Quantales

Georg Struth

September 18, 2020

Abstract

These mathematical components formalise basic properties of quantales, together with some important models, constructions, and concepts, including quantic nuclei and conuclei.

Contents

1 Introductory Remarks

Quantales are complete lattices equipped with an associative composition that preserves suprema in both arguments. They have been used—under various names and in various guises—in mathematics for almost a century. One important context is the structure of ideals in rings and C^*-algebras, another one the foundations of quantum mechanics, a third one lies in approaches to generalised metric spaces. In computing, quantales occur naturally in program semantics—algebras of predicate transformers, for instance, form quantales, the semantics of linear logic, the foundations of fuzzy systems and program construction; but also languages or binary relations form quantales.

These components formalise the basic concepts and properties of quantales, following by and large Rosenthal's monograph [6]. Because of applications to predicate transformer semantics, families of quantales are considered in which certain Sup-preservation laws are absent (nomenclature diverges from Rosenthal, but is consistent with AFP entries for dioids and Kleene algebras [2]). Beyond basic equational reasoning, some models of quantales are presented, though those that arise from ring theory or C^*-algebras are currently not supported.

Nuclei and conuclei of quantales are also investigated, and some important relationships with quotients and subalgebras of quantales are formalised, following Rosenthal. In particular, I (re)prove his representation theorem that every quantale is isomorphic to a nucleus of a powerset quantale over some semigroup. Beyond that it is shown how left-sided elements give rise to nuclei and conuclei.

Another subject of study are quantale-modules, which have been introduced by Abramsky and Vickers [1] and widely used since, with some original results on semidirect products over these [4] and some new results on the Kleene star in this setting.

Quantales draw heavily on lattice and order theory, Galois connections and the associated monads and comonads. They are also strongly related to complete Heyting algebras, frames and locales [5], for which future AFP entries might be worth creating. Further variants, such as Girard quantales, might also be worth exploring.

2 Quantales

theory *Quantales*
imports
Order-Lattice-Props.Closure-Operators
Kleene-Algebra.Dioid
begin

2.1 Families of Proto-Quantales

Proto-Quanales are complete lattices equipped with an operation of composition or multiplication that need not be associative. The notation in this component differs from Rosenthal's [6], but is consistent with the one we use for semirings and Kleene algebras.

class *proto-near-quantale* = *complete-lattice* + *times* +
assumes *Sup-distr*: $\bigsqcup X \cdot y = (\bigsqcup x \in X.\ x \cdot y)$

lemma *Sup-pres-multr*: *Sup-pres* $(\lambda(z::'a::proto\text{-}near\text{-}quantale).\ z \cdot y)$
unfolding *fun-eq-iff comp-def Sup-distr* **by** *simp*

lemma *sup-pres-multr*: *sup-pres* $(\lambda(z::'a::proto\text{-}near\text{-}quantale).\ z \cdot y)$
using *Sup-pres-multr Sup-sup-pres* **by** *fastforce*

lemma *bot-pres-multr*: *bot-pres* $(\lambda(z::'a::proto\text{-}near\text{-}quantale).\ z \cdot y)$
by (*metis SUP-empty Sup-distr Sup-empty*)

context *proto-near-quantale*
begin

lemma *mult-botl* [*simp*]: $\bot \cdot x = \bot$
proof −
have $\bot \cdot x = (\bigsqcup a \in \{\}.\ a \cdot x)$
using *Sup-distr Sup-empty* **by** *blast*
thus *?thesis*
by *simp*
qed

lemma *sup-distr*: $(x \sqcup y) \cdot z = (x \cdot z) \sqcup (y \cdot z)$
by (*smt SUP-empty SUP-insert Sup-distr sup-Sup sup-bot.right-neutral*)

lemma *mult-isor*: $x \leq y \Longrightarrow x \cdot z \leq y \cdot z$
by (*metis sup.absorb-iff1 sup-distr*)

Left and right residuals can be defined in every proto-nearquantale.

definition *bres* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\rightarrow$ *60*) **where**
$x \rightarrow z = \bigsqcup \{y.\ x \cdot y \leq z\}$

definition *fres* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\leftarrow$ *60*) **where**

$z \leftarrow y = \bigsqcup \{x.\ x \cdot y \leq z\}$

The left one is a right adjoint to composition. For the right one, additional assumptions are needed

lemma *bres-galois-imp*: $x \cdot y \leq z \Longrightarrow y \leq x \rightarrow z$
by (*simp add*: *Sup-upper bres-def*)

lemma *fres-galois*: $(x \cdot y \leq z) = (x \leq z \leftarrow y)$
proof
show $x \cdot y \leq z \Longrightarrow x \leq z \leftarrow y$
by (*simp add*: *Sup-upper fres-def*)
next
assume $x \leq z \leftarrow y$
hence $x \cdot y \leq \bigsqcup \{x.\ x \cdot y \leq z\} \cdot y$
by (*simp add*: *fres-def mult-isor*)
also have $... = \bigsqcup \{x \cdot y \mid x.\ x \cdot y \leq z\}$
by (*simp add*: *Sup-distr setcompr-eq-image*)
also have $... \leq z$
by (*rule Sup-least*, *auto*)
finally show $x \cdot y \leq z$ **.**
qed

end

lemma *fres-adj*: $(\lambda(x::'a::proto\text{-}near\text{-}quantale).\ x \cdot y) \dashv (\lambda x.\ x \leftarrow y)$
by (*simp add*: *adj-def fres-galois*)

context *proto-near-quantale*
begin

lemma *fres-canc1*: $(y \leftarrow x) \cdot x \leq y$
by (*simp add*: *fres-galois*)

lemma *fres-canc2*: $y \leq (y \cdot x) \leftarrow x$
using *fres-galois* **by** *force*

lemma *inf-fres*: $y \cdot x = \bigsqcap \{z.\ y \leq z \leftarrow x\}$
by (*metis* (*mono-tags*, *lifting*) *fres-canc2 Inf-eqI fres-galois mem-Collect-eq*)

lemma *bres-iso*: $x \leq y \Longrightarrow z \rightarrow x \leq z \rightarrow y$
using *Sup-le-iff bres-def bres-galois-imp* **by** *force*

lemma *bres-anti*: $x \leq y \Longrightarrow y \rightarrow z \leq x \rightarrow z$
by (*smt Sup-le-iff bres-def bres-galois-imp fres-galois order-trans mem-Collect-eq*)

lemma *fres-iso*: $x \leq y \Longrightarrow x \leftarrow z \leq y \leftarrow z$
using *fres-galois dual-order.trans* **by** *blast*

lemma *bres-top-top* [*simp*]: $\top \rightarrow \top = \top$

by (*simp add*: *bres-galois-imp dual-order.antisym*)

lemma *fres-top-top* [*simp*]: $\top \leftarrow \top = \top$
using *fres-galois top-greatest top-le* **by** *blast*

lemma *bres-bot-bot* [*simp*]: $\bot \rightarrow \bot = \top$
by (*simp add*: *bres-galois-imp top-le*)

lemma *left-sided-localp*: $\top \cdot y = y \Longrightarrow x \cdot y \leq y$
by (*metis mult-isor top-greatest*)

lemma *fres-sol*: $((y \leftarrow x) \cdot x = y) = (\exists z.\ z \cdot x = y)$
using *dual-order.antisym fres-canc1 fres-canc2 mult-isor* **by** *fastforce*

lemma *sol-fres*: $((y \cdot x) \leftarrow x = y) = (\exists z.\ y = z \leftarrow x)$
by (*metis fres-canc1 fres-canc2 fres-sol eq-iff fres-galois*)

end

class *proto-pre-quantale* = *proto-near-quantale* +
assumes *Sup-subdistl*: $(\bigsqcup y \in Y.\ x \cdot y) \leq x \cdot \bigsqcup Y$

begin

lemma *sup-subdistl*: $(x \cdot y) \sqcup (x \cdot z) \leq x \cdot (y \sqcup z)$
by (*smt SUP-empty SUP-insert Sup-subdistl sup-Sup sup-bot-right*)

lemma *mult-isol*: $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$
by (*metis le-iff-sup le-sup-iff sup-subdistl*)

lemma *fres-anti*: $x \leq y \Longrightarrow z \leftarrow y \leq z \leftarrow x$
using *dual-order.trans fres-galois mult-isol* **by** *blast*

end

class *weak-proto-quantale* = *proto-near-quantale* +
assumes *weak-Sup-distl*: $Y \neq \{\} \Longrightarrow x \cdot \bigsqcup Y = (\bigsqcup y \in Y.\ x \cdot y)$

begin

subclass *proto-pre-quantale*
proof *unfold-locales*
have *a*: $\bigwedge x\ Y.\ Y = \{\} \Longrightarrow (\bigsqcup y \in Y.\ x \cdot y) \leq x \cdot \bigsqcup Y$
by *simp*
have *b*: $\bigwedge x\ Y.\ Y \neq \{\} \Longrightarrow (\bigsqcup y \in Y.\ x \cdot y) \leq x \cdot \bigsqcup Y$
by (*simp add*: *weak-Sup-distl*)
show $\bigwedge x\ Y.\ (\bigsqcup y \in Y.\ x \cdot y) \leq x \cdot \bigsqcup Y$
using *a b* **by** *blast*
qed

lemma *sup-distl*: $x \cdot (y \sqcup z) = (x \cdot y) \sqcup (x \cdot z)$
using *weak-Sup-distl*[**where** $Y=\{y, z\}$] **by** (*fastforce intro*!: *Sup-eqI*)

lemma $y \leq x \rightarrow z \longrightarrow x \cdot y \leq z$
oops

end

class *proto-quantale* = *proto-near-quantale* +
assumes *Sup-distl*: $x \cdot \bigsqcup Y = (\bigsqcup y \in Y.\ x \cdot y)$

lemma *Sup-pres-multl*: *Sup-pres* (λ(*z*::'*a*::*proto-quantale*). $x \cdot z$)
unfolding *fun-eq-iff comp-def Sup-distl* **by** *simp*

lemma *sup-pres-multl*: *sup-pres* (λ(*z*::'*a*::*proto-quantale*). $x \cdot z$)
by (*metis* (*no-types*, *lifting*) *SUP-insert Sup-distl Sup-empty Sup-insert sup-bot-right*)

lemma *bot-pres-multl*: *bot-pres* (λ(*z*::'*a*::*proto-quantale*). $x \cdot z$)
by (*metis SUP-empty Sup-distl Sup-empty*)

context *proto-quantale*
begin

subclass *weak-proto-quantale*
by *standard* (*simp add*: *Sup-distl*)

lemma *mult-botr* [*simp*]: $x \cdot \bot = \bot$
by (*smt image-empty Sup-distl Sup-empty*)

Now there is also an adjunction for the other residual.

lemma *bres-galois*: $x \cdot y \leq z \longleftrightarrow y \leq x \rightarrow z$
proof
show $x \cdot y \leq z \Longrightarrow y \leq x \rightarrow z$
by (*simp add*: *Sup-upper bres-def*)
next
assume $y \leq x \rightarrow z$
hence $x \cdot y \leq x \cdot \bigsqcup \{y.\ x \cdot y \leq z\}$
by (*simp add*: *bres-def mult-isol*)
also have ... $= \bigsqcup \{x \cdot y \mid y.\ x \cdot y \leq z\}$
by (*simp add*: *Sup-distl setcompr-eq-image*)
also have ... $\leq z$
by (*rule Sup-least*, *safe*)
finally show $x \cdot y \leq z$.
qed

end

lemma *bres-adj*: (λ(*y*::'*a*::*proto-quantale*). $x \cdot y$) $\dashv$ ($\lambda y.\ x \rightarrow y$)

by (*simp add*: *adj-def bres-galois*)

context *proto-quantale*
begin

lemma *bres-canc1*: $x \cdot (x \rightarrow y) \leq y$
by (*simp add*: *bres-galois*)

lemma *bres-canc2*: $y \leq x \rightarrow (x \cdot y)$
by (*simp add*: *bres-galois-imp*)

lemma *inf-bres*: $x \cdot y = \bigsqcap \{z.\ y \leq x \rightarrow z\}$
using *bres-galois fres-galois inf-fres* **by** *force*

lemma *bres-sol*: $(x \cdot (x \rightarrow y) = y) = (\exists\, z.\ x \cdot z = y)$
using *bres-galois antisym mult-isol* **by** *force*

lemma *sol-bres*: $(x \rightarrow (x \cdot y) = y) = (\exists\, z.\ y = x \rightarrow z)$
by (*metis bres-canc1 bres-canc2 bres-iso eq-iff*)

end

lemma *bres-fres-clop*: *clop* $(\lambda x{::}'a{::}\textit{proto-quantale}.\ y \leftarrow (x \rightarrow y))$
unfolding *clop-def comp-def mono-def le-fun-def*
by (*metis bres-anti bres-canc1 bres-galois-imp fres-anti fres-galois id-apply*)

lemma *fres-bres-clop*: *clop* $(\lambda x{::}'a{::}\textit{proto-quantale}.\ (y \leftarrow x) \rightarrow y)$
unfolding *clop-def comp-def mono-def le-fun-def*
by (*metis bres-anti bres-canc1 bres-galois-imp fres-anti fres-canc1 fres-galois id-apply*)

2.2 Families of Quantales

class *near-quantale* = *proto-near-quantale* + *semigroup-mult*

sublocale *near-quantale* $\subseteq$ *nsrnq*: *near-dioid* $(\sqcup)$ $(\cdot)$ $(\leq)$ $(<)$
apply *unfold-locales*
apply (*simp add*: *sup-assoc*)
apply (*simp add*: *sup-commute*)
apply (*simp-all add*: *sup-distr*)
apply (*simp add*: *le-iff-sup*)
by *auto*

context *near-quantale*
begin

lemma *fres-curry*: $(z \leftarrow y) \leftarrow x = z \leftarrow (x \cdot y)$
by (*metis eq-iff fres-canc1 fres-galois mult-assoc*)

end

class *unital-near-quantale* = *near-quantale* + *monoid-mult*

sublocale *unital-near-quantale* ⊆ *nsrnqo*: *near-dioid-one* (⊔) (·) *1*(≤) (<)
by (*unfold-locales*, *simp-all*)

context *unital-near-quantale*
begin

definition *iter* :: *'a* ⇒ *'a* **where**
iter x ≡ ⨆ *i*. *x* ^ *i*

lemma *iter-ref* [*simp*]: *iter x* ≤ *1*
by (*metis iter-def Inf-lower power.power-0 rangeI*)

lemma *le-top*: *x* ≤ ⊤ · *x*
by (*metis mult.left-neutral mult-isor top-greatest*)

lemma *top-times-top* [*simp*]: ⊤ · ⊤ = ⊤
by (*simp add*: *le-top top-le*)

lemma *bres-one*: *1* ≤ *x* → *x*
by (*simp add*: *bres-galois-imp*)

lemma *fres-one*: *1* ≤ *x* ← *x*
using *fres-galois* **by** *fastforce*

end

class *pre-quantale* = *proto-pre-quantale* + *semigroup-mult*

begin

subclass *near-quantale* **..**

lemma *fres-interchange*: *z* · (*x* ← *y*) ≤ (*z* · *x*) ← *y*
using *Sup-upper fres-canc1 fres-def mult-isol mult-assoc* **by** *fastforce*

end

sublocale *pre-quantale* ⊆ *psrpq*: *pre-dioid* (⊔) (·) (≤) (<)
by (*unfold-locales*, *simp add*: *mult-isol*)

class *unital-pre-quantale* = *pre-quantale* + *monoid-mult*

begin

subclass *unital-near-quantale* **..**

Abstract rules of Hoare logic without the star can be derived.

