $Co6 \leq Co6'$;
$Contra1' \leq Contra1$; $Contra2' \leq Contra2$; $Contra3' \leq Contra3$; $Contra4' \leq Contra4$ ⟧ $\Longrightarrow$
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'
⟨*proof*⟩

lemma *rel-FGco-eq*: *rel-FGco* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

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*)$^{-1-1}$
⟨*proof*⟩

lemma *map-FGco-id0*: *map-FGco id id id id id id id id id id id = id*
⟨*proof*⟩

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)
⟨*proof*⟩

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-fun-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
⟨*proof*⟩

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) ⇒

$('co3 \Rightarrow 'co3' \Rightarrow bool) \Rightarrow ('co3' \Rightarrow 'co3'' \Rightarrow bool) \Rightarrow$
$('co4 \Rightarrow 'co4' \Rightarrow bool) \Rightarrow ('co4' \Rightarrow 'co4'' \Rightarrow bool) \Rightarrow$
$('co5 \Rightarrow 'co5' \Rightarrow bool) \Rightarrow ('co5' \Rightarrow 'co5'' \Rightarrow bool) \Rightarrow$
$('co6 \Rightarrow 'co6' \Rightarrow bool) \Rightarrow ('co6' \Rightarrow 'co6'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('contra3 \Rightarrow 'contra3' \Rightarrow bool) \Rightarrow ('contra3' \Rightarrow 'contra3'' \Rightarrow bool) \Rightarrow$
$('contra4 \Rightarrow 'contra4' \Rightarrow bool) \Rightarrow ('contra4' \Rightarrow 'contra4'' \Rightarrow bool) \Rightarrow$
$('l1 \times 'l1' \times 'l1'' \times 'f1 \times 'f2)$ *itself* $\Rightarrow$ *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' - $\longleftrightarrow$
$(\forall (L1 :: 'l1 \Rightarrow 'l1' \Rightarrow bool)\ (L1' :: 'l1' \Rightarrow 'l1'' \Rightarrow bool).$
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') $\leq$
(rel-FGco L1 Co1 Co2 Co3 Co4 Co5 Co6 Contra1 Contra2 Contra3 Contra4 ::
(-, -, -, -, -, -, -, -, -, -, $'f1, 'f2$) *FGco* $\Rightarrow$ -) *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* :: $'co1 \Rightarrow 'co1' \Rightarrow bool$ **and** *Co1'* :: $'co1' \Rightarrow 'co1'' \Rightarrow bool$
and *Co2* :: $'co2 \Rightarrow 'co2' \Rightarrow bool$ **and** *Co2'* :: $'co2' \Rightarrow 'co2'' \Rightarrow bool$
and *Co5* :: $'co5 \Rightarrow 'co5' \Rightarrow bool$ **and** *Co5'* :: $'co5' \Rightarrow 'co5'' \Rightarrow bool$
and *tytok-F* :: $('l1 \times 'l1' \times 'l1'' \times 'co1 \times 'co1' \times 'co1'' \times 'co5 \times 'co5' \times$
$'co5'' \times$
$'f2)$ *itself*
and *tytok-G* :: $('co1 \times 'co1' \times 'co1'' \times 'co2 \times 'co2' \times 'co2'' \times 'f1)$ *itself*
and *tytok-FGco* :: $('l1 \times 'l1' \times 'l1'' \times 'f1 \times 'f2)$ *itself*
assumes *rel-F-pos-distr-cond*
(*rel-G Co1 Co2 Co3 Co4 Contra1 Contra2* :: (-, -, -, -, -, -, $'f1$) *G* $\Rightarrow$ -)
(*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
⟨*proof*⟩

lemma *rel-FGco-neg-distr-imp*:
fixes *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 *Co5* :: $'co5 \Rightarrow 'co5' \Rightarrow bool$ **and** *Co5'* :: $'co5' \Rightarrow 'co5'' \Rightarrow bool$
and *tytok-F* :: $('l1 \times 'l1' \times 'l1'' \times 'co1 \times 'co1' \times 'co1'' \times 'co5 \times 'co5' \times$
$'co5'' \times 'f2)$ *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*
⟨*proof*⟩

