An Algebraic Approach to Multirelations and their Properties

Rudolf Berghammer, Christian-Albrechts-Universität zu Kiel
Walter Guttmann, University of Canterbury

2016.01.29

theory *AAMP*

imports *Main*

begin

— This theory and the subsequent theories have been developed with Isabelle2015.

class *mult* = *times*

begin

notation
  times (**infixl** · *70*) **and**
  times (**infixl** ; *70*)

end

class *neg* = *uminus*

begin

no-notation
  uminus (− - [*81*] *80*)

notation
  uminus (− - [*80*] *80*)

end

class *while* =
  fixes *while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\star$ *59*)

class *L* =
  fixes *L* :: $'a$

class *n* =
  fixes *n* :: $'a \Rightarrow 'a$

class *d* =
  fixes *d* :: $'a \Rightarrow 'a$

class *diamond* =
  fixes *diamond* :: $'a \Rightarrow 'a \Rightarrow 'a$ (| - > - [*50*,*90*] *95*)

class *box* =
  fixes *box* :: $'a \Rightarrow 'a \Rightarrow 'a$ (| -] - [*50*,*90*] *95*)

context *ord*

begin

definition *isotone* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
  where *isotone* $f \leftrightarrow (\forall x\ y\ .\ x \leq y \rightarrow f(x) \leq f(y))$

definition *lifted-less-eq* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow$ *bool* ((- $\leq\leq$ -) [*51*, *51*] *50*)
  where $f \leq\leq g \leftrightarrow (\forall x\ .\ f(x) \leq g(x))$

definition *galois* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow$ *bool*
  where *galois* $l\ u \leftrightarrow (\forall x\ y\ .\ l(x) \leq y \leftrightarrow x \leq u(y))$

definition *ascending-chain* :: $(nat \Rightarrow 'a) \Rightarrow$ *bool*
  where *ascending-chain* $f \leftrightarrow (\forall n\ .\ f\ n \leq f\ (Suc\ n))$

definition *descending-chain* :: $(nat \Rightarrow 'a) \Rightarrow$ *bool*
  where *descending-chain* $f \leftrightarrow (\forall n\ .\ f\ (Suc\ n) \leq f\ n)$

definition *directed* :: $'a\ set \Rightarrow$ *bool*
  where *directed* $X \leftrightarrow X \neq \{\} \wedge (\forall x{\in}X\ .\ \forall y{\in}X\ .\ \exists z{\in}X\ .\ x \leq z \wedge y \leq z)$

definition *codirected* :: $'a\ set \Rightarrow$ *bool*
  where *codirected* $X \leftrightarrow X \neq \{\} \wedge (\forall x{\in}X\ .\ \forall y{\in}X\ .\ \exists z{\in}X\ .\ z \leq x \wedge z \leq y)$

definition *chain* :: $'a\ set \Rightarrow$ *bool*
  where *chain* $X \leftrightarrow (\forall x{\in}X\ .\ \forall y{\in}X\ .\ x \leq y \vee y \leq x)$

end

context *order*

begin

lemma *lifted-reflexive*: $f = g \longrightarrow f \leq\leq g$
  by (*metis lifted-less-eq-def order-refl*)

lemma *lifted-transitive*: $f \leq\leq g \wedge g \leq\leq h \longrightarrow f \leq\leq h$
  by (*smt lifted-less-eq-def order-trans*)

lemma *lifted-antisymmetric*: $f \leq\leq g \wedge g \leq\leq f \longrightarrow f = g$
  by (*metis* (*full-types*) *antisym ext lifted-less-eq-def*)

lemma *galois-char*: *galois* $l\ u \leftrightarrow (\forall x\ .\ x \leq u(l(x))) \wedge (\forall x\ .\ l(u(x)) \leq x) \wedge$ *isotone* $l \wedge$ *isotone* u
  apply (*rule iffI*)
  apply (*metis* (*full-types*) *galois-def isotone-def order-refl order-trans*)
  apply (*metis galois-def isotone-def order-trans*)
  done

lemma *galois-closure*: *galois* $l\ u \longrightarrow l(x) = l(u(l(x))) \wedge u(x) = u(l(u(x)))$
  by (*simp add*: *galois-char isotone-def order.antisym*)

lemma *ascending-chain-k*: *ascending-chain* $f \longrightarrow f\ m \leq f\ (m + k)$
  apply (*induct k*)
  apply *simp*
  apply (*metis add-Suc-right ascending-chain-def order-trans*)
  done

lemma *ascending-chain-isotone*: *ascending-chain* $f \wedge m \leq k \longrightarrow f\ m \leq f\ k$
  by (*metis ascending-chain-k le-iff-add*)

lemma *ascending-chain-comparable*: *ascending-chain* $f \longrightarrow f\ k \leq f\ m \vee f\ m \leq f\ k$
  by (*metis nat-le-linear ascending-chain-isotone*)

lemma *ascending-chain-chain*: *ascending-chain* $f \longrightarrow$ *chain* (*range* f)
  by (*smt ascending-chain-comparable chain-def image-iff*)

lemma *chain-directed*: $X \neq \{\} \wedge$ *chain* $X \longrightarrow$ *directed* X
  by (*metis chain-def directed-def*)

lemma *ascending-chain-directed*: *ascending-chain* $f \longrightarrow$ *directed* (*range* f)
  by (*metis UNIV-not-empty ascending-chain-chain chain-directed empty-is-image*)

lemma *descending-chain-k*: *descending-chain* $f \longrightarrow f\ (m + k) \leq f\ m$
  apply (*induct k*)
  apply *simp*

apply (*metis add-Suc-right descending-chain-def order-trans*)
done

lemma *descending-chain-antitone*: *descending-chain f* $\wedge$ $m \leq k \longrightarrow f\ k \leq f\ m$
by (*metis descending-chain-k le-iff-add*)

lemma *descending-chain-comparable*: *descending-chain f* $\longrightarrow f\ k \leq f\ m \vee f\ m \leq f\ k$
by (*metis nat-le-linear descending-chain-antitone*)

lemma *descending-chain-chain*: *descending-chain f* $\longrightarrow$ *chain* (*range f*)
unfolding *chain-def*
apply *simp*
apply (*smt descending-chain-comparable*)
done

lemma *chain-codirected*: $X \neq \{\}$ $\wedge$ *chain X* $\longrightarrow$ *codirected X*
by (*metis chain-def codirected-def*)

lemma *descending-chain-codirected*: *descending-chain f* $\longrightarrow$ *codirected* (*range f*)
by (*metis UNIV-not-empty descending-chain-chain chain-codirected empty-is-image*)

end

context *complete-lattice*

begin

lemma *sup-Sup*: **assumes** *nonempty*: $A \neq \{\}$
shows *sup x* (*Sup A*) = *Sup* ((*sup x*) ‘ *A*)
apply (*rule antisym*)
apply (*metis Sup-mono Sup-upper2 assms ex-in-conv imageI le-supI sup-ge1 sup-ge2*)
apply (*smt Sup-least Sup-upper image-iff le-iff-sup sup.commute sup-ge1 sup-left-commute*)
done

lemma *sup-SUP*: $Y \neq \{\} \longrightarrow$ *sup x* (*SUP y*:*Y* . *f y*) = (*SUP y*:*Y*. *sup x* (*f y*))
unfolding *SUP-def*
apply *rule*
apply (*subst sup-Sup*)
apply (*smt empty-is-image*)
apply (*metis image-image*)
done

lemma *inf-Inf*: **assumes** *nonempty*: $A \neq \{\}$
shows *inf x* (*Inf A*) = *Inf* ((*inf x*) ‘ *A*)
apply (*rule antisym*)
apply (*smt Inf-greatest Inf-lower image-iff le-iff-inf inf.commute inf-le1 inf-left-commute*)
apply (*metis Inf-mono Inf-lower2 assms ex-in-conv imageI le-infI inf-le1 inf-le2*)

done

lemma *inf-INF*: $Y \neq \{\} \rightarrow inf\ x\ (INF\ y{:}Y\ .\ f\ y) = (INF\ y{:}Y.\ inf\ x\ (f\ y))$
unfolding *INF-def*
apply *rule*
apply (*subst inf-Inf*)
apply (*smt empty-is-image*)
apply (*metis image-image*)
done

lemma *SUP-image-id*[*simp*]: $(SUP\ x{:}f`A\ .\ x) = (SUP\ x{:}A\ .\ f\ x)$
by *simp*

lemma *INF-image-id*[*simp*]: $(INF\ x{:}f`A\ .\ x) = (INF\ x{:}A\ .\ f\ x)$
by *simp*

end

lemma *image-Collect-2*: $f\ `\ \{\ g\ x\ |\ x\ .\ P\ x\ \} = \{\ f\ (g\ x)\ |\ x\ .\ P\ x\ \}$
by *auto*

— The following instantiation and four lemmas are from Jose Divason Mallagaray.

instantiation *fun* :: (*type*, *type*) *power*

begin

definition *one-fun* :: $'a \Rightarrow 'a$
where *one-fun-def*: *one-fun* $\equiv$ *id*

definition *times-fun* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a)$
where *times-fun-def*: *times-fun* $\equiv$ *comp*

instance
by *intro-classes*

end

lemma *id-power*: *id^m* = *id*
apply (*induct m*)
apply (*metis one-fun-def power-0*)
apply (*simp add*: *times-fun-def*)
done

lemma *power-zero-id*: *f^0* = *id*
by (*metis one-fun-def power-0*)

lemma *power-succ-unfold*: $f\hat{}Suc\ m = f \circ f\hat{}m$
by (*metis power-Suc times-fun-def*)

lemma *power-succ-unfold-ext*: $(f\hat{}Suc\ m)\ x = f\ ((f\hat{}m)\ x)$
by (*metis o-apply power-succ-unfold*)

context *order*

begin

definition *is-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-fixpoint* $f\ x \leftrightarrow f(x) = x$
definition *is-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-prefixpoint* $f\ x \leftrightarrow f(x) \leq x$
definition *is-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-postfixpoint* $f\ x \leftrightarrow f(x) \geq x$
definition *is-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-least-fixpoint* $f\ x \leftrightarrow f(x) = x \wedge (\forall\, y\ .\ f(y) = y \rightarrow x \leq y)$
definition *is-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-greatest-fixpoint* $f\ x \leftrightarrow f(x) = x \wedge (\forall\, y\ .\ f(y) = y \rightarrow x \geq y)$
definition *is-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-least-prefixpoint* $f\ x \leftrightarrow f(x) \leq x \wedge (\forall\, y\ .\ f(y) \leq y \rightarrow x \leq y)$
definition *is-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-greatest-postfixpoint* $f\ x \leftrightarrow f(x) \geq x \wedge (\forall\, y\ .\ f(y) \geq y \rightarrow x \geq y)$
definition *has-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-fixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}fixpoint\ f\ x)$
definition *has-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-prefixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}prefixpoint\ f\ x)$
definition *has-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-postfixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}postfixpoint\ f\ x)$
definition *has-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-least-fixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}least\text{-}fixpoint\ f\ x)$
definition *has-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-greatest-fixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}greatest\text{-}fixpoint\ f\ x)$
definition *has-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-least-prefixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}least\text{-}prefixpoint\ f\ x)$
definition *has-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-greatest-postfixpoint* $f \leftrightarrow (\exists\, x\ .\ is\text{-}greatest\text{-}postfixpoint\ f\ x)$
definition *the-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a\ (\mu\ -\ [201]\ 200)$ **where** $\mu\ \ f = (THE\ x\ .\ is\text{-}least\text{-}fixpoint\ f\ x)$
definition *the-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a\ (\nu\ -\ [201]\ 200)$ **where** $\nu\ \ f = (THE\ x\ .\ is\text{-}greatest\text{-}fixpoint\ f\ x)$
definition *the-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a\ (p\mu\ -\ [201]\ 200)$ **where** $p\mu\ f = (THE\ x\ .\ is\text{-}least\text{-}prefixpoint\ f\ x)$
definition *the-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a\ (p\nu\ -\ [201]\ 200)$ **where** $p\nu\ f = (THE\ x\ .\ is\text{-}greatest\text{-}postfixpoint\ f\ x)$

lemma *least-fixpoint-unique*: $has\text{-}least\text{-}fixpoint\ f \rightarrow (\exists!x\ .\ is\text{-}least\text{-}fixpoint\ f\ x)$
by (*smt antisym has-least-fixpoint-def is-least-fixpoint-def*)

lemma *greatest-fixpoint-unique*: $has\text{-}greatest\text{-}fixpoint\ f \rightarrow (\exists!x\ .\ is\text{-}greatest\text{-}fixpoint\ f\ x)$
by (*smt antisym has-greatest-fixpoint-def is-greatest-fixpoint-def*)

lemma *least-prefixpoint-unique*: $has\text{-}least\text{-}prefixpoint\ f \rightarrow (\exists!x\ .\ is\text{-}least\text{-}prefixpoint\ f\ x)$
by (*smt antisym has-least-prefixpoint-def is-least-prefixpoint-def*)

lemma *greatest-postfixpoint-unique*: $has\text{-}greatest\text{-}postfixpoint\ f \rightarrow (\exists!x\ .\ is\text{-}greatest\text{-}postfixpoint\ f\ x)$
by (*smt antisym has-greatest-postfixpoint-def is-greatest-postfixpoint-def*)

lemma *least-fixpoint*: $has\text{-}least\text{-}fixpoint\ f \rightarrow is\text{-}least\text{-}fixpoint\ f\ (\mu\ f)$
proof
assume *has-least-fixpoint f*
hence $is\text{-}least\text{-}fixpoint\ f\ (THE\ x\ .\ is\text{-}least\text{-}fixpoint\ f\ x)$

by (*smt least-fixpoint-unique theI'*)
thus *is-least-fixpoint f* (*μ f*)
by (*simp add*: *is-least-fixpoint-def the-least-fixpoint-def*)
qed

lemma *greatest-fixpoint*: *has-greatest-fixpoint f* ⟶ *is-greatest-fixpoint f* (*ν f*)
proof
assume *has-greatest-fixpoint f*
hence *is-greatest-fixpoint f* (*THE x . is-greatest-fixpoint f x*)
by (*smt greatest-fixpoint-unique theI'*)
thus *is-greatest-fixpoint f* (*ν f*)
by (*simp add*: *is-greatest-fixpoint-def the-greatest-fixpoint-def*)
qed

lemma *least-prefixpoint*: *has-least-prefixpoint f* ⟶ *is-least-prefixpoint f* (*pμ f*)
proof
assume *has-least-prefixpoint f*
hence *is-least-prefixpoint f* (*THE x . is-least-prefixpoint f x*)
by (*smt least-prefixpoint-unique theI'*)
thus *is-least-prefixpoint f* (*pμ f*)
by (*simp add*: *is-least-prefixpoint-def the-least-prefixpoint-def*)
qed

lemma *greatest-postfixpoint*: *has-greatest-postfixpoint f* ⟶ *is-greatest-postfixpoint f* (*pν f*)
proof
assume *has-greatest-postfixpoint f*
hence *is-greatest-postfixpoint f* (*THE x . is-greatest-postfixpoint f x*)
by (*smt greatest-postfixpoint-unique theI'*)
thus *is-greatest-postfixpoint f* (*pν f*)
by (*simp add*: *is-greatest-postfixpoint-def the-greatest-postfixpoint-def*)
qed

lemma *least-fixpoint-same*: *is-least-fixpoint f x* ⟶ *x* = *μ f*
by (*metis least-fixpoint least-fixpoint-unique has-least-fixpoint-def*)

lemma *greatest-fixpoint-same*: *is-greatest-fixpoint f x* ⟶ *x* = *ν f*
by (*metis greatest-fixpoint greatest-fixpoint-unique has-greatest-fixpoint-def*)

lemma *least-prefixpoint-same*: *is-least-prefixpoint f x* ⟶ *x* = *pμ f*
by (*metis least-prefixpoint least-prefixpoint-unique has-least-prefixpoint-def*)

lemma *greatest-postfixpoint-same*: *is-greatest-postfixpoint f x* ⟶ *x* = *pν f*
by (*metis greatest-postfixpoint greatest-postfixpoint-unique has-greatest-postfixpoint-def*)

lemma *least-fixpoint-char*: *is-least-fixpoint f x* ⟷ *has-least-fixpoint f* ∧ *x* = *μ f*
by (*metis least-fixpoint-same has-least-fixpoint-def*)

lemma *least-prefixpoint-char*: *is-least-prefixpoint f x* $\leftrightarrow$ *has-least-prefixpoint f* $\wedge$ $x = p\mu\ f$
by (*metis least-prefixpoint-same has-least-prefixpoint-def*)

lemma *greatest-fixpoint-char*: *is-greatest-fixpoint f x* $\leftrightarrow$ *has-greatest-fixpoint f* $\wedge$ $x = \nu\ f$
by (*metis greatest-fixpoint-same has-greatest-fixpoint-def*)

lemma *greatest-postfixpoint-char*: *is-greatest-postfixpoint f x* $\leftrightarrow$ *has-greatest-postfixpoint f* $\wedge$ $x = p\nu\ f$
by (*metis greatest-postfixpoint-same has-greatest-postfixpoint-def*)

lemma *mu-unfold*: *has-least-fixpoint f* $\rightarrow$ $f\ (\mu\ f) = \mu\ f$
by (*metis is-least-fixpoint-def least-fixpoint*)

lemma *pmu-unfold*: *has-least-prefixpoint f* $\rightarrow$ $f\ (p\mu\ f) \leq p\mu\ f$
by (*metis is-least-prefixpoint-def least-prefixpoint*)

lemma *nu-unfold*: *has-greatest-fixpoint f* $\rightarrow$ $\nu\ f = f\ (\nu\ f)$
by (*metis is-greatest-fixpoint-def greatest-fixpoint*)