lemma *h-w1*: $x \leq x' \Longrightarrow\ x' \cdot y \leq z \Longrightarrow x \cdot y \leq z$
by (*simp add*: *fres-galois*)

lemma *h-w2*: $x \cdot y \leq z' \Longrightarrow z' \leq z \Longrightarrow x \cdot y \leq z$
using *order-trans* **by** *blast*

lemma *h-seq*: $x \cdot v \leq z \Longrightarrow y \cdot w \leq v \Longrightarrow x \cdot y \cdot w \leq z$
using *dual-order.trans mult-isol mult-assoc* **by** *presburger*

lemma *h-sup*: $x \cdot w \leq z \Longrightarrow y \cdot w \leq z \Longrightarrow (x \sqcup y) \cdot w \leq z$
by (*simp add*: *fres-galois*)

lemma *h-Sup*: $\forall\, x \in X.\ x \cdot w \leq z \Longrightarrow \bigsqcup X \cdot w \leq z$
by (*simp add*: *Sup-least fres-galois*)

end

sublocale *unital-pre-quantale* $\subseteq$ *psrpqo*: *pre-dioid-one* $(\sqcup)$ $(\cdot)$ *1* $(\leq)$ $(<)$**..**

class *weak-quantale* = *weak-proto-quantale* + *semigroup-mult*

begin

subclass *pre-quantale* **..**

The following counterexample shows an important consequence of weakness: the absence of right annihilation.

lemma $x \cdot \bot = \bot$
oops

end

class *unital-weak-quantale* = *weak-quantale* + *monoid-mult*

lemma (**in** *unital-weak-quantale*) $x \cdot \bot = \bot$
oops

subclass (**in** *unital-weak-quantale*) *unital-pre-quantale* **..**

sublocale *unital-weak-quantale* $\subseteq$ *wswq*: *dioid-one-zerol* $(\sqcup)$ $(\cdot)$ *1* $\bot$ $(\leq)$ $(<)$
by (*unfold-locales*, *simp-all add*: *sup-distl*)

class *quantale* = *proto-quantale* + *semigroup-mult*

begin

subclass *weak-quantale* **..**

lemma *Inf-subdistl*: $x \cdot \bigsqcap Y \leq (\bigsqcap y \in Y.\ x \cdot y)$
by (*auto intro*!: *Inf-greatest Inf-lower mult-isol*)

lemma *Inf-subdistr*: $\bigsqcap X \cdot y \leq (\bigsqcap x \in X.\ x \cdot y)$
by (*auto intro*!: *Inf-greatest Inf-lower mult-isor*)

lemma *fres-bot-bot* [*simp*]: $\bot \leftarrow \bot = \top$
by (*simp add*: *fres-def*)

lemma *bres-interchange*: $(x \rightarrow y) \cdot z \leq x \rightarrow (y \cdot z)$
by (*metis bres-canc1 bres-galois mult-isor mult-assoc*)

lemma *bres-curry*: $x \rightarrow (y \rightarrow z) = (y \cdot x) \rightarrow z$
by (*metis bres-canc1 bres-galois dual-order.antisym mult-assoc*)

lemma *fres-bres*: $x \rightarrow (y \leftarrow z) = (x \rightarrow y) \leftarrow z$
proof−
{fix *w*
have $(w \leq x \rightarrow (y \leftarrow z)) = (x \cdot w \leq y \leftarrow z)$
by (*simp add*: *bres-galois*)
also have ... $= (x \cdot w \cdot z \leq y)$
by (*simp add*: *fres-galois*)
also have ... $= (w \cdot z \leq x \rightarrow y)$
by (*simp add*: *bres-galois mult-assoc*)
also have ... $= (w \leq (x \rightarrow y) \leftarrow z)$
by (*simp add*: *fres-galois*)
finally have $(w \leq x \rightarrow (y \leftarrow z)) = (w \leq (x \rightarrow y) \leftarrow z)$**.}**
thus *?thesis*
using *eq-iff* **by** *blast*
qed

end

class *quantale-with-dual* = *quantale* + *complete-lattice-with-dual*

class *unital-quantale* = *quantale* + *monoid-mult*

class *unital-quantale-with-dual* = *unital-quantale* + *quantale-with-dual*

subclass (**in** *unital-quantale*) *unital-weak-quantale* **..**

sublocale *unital-quantale* ⊆ *wswq*: *dioid-one-zero* (⊔) (·) *1* ⊥ (≤) (<)
by (*unfold-locales*, *simp*)

class *ab-quantale* = *quantale* + *ab-semigroup-mult*

begin

lemma *bres-fres-eq*: $x \rightarrow y = y \leftarrow x$
by (*simp add*: *fres-def bres-def mult-commute*)

end

class *ab-unital-quantale* = *ab-quantale* + *unital-quantale*

sublocale *complete-heyting-algebra* $\subseteq$ *chaq*: *ab-unital-quantale* ($\sqcap$) - - - - - - - - $\top$
by (*unfold-locales*, *simp add*: *inf.assoc*, *simp-all add*: *inf.assoc ch-dist inf.commute*)

class *distrib-quantale* = *quantale* + *distrib-lattice*

class *bool-quantale* = *quantale* + *complete-boolean-algebra-alt*

class *distrib-unital-quantale* = *unital-quantale* + *distrib-lattice*

class *bool-unital-quantale* = *unital-quantale* + *complete-boolean-algebra-alt*

class *distrib-ab-quantale* = *distrib-quantale* + *ab-quantale*

class *bool-ab-quantale* = *bool-quantale* + *ab-quantale*

class *distrib-ab-unital-quantale* = *distrib-quantale* + *unital-quantale*

class *bool-ab-unital-quantale* = *bool-ab-quantale* + *unital-quantale*

sublocale *complete-boolean-algebra* $\subseteq$ *cba-quantale*: *bool-ab-unital-quantale inf* - -
- - - - - - - - $\top$
by (*unfold-locales*, *simp add*: *inf.assoc*, *simp-all add*: *inf.commute Setcompr-eq-image inf-Sup Sup-inf*)

context *complete-boolean-algebra*
begin

In this setting, residuation is classical implication.

lemma *cba-bres1*: $x \sqcap y \leq z \longleftrightarrow x \leq$ *cba-quantale.bres y z*
using *cba-quantale.bres-galois inf.commute* **by** *fastforce*

lemma *cba-bres2*: $x \leq -y \sqcup z \longleftrightarrow x \leq$ *cba-quantale.bres y z*
using *cba-bres1 shunt1* **by** *auto*

lemma *cba-bres-prop*: *cba-quantale.bres x y* $= -x \sqcup y$
using *cba-bres2 eq-iff* **by** *blast*

end

2.3 Quantales Based on Sup-Lattices and Inf-Lattices

These classes are defined for convenience in instantiation and interpretation proofs, or likewise. They are useful, e.g., in the context of predicate transformers, where only one of Sup or Inf may be well behaved.

class *Sup-quantale* = *Sup-lattice* + *semigroup-mult* +
  assumes *Supq-distr*: $\bigsqcup X \cdot y = (\bigsqcup x \in X.\ x \cdot y)$
  and *Supq-distl*: $x \cdot \bigsqcup Y = (\bigsqcup y \in Y.\ x \cdot y)$

class *unital-Sup-quantale* = *Sup-quantale* + *monoid-mult*

class *Inf-quantale* = *Inf-lattice* + *monoid-mult* +
  assumes *Supq-distr*: $\bigsqcap X \cdot y = (\bigsqcap x \in X.\ x \cdot y)$
  and *Supq-distl*: $x \cdot \bigsqcap Y = (\bigsqcap y \in Y.\ x \cdot y)$

class *unital-Inf-quantale* = *Inf-quantale* + *monoid-mult*

sublocale *Inf-quantale* $\subseteq$ *qdual*: *Sup-quantale - Inf* $(\geq)$
  by (*unfold-locales*, *simp-all add*: *Supq-distr Supq-distl*)

sublocale *unital-Inf-quantale* $\subseteq$ *uqdual*: *unital-Sup-quantale - - Inf* $(\geq)$**..**

sublocale *Sup-quantale* $\subseteq$ *supq*: *quantale - Infs Sup-class.Sup infs* $(\leq)$ *le sups bots tops*
  by (*unfold-locales*, *simp-all add*: *Supq-distr Supq-distl*)

sublocale *unital-Sup-quantale* $\subseteq$ *usupq*: *unital-quantale - - Infs Sup-class.Sup infs* $(\leq)$ *le sups bots tops***..**

2.4 Products of Quantales

definition *Inf-prod* $X = ((\bigsqcap x \in X.\ fst\ x), (\bigsqcap x \in X.\ snd\ x))$

definition *inf-prod* $x\ y = (fst\ x \sqcap fst\ y,\ snd\ x \sqcap snd\ y)$

definition *bot-prod* $= (bot, bot)$

definition *Sup-prod* $X = ((\bigsqcup x \in X.\ fst\ x), (\bigsqcup x \in X.\ snd\ x))$

definition *sup-prod* $x\ y = (fst\ x \sqcup fst\ y,\ snd\ x \sqcup snd\ y)$

definition *top-prod* $= (top, top)$

definition *less-eq-prod* $x\ y \equiv$ *less-eq* $(fst\ x)\ (fst\ y) \wedge$ *less-eq* $(snd\ x)\ (snd\ y)$

definition *less-prod* $x\ y \equiv$ *less-eq* $(fst\ x)\ (fst\ y) \wedge$ *less-eq* $(snd\ x)\ (snd\ y) \wedge x \neq y$

definition *times-prod'* $x\ y = (fst\ x \cdot fst\ y,\ snd\ x \cdot snd\ y)$

definition *one-prod* = (*1*,*1*)

definition *dual-prod x* = (∂ (*fst x*),∂ (*snd x*))

interpretation *prod*: *complete-lattice Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod* :: ('*a*::*complete-lattice* × '*b*::*complete-lattice*)
by *standard* (*auto simp add*: *Inf-prod-def Sup-prod-def inf-prod-def sup-prod-def bot-prod-def top-prod-def less-eq-prod-def less-prod-def Sup-distl Sup-distr intro*: *Inf-lower Inf-greatest Sup-upper Sup-least*)

interpretation *prod*: *complete-lattice-with-dual Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod* :: ('*a*::*complete-lattice-with-dual* × '*b*::*complete-lattice-with-dual*) *dual-prod*
by *standard* (*simp-all add*: *dual-prod-def fun-eq-iff inj-def Sup-prod-def Inf-prod-def inj-dual-iff Sup-dual-def-var image-comp*)

interpretation *prod*: *proto-near-quantale Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod* :: ('*a*::*proto-near-quantale* × '*b*::*proto-near-quantale*) *times-prod'*
by *standard* (*simp add*: *times-prod'-def Sup-prod-def Sup-distr image-comp*)

interpretation *prod*: *proto-quantale Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod* :: ('*a*::*proto-quantale* × '*b*::*proto-quantale*) *times-prod'*
by *standard* (*simp add*: *times-prod'-def Sup-prod-def less-eq-prod-def Sup-distl image-comp*)

interpretation *prod*: *unital-quantale one-prod times-prod' Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod* :: ('*a*::*unital-quantale* × '*b*::*unital-quantale*)

by *standard* (*simp-all add*: *one-prod-def times-prod'-def ac-simps image-comp*)

2.5 Quantale Morphisms

There are various ways of defining quantale morphisms, depending on the application. Following Rosenthal, I present the most important one.

abbreviation *comp-pres* :: ('*a*::*times* ⇒ '*b*::*times*) ⇒ *bool* **where**
comp-pres f ≡ (∀ *x y*. *f* (*x* · *y*) = *f x* · *f y*)

abbreviation *un-pres* :: ('*a*::*one* ⇒ '*b*::*one*) ⇒ *bool* **where**
un-pres f ≡ (*f 1* = *1*)

definition *comp-closed-set X* = (∀ *x* ∈ *X*. ∀ *y* ∈ *X*. *x* · *y* ∈ *X*)

definition *un-closed-set X* = (*1* ∈ *X*)

definition *quantale-homset* :: ('*a*::*quantale* ⇒ '*b*::*quantale*) *set* **where**
quantale-homset = {*f*. *comp-pres f* ∧ *Sup-pres f*}

lemma *quantale-homset-iff*: *f* ∈ *quantale-homset* = (*comp-pres f* ∧ *Sup-pres f*)

unfolding *quantale-homset-def* **by** *clarsimp*

definition *unital-quantale-homset* :: ('*a*::*unital-quantale* ⇒ '*b*::*unital-quantale*) *set*
where
unital-quantale-homset = {*f*. *comp-pres f* ∧ *Sup-pres f* ∧ *un-pres f*}

lemma *unital-quantale-homset-iff*: *f* ∈ *unital-quantale-homset* = (*comp-pres f* ∧ *Sup-pres f* ∧ *un-pres f*)
unfolding *unital-quantale-homset-def* **by** *clarsimp*

Though Infs can be defined from Sups in any quantale, quantale morphisms do not generally preserve Infs. A different kind of morphism is needed if this is to be guaranteed.

lemma *f* ∈ *quantale-homset* ⟹ *Inf-pres f*
oops

The images of quantale morphisms are closed under compositions and Sups, hence they form quantales.

lemma *quantale-hom-q-pres*: *f* ∈ *quantale-homset* ⟹ *Sup-closed-set* (*range f*) ∧ *comp-closed-set* (*range f*)
apply *safe*
apply (*simp add*: *Sup-pres-Sup-closed quantale-homset-iff*)
unfolding *quantale-homset-iff comp-closed-set-def* **by** (*metis* (*no-types*, *lifting*) *imageE range-eqI*)

Yet the image need not be Inf-closed.

lemma *f* ∈ *quantale-homset* ⟹ *Inf-closed-set* (*range f*)
oops

Of course Sups are preserved by quantale-morphisms, hence they are the same in subsets as in the original set. Infs in the subset, however, exist, since they subset forms a quantale in which Infs can be defined, but these are generally different from the Infs in the superstructure.