lemma *rel-FGco-pos-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGco-pos-distr-cond* (=) *tytok*
⟨*proof*⟩

lemma *rel-FGco-neg-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGco-neg-distr-cond* (=) *tytok*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *bd-FGco-card-order*: *card-order bd-FGco*
⟨*proof*⟩

lemma *bd-FGco-cinfinite*: *cinfinite bd-FGco*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mapl-FGco-cong*:
assumes ⋀*z*. *z* ∈ *set1-FGco x* ⟹ *l1 z* = *l1' z*
shows *mapl-FGco l1 x* = *mapl-FGco l1' x*
⟨*proof*⟩

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*
⟨*proof*⟩

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' Con-

tra4' Contra5'
⟨*proof*⟩

lemma *rel-FGcontra-eq*: *rel-FGcontra* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

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*)$^{-1-1}$
⟨*proof*⟩

lemma *map-FGcontra-id0*: *map-FGcontra id id id id id id id id id id* = *id*
⟨*proof*⟩

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*)
⟨*proof*⟩

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-fun-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
⟨*proof*⟩

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
⟨*proof*⟩

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

⟨*proof*⟩

lemma *rel-FGcontra-pos-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGcontra-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=)
(=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

lemma *rel-FGcontra-neg-distr-cond-eq*:
fixes *tytok* :: (*'l1* × *'l1'* × *'l1''* × *'f1* × *'f2*) *itself*
shows *rel-FGcontra-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) (=) (=)
(=) (=) (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *bd-FGcontra-card-order*: *card-order bd-FGcontra*
⟨*proof*⟩

lemma *bd-FGcontra-cinfinite*: *cinfinite bd-FGcontra*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mapl-FGcontra-contrang*:
assumes $\bigwedge z.\ z \in$ *set1-FGcontra x* $\Longrightarrow$ *l1 z* = *l1' z*
shows *mapl-FGcontra l1 x* = *mapl-FGcontra l1' x*
⟨*proof*⟩

lemma *rell-FGcontra-mono-strong*:
assumes *rell-FGcontra L1 x y*
and $\bigwedge a\ b.\ a \in$ *set1-FGcontra x* $\Longrightarrow$ $b \in$ *set1-FGcontra y* $\Longrightarrow$ *L1 a b* $\Longrightarrow$ *L1' a b*
shows *rell-FGcontra L1' x y*
⟨*proof*⟩

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* ⟧ $\Longrightarrow$
rel-FGf L1 L2 Co1 Co2 Contra1 Contra2 ≤ *rel-FGf L1' L2' Co1' Co2' Contra1' Contra2'*
⟨*proof*⟩

lemma *rel-FGf-eq*: *rel-FGf* (=) (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

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*)$^{-1-1}$
⟨*proof*⟩

lemma *map-FGf-id0*: *map-FGf id id id id id id* = *id*
⟨*proof*⟩

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*)

⟨*proof*⟩

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-fun-FGf L1 L2 Co1 Co2 Contra1 Contra2*)
(*rel-FGf L1' L2' Co1' Co2' Contra1' Contra2'*)))))))
map-FGf map-FGf
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *bd-FGf-card-order*: *card-order bd-FGf*
⟨*proof*⟩

lemma *bd-FGf-cinfinite*: *cinfinite bd-FGf*
⟨*proof*⟩

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*) ≤*o* (*bd-FGf* :: (*'co1*, *'co2*, *'contra1*, *'contra2*,
'f1, *'f2*, *'f3*, *'f4*, *'f5*, *'f6*, *'f7*) *FGfbd rel*)
⟨*proof*⟩

lemma *mapl-FGf-cong*:
assumes ⋀*z*. *z* ∈ *set1-FGf x* ⟹ *l1 z* = *l1' z* **and** ⋀*z*. *z* ∈ *set2-FGf x* ⟹ *l2 z* = *l2' z*
shows *mapl-FGf l1 l2 x* = *mapl-FGf l1' l2' x*
⟨*proof*⟩

lemma *rell-FGf-mono-strong*:
assumes *rell-FGf L1 L2 x y*
and ⋀*a b*. *a* ∈ *set1-FGf x* ⟹ *b* ∈ *set1-FGf y* ⟹ *L1 a b* ⟹ *L1' a b*
and ⋀*a b*. *a* ∈ *set2-FGf x* ⟹ *b* ∈ *set2-FGf y* ⟹ *L2 a b* ⟹ *L2' a b*
shows *rell-FGf L1' L2' x y*
⟨*proof*⟩

