Closure, Properties and Closure Properties of Multirelations

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

2014.11.27

theory *CPCP*

imports *Main*

begin

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

context *ord*

begin

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

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

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 *ascending-chain-k*: $ascending\text{-}chain\ f \rightarrow f\ n \leq f\ (n + k)$
  apply (*induct k*)
  apply *simp*
  apply (*metis add-Suc-right ascending-chain-def order-trans*)
  done

lemma *ascending-chain-isotone*: $ascending\text{-}chain\ f \wedge m \leq n \rightarrow f\ m \leq f\ n$
  by (*metis ascending-chain-k le-iff-add*)

lemma *ascending-chain-comparable*: $ascending\text{-}chain\ f \rightarrow f\ n \leq f\ m \vee f\ m \leq f\ n$
  by (*metis nat-le-linear ascending-chain-isotone*)

lemma *ascending-chain-chain*: $ascending\text{-}chain\ f \rightarrow chain\ (range\ f)$
  by (*smt ascending-chain-comparable chain-def image-iff*)

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

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

lemma *descending-chain-k*: $descending\text{-}chain\ f \rightarrow f\ (n + k) \leq f\ n$
  apply (*induct k*)

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

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

lemma *descending-chain-comparable*: *descending-chain f* $\longrightarrow f\ n \leq f\ m \vee f\ m \leq f\ n$
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*)
by (*metis image-image*)

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*)
by (*metis image-image*)

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*

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*: $id\hat{}n = id$
apply (*induct n*)
apply (*metis one-fun-def power-0*)
apply (*simp add*: *times-fun-def*)
done

lemma *power-zero-id*: $f\hat{}0 = id$
by (*metis one-fun-def power-0*)

lemma *power-succ-unfold*: $f\hat{}Suc\ n = f \circ f\hat{}n$

by (*metis power-Suc times-fun-def*)

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

context *order*

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

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

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

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

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

lemma *pmu-isotone*: *has-least-prefixpoint f* $\wedge$ *has-least-prefixpoint g* $\wedge$ *f* $\leq\leq$ *g* $\rightarrow$ *p*μ *f* $\leq$ *p*μ *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* $\rightarrow$ μ *f* $\leq$ μ *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*ν *f* $\leq$ *p*ν *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* $\rightarrow$ ν *f* $\leq$ ν *g*
by (*metis pnu-isotone pnu-nu*)

lemma *mu-square*: *isotone f* $\wedge$ *has-least-fixpoint f* $\wedge$ *has-least-fixpoint* (*f* $\circ$ *f*) $\rightarrow$ μ *f* = μ (*f* $\circ$ *f*)
by (*metis antisym is-least-fixpoint-def isotone-def least-fixpoint-char least-fixpoint-unique o-apply*)

lemma *nu-square*: *isotone f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *has-greatest-fixpoint* (*f* $\circ$ *f*) $\rightarrow$ ν *f* = ν (*f* $\circ$ *f*)
by (*metis antisym is-greatest-fixpoint-def isotone-def greatest-fixpoint-char greatest-fixpoint-unique o-apply*)

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

lemma *pmu-below-pnu-fix*: *has-fixpoint f* $\wedge$ *has-least-prefixpoint f* $\wedge$ *has-greatest-postfixpoint f* $\rightarrow$ *p*μ *f* $\leq$ *p*ν *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* $\wedge$ *has-least-prefixpoint f* $\wedge$ *has-greatest-postfixpoint f* $\rightarrow$ *p*μ *f* $\leq$ *p*ν *f*
by (*metis has-fixpoint-def is-fixpoint-def is-least-fixpoint-def least-prefixpoint-fixpoint pmu-below-pnu-fix*)

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*))