This fact is hidden in Isabelle's definition of complete lattices, where Infs are axiomatised. There is no easy way in general to show that images of quantale morphisms form quantales, though the statement for Sup-quantales is straightforward. I show this for quantic nuclei and left-sided elements.

typedef (**overloaded**) ('*a*,'*b*) *quantale-homset* = *quantale-homset*::('*a*::*quantale* ⇒ '*b*::*quantale*) *set*
proof−
have *a*: *comp-pres* (λ*x*::'*a*::*quantale*. *bot*::'*b*)
by *simp*
have *b*: *Sup-pres* (λ*x*::'*a*::*quantale*. *bot*::'*b*)
unfolding *fun-eq-iff comp-def* **by** *simp*
hence (λ*x*::'*a*::*quantale*. *bot*::'*b*) ∈ *quantale-homset*
by (*simp add*: *quantale-homset-iff*)
thus *?thesis*

by *auto*
qed

setup-lifting *type-definition-quantale-homset*

Interestingly, the following type is not (gobally) inhabited.

typedef (**overloaded**) ('*a*,'*b*) *unital-quantale-homset* = *unital-quantale-homset*::('*a*::*unital-quantale* ⇒ '*b*::*unital-quantale*) *set*
oops

lemma *quantale-hom-radj*:
fixes *f* :: '*a*::*quantale-with-dual* ⇒ '*b*::*quantale-with-dual*
shows *f* ∈ *quantale-homset* ⟹ *f* ⊣ *radj f*
unfolding *quantale-homset-iff* **by** (*simp add*: *Sup-pres-ladj-aux*)

lemma *quantale-hom-prop1*:
fixes *f* :: '*a*::*quantale-with-dual* ⇒ '*b*::*quantale-with-dual*
shows *f* ∈ *quantale-homset* ⟹ *radj f* (*f x* → *y*) = *x* → *radj f y*
proof−
assume *h*: *f* ∈ *quantale-homset*
have *f x* · *f* (*radj f* (*f x* → *y*)) ≤ *y*
by (*meson h adj-def bres-galois order-refl quantale-hom-radj*)
hence *f* (*x* · *radj f* (*f x* → *y*)) ≤ *y*
by (*metis h quantale-homset-iff*)
hence *x* · *radj f* (*f x* → *y*) ≤ *radj f y*
using *adj-def h quantale-hom-radj* **by** *blast*
hence *le*: *radj f* (*f x* → *y*) ≤ *x* → *radj f y*
by (*simp add*: *bres-galois*)
have *x* · (*x* → *radj f y*) ≤ *radj f y*
by (*simp add*: *bres-canc1*)
hence *f* (*x* · (*x* → *radj f y*)) ≤ *y*
using *adj-def h quantale-hom-radj* **by** *blast*
hence *f x* · *f* (*x* → *radj f y*) ≤ *y*
by (*metis h quantale-homset-iff*)
hence *f* (*x* → *radj f y*) ≤ *f x* → *y*
by (*simp add*: *bres-galois*)
hence *x* → *radj f y* ≤ *radj f* (*f x* → *y*)
using *adj-def h quantale-hom-radj* **by** *blast*
thus *?thesis*
by (*simp add*: *dual-order.antisym le*)
qed

lemma *quantale-hom-prop2*:
fixes *f* :: '*a*::*quantale-with-dual* ⇒ '*b*::*quantale-with-dual*
shows *f* ∈ *quantale-homset* ⟹ *radj f* (*y* ← *f x*) = *radj f y* ← *x*
proof−
assume *h*: *f* ∈ *quantale-homset*
have *f* (*radj f* (*y* ← *f x*)) · *f x* ≤ *y*
by (*meson adj-def fres-galois h order-refl quantale-hom-radj*)

hence $f\ (radj\ f\ (y \leftarrow f\ x) \cdot x) \leq y$
by (*metis h quantale-homset-iff*)
hence $radj\ f\ (y \leftarrow f\ x) \cdot x \leq radj\ f\ y$
using *adj-def h quantale-hom-radj* **by** *blast*
hence *le*: $radj\ f\ (y \leftarrow f\ x) \leq radj\ f\ y \leftarrow x$
by (*simp add*: *fres-galois*)
have $(radj\ f\ y \leftarrow x) \cdot x \leq radj\ f\ y$
by (*simp add*: *fres-canc1*)
hence $f\ ((radj\ f\ y \leftarrow x) \cdot x) \leq y$
using *adj-def h quantale-hom-radj* **by** *blast*
hence $f\ (radj\ f\ y \leftarrow x) \cdot f\ x \leq y$
by (*metis h quantale-homset-iff*)
hence $f\ (radj\ f\ y \leftarrow x) \leq y \leftarrow f\ x$
by (*simp add*: *fres-galois*)
hence $radj\ f\ y \leftarrow x \leq radj\ f\ (y \leftarrow f\ x)$
using *adj-def h quantale-hom-radj* **by** *blast*
thus *?thesis*
by (*simp add*: *dual-order.antisym le*)
qed

definition *quantale-closed-maps* :: ('*a*::*quantale* ⇒ '*b*::*quantale*) *set* **where**
quantale-closed-maps = {*f*. (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*))}

lemma *quantale-closed-maps-iff*: *f* ∈ *quantale-closed-maps* = (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*))
unfolding *quantale-closed-maps-def* **by** *clarsimp*

definition *quantale-closed-Sup-maps* :: ('*a*::*quantale* ⇒ '*b*::*quantale*) *set* **where**
quantale-closed-Sup-maps = {*f*. (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*)) ∧ *Sup-pres f*}

lemma *quantale-closed-Sup-maps-iff*: *f* ∈ *quantale-closed-Sup-maps* = (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*) ∧ *Sup-pres f*)
unfolding *quantale-closed-Sup-maps-def* **by** *clarsimp*

definition *quantale-closed-unital-maps* :: ('*a*::*unital-quantale* ⇒ '*b*::*unital-quantale*) *set* **where**
quantale-closed-unital-maps = {*f*. (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*)) ∧ *1* ≤ *f 1*}

lemma *quantale-closed-unital-maps-iff*: *f* ∈ *quantale-closed-unital-maps* = (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*) ∧ *1* ≤ *f 1*)
unfolding *quantale-closed-unital-maps-def* **by** *clarsimp*

definition *quantale-closed-unital-Sup-maps* :: ('*a*::*unital-quantale* ⇒ '*b*::*unital-quantale*) *set* **where**
quantale-closed-unital-Sup-maps = {*f*. (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*)) ∧ *Sup-pres f* ∧ *1* ≤ *f 1*}

lemma *quantale-closed-unital-Sup-maps-iff*: *f* ∈ *quantale-closed-unital-Sup-maps* = (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*) ∧ *Sup-pres f* ∧ *1* ≤ *f 1*)

unfolding *quantale-closed-unital-Sup-maps-def* **by** *clarsimp*

Closed maps are the right adjoints of quantale morphisms.

lemma *quantale-hom-closed-map*:
fixes $f :: 'a{::}quantale\text{-}with\text{-}dual \Rightarrow 'b{::}quantale\text{-}with\text{-}dual$
shows $(f \in quantale\text{-}homset) \Longrightarrow (radj\ f \in quantale\text{-}closed\text{-}maps)$
proof−
assume *h*: $f \in quantale\text{-}homset$
have $\forall x\ y.\ f\ (radj\ f\ x) \cdot f\ (radj\ f\ y) \leq x \cdot y$
by (*metis adj-def h order-refl psrpq.mult-isol-var quantale-hom-radj*)
hence $\forall x\ y.\ f\ (radj\ f\ x \cdot radj\ f\ y) \leq x \cdot y$
by (*metis h quantale-homset-iff*)
hence $\forall x\ y.\ radj\ f\ x \cdot radj\ f\ y \leq radj\ f\ (x \cdot y)$
using *adj-def h quantale-hom-radj* **by** *blast*
thus *?thesis*
by (*simp add*: *quantale-closed-maps-iff*)
qed

lemma *unital-quantale-hom-closed-unital-map*:
fixes $f :: 'a{::}unital\text{-}quantale\text{-}with\text{-}dual \Rightarrow 'b{::}unital\text{-}quantale\text{-}with\text{-}dual$
shows $(f \in unital\text{-}quantale\text{-}homset) \Longrightarrow (radj\ f \in quantale\text{-}closed\text{-}unital\text{-}maps)$
by (*metis* (*no-types*, *hide-lams*) *adj-def order-refl quantale-closed-maps-iff quantale-closed-unital-maps-iff quantale-hom-closed-map quantale-hom-radj quantale-homset-iff unital-quantale-homset-iff*)

end

3 Star and Fixpoints in Quantales

theory *Quantale-Star*
imports *Quantales*
Kleene-Algebra.Kleene-Algebra

begin

This component formalises properties of the star in quantales. For pre-quantales these are modelled as fixpoints. For weak quantales they are given by iteration.

3.1 Star and Fixpoints in Pre-Quantales

context *unital-near-quantale*
begin

definition $qiter\text{-}fun\ x\ y = (\sqcup)\ x \circ (\cdot)\ y$

definition $qiterl\ x\ y = lfp\ (qiter\text{-}fun\ x\ y)$

definition $qiterg\ x\ y = gfp\ (qiter\text{-}fun\ x\ y)$

abbreviation *qiterl-id* ≡ *qiterl 1*

abbreviation *qiterq-id* ≡ *qiterg 1*

definition *qstar x* = (⨆ *i*. *x* ^ *i*)

lemma *qiter-fun-exp*: *qiter-fun x y z* = *x* ⊔ *y* · *z*
unfolding *qiter-fun-def* **by** *simp*

end

Many properties of fixpoints have been developed for Isabelle's monotone functions. These carry two type parameters, and must therefore be used outside of contexts.

lemma *iso-qiter-fun*: *mono* (λ*z*::′*a*::*unital-pre-quantale*. *qiter-fun x y z*)
by (*metis monoI proto-pre-quantale-class.mult-isol qiter-fun-exp sup.idem sup.mono sup.orderI*)

I derive the left unfold and induction laws of Kleene algebra. I link with the Kleene algebra components at the end of this section, to bring properties into scope.

lemma *qiterl-unfoldl* [*simp*]: *x* ⊔ *y* · *qiterl x y* = *qiterl* (*x*::′*a*::*unital-pre-quantale*) *y*
by (*metis iso-qiter-fun lfp-unfold qiter-fun-exp qiterl-def*)

lemma *qiterg-unfoldl* [*simp*]: *x* ⊔ *y* · *qiterg x y* = *qiterg* (*x*::′*a*::*unital-pre-quantale*) *y*
by (*metis gfp-fixpoint iso-qiter-fun qiter-fun-exp qiterg-def*)

lemma *qiterl-inductl*: *x* ⊔ *y* · *z* ≤ *z* ⟹ *qiterl* (*x*::′*a*::*unital-near-quantale*) *y* ≤ *z*
by (*simp add*: *lfp-lowerbound qiterl-def qiter-fun-def*)

lemma *qiterg-coinductl*: *z* ≤ *x* ⊔ *y* · *z* ⟹ *z* ≤ *qiterg* (*x*::′*a*::*unital-near-quantale*) *y*
by (*simp add*: *gfp-upperbound qiterg-def qiter-fun-def*)

context *unital-near-quantale*
begin

lemma *powers-distr*: *qstar x* · *y* = (⨆ *i*. *x* ^ *i* · *y*)
by (*simp add*: *qstar-def Sup-distr image-comp*)

lemma *Sup-iter-unfold*: *x* ^ *0* ⊔ (⨆ *n*. *x* ^ (*Suc n*)) = (⨆ *n*. *x* ^ *n*)
using *fSup-unfold* **by** *presburger*

lemma *Sup-iter-unfold-var*: *1* ⊔ (⨆ *n*. *x* · *x* ^ *n*) = (⨆ *n*. *x* ^ *n*)
by (*simp only*: *Sup-iter-unfold*[*symmetric*]) *auto*

lemma *power-inductr*: $z \sqcup y \cdot x \leq y \Longrightarrow z \cdot x \hat{} i \leq y$
by (*induct i*, *simp-all*, *metis power-commutes eq-iff mult-isor order.trans mult-assoc power.power.power-Suc*)

end

context *unital-pre-quantale*
begin

lemma *powers-subdistl*: $(\bigsqcup i.\ x \cdot y \hat{} i) \leq x \cdot qstar\ y$
by (*simp add*: *SUP-le-iff SUP-upper mult-isol qstar-def*)

lemma *qstar-subcomm*: $qstar\ x \cdot x \leq x \cdot qstar\ x$
by (*simp add*: *powers-distr powers-subdistl power-commutes*)

lemma *power-inductl*: $z \sqcup x \cdot y \leq y \Longrightarrow x \hat{} i \cdot z \leq y$
by (*induct i*, *simp-all*, *metis dual-order.trans mult-isol mult-assoc*)

end

3.2 Star and Iteration in Weak Quantales

context *unital-weak-quantale*
begin

In unital weak quantales one can derive the star axioms of Kleene algebra for iteration. This generalises the language and relation models from our Kleene algebra components.

lemma *powers-distl*: $x \cdot qstar\ y = (\bigsqcup i.\ x \cdot y \hat{} i)$
by (*simp add*: *qstar-def weak-Sup-distl image-comp*)

lemma *qstar-unfoldl*: $1 \sqcup x \cdot qstar\ x \leq qstar\ x$
by (*simp only*: *powers-distl Sup-iter-unfold-var*, *simp add*: *qstar-def*)

lemma *qstar-comm*: $x \cdot qstar\ x = qstar\ x \cdot x$
using *power-commutes powers-distr powers-distl* **by** *auto*

lemma *qstar-unfoldr* [*simp*]: $1 \sqcup qstar\ x \cdot x = qstar\ x$
using *qstar-comm qstar-unfoldl Sup-iter-unfold-var qstar-def powers-distl* **by** *auto*

lemma *qstar-inductl*: $z \sqcup x \cdot y \leq y \Longrightarrow qstar\ x \cdot z \leq y$
by (*subst powers-distr*, *auto intro*!: *Sup-least power-inductl*)

lemma *qstar-inductr*: $z \sqcup y \cdot x \leq y \Longrightarrow z \cdot qstar\ x \leq y$
by (*subst powers-distl*, *auto intro*!: *Sup-least power-inductr*)

Hence in this setting one also obtains the right Kleene algebra axioms.

end

sublocale *unital-weak-quantale* $\subseteq$ *uwqlka*: *left-kleene-algebra-zerol* $(\sqcup)$ $(\cdot)$ *1* $\bot$ $(\leq)$ $(<)$ *qstar*
apply *unfold-locales*
using *qstar-unfoldl* **apply** *blast*
by (*simp add*: *qstar-inductl*)

sublocale *unital-quantale* $\subseteq$ *uqka*: *kleene-algebra* $(\sqcup)$ $(\cdot)$ *1* $\bot$ $(\leq)$ $(<)$ *qstar*
by *unfold-locales* (*simp add*: *qstar-inductr*)

The star is indeed the least fixpoint.

lemma *qstar-qiterl*: *qstar* $(x::'a::$*unital-weak-quantale*$)$ = *qiterl-id x*
by (*simp add*: *antisym qiterl-inductl uwqlka.star-inductl-one*)

context *unital-weak-quantale*
begin

3.3 Deriving the Star Axioms of Action Algebras

Finally the star axioms of action algebras are derived.