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* (=) (=) (=) (=)
⟨*proof*⟩

lemma *mapl-T-alt-def*: *mapl-T l1* = *map-T l1 id id id id*
⟨*proof*⟩

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'*
⟨*proof*⟩

lemma *rel-T-eq*: *rel-T* (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

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*)$^{-1-1}$
⟨*proof*⟩

lemma *map-T-id0*: *map-T id id id id id = id*
⟨*proof*⟩

lemma *map-T-id*: *map-T id id id id id x = x*
⟨*proof*⟩

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)
⟨*proof*⟩

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
⟨*proof*⟩

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* :: $('l1 \Rightarrow 'l1'' \Rightarrow bool) \Rightarrow$
$('co1 \Rightarrow 'co1' \Rightarrow bool) \Rightarrow ('co1' \Rightarrow 'co1'' \Rightarrow bool) \Rightarrow$
$('co2 \Rightarrow 'co2' \Rightarrow bool) \Rightarrow ('co2' \Rightarrow 'co2'' \Rightarrow bool) \Rightarrow$
$('contra1 \Rightarrow 'contra1' \Rightarrow bool) \Rightarrow ('contra1' \Rightarrow 'contra1'' \Rightarrow bool) \Rightarrow$
$('contra2 \Rightarrow 'contra2' \Rightarrow bool) \Rightarrow ('contra2' \Rightarrow 'contra2'' \Rightarrow bool) \Rightarrow$
$('l1, 'co1, 'co2, 'contra1, 'contra2, 'f)\ T \Rightarrow$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f)\ T \Rightarrow$
$('l1 \times 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f)\ T$ **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' Contra2 Contra2' x)) id x,
D-T Cy)))

lemma *rel-T-pos-distr-imp*:
fixes *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-G* :: $('l1, 'co1, 'co2, 'contra1, 'contra2, 'f)\ T \times$
$('l1', 'co1', 'co2', 'contra1', 'contra2', 'f)\ T \times$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f)\ T \times 'l1 \times 'l1' \times 'l1'' \times 'f)$ *itself*
and *tytok-T* :: $('l1 \times 'l1' \times 'l1'' \times 'f)$ *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 Contra2' tytok-T*
⟨*proof*⟩

lemma
fixes *L1* :: $'l1 \Rightarrow 'l1'' \Rightarrow bool$
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-G* :: $((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f)\ T \times$
$(('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f)\ T$
$\Rightarrow ('l1 \times 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f)\ T)) \times$

$(((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T \times$
$(('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T$
$\Rightarrow ('l1 \times 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) \times$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T) \times$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T \times$
$'l1 \times ('l1 \times 'l1'') \times 'l1'' \times 'f) itself$
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* $(\lambda x\ (x', y).\ x' = x \wedge L1\ x\ y)$ *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* $(\lambda(x, y').\ y.\ y' = y \wedge L1\ x\ y)$ *Co1' Co2' Contra1' Contra2'*
(rel-T-witness L1 Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' x y) y
⟨*proof*⟩

lemma *rel-T-neg-distr-imp*:
fixes *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-G* :: $(((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T \times$
$(('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T$
$\Rightarrow ('l1 \times 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) \times$
$(((('l1, 'co1, 'co2, 'contra1, 'contra2, 'f) T \times$
$(('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T$
$\Rightarrow ('l1 \times 'l1'', 'co1', 'co2', 'contra1', 'contra2', 'f) T)) \times$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T) \times$
$('l1'', 'co1'', 'co2'', 'contra1'', 'contra2'', 'f) T \times$
$'l1 \times ('l1 \times 'l1'') \times 'l1'' \times 'f) itself$
and *tytok-T* :: $('l1 \times 'l1' \times 'l1'' \times 'f) itself$
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 Contra2' tytok-T*
⟨*proof*⟩

lemma *rel-T-pos-distr-cond-eq*:
$\bigwedge tytok.$ *rel-T-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

lemma *rel-T-neg-distr-cond-eq*:

⋀*tytok*. *rel-T-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

end

4.2 Greatest fixpoints

4.2.1 $\mathbf{BNF_{CC}}$ 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* (=) (=) (=) (=)
⟨*proof*⟩

lemma *mapl-U-alt-def*: *mapl-U l1* = *map-U l1 id id id id*
⟨*proof*⟩

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'*
⟨*proof*⟩

lemma *rel-U-eq*: *rel-U* (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

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*)$^{-1-1}$
⟨*proof*⟩

lemma *map-U-id0*: *map-U id id id id id* = *id*
⟨*proof*⟩

lemma *map-U-id*: *map-U id id id id id x* = *x*
⟨*proof*⟩

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*)
⟨*proof*⟩

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
⟨*proof*⟩

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 Contra2'*) *id*
(*rel-G-witness* (*rel-U L1* (*Co1 OO Co1'*) (*Co2 OO Co2'*) (*Contra1 OO Contra1'*) (*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 Contra2' tytok-T*
⟨*proof*⟩

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*))
⟨*proof*⟩

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' 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*, *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*
⟨*proof*⟩

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 Contra2' tytok-T*
⟨*proof*⟩

lemma *rel-U-pos-distr-cond-eq*:
⋀*tytok*. *rel-U-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

lemma *rel-U-neg-distr-cond-eq*:
⋀*tytok*. *rel-U-neg-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*) $G.\ P\ x\}$ ⟨*proof*⟩

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 −−−> 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'
⟨proof⟩

lemma rel-S-eq: rel-S (=) (=) (=) (=) (=) (=) = (=)
⟨proof⟩

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)$^{-1-1}$
⟨proof⟩

lemma map-S-id0: map-S id id id id id id = id
⟨proof⟩

lemma map-S-id: map-S id id id id id id x = x
⟨proof⟩

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)
⟨proof⟩

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
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rel-S-pos-distr-cond-eq*:
⋀*tytok*. *rel-S-pos-distr-cond* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

bnf ('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) *S*
map: λ*l1 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: λ*L1 L2*. *rel-S L1 L2* (=) (=) (=) (=)
⟨*proof*⟩

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*
⟨*proof*⟩

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 ⟹
rel-G-neg-distr-cond Co1 Co1' Co2 Co2' Contra1 Contra1' Contra2 Contra2' tytok
and
rel-S-neg-distr-cond'-eq:
⋀*tytok*. *rel-S-neg-distr-cond'* (=) (=) (=) (=) (=) (=) (=) (=) *tytok*

context includes *lifting-syntax*
begin

definition *rel-S-witness'* :: ('*live1* ⇒ '*live1''* ⇒ *bool*) ⇒ ('*live2* ⇒ '*live2''* ⇒ *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*) ⇒
('*live1*, '*live2*, '*co1*, '*co2*, '*contra1*, '*contra2*, '*fixed*) *S* ×
('*live1''*, '*live2''*, '*co1''*, '*co2''*, '*contra1''*, '*contra2''*, '*fixed*) *S* ⇒
('*live1* × '*live1''*, '*live2* × '*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* ⇒ '*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 Con-*

tra2' tytok
shows *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*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*: ⋀*z* :: (*'a* × *'c*) *H*. ⟦ *Q* (*map-H fst z*); *Q* (*map-H snd z*) ⟧ ⟹ *Q z*
and *Q x* **and** *Q y* **and** *rel-H L x y*
and *related*: *rel-H* (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L x y*) *x z* *rel-H* (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L x y*) *z y*
shows *Q z*
⟨*proof*⟩

lemma *Q-zip*:
fixes *z* :: (*'a* × *'c*) *H*
assumes *Q-rel-H-zipping*: ⋀(*L* :: *'a* ⇒ *'c* ⇒ -) *x y z*.
⟦ *Q x*; *Q y*; *rel-H L x y*; *rel-H* (λ*x* (*x'*, *y*). *x'* = *x* ∧ *L x y*) *x z*;
rel-H (λ(*x*, *y'*) *y*. *y'* = *y* ∧ *L x y*) *z y* ⟧ ⟹ *Q z*
and *Q* (*map-H fst z*) **and** *Q* (*map-H snd z*)
shows *Q z*
⟨*proof*⟩