lemma *pnu-unfold*: *has-greatest-postfixpoint f* $\rightarrow$ $p\nu\ f \leq f\ (p\nu\ f)$
by (*metis is-greatest-postfixpoint-def greatest-postfixpoint*)

lemma *least-prefixpoint-fixpoint*: *has-least-prefixpoint f* $\wedge$ *isotone f* $\rightarrow$ *is-least-fixpoint f* $(p\mu\ f)$
by (*smt eq-iff is-least-fixpoint-def is-least-prefixpoint-def isotone-def least-prefixpoint*)

lemma *pmu-mu*: *has-least-prefixpoint f* $\wedge$ *isotone f* $\rightarrow$ $p\mu\ f = \mu\ f$
by (*smt has-least-fixpoint-def is-least-fixpoint-def least-fixpoint-unique least-prefixpoint-fixpoint least-fixpoint*)

lemma *greatest-postfixpoint-fixpoint*: *has-greatest-postfixpoint f* $\wedge$ *isotone f* $\rightarrow$ *is-greatest-fixpoint f* $(p\nu\ f)$
by (*smt eq-iff is-greatest-fixpoint-def is-greatest-postfixpoint-def isotone-def greatest-postfixpoint*)

lemma *pnu-nu*: *has-greatest-postfixpoint f* $\wedge$ *isotone f* $\rightarrow$ $p\nu\ f = \nu\ f$
by (*smt has-greatest-fixpoint-def is-greatest-fixpoint-def greatest-fixpoint-unique greatest-postfixpoint-fixpoint greatest-fixpoint*)

lemma *pmu-isotone*: *has-least-prefixpoint f* $\wedge$ *has-least-prefixpoint g* $\wedge$ $f \leq\leq g \rightarrow p\mu\ f \leq p\mu\ g$
by (*smt is-least-prefixpoint-def least-prefixpoint lifted-less-eq-def order-trans*)

lemma *mu-isotone*: *has-least-prefixpoint f* $\wedge$ *has-least-prefixpoint g* $\wedge$ *isotone f* $\wedge$ *isotone g* $\wedge$ $f \leq\leq g \rightarrow \mu\ f \leq \mu\ g$
by (*metis pmu-isotone pmu-mu*)

lemma *pnu-isotone*: *has-greatest-postfixpoint f* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ $f \leq\leq g \rightarrow p\nu\ f \leq p\nu\ g$
by (*smt is-greatest-postfixpoint-def lifted-less-eq-def order-trans greatest-postfixpoint*)

lemma *nu-isotone*: *has-greatest-postfixpoint f* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *isotone f* $\wedge$ *isotone g* $\wedge$ $f \leq\leq g \rightarrow \nu\ f \leq \nu\ g$
by (*metis pnu-isotone pnu-nu*)

lemma *mu-square*: *isotone f* $\wedge$ *has-least-fixpoint f* $\wedge$ *has-least-fixpoint* $(f \circ f) \rightarrow \mu\ f = \mu\ (f \circ f)$
proof

assume *1*: *isotone f* ∧ *has-least-fixpoint f* ∧ *has-least-fixpoint* (*f* ∘ *f*)
hence *2*: *is-least-fixpoint* (*f* ∘ *f*) (*μ* (*f* ∘ *f*)) ∧ *is-least-fixpoint f* (*μ f*)
using *least-fixpoint* **by** *auto*
hence *f* (*μ* (*f* ∘ *f*)) ≤ *μ* (*f* ∘ *f*) **using** *1*
by (*metis* (*no-types*) *comp-def is-least-fixpoint-def isotone-def*)
thus *μ f* = *μ* (*f* ∘ *f*) **using** *2*
by (*metis comp-def is-least-fixpoint-def antisym-conv*)
qed

lemma *nu-square*: *isotone f* ∧ *has-greatest-fixpoint f* ∧ *has-greatest-fixpoint* (*f* ∘ *f*) ⟶ *ν f* = *ν* (*f* ∘ *f*)
proof
assume *1*: *isotone f* ∧ *has-greatest-fixpoint f* ∧ *has-greatest-fixpoint* (*f* ∘ *f*)
hence *2*: *is-greatest-fixpoint* (*f* ∘ *f*) (*ν* (*f* ∘ *f*)) ∧ *is-greatest-fixpoint f* (*ν f*)
using *greatest-fixpoint* **by** *auto*
hence *ν* (*f* ∘ *f*) ≤ *f* (*ν* (*f* ∘ *f*)) **using** *1*
by (*metis* (*no-types*) *comp-def is-greatest-fixpoint-def isotone-def*)
thus *ν f* = *ν* (*f* ∘ *f*) **using** *2*
by (*metis comp-def is-greatest-fixpoint-def antisym-conv*)
qed

lemma *mu-roll*: *isotone g* ∧ *has-least-fixpoint* (*f* ∘ *g*) ∧ *has-least-fixpoint* (*g* ∘ *f*) ⟶ *μ* (*g* ∘ *f*) = *g*(*μ* (*f* ∘ *g*))
apply (*rule impI*)
apply (*rule antisym*)
apply (*smt is-least-fixpoint-def least-fixpoint o-apply*)
apply (*smt is-least-fixpoint-def isotone-def least-fixpoint o-apply*)
done

lemma *nu-roll*: *isotone g* ∧ *has-greatest-fixpoint* (*f* ∘ *g*) ∧ *has-greatest-fixpoint* (*g* ∘ *f*) ⟶ *ν* (*g* ∘ *f*) = *g*(*ν* (*f* ∘ *g*))
apply (*rule impI*)
apply (*rule antisym*)
apply (*smt is-greatest-fixpoint-def greatest-fixpoint isotone-def o-apply*)
apply (*smt is-greatest-fixpoint-def greatest-fixpoint o-apply*)
done

lemma *mu-below-nu*: *has-least-fixpoint f* ∧ *has-greatest-fixpoint f* ⟶ *μ f* ≤ *ν f*
by (*metis is-greatest-fixpoint-def is-least-fixpoint-def least-fixpoint greatest-fixpoint*)

lemma *pmu-below-pnu-fix*: *has-fixpoint f* ∧ *has-least-prefixpoint f* ∧ *has-greatest-postfixpoint f* ⟶ *pμ f* ≤ *pν f*
by (*smt has-fixpoint-def is-fixpoint-def is-greatest-postfixpoint-def is-least-prefixpoint-def le-less order-trans least-prefixpoint greatest-postfixpoint*)

lemma *pmu-below-pnu-iso*: *isotone f* ∧ *has-least-prefixpoint f* ∧ *has-greatest-postfixpoint f* ⟶ *pμ f* ≤ *pν f*
by (*metis has-fixpoint-def is-fixpoint-def is-least-fixpoint-def least-prefixpoint-fixpoint pmu-below-pnu-fix*)

lemma *mu-fusion-1*: *galois l u* ∧ *isotone h* ∧ *has-least-prefixpoint g* ∧ *has-least-fixpoint h* ∧ *l*(*g*(*u*(*μ h*))) ≤ *h*(*l*(*u*(*μ h*))) ⟶ *l*(*pμ g*) ≤ *μ h*
proof
assume *1*: *galois l u* ∧ *isotone h* ∧ *has-least-prefixpoint g* ∧ *has-least-fixpoint h* ∧ *l*(*g*(*u*(*μ h*))) ≤ *h*(*l*(*u*(*μ h*)))
hence *l*(*g*(*u*(*μ h*))) ≤ *μ h*

by (*metis galois-char least-fixpoint-same least-fixpoint-unique is-least-fixpoint-def isotone-def order-trans*)
thus $l(p\mu\ g) \leq \mu\ h$ **using** *1*
by (*metis galois-def least-prefixpoint is-least-prefixpoint-def least-fixpoint-same least-fixpoint-unique*)
qed

lemma *mu-fusion-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l \circ g \leq\leq h \circ l \rightarrow l(p\mu\ g) \leq \mu\ h$
by (*metis lifted-less-eq-def mu-fusion-1 o-apply*)

lemma *mu-fusion-equal-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h)))$ $\wedge$ $l(g(p\mu\ g)) = h(l(p\mu\ g)) \rightarrow \mu\ h = l(p\mu\ g)$ $\wedge$ $\mu\ h = l(\mu\ g)$
by (*metis antisym least-fixpoint least-prefixpoint-fixpoint is-least-fixpoint-def mu-fusion-1 pmu-mu*)

lemma *mu-fusion-equal-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-prefixpoint h* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h)))$ $\wedge$ $l(g(p\mu\ g)) = h(l(p\mu\ g)) \rightarrow p\mu\ h = l(p\mu\ g)$ $\wedge$ $\mu\ h = l(p\mu\ g)$
proof −
have *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-prefixpoint h* $\wedge$ $l(g(p\mu\ g)) = h(l(p\mu\ g)) \rightarrow \mu\ h \leq l(p\mu\ g)$
by (*metis galois-char is-least-prefixpoint-def isotone-def least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint*)
thus *?thesis*
by (*metis antisym least-fixpoint-char least-prefixpoint-fixpoint mu-fusion-1*)
qed

lemma *mu-fusion-equal-3*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l \circ g = h \circ l \rightarrow \mu\ h = l(p\mu\ g) \wedge \mu\ h = l(\mu\ g)$
by (*metis mu-fusion-equal-1 o-apply order-refl*)

lemma *mu-fusion-equal-4*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-prefixpoint h* $\wedge$ $l \circ g = h \circ l \rightarrow p\mu\ h = l(p\mu\ g) \wedge \mu\ h = l(p\mu\ g)$
by (*metis mu-fusion-equal-2 o-apply order-refl*)

lemma *nu-fusion-1*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h))) \rightarrow \nu\ h \leq u(p\nu\ g)$
proof
assume *1*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h)))$
hence $\nu\ h \leq u(g(l(\nu\ h)))$ **using** *1*
by (*metis galois-char greatest-fixpoint-same greatest-fixpoint-unique is-greatest-fixpoint-def isotone-def order-trans*)
thus $\nu\ h \leq u(p\nu\ g)$ **using** *1*
by (*smt galois-def greatest-postfixpoint is-greatest-postfixpoint-def greatest-fixpoint-same greatest-fixpoint-unique*)
qed

lemma *nu-fusion-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h \circ u \leq\leq u \circ g \rightarrow \nu\ h \leq u(p\nu\ g)$
by (*metis lifted-less-eq-def nu-fusion-1 o-apply*)

lemma *nu-fusion-equal-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h)))$ $\wedge$ $h(u(p\nu\ g)) = u(g(p\nu\ g)) \rightarrow \nu\ h = u(p\nu\ g)$ $\wedge$ $\nu\ h = u(\nu\ g)$
by (*metis antisym greatest-fixpoint greatest-postfixpoint-fixpoint is-greatest-fixpoint-def nu-fusion-1 pnu-nu*)

lemma *nu-fusion-equal-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-postfixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h)))$ $\wedge$ $h(u(p\nu\ g)) = u(g(p\nu\ g)) \rightarrow p\nu\ h = u(p\nu\ g)$ $\wedge$ $\nu\ h = u(p\nu\ g)$
proof −
have *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-postfixpoint h* $\wedge$ $h(u(p\nu\ g)) = u(g(p\nu\ g)) \rightarrow u(p\nu\ g) \leq \nu\ h$

by (*metis galois-char greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint is-greatest-postfixpoint-def isotone-def*)
thus *?thesis*
by (*metis antisym greatest-fixpoint-char greatest-postfixpoint-fixpoint nu-fusion-1*)
qed

lemma *nu-fusion-equal-3*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h \circ u = u \circ g \rightarrow \nu\ h = u(p\nu\ g) \wedge \nu\ h = u(\nu\ g)$
by (*metis nu-fusion-equal-1 o-apply order-refl*)

lemma *nu-fusion-equal-4*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-postfixpoint h* $\wedge$ $h \circ u = u \circ g \rightarrow p\nu\ h = u(p\nu\ g) \wedge \nu\ h = u(p\nu\ g)$
by (*metis nu-fusion-equal-2 o-apply order-refl*)

lemma *mu-exchange-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h)$ $\wedge$ $l \circ h \circ g \leq\leq g \circ h \circ l \rightarrow \mu(l \circ h) \leq \mu(g \circ h)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h)$ $\wedge$ $l \circ h \circ g \leq\leq g \circ h \circ l$
hence *2*: $l \circ (h \circ g) \leq\leq (g \circ h) \circ l$
by (*metis o-assoc*)
have $(l \circ h)\ (\mu(g \circ h)) = l(\mu(h \circ g))$ **using** *1*
by (*metis isotone-def least-fixpoint-char least-prefixpoint-fixpoint mu-roll o-apply*)
also have $\ldots \leq \mu(g \circ h)$ **using** *1 2*
by (*metis isotone-def least-fixpoint-char least-prefixpoint-fixpoint mu-fusion-2 o-apply*)
finally have $p\mu(l \circ h) \leq \mu(g \circ h)$ **using** *1*
by (*metis is-least-prefixpoint-def least-prefixpoint*)
thus $\mu(l \circ h) \leq \mu(g \circ h)$ **using** *1*
by (*metis galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint o-apply*)
qed

lemma *mu-exchange-2*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ l)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h)$ $\wedge$ *has-least-fixpoint* $(h \circ g)$ $\wedge$ $l \circ h \circ g \leq\leq g \circ h \circ l \rightarrow \mu(h \circ l) \leq \mu(h \circ g)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ l)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h)$ $\wedge$ *has-least-fixpoint* $(h \circ g)$ $\wedge$ $l \circ h \circ g \leq\leq g \circ h \circ l$
have $\mu(h \circ l) = h(\mu(l \circ h))$ **using** *1*
by (*metis* (*no-types*, *lifting*) *galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint mu-roll o-apply*)
also have $\ldots \leq h(\mu(g \circ h))$ **using** *1*
using *isotone-def mu-exchange-1* **by** *blast*
also have $\ldots = \mu(h \circ g)$ **using** *1*
by (*metis mu-roll*)
finally show $\mu(h \circ l) \leq \mu(h \circ g)$
.
qed

lemma *mu-exchange-equal*: *galois l u* $\wedge$ *galois k t* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ l)$ $\wedge$ *has-least-prefixpoint* $(k \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ k)$ $\wedge$ $l \circ h \circ k = k \circ h \circ l \rightarrow \mu(l \circ h) = \mu(k \circ h) \wedge \mu(h \circ l) = \mu(h \circ k)$
using *antisym galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint lifted-reflexive mu-exchange-1 mu-exchange-2* **by** *force*

lemma *nu-exchange-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h)$ $\wedge$ $g \circ h \circ u \leq\leq u \circ h \circ g$

$\rightarrow \nu(g \circ h) \leq \nu(u \circ h)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h)$ $\wedge$ $g \circ h \circ u \leq\leq u \circ h \circ g$
hence $(g \circ h) \circ u \leq\leq u \circ (h \circ g)$
by (*metis o-assoc*)
hence $\nu(g \circ h) \leq u(\nu(h \circ g))$ **using** *1*
by (*metis greatest-fixpoint-char greatest-postfixpoint-fixpoint isotone-def nu-fusion-2 o-apply*)
also have $... = (u \circ h)\ (\nu(g \circ h))$ **using** *1*
by (*metis greatest-fixpoint-char greatest-postfixpoint-fixpoint isotone-def nu-roll o-apply*)
finally have $\nu(g \circ h) \leq p\nu(u \circ h)$ **using** *1*
using *greatest-postfixpoint is-greatest-postfixpoint-def* **by** *blast*
thus $\nu(g \circ h) \leq \nu(u \circ h)$ **using** *1*
using *galois-char greatest-fixpoint-char greatest-postfixpoint-fixpoint isotone-def* **by** *auto*
qed

lemma *nu-exchange-2*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ u)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h)$ $\wedge$ *has-greatest-fixpoint* $(h \circ g)$ $\wedge$ $g \circ h \circ u \leq\leq u \circ h \circ g \rightarrow \nu(h \circ g) \leq \nu(h \circ u)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ u)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h)$ $\wedge$ *has-greatest-fixpoint* $(h \circ g)$ $\wedge$ $g \circ h \circ u \leq\leq u \circ h \circ g$
have $\nu(h \circ g) = h(\nu(g \circ h))$ **using** *1*
using *nu-roll* **by** *blast*
also have $... \leq h(\nu(u \circ h))$ **using** *1*
using *isotone-def nu-exchange-1* **by** *blast*
also have $... = \nu(h \circ u)$ **using** *1*
by (*metis* (*no-types*, *lifting*) *galois-char greatest-fixpoint-char greatest-postfixpoint-fixpoint isotone-def nu-roll o-apply*)
finally show $\nu(h \circ g) \leq \nu(h \circ u)$
.
qed

lemma *nu-exchange-equal*: *galois l u* $\wedge$ *galois k t* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ u)$ $\wedge$ *has-greatest-postfixpoint* $(t \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ t)$ $\wedge$ $u \circ h \circ t = t \circ h \circ u \rightarrow \nu(u \circ h) = \nu(t \circ h) \wedge \nu(h \circ u) = \nu(h \circ t)$
using *antisym galois-char greatest-fixpoint-char greatest-postfixpoint-fixpoint isotone-def lifted-reflexive nu-exchange-1 nu-exchange-2* **by** *auto*

lemma *mu-commute-fixpoint-1*: *isotone f* $\wedge$ *has-least-fixpoint* $(f \circ g)$ $\wedge$ $f \circ g = g \circ f \rightarrow$ *is-fixpoint f* $(\mu(f \circ g))$
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-fixpoint-2*: *isotone g* $\wedge$ *has-least-fixpoint* $(f \circ g)$ $\wedge$ $f \circ g = g \circ f \rightarrow$ *is-fixpoint g* $(\mu(f \circ g))$
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-least-fixpoint*: *isotone f* $\wedge$ *isotone g* $\wedge$ *has-least-fixpoint f* $\wedge$ *has-least-fixpoint g* $\wedge$ *has-least-fixpoint* $(f \circ g)$ $\wedge$ $f \circ g = g \circ f \rightarrow (\mu(f \circ g) = \mu\ f \rightarrow \mu\ g \leq \mu\ f)$
by (*metis is-least-fixpoint-def least-fixpoint-same least-fixpoint-unique mu-roll*)