lemma *act-star1*: *1* $\sqcup$ *x* $\sqcup$ (*qstar x*) $\cdot$ (*qstar x*) $\leq$ (*qstar x*)
by *simp*

lemma (**in** *unital-quantale*) *act-star3*: *qstar* $(x \rightarrow x) \leq x \rightarrow x$
by (*simp add*: *bres-canc1 bres-galois-imp uqka.star-inductr-var*)

lemma *act-star3-var*: *qstar* $(x \leftarrow x) \leq x \leftarrow x$
using *fres-galois qstar-inductl* **by** *auto*

end

An integration of action algebras requires first resolving some notational issues within the components where these algebras are located.

end

4 Quantale Modules and Semidirect Products

theory *Quantale-Modules*
imports *Quantale-Star*

begin

4.1 Quantale Modules

Quantale modules are extensions of semigroup actions in that a quantale acts on a complete lattice. These have been investigated by Abramsky and Vickers [1] and others, predominantly in the context of pointfree topology.

locale *unital-quantale-module* =
  fixes *act* :: *'a*::*unital-quantale* ⇒ *'b*::*complete-lattice-with-dual* ⇒ *'b* (*α*)
  assumes *act1*: *α* (*x* · *y*) *p* = *α x* (*α y p*)
    and *act2* [*simp*]: *α 1 p* = *p*
    and *act3*: *α* (⨆ *X*) *p* = (⨆ *x* ∈ *X*. *α x p*)
    and *act4*: *α x* (⨆ *P*) = (⨆ *p* ∈ *P*. *α x p*)

context *unital-quantale-module*
begin

Actions are morphisms. The curried notation is particularly convenient for this.

lemma *act-morph1*: *α* (*x* · *y*) = (*α x*) ∘ (*α y*)
  by (*simp add*: *fun-eq-iff act1*)

lemma *act-morph2* [*simp*]: *α 1* = *id*
  by (*simp add*: *fun-eq-iff*)

lemma *emp-act* [*simp*]: *α* (⨆ {}) *p* = ⊥
  by (*simp only*: *act3*, *force*)

lemma *emp-act-var* [*simp*]: *α* ⊥ *p* = ⊥
  using *emp-act* **by** *auto*

lemma *act-emp* [*simp*]: *α x* (⨆ {}) = ⊥
  by (*simp only*: *act4*, *force*)

lemma *act-emp-var* [*simp*]: *α x* ⊥ = ⊥
  using *act-emp* **by** *auto*

lemma *act-sup-distl*: *α x* (*p* ⊔ *q*) = (*α x p*) ⊔ (*α x q*)
  by (*metis* (*mono-tags*, *lifting*) *act4 image-insert image-is-empty sup-Sup*)

lemma *act-sup-distr*: *α* (*x* ⊔ *y*) *p* = (*α x p*) ⊔ (*α y p*)
  by (*metis* (*no-types*, *lifting*) *SUP-insert Sup-empty Sup-insert act3 sup-bot-right*)

lemma *act-sup-distr-var*: *α* (*x* ⊔ *y*) = (*α x*) ⊔ (*α y*)
  by (*simp add*: *fun-eq-iff act-sup-distr*)

4.2 Semidirect Products and Weak Quantales

Next, the semidirect product of a unital quantale and a complete lattice is defined.

definition *sd-prod x y* = (*fst x* · *fst y*, *snd x* ⊔ *α* (*fst x*) (*snd y*))

lemma *sd-distr*: *sd-prod* (*Sup-prod X*) *y* = *Sup-prod* {*sd-prod x y* |*x*. *x* ∈ *X*}
proof −
  have *sd-prod* (*Sup-prod X*) *y* = *sd-prod* ((⨆ *x* ∈ *X*. *fst x*), (⨆ *x* ∈ *X*. *snd x*)) *y*

by (*simp add*: *Sup-prod-def*)
also have ... = ((⨆ *x* ∈ *X*. *fst x*) · *fst y*, (⨆ *x* ∈ *X*. *snd x*) ⊔ (α (⨆ *x* ∈ *X*. *fst x*) (*snd y*)))
by (*simp add*: *sd-prod-def*)
also have ... = ((⨆ *x* ∈ *X*. (*fst x* · *fst y*)), (⨆ *x* ∈ *X*. *snd x*) ⊔ (α (⨆ *x* ∈ *X*. *fst x*) (*snd y*)))
by (*simp add*: *Sup-distr image-comp*)
also have ... = ((⨆ *x* ∈ *X*. (*fst x* · *fst y*)), (⨆ *x* ∈ *X*. *snd x*) ⊔ (⨆ *x* ∈ *X*. (α (*fst x*) (*snd y*))))
by (*simp add*: *act3 image-comp*)
also have ... = ((⨆ *x* ∈ *X*. (*fst x* · *fst y*)), (⨆ *x* ∈ *X*. (*snd x* ⊔ (α (*fst x*) (*snd y*)))))
using *complete-lattice-class.SUP-sup-distrib* **by** *fastforce*
also have ... = *Sup-prod* {(*fst x* · *fst y*, *snd x* ⊔ (α (*fst x*) (*snd y*))) |*x*. *x* ∈ *X*}
apply (*unfold Sup-prod-def*, *safe*)
by (*simp add*: *SUP-Sup-eq*, *rule-tac f*=*Sup* **in** *arg-cong*, *force*)+
finally show *?thesis*
by (*simp add*: *sd-prod-def*)
qed

lemma *sd-distl-aux*: *Y* ≠ {} ⟹ *p* ⊔ (⨆ *y* ∈ *Y*. α *x* (*snd y*)) = (⨆ *y* ∈ *Y*. *p* ⊔ α *x* (*snd y*))
apply (*rule antisym*)
apply (*rule sup-least*, *metis SUP-bot-conv*(*2*) *SUP-const SUP-upper2 sup-ge1*)
apply (*rule Sup-least*, *force simp*: *SUP-upper2 image-iff*)
by (*rule Sup-least*, *safe*, *metis Sup-upper image-iff sup.idem sup.mono sup-ge2*)

lemma *sd-distl*: *Y* ≠ {} ⟹ *sd-prod x* (*Sup-prod Y*) = *Sup-prod* {*sd-prod x y* |*y*. *y* ∈ *Y*}
proof −
assume *a*: *Y* ≠ {}
have *sd-prod x* (*Sup-prod Y*) = *sd-prod x* ((⨆ *y* ∈ *Y*. *fst y*), (⨆ *y* ∈ *Y*. *snd y*))
by (*simp add*: *Sup-prod-def*)
also have ... = ((*fst x*) · (⨆ *y* ∈ *Y*. *fst y*), (*snd x* ⊔ (α (*fst x*) (⨆ *y* ∈ *Y*. *snd y*))))
by (*simp add*: *sd-prod-def*)
also have ... = ((⨆ *y* ∈ *Y*. *fst x* · *fst y*), (*snd x* ⊔ (α (*fst x*) (⨆ *y* ∈ *Y*. *snd y*))))
by (*simp add*: *Sup-distl image-comp*)
also have ... = ((⨆ *y* ∈ *Y*. *fst x* · *fst y*), (*snd x* ⊔ (⨆ *y* ∈ *Y*. α (*fst x*) (*snd y*))))
by (*simp add*: *act4 image-comp*)
also have ... = ((⨆ *y* ∈ *Y*. *fst x* · *fst y*), (⨆ *y* ∈ *Y*. *snd x* ⊔ (α (*fst x*) (*snd y*))))
using *a sd-distl-aux* **by** *blast*
also have ... = *Sup-prod* {(*fst x* · *fst y*, *snd x* ⊔ (α (*fst x*) (*snd y*))) |*y*. *y* ∈ *Y*}
apply (*unfold Sup-prod-def*, *safe*)
by (*simp add*: *SUP-Sup-eq*, *rule-tac f*=*Sup* **in** *arg-cong*, *force*)+
finally show *?thesis*
by (*simp add*: *sd-prod-def*)
qed

definition *sd-unit* = (*1*,⊥)

lemma *sd-unitl* [*simp*]: *sd-prod sd-unit x* = *x*
by (*simp add*: *sd-prod-def sd-unit-def*)

lemma *sd-unitr* [*simp*]: *sd-prod x sd-unit* = *x*
by (*simp add*: *sd-prod-def sd-unit-def*)

The following counterexamples rule out that semidirect products of quantales and complete lattices form quantales. The reason is that the right annihilation law fails.

lemma *sd-prod x* (*Sup-prod Y*) = *Sup-prod* {*sd-prod x y* |*y*. *y* ∈ *Y*}
oops

lemma *sd-prod x bot-prod* = *bot-prod*
oops

However one can show that semidirect products of (unital) quantales with complete lattices form weak (unital) quantales. This provides an example of how weak quantales arise quite naturally.

sublocale *dp-quantale*: *weak-quantale sd-prod Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod*
apply *unfold-locales*
apply (*simp add*: *act1 act-sup-distl mult.assoc sd-prod-def sup-assoc*)
apply (*metis Setcompr-eq-image sd-distr*)
by (*metis Setcompr-eq-image sd-distl*)

sublocale *dpu-quantale*: *unital-weak-quantale sd-unit sd-prod Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod*
by *unfold-locales simp-all*

4.3 The Star in Semidirect Products

I define the star operation for a semidirect product of a quantale and a complete lattice, and prove a characteristic property.

abbreviation *sd-power* ≡ *dpu-quantale.power*

abbreviation *sd-star* ≡ *dpu-quantale.qstar*

lemma *sd-power-zero* [*simp*]: *sd-power x 0* = (*1*,⊥)
using *sd-unit-def* **by** *simp*

lemma *sd-power-prop-aux*: *α* (*x* ^ *0*) *y* ⊔ (⨆{*α* (*x* ^ *k*) *y* |*k*. *0* < *k* ∧ *k* ≤ *Suc i*}) = ⨆{*α* (*x* ^ *k*) *y* |*k*. *k* ≤ *Suc i*}
apply (*rule antisym*)
apply (*rule sup-least*, *intro Sup-upper*, *blast*, *intro Sup-mono*, *blast*)
apply (*rule Sup-least*, *safe*)

by (*metis* (*mono-tags*, *lifting*) *Sup-insert Sup-upper insert-iff le-0-eq linorder-not-le mem-Collect-eq*)

lemma *sd-power-prop1* [*simp*]: *sd-power* (*x*,*y*) (*Suc i*) = (*x* ^ (*Suc i*), ⨆{*α* (*x* ^ *k*) *y*|*k*. *k* ≤ *i*})
proof (*induct i*)
case *0* **thus** *?case*
by *auto*
next
case (*Suc i*) **thus** *?case*
proof−
assume *sd-power* (*x*,*y*) (*Suc i*) = (*x* ^ *Suc i*, ⨆{*α* (*x* ^ *k*) *y* |*k*. *k* ≤ *i*})
hence *sd-power* (*x*,*y*) (*Suc* (*Suc i*)) = (*x* ^ *Suc* (*Suc i*), *α* (*x* ^ *0*) *y* ⊔ (⨆{*α* (*x* ^ (*Suc k*)) *y* |*k*. *k* ≤ *i*}))
apply (*simp add*: *sd-prod-def act1 act4*)
apply *safe*
by (*metis act4 image-Collect image-image*)
also have ... = (*x* ^ *Suc* (*Suc i*), *α* (*x* ^ *0*) *y* ⊔ (⨆{*α* (*x* ^ *k*) *y* |*k*. *0* < *k* ∧ *k* ≤ *Suc i*}))
by (*metis* (*no-types*, *hide-lams*) *Suc-le-eq Suc-le-mono le-0-eq not0-implies-Suc zero-less-Suc*)
finally show *?thesis*
using *sd-power-prop-aux* **by** *simp*
qed
qed

lemma *sd-power-prop2* [*simp*]: *sd-power* (*x*,*y*) *i* = (*x* ^ *i*, ⨆{*α* (*x* ^ *k*) *y*|*k*. *k* < *i*})
apply (*case-tac i*)
using *sd-unit-def* **apply** *force*
using *le-simps*(*2*) *unital-quantale-module.sd-power-prop1 unital-quantale-module-axioms* **by** *fastforce*

lemma *sdstar-prop*: *sd-star* (*x*,*y*) = (*qstar x*, *α* (*qstar x*) *y*)
proof −
have *sd-star* (*x*,*y*) = *sup-prod* (*1*,⊥) (*Sup-prod* {*sd-power* (*x*,*y*) (*Suc i*) |*i*. *True*})
by (*metis dpu-quantale.Sup-iter-unfold dpu-quantale.qstar-def full-SetCompr-eq sd-power-zero*)
also have ... = *sup-prod* (*1*,⊥) (*Sup-prod* {(*x*^(*Suc i*), ⨆{*α* (*x* ^ *k*) *y*|*k*. *k* ≤ *i*}) |*i*. *True*})
using *sd-power-prop1* **by** *auto*
also have ... = (*1* ⊔(⨆*i*. *x* ^ (*Suc i*)), (⨆ *i*. ⨆{*α* (*x* ^ *k*) *y*|*k*. *k* ≤ *i*}))
apply (*unfold sup-prod-def Sup-prod-def*, *safe*)
apply (*simp*, *rule-tac f*=*λx*. *1* ⊔ *x* **in** *arg-cong*)
by (*simp add*: *SUP-Sup-eq*, *rule-tac f*=*Sup* **in** *arg-cong*, *force*)+
also have ... = (*qstar x*, (⨆ *i*. ⨆{*α* (*x* ^ *k*) *y*|*k*. *k* ≤ *i*}))
by (*safe*, *metis* (*no-types*) *fSup-unfold power-0 qstar-def*)
also have ... = (*qstar x*, (⨆ *i*. *α* (*x* ^ *i*) *y*))
apply *safe*

apply (*rule antisym*)
apply (*safe intro*!: *Sup-least*, *simp add*: *Sup-upper*)
by (*metis* (*mono-tags*, *lifting*) *SUP-upper2 Sup-upper mem-Collect-eq order-refl*)
finally show *?thesis*
by (*simp add*: *qstar-def act3 image-comp*)
qed

end

end

5 Models of Quantales

theory *Quantale-Models*
imports *Quantales*
Dioid-Models-New

begin

Most of the models in this section expand those of dioids (and hence Kleene algebras). They really belong here, because quantales form a stronger setting than dioids or Kleene algebras. They are, however, subsumed by the general lifting results of partial semigroups and monoids from another AFP entry [3].