lemma *Q-neg-distr*:
fixes *x* :: *'a H* **and** *y* :: *'c H*
assumes *Q-zip-weak*: ⋀*z* :: (*'a* × *'c*) *H*. ⟦ *Q* (*map-H fst z*); *Q* (*map-H snd z*) ⟧ ⟹
∃*z'*. *Q z'* ∧ *set-H z'* ⊆ *set-H z* ∧ *map-H fst z'* = *map-H fst z* ∧ *map-H snd z'* = *map-H snd z*
and *Q x* **and** *Q y* **and** *related*: *rel-H* (*L OO L'*) *x y*
shows (*rel-H L OO eq-onp Q OO rel-H L'*) *x y*
⟨*proof*⟩

lemma *Q-zip-weak*:
fixes *z* :: (*'a* × *'c*) *H*
assumes *Q-neg-distr*: ⋀(*L* :: *'a* ⇒ (*'a* × *'c*) ⇒ -) (*L'* :: (*'a* × *'c*) ⇒ *'c* ⇒ *bool*) *x y*.
⟦ *Q x*; *Q y*; *rel-H* (*L OO L'*) *x y* ⟧ ⟹ (*rel-H L OO eq-onp Q OO rel-H L'*) *x y*
and *Q* (*map-H fst z*) **and** *Q* (*map-H snd z*)
obtains *z'* **where** *Q z'* **and** *set-H z'* ⊆ *set-H z*
and *map-H fst z'* = *map-H fst z* **and** *map-H snd z'* = *map-H snd z*
⟨*proof*⟩

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* ⇒ -)
⟨*proof*⟩

end

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

end

7 Concrete BNF$_{\mathbf{CC}}$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*) ≤ (*A'* ===> *B'*) **if** *A'* ≤ *A* *B* ≤ *B'*

⟨*proof*⟩

lemma *rel-fun-eq*: $((=) ===> (=)) = (=)$ ⟨*proof*⟩

lemma *rel-fun-conversep*: $(A^{-1-1} ===> B^{-1-1}) = (A ===> B)^{-1-1}$ ⟨*proof*⟩

lemma *map-fun-id0*: $(id ---> id) = id$ ⟨*proof*⟩

lemma *map-fun-comp*: $(f ---> g) \circ (f' ---> g') = ((f' \circ f) ---> (g \circ g'))$
⟨*proof*⟩

lemma *map-fun-parametric*: $((A ===> A') ===> (B ===> B') ===> (A' ===> B) ===> (A ===> B')) (--->) (--->)$
⟨*proof*⟩

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' - $\longleftrightarrow$ $(\forall (B :: 'b \Rightarrow 'b' \Rightarrow bool)\ (B' :: 'b' \Rightarrow 'b'' \Rightarrow bool).$
$(A ===> B)\ OO\ (A' ===> B') \leq (A\ OO\ A') ===> (B\ OO\ B'))$

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' - $\longleftrightarrow$ $(\forall (B :: 'b \Rightarrow 'b' \Rightarrow bool)\ (B' :: 'b' \Rightarrow 'b'' \Rightarrow bool).$
$(A\ OO\ A') ===> (B\ OO\ B') \leq (A ===> B)\ OO\ (A' ===> B'))$

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*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *rel-fun-pos-distr-cond-eq*: *rel-fun-pos-distr-cond* (=) (=) *tytok*
⟨*proof*⟩

lemma *rel-fun-neg-distr-cond-eq*: *rel-fun-neg-distr-cond* (=) (=) *tytok*
⟨*proof*⟩