lemma *nu-commute-fixpoint-1*: *isotone f* $\wedge$ *has-greatest-fixpoint* $(f \circ g)$ $\wedge$ $f \circ g = g \circ f \rightarrow$ *is-fixpoint f* $(\nu(f \circ g))$
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-fixpoint-2*: *isotone* $g \wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow$ *is-fixpoint* g $(\nu(f \circ g))$
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-greatest-fixpoint*: *isotone* $f \wedge$ *isotone* $g \wedge$ *has-greatest-fixpoint* $f \wedge$ *has-greatest-fixpoint* $g \wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow (\nu(f \circ g) = \nu\ f \longrightarrow \nu\ f \leq \nu\ g)$
by (*smt is-greatest-fixpoint-def greatest-fixpoint-same greatest-fixpoint-unique nu-roll*)

lemma *mu-diagonal-1*: *isotone* $(\lambda x\ .\ f\ x\ x) \wedge (\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y)) \wedge$ *isotone* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \wedge (\forall x\ .$ *has-least-fixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \longrightarrow \mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
by (*smt is-least-fixpoint-def is-least-prefixpoint-def least-fixpoint-same least-fixpoint-unique least-prefixpoint least-prefixpoint-fixpoint*)

lemma *mu-diagonal-2*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-least-prefixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \longrightarrow \mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
proof
assume *1*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-least-prefixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
hence *isotone* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
by (*smt isotone-def lifted-less-eq-def mu-isotone*)
thus $\mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$ **using** *1*
by (*smt is-least-fixpoint-def is-least-prefixpoint-def least-fixpoint-same least-prefixpoint least-prefixpoint-fixpoint*)
qed

lemma *nu-diagonal-1*: *isotone* $(\lambda x\ .\ f\ x\ x) \wedge (\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y)) \wedge$ *isotone* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \wedge (\forall x\ .$ *has-greatest-fixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \longrightarrow \nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
by (*smt is-greatest-fixpoint-def is-greatest-postfixpoint-def greatest-fixpoint-same greatest-fixpoint-unique greatest-postfixpoint greatest-postfixpoint-fixpoint*)

lemma *nu-diagonal-2*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-greatest-postfixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \longrightarrow \nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
proof
assume *1*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-greatest-postfixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
hence *isotone* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
by (*smt isotone-def lifted-less-eq-def nu-isotone*)
thus $\nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$ **using** *1*
by (*smt greatest-fixpoint-same greatest-postfixpoint greatest-postfixpoint-fixpoint is-greatest-fixpoint-def is-greatest-postfixpoint-def*)
qed

end

class *join-semilattice* = *plus* + *ord* +
assumes *add-associative*: $(x + y) + z = x + (y + z)$
assumes *add-commutative*: $x + y = y + x$
assumes *add-idempotent*: $x + x = x$
assumes *less-eq-def*: $x \leq y \leftrightarrow x + y = y$
assumes *less-def*: $x < y \leftrightarrow x \leq y \wedge \neg\ (y \leq x)$

begin

subclass *order*
  apply *unfold-locales*
  apply (*metis less-def*)
  apply (*metis add-idempotent less-eq-def*)
  apply (*metis add-associative less-eq-def*)
  apply (*metis add-commutative less-eq-def*)
  done

lemma *add-left-isotone*: $x \leq y \longrightarrow x + z \leq y + z$
  by (*smt add-associative add-commutative add-idempotent less-eq-def*)

lemma *add-right-isotone*: $x \leq y \longrightarrow z + x \leq z + y$
  by (*metis add-commutative add-left-isotone*)

lemma *add-isotone*: $w \leq y \wedge x \leq z \longrightarrow w + x \leq y + z$
  by (*smt add-associative add-commutative less-eq-def*)

lemma *add-left-upper-bound*: $x \leq x + y$
  by (*metis add-associative add-idempotent less-eq-def*)

lemma *add-right-upper-bound*: $y \leq x + y$
  by (*metis add-commutative add-left-upper-bound*)

lemma *add-least-upper-bound*: $x \leq z \wedge y \leq z \leftrightarrow x + y \leq z$
  by (*smt add-associative add-commutative add-left-upper-bound less-eq-def*)

lemma *add-left-divisibility*: $x \leq y \leftrightarrow (\exists z \,.\, x + z = y)$
  by (*metis add-left-upper-bound less-eq-def*)

lemma *add-right-divisibility*: $x \leq y \leftrightarrow (\exists z \,.\, z + x = y)$
  by (*metis add-commutative add-left-divisibility*)

lemma *add-same-context*:$x \leq y + z \wedge y \leq x + z \longrightarrow x + z = y + z$
  by (*smt add-associative add-commutative less-eq-def*)

lemma *add-relative-same-increasing*: $x \leq y \wedge x + z = x + w \longrightarrow y + z = y + w$
  by (*smt add-associative add-right-divisibility*)

lemma *ascending-chain-left-add*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n \,.\, x + f\ n)$
  by (*metis ascending-chain-def add-right-isotone*)

lemma *ascending-chain-right-add*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n \,.\, f\ n + x)$
  by (*metis ascending-chain-def add-left-isotone*)

lemma *descending-chain-left-add*: *descending-chain* $f \longrightarrow$ *descending-chain* $(\lambda n \,.\, x + f\ n)$

by (*metis descending-chain-def add-right-isotone*)

lemma *descending-chain-right-add*: *descending-chain f* ⟶ *descending-chain* (λ*n* . *f n* + *x*)
by (*metis descending-chain-def add-left-isotone*)

primrec *pSum0* :: (*nat* ⇒ *'a*) ⇒ *nat* ⇒ *'a*
where *pSum0 f 0* = *f 0*
| *pSum0 f* (*Suc m*) = *pSum0 f m* + *f m*

lemma *pSum0-below*: (∀ *i* . *f i* ≤ *x*) ⟶ *pSum0 f m* ≤ *x*
apply (*induct m*)
apply (*metis pSum0.simps*(*1*))
by (*metis add-least-upper-bound pSum0.simps*(*2*))

end

class *bounded-join-semilattice* = *join-semilattice* + *zero* +
assumes *add-left-zero*: *0* + *x* = *x*

begin

lemma *add-right-zero*: *x* + *0* = *x*
by (*metis add-commutative add-left-zero*)

lemma *zero-least*: *0* ≤ *x*
by (*metis add-left-upper-bound add-left-zero*)

end

class *meet* =
fixes *meet* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ⌢ *65*)

class *meet-semilattice* = *meet* + *ord* +
assumes *meet-associative*: (*x* ⌢ *y*) ⌢ *z* = *x* ⌢ (*y* ⌢ *z*)
assumes *meet-commutative*: *x* ⌢ *y* = *y* ⌢ *x*
assumes *meet-idempotent*: *x* ⌢ *x* = *x*
assumes *meet-less-eq-def*: *x* ≤ *y* ↔ *x* ⌢ *y* = *x*
assumes *meet-less-def*: *x* < *y* ↔ *x* ≤ *y* ∧ ¬ (*y* ≤ *x*)

sublocale *meet-semilattice* < *meet*!: *join-semilattice* **where** *plus* = *meet* **and** *less-eq* = (λ*x y* . *y* ≤ *x*) **and** *less* = (λ*x y* . *y* < *x*)
apply *unfold-locales*
apply (*rule meet-associative*)
apply (*rule meet-commutative*)
apply (*rule meet-idempotent*)
apply (*metis meet-commutative meet-less-eq-def*)
apply (*metis meet-less-def*)
done

class $T =$
  fixes $T :: 'a\ (\top)$

class *bounded-meet-semilattice* $=$ *meet-semilattice* $+$ $T +$
  assumes *meet-left-top*: $T \frown x = x$

sublocale *bounded-meet-semilattice* $<$ *meet*!: *bounded-join-semilattice* **where** *plus* $=$ *meet* **and** *less-eq* $= (\lambda x\ y\ .\ y \leq x)$ **and** *less* $= (\lambda x\ y\ .\ y < x)$ **and** *zero* $= T$
  apply *unfold-locales*
  apply (*rule meet-left-top*)
  done

class *bounded-distributive-lattice* $=$ *bounded-join-semilattice* $+$ *bounded-meet-semilattice* $+$
  assumes *meet-left-dist-add*: $x \frown (y + z) = (x \frown y) + (x \frown z)$
  assumes *add-left-dist-meet*: $x + (y \frown z) = (x + y) \frown (x + z)$
  assumes *meet-absorb*: $x \frown (x + y) = x$
  assumes *add-absorb*: $x + (x \frown y) = x$

begin

lemma *meet-left-zero*: $0 \frown x = 0$
  by (*metis add-absorb add-left-zero*)

lemma *meet-right-zero*: $x \frown 0 = 0$
  by (*metis meet-commutative meet-left-zero*)

lemma *add-left-top-1*: $T + x = T$
  by (*metis add-absorb meet-left-top*)

lemma *add-right-top-1*: $x + T = T$
  by (*metis add-commutative add-left-top-1*)

lemma *meet-same-context*:$x \leq y \frown z \wedge y \leq x \frown z \rightarrow x \frown z = y \frown z$
  by (*metis eq-iff meet.add-least-upper-bound*)

lemma *relative-equality*: $x + z = y + z \wedge x \frown z = y \frown z \rightarrow x = y$
  by (*metis add-absorb add-commutative add-left-dist-meet*)

end

class *monoid* $=$ *mult* $+$ *one* $+$
  assumes *mult-associative*: $(x\ ;\ y)\ ;\ z = x\ ;\ (y\ ;\ z)$
  assumes *mult-left-one-1*: $1\ ;\ x = x$
  assumes *mult-right-one*: $x\ ;\ 1 = x$

class *non-associative-left-semiring* = *bounded-join-semilattice* + *mult* + *one* +
  assumes *mult-left-sub-dist-add*: $x\ ;\ y + x\ ;\ z \leq x\ ;\ (y + z)$
  assumes *mult-right-dist-add*: $(x + y)\ ;\ z = x\ ;\ z + y\ ;\ z$
  assumes *mult-left-zero*: $0\ ;\ x = 0$
  assumes *mult-left-one*: $1\ ;\ x = x$
  assumes *mult-sub-right-one*: $x \leq x\ ;\ 1$

begin

lemma *mult-left-isotone*: $x \leq y \longrightarrow x\ ;\ z \leq y\ ;\ z$
  by (*metis less-eq-def mult-right-dist-add*)

lemma *mult-right-isotone*: $x \leq y \longrightarrow z\ ;\ x \leq z\ ;\ y$
  by (*metis add-least-upper-bound less-eq-def mult-left-sub-dist-add*)

lemma *mult-isotone*: $w \leq y \wedge x \leq z \longrightarrow w\ ;\ x \leq y\ ;\ z$
  by (*metis mult-left-isotone mult-right-isotone order-trans*)

lemma *affine-isotone*: *isotone* $(\lambda x\ .\ y\ ;\ x + z)$
  by (*smt add-commutative add-right-isotone isotone-def mult-right-isotone*)

lemma *mult-left-sub-dist-add-left*: $x\ ;\ y \leq x\ ;\ (y + z)$
  by (*metis add-left-upper-bound mult-right-isotone*)

lemma *mult-left-sub-dist-add-right*: $x\ ;\ z \leq x\ ;\ (y + z)$
  by (*metis add-right-upper-bound mult-right-isotone*)

lemma *mult-right-sub-dist-add-left*: $x\ ;\ z \leq (x + y)\ ;\ z$
  by (*metis add-left-upper-bound mult-right-dist-add*)

lemma *mult-right-sub-dist-add-right*: $y\ ;\ z \leq (x + y)\ ;\ z$
  by (*metis add-right-upper-bound mult-right-dist-add*)

lemma *case-split-left*: $1 \leq w + z \wedge w\ ;\ x \leq y \wedge z\ ;\ x \leq y \longrightarrow x \leq y$
  by (*smt add-associative add-commutative less-eq-def mult-left-one mult-right-dist-add order-refl*)

lemma *case-split-left-equal*: $w + z = 1 \wedge w\ ;\ x = w\ ;\ y \wedge z\ ;\ x = z\ ;\ y \longrightarrow x = y$
  by (*metis mult-left-one mult-right-dist-add*)

lemma *ascending-chain-left-mult*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n\ .\ x\ ;\ f\ n)$
  by (*metis ascending-chain-def mult-right-isotone*)

lemma *ascending-chain-right-mult*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n\ .\ f\ n\ ;\ x)$
  by (*metis ascending-chain-def mult-left-isotone*)

lemma *descending-chain-left-mult*: *descending-chain* $f \longrightarrow$ *descending-chain* $(\lambda n\ .\ x\ ;\ f\ n)$
  by (*metis descending-chain-def mult-right-isotone*)

lemma *descending-chain-right-mult*: *descending-chain f* $\longrightarrow$ *descending-chain* (λn . *f n* ; *x*)
  by (*metis descending-chain-def mult-left-isotone*)

— Theorem 5

abbreviation *Lf* :: $'a \Rightarrow ('a \Rightarrow 'a)$ **where** *Lf y* $\equiv$ (λ *x* . *1* + *x* ; *y*)
abbreviation *Rf* :: $'a \Rightarrow ('a \Rightarrow 'a)$ **where** *Rf y* $\equiv$ (λ *x* . *1* + *y* ; *x*)
abbreviation *Sf* :: $'a \Rightarrow ('a \Rightarrow 'a)$ **where** *Sf y* $\equiv$ (λ *x* . *1* + *y* + *x* ; *x*)

abbreviation *lstar* :: $'a \Rightarrow 'a$ **where** *lstar y* $\equiv p\mu$ (*Lf y*)
abbreviation *rstar* :: $'a \Rightarrow 'a$ **where** *rstar y* $\equiv p\mu$ (*Rf y*)
abbreviation *sstar* :: $'a \Rightarrow 'a$ **where** *sstar y* $\equiv p\mu$ (*Sf y*)

lemma *lstar-rec-isotone*: *isotone* (*Lf y*)
  by (*smt add-isotone add-right-divisibility isotone-def mult-right-sub-dist-add-right order.refl*)

lemma *rstar-rec-isotone*: *isotone* (*Rf y*)
  by (*metis add-isotone isotone-def less-eq-def mult-left-sub-dist-add-left order-refl*)

lemma *sstar-rec-isotone*: *isotone* (*Sf y*)
  by (*simp add*: *add-right-isotone isotone-def mult-isotone*)

lemma *lstar-fixpoint*: *has-least-prefixpoint* (*Lf y*) $\longrightarrow$ *lstar y* = μ (*Lf y*)
  by (*metis pmu-mu lstar-rec-isotone*)

lemma *rstar-fixpoint*: *has-least-prefixpoint* (*Rf y*) $\longrightarrow$ *rstar y* = μ (*Rf y*)
  by (*metis pmu-mu rstar-rec-isotone*)

lemma *sstar-fixpoint*: *has-least-prefixpoint* (*Sf y*) $\longrightarrow$ *sstar y* = μ (*Sf y*)
  by (*metis pmu-mu sstar-rec-isotone*)

lemma *sstar-increasing*: *has-least-prefixpoint* (*Sf y*) $\longrightarrow$ *y* $\leq$ *sstar y*
  by (*metis add-least-upper-bound add-left-upper-bound order.trans pmu-unfold*)

— Theorem 5 part 2

lemma *rstar-below-sstar*: *has-least-prefixpoint* (*Rf y*) $\wedge$ *has-least-prefixpoint* (*Sf y*) $\longrightarrow$ *rstar y* $\leq$ *sstar y*
proof
  assume *1*: *has-least-prefixpoint* (*Rf y*) $\wedge$ *has-least-prefixpoint* (*Sf y*)
  hence *Rf y* (*sstar y*) $\leq$ *Sf y* (*sstar y*)
    by (*smt add-isotone add-left-upper-bound add-right-upper-bound dual-order.trans mult-left-isotone pmu-unfold*)
  also have ... $\leq$ *sstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *pmu-unfold*)
  finally show *rstar y* $\leq$ *sstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint*)
qed

end

class *pre-left-semiring* = *non-associative-left-semiring* +
  assumes *mult-semi-associative*: $(x\ ;\ y)\ ;\ z \leq x\ ;\ (y\ ;\ z)$

begin

lemma *mult-one-associative*: $x\ ;\ 1\ ;\ y = x\ ;\ y$
  by (*metis dual-order.antisym mult-left-isotone mult-left-one mult-semi-associative mult-sub-right-one*)

lemma *mult-sup-associative-one*: $(x\ ;\ (y\ ;\ 1))\ ;\ z \leq x\ ;\ (y\ ;\ z)$
  by (*metis mult-semi-associative mult-one-associative*)

lemma *rstar-increasing*: *has-least-prefixpoint* $(Rf\ y) \rightarrow y \leq rstar\ y$
proof
  assume *has-least-prefixpoint* $(Rf\ y)$
  hence $Rf\ y\ (rstar\ y) \leq rstar\ y$
    by (*metis pmu-unfold*)
  thus $y \leq rstar\ y$
    by (*metis add-least-upper-bound mult-right-isotone mult-sub-right-one order.trans*)
qed

end

class *residuated-pre-left-semiring* = *pre-left-semiring* + *inverse* +
  assumes *lres-galois*: $x\ ;\ y \leq z \leftrightarrow x \leq z\ /\ y$

begin

lemma *lres-left-isotone*: $x \leq y \rightarrow x\ /\ z \leq y\ /\ z$
  by (*metis lres-galois order.refl order.trans*)

lemma *lres-right-antitone*: $x \leq y \rightarrow z\ /\ y \leq z\ /\ x$
  by (*metis lres-galois mult-right-isotone order.refl order-trans*)

lemma *lres-inverse*: $(x\ /\ y)\ ;\ y \leq x$
  by (*metis lres-galois order-refl*)

lemma *lres-one*: $x\ /\ 1 \leq x$
  by (*metis dual-order.trans mult-sub-right-one lres-inverse*)

lemma *lres-mult-sub-lres-lres*: $x\ /\ (z\ ;\ y) \leq (x\ /\ y)\ /\ z$
  by (*metis lres-galois lres-inverse mult-semi-associative order.trans*)

lemma *mult-lres-sub-assoc*: $x\ ;\ (y\ /\ z) \leq (x\ ;\ y)\ /\ z$
  by (*metis* (*full-types*) *lres-galois lres-inverse mult-right-isotone mult-semi-associative order-trans*)