5.1 Quantales of Booleans

instantiation *bool* :: *bool-ab-unital-quantale*
begin

definition *one-bool* = *True*

definition *times-bool* = (∧)

instance
by *intro-classes* (*auto simp*: *times-bool-def one-bool-def*)

end

lemma *bool-sep-eq-conj* [*simp*]: ((*x* :: *bool*) ∗ *y*) = (*x* ∧ *y*)
by (*auto simp*: *times-bool-def*)

lemma *bool-impl-eq* [*simp*]: (*x* :: *bool*) → *y* = −*x* ⊔ *y*
by (*clarsimp simp*: *bres-def*)

5.2 Powerset Quantales of Semigroups and Monoids

instance *set* :: (*semigroup-mult*) *quantale*
by *intro-classes* (*auto simp*: *times-set-def*)

With the definition of products on powersets (from the component of models of dioids) one can prove the following mixed distributivity law.

lemma *comp-dist-mix*: $\bigsqcup(X::'a::quantale\ set) \cdot \bigsqcup Y = \bigsqcup(X \cdot Y)$
proof −
have $\bigsqcup X \cdot \bigsqcup Y = (\bigsqcup x \in X.\ \bigsqcup y \in Y.\ x \cdot y)$
by (*metis* (*no-types*, *lifting*) *SUP-cong Sup-distl Sup-distr*)
also have ... $= \bigsqcup\{\bigsqcup\{x \cdot y | y.\ y \in Y\}\ | x.\ x \in X\}$
by (*simp add*: *setcompr-eq-image*)
also have ... $= \bigsqcup\{x \cdot y\ | x\ y.\ x \in X \wedge y \in Y\}$
apply (*rule antisym*)
apply (*rule Sup-least*, *smt Collect-mono Sup-subset-mono mem-Collect-eq*)
by (*rule Sup-least*, *smt Sup-upper calculation mem-Collect-eq psrpq.mult-isol-var*)
finally show *?thesis*
by (*simp add*: *times-set-def*)
qed

Powerset quantales of monoids can nevertheless be formalised as instances.

instance *set* :: (*monoid-mult*) *unital-quantale***..**

There is much more to this example. In fact, every quantale can be represented, up to isomorphism, as a certain quotient of a powerset quantale over some semigroup [6]. This representation theorem is formalised in the section on nuclei.

5.3 Language, Relation, Trace and Path Quantales

The language quantale is implcit in the powerset quantale over a semigroup or monoid. The free list monoid has already been linked with the class of monoid as an instance in Isabelle's dioid components [2]. I provide an alternative interpretation. In all of the following locale statements, an interpretation for Sup-quantales fails, due to the occurance of some low level less operations in the underlying model...

interpretation *lan-quantale*: *unital-quantale 1*::*'a lan* (·) *Inf Sup inf* (⊆) (⊂) *sup 0 UNIV*
by *unfold-locales* (*simp-all add*: *Inf-lower Inter-greatest Sup-upper Sup-least zero-set-def Sup-distr Sup-distl*)

The relation quantale follows.

interpretation *rel-quantale*: *unital-quantale Id relcomp Inf Sup inf* (⊆) (⊂) *sup* {} *UNIV*
by (*unfold-locales*, *auto*)

Traces alternate between state and action symbols, the first and last symbol of a trace being state symbols. They can be associated with behaviours of automata or executions of programs.

interpretation *trace-quantale*: *unital-quantale t-one t-prod Inf Sup inf* (⊆) (⊂) *sup t-zero UNIV*
by *unfold-locales* (*auto simp*: *Inf-lower Inf-greatest Sup-upper Sup-least t-zero-def t-prod-def t-fusion-def*)

The final model corresponds to paths as sequences of states of an automata, transition system or graph.

interpretation *path-quantale*: *unital-quantale p-one p-prod Inter Union* (∩) (⊆) (⊂) (∪) {} *UNIV*
by *unfold-locales* (*auto simp*: *p-prod-def*)

Rosenthal's book contains a wealth of other examples. Many of them come from ring theory (e.g. the additive subgroups of a ring form a quantale and so do the left, right and two-sided ideals). Others are based on the interval $[0, \infty]$. The first line of models was the original motivation for the study of quantales, the second one is important to Lawvere's categorical generalisation of metric spaces. These examples are left for future consideration.

end

6 Quantic Nuclei and Conuclei

theory *Quantic-Nuclei-Conuclei*
imports *Quantale-Models*

begin

Quantic nuclei and conuclei are an important part of the structure theory of quantales. I formalise the basics, following Rosenthal's book [6]. In the structure theorems, I collect all parts of the proof, but do not present the theorems in compact form due to difficulties to speak about subalgebras and homomorphic images without explicit carrier sets. Nuclei also arise in the context of complete Heyting algebras, frames and locales [5]. Their formalisation seems an interesting future task.

6.1 Nuclei

definition *nucleus* :: ('*a*::*quantale* ⇒ '*a*::*quantale*) ⇒ *bool* **where**
nucleus f = (*clop f* ∧ (∀ *x y*. *f x* · *f y* ≤ *f* (*x* · *y*)))

lemma *nuc-lax*: *nucleus f* ⟹ *f x* · *f y* ≤ *f* (*x* · *y*)
by (*simp add*: *nucleus-def*)

definition *unucleus* :: ('*a*::*unital-quantale* ⇒ '*a*::*unital-quantale*) ⇒ *bool* **where**
unucleus f = (*nucleus f* ∧ *1* ≤ *f 1*)

lemma *nucleus f* ⟹ *f* ⊥ = ⊥
oops

lemma *conucleus f* $\Longrightarrow$ *f* $\top = \top$
oops

lemma *nuc-prop1*: *nucleus f* $\Longrightarrow$ *f* $(x \cdot y) = f\ (x \cdot f\ y)$
apply (*rule antisym*)
apply (*simp add*: *clop-extensive-var clop-iso-var nucleus-def psrpq.mult-isol*)
by (*metis clop-alt clop-extensive-var dual-order.trans nucleus-def psrpq.mult-isol-var*)

lemma *nuc-prop2*: *nucleus f* $\Longrightarrow$ *f* $(x \cdot y) = f\ (f\ x \cdot y)$
apply (*rule antisym*)
apply (*simp add*: *clop-extensive-var clop-iso-var nsrnq.mult-isor nucleus-def*)
by (*metis* (*mono-tags*, *hide-lams*) *clop-alt nuc-prop1 nucleus-def*)

lemma *nuc-comp-prop*: *nucleus f* $\Longrightarrow$ *f* $(f\ x \cdot f\ y) = f\ (x \cdot y)$
using *nuc-prop1 nuc-prop2* **by** *force*

lemma *nucleus-alt-def1*: *nucleus f* $\Longrightarrow f\ x \rightarrow f\ y = x \rightarrow f\ y$
proof (*rule antisym*)
assume *h*: *nucleus f*
hence $x \leq f\ x$
by (*simp add*: *clop-def nucleus-def clop-extensive-var*)
thus $f\ x \rightarrow f\ y \leq x \rightarrow f\ y$
by (*simp add*: *bres-anti*)
have $f\ x \cdot (x \rightarrow f\ y) \leq f\ x \cdot f\ (x \rightarrow f\ y)$
using *clop-extensive-var h nucleus-def proto-pre-quantale-class.mult-isol* **by** *blast*
also have ... $\leq f\ (x \cdot (x \rightarrow f\ y))$
by (*simp add*: *h nuc-lax*)
also have ... $\leq f\ (f\ y)$
using *h* **by** (*simp add*: *bres-canc1 nucleus-def clop-iso-var*)
finally have $f\ x \cdot (x \rightarrow f\ y) \leq f\ y$
using *h* **by** (*metis clop-alt dual-order.trans nucleus-def order-refl*)
thus $x \rightarrow f\ y \leq f\ x \rightarrow f\ y$
by (*simp add*: *bres-galois-imp*)
qed

lemma *nucleus-alt-def2*: *nucleus f* $\Longrightarrow f\ y \leftarrow f\ x = f\ y \leftarrow x$
proof (*rule antisym*)
assume *h*: *nucleus f*
hence $x \leq f\ x$
by (*simp add*: *clop-extensive-var nucleus-def*)
thus $f\ y \leftarrow f\ x \leq f\ y \leftarrow x$
by (*simp add*: *fres-anti*)
have $(f\ y \leftarrow x) \cdot f\ x \leq f\ (f\ y \leftarrow x) \cdot f\ x$
using *clop-extensive-var h nsrnq.mult-isor nucleus-def* **by** *blast*
also have ... $\leq f\ ((f\ y \leftarrow x) \cdot x)$
by (*simp add*: *h nuc-lax*)
also have ... $\leq f\ (f\ y)$

using *clop-iso-var fres-canc1 h nucleus-def* **by** *blast*
finally have $(f\ y \leftarrow x) \cdot f\ x \leq f\ y$
using *h* **by** (*metis clop-alt dual-order.trans nucleus-def order-refl*)
thus $f\ y \leftarrow x \leq f\ y \leftarrow f\ x$
by (*simp add*: *fres-galois*)
qed

lemma *nucleus-alt-def3*:
fixes $f :: 'a{::}unital\text{-}quantale \Rightarrow 'a$
shows $\forall\, x\ y.\ f\ x \rightarrow f\ y = x \rightarrow f\ y \Longrightarrow \forall\, x\ y.\ f\ y \leftarrow f\ x = f\ y \leftarrow x \Longrightarrow nucleus\ f$
proof−
assume *h1*: $\forall\, x\ y.\ f\ x \rightarrow f\ y = x \rightarrow f\ y$
and *h2*: $\forall\, x\ y.\ f\ y \leftarrow f\ x = f\ y \leftarrow x$
hence *ext*: $\forall\, x.\ x \leq f\ x$
by (*metis* (*full-types*) *fres-galois fres-one mult.left-neutral*)
have *iso*: $\forall\, x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y$
by (*metis* (*full-types*) *bres-galois dual-order.trans h1 ext nsrnqo.mult-oner*)
have *trans*: $\forall\, x.\ f\ (f\ x) \leq f\ x$
by (*metis fres-canc2 fres-galois h2 nsrnqo.mult-onel*)
have *lax*: $\forall\, x\ y.\ f\ x \cdot f\ y \leq f\ (x \cdot y)$
by (*metis h1 h2 bres-galois ext fres-galois*)
show *?thesis*
by (*simp add*: *clop-def iso lax le-funI ext trans monoI nucleus-def*)
qed

lemma *nucleus-alt-def*:
fixes $f :: 'a{::}unital\text{-}quantale \Rightarrow 'a$
shows $nucleus\ f = (\forall\ x\ y.\ f\ x \rightarrow f\ y = x \rightarrow f\ y \wedge f\ y \leftarrow f\ x = f\ y \leftarrow x)$
using *nucleus-alt-def1 nucleus-alt-def2 nucleus-alt-def3* **by** *blast*

lemma *nucleus-alt-def-cor1*: $nucleus\ f \Longrightarrow f\ (x \rightarrow y) \leq x \rightarrow f\ y$
by (*metis bres-galois bres-iso clop-extensive-var fres-galois nucleus-alt-def2 nucleus-def*)

lemma *nucleus-alt-def-cor2*: $nucleus\ f \Longrightarrow f\ (y \leftarrow x) \leq f\ y \leftarrow x$
by (*metis bres-galois clop-extensive-var fres-galois fres-iso nucleus-alt-def1 nucleus-def*)

lemma *nucleus-ab-unital*:
fixes $f :: 'a{::}ab\text{-}unital\text{-}quantale \Rightarrow 'a$
shows $nucleus\ f = (\forall\, x\ y.\ f\ x \rightarrow f\ y = x \rightarrow f\ y)$
by (*simp add*: *bres-fres-eq nucleus-alt-def*)

lemma *nuc-comp-assoc*: $nucleus\ f \Longrightarrow f\ (x \cdot f\ (y \cdot z)) = f\ (f\ (x \cdot y) \cdot z)$
by (*metis mult.assoc nuc-prop1 nuc-prop2*)

lemma *nuc-Sup-closed*: $nucleus\ f \Longrightarrow f \circ Sup \circ (`)\ f = (f \circ Sup)$
apply (*simp add*: *nucleus-def fun-eq-iff*, *safe*, *rule antisym*)
apply (*meson SUP-least Sup-upper clop-alt clop-def monoD*)
by (*simp add*: *SUP-upper2 Sup-le-iff clop-extensive-var clop-iso-var*)

lemma *nuc-Sup-closed-var*: *nucleus f* $\Longrightarrow$ *f* $(\bigsqcup x \in X.\ f\ x) = f\ (\bigsqcup X)$
by (*metis nuc-Sup-closed o-apply*)

lemma *nuc-Inf-closed*: *nucleus f* $\Longrightarrow$ *Sup-closed-set* (*Fix f*)
oops

lemma *nuc-Inf-closed*: *nucleus f* $\Longrightarrow$ *Inf-closed-set* (*Fix f*)
by (*simp add*: *clop-Inf-closed nucleus-def*)

lemma *nuc-comp-distl*: *nucleus f* $\Longrightarrow$ *f* $(x \cdot f\ (\bigsqcup Y)) = f\ (\bigsqcup y \in Y.\ f\ (x \cdot y))$
by (*metis Sup-distl image-image nuc-Sup-closed-var nuc-prop1*)

lemma *nuc-comp-distr*: *nucleus f* $\Longrightarrow$ *f* $(f\ (\bigsqcup X) \cdot y) = f\ (\bigsqcup x \in X.\ f\ (x \cdot y))$
by (*metis image-image Sup-distr nuc-Sup-closed-var nuc-prop2*)

lemma *nucleus f* $\Longrightarrow$ *f* $(x \cdot y) = f\ x \cdot f\ y$
oops

lemma *nuc-bres-closed*: *nucleus f* $\Longrightarrow$ *f* $(f\ x \rightarrow f\ y) = f\ x \rightarrow f\ y$
unfolding *nucleus-def* **apply** *clarsimp*
by (*smt clop-closure clop-extensive-var nucleus-alt-def-cor1 nucleus-def order-class.order.antisym rangeI*)

lemma *nucleus f* $\Longrightarrow$ *f* $(x \rightarrow y) = f\ x \rightarrow f\ y$
oops

lemma *nuc-fres-closed*: *nucleus f* $\Longrightarrow$ *f* $(f\ x \leftarrow f\ y) = f\ x \leftarrow f\ y$
unfolding *nucleus-def* **apply** *clarsimp*
by (*smt clop-closure clop-extensive-var eq-iff nucleus-alt-def-cor2 nucleus-def rangeI*)

lemma *nuc-fres-closed*: *nucleus f* $\Longrightarrow$ *f* $(x \leftarrow y) = f\ x \leftarrow f\ y$
oops

lemma *nuc-inf-closed*: *nucleus f* $\Longrightarrow$ *inf-closed-set* (*Fix f*)
by (*simp add*: *Inf-inf-closed nuc-Inf-closed*)

lemma *nuc-inf-closed-var*: *nucleus f* $\Longrightarrow$ *f* $(f\ x \sqcap f\ y) = f\ x \sqcap f\ y$
by (*smt antisym-conv clop-alt clop-extensive-var inf-le2 inf-sup-ord*(*1*) *le-inf-iff nucleus-def*)

Taken together these facts show that, for $f : Q \to Q$, $f[Q]$ forms a quantale with composition $f\,(- \cdot -)$ and sup $f\,(\bigsqcup -)$, and that $f : Q \to f[Q]$ is a quantale morphism. This is the first part of Theorem 3.1.1 in Rosenthal's book.