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 (*smt antisym galois-char isotone-def*)

lemma *mu-fusion-1*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h))) \longrightarrow l(p\mu\ g) \leq \mu\ h$
proof
assume *1*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h)))$
hence $l(g(u(\mu\ h))) \leq \mu\ h$
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* $\wedge$ $l \circ g \leq\leq h \circ l \longrightarrow l(p\mu\ g) \leq \mu\ h$
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* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h))) \wedge l(g(p\mu\ g)) = h(l(p\mu\ g)) \longrightarrow \mu\ h = l(p\mu\ g) \wedge \mu\ h = l(\mu\ g)$
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* $\wedge$ $l(g(u(\mu\ h))) \leq h(l(u(\mu\ h))) \wedge l(g(p\mu\ g)) = h(l(p\mu\ g)) \longrightarrow p\mu\ h = l(p\mu\ g) \wedge \mu\ h = l(p\mu\ g)$
by (*smt antisym galois-char least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint is-least-prefixpoint-def isotone-def mu-fusion-1*)

lemma *mu-fusion-equal-3*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint g* $\wedge$ *has-least-fixpoint h* $\wedge$ $l \circ g = h \circ l \longrightarrow \mu\ h = l(p\mu\ g) \wedge \mu\ h = l(\mu\ g)$
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* $\wedge$ $l \circ g = h \circ l \longrightarrow p\mu\ h = l(p\mu\ g) \wedge \mu\ h = l(p\mu\ g)$
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* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h))) \longrightarrow \nu\ h \leq u(p\nu\ g)$
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* $\wedge$ $h \circ u \leq\leq u \circ g \longrightarrow \nu\ h \leq u(p\nu\ g)$
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* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h))) \wedge h(u(p\nu\ g)) = u(g(p\nu\ g)) \longrightarrow \nu\ h = u(p\nu\ g) \wedge \nu\ h = u(\nu\ g)$
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* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h)))$ $\wedge$ $h(u(p\nu\ g)) = u(g(p\nu\ g)) \rightarrow p\nu\ h = u(p\nu\ g)$ $\wedge$ $\nu\ h = u(p\nu\ g)$
by (*smt antisym galois-char greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint is-greatest-postfixpoint-def isotone-def nu-fusion-1*)

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 $l \circ (h \circ g) \leq\leq (g \circ h) \circ l$
by (*metis o-assoc*)
thus $\mu(l \circ h) \leq \mu(g \circ h)$ **using** *1*
by (*smt galois-char is-least-prefixpoint-def isotone-def least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint mu-fusion-2 mu-roll 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* $(h \circ g)$ $\wedge$ $l \circ h \circ g \leq\leq g \circ h \circ l \rightarrow \mu(h \circ l) \leq \mu(h \circ g)$
by (*smt galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint mu-exchange-1 mu-roll o-apply*)

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* $(h \circ k)$ $\wedge$ $l \circ h \circ k = k \circ h \circ l \rightarrow \mu(l \circ h) = \mu(k \circ h)$ $\wedge$ $\mu(h \circ l) = \mu(h \circ k)$
by (*smt antisym galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint lifted-reflexive mu-exchange-1 mu-exchange-2 o-apply*)

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*)
thus $\nu(g \circ h) \leq \nu(u \circ h)$ **using** *1*
by (*smt galois-char is-greatest-postfixpoint-def isotone-def greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint nu-fusion-2 nu-roll o-apply*)
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* $(h \circ g)$ $\wedge$ $g \circ h \circ u \leq\leq u \circ h \circ g \rightarrow \nu(h \circ g) \leq \nu(h \circ u)$
by (*smt galois-char isotone-def greatest-fixpoint-char greatest-postfixpoint-fixpoint nu-exchange-1 nu-roll o-apply*)