— Theorem 5 part 1

lemma *lstar-below-rstar*: *has-least-prefixpoint* (*Lf y*) ∧ *has-least-prefixpoint* (*Rf y*) → *lstar y* ≤ *rstar y*
proof
  assume *1*: *has-least-prefixpoint* (*Lf y*) ∧ *has-least-prefixpoint* (*Rf y*)
  have *y* ; (*rstar y* / *y*) ; *y* ≤ *y* ; *rstar y*
    by (*metis mult-right-isotone mult-semi-associative order-trans lres-inverse*)
  also have ... ≤ *rstar y* **using** *1*
    by (*metis add-least-upper-bound pmu-unfold*)
  finally have *y* ; (*rstar y* / *y*) ≤ *rstar y* / *y*
    by (*metis lres-galois*)
  hence *Rf y* (*rstar y* / *y*) ≤ *rstar y* / *y* **using** *1*
    by (*metis add-least-upper-bound lres-galois mult-left-one rstar-increasing*)
  hence *rstar y* ≤ *rstar y* / *y* **using** *1*
    by (*metis is-least-prefixpoint-def least-prefixpoint*)
  hence *Lf y* (*rstar y*) ≤ *rstar y* **using** *1*
    by (*metis add-least-upper-bound lres-galois pmu-unfold*)
  thus *lstar y* ≤ *rstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint*)
qed

— Theorem 5 part 3

lemma *rstar-sstar*: *has-least-prefixpoint* (*Rf y*) ∧ *has-least-prefixpoint* (*Sf y*) → *rstar y* = *sstar y*
proof
  assume *1*: *has-least-prefixpoint* (*Rf y*) ∧ *has-least-prefixpoint* (*Sf y*)
  have *Rf y* (*rstar y* / *rstar y*) ; *rstar y* ≤ *rstar y* + *y* ; ((*rstar y* / *rstar y*) ; *rstar y*)
    by (*metis add-isotone mult-left-one mult-right-dist-add mult-semi-associative*)
  also have ... ≤ *rstar y* + *y* ; *rstar y*
    by (*metis add-right-isotone mult-right-isotone lres-inverse*)
  also have ... ≤ *rstar y* **using** *1*
    by (*metis* (*full-types*) *add-least-upper-bound order-refl pmu-unfold*)
  finally have *Rf y* (*rstar y* / *rstar y*) ≤ *rstar y* / *rstar y*
    by (*metis lres-galois*)
  hence *rstar y* ; *rstar y* ≤ *rstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint lres-galois*)
  hence *y* + *rstar y* ; *rstar y* ≤ *rstar y* **using** *1*
    by (*metis add-least-upper-bound rstar-increasing*)
  hence *Sf y* (*rstar y*) ≤ *rstar y* **using** *1*
    by (*metis* (*full-types*) *add-least-upper-bound pmu-unfold*)
  hence *sstar y* ≤ *rstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint*)
  thus *rstar y* = *sstar y* **using** *1*
    by (*metis antisym rstar-below-sstar*)
qed

end

class *idempotent-left-semiring* = *non-associative-left-semiring* + *monoid*

begin

subclass *pre-left-semiring*
apply *unfold-locales*
apply (*metis mult-associative order-refl*)
done

lemma *zero-right-mult-decreasing*: $x \ ; \ 0 \leq x$
by (*metis add-right-zero mult-left-sub-dist-add-right mult-right-one*)

lemma *test-preserves-equation*: $p \leq p \ ; \ p \wedge p \leq 1 \rightarrow (p \ ; \ x \leq x \ ; \ p \leftrightarrow p \ ; \ x = p \ ; \ x \ ; \ p)$
apply *rule*
apply *rule*
apply (*rule antisym*)
apply (*metis antisym mult-associative mult-right-isotone mult-right-one*)
apply (*metis mult-right-isotone mult-right-one*)
apply (*metis mult-left-isotone mult-left-one*)
done

end

class *idempotent-left-zero-semiring* = *idempotent-left-semiring* +
assumes *mult-left-dist-add*: $x \ ; \ (y + z) = x \ ; \ y + x \ ; \ z$

begin

lemma *case-split-right*: $1 \leq w + z \wedge x \ ; \ w \leq y \wedge x \ ; \ z \leq y \rightarrow x \leq y$
proof
assume *1*: $1 \leq w + z \wedge x \ ; \ w \leq y \wedge x \ ; \ z \leq y$
hence $x \leq x \ ; \ (w + z)$
by (*metis less-eq-def mult-left-dist-add mult-right-one*)
also have $... \leq y$ **using** *1*
by (*smt add-associative less-eq-def mult-left-dist-add*)
finally show $x \leq y$
.
qed

lemma *case-split-right-equal*: $w + z = 1 \wedge x \ ; \ w = y \ ; \ w \wedge x \ ; \ z = y \ ; \ z \rightarrow x = y$
by (*metis mult-left-dist-add mult-right-one*)

end

class *idempotent-semiring* = *idempotent-left-zero-semiring* +

assumes *mult-right-zero*: $x\ ;\ 0 = 0$

class *bounded-pre-left-semiring* = *pre-left-semiring* + T +
assumes *add-right-top*: $x + T = T$

begin

lemma *add-left-top*: $T + x = T$
by (*metis add-commutative add-right-top*)

lemma *top-greatest*: $x \leq T$
by (*metis add-left-top add-right-upper-bound*)

lemma *top-left-mult-increasing*: $x \leq T\ ;\ x$
by (*metis mult-left-isotone mult-left-one top-greatest*)

lemma *top-right-mult-increasing*: $x \leq x\ ;\ T$
by (*metis order-trans mult-right-isotone mult-sub-right-one top-greatest*)

lemma *top-mult-top*: $T\ ;\ T = T$
by (*metis add-right-divisibility add-right-top top-right-mult-increasing*)

definition *vector* :: $'a \Rightarrow$ *bool*
where *vector* $x \leftrightarrow x = x\ ;\ T$

lemma *vector-zero*: *vector 0*
by (*metis mult-left-zero vector-def*)

lemma *vector-top*: *vector* T
by (*metis top-mult-top vector-def*)

lemma *vector-add-closed*: *vector* $x \wedge$ *vector* $y \rightarrow$ *vector* $(x + y)$
by (*metis mult-right-dist-add vector-def*)

lemma *vector-left-mult-closed*: *vector* $y \rightarrow$ *vector* $(x\ ;\ y)$
by (*metis antisym mult-semi-associative top-right-mult-increasing vector-def*)

end

class *bounded-residuated-pre-left-semiring* = *residuated-pre-left-semiring* + *bounded-pre-left-semiring*

begin

lemma *lres-top-decreasing*: $x\ /\ T \leq x$
by (*metis lres-one lres-right-antitone order-trans top-greatest*)

lemma *top-lres-absorb*: $T\ /\ x = T$

by (*metis eq-iff lres-galois top-greatest*)

end

class *bounded-idempotent-left-semiring* = *bounded-pre-left-semiring* + *idempotent-left-semiring*

class *bounded-idempotent-left-zero-semiring* = *bounded-idempotent-left-semiring* + *idempotent-left-zero-semiring*

class *bounded-idempotent-semiring* = *bounded-idempotent-left-zero-semiring* + *idempotent-semiring*

notation
sup (**infixl** $\sqcup$ *65*) **and**
inf (**infixl** $\sqcap$ *69*) **and**
uminus (- ′ [*80*] *80*)

context *order*

begin

lemma *order-lesseq-imp*: $(\forall z\ .\ x \leq z \longrightarrow y \leq z) \leftrightarrow y \leq x$
using *order-trans* **by** *blast*

end

context *semilattice-sup*

begin

lemma *sup-left-isotone*: $x \leq y \longrightarrow x \sqcup z \leq y \sqcup z$
using *sup.mono* **by** *blast*

lemma *sup-right-isotone*: $x \leq y \longrightarrow z \sqcup x \leq z \sqcup y$
using *sup.mono* **by** *blast*

lemma *sup-isotone*: $w \leq y \wedge x \leq z \longrightarrow w \sqcup x \leq y \sqcup z$
using *sup.mono* **by** *blast*

lemma *sup-least-upper-bound*: $x \leq z \wedge y \leq z \leftrightarrow x \sqcup y \leq z$
by *simp*

lemma *sup-left-divisibility*: $x \leq y \leftrightarrow (\exists z\ .\ x \sqcup z = y)$
using *sup.absorb2 sup.cobounded1* **by** *blast*

lemma *sup-right-divisibility*: $x \leq y \leftrightarrow (\exists z\ .\ z \sqcup x = y)$
by (*metis sup.cobounded2 sup.orderE*)

lemma *sup-same-context*:$x \leq y \sqcup z \wedge y \leq x \sqcup z \longrightarrow x \sqcup z = y \sqcup z$
by (*metis* (*full-types*) *sup.orderE sup-assoc sup-commute*)

lemma *sup-relative-same-increasing*: $x \leq y \wedge x \sqcup z = x \sqcup w \longrightarrow y \sqcup z = y \sqcup w$
using *sup-assoc sup-commute sup-left-divisibility* **by** *auto*

end

sublocale *bounded-semilattice-inf-top* $<$ *inf*!: *bounded-semilattice-sup-bot* **where** *sup* $=$ *inf* **and** *less-eq* $= (\lambda x\ y\ .\ y \leq x)$ **and** *less* $= (\lambda x\ y\ .\ y < x)$ **and** *bot* $=$ *top*
apply *unfold-locales*
apply *simp-all*
apply (*simp add*: *less-le-not-le*)
done

context *semilattice-inf*

begin

lemma *inf-same-context*:$x \leq y \sqcap z \wedge y \leq x \sqcap z \longrightarrow x \sqcap z = y \sqcap z$
by (*metis inf.absorb2 inf.boundedE inf.orderE*)

end

context *distrib-lattice*

begin

lemma *relative-equality*: $x \sqcup z = y \sqcup z \wedge x \sqcap z = y \sqcap z \longrightarrow x = y$
by (*metis inf.commute inf-sup-absorb inf-sup-distrib1*)

end

context *boolean-algebra*

begin

— We follow Roger Maddux's 1996 paper.
Maddux Axioms (Ba1) *sup-assoc* (Ba2) *sup-commute*
Maddux Theorem 3 (ii) *double-compl* (vi) *sup-idem* (vii) *inf-idem* (viii) *inf-commute* (ix) *inf-assoc* (xiv) *sup-inf-absorb* (xv) *inf-sup-distrib1* (xvi) *compl-sup* (xvii) *compl-inf* (xviii) *sup-inf-distrib1*
Maddux Theorem 5 (i) *sup-compl-top* (ii) *inf-compl-bot* (iii) *compl-top-eq* (iv) *compl-bot-eq* (v) *sup-top-right* (vi) *inf-bot-right* (vii) *sup-bot-right* (viii) *inf-top-right*

lemma *compl-le-swap1-iff*: $x \leq -y \leftrightarrow y \leq -x$
using *compl-le-swap1* **by** *blast*

lemma *compl-le-swap2-iff*: $-x \leq y \leftrightarrow -y \leq x$

using *compl-le-swap2* **by** *blast*

lemma *huntington-3*: $x = -(-x \sqcup -y) \sqcup -(-x \sqcup y)$
by (*metis compl-inf compl-inf-bot double-compl sup-bot-right sup-inf-distrib1*)

— Maddux Theorem 3

lemma *maddux-3-1*: $x \sqcup -x = y \sqcup -y$
by (*simp add*: *sup-compl-top*)

lemma *maddux-3-3*: $-x = -(x \sqcup y) \sqcup -(x \sqcup -y)$
by (*metis huntington-3 double-compl*)

lemma *maddux-3-4*: $x \sqcup (y \sqcup -y) = z \sqcup -z$
by (*metis sup-assoc sup-idem maddux-3-1*)

lemma *maddux-3-5*: $x \sqcup x = x \sqcup -(y \sqcup -y)$
by *simp*

lemma *maddux-3-11*: $x = (x \sqcap y) \sqcup (x \sqcap -y)$
by (*metis inf-sup-distrib1 inf-top.right-neutral sup-compl-top*)

lemma *maddux-3-12*: $x = (x \sqcup -y) \sqcap (x \sqcup y)$
by (*metis compl-inf-bot sup-bot.right-neutral sup-inf-distrib1*)

lemma *maddux-3-13*: $(x \sqcup y) \sqcap -x = y \sqcap -x$
by (*simp add*: *inf-compl-bot inf-sup-distrib2*)

lemma *maddux-3-19*: $(-x \sqcap y) \sqcup (x \sqcap z) = (x \sqcup y) \sqcap (-x \sqcup z)$
by (*smt double-compl inf.commute inf-sup-absorb sup-ge1 sup-inf-absorb sup-inf-distrib2 sup-relative-same-increasing maddux-3-11 maddux-3-13*)

lemma *compl-inter-eq*: $x \sqcap y = x \sqcap z \wedge -x \sqcap y = -x \sqcap z \longrightarrow y = z$
by (*metis inf-commute maddux-3-11*)

lemma *maddux-3-20*: $((v \sqcap w) \sqcup (-v \sqcap x)) \sqcap -((v \sqcap y) \sqcup (-v \sqcap z)) = (v \sqcap w \sqcap -y) \sqcup (-v \sqcap x \sqcap -z)$
proof −
have *1*: $v \sqcap (((v \sqcap w) \sqcup (-v \sqcap x)) \sqcap -((v \sqcap y) \sqcup (-v \sqcap z))) = v \sqcap w \sqcap -y$
by (*smt compl-inf compl-sup double-compl inf.assoc inf.commute sup.left-commute sup-inf-absorb sup-inf-distrib1 maddux-3-13*)
have *2*: $... = v \sqcap ((v \sqcap w \sqcap -y) \sqcup (-v \sqcap x \sqcap -z))$
by (*smt inf.left-commute inf.orderE inf-bot-right inf-commute inf-compl-bot inf-le1 inf-sup-distrib1 sup-bot.right-neutral*)
have *31*: $-v \sqcap ((v \sqcap w) \sqcup (-v \sqcap x)) = -v \sqcap x$ **using** *maddux-3-11 maddux-3-13*
by (*smt comp-fun-idem.fun-left-idem compl-inf-bot inf.comp-fun-idem-sup inf-bot-right inf-sup-distrib1 sup-inf-absorb*)
have *32*: $-v \sqcap -((v \sqcap y) \sqcup (-v \sqcap z)) = -v \sqcap -z$ **using** *maddux-3-13*
by (*metis compl-sup double-compl inf-commute inf-idem sup-assoc sup-commute sup-inf-absorb sup-inf-distrib1 maddux-3-11*)
have *3*: $-v \sqcap (((v \sqcap w) \sqcup (-v \sqcap x)) \sqcap -((v \sqcap y) \sqcup (-v \sqcap z))) = -v \sqcap x \sqcap -z$ **using** *31 32*
by (*smt inf-commute inf-left-commute*)
have *4*: $... = -v \sqcap ((v \sqcap w \sqcap -y) \sqcup (-v \sqcap x \sqcap -z))$ **using** *1 3*

by (*smt inf-assoc inf-sup-distrib2 inf-top.right-neutral sup-compl-top*)
show *?thesis* **using** *1 2 3 4*
by (*metis compl-inter-eq*)
qed

lemma *maddux-3-21*: $x \sqcup y = x \sqcup (-x \sqcap y)$
by (*simp add*: *sup-compl-top sup-inf-distrib1*)

— Maddux Theorem 7(v)

lemma *shunting-1*: $x \leq y \leftrightarrow x \sqcap y\ ' = bot$
apply *rule*
apply (*smt compl-inf-bot inf-absorb1 inf-bot-right inf-commute inf-left-commute*)
apply (*metis inf-commute inf-sup-distrib1 inf-top-left le-iff-inf sup-commute sup-bot-left sup-compl-top*)
done

lemma *shunting-2*: $x \leq y \leftrightarrow x\ ' \sqcup y = top$
by (*metis compl-bot-eq compl-inf double-compl shunting-1*)

— Maddux Definition 13

definition *conjugate* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
where *conjugate f g* $\leftrightarrow (\forall x\ y\ .\ f\ x \sqcap y = bot \leftrightarrow x \sqcap g\ y = bot)$

— Maddux Theorem 14

lemma *conjugate-unique*: *conjugate f g* $\wedge$ *conjugate f h* $\rightarrow g = h$
proof
assume *conjugate f g* $\wedge$ *conjugate f h*
hence $\forall x\ y\ .\ g\ y \leq x\ ' \leftrightarrow h\ y \leq x\ '$
by (*smt double-compl inf-commute shunting-1 conjugate-def*)
hence $\forall y\ .\ g\ y = h\ y$
by (*metis double-compl eq-iff*)
thus $g = h$
by (*metis lifted-antisymmetric lifted-less-eq-def order-refl*)
qed

lemma *conjugate-symmetric*: *conjugate f g* $\rightarrow$ *conjugate g f*
by (*smt conjugate-def inf-commute*)