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' ⟨*proof*⟩

lemma *rel-set-eq*: *rel-set* $(=) = (=)$ ⟨*proof*⟩

lemma *rel-set-conversep*: *rel-set* $A^{-1-1} =$ (*rel-set* $A)^{-1-1}$ ⟨*proof*⟩

lemma *map-set-id0*: *image id* $=$ *id* ⟨*proof*⟩

lemma *map-set-comp*: *image* $f \circ$ *image* $g =$ *image* $(f \circ g)$ ⟨*proof*⟩

lemma *map-set-parametric*: **includes** *lifting-syntax* **shows**
$((A ===> B) ===>$ *rel-set* $A ===>$ *rel-set* $B)$ *image image*
⟨*proof*⟩

definition *rel-set-pos-distr-cond* :: $('a \Rightarrow 'a' \Rightarrow$ *bool*$) \Rightarrow ('a' \Rightarrow 'a'' \Rightarrow$ *bool*$) \Rightarrow$ *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* :: $('a \Rightarrow 'a' \Rightarrow$ *bool*$) \Rightarrow ('a' \Rightarrow 'a'' \Rightarrow$ *bool*$) \Rightarrow$ *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*
⟨*proof*⟩

lemma *rel-set-neg-distr-iff* [*simp*]: *rel-set-neg-distr-cond* $A\ A' =$ *True*
⟨*proof*⟩

lemma *rel-set-pos-distr-eq*: *rel-set-pos-distr-cond* $(=)$ $(=)$
⟨*proof*⟩

lemma *rel-set-neg-distr-eq*: *rel-set-neg-distr-cond* $(=)$ $(=)$
⟨*proof*⟩

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* $A \wedge$ *card* $A \leq$ *CARD*$('k)\}$
⟨*proof*⟩

setup-lifting *type-definition-bset*

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

lift-definition *map-bset* :: ($'a \Rightarrow 'b$) $\Rightarrow$ ($'a$, $'k$) *bset* $\Rightarrow$ ($'b$, $'k$) *bset* **is** *image*
⟨*proof*⟩

lift-definition *rel-bset* :: ($'a \Rightarrow 'b \Rightarrow$ *bool*) $\Rightarrow$ ($'a$, $'k$) *bset* $\Rightarrow$ ($'b$, $'k$) *bset* $\Rightarrow$ *bool*
is *rel-set* ⟨*proof*⟩

definition *neg-distr-cond-bset* :: ($'a \Rightarrow 'b \Rightarrow$ *bool*) $\Rightarrow$ ($'b \Rightarrow 'c \Rightarrow$ *bool*) $\Rightarrow$ $'k$ *itself* $\Rightarrow$ *bool* **where**
neg-distr-cond-bset C C' - $\longleftrightarrow$ *rel-bset* (C *OO* C') $\leq$ *rel-bset* C *OO* (*rel-bset* C' :: (-, $'k$) *bset* $\Rightarrow$ -)

lemma *right-unique-rel-set-lemma*:
assumes *right-unique* R **and** *rel-set* R X Y
obtains f **where** $Y =$ *image* f X **and** $\forall x{\in}X.\ R\ x\ (f\ x)$
⟨*proof*⟩

lemma *left-unique-rel-set-lemma*:
assumes *left-unique* R **and** *rel-set* R Y X
obtains f **where** $Y =$ *image* f X **and** $\forall x{\in}X.\ R\ (f\ x)\ x$
⟨*proof*⟩

lemma *neg-distr-cond-bset-right-unique*:
right-unique $C \Longrightarrow$ *neg-distr-cond-bset* C D *tytok*
⟨*proof*⟩

lemma *neg-distr-cond-bset-left-unique*:
left-unique $D \Longrightarrow$ *neg-distr-cond-bset* C D *tytok*
⟨*proof*⟩

lemma *neg-distr-cond-bset-eq*: *neg-distr-cond-bset* (=) (=) *tytok*
⟨*proof*⟩

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* ===> (⟷))

lemma *rel-pred-mono*: *A′* ≤ *A* ⟹ *rel-pred A* ≤ *rel-pred A′* ⟨*proof*⟩

lemma *rel-pred-eq*: *rel-pred* (=) = (=)
⟨*proof*⟩

lemma *rel-pred-conversep*: *rel-pred* A^{-1-1} = (*rel-pred A*)$^{-1-1}$
⟨*proof*⟩

lemma *map-pred-id0*: *map-pred id* = *id*
⟨*proof*⟩

lemma *map-pred-comp*: *map-pred f* ∘ *map-pred g* = *map-pred* (*g* ∘ *f*)
⟨*proof*⟩

lemma *map-pred-parametric*: ((*A′* ===> *A*) ===> *rel-pred A* ===> *rel-pred A′*) *map-pred map-pred*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rel-pred-pos-distr-cond-eq*: *rel-pred-pos-distr-cond* (=) (=)
⟨*proof*⟩