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* $(h \circ t)$ $\wedge$ $u \circ h \circ t = t \circ h \circ u \rightarrow \nu(u \circ h) = \nu(t \circ h)$ $\wedge$ $\nu(h \circ u) = \nu(h \circ t)$
by (*smt antisym galois-char isotone-def greatest-fixpoint-char greatest-postfixpoint-fixpoint lifted-reflexive nu-exchange-1 nu-exchange-2 o-apply*)

lemma *mu-commute-fixpoint-1*: *isotone f* ∧ *has-least-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → *is-fixpoint f* (*μ*(*f* ∘ *g*))
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-fixpoint-2*: *isotone g* ∧ *has-least-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → *is-fixpoint g* (*μ*(*f* ∘ *g*))
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-least-fixpoint*: *isotone f* ∧ *isotone g* ∧ *has-least-fixpoint f* ∧ *has-least-fixpoint g* ∧ *has-least-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → (*μ*(*f* ∘ *g*) = *μ f* → *μ g* ≤ *μ f*)
by (*metis is-least-fixpoint-def least-fixpoint-same least-fixpoint-unique mu-roll*)

lemma *nu-commute-fixpoint-1*: *isotone f* ∧ *has-greatest-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → *is-fixpoint f* (*ν*(*f* ∘ *g*))
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-fixpoint-2*: *isotone g* ∧ *has-greatest-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → *is-fixpoint g* (*ν*(*f* ∘ *g*))
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-greatest-fixpoint*: *isotone f* ∧ *isotone g* ∧ *has-greatest-fixpoint f* ∧ *has-greatest-fixpoint g* ∧ *has-greatest-fixpoint* (*f* ∘ *g*) ∧ *f* ∘ *g* = *g* ∘ *f* → (*ν*(*f* ∘ *g*) = *ν f* → *ν f* ≤ *ν g*)
by (*smt is-greatest-fixpoint-def greatest-fixpoint-same greatest-fixpoint-unique nu-roll*)

lemma *mu-diagonal-1*: *isotone* (*λx* . *f x x*) ∧ (∀ *x* . *isotone* (*λy* . *f x y*)) ∧ *isotone* (*λx* . *μ*(*λy* . *f x y*)) ∧ (∀ *x* . *has-least-fixpoint* (*λy* . *f x y*)) ∧ *has-least-prefixpoint* (*λx* . *μ*(*λy* . *f x y*)) → *μ*(*λx* . *f x x*) = *μ*(*λx* . *μ*(*λy* . *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*: (∀ *x* . *isotone* (*λy* . *f x y*) ∧ *isotone* (*λy* . *f y x*) ∧ *has-least-prefixpoint* (*λy* . *f x y*)) ∧ *has-least-prefixpoint* (*λx* . *μ*(*λy* . *f x y*)) → *μ*(*λx* . *f x x*) = *μ*(*λx* . *μ*(*λy* . *f x y*))
proof
assume *1*: (∀ *x* . *isotone* (*λy* . *f x y*) ∧ *isotone* (*λy* . *f y x*) ∧ *has-least-prefixpoint* (*λy* . *f x y*)) ∧ *has-least-prefixpoint* (*λx* . *μ*(*λy* . *f x y*))
hence *isotone* (*λx* . *μ*(*λy* . *f x y*))
by (*smt isotone-def lifted-less-eq-def mu-isotone*)
thus *μ*(*λx* . *f x x*) = *μ*(*λx* . *μ*(*λy* . *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* (*λx* . *f x x*) ∧ (∀ *x* . *isotone* (*λy* . *f x y*)) ∧ *isotone* (*λx* . *ν*(*λy* . *f x y*)) ∧ (∀ *x* . *has-greatest-fixpoint* (*λy* . *f x y*)) ∧ *has-greatest-postfixpoint* (*λx* . *ν*(*λy* . *f x y*)) → *ν*(*λx* . *f x x*) = *ν*(*λx* . *ν*(*λy* . *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*: (∀ *x* . *isotone* (*λy* . *f x y*) ∧ *isotone* (*λy* . *f y x*) ∧ *has-greatest-postfixpoint* (*λy* . *f x y*)) ∧ *has-greatest-postfixpoint* (*λx* . *ν*(*λy* . *f x y*)) → *ν*(*λx* . *f x x*) = *ν*(*λx* . *ν*(*λy* . *f x y*))
proof
assume *1*: (∀ *x* . *isotone* (*λy* . *f x y*) ∧ *isotone* (*λy* . *f y x*) ∧ *has-greatest-postfixpoint* (*λy* . *f x y*)) ∧ *has-greatest-postfixpoint* (*λx* . *ν*(*λy* . *f x y*))
hence *isotone* (*λx* . *ν*(*λy* . *f x y*))
by (*smt isotone-def lifted-less-eq-def nu-isotone*)
thus *ν*(*λx* . *f x x*) = *ν*(*λx* . *ν*(*λy* . *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*: $x \leq y \rightarrow x + z \leq y + z$
  by (*smt add-associative add-commutative add-idempotent less-eq-def*)