— Maddux Definition 15(iii)

definition *additive* :: $('a \Rightarrow 'a) \Rightarrow bool$
where *additive f* $\leftrightarrow (\forall x\ y\ .\ f\ (x \sqcup y) = f\ x \sqcup f\ y)$

— part of Maddux Theorem 17(i)

lemma *additive-isotone*: *additive f* $\longrightarrow$ *isotone f*
by (*metis additive-def isotone-def le-iff-sup*)

— part of Maddux Theorem 18(ii)

lemma *conjugate-additive*: *conjugate f g* $\longrightarrow$ *additive f*
proof
assume *1*: *conjugate f g*
have *2*: $\forall x\ y\ z\ .\ f\ (x \sqcup y) \leq z \leftrightarrow f\ x \leq z \wedge f\ y \leq z$
proof
fix *x*
show $\forall y\ z\ .\ f\ (x \sqcup y) \leq z \leftrightarrow f\ x \leq z \wedge f\ y \leq z$
proof
fix *y*
show $\forall z\ .\ f\ (x \sqcup y) \leq z \leftrightarrow f\ x \leq z \wedge f\ y \leq z$
proof
fix *z*
have $(f\ (x \sqcup y) \leq z) = (f\ (x \sqcup y) \sqcap z\ ' = bot)$
by (*metis shunting-1*)
also have ... $= ((x \sqcup y) \sqcap g(z\ ') = bot)$ **using** *1*
by (*metis conjugate-def*)
also have ... $= (x \sqcup y \leq (g(z\ '))')$
by (*metis double-compl shunting-1*)
also have ... $= (x \leq (g(z\ '))' \wedge y \leq (g(z\ '))')$
by (*metis le-sup-iff*)
also have ... $= (x \sqcap g(z\ ') = bot \wedge y \sqcap g(z\ ') = bot)$
by (*metis double-compl shunting-1*)
also have ... $= (f\ x \sqcap z\ ' = bot \wedge f\ y \sqcap z\ ' = bot)$ **using** *1*
by (*metis conjugate-def*)
also have ... $= (f\ x \leq z \wedge f\ y \leq z)$
by (*metis shunting-1*)
finally show $f\ (x \sqcup y) \leq z \leftrightarrow f\ x \leq z \wedge f\ y \leq z$
by *metis*
qed
qed
qed
have $\forall x\ y\ .\ f\ (x \sqcup y) = f\ x \sqcup f\ y$
proof
fix *x*
show $\forall y\ .\ f\ (x \sqcup y) = f\ x \sqcup f\ y$
proof
fix *y*
have $f(x \sqcup y) \leq f(x) \sqcup f(y)$ **using** *2*
by (*metis sup-ge1 sup-ge2*)
thus $f\ (x \sqcup y) = f\ x \sqcup f\ y$ **using** *2*
by (*metis le-supI order-refl antisym*)
qed

qed
thus *additive f*
by (*metis additive-def*)
qed

lemma *conjugate-isotone*: *conjugate f g* $\rightarrow$ *isotone f*
by (*metis additive-isotone conjugate-additive*)

— Maddux Theorem 19

lemma *conjugate-char-1*: *conjugate f g* $\leftrightarrow$ $(\forall\, x\ y\ .\ f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ ' \wedge g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ ')$
proof
assume *1*: *conjugate f g*
show $\forall\, x\ y\ .\ f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ ' \wedge g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ '$
proof
fix *x*
show $\forall\, y\ .\ f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ ' \wedge g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ '$
proof
fix *y*
have $f(x \sqcap (g\ y)') \leq y\ '$ **using** *1*
by (*smt compl-inf-bot conjugate-def double-compl inf-assoc inf-bot-right shunting-1*)
hence *2*: $f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ '$ **using** *1*
by (*metis conjugate-isotone inf-le1 isotone-def le-inf-iff*)
have $g(y \sqcap (f\ x)') \leq x\ '$ **using** *1*
by (*smt compl-inf-bot conjugate-def double-compl inf-assoc inf-bot-right shunting-1 inf-commute*)
hence $g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ '$ **using** *1*
by (*metis conjugate-isotone inf-le1 isotone-def le-inf-iff conjugate-symmetric*)
thus $f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ ' \wedge g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ '$ **using** *2*
by *metis*
qed
qed
next
assume *2*: $\forall\, x\ y\ .\ f(x \sqcap (g\ y)') \leq f\ x \sqcap y\ ' \wedge g(y \sqcap (f\ x)') \leq g\ y \sqcap x\ '$
hence *3*: $\forall\, x\ y\ .\ f\ x \sqcap y = bot \rightarrow x \sqcap g\ y = bot$
by (*smt double-compl inf-commute inf-le2 le-iff-inf shunting-1*)
have $\forall\, x\ y\ .\ x \sqcap g\ y = bot \rightarrow f\ x \sqcap y = bot$ **using** *2*
by (*smt double-compl inf-commute inf-le2 le-iff-inf shunting-1*)
thus *conjugate f g* **using** *3*
by (*metis conjugate-def*)
qed

lemma *conjugate-char-2*: *conjugate f g* $\leftrightarrow$ $f\ bot = bot \wedge g\ bot = bot \wedge (\forall\, x\ y\ .\ f\ x \sqcap y \leq f(x \sqcap g\ y) \wedge g\ y \sqcap x \leq g(y \sqcap f\ x))$
proof
assume *1*: *conjugate f g*
show $f\ bot = bot \wedge g\ bot = bot \wedge (\forall\, x\ y\ .\ f\ x \sqcap y \leq f(x \sqcap g\ y) \wedge g\ y \sqcap x \leq g(y \sqcap f\ x))$
proof
show $f\ bot = bot$ **using** *1*

```
    by (metis conjugate-def inf-idem inf-bot-left)
  next
    show g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
    proof
      show g bot = bot using 1
        by (metis conjugate-def inf-idem inf-bot-right)
    next
      show ∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
      proof
        fix x
        show ∀ y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
        proof
          fix y
          show f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
          proof
            have f x ⊓ y = (f(x ⊓ g y) ⊔ f(x ⊓ (g y)′)) ⊓ y using 1
              by (metis additive-def conjugate-additive inf-sup-distrib1 inf-top-right sup-compl-top)
            also have ... ≤ (f(x ⊓ g y) ⊔ (f x ⊓ y ′)) ⊓ y using 1
              by (metis conjugate-char-1 inf-mono order-refl sup-mono)
            also have ... ≤ f(x ⊓ g y)
              by (smt inf-idem inf-assoc inf-commute inf-compl-bot inf-sup-distrib1 le-iff-inf sup-commute sup-bot-left)
            finally show f x ⊓ y ≤ f(x ⊓ g y)
              by metis
          next
            have g y ⊓ x = (g(y ⊓ f x) ⊔ g(y ⊓ (f x)′)) ⊓ x using 1
              by (metis additive-def conjugate-additive conjugate-symmetric inf-sup-distrib1 inf-top-right sup-compl-top)
            also have ... ≤ (g(y ⊓ f x) ⊔ (g y ⊓ x ′)) ⊓ x using 1
              by (metis conjugate-char-1 inf-mono order-refl sup-mono)
            also have ... ≤ g(y ⊓ f x)
              by (smt inf-idem inf-assoc inf-commute inf-compl-bot inf-sup-distrib1 le-iff-inf sup-commute sup-bot-left)
            finally show g y ⊓ x ≤ g(y ⊓ f x)
              by metis
          qed
        qed
      qed
    qed
  qed
next
  assume f bot = bot ∧ g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
  thus conjugate f g
    by (smt conjugate-def inf-commute le-bot)
qed

end
```

— M0-algebra

class *lattice-ordered-pre-left-semiring* = *pre-left-semiring* + *bounded-distributive-lattice*

begin

subclass *bounded-pre-left-semiring*
  apply *unfold-locales*
  apply (*metis add-right-top-1*)
  done

lemma *top-mult-right-one*: $x \ ; \ T = x \ ; \ T \ ; \ 1$
  by (*metis add-commutative add-left-top less-eq-def mult-semi-associative mult-sub-right-one*)

lemma *mult-left-sub-dist-meet-left*: $x \ ; \ (y \frown z) \leq x \ ; \ y$
  by (*metis meet.add-left-upper-bound mult-right-isotone*)

lemma *mult-left-sub-dist-meet-right*: $x \ ; \ (y \frown z) \leq x \ ; \ z$
  by (*metis meet-commutative mult-left-sub-dist-meet-left*)

lemma *mult-right-sub-dist-meet-left*: $(x \frown y) \ ; \ z \leq x \ ; \ z$
  by (*metis meet.add-left-upper-bound mult-left-isotone*)

lemma *mult-right-sub-dist-meet-right*: $(x \frown y) \ ; \ z \leq y \ ; \ z$
  by (*metis meet.add-right-upper-bound mult-left-isotone*)

lemma *mult-right-sub-dist-meet*: $(x \frown y) \ ; \ z \leq x \ ; \ z \frown y \ ; \ z$
  by (*metis meet.add-least-upper-bound mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right*)

— Figure 1: fundamental properties

definition *total* :: $'a \Rightarrow$ *bool* **where** *total* $x \leftrightarrow x \ ; \ T = T$
definition *co-total* :: $'a \Rightarrow$ *bool* **where** *co-total* $x \leftrightarrow x \ ; \ 0 = 0$
definition *transitive* :: $'a \Rightarrow$ *bool* **where** *transitive* $x \leftrightarrow x \ ; \ x \leq x$
definition *dense* :: $'a \Rightarrow$ *bool* **where** *dense* $x \leftrightarrow x \leq x \ ; \ x$
definition *reflexive* :: $'a \Rightarrow$ *bool* **where** *reflexive* $x \leftrightarrow 1 \leq x$
definition *co-reflexive* :: $'a \Rightarrow$ *bool* **where** *co-reflexive* $x \leftrightarrow x \leq 1$
definition *idempotent* :: $'a \Rightarrow$ *bool* **where** *idempotent* $x \leftrightarrow x \ ; \ x = x$
definition *up-closed* :: $'a \Rightarrow$ *bool* **where** *up-closed* $x \leftrightarrow x \ ; \ 1 = x$
definition *add-distributive* :: $'a \Rightarrow$ *bool* **where** *add-distributive* $x \leftrightarrow (\forall \, y \ z \ . \ x \ ; \ (y + z) = x \ ; \ y + x \ ; \ z)$
definition *meet-distributive* :: $'a \Rightarrow$ *bool* **where** *meet-distributive* $x \leftrightarrow (\forall \, y \ z \ . \ x \ ; \ (y \frown z) = x \ ; \ y \frown x \ ; \ z)$
definition *contact* :: $'a \Rightarrow$ *bool* **where** *contact* $x \leftrightarrow x \ ; \ x + 1 = x$
definition *kernel* :: $'a \Rightarrow$ *bool* **where** *kernel* $x \leftrightarrow x \ ; \ x \frown 1 = x \ ; \ 1$
definition *add-dist-contact* :: $'a \Rightarrow$ *bool* **where** *add-dist-contact* $x \leftrightarrow$ *add-distributive* $x \wedge$ *contact* x
definition *meet-dist-kernel* :: $'a \Rightarrow$ *bool* **where** *meet-dist-kernel* $x \leftrightarrow$ *meet-distributive* $x \wedge$ *kernel* x
definition *test* :: $'a \Rightarrow$ *bool* **where** *test* $x \leftrightarrow x \ ; \ T \frown 1 = x$
definition *co-test* :: $'a \Rightarrow$ *bool* **where** *co-test* $x \leftrightarrow x \ ; \ 0 + 1 = x$

definition *co-vector* :: $'a \Rightarrow bool$ **where** *co-vector* $x \leftrightarrow x \ ; \ 0 = x$

— Theorem 6 / Figure 2: relations between properties

lemma *reflexive-total*: *reflexive* $x \longrightarrow$ *total* x
by (*metis eq-iff mult-isotone mult-left-one meet.zero-least reflexive-def total-def*)

lemma *reflexive-dense*: *reflexive* $x \longrightarrow$ *dense* x
by (*metis mult-left-isotone mult-left-one reflexive-def dense-def*)

lemma *reflexive-transitive-up-closed*: *reflexive* $x \wedge$ *transitive* $x \longrightarrow$ *up-closed* x
by (*metis antisym-conv mult-isotone mult-sub-right-one reflexive-def reflexive-dense transitive-def dense-def up-closed-def*)

lemma *co-reflexive-co-total*: *co-reflexive* $x \longrightarrow$ *co-total* x
by (*metis co-reflexive-def co-total-def eq-iff mult-left-isotone mult-left-one zero-least*)

lemma *co-reflexive-transitive*: *co-reflexive* $x \longrightarrow$ *transitive* x
by (*metis co-reflexive-def mult-left-isotone mult-left-one transitive-def*)

lemma *idempotent-transitive-dense*: *idempotent* $x \leftrightarrow$ *transitive* $x \wedge$ *dense* x
by (*metis eq-iff transitive-def dense-def idempotent-def*)

lemma *contact-reflexive*: *contact* $x \longrightarrow$ *reflexive* x
by (*metis contact-def add-right-upper-bound reflexive-def*)

lemma *contact-transitive*: *contact* $x \longrightarrow$ *transitive* x
by (*metis contact-def add-left-upper-bound transitive-def*)

lemma *contact-dense*: *contact* $x \longrightarrow$ *dense* x
by (*metis contact-reflexive reflexive-dense*)

lemma *contact-idempotent*: *contact* $x \longrightarrow$ *idempotent* x
by (*metis contact-transitive contact-dense idempotent-transitive-dense*)

lemma *contact-up-closed*: *contact* $x \longrightarrow$ *up-closed* x
by (*metis contact-def contact-idempotent dual-order.antisym mult-left-sub-dist-add-right mult-sub-right-one idempotent-def up-closed-def*)

lemma *contact-reflexive-idempotent-up-closed*: *contact* $x \leftrightarrow$ *reflexive* $x \wedge$ *idempotent* $x \wedge$ *up-closed* x
by (*metis contact-def contact-idempotent contact-up-closed add-commutative less-eq-def reflexive-def idempotent-def*)

lemma *kernel-co-reflexive*: *kernel* $x \longrightarrow$ *co-reflexive* x
by (*metis co-reflexive-def kernel-def meet.add-least-upper-bound mult-sub-right-one*)

lemma *kernel-transitive*: *kernel* $x \longrightarrow$ *transitive* x
by (*metis co-reflexive-transitive kernel-co-reflexive*)

lemma *kernel-dense*: *kernel* $x \longrightarrow$ *dense* x

by (*metis kernel-def meet.add-least-upper-bound mult-sub-right-one dense-def*)

lemma *kernel-idempotent*: *kernel x* ⟶ *idempotent x*
by (*metis kernel-transitive kernel-dense idempotent-transitive-dense*)

lemma *kernel-up-closed*: *kernel x* ⟶ *up-closed x*
by (*metis co-reflexive-def kernel-co-reflexive kernel-def kernel-idempotent meet-less-eq-def idempotent-def up-closed-def*)

lemma *kernel-co-reflexive-idempotent-up-closed*: *kernel x* ⟷ *co-reflexive x* ∧ *idempotent x* ∧ *up-closed x*
by (*metis co-reflexive-def kernel-def kernel-idempotent kernel-up-closed meet.less-eq-def meet-commutative idempotent-def up-closed-def*)

lemma *test-co-reflexive*: *test x* ⟶ *co-reflexive x*
by (*metis co-reflexive-def meet.add-right-upper-bound test-def*)

lemma *test-up-closed*: *test x* ⟶ *up-closed x*
by (*metis eq-iff mult-left-one mult-sub-right-one mult-right-sub-dist-meet test-def top-mult-right-one up-closed-def*)

lemma *co-test-reflexive*: *co-test x* ⟶ *reflexive x*
by (*metis co-test-def add-right-upper-bound reflexive-def*)

lemma *co-test-transitive*: *co-test x* ⟶ *transitive x*
by (*smt co-test-def add-associative less-eq-def mult-left-one mult-left-zero mult-right-dist-add mult-semi-associative transitive-def*)

lemma *co-test-idempotent*: *co-test x* ⟶ *idempotent x*
by (*metis co-test-reflexive co-test-transitive reflexive-dense idempotent-transitive-dense*)

lemma *co-test-up-closed*: *co-test x* ⟶ *up-closed x*
by (*metis co-test-reflexive co-test-idempotent contact-def contact-up-closed add-commutative less-eq-def reflexive-def idempotent-def*)

lemma *co-test-contact*: *co-test x* ⟶ *contact x*
by (*metis co-test-reflexive co-test-idempotent co-test-up-closed contact-reflexive-idempotent-up-closed*)

lemma *vector-transitive*: *vector x* ⟶ *transitive x*
by (*metis mult-right-isotone meet.zero-least vector-def transitive-def*)

lemma *vector-up-closed*: *vector x* ⟶ *up-closed x*
by (*metis vector-def top-mult-right-one up-closed-def*)

— Theorem 10 / Figure 3: closure properties

— total

lemma *one-total*: *total 1*
by (*metis mult-left-one total-def*)

lemma *top-total*: *total T*
by (*metis top-mult-top total-def*)

lemma *add-total*: *total x* ∧ *total y* → *total* (*x* + *y*)
by (*metis add-left-top mult-right-dist-add total-def*)

— co-total

lemma *zero-co-total*: *co-total 0*
by (*metis co-total-def mult-left-zero*)

lemma *one-co-total*: *co-total 1*
by (*metis co-total-def mult-left-one*)

lemma *add-co-total*: *co-total x* ∧ *co-total y* → *co-total* (*x* + *y*)
by (*metis co-total-def add-right-zero mult-right-dist-add*)

lemma *meet-co-total*: *co-total x* ∧ *co-total y* → *co-total* (*x* ⌢ *y*)
by (*metis co-total-def add-left-zero antisym-conv less-eq-def mult-right-sub-dist-meet-left*)

lemma *comp-co-total*: *co-total x* ∧ *co-total y* → *co-total* (*x* ; *y*)
by (*metis co-total-def eq-iff mult-semi-associative zero-least*)