class *quantale-with-nuc* = *quantale* + *cl-op* +
assumes *cl-op-nuc*: *cl-op x* $\cdot$ *cl-op y* $\leq$ *cl-op* $(x \cdot y)$

begin

subclass *clattice-with-clop*..

end

class *unital-quantale-with-nuc* = *quantale-with-nuc* + *unital-quantale* +
  assumes *one-nuc*: *1* ≤ *cl-op 1*

lemma *nucleus-cl-op*: *nucleus* (*cl-op*::*'a*::*quantale-with-nuc* ⇒ *'a*)
  by (*simp add*: *cl-op-class.clop-iso cl-op-nuc clop-def clop-ext le-funI monoI nucleus-def*)

lemma *unucleus-cl-op*: *unucleus* (*cl-op*::*'a*::*unital-quantale-with-nuc* ⇒ *'a*)
  by (*simp add*: *nucleus-cl-op one-nuc unucleus-def*)

instantiation *cl-op-im* :: (*quantale-with-nuc*) *quantale*
begin

lift-definition *times-cl-op-im* :: *'a*::*quantale-with-nuc cl-op-im* ⇒ *'a cl-op-im* ⇒ *'a cl-op-im* **is** λ*x y*. *cl-op* (*x* · *y*)
  by *simp*

instance
  by (*intro-classes*; *transfer*, *auto simp*: *nuc-comp-assoc nuc-comp-distr nucleus-cl-op nuc-comp-distl*)

end

instantiation *cl-op-im* :: (*unital-quantale-with-nuc*) *unital-quantale*
begin

lift-definition *one-cl-op-im* :: *'a*::*unital-quantale-with-nuc cl-op-im* **is** *cl-op 1*
  by *simp*

instance
  by (*intro-classes*; *transfer*) (*metis clop-closure nsrnqo.mult-onel nuc-prop2 nucleus-cl-op nucleus-def nsrnqo.mult-oner nuc-prop1*)+

end

The usefulness of these theorems remains unclear; it seems difficult to make them collaborate with concrete nuclei.

lemma *nuc-hom*: *Abs-cl-op-im* ∘ *cl-op* ∈ *quantale-homset*
  unfolding *quantale-homset-iff comp-def fun-eq-iff*
  apply *safe*
  apply (*metis* (*no-types*, *lifting*) *Abs-cl-op-im-inverse Rep-cl-op-im-inverse nuc-comp-prop nucleus-cl-op rangeI times-cl-op-im.rep-eq*)
  unfolding *Sup-cl-op-im-def* **by** (*smt Abs-cl-op-im-inverse SUP-cong image-image map-fun-apply nuc-Sup-closed-var nucleus-cl-op rangeI*)

This finishes the first statement of Theorem 3.1.1. The second part follows. It states that for every surjective quantale homomorphism there is a nucleus such that the range of the nucleus is isomorphic to the range of the surjection.

lemma *quant-morph-nuc*:
fixes f :: *'a*::*quantale-with-dual* ⇒ *'b*::*quantale-with-dual*
assumes $f \in$ *quantale-homset*
shows *nucleus* ((*radj f*) ∘ f)
proof−
let $?\varphi$ = (*radj f*) ∘ f
have *adj*: $f \dashv$ (*radj f*)
by (*simp add*: *assms quantale-hom-radj*)
hence *a*: *clop* $?\varphi$
by (*simp add*: *clop-adj*)
{fix x y
have f ($?\varphi$ x · $?\varphi$ y) = (f ∘ $?\varphi$) x · (f ∘ $?\varphi$) y
by (*metis assms comp-eq-dest-lhs quantale-homset-iff*)
also have ... = f x · f y
by (*simp add*: *adj adj-cancel-eq1 fun.map-comp*)
also have ... = f (x · y)
by (*metis assms quantale-homset-iff*)
finally have $?\varphi$ x · $?\varphi$ y ≤ $?\varphi$ (x · y)
by (*metis adj adj-def comp-apply order-refl*)**}**
thus *?thesis*
by (*simp add*: *a nucleus-def*)
qed

lemma *surj-quantale-hom-bij-on*:
fixes f :: *'a*::*quantale-with-dual* ⇒ *'b*::*quantale-with-dual*
assumes *surj f*
and $f \in$ *quantale-homset*
shows *bij-betw f* (*range* (*radj f* ∘ f)) *UNIV*
using *assms quantale-homset-iff surj-Sup-pres-bij-on* **by** *blast*

This establishes the bijection, extending a similar fact about closure operators and complete lattices (surj-Sup-pres-bij). It remains to show that f is a quantale morphism, that is, it preserves Sups and compositions of closed elements with operations defined as in the previous instantiation statement. Sup-preservation holds already for closure operators on complete lattices (surj-Sup-pres-iso). Hence it remains to prove preservation of compositions.

lemma *surj-comp-pres-iso*:
fixes f :: *'a*::*quantale-with-dual* ⇒ *'b*::*quantale-with-dual*
assumes $f \in$ *quantale-homset*
shows f ((*radj f* ∘ f) (x · y)) = f x · f y
proof−
have $f \dashv$ (*radj f*)
by (*simp add*: *assms quantale-hom-radj*)
hence f ((*radj f* ∘ f) (x · y)) = f (x · y)
by (*metis adj-cancel-eq1 comp-eq-dest-lhs*)

thus *?thesis*
using *assms quantale-homset-iff* **by** *auto*
qed

This establishes the quantale isomorphism and finishes the proof of Theorem 3.1.1.

Next I prove Theorem 3.1.2 in Rosenthal's book. nuc-Inf-closed shows that *Fix f* is Inf-closed. Hence the two following lemmas show one direction.

lemma *nuc-bres-pres*: *nucleus f* $\Longrightarrow$ $y \in$ *Fix f* $\Longrightarrow$ $x \rightarrow y \in$ *Fix f*
proof −
assume *a1*: *nucleus f*
assume *a2*: $y \in$ *Fix f*
have *clop f*
using *a1* **by** (*simp add*: *nucleus-def*)
thus *?thesis*
using *a2 a1*
by (*metis clop-Fix-range clop-closure clop-extensive-var dual-order.antisym nucleus-alt-def-cor1*)
qed

lemma *nuc-fres-pres*: *nucleus f* $\Longrightarrow$ $y \in$ *Fix f* $\Longrightarrow$ $y \leftarrow x \in$ *Fix f*
proof −
assume *a1*: *nucleus f*
assume *a2*: $y \in$ *Fix f*
have *clop f*
using *a1* **by** (*simp add*: *nucleus-def*)
thus *?thesis*
using *a2 a1* **by** (*metis antisym-conv clop-Fix-range clop-closure clop-extensive-var nucleus-alt-def-cor2*)
qed

lemma *lax-aux*:
fixes X :: *'a*::*quantale set*
assumes $\forall x. \forall y \in X.\ x \rightarrow y \in X$
and $\forall x.\ \forall y \in X.\ y \leftarrow x \in X$
shows $\bigsqcap\{z \in X.\ x \leq z\} \cdot \bigsqcap\{z \in X.\ y \leq z\} \leq \bigsqcap\{z \in X.\ x \cdot y \leq z\}$
proof−
let $?\varphi = \lambda x.\ \bigsqcap\{w \in X.\ x \leq w\}$
{fix z
assume *a*: $x \cdot y \leq z$
and *b*: $z \in X$
hence *c*: $x \leq z \leftarrow y$
by (*simp add*: *fres-galois*)
hence $z \leftarrow y \in X$
by (*simp add*: *assms*(*2*) *b*)
hence $?\varphi\ x \leq z \leftarrow y$
by (*simp add*: *Inf-lower c*)
hence *d*: $y \leq ?\varphi\ x \rightarrow z$

by (*simp add*: *bres-galois-imp fres-galois*)
hence *?φ x* $\rightarrow z \in X$
by (*simp add*: *assms*(*1*) *b*)
hence *?φ x* · *?φ y* $\leq z$
by (*simp add*: *Inf-lower d bres-galois*)**}**
thus *?thesis*
by (*simp add*: *le-Inf-iff*)
qed

lemma *Inf-closed-res-nuc*:
fixes X :: *'a*::*quantale set*
assumes *Inf-closed-set X*
and $\forall x.\ \forall y \in X.\ x \rightarrow y \in X$
and $\forall x.\ \forall y \in X.\ y \leftarrow x \in X$
shows *nucleus* $(\lambda y.\ \bigsqcap\{x \in X.\ y \leq x\})$
unfolding *nucleus-def* **by** (*simp add*: *Inf-closed-clop assms lax-aux*)

lemma *Inf-closed-res-Fix*:
fixes X :: *'a*::*quantale set*
assumes *Inf-closed-set X*
and $\forall x.\ \forall y \in X.\ x \rightarrow y \in X$
and $\forall x.\ \forall y \in X.\ y \leftarrow x \in X$
shows $X = Fix\ (\lambda y.\ \bigsqcap\{x \in X.\ y \leq x\})$
unfolding *Fix-def*
apply *safe*
apply (*rule antisym*, *simp-all add*: *Inf-lower le-Inf-iff*)
by (*metis* (*no-types*, *lifting*) *Inf-closed-set-def assms*(*1*) *mem-Collect-eq subsetI*)

This finishes the proof of Theorem 3.1.2

6.2 A Representation Theorem

The final proofs for nuclei lead to Rosenthal's representation theorem for quantales (Theorem 3.1.2).

lemma *down-set-lax-morph*: $(\downarrow \circ Sup)\ (X$::*'a*::*quantale set*$) \cdot (\downarrow \circ Sup)\ Y \subseteq (\downarrow \circ Sup)\ (X \cdot Y)$
proof−
have $\bigsqcup((\downarrow \circ Sup)\ X \cdot (\downarrow \circ Sup)\ Y) = \bigsqcup(X \cdot Y)$
by (*smt Sup-downset-adj adj-cancel-eq1 comp-apply comp-dist-mix*)
thus *?thesis*
by (*simp add*: *Sup-downset-adj-var eq-iff*)
qed

lemma *downset-Sup-nuc*: *nucleus* $(\downarrow \circ (Sup$::*'a*::*quantale set* $\Rightarrow$ *'a*$))$
using *Sup-downset-adj clop-adj down-set-lax-morph nucleus-def* **by** *blast*

lemma *downset-surj*: *surj-on* $\downarrow$ (*range* $(\downarrow \circ Sup)$)
using *surj-on-def* **by** *fastforce*

In addition, $\downarrow$ is injective by Lemma downset-inj. Hence it is a bijection between the quantale and the powerset quantale. It remains to show that $\downarrow$ is a quantale morphism.

lemma *downset-Sup-pres-var*: $\downarrow (\bigsqcup X) = (\downarrow \circ Sup)\ (\Downarrow (X::'a::quantale\ set))$
unfolding *comp-def downset-prop downset-set-def*
apply *safe*
apply (*smt Sup-subset-mono dual-order.refl mem-Collect-eq order-subst1 subset-iff*)
by (*smt Sup-mono le-iff-inf le-infI2 mem-Collect-eq*)

lemma *downset-Sup-pres*: $\downarrow (\bigsqcup X) = (\downarrow \circ Sup)\ (\bigcup (\downarrow `\ (X::'a::quantale\ set)))$
by (*metis downset-Sup-pres-var downset-set-prop-var*)

lemma *downset-comp-pres*: $\downarrow ((x::'a::quantale) \cdot y) = (\downarrow \circ Sup)\ (\downarrow x \cdot \downarrow y)$
by (*metis* (*no-types*, *hide-lams*) *Sup-downset-adj-var comp-apply comp-dist-mix downset-iso-iff dual-order.antisym order-refl*)

This finishes the proof of Theorem 3.1.2.

6.3 Conuclei

definition *conucleus* :: $('a::quantale \Rightarrow 'a::quantale) \Rightarrow bool$ **where**
$conucleus\ f = (coclop\ f \wedge (\forall x\ y.\ f\ x \cdot f\ y \leq f\ (x \cdot y)))$

lemma *conuc-lax*: $conucleus\ f \Longrightarrow f\ x \cdot f\ y \leq f\ (x \cdot y)$
by (*simp add*: *conucleus-def*)

definition *uconucleus* :: $('a::unital\text{-}quantale \Rightarrow 'a::unital\text{-}quantale) \Rightarrow bool$ **where**

$uconucleus\ f = (conucleus\ f \wedge f\ 1 \leq 1)$

Next I prove Theorem 3.1.3.

lemma *conuc-Sup-closed*: $conucleus\ f \Longrightarrow f \circ Sup \circ (`)\ f = Sup \circ (`)\ f$
unfolding *conucleus-def fun-eq-iff comp-def*
by (*smt coclop-coextensive-var coclop-idem coclop-iso image-comp image-image le-iff-sup mono-SUP sup.orderE*)

lemma *conuc-comp-closed*: $conucleus\ f \Longrightarrow f\ (f\ x \cdot f\ y) = f\ x \cdot f\ y$
unfolding *conucleus-def* **by** (*metis antisym-conv coclop-coextensive-var coclop-idem-var*)

The sets of fixpoints of conuclei are closed under Sups and composition; hence they form subquantales.

lemma *conuc-Sup-qclosed*: $conucleus\ f \Longrightarrow Sup\text{-}closed\text{-}set\ (Fix\ f) \wedge comp\text{-}closed\text{-}set\ (Fix\ f)$
apply *safe*
apply (*simp add*: *coclop-Sup-closed conucleus-def*)
unfolding *conucleus-def comp-closed-set-def* **by** (*metis coclop-coclosure coclop-coclosure-set coclop-coextensive-var dual-order.antisym*)

This shows that the subset $Fix\,f$ of a quantale, for conucleus f, is closed under Sups and composition. It is therefore a subquantale. $f : f[Q] \to Q$ is an embedding. As before, this could be shown by formalising a type class of quantales with a conucleus operation, converting the range of the conucleus into a type and providing a sublocale proof. First, this would require showing that the coclosed elements of a complete lattice form a complete sublattice. Relative to the proofs for closure operators and nuclei there is nothing to be learned. I provide this proof in the section on left-sided elements, where the conucleus can be expressed within the language of quantales.