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′*
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *rel-pred-neg-distr-cond-eq*: *rel-pred-neg-distr-cond* (=) (=)
⟨*proof*⟩

lemma *left-unique-rel-pred*: *left-total A* ⟹ *left-unique* (*rel-pred A*)
⟨*proof*⟩

lemma *right-unique-rel-pred*: *right-total A* ⟹ *right-unique* (*rel-pred A*)
⟨*proof*⟩

lemma *left-total-rel-pred*: *left-unique A* ⟹ *left-total* (*rel-pred A*)
⟨*proof*⟩

lemma *right-total-rel-pred*: *right-unique A* ⟹ *right-total* (*rel-pred A*)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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

context includes *lifting-syntax*
begin

definition *rel-filter-neg-distr-cond'* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow$ *bool* **where**
rel-filter-neg-distr-cond' C C' $\longleftrightarrow$ *left-total C* $\wedge$ *right-unique C* $\vee$ *right-total C'* $\wedge$ *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'*
⟨*proof*⟩

lemma *rel-filter-neg-distr-cond'-eq*: *rel-filter-neg-distr-cond'* (=) (=)
⟨*proof*⟩

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*⟩

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* = (λ*P*. *F* (λ*x*. *P* (*f x*)))
⟨*proof*⟩

lemma *filtermap f* = *Abs-filter* ∘ *raw-filtermap f* ∘ *Rep-filter*
⟨*proof*⟩

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

abbreviation *Z'* ≡ (λ*P*. *Collect P* ∈ *Z*)

lemma *is-filter* (*raw-filtermap fst Z'*)
⟨*proof*⟩

lemma *is-filter* (*raw-filtermap snd Z'*)
⟨*proof*⟩

lemma ¬ *is-filter Z'*
⟨*proof*⟩

end

7.6 Almost-everywhere equal sequences

inductive *aeseq-eq* :: (*nat* ⇒ *'a*) ⇒ (*nat* ⇒ *'a*) ⇒ *bool* **for** *f g* **where**
aeseq-eq f g **if** *finite* {*n*. *f n* ≠ *g n*}

lemma *equivp-aeseq-eq*: *equivp aeseq-eq*
⟨*proof*⟩

quotient-type *'a aeseq* = *nat* ⇒ *'a* / *aeseq-eq* ⟨*proof*⟩

lift-definition *map-aeseq* :: (*'a* ⇒ *'b*) ⇒ *'a aeseq* ⇒ *'b aeseq* **is** (∘)
⟨*proof*⟩

lemma *map-aeseq-id*: *map-aeseq id x* = *x*
⟨*proof*⟩

lemma *map-aeseq-comp*: *map-aeseq f* (*map-aeseq g x*) = *map-aeseq* (*f* ∘ *g*) *x*
⟨*proof*⟩

lift-definition *rel-aeseq* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *'a aeseq* ⇒ *'b aeseq* ⇒ *bool* **is**
λR f g. *finite* {*n*. ¬ *R* (*f n*) (*g n*)}
⟨*proof*⟩

lemma *rel-aeseq-mono*: *R* ≤ *S* ⟹ *rel-aeseq R* ≤ *rel-aeseq S*
⟨*proof*⟩

lemma *rel-aeseq-eq*: *rel-aeseq* (=) = (=)
⟨*proof*⟩

lemma *rel-aeseq-conversep*: *rel-aeseq* R^{-1-1} = (*rel-aeseq R*)$^{-1-1}$
⟨*proof*⟩

lemma *map-aeseq-parametric*: **includes** *lifting-syntax* **shows**
(*A* ===> *B*) ===> *rel-aeseq A* ===> *rel-aeseq B*) *map-aeseq map-aeseq*
⟨*proof*⟩

lemma *rel-aeseq-distr*: *rel-aeseq* (*R OO S*) = *rel-aeseq R OO rel-aeseq S*
⟨*proof*⟩