— sub-transitive

lemma *zero-transitive*: *transitive 0*
by (*metis mult-left-zero zero-least transitive-def*)

lemma *one-transitive*: *transitive 1*
by (*metis mult-left-one order-refl transitive-def*)

lemma *top-transitive*: *transitive T*
by (*metis meet.zero-least transitive-def*)

lemma *meet-transitive*: *transitive x* ∧ *transitive y* → *transitive* (*x* ⌢ *y*)
by (*smt meet.less-eq-def meet-associative meet-commutative mult-left-sub-dist-meet-left mult-right-sub-dist-meet-left transitive-def*)

— dense

lemma *zero-dense*: *dense 0*
by (*metis zero-least dense-def*)

lemma *one-dense*: *dense 1*
by (*metis mult-sub-right-one dense-def*)

lemma *top-dense*: *dense T*
by (*metis top-left-mult-increasing dense-def*)

lemma *add-dense*: *dense x* ∧ *dense y* → *dense* (*x* + *y*)

proof
  assume *dense* $x \wedge$ *dense* y
  hence $x \leq x$; $x \wedge y \leq y$; y
    by (*metis dense-def*)
  hence $x \leq (x + y)$; $(x + y) \wedge y \leq (x + y)$; $(x + y)$
    by (*metis add-left-upper-bound dual-order.trans mult-isotone add-right-upper-bound*)
  hence $x + y \leq (x + y)$; $(x + y)$
    by (*metis add-least-upper-bound*)
  thus *dense* $(x + y)$
    by (*metis dense-def*)
qed

— reflexive

lemma *one-reflexive*: *reflexive 1*
  by (*metis order-refl reflexive-def*)

lemma *top-reflexive*: *reflexive T*
  by (*metis meet.zero-least reflexive-def*)

lemma *add-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \rightarrow$ *reflexive* $(x + y)$
  by (*metis add-associative less-eq-def reflexive-def*)

lemma *meet-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \rightarrow$ *reflexive* $(x \frown y)$
  by (*metis meet.add-least-upper-bound reflexive-def*)

lemma *comp-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \rightarrow$ *reflexive* $(x ; y)$
  by (*metis mult-left-isotone mult-left-one order-trans reflexive-def*)

— co-reflexive

lemma *zero-co-reflexive*: *co-reflexive 0*
  by (*metis co-reflexive-def zero-least*)

lemma *one-co-reflexive*: *co-reflexive 1*
  by (*metis co-reflexive-def order-refl*)

lemma *add-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \rightarrow$ *co-reflexive* $(x + y)$
  by (*metis co-reflexive-def add-least-upper-bound*)

lemma *meet-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \rightarrow$ *co-reflexive* $(x \frown y)$
  by (*metis co-reflexive-def meet.less-eq-def meet-associative*)

lemma *comp-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \rightarrow$ *co-reflexive* $(x ; y)$
  by (*metis co-reflexive-def mult-isotone mult-left-one*)

— idempotent

lemma *zero-idempotent*: *idempotent 0*
by (*metis mult-left-zero idempotent-def*)

lemma *one-idempotent*: *idempotent 1*
by (*metis mult-left-one idempotent-def*)

lemma *top-idempotent*: *idempotent T*
by (*metis top-mult-top idempotent-def*)

— up-closed

lemma *zero-up-closed*: *up-closed 0*
by (*metis mult-left-zero up-closed-def*)

lemma *one-up-closed*: *up-closed 1*
by (*metis mult-left-one up-closed-def*)

lemma *top-up-closed*: *up-closed T*
by (*metis top-mult-top vector-def vector-up-closed*)

lemma *add-up-closed*: *up-closed x* $\wedge$ *up-closed y* $\longrightarrow$ *up-closed* $(x + y)$
by (*metis mult-right-dist-add up-closed-def*)

lemma *meet-up-closed*: *up-closed x* $\wedge$ *up-closed y* $\longrightarrow$ *up-closed* $(x \frown y)$
by (*metis dual-order.antisym mult-sub-right-one mult-right-sub-dist-meet up-closed-def*)

lemma *comp-up-closed*: *up-closed x* $\wedge$ *up-closed y* $\longrightarrow$ *up-closed* $(x\ ;\ y)$
by (*metis dual-order.antisym mult-semi-associative mult-sub-right-one up-closed-def*)

— add-distributive

lemma *zero-add-distributive*: *add-distributive 0*
by (*metis add-distributive-def add-idempotent mult-left-zero*)

lemma *one-add-distributive*: *add-distributive 1*
by (*metis add-distributive-def mult-left-one*)

lemma *add-add-distributive*: *add-distributive x* $\wedge$ *add-distributive y* $\longrightarrow$ *add-distributive* $(x + y)$
by (*smt add-distributive-def add-associative add-commutative mult-right-dist-add*)

— meet-distributive

lemma *zero-meet-distributive*: *meet-distributive 0*
by (*metis meet-left-zero mult-left-zero meet-distributive-def*)

lemma *one-meet-distributive*: *meet-distributive 1*

by (*metis mult-left-one meet-distributive-def*)

— contact

lemma *one-contact*: *contact 1*
by (*metis contact-def add-idempotent mult-left-one*)

lemma *top-contact*: *contact T*
by (*metis contact-def add-left-top top-mult-top*)

lemma *meet-contact*: *contact x* ∧ *contact y* → *contact* (*x* ⌢ *y*)
by (*smt contact-def contact-reflexive contact-transitive contact-up-closed meet.less-eq-def meet-commutative meet-left-dist-add mult-left-sub-dist-add-right meet-transitive meet-up-closed reflexive-def transitive-def up-closed-def*)

— kernel

lemma *zero-kernel*: *kernel 0*
by (*metis kernel-co-reflexive-idempotent-up-closed zero-co-reflexive zero-idempotent zero-up-closed*)

lemma *one-kernel*: *kernel 1*
by (*metis kernel-def meet-idempotent mult-left-one*)

lemma *add-kernel*: *kernel x* ∧ *kernel y* → *kernel* (*x* + *y*)
by (*metis add-co-reflexive add-dense add-up-closed co-reflexive-transitive kernel-co-reflexive-idempotent-up-closed idempotent-transitive-dense*)

— add-distributive contact

lemma *one-add-dist-contact*: *add-dist-contact 1*
by (*metis add-dist-contact-def one-add-distributive one-contact*)

— meet-distributive kernel

lemma *zero-meet-dist-kernel*: *meet-dist-kernel 0*
by (*metis meet-dist-kernel-def zero-kernel zero-meet-distributive*)

lemma *one-meet-dist-kernel*: *meet-dist-kernel 1*
by (*metis meet-dist-kernel-def one-kernel one-meet-distributive*)

— test

lemma *zero-test*: *test 0*
by (*metis meet-commutative meet-right-zero mult-left-zero test-def*)

lemma *one-test*: *test 1*
by (*metis meet-left-top mult-left-one test-def*)

lemma *add-test*: *test x* ∧ *test y* → *test* (*x* + *y*)

by (*metis* (*no-types*, *lifting*) *meet-commutative meet-left-dist-add mult-right-dist-add test-def*)

lemma *meet-test*: *test x* ∧ *test y* → *test* (*x* ⌢ *y*)
by (*smt test-def meet-commutative meet.add-least-upper-bound meet.add-right-isotone mult-right-sub-dist-meet-left meet.add-left-upper-bound top-right-mult-increasing antisym*)

— co-test

lemma *one-co-test*: *co-test 1*
by (*metis co-test-def co-total-def add-left-zero one-co-total*)

lemma *add-co-test*: *co-test x* ∧ *co-test y* → *co-test* (*x* + *y*)
by (*smt co-test-contact co-test-def contact-def add-associative add-commutative add-left-zero mult-left-one mult-right-dist-add*)

— vector

lemma *zero-vector*: *vector 0*
by (*metis mult-left-zero vector-def*)

lemma *top-vector*: *vector T*
by (*metis top-mult-top vector-def*)

lemma *add-vector*: *vector x* ∧ *vector y* → *vector* (*x* + *y*)
by (*metis mult-right-dist-add vector-def*)

lemma *meet-vector*: *vector x* ∧ *vector y* → *vector* (*x* ⌢ *y*)
by (*metis antisym meet.add-least-upper-bound mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right top-right-mult-increasing vector-def*)

lemma *comp-vector*: *vector y* → *vector* (*x* ; *y*)
by (*metis antisym-conv mult-semi-associative top-right-mult-increasing vector-def*)

end

class *lattice-ordered-pre-left-semiring-1* = *non-associative-left-semiring* + *bounded-distributive-lattice* +
assumes *mult-associative-one*: *x* ; (*y* ; *z*) = (*x* ; (*y* ; *1*)) ; *z*
assumes *mult-right-dist-meet-one*: (*x* ; *1* ⌢ *y* ; *1*) ; *z* = *x* ; *z* ⌢ *y* ; *z*

begin

subclass *pre-left-semiring*
apply *unfold-locales*
apply (*metis mult-associative-one mult-left-isotone mult-right-isotone mult-sub-right-one*)
done

subclass *lattice-ordered-pre-left-semiring*
..

lemma *mult-zero-associative*: *x* ; *0* ; *y* = *x* ; *0*

by (*smt mult-left-zero mult-associative-one*)

lemma *mult-zero-add-one-dist*: $(x\ ;\ 0 + 1)\ ;\ z = x\ ;\ 0 + z$
by (*metis mult-left-one mult-right-dist-add mult-zero-associative*)

lemma *mult-zero-add-dist*: $(x\ ;\ 0 + y)\ ;\ z = x\ ;\ 0 + y\ ;\ z$
by (*metis mult-right-dist-add mult-zero-associative*)

lemma *vector-zero-meet-one-comp*: $(x\ ;\ 0 \frown 1)\ ;\ y = x\ ;\ 0 \frown y$
by (*metis mult-left-one mult-right-dist-meet-one mult-zero-associative*)

— Theorem 6 / Figure 2: relations between properties

lemma *co-test-meet-distributive*: *co-test* $x \longrightarrow$ *meet-distributive* x
by (*metis add-left-dist-meet co-test-def meet-distributive-def mult-zero-add-one-dist*)

lemma *co-test-add-distributive*: *co-test* $x \longrightarrow$ *add-distributive* x
by (*smt add-associative add-commutative add-distributive-def add-left-upper-bound co-test-def less-eq-def mult-zero-add-one-dist*)

lemma *co-test-add-dist-contact*: *co-test* $x \longrightarrow$ *add-dist-contact* x
by (*metis co-test-add-distributive add-dist-contact-def co-test-contact*)

— Theorem 10 / Figure 3: closure properties

— co-test

lemma *meet-co-test*: *co-test* $x \wedge$ *co-test* $y \longrightarrow$ *co-test* $(x \frown y)$
by (*smt add-commutative add-left-dist-meet co-test-def co-test-up-closed up-closed-def mult-right-dist-meet-one*)

lemma *comp-co-test*: *co-test* $x \wedge$ *co-test* $y \longrightarrow$ *co-test* $(x\ ;\ y)$
by (*metis add-associative co-test-def mult-zero-add-dist mult-zero-add-one-dist*)

end

class *lattice-ordered-pre-left-semiring-2* = *lattice-ordered-pre-left-semiring* +
assumes *mult-sub-associative-one*: $x\ ;\ (y\ ;\ z) \leq (x\ ;\ (y\ ;\ 1))\ ;\ z$
assumes *mult-right-dist-meet-one-sub*: $x\ ;\ z \frown y\ ;\ z \leq (x\ ;\ 1 \frown y\ ;\ 1)\ ;\ z$

begin

subclass *lattice-ordered-pre-left-semiring-1*
apply *unfold-locales*
apply (*metis meet.eq-iff mult-sub-associative-one mult-sup-associative-one*)
apply (*metis meet.antisym-conv mult-one-associative mult-right-dist-meet-one-sub mult-right-sub-dist-meet*)
done

end

class *multirelation-algebra-1* = *lattice-ordered-pre-left-semiring* +
  assumes *mult-left-top*: $T \ ; \ x = T$

begin

— Theorem 10 / Figure 3: closure properties

lemma *top-add-distributive*: *add-distributive* T
  by (*metis add-distributive-def add-left-top mult-left-top*)

lemma *top-meet-distributive*: *meet-distributive* T
  by (*metis meet-idempotent meet-distributive-def mult-left-top*)

lemma *top-add-dist-contact*: *add-dist-contact* T
  by (*metis add-dist-contact-def top-add-distributive top-contact*)

lemma *top-co-test*: *co-test* T
  by (*metis co-test-def add-left-top mult-left-top*)

end

— M1-algebra

class *multirelation-algebra-2* = *multirelation-algebra-1* + *lattice-ordered-pre-left-semiring-2*

begin

lemma *mult-top-associative*: $x \ ; \ T \ ; \ y = x \ ; \ T$
  by (*metis mult-left-top mult-associative-one*)

lemma *vector-meet-one-comp*: $(x \ ; \ T \sqinter 1) \ ; \ y = x \ ; \ T \sqinter y$
  by (*metis mult-left-one mult-left-top mult-associative-one mult-right-dist-meet-one*)

lemma *vector-left-annihilator*: *vector* $x \rightarrow x \ ; \ y = x$
  by (*metis mult-left-top vector-def mult-associative-one*)

— properties

lemma *test-comp-meet*: *test* $x \wedge$ *test* $y \rightarrow x \ ; \ y = x \sqinter y$
  by (*smt meet-associative meet-commutative meet-idempotent test-def vector-meet-one-comp*)

— Theorem 6 / Figure 2: relations between properties

lemma *test-add-distributive*: *test* $x \rightarrow$ *add-distributive* x
  by (*metis add-distributive-def meet-left-dist-add test-def vector-meet-one-comp*)

lemma *test-meet-distributive*: *test x* $\rightarrow$ *meet-distributive x*
by (*smt meet.less-eq-def meet-associative meet-commutative meet-distributive-def meet.add-right-upper-bound mult-left-one test-def vector-meet-one-comp*)

lemma *test-meet-dist-kernel*: *test x* $\rightarrow$ *meet-dist-kernel x*
by (*metis kernel-co-reflexive-idempotent-up-closed meet-associative meet-dist-kernel-def meet-idempotent test-co-reflexive test-def test-up-closed idempotent-def vector-meet-one-comp test-meet-distributive*)

lemma *vector-idempotent*: *vector x* $\rightarrow$ *idempotent x*
by (*metis idempotent-def vector-left-annihilator*)

lemma *vector-add-distributive*: *vector x* $\rightarrow$ *add-distributive x*
by (*metis add-distributive-def add-idempotent vector-left-annihilator*)

lemma *vector-meet-distributive*: *vector x* $\rightarrow$ *meet-distributive x*
by (*metis meet-distributive-def meet-idempotent vector-left-annihilator*)

lemma *vector-co-vector*: *vector x* $\leftrightarrow$ *co-vector x*
by (*metis co-vector-def vector-def mult-zero-associative vector-left-annihilator*)

— Theorem 10 / Figure 3: closure properties

— test

lemma *comp-test*: *test x* $\wedge$ *test y* $\rightarrow$ *test* (*x* ; *y*)
by (*metis meet-associative meet-distributive-def meet.add-right-zero test-def test-up-closed up-closed-def mult-associative-one test-meet-distributive*)

end

class *dual* =
fixes *dual* :: $'a \Rightarrow 'a$ ($-^d$ [100] 100)

class *multirelation-algebra-3* = *lattice-ordered-pre-left-semiring* + *dual* +
assumes *dual-involutive*: $x^{dd} = x$
assumes *dual-dist-add*: $(x + y)^d = x^d \frown y^d$
assumes *dual-one*: $1^d = 1$

begin

lemma *dual-dist-meet*: $(x \frown y)^d = x^d + y^d$
by (*metis dual-dist-add dual-involutive*)

lemma *dual-antitone*: $x \leq y \rightarrow y^d \leq x^d$
by (*metis dual-dist-meet add-left-divisibility meet.add-left-divisibility*)

lemma *dual-zero*: $0^d = T$
by (*metis dual-dist-meet add-right-top dual-involutive meet-left-zero*)

lemma *dual-top*: $T^d = 0$
by (*metis dual-zero dual-involutive*)

— Theorem 10 / Figure 3: closure properties

lemma *reflexive-co-reflexive-dual*: *reflexive* $x \leftrightarrow$ *co-reflexive* (x^d)
by (*metis co-reflexive-def dual-antitone dual-involutive dual-one reflexive-def*)

end

class *multirelation-algebra-4* = *multirelation-algebra-3* +
assumes *dual-sub-dist-comp*: $(x \ ; \ y)^d \leq x^d \ ; \ y^d$

begin

subclass *multirelation-algebra-1*
apply *unfold-locales*
apply (*metis dual-zero dual-sub-dist-comp dual-involutive meet.less-eq-def meet-commutative meet-left-top mult-left-zero*)
done

lemma *dual-sub-dist-comp-one*: $(x \ ; \ y)^d \leq (x \ ; \ 1)^d \ ; \ y^d$
by (*metis dual-sub-dist-comp mult-one-associative*)

— Theorem 10 / Figure 3: closure properties

lemma *co-total-total-dual*: *co-total* $x \rightarrow$ *total* (x^d)
by (*metis co-total-def dual-sub-dist-comp dual-zero meet.less-eq-def meet-commutative meet-left-top total-def*)

lemma *transitive-dense-dual*: *transitive* $x \rightarrow$ *dense* (x^d)
by (*metis dual-antitone dual-sub-dist-comp order-trans transitive-def dense-def*)

end

— M2-algebra

class *multirelation-algebra-5* = *multirelation-algebra-3* +
assumes *dual-dist-comp-one*: $(x \ ; \ y)^d = (x \ ; \ 1)^d \ ; \ y^d$

begin

subclass *multirelation-algebra-4*
apply *unfold-locales*
apply (*metis dual-antitone mult-sub-right-one mult-left-isotone dual-dist-comp-one*)
done

lemma *strong-up-closed*: $x \ ; \ 1 \leq x \rightarrow x^d \ ; \ y^d \leq (x \ ; \ y)^d$
by (*metis dual-dist-comp-one eq-iff mult-sub-right-one*)

lemma *strong-up-closed-2*: *up-closed* $x \rightarrow (x\ ;\ y)^d = x^d\ ;\ y^d$
by (*metis dual-sub-dist-comp eq-iff strong-up-closed up-closed-def*)

subclass *lattice-ordered-pre-left-semiring-2*
apply *unfold-locales*
apply (*smt comp-up-closed dual-antitone dual-dist-comp-one dual-involutive dual-one mult-left-one mult-one-associative mult-semi-associative up-closed-def strong-up-closed-2*)
apply (*smt dual-dist-comp-one dual-dist-meet dual-involutive eq-refl mult-one-associative mult-right-dist-add*)
done

— Theorem 8

subclass *multirelation-algebra-2*
..