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

lemma *add-isotone*: $w \leq y \wedge x \leq z \rightarrow w + x \leq y + z$
  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* $f \longrightarrow$ *descending-chain* $(\lambda n\ .\ f\ n + x)$
by (*metis descending-chain-def add-left-isotone*)

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*: $0 \leq x$
by (*metis add-left-upper-bound add-left-zero*)

end

class *semiring-0* = *bounded-join-semilattice* + *mult* + *one* +
assumes *mult-left-one* : $1\ ;\ x = x$
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-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*: $w \leq y \wedge x \leq z \rightarrow w \ ; \ x \leq y \ ; \ z$
by (*smt mult-left-isotone mult-right-isotone order-trans*)

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

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

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

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

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

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

abbreviation $L :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $L \ y \equiv (\lambda \ x \ . \ 1 + x \ ; \ y)$
abbreviation $R :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $R \ y \equiv (\lambda \ x \ . \ 1 + y \ ; \ x)$
abbreviation $S :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $S \ y \equiv (\lambda \ x \ . \ 1 + y + x \ ; \ x)$

abbreviation *lstar* $:: 'a \Rightarrow 'a$ **where** *lstar* $y \equiv p\mu \ (L \ y)$
abbreviation *rstar* $:: 'a \Rightarrow 'a$ **where** *rstar* $y \equiv p\mu \ (R \ y)$
abbreviation *sstar* $:: 'a \Rightarrow 'a$ **where** *sstar* $y \equiv p\mu \ (S \ y)$

lemma *lstar-rec-isotone*: *isotone* $(L \ y)$
by (*smt2 add-isotone add-right-divisibility isotone-def mult-right-sub-dist-add-right order.refl*)

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

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

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

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

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

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

lemma *rstar-below-sstar*: *has-least-prefixpoint* $(R\ y) \wedge$ *has-least-prefixpoint* $(S\ y) \longrightarrow rstar\ y \leq sstar\ y$
proof
assume *1*: *has-least-prefixpoint* $(R\ y) \wedge$ *has-least-prefixpoint* $(S\ y)$
hence $R\ y\ (sstar\ y) \leq S\ y\ (sstar\ y)$
by (*smt2 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 *semiring-1* = *semiring-0* +
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* $(R\ y) \longrightarrow y \leq rstar\ y$