The second part of Theorem 3.1.3 states that every subquantale of a given quantale is equal to $Fix\,f$ for some conucleus f.

lemma *lax-aux2*:
fixes $X :: 'a::quantale\ set$
assumes *Sup-closed-set X*
and *comp-closed-set X*
shows $\bigsqcup\{z \in X.\ z \leq x\} \cdot \bigsqcup\{z \in X.\ z \leq y\} \leq \bigsqcup\{z \in X.\ z \leq x \cdot y\}$
proof−
let $?\varphi = \lambda x.\ \bigsqcup\{z \in X.\ z \leq x\}$
have $?\varphi\ x \cdot ?\varphi\ y = \bigsqcup\{\bigsqcup\{v \cdot w\ |v.\ v \in X \wedge v \leq x\}\ |w.\ w \in X \wedge w \leq y\}$
by (*simp add*: *Sup-distr Sup-distl setcompr-eq-image*)
also have $... = \bigsqcup\{v \cdot w\ |v\ w.\ v \in X \wedge v \leq x \wedge w \in X \wedge w \leq y\}$
apply (*rule antisym*)
apply (*rule Sup-least*, *clarsimp*, *smt Collect-mono-iff Sup-subset-mono*)
by (*rule Sup-least*, *clarsimp*, *smt Sup-upper2 eq-iff mem-Collect-eq*)
also have $... \leq ?\varphi\ (x \cdot y)$
by (*smt Collect-mono-iff Sup-subset-mono assms(2) comp-closed-set-def psrpq.mult-isol-var*)
finally show *?thesis*
by *force*
qed

lemma *subquantale-conucleus*:
fixes $X :: 'a::quantale\ set$
assumes *Sup-closed-set X*
and *comp-closed-set X*
shows *conucleus* $(\lambda x.\ \bigsqcup\{y \in X.\ y \leq x\})$
unfolding *conucleus-def* **by** (*safe*, *simp-all add*: *Sup-closed-coclop assms(1) assms(2) lax-aux2*)

lemma *subquantale-Fix*:
fixes $X :: 'a::quantale\ set$
assumes *Sup-closed-set X*
and *comp-closed-set X*
shows $X = Fix\ (\lambda x.\ \bigsqcup\{y \in X.\ y \leq x\})$
unfolding *Fix-def*
apply *safe*
apply (*metis* (*mono-tags*, *lifting*) *Sup-least Sup-upper antisym mem-Collect-eq order-refl*)

by (*metis* (*no-types*, *lifting*) *Sup-closed-set-def assms*(*1*) *mem-Collect-eq subsetI*)

This finishes the proof of Theorem 3.1.3

end

7 Left-Sided Elements in Quantales

theory *Quantale-Left-Sided*
imports *Quantic-Nuclei-Conuclei*
begin

context *quantale*
begin

7.1 Basic Definitions

Left-sided elements are well investigated in quantale theory. They could be defined in weaker settings, for instance, ord with a top element.

definition $lsd\ x = (\top \cdot x \leq x)$

definition $rsd\ x = (x \cdot \top \leq x)$

definition $twosd\ x = (lsd\ x \wedge rsd\ x)$

definition $slsd\ x = (\top \cdot x = x)$

definition $srsd\ x = (x \cdot \top = x)$

definition $stwsd\ x = (slsd\ x \wedge srsd\ x)$

lemma *lsided-bres*: $(lsd\ x) = (x \leq \top \rightarrow x)$
by (*simp add*: *bres-galois lsd-def*)

lemma *lsided-fres*: $(lsd\ x) = (\top \leq x \leftarrow x)$
by (*simp add*: *fres-galois lsd-def*)

lemma *lsided-fres-eq*: $(lsd\ x) = (x \leftarrow x = \top)$
using *fres-galois top-le lsd-def* **by** *force*

lemma *lsided-slsided*: $lsd\ x \Longrightarrow x \cdot x = x \Longrightarrow slsd\ x$
using *fres-sol lsided-fres-eq slsd-def* **by** *force*

lemma *lsided-prop*: $lsd\ x \Longrightarrow y \cdot x \leq x$
by (*simp add*: *fres-galois lsided-fres-eq*)

lemma *rsided-fres*: $(rsd\ x) = (x \leq x \leftarrow \top)$
by (*simp add*: *fres-galois rsd-def*)

lemma *rsided-bres*: $(rsd\ x) = (\top \leq x \rightarrow x)$
by (*simp add*: *bres-galois rsd-def*)

lemma *rsided-bres-eq*: $(rsd\ x) = (x \rightarrow x = \top)$
using *top-le rsided-bres* **by** *blast*

lemma *rsided-srsided*: $rsd\ x \Longrightarrow x \cdot x = x \Longrightarrow srsd\ x$
using *bres-sol rsided-bres-eq srsd-def* **by** *auto*

lemma *rsided-prop*: $rsd\ x \Longrightarrow x \cdot y \leq x$
by (*simp add*: *bres-galois rsided-bres-eq*)

lemma *lsided-top*: $lsd\ \top$
by (*simp add*: *lsd-def*)

lemma *lsided-bot*: $lsd\ \bot$
by (*simp add*: *lsd-def*)

lemma *rsided-top*: $rsd\ \top$
by (*simp add*: *rsd-def*)

lemma *rsided-bot*: $rsd\ \bot$
by (*simp add*: *rsd-def*)

end

Right-sided elements are henceforth not considered. Their properties arise by opposition duality, which is not formalised.

The following functions have left-sided elements as fixpoints.

definition *lsl*:: $'a$::*quantale* $\Rightarrow$ $'a$ (ν) **where**
$\nu\ x = \top \cdot x$

definition *lsu* :: $'a$::*quantale* $\Rightarrow$ $'a$ $(\nu^\natural)$ **where**
$\nu^\natural\ x = \top \rightarrow x$

These functions are adjoints.

lemma *ls-galois*: $\nu \dashv \nu^\natural$
by (*simp add*: *adj-def bres-galois lsl-def lsu-def*)

Due to this, the following properties hold.

lemma *lsl-iso*: *mono* ν
using *adj-iso1 ls-galois* **by** *blast*

lemma *lsl-iso-var*: $x \leq y \Longrightarrow \nu\ x \leq \nu\ y$
by (*simp add*: *lsl-iso monoD*)

lemma *lsu-iso*: *mono* $\nu^\natural$
using *adj-iso2 ls-galois* **by** *blast*

lemma *lsu-iso-var*: $x \leq y \Longrightarrow \nu^\natural\ x \leq \nu^\natural\ y$
by (*simp add*: *lsu-iso monoD*)

lemma *lsl-bot* [*simp*]: $\nu\ \bot = \bot$
by (*simp add*: *lsl-def*)

lemma *lsu-top* [*simp*]: $\nu^\natural\ \top = \top$
by (*simp add*: *lsu-def bres-galois-imp top-le*)

lemma *lsu-Inf-pres*: *Inf-pres* $\nu^\natural$
unfolding *fun-eq-iff* **by** (*metis ls-galois radj-Inf-pres*)

lemma *lsl-Sup-pres*: *Sup-pres* $(\nu::'a::quantale \Rightarrow 'a)$
by (*simp add*: *fun-eq-iff*, *metis SUP-cong Sup-distl lsl-def*)

lemma *lsu-Inf-closed*: *Inf-closed-set* (*range* $\nu^\natural$)
by (*simp add*: *Inf-pres-Inf-closed lsu-Inf-pres*)

lemma *lsl-Sup-closed*: *Sup-closed-set* (*range* $(\nu::'a::quantale \Rightarrow 'a)$)
by (*simp add*: *Sup-pres-Sup-closed lsl-Sup-pres*)

lemma *lsl-lsu-canc1*: $\nu \circ \nu^\natural \leq id$
by (*simp add*: *adj-cancel1 ls-galois*)

lemma *lsl-lsu-canc2*: $id \leq \nu^\natural \circ \nu$
by (*simp add*: *adj-cancel2 ls-galois*)

lemma *clop-lsu-lsl*: *clop* $(\nu^\natural \circ \nu)$
by (*simp add*: *clop-adj ls-galois*)

lemma *coclop-lsl-lsu*: *coclop* $(\nu \circ \nu^\natural)$
by (*simp add*: *coclop-adj ls-galois*)

lemma *dang1*: $\nu\ (\nu\ x \sqcap y) \leq \nu\ x$
unfolding *lsl-def* **by** (*metis bres-galois bres-interchange bres-top-top inf.coboundedI1*)

lemma *lsl-trans*: $\nu \circ \nu \leq \nu$
unfolding *lsl-def fun-eq-iff comp-def*
by (*metis* (*no-types*, *lifting*) *bres-galois bres-interchange bres-top-top le-funI*)

lemma *lsl-lsu-prop*: $\nu \circ \nu^\natural \leq \nu^\natural$
unfolding *le-fun-def comp-def*
by (*metis adj-def dang1 dual-order.trans le-iff-inf ls-galois order-refl top-greatest*)

lemma *lsu-lsl-prop*: $\nu \leq \nu^\natural \circ \nu$
unfolding *le-fun-def comp-def* **by** (*metis adj-def comp-def le-fun-def ls-galois lsl-trans*)

context *unital-quantale*
begin

Left-sidedness and strict left-sidedness now coincide.

lemma *lsided-eq*: *lsd* = *slsd*
unfolding *fun-eq-iff* **by** (*simp add*: *eq-iff le-top lsd-def slsd-def*)

lemma *lsided-eq-var1*: $(x \leq \top \rightarrow x) = (x = \top \rightarrow x)$
using *bres-galois dual-order.trans eq-iff le-top* **by** *blast*

lemma *lsided-eq-var2*: *lsd x* = $(x = \top \rightarrow x)$
using *bres-galois lsided-eq lsided-eq-var1 lsd-def* **by** *simp*

end

lemma *lsided-def3*: $(\nu\ (x{::}'a{::}\textit{unital-quantale}) = x) = (\nu^\natural\ x = x)$
unfolding *lsl-def lsu-def* **by** (*metis lsided-eq lsided-eq-var2 slsd-def*)

lemma *Fix-lsl-lsu*: *Fix* $(\nu{::}'a{::}\textit{unital-quantale} \Rightarrow 'a)$ = *Fix* $\nu^\natural$
unfolding *Fix-def* **by** (*simp add*: *lsided-def3*)

lemma *Fix-lsl-left-slsided*: *Fix* $\nu = \{(x{::}'a{::}\textit{unital-quantale}).\ \textit{lsd}\ x\}$
unfolding *Fix-def lsl-def lsd-def* **using** *eq-iff le-top* **by** *blast*

lemma *Fix-lsl-iff* [*simp*]: $(x \in \textit{Fix}\ \nu) = (\nu\ x = x)$
by (*simp add*: *Fix-def*)

lemma *Fix-lsu-iff* [*simp*]: $(x \in \textit{Fix}\ \nu^\natural) = (\nu^\natural\ x = x)$
by (*simp add*: *Fix-def*)

lemma *lsl-lsu-prop-eq* [*simp*]: $(\nu{::}'a{::}\textit{unital-quantale} \Rightarrow 'a) \circ \nu^\natural = \nu^\natural$
by (*smt antisym clop-extensive-var clop-lsu-lsl comp-apply le-fun-def lsided-eq-var1 lsl-lsu-prop lsu-def lsu-lsl-prop*)

lemma *lsu-lsl-prop-eq* [*simp*]: $\nu^\natural \circ \nu = (\nu{::}'a{::}\textit{unital-quantale} \Rightarrow 'a)$
by (*metis adj-cancel-eq1 ls-galois lsl-lsu-prop-eq*)

7.2 Connection with Closure and Coclosure Operators, Nuclei and Conuclei

lsl is therefore a closure operator, lsu a cocolosure operator.

lemma *lsl-clop*: *clop* $(\nu{::}'a{::}\textit{unital-quantale} \Rightarrow 'a)$
by (*metis clop-lsu-lsl lsu-lsl-prop-eq*)

lemma *lsu-coclop*: *coclop* $(\nu^\natural{::}'a{::}\textit{unital-quantale} \Rightarrow 'a)$
by (*metis coclop-lsl-lsu lsl-lsu-prop-eq*)

lemma *lsl-range-fix*: *range* $(\nu{::}'a{::}\textit{unital-quantale} \Rightarrow 'a)$ = *Fix* ν

by (*simp add*: *clop-closure-set lsl-clop*)

lemma *lsu-range-fix*: *range* ($\nu^\natural$::'*a*::*unital-quantale* ⇒ '*a*) = *Fix* $\nu^\natural$
by (*simp add*: *coclop-coclosure-set lsu-coclop*)

lemma *range-lsl-iff* [*simp*]: ((*x*::'*a*::*unital-quantale*) ∈ *range* ν) = (ν *x* = *x*)
by (*simp add*: *lsl-range-fix*)

lemma *range-lsu-iff* [*simp*]: ((*x*::'*a*::*unital-quantale*) ∈ *range* $\nu^\natural$) = ($\nu^\natural$ *x* = *x*)
by (*simp add*: *lsu-range-fix*)

lsl and lsu are therefore both Sup and Inf closed.

lemma *lsu-Sup-closed*: *Sup-closed-set* (*Fix* ($\nu^\natural$::'*a*::*unital-quantale* ⇒ '*a*))
by (*metis Fix-lsl-lsu lsl-Sup-closed lsl-range-fix*)

lemma *lsl-Inf-closed*: *Inf-closed-set* (*Fix* (ν::'*a*::*unital-quantale* ⇒ '*a*))
by (*metis Fix-lsl-lsu lsu-Inf-closed lsu-range-fix*)

lsl is even a quantic conucleus.

lemma *lsu-lax*: $\nu^\natural$ (*x*::'*a*::*unital-quantale*) · $\nu^\natural$ *y* ≤ $\nu^\natural$ (*x* · *y*)
unfolding *lsu-def* **by** (*meson bres-canc1 bres-galois bres-interchange le-top order-trans*)

lemma *lsu-one*: $\nu^\natural$ *1* ≤ (*1*::'*a*::*unital-quantale*)
unfolding *lsu-def* **using** *bres-canc1 dual-order.trans le-top* **by** *blast*

lemma *lsl-one*: *1* ≤ ν (*1*::'*a*::*unital-quantale*)
unfolding *lsl-def* **by** *simp*

lemma *lsu-conuc*: *conucleus* ($\nu^\natural$::'*a*::*unital-quantale* ⇒ '*a*)
by (*simp add*: *lsu-coclop conucleus-def lsu-lax*)

It is therefore closed under composition.

lemma *lsu-comp-closed-var* [*simp*]: $\nu^\natural$ ($\nu^\natural$ *x* · $\nu^\natural$ *y*) = $\nu^\natural$ *x* · $\nu^\natural$ (*y*::'*a*::*unital-quantale*)
by (*simp add*: *conuc-comp-closed lsu-conuc*)

lemma *lsu-comp-closed*: *comp-closed-set* (*Fix* ($\nu^\natural$::'*a*::*unital-quantale* ⇒ '*a*))
by (*simp add*: *conuc-Sup-qclosed lsu-conuc*)

lsl is not a quantic nucleus unless composition is commutative.

lemma ν *x* · ν *y* ≤ ν (*x* · (*y*::'*a*::*unital-quantale*))
oops

Yet it is closed under composition (because the set of fixpoints are the same).

lemma *lsl-comp-closed*: *comp-closed-set* (*Fix* (ν::'*a*::*unital-quantale* ⇒ '*a*))
by (*simp add*: *Fix-lsl-lsu lsu-comp-closed*)

lemma *lsl-comp-closed-var* [*simp*]: ν (ν *x* · ν (*y*::'*a*::*unital-quantale*)) = ν *x* · ν *y*
by (*metis Fix-lsl-iff lsl-def lsl-range-fix mult.assoc rangeI*)