— Theorem 10 / Figure 3: closure properties

— up-closed

lemma *dual-up-closed*: *up-closed* $x \leftrightarrow$ *up-closed* (x^d)
by (*metis dual-involutive dual-one up-closed-def strong-up-closed-2*)

— contact

lemma *contact-kernel-dual*: *contact* $x \leftrightarrow$ *kernel* (x^d)
by (*metis contact-def contact-up-closed dual-dist-add dual-involutive dual-one kernel-def kernel-up-closed up-closed-def strong-up-closed-2*)

— add-distributive contact

lemma *add-dist-contact-meet-dist-kernel-dual*: *add-dist-contact* $x \leftrightarrow$ *meet-dist-kernel* (x^d)
proof
assume *1*: *add-dist-contact x*
hence *2*: *up-closed x*
by (*metis add-dist-contact-def contact-up-closed*)
have *add-distributive x* **using** *1*
by (*metis add-dist-contact-def*)
hence *meet-distributive* (x^d) **using** *2*
by (*smt add-distributive-def dual-dist-comp-one dual-dist-meet dual-involutive meet-distributive-def up-closed-def*)
thus *meet-dist-kernel* (x^d) **using** *1*
by (*metis contact-kernel-dual add-dist-contact-def meet-dist-kernel-def*)
next
assume *3*: *meet-dist-kernel* (x^d)
hence *2*: *up-closed* (x^d)
by (*metis kernel-up-closed meet-dist-kernel-def*)
have *meet-distributive* (x^d) **using** *3*
by (*metis meet-dist-kernel-def*)
hence *add-distributive* (x^{dd}) **using** *2*

by (*smt meet-distributive-def add-distributive-def dual-dist-add dual-involutive strong-up-closed-2*)
thus *add-dist-contact x* **using** *3*
by (*metis contact-kernel-dual add-dist-contact-def meet-dist-kernel-def dual-involutive*)
qed

— test

lemma *test-co-test-dual*: *test x* ↔ *co-test* (x^d)
by (*smt co-test-def co-test-up-closed dual-dist-meet dual-involutive dual-one dual-top test-def test-up-closed strong-up-closed-2*)

— vector

lemma *vector-dual*: *vector x* ↔ *vector* (x^d)
by (*metis dual-dist-comp-one comp-vector dual-involutive dual-top vector-def zero-vector*)

end

class *multirelation-algebra-6* = *multirelation-algebra-4* +
assumes *dual-sub-dist-comp-one*: $(x\ ;\ 1)^d\ ;\ y^d \leq (x\ ;\ y)^d$

begin

subclass *multirelation-algebra-5*
apply *unfold-locales*
apply (*metis dual-sub-dist-comp dual-sub-dist-comp-one meet.eq-iff mult-one-associative*)
done

end

— M3-algebra

class *up-closed-multirelation-algebra* = *multirelation-algebra-3* +
assumes *dual-dist-comp*: $(x\ ;\ y)^d = x^d\ ;\ y^d$

begin

lemma *mult-right-dist-meet*: $(x \frown y)\ ;\ z = x\ ;\ z \frown y\ ;\ z$
by (*metis dual-dist-add dual-dist-comp dual-involutive mult-right-dist-add*)

— Theorem 9

subclass *idempotent-left-semiring*
apply *unfold-locales*
apply (*metis antisym dual-antitone dual-dist-comp dual-involutive mult-semi-associative*)

apply (*metis mult-left-one*)
apply (*metis dual-dist-add dual-dist-comp dual-involutive dual-one less-eq-def meet-absorb mult-sub-right-one*)
done

subclass *multirelation-algebra-6*
apply *unfold-locales*
apply (*metis dual-dist-comp eq-iff*)
apply (*metis dual-dist-comp eq-iff mult-right-one*)
done

lemma *vector-meet-comp*: $(x\ ;\ T \frown y)\ ;\ z = x\ ;\ T \frown y\ ;\ z$
by (*metis mult-associative mult-left-top mult-right-dist-meet*)

lemma *vector-zero-meet-comp*: $(x\ ;\ 0 \frown y)\ ;\ z = x\ ;\ 0 \frown y\ ;\ z$
by (*metis mult-associative mult-left-zero mult-right-dist-meet*)

— Theorem 10 / Figure 3: closure properties

— total

lemma *meet-total*: $total\ x \wedge total\ y \longrightarrow total\ (x \frown y)$
by (*metis meet-left-top total-def mult-right-dist-meet*)

lemma *comp-total*: $total\ x \wedge total\ y \longrightarrow total\ (x\ ;\ y)$
by (*metis mult-associative total-def*)

lemma *total-co-total-dual*: $total\ x \longleftrightarrow co\text{-}total\ (x^d)$
by (*metis co-total-def dual-dist-comp dual-involutive dual-top total-def*)

— dense

lemma *transitive-iff-dense-dual*: $transitive\ x \longleftrightarrow dense\ (x^d)$
by (*metis dense-def dual-antitone dual-dist-comp dual-involutive transitive-def*)

— idempotent

lemma *idempotent-dual*: $idempotent\ x \longleftrightarrow idempotent\ (x^d)$
by (*metis dual-involutive idempotent-transitive-dense transitive-iff-dense-dual*)

— add-distributive

lemma *comp-add-distributive*: $add\text{-}distributive\ x \wedge add\text{-}distributive\ y \longrightarrow add\text{-}distributive\ (x\ ;\ y)$
by (*metis add-distributive-def mult-associative*)

lemma *add-meet-distributive-dual*: $add\text{-}distributive\ x \longleftrightarrow meet\text{-}distributive\ (x^d)$
by (*metis* (*no-types*, *hide-lams*) *add-distributive-def dual-dist-add dual-dist-comp dual-involutive meet-distributive-def*)

— meet-distributive

lemma *meet-meet-distributive*: *meet-distributive x* ∧ *meet-distributive y* → *meet-distributive* (*x* ⌢ *y*)
by (*smt meet-distributive-def meet-associative meet-commutative mult-right-dist-meet*)

lemma *comp-meet-distributive*: *meet-distributive x* ∧ *meet-distributive y* → *meet-distributive* (*x* ; *y*)
by (*metis meet-distributive-def mult-associative*)

end

class *multirelation-algebra-7* = *multirelation-algebra-4* +
assumes *vector-meet-comp*: (*x* ; *T* ⌢ *y*) ; *z* = *x* ; *T* ⌢ *y* ; *z*

begin

lemma *vector-zero-meet-comp*: (*x* ; *0* ⌢ *y*) ; *z* = *x* ; *0* ⌢ *y* ; *z*
by (*metis vector-def comp-vector vector-meet-comp zero-vector*)

lemma *test-add-distributive*: *test x* → *add-distributive x*
by (*metis add-distributive-def meet-left-dist-add mult-left-one test-def vector-meet-comp*)

lemma *test-meet-distributive*: *test x* → *meet-distributive x*
by (*smt meet.less-eq-def meet-associative meet-commutative meet-distributive-def meet.add-right-upper-bound mult-left-one test-def vector-meet-comp*)

lemma *test-meet-dist-kernel*: *test x* → *meet-dist-kernel x*
by (*metis kernel-co-reflexive-idempotent-up-closed meet-associative meet-dist-kernel-def meet-idempotent mult-left-one test-co-reflexive test-def test-up-closed idempotent-def vector-meet-comp test-meet-distributive*)

lemma *co-test-meet-distributive*: *co-test x* → *meet-distributive x*
proof
assume *co-test x*
hence *x* = *x* ; *0* + *1*
by (*metis co-test-def*)
hence ∀ *y z* . *x* ; *y* ⌢ *x* ; *z* = *x* ; (*y* ⌢ *z*)
by (*metis mult-left-one mult-left-top mult-right-dist-add meet.add-right-zero vector-zero-meet-comp add-left-dist-meet*)
thus *meet-distributive x*
by (*metis meet-distributive-def*)
qed

lemma *co-test-add-distributive*: *co-test x* → *add-distributive x*
proof
assume *co-test x*
hence *1*: *x* = *x* ; *0* + *1*

by (*metis co-test-def*)
hence $\forall\, y\ z\ .\ x\ ;\ (y + z) = x\ ;\ y + x\ ;\ z$
by (*metis add-associative add-commutative add-idempotent mult-left-one mult-left-top mult-right-dist-add meet.add-right-zero vector-zero-meet-comp*)
thus *add-distributive x*
by (*metis add-distributive-def*)
qed

lemma *co-test-add-dist-contact*: *co-test x* $\longrightarrow$ *add-dist-contact x*
by (*metis co-test-add-distributive add-dist-contact-def co-test-contact*)

end

class *complemented-distributive-lattice* = *bounded-distributive-lattice* + *uminus* +
assumes *meet-complement*: $x \frown (-x) = 0$
assumes *add-complement*: $x + (-x) = T$

begin

sublocale *boolean-algebra* **where** *minus* = $\lambda x\ y\ .\ x \frown (-y)$ **and** *inf* = *meet* **and** *sup* = *plus* **and** *bot* = *0* **and** *top* = T
apply *unfold-locales*
apply (*simp add*: *meet.add-left-upper-bound*)
apply (*simp add*: *meet.add-right-upper-bound*)
using *meet.add-least-upper-bound* **apply** *blast*
apply (*simp add*: *add-left-upper-bound*)
apply (*simp add*: *add-right-upper-bound*)
using *add-least-upper-bound* **apply** *blast*
apply (*simp add*: *zero-least*)
apply (*simp add*: *meet.zero-least*)
apply (*simp add*: *add-left-dist-meet*)
apply (*simp add*: *meet-complement*)
apply (*simp add*: *add-complement*)
apply *simp*
done

end

— M0-algebra

context *lattice-ordered-pre-left-semiring*

begin

— Section 7

lemma *vector-1*: *vector x* $\leftrightarrow$ $x\ ;\ T \leq x$

by (*simp add*: *meet.eq-iff top-right-mult-increasing vector-def*)

definition *zero-vector* :: *'a* ⇒ *bool* **where** *zero-vector x* ↔ *x* ≤ *x* ; *0*
definition *one-vector* :: *'a* ⇒ *bool* **where** *one-vector x* ↔ *x* ; *0* ≤ *x*

lemma *zero-vector-left-zero*: *zero-vector x* → *x* ; *y* = *x* ; *0*
proof −
have *zero-vector x* → *x* ; *y* ≤ *x* ; *0*
using *mult-isotone top-greatest vector-def vector-left-mult-closed zero-vector zero-vector-def* **by** *fastforce*
thus *?thesis*
by (*simp add*: *meet.antisym mult-right-isotone zero-least*)
qed

lemma *zero-vector-1*: *zero-vector x* ↔ (∀ *y* . *x* ; *y* = *x* ; *0*)
by (*metis top-right-mult-increasing zero-vector-def zero-vector-left-zero*)

lemma *zero-vector-2*: *zero-vector x* ↔ (∀ *y* . *x* ; *y* ≤ *x* ; *0*)
by (*simp add*: *meet.eq-iff mult-right-isotone zero-least zero-vector-1*)

lemma *zero-vector-3*: *zero-vector x* ↔ *x* ; *1* = *x* ; *0*
by (*metis mult-sub-right-one zero-vector-def zero-vector-left-zero*)

lemma *zero-vector-4*: *zero-vector x* ↔ *x* ; *1* ≤ *x* ; *0*
by (*simp add*: *meet.eq-iff mult-right-isotone zero-least zero-vector-3*)

lemma *zero-vector-5*: *zero-vector x* ↔ *x* ; *T* = *x* ; *0*
by (*metis top-right-mult-increasing zero-vector-def zero-vector-left-zero*)

lemma *zero-vector-6*: *zero-vector x* ↔ *x* ; *T* ≤ *x* ; *0*
by (*simp add*: *meet.eq-iff mult-right-isotone top-greatest zero-vector-5*)

lemma *zero-vector-7*: *zero-vector x* ↔ (∀ *y* . *x* ; *T* = *x* ; *y*)
by (*metis zero-vector-5 zero-vector-left-zero*)

lemma *zero-vector-8*: *zero-vector x* ↔ (∀ *y* . *x* ; *T* ≤ *x* ; *y*)
by (*metis zero-vector-6 zero-vector-left-zero*)

lemma *zero-vector-9*: *zero-vector x* ↔ (∀ *y* . *x* ; *1* = *x* ; *y*)
by (*metis zero-vector-1*)

lemma *zero-vector-0*: *zero-vector x* ↔ (∀ *y z* . *x* ; *y* = *x* ; *z*)
by (*metis zero-vector-5 zero-vector-left-zero*)

— Theorem 6 / Figure 2: relations between properties

lemma *co-vector-zero-vector-one-vector*: *co-vector x* ↔ *zero-vector x* ∧ *one-vector x*
by (*simp add*: *co-vector-def meet.eq-iff one-vector-def zero-vector-def*)

lemma *up-closed-one-vector*: *up-closed x* ⟶ *one-vector x*
by (*metis mult-right-isotone up-closed-def zero-least one-vector-def*)

lemma *zero-vector-dense*: *zero-vector x* ⟶ *dense x*
by (*metis dense-def zero-vector-def zero-vector-left-zero*)

lemma *zero-vector-add-distributive*: *zero-vector x* ⟶ *add-distributive x*
by (*metis add-distributive-def add-idempotent zero-vector-left-zero*)

lemma *zero-vector-meet-distributive*: *zero-vector x* ⟶ *meet-distributive x*
by (*metis meet-distributive-def meet-idempotent zero-vector-left-zero*)

lemma *up-closed-zero-vector-vector*: *up-closed x* ∧ *zero-vector x* ⟶ *vector x*
by (*metis up-closed-def zero-vector-8 vector-1*)

lemma *zero-vector-one-vector-vector*: *zero-vector x* ∧ *one-vector x* ⟶ *vector x*
by (*simp add*: *one-vector-def vector-1 zero-vector-5*)

lemma *co-vector-vector*: *co-vector x* ⟶ *vector x*
by (*simp add*: *co-vector-zero-vector-one-vector zero-vector-one-vector-vector*)

— Theorem 10 / Figure 3: closure properties

— zero-vector

lemma *zero-zero-vector*: *zero-vector 0*
by (*simp add*: *mult-left-zero zero-vector-def*)

lemma *add-zero-vector*: *zero-vector x* ∧ *zero-vector y* ⟶ *zero-vector* (*x* + *y*)
by (*simp add*: *mult-right-dist-add zero-vector-5*)

lemma *comp-zero-vector*: *zero-vector x* ∧ *zero-vector y* ⟶ *zero-vector* (*x* ; *y*)
by (*metis mult-one-associative zero-vector-0*)

— one-vector

lemma *zero-one-vector*: *one-vector 0*
by (*simp add*: *zero-up-closed up-closed-one-vector*)

lemma *one-one-vector*: *one-vector 1*
by (*simp add*: *one-up-closed up-closed-one-vector*)

lemma *top-one-vector*: *one-vector T*
by (*simp add*: *top-greatest one-vector-def*)

lemma *add-one-vector*: *one-vector x* ∧ *one-vector y* ⟶ *one-vector* (*x* + *y*)

by (*smt add-least-upper-bound add-left-upper-bound add-right-upper-bound mult-right-dist-add order-trans one-vector-def*)

lemma *meet-one-vector*: *one-vector x* ∧ *one-vector y* → *one-vector* (*x* ⌢ *y*)
by (*meson dual-order.trans meet.add-least-upper-bound mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right one-vector-def*)

lemma *comp-one-vector*: *one-vector x* ∧ *one-vector y* → *one-vector* (*x* ; *y*)
using *mult-isotone mult-semi-associative order-lesseq-imp one-vector-def* **by** *blast*

end

context *multirelation-algebra-1*

begin

— Theorem 10 / Figure 3: closure properties

— zero-vector

lemma *top-zero-vector*: *zero-vector T*
by (*simp add*: *mult-left-top zero-vector-def*)

end

— M1-algebra

context *multirelation-algebra-2*

begin

— Section 7

lemma *zero-vector-10*: *zero-vector x* ↔ *x* ; *T* = *x* ; *1*
by (*metis mult-one-associative mult-top-associative zero-vector-7*)

lemma *zero-vector-11*: *zero-vector x* ↔ *x* ; *T* ≤ *x* ; *1*
using *antisym-conv mult-right-isotone reflexive-def top-reflexive zero-vector-10* **by** *blast*