proof
  assume *has-least-prefixpoint* $(R\ y)$
  hence $R\ y\ (rstar\ y) \leq rstar\ y$
    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-semiring-1* = *semiring-1* + *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 *lstar-below-rstar*: *has-least-prefixpoint* $(L\ y) \wedge$ *has-least-prefixpoint* $(R\ y) \rightarrow lstar\ y \leq rstar\ y$
proof
  assume *1*: *has-least-prefixpoint* $(L\ y) \wedge$ *has-least-prefixpoint* $(R\ y)$
  have $y\ ;\ (rstar\ y\ /\ y)\ ;\ y \leq y\ ;\ rstar\ y$
    by (*metis mult-right-isotone mult-semi-associative order-trans lres-inverse*)
  also have $... \leq rstar\ y$ **using** *1*
    by (*metis add-least-upper-bound pmu-unfold*)
  finally have $y\ ;\ (rstar\ y\ /\ y) \leq rstar\ y\ /\ y$
    by (*metis lres-galois*)
  hence $R\ y\ (rstar\ y\ /\ y) \leq rstar\ y\ /\ y$ **using** *1*
    by (*metis add-least-upper-bound lres-galois mult-left-one rstar-increasing*)
  hence $rstar\ y \leq rstar\ y\ /\ y$ **using** *1*
    by (*metis is-least-prefixpoint-def least-prefixpoint*)
  hence $L\ y\ (rstar\ y) \leq rstar\ y$ **using** *1*
    by (*metis add-least-upper-bound lres-galois pmu-unfold*)
  thus $lstar\ y \leq rstar\ y$ **using** *1*

by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint*)
qed

lemma *rstar-sstar*: *has-least-prefixpoint* (*R y*) ∧ *has-least-prefixpoint* (*S y*) → *rstar y* = *sstar y*
proof
assume *1*: *has-least-prefixpoint* (*R y*) ∧ *has-least-prefixpoint* (*S y*)
have *R 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 *R 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 *S 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 *semiring-2* = *semiring-1* +
assumes *mult-associative*: (*x* ; *y*) ; *z* = *x* ; (*y* ; *z*)
assumes *mult-right-one* : *x* ; *1* = *x*

begin

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

lemma *test-preserves-equation*: *p* ≤ *p* ; *p* ∧ *p* ≤ *1* → (*p* ; *x* ≤ *x* ; *p* ↔ *p* ; *x* = *p* ; *x* ; *p*)
apply *rule*
apply *rule*
apply (*rule antisym*)
apply (*smt 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 *meet* =
fixes *meet* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\frown$ 65)

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

sublocale *meet-semilattice* < *meet*!: *join-semilattice* **where** *plus* = *meet* **and** *less-eq* = $(\lambda x\ y\ .\ y \leq x)$ **and** *less* = $(\lambda x\ y\ .\ y < x)$
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*: $T + x = T$
by (*metis add-absorb meet-left-top*)

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

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 *los-1* = *semiring-1* + *bounded-distributive-lattice*

begin

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

lemma *top-right-mult-increasing*: $x \leq x \ ; \ T$
by (*metis dual-order.trans meet.zero-least mult-right-isotone mult-sub-right-one*)

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

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*)

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 *vector* :: $'a \Rightarrow$ *bool* **where** *vector* $x \leftrightarrow x\ ;\ T = x$
definition *co-vector* :: $'a \Rightarrow$ *bool* **where** *co-vector* $x \leftrightarrow x\ ;\ 0 = x$

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* ⟶ *transitive x*
by (*metis contact-def add-left-upper-bound transitive-def*)

lemma *contact-dense*: *contact x* ⟶ *dense x*
by (*metis contact-reflexive reflexive-dense*)

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

lemma *contact-up-closed*: *contact x* ⟶ *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* ⟷ *reflexive x* ∧ *idempotent x* ∧ *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* ⟶ *co-reflexive x*
by (*metis co-reflexive-def kernel-def meet.add-least-upper-bound mult-sub-right-one*)

lemma *kernel-transitive*: *kernel x* ⟶ *transitive x*
by (*metis co-reflexive-transitive kernel-co-reflexive*)

lemma *kernel-dense*: *kernel x* ⟶ *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 (*smt2 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*)

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*)

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*)

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 (*smt2 meet.less-eq-def meet-associative meet-commutative mult-left-sub-dist-meet-left mult-right-sub-dist-meet-left transitive-def*)