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*) = $(\lambda a.$ *map-prod g* (*map-dds f g*) (*run S* (*f a*)))

lemma *map-dds-id*: *map-dds id id S* = *S*
⟨*proof*⟩

lemma *map-dds-comp*: *map-dds f g* (*map-dds f' g' S*) = *map-dds* $(f' \circ f)$ $(g \circ g')$ *S*
⟨*proof*⟩

coinductive *rel-dds* :: $('a \Rightarrow 'a' \Rightarrow$ *bool*$) \Rightarrow ('b \Rightarrow 'b' \Rightarrow$ *bool*$) \Rightarrow ('a, 'b)$ *dds* $\Rightarrow ('a', 'b')$ *dds* $\Rightarrow$ *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*
⟨*proof*⟩

lemma *rel-dds-eq* [*relator-eq*]: *rel-dds* (=) (=) = (=)
⟨*proof*⟩

lemma *rel-dds-mono* [*relator-mono*]: *rel-dds A B* ≤ *rel-dds A' B'* **if** $A' \leq A$ $B \leq B'$
⟨*proof*⟩

lemma *rel-dds-conversep*: *rel-dds* A^{-1-1} B^{-1-1} = (*rel-dds A B*)$^{-1-1}$
⟨*proof*⟩

lemma *DDS-parametric* [*transfer-rule*]:
((*A* ===> *rel-prod B* (*rel-dds A B*)) ===> *rel-dds A B*) *DDS DDS*
⟨*proof*⟩

lemma *run-parametric* [*transfer-rule*]:

(*rel-dds A B ===> A ===> rel-prod B (rel-dds A B)*) *run run*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *map-dds-parametric* [*transfer-rule*]:
((*A′ ===> A*) *===> (B ===> B′) ===> rel-dds A B ===> rel-dds A′ B′*) *map-dds map-dds*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *rel-dds-pos-distr* [*relator-distr*]:
rel-dds A B OO rel-dds C D ≤ *rel-dds* (*A OO C*) (*B OO D*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

8.2 Evenness of partial sums

definition *even-psum* :: (*int*, *bool*) *dds* **where**
even-psum = *dds-of* (*λpsum 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*) (·) (·)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)) (‖) (‖)
⟨*proof*⟩

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*)
⟨*proof*⟩

8.4 Graph traversal: refinement and quotients

lemma *finite-Image*:
finite A ⟹ *finite* (*R* `` *A*) ⟷ (∀ *x*∈*A*. *finite* {*y*. (*x*, *y*) ∈ *R*})
⟨*proof*⟩

context includes *fset.lifting* **begin**
lift-definition *fImage* :: ('*a* × '*b*) *fset* ⇒ '*a fset* ⇒ '*b fset* **is** *Image* **parametric** *Image-parametric*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *fset-of-list-parametric* [*transfer-rule*]:
(*list-all2 A* ===> *rel-fset A*) *fset-of-list fset-of-list*
including *fset.lifting* ⟨*proof*⟩

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*
⟨*proof*⟩

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* (=)
⟨*proof*⟩

lift-definition *traverse''* :: (*'a* × *'a*) *fset'* ⇒ (*'a fset'*, *'a fset'*) *dds*
is *traverse-impl* :: *'a graph'* ⇒ - **parametric** *traverse-impl-parametric*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *traverse-mono*: ((|⊆|) ===> *rel-dds* (=) (|⊆|)) *traverse traverse*
⟨*proof*⟩

lemma
assumes *G* |⊆| *H*
shows *rel-dds* (=) (|⊆|) (*traverse G* · *accumulate*) (*traverse H* · *accumulate*)
⟨*proof*⟩

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

lemma *seen-mono*: *rel-dds* (|⊆|) (|⊆|) *seen seen*
⟨*proof*⟩

lemma
assumes *G* |⊆| *H*
shows *rel-dds* (=) (|⊆|) (*traverse G* · *seen*) (*traverse H* · *seen*)
⟨*proof*⟩

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.