— Theorem 6 / Figure 2: relations between properties

lemma *vector-zero-vector*: *vector x* → *zero-vector x*
by (*simp add*: *zero-vector-def vector-left-annihilator*)

lemma *vector-up-closed-zero-vector*: *vector x* ↔ *up-closed x* ∧ *zero-vector x*
using *up-closed-zero-vector-vector vector-up-closed vector-zero-vector* **by** *blast*

lemma *vector-zero-vector-one-vector*: *vector x* ↔ *zero-vector x* ∧ *one-vector x*
using *up-closed-one-vector zero-vector-one-vector-vector vector-up-closed-zero-vector* **by** *blast*

end

— M3-algebra

context *up-closed-multirelation-algebra*

begin

lemma *up-closed*: *up-closed x*
by (*simp add*: *mult-right-one up-closed-def*)

lemma *dedekind-1-left*: $x \ ; \ 1 \frown y \leq (x \frown y \ ; \ 1) \ ; \ 1$
by (*simp add*: *mult-right-one*)

— Theorem 10 / Figure 3: closure properties

— zero-vector

lemma *zero-vector-dual*: *zero-vector x* $\leftrightarrow$ *zero-vector* (x^d)
using *up-closed-zero-vector-vector vector-dual vector-zero-vector up-closed* **by** *blast*

end

— complemented M0-algebra

class *lattice-ordered-pre-left-semiring-b* = *lattice-ordered-pre-left-semiring* + *complemented-distributive-lattice*

begin

definition *down-closed* :: $'a \Rightarrow$ *bool* **where** *down-closed x* $\leftrightarrow -x \ ; \ 1 \leq -x$

— Theorem 10 / Figure 3: closure properties

— down-closed

lemma *zero-down-closed*: *down-closed 0*
by (*simp add*: *down-closed-def*)

lemma *top-down-closed*: *down-closed T*
using *down-closed-def up-closed-def zero-up-closed* **by** *auto*

lemma *complement-down-closed-up-closed*: *down-closed x* $\leftrightarrow$ *up-closed* $(-x)$
by (*simp add*: *down-closed-def meet.eq-iff mult-sub-right-one up-closed-def*)

lemma *add-down-closed*: *down-closed x* ∧ *down-closed y* → *down-closed* (*x* + *y*)
by (*simp add*: *complement-down-closed-up-closed meet-up-closed*)

lemma *meet-down-closed*: *down-closed x* ∧ *down-closed y* → *down-closed* (*x* ⌢ *y*)
by (*simp add*: *complement-down-closed-up-closed add-up-closed*)

end

class *multirelation-algebra-1b* = *multirelation-algebra-1* + *complemented-distributive-lattice*

begin

subclass *lattice-ordered-pre-left-semiring-b* **..**

— Theorem 7.1

lemma *complement-mult-zero-sub*: −(*x* ; *0*) ≤ −*x* ; *0*
proof −
have *T* = −*x* ; *0* + *x* ; *0*
by (*metis compl-sup-top mult-left-top mult-right-dist-add*)
thus *?thesis*
by (*simp add*: *shunting-2 sup.commute*)
qed

— Theorem 7.2

lemma *transitive-zero-vector-complement*: *transitive x* → *zero-vector* (−*x*)
by (*meson complement-mult-zero-sub compl-mono mult-right-isotone order-trans zero-vector-def transitive-def zero-least*)

lemma *transitive-dense-complement*: *transitive x* → *dense* (−*x*)
by (*simp add*: *zero-vector-dense transitive-zero-vector-complement*)

lemma *transitive-add-distributive-complement*: *transitive x* → *add-distributive* (−*x*)
by (*simp add*: *zero-vector-add-distributive transitive-zero-vector-complement*)

lemma *transitive-meet-distributive-complement*: *transitive x* → *meet-distributive* (−*x*)
by (*simp add*: *zero-vector-meet-distributive transitive-zero-vector-complement*)

lemma *up-closed-zero-vector-complement*: *up-closed x* → *zero-vector* (−*x*)
by (*metis complement-mult-zero-sub compl-mono mult-right-isotone order-trans zero-vector-def up-closed-def zero-least*)

lemma *up-closed-dense-complement*: *up-closed x* → *dense* (−*x*)
by (*simp add*: *zero-vector-dense up-closed-zero-vector-complement*)

lemma *up-closed-add-distributive-complement*: *up-closed x* → *add-distributive* (−*x*)
by (*simp add*: *zero-vector-add-distributive up-closed-zero-vector-complement*)

lemma *up-closed-meet-distributive-complement*: *up-closed x* $\longrightarrow$ *meet-distributive* $(-x)$
by (*simp add*: *zero-vector-meet-distributive up-closed-zero-vector-complement*)

— Theorem 10 / Figure 3: closure properties

— closure under complement

lemma *co-total-total*: *co-total x* $\longrightarrow$ *total* $(-x)$
by (*metis complement-mult-zero-sub co-total-def compl-bot-eq mult-left-sub-dist-add-right sup-bot-right top-le total-def*)

lemma *complement-one-vector-zero-vector*: *one-vector x* $\longrightarrow$ *zero-vector* $(-x)$
using *compl-mono complement-mult-zero-sub one-vector-def order-trans zero-vector-def* **by** *blast*

— Theorem 6 / Figure 2: relations between properties

lemma *down-closed-zero-vector*: *down-closed x* $\longrightarrow$ *zero-vector x*
using *complement-down-closed-up-closed up-closed-zero-vector-complement* **by** *force*

lemma *down-closed-one-vector-vector*: *down-closed x* $\wedge$ *one-vector x* $\longrightarrow$ *vector x*
by (*simp add*: *down-closed-zero-vector zero-vector-one-vector-vector*)

end

class *multirelation-algebra-1c* = *multirelation-algebra-1b* +
assumes *dedekind-T-left*: $x \mathbin{;} T \frown y \leq (x \frown y \mathbin{;} T) \mathbin{;} T$
assumes *comp-zero-meet*: $(x \mathbin{;} 0 \frown y) \mathbin{;} 0 \leq (x \frown y) \mathbin{;} 0$

begin

— Theorem 7.3

lemma *schroeder-top-sub*: $-(x \mathbin{;} T) \mathbin{;} T \leq -x$
proof −
have $-(x \mathbin{;} T) \mathbin{;} T \frown x \leq 0$
by (*metis compl-inf-bot dedekind-T-left vector-def zero-vector*)
thus *?thesis*
by (*simp add*: *shunting-1*)
qed

— Theorem 7.4

lemma *schroeder-top*: $x \mathbin{;} T \leq y \leftrightarrow -y \mathbin{;} T \leq -x$
apply *rule*
using *compl-mono meet.order-trans mult-left-isotone schroeder-top-sub* **apply** *blast*
by (*metis compl-mono double-compl mult-left-isotone order-trans schroeder-top-sub*)

— Theorem 7.5

lemma *schroeder-top-eq*: $-(x\ ;\ T)\ ;\ T = -(x\ ;\ T)$
  by (*metis meet.antisym-conv mult-semi-associative top-mult-top top-right-mult-increasing schroeder-top*)

lemma *schroeder-one-eq*: $-(x\ ;\ T)\ ;\ 1 = -(x\ ;\ T)$
  by (*metis top-mult-right-one schroeder-top-eq*)

— Theorem 7.6

lemma *vector-meet-comp*: $x\ ;\ T \sqinter y\ ;\ z = (x\ ;\ T \sqinter y)\ ;\ z$
proof (*rule antisym*)
  have $x\ ;\ T \sqinter y\ ;\ z = x\ ;\ T \sqinter ((x\ ;\ T \sqinter y) + (-(x\ ;\ T) \sqinter y))\ ;\ z$
    using *inf.commute maddux-3-11 meet-left-dist-add* **by** *auto*
  also have $\ldots = x\ ;\ T \sqinter ((x\ ;\ T \sqinter y)\ ;\ z + (-(x\ ;\ T) \sqinter y)\ ;\ z)$
    by (*simp add*: *inf-sup-distrib2 mult-right-dist-add*)
  also have $\ldots = (x\ ;\ T \sqinter (x\ ;\ T \sqinter y)\ ;\ z) + (x\ ;\ T \sqinter (-(x\ ;\ T) \sqinter y)\ ;\ z)$
    by (*simp add*: *meet-left-dist-add*)
  also have $\ldots \leq (x\ ;\ T \sqinter y)\ ;\ z + (x\ ;\ T \sqinter (-(x\ ;\ T) \sqinter y)\ ;\ z)$
    by (*simp add*: *le-infI2*)
  also have $\ldots \leq (x\ ;\ T \sqinter y)\ ;\ z + (x\ ;\ T \sqinter -(x\ ;\ T)\ ;\ z)$
    using *meet.add-right-isotone meet.add-right-upper-bound meet-commutative mult-left-isotone sup-right-isotone* **by** *presburger*
  also have $\ldots \leq (x\ ;\ T \sqinter y)\ ;\ z + (x\ ;\ T \sqinter -(x\ ;\ T)\ ;\ T)$
    using *meet.add-right-isotone mult-right-isotone sup-right-isotone* **by** *auto*
  also have $\ldots = (x\ ;\ T \sqinter y)\ ;\ z$
    by (*simp add*: *meet-complement schroeder-top-eq*)
  finally show $x\ ;\ T \sqinter y\ ;\ z \leq (x\ ;\ T \sqinter y)\ ;\ z$
    .
next
  show $(x\ ;\ T \sqinter y)\ ;\ z \leq x\ ;\ T \sqinter y\ ;\ z$
    by (*metis inf.bounded-iff mult-left-top mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right mult-semi-associative order-lesseq-imp*)
qed

— Theorem 7.7

lemma *vector-zero-meet-comp*: $(x\ ;\ 0 \sqinter y)\ ;\ z = x\ ;\ 0 \sqinter y\ ;\ z$
  by (*metis vector-def comp-vector vector-meet-comp zero-vector*)

lemma *vector-zero-meet-comp-2*: $(x\ ;\ 0 \sqinter y)\ ;\ z = (x\ ;\ 0 \sqinter y\ ;\ 1)\ ;\ z$
  by (*simp add*: *mult-one-associative vector-zero-meet-comp*)

— Theorem 7.8

lemma *comp-zero-meet-2*: $x\ ;\ 0 \sqinter y\ ;\ 0 = (x \sqinter y)\ ;\ 0$

using *meet.antisym mult-right-sub-dist-meet comp-zero-meet vector-zero-meet-comp* **by** *auto*

lemma *comp-zero-meet-3*: $x\ ;\ 0 \frown y\ ;\ 0 = (x\ ;\ 0 \frown y)\ ;\ 0$
by (*simp add*: *vector-zero-meet-comp*)

lemma *comp-zero-meet-4*: $x\ ;\ 0 \frown y\ ;\ 0 = (x\ ;\ 0 \frown y\ ;\ 0)\ ;\ 0$
by (*metis comp-zero-meet-2 inf.commute vector-zero-meet-comp*)

lemma *comp-zero-meet-5*: $x\ ;\ 0 \frown y\ ;\ 0 = (x\ ;\ 1 \frown y\ ;\ 1)\ ;\ 0$
by (*metis comp-zero-meet-2 mult-one-associative*)

lemma *comp-zero-meet-6*: $x\ ;\ 0 \frown y\ ;\ 0 = (x\ ;\ 1 \frown y\ ;\ 0)\ ;\ 0$
using *meet-commutative mult-one-associative vector-zero-meet-comp* **by** *auto*

lemma *comp-zero-meet-7*: $x\ ;\ 0 \frown y\ ;\ 0 = (x\ ;\ 1 \frown y)\ ;\ 0$
by (*metis comp-zero-meet-2 mult-one-associative*)

— Theorem 10 / Figure 3: closure properties

— zero-vector

lemma *meet-zero-vector*: *zero-vector* $x \wedge$ *zero-vector* $y \longrightarrow$ *zero-vector* $(x \frown y)$
by (*metis comp-zero-meet-2 inf.sup-mono zero-vector-def*)

— down-closed

lemma *comp-down-closed*: *down-closed* $x \wedge$ *down-closed* $y \longrightarrow$ *down-closed* $(x\ ;\ y)$
by (*metis complement-down-closed-up-closed down-closed-zero-vector up-closed-def zero-vector-0 schroeder-one-eq*)

— closure under complement

lemma *complement-vector*: *vector* $x \leftrightarrow$ *vector* $(-x)$
by (*metis double-compl vector-def schroeder-top-eq*)

lemma *complement-zero-vector-one-vector*: *zero-vector* $x \longrightarrow$ *one-vector* $(-x)$
by (*smt comp-zero-meet-2 idempotent-def meet.add-relative-same-increasing meet-commutative meet-complement one-vector-def shunting-1 zero-idempotent zero-vector-def*)

lemma *complement-zero-vector-one-vector-iff*: *zero-vector* $x \leftrightarrow$ *one-vector* $(-x)$
using *complement-zero-vector-one-vector complement-one-vector-zero-vector* **by** *force*

lemma *complement-one-vector-zero-vector-iff*: *one-vector* $x \leftrightarrow$ *zero-vector* $(-x)$
using *complement-zero-vector-one-vector complement-one-vector-zero-vector* **by** *force*

— Theorem 6 / Figure 2: relations between properties

lemma *vector-down-closed*: *vector* $x \longrightarrow$ *down-closed* x
using *complement-vector complement-down-closed-up-closed vector-up-closed* **by** *blast*

lemma *co-vector-down-closed*: *co-vector x* $\longrightarrow$ *down-closed x*
by (*metis co-vector-def down-closed-def meet.eq-iff mult-left-zero mult-right-isotone mult-semi-associative zero-least schroeder-one-eq*)

lemma *vector-down-closed-one-vector*: *vector x* $\longleftrightarrow$ *down-closed x* $\wedge$ *one-vector x*
using *down-closed-one-vector-vector up-closed-one-vector vector-up-closed vector-down-closed* **by** *blast*

lemma *vector-up-closed-down-closed*: *vector x* $\longleftrightarrow$ *up-closed x* $\wedge$ *down-closed x*
using *down-closed-zero-vector up-closed-zero-vector-vector vector-up-closed vector-down-closed* **by** *blast*

— Section 7

lemma *vector-b1*: *vector x* $\longleftrightarrow$ $-x$; $T = -x$
using *complement-vector vector-def* **by** *force*

lemma *vector-b2*: *vector x* $\longleftrightarrow$ $-x$; $0 = -x$
by (*metis double-compl up-closed-zero-vector-complement vector-left-mult-closed vector-up-closed zero-vector zero-vector-0 vector-b1*)

lemma *covector-b1*: *co-vector x* $\longleftrightarrow$ $-x$; $T = -x$
using *co-vector-def co-vector-vector vector-b1 vector-b2* **by** *force*

lemma *covector-b2*: *co-vector x* $\longleftrightarrow$ $-x$; $0 = -x$
using *covector-b1 vector-b1 vector-b2* **by** *auto*

lemma *vector-co-vector-iff*: *vector x* $\longleftrightarrow$ *co-vector x*
by (*simp add*: *covector-b1 vector-b1*)

lemma *zero-vector-b*: *zero-vector x* $\longleftrightarrow$ $-x$; $0 \leq -x$
by (*simp add*: *complement-zero-vector-one-vector-iff one-vector-def*)

lemma *one-vector-b1*: *one-vector x* $\longleftrightarrow$ $-x \leq -x$; 0
by (*simp add*: *complement-one-vector-zero-vector-iff zero-vector-def*)

lemma *one-vector-b0*: *one-vector x* $\longleftrightarrow$ $(\forall\, y\ z\ .\ -x\ ;\ y = -x\ ;\ z)$
by (*simp add*: *complement-one-vector-zero-vector-iff zero-vector-0*)

end

class *multirelation-algebra-2b* = *multirelation-algebra-2* + *complemented-distributive-lattice*

begin

subclass *multirelation-algebra-1b* **..**

end

— complemented M1-algebra

class *multirelation-algebra-2c* = *multirelation-algebra-2b* + *multirelation-algebra-1c*

class *multirelation-algebra-3b* = *multirelation-algebra-3* + *complemented-distributive-lattice*

begin

subclass *lattice-ordered-pre-left-semiring-b* **..**

lemma *dual-complement-commute*: $-(x^d) = (-x)^d$
by (*metis compl-unique dual-dist-add dual-dist-meet dual-top dual-zero meet-complement sup-compl-top*)

end

— complemented M2-algebra

class *multirelation-algebra-5b* = *multirelation-algebra-5* + *complemented-distributive-lattice*

begin

subclass *multirelation-algebra-2b* **..**

subclass *multirelation-algebra-3b* **..**

lemma *dual-down-closed*: *down-closed* $x \leftrightarrow$ *down-closed* (x^d)
using *complement-down-closed-up-closed dual-complement-commute dual-up-closed* **by** *auto*

end

class *multirelation-algebra-5c* = *multirelation-algebra-5b* + *multirelation-algebra-1c*

begin

lemma *complement-mult-zero-below*: $-x\ ;\ 0 \leq -(x\ ;\ 0)$
by (*simp add*: *comp-zero-meet-2 mult-left-zero shunting-1*)

end

class *up-closed-multirelation-algebra-b* = *up-closed-multirelation-algebra* + *complemented-distributive-lattice*

begin

subclass *multirelation-algebra-5c*
apply *unfold-locales*
apply (*metis mult-associative mult-right-dist-meet top-mult-top top-right-mult-increasing*)
by (*simp add*: *le-infI2 meet-commutative mult-left-isotone zero-right-mult-decreasing*)

lemma *complement-zero-vector*: *zero-vector* $x \leftrightarrow$ *zero-vector* $(-x)$
by (*metis compl-sup-top double-compl mult-right-dist-add mult-right-one zero-vector-def zero-vector-left-zero shunting-2 top-zero-vector*)

lemma *down-closed*: *down-closed* x
by (*simp add*: *down-closed-def mult-right-one*)

lemma *vector*: *vector* x
by (*simp add*: *down-closed complement-one-vector-zero-vector-iff down-closed-zero-vector vector-down-closed-one-vector*)

end

end