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* ∧ *dense y*
hence *x* ≤ *x* ; *x* ∧ *y* ≤ *y* ; *y*
by (*metis dense-def*)
hence *x* ≤ (*x* + *y*) ; (*x* + *y*) ∧ *y* ≤ (*x* + *y*) ; (*x* + *y*)
by (*metis add-left-upper-bound dual-order.trans mult-isotone add-right-upper-bound*)
hence *x* + *y* ≤ (*x* + *y*) ; (*x* + *y*)
by (*metis add-least-upper-bound*)
thus *dense* (*x* + *y*)
by (*metis dense-def*)
qed

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* ∧ *reflexive y* → *reflexive* (*x* + *y*)
by (*metis add-associative less-eq-def reflexive-def*)

lemma *meet-reflexive*: *reflexive x* ∧ *reflexive y* → *reflexive* (*x* ⌢ *y*)

by (*metis meet.add-least-upper-bound reflexive-def*)

lemma *comp-reflexive*: *reflexive x* ∧ *reflexive y* → *reflexive* (*x* ; *y*)
by (*metis mult-left-isotone mult-left-one order-trans reflexive-def*)

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* ∧ *co-reflexive y* → *co-reflexive* (*x* + *y*)
by (*metis co-reflexive-def add-least-upper-bound*)

lemma *meet-co-reflexive*: *co-reflexive x* ∧ *co-reflexive y* → *co-reflexive* (*x* ⌢ *y*)
by (*metis co-reflexive-def meet.less-eq-def meet-associative*)

lemma *comp-co-reflexive*: *co-reflexive x* ∧ *co-reflexive y* → *co-reflexive* (*x* ; *y*)
by (*metis co-reflexive-def mult-isotone mult-left-one*)

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*)

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* ∧ *up-closed y* → *up-closed* (*x* + *y*)
by (*metis mult-right-dist-add up-closed-def*)

lemma *meet-up-closed*: *up-closed x* ∧ *up-closed y* → *up-closed* (*x* ⌢ *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* ∧ *up-closed y* → *up-closed* (*x* ; *y*)
by (*metis dual-order.antisym mult-semi-associative mult-sub-right-one up-closed-def*)

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* ∧ *add-distributive y* → *add-distributive* (*x* + *y*)
by (*smt2 add-distributive-def add-associative add-commutative mult-right-dist-add*)

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*)

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 (*smt2 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*)

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*)

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

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*)

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* $\wedge$ *test y* $\rightarrow$ *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* $\wedge$ *test y* $\rightarrow$ *test* (*x* $\frown$ *y*)
by (*smt2 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*)

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* $\wedge$ *co-test y* $\rightarrow$ *co-test* (*x* + *y*)
by (*smt2 co-test-contact co-test-def contact-def add-associative add-commutative add-left-zero mult-left-one mult-right-dist-add*)

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* $\wedge$ *vector y* $\rightarrow$ *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 *los-2* = *semiring-0* + *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 *semiring-1*
apply *unfold-locales*
apply (*metis mult-associative-one mult-left-isotone mult-right-isotone mult-sub-right-one*)
done

subclass *los-1*
..

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 ⌢ 1) ; *y* = *x* ; 0 ⌢ *y*
by (*metis mult-left-one mult-right-dist-meet-one mult-zero-associative*)

lemma *co-test-meet-distributive*: *co-test x* → *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* → *add-distributive x*
by (*smt2 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* → *add-dist-contact x*
by (*metis co-test-add-distributive add-dist-contact-def co-test-contact*)

lemma *meet-co-test*: *co-test x* ∧ *co-test y* → *co-test* (*x* ⌢ *y*)
by (*smt2 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* ∧ *co-test y* → *co-test* (*x* ; *y*)
by (*metis add-associative co-test-def mult-zero-add-dist mult-zero-add-one-dist*)

end

class *los-3* = *los-1* +
assumes *mult-sub-associative-one*: *x* ; (*y* ; *z*) ≤ (*x* ; (*y* ; *1*)) ; *z*
assumes *mult-right-dist-meet-one-sub*: *x* ; *z* ⌢ *y* ; *z* ≤ (*x* ; *1* ⌢ *y* ; *1*) ; *z*

begin

subclass *los-2*
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 *mra-0* = *los-1* +
assumes *mult-left-top*: *T* ; *x* = *T*

begin

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

class *mra-1* = *mra-0* + *los-3*

begin

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

lemma *vector-meet-one-comp*: $(x\ ;\ T \frown 1)\ ;\ y = x\ ;\ T \frown y$
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*)

lemma *test-comp-meet*: *test* $x \wedge$ *test* $y \rightarrow x\ ;\ y = x \frown y$
by (*smt2 meet-associative meet-commutative meet-idempotent test-def vector-meet-one-comp*)

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 (*smt2 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*)

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 *mra-2* = *los-1* + *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*)

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 *mra-3* = *mra-2* +
assumes *dual-sub-dist-comp*: $(x \ ; \ y)^d \leq x^d \ ; \ y^d$

begin

subclass *mra-0*
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*)

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

end

class *mra-4* = *mra-2* +
assumes *dual-dist-comp-one*: $(x\ ;\ y)^d = (x\ ;\ 1)^d\ ;\ y^d$

begin

subclass *mra-3*
apply *unfold-locales*
apply (*metis dual-antitone mult-sub-right-one mult-left-isotone dual-dist-comp-one*)
done

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

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

subclass *los-3*
apply *unfold-locales*
apply (*smt2 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 (*smt2 dual-dist-comp-one dual-dist-meet dual-involutive eq-refl mult-one-associative mult-right-dist-add*)
done

subclass *mra-1*
..

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

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*)

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 (*smt2 meet-distributive-def add-distributive-def dual-dist-add dual-involutive strong-up-closed-2*)
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 (*smt2 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

lemma *test-co-test-dual*: *test* $x \leftrightarrow$ *co-test* (x^d)
by (*smt2 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*)

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

end

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

begin

subclass *mra-4*
apply *unfold-locales*

apply (*metis dual-sub-dist-comp dual-sub-dist-comp-one meet.eq-iff mult-one-associative*)
done

end

class *mra-up-closed* = *mra-2* +
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*)

subclass *semiring-2*
apply *unfold-locales*
apply (*metis antisym dual-antitone dual-dist-comp dual-involutive mult-semi-associative*)
apply (*metis dual-dist-add dual-dist-comp dual-involutive dual-one less-eq-def meet-absorb mult-sub-right-one*)
done

subclass *mra-5*
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*)

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

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

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

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

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

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

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

lemma *meet-meet-distributive*: *meet-distributive* $x \wedge$ *meet-distributive* $y \longrightarrow$ *meet-distributive* $(x \frown y)$
by (*smt2 meet-distributive-def meet-associative meet-commutative mult-right-dist-meet*)

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

end

class *mra-6* = *mra-3* +
assumes *vector-meet-comp*: $(x \ ; \ T \frown y) \ ; \ z = x \ ; \ T \frown y \ ; \ z$

begin

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

lemma *test-add-distributive*: *test* $x \longrightarrow$ *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* $x \longrightarrow$ *meet-distributive* x
by (*smt2 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* $\longrightarrow$ *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* $\longrightarrow$ *meet-distributive x*
proof
assume *co-test x*
hence $x = x \mathbin{;} 0 + 1$
by (*metis co-test-def*)
hence $\forall\, y\ z\ .\ x \mathbin{;} y \frown x \mathbin{;} z = x \mathbin{;} (y \frown z)$
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* $\longrightarrow$ *add-distributive x*
proof
assume *co-test x*
hence *1*: $x = x \mathbin{;} 0 + 1$
by (*metis co-test-def*)
hence $\forall\, y\ z\ .\ x \mathbin{;} (y + z) = x \mathbin{;} y + x \mathbin{;} z$
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

end