The following simple properties go beyond nuclei and conuclei.

lemma *lsl-lsu-ran*: *range* $\nu^\natural$ = *range* (ν::'*a*::*unital-quantale* ⇒ '*a*)
by (*simp add*: *Fix-lsl-lsu lsl-range-fix lsu-range-fix*)

lemma *lsl-top* [*simp*]: $\nu\ \top$ = ($\top$::'*a*::*unital-quantale*)
by (*simp add*: *clop-top lsl-clop*)

lemma *lsu-bot* [*simp*]: $\nu^\natural\ \bot$ = ($\bot$::'*a*::*unital-quantale*)
using *lsided-def3 lsl-bot* **by** *blast*

lemma *lsu-inj-on*: *inj-on* $\nu^\natural$ (*Fix* ($\nu^\natural$::'*a*::*unital-quantale* ⇒ '*a*))
by (*metis coclop-coclosure inj-onI lsu-coclop lsu-range-fix*)

lemma *lsl-inj-on*: *inj-on* ν (*Fix* (ν::'*a*::*unital-quantale* ⇒ '*a*))
by (*metis clop-closure inj-onI lsl-clop lsl-range-fix*)

Additional preservation properties of lsl and lsu are useful in proofs.

lemma *lsl-Inf-closed-var* [*simp*]: ν ($\bigsqcap x \in X.\ \nu\ x$) = ($\bigsqcap x \in X.\ \nu$ (*x*::'*a*::*unital-quantale*))
by (*metis* (*no-types*, *hide-lams*) *INF-image lsided-def3 lsu-Inf-pres lsu-lsl-prop-eq o-apply*)

lemma *lsl-Sup-closed-var* [*simp*]: ν ($\bigsqcup x \in X.\ \nu\ x$) = ($\bigsqcup x \in X.\ \nu$ (*x*::'*a*::*unital-quantale*))
unfolding *lsl-def* **by** (*metis Sup-distl mult.assoc top-times-top*)

lemma *lsu-Inf-closed-var* [*simp*]: $\nu^\natural$ ($\bigsqcap x \in X.\ \nu^\natural\ x$) = ($\bigsqcap x \in X.\ \nu^\natural$ (*x*::'*a*::*unital-quantale*))
by (*metis INF-image lsided-def3 lsl-Inf-closed-var lsl-lsu-prop-eq*)

lemma *lsu-Sup-closed-var* [*simp*]: $\nu^\natural$ ($\bigsqcup x \in X.\ \nu^\natural\ x$) = ($\bigsqcup x \in X.\ \nu^\natural$ (*x*::'*a*::*unital-quantale*))
by (*metis SUP-image lsided-def3 lsl-Sup-closed-var lsl-lsu-prop-eq*)

lemma *lsl-inf-closed-var* [*simp*]: ν ($\nu\ x \sqcap \nu\ y$) = ν (*x*::'*a*::*unital-quantale*) $\sqcap\ \nu\ y$
by (*smt antisym clop-extensive-var dang1 inf-sup-aci*(*1*) *le-inf-iff lsl-clop*)

lemma *lsl-sup-closed-var* [*simp*]: ν ($\nu\ x \sqcup \nu\ y$) = ν (*x*::'*a*::*unital-quantale*) $\sqcup\ \nu\ y$
by (*meson Sup-sup-closed lsl-Sup-closed range-eqI range-lsl-iff sup-closed-set-def*)

lemma *lsu-inf-closed-var* [*simp*]: $\nu^\natural$ ($\nu^\natural\ x \sqcap \nu^\natural\ y$) = $\nu^\natural$ (*x*::'*a*::*unital-quantale*) $\sqcap$ $\nu^\natural\ y$
by (*metis* (*no-types*, *lifting*) *lsided-def3 lsl-inf-closed-var lsl-lsu-prop-eq o-apply*)

lemma *lsu-sup-closed-var* [*simp*]: $\nu^\natural$ ($\nu^\natural\ x \sqcup \nu^\natural\ y$) = $\nu^\natural$ (*x*::'*a*::*unital-quantale*) $\sqcup$ $\nu^\natural\ y$
by (*metis* (*no-types*, *lifting*) *lsided-def3 lsl-lsu-prop-eq lsl-sup-closed-var o-def*)

lemma *lsu-fres-closed* [*simp*]: $\nu^\natural$ ($\nu^\natural\ x \leftarrow \nu^\natural\ y$) = $\nu^\natural\ x \leftarrow \nu^\natural$ (*y*::'*a*::*unital-quantale*)
unfolding *lsu-def* **by** (*simp add*: *bres-curry fres-bres*)

lemma *lsl-fres-closed* [*simp*]: ν ($\nu\ x \leftarrow \nu\ y$) = $\nu\ x \leftarrow \nu$ (*y*::'*a*::*unital-quantale*)
unfolding *lsl-def* **by** (*metis fres-interchange lsd-def lsided-eq mult.assoc slsd-def*

top-times-top)

lemma *lsl-almost-lax*: $x \cdot \nu\ y \leq \nu\ (y::'a::unital\text{-}quantale)$
by (*metis inf-top.right-neutral lsided-eq lsided-prop lsl-def lsl-inf-closed-var mult.right-neutral slsd-def*)

Finally, here are two counterexamples.

lemma $\nu^\natural\ (\nu^\natural\ x \rightarrow \nu^\natural\ y) = \nu^\natural\ x \rightarrow \nu^\natural\ (y::'a::unital\text{-}quantale)$
oops

lemma $\nu\ (\nu\ x \rightarrow \nu\ y) = \nu\ x \rightarrow \nu\ (y::'a::unital\text{-}quantale)$
oops

context *ab-quantale*
begin

lemma *lsided-times-top*: *lsd* $(\top \cdot x)$
by (*metis lsd-def mult-isor top-greatest mult-assoc*)

lemma *lsided-le2*: *lsd* $x \Longrightarrow x \cdot y \leq x \sqcap (y \cdot \top)$
using *lsided-prop psrpq.mult-isol mult-commute* **by** *auto*

lemma *lsided-le3*: *lsd* $x \Longrightarrow (x \sqcap y) \cdot z \leq x$
by (*metis inf-le1 lsided-prop mult-isol order-trans mult-commute*)

lemma *lsided-le4*: *lsd* $x \Longrightarrow (x \sqcap y) \cdot z \leq x \sqcap (y \cdot z)$
by (*simp add*: *mult-isor lsided-le3*)

lemma *lsided-times-le-inf*: *lsd* $x \Longrightarrow$ *lsd* $y \Longrightarrow x \cdot y \leq x \sqcap y$
using *lsided-prop lsided-le2* **by** *force*

end

Now lsl is a quantic nucleus.

lemma *lsl-lax*: $\nu\ x \cdot \nu\ y \leq \nu\ (x \cdot (y::'aa::ab\text{-}unital\text{-}quantale))$
by (*metis* (*no-types*, *hide-lams*) *lsl-almost-lax lsl-def mult.commute mult.left-commute*)

lemma *lsl-nuc*: *nucleus* $(\nu::'a::ab\text{-}unital\text{-}quantale \Rightarrow 'a)$
by (*simp add*: *lsl-clop nucleus-def lsl-lax*)

The following properties go beyond nuclei.

lemma *lsl-comp-pres*: $\nu\ (x \cdot y) = \nu\ x \cdot \nu\ (y::'a::ab\text{-}unital\text{-}quantale)$
by (*metis* (*no-types*, *lifting*) *lsl-comp-closed-var lsl-nuc nuc-prop1 nuc-prop2*)

lemma *lsl-bres-closed* [*simp*]: $\nu\ (\nu\ x \rightarrow \nu\ y) = \nu\ x \rightarrow \nu\ (y::'a::ab\text{-}unital\text{-}quantale)$
by (*simp add*: *lsl-nuc nuc-bres-closed*)

lemma *lsu-bres-pres* [*simp*]: $\nu^\natural\ (\nu^\natural\ x \rightarrow \nu^\natural\ y) = \nu^\natural\ x \rightarrow \nu^\natural\ (y::'a::ab\text{-}unital\text{-}quantale)$
by (*simp add*: *bres-fres-eq*)

lemma *lsl-prelocalic-var*: $\nu\ x \cdot y \leq \nu\ x \sqcap \nu\ (y::'a::ab\text{-}unital\text{-}quantale)$
by (*metis inf-top.right-neutral lsided-le4 lsided-times-top lsl-def*)

lemma *dang3*: $(\nu\ x \sqcap y) \cdot z \leq \nu\ x \sqcap (y \cdot (z::'a::ab\text{-}unital\text{-}quantale))$
by (*metis lsided-le4 lsided-times-top lsl-def*)

lemma *lsl-prelocalic*: $\nu\ x \cdot \nu\ y \leq \nu\ x \sqcap \nu\ (y::'a::ab\text{-}unital\text{-}quantale)$
by (*metis lsided-times-le-inf lsided-times-top lsl-def*)

lemma *lsl-local*: $x \cdot \nu\ y \leq \nu\ (x \cdot (y::'a::ab\text{-}unital\text{-}quantale))$
by (*simp add*: *lsl-def mult.left-commute*)

lemma *lsl-local-eq*: $x \cdot \nu\ y = \nu\ (x \cdot (y::'a::ab\text{-}unital\text{-}quantale))$
by (*simp add*: *lsl-def mult.left-commute*)

Relative pseudocomplementation can be defined on the subquantale induced by lsu.

definition *rpc x y* = $\nu^\natural\ (-x \sqcup (y::'a::bool\text{-}unital\text{-}quantale))$

lemma *lsl-rpc* [*simp*]: $\nu\ (rpc\ x\ y) = rpc\ x\ y$
by (*metis lsl-lsu-prop-eq o-apply rpc-def*)

lemma *lsu-rpc* [*simp*]: $\nu^\natural\ (rpc\ x\ y) = rpc\ x\ y$
using *lsided-def3 lsl-rpc* **by** *blast*

lemma *lsl-rpc-galois*: $(x \sqcap \nu\ z \leq y) = (z \leq rpc\ x\ (y::'a::bool\text{-}unital\text{-}quantale))$
unfolding *rpc-def* **by** (*metis adj-def inf-commute ls-galois shunt1*)

lemma *lsl-rpc-galois-var*: $x \sqcap \nu\ z \leq y \longleftrightarrow \nu\ z \leq rpc\ x\ y$
by (*metis adj-def ls-galois lsl-rpc-galois lsu-rpc*)

lemma *rpc-expl-aux*: $\bigsqcup\{z.\ x \sqcap \nu\ z \leq y\} = \bigsqcup\{z.\ x \sqcap z \leq y \wedge \nu\ z = (z::'a::bool\text{-}unital\text{-}quantale)\}$
proof (*rule antisym*)
show $\bigsqcup\{z.\ x \sqcap \nu\ z \leq y\} \leq \bigsqcup\{z.\ x \sqcap z \leq y \wedge \nu\ z = z\}$
apply (*rule Sup-mono*, *safe*)
by (*smt lsided-eq lsided-prop lsl-def lsl-lsu-prop-eq lsl-rpc-galois mem-Collect-eq o-apply rpc-def slsd-def*)
show $\bigsqcup\{z.\ x \sqcap z \leq y \wedge \nu\ z = z\} \leq \bigsqcup\{z.\ x \sqcap \nu\ z \leq y\}$
by (*simp add*: *Collect-mono-iff Sup-subset-mono*)
qed

lemma *rpc-expl*: $rpc\ x\ y = \bigsqcup\{z.\ x \sqcap z \leq y \wedge \nu\ z = (z::'a::bool\text{-}unital\text{-}quantale)\}$
proof−
have $rpc\ x\ y = \bigsqcup\{z.\ z \leq rpc\ x\ y\}$
using *Sup-atMost atMost-def* **by** *metis*
also have ... = $\bigsqcup\{z.\ x \sqcap \nu\ z \leq y\}$
using *lsl-rpc-galois* **by** *metis*
finally show *?thesis*

by (*simp add*: *rpc-expl-aux*)
qed

7.3 Non-Preservation and Lack of Closure

context *bool-ab-unital-quantale*
begin

A few counterexamples deal in particular with lack of closure with respect to boolean complementation.

lemma $\nu^\natural\ (x \cdot y) = \nu^\natural\ x \cdot \nu^\natural\ y$
oops

lemma $\nu\ 1 = 1$
oops

lemma $\nu^\natural\ x = \nu\ x$
oops

lemma $\nu^\natural\ (\bigsqcup\ P) = \bigsqcup \{\nu^\natural\ p\ |p.\ p \in P\}$
oops

lemma *rpc* $(\nu^\natural\ x)\ (\nu^\natural\ y) = -(\nu^\natural\ x) \sqcup (\nu^\natural\ y)$
oops

lemma *rpc* $(\nu\ x)\ (\nu\ y) = -(\nu\ x) \sqcup (\nu\ y)$
oops

lemma $\nu\ (-(\nu^\natural\ x) \sqcup (\nu^\natural\ y)) = -(\nu^\natural\ x) \sqcup (\nu^\natural\ y)$
oops

lemma $\nu\ (-(\nu\ x) \sqcup (\nu\ y)) = -(\nu\ x) \sqcup (\nu\ y)$
oops

lemma $\nu\ x \cdot \nu\ y = \nu\ x \sqcap \nu\ y$
oops

lemma $\nu\ (-\ (\nu\ x)) = -\ (\nu\ x)$
oops

lemma $\nu^\natural\ (-\ (\nu^\natural\ x)) = -\ (\nu^\natural\ x)$
oops

lemma $\nu\ (-\ (\nu\ x) \sqcup (\nu\ y)) = -\ (\nu\ x) \sqcup (\nu\ y)$
oops

lemma $\nu^\natural\ (-\ (\nu^\natural\ x) \sqcup (\nu^\natural\ y)) = -\ (\nu^\natural\ x) \sqcup (\nu^\natural\ y)$
oops

end

7.4 Properties of Quotient Algebras and Subalgebras

Finally I replay Rosenthal's quotient and subalgebra proofs for nuclei and conuclei in the concrete setting of left-sided elements. Ideally it should be sufficient to show that lsl is a (quantale with) nucleus and then pick up the quotient algebra proof from the nucleus section. But there is no way of achieving this, apart from creating a type class for quantales with the lsl operation. This seems bizarre, since lsl is just a definition. On the other hand, an approach in which interpretations are used instead of instantiations might do the job.

The first proof shows that lsu, as a conucleus, yields a subalgebra.

typedef (**overloaded**) *'a lsu-set* = *Fix* ($\nu^{\natural}$::*'a*::*unital-quantale* ⇒ *'a*)
using *Fix-lsu-iff lsu-bot* **by** *blast*

setup-lifting *type-definition-lsu-set*

instantiation *lsu-set* :: (*unital-quantale*) *quantale*
begin

lift-definition *times-lsu-set* :: *'a lsu-set* ⇒ *'a lsu-set* ⇒ *'a lsu-set* **is** *times*
using *comp-closed-set-def lsu-comp-closed* **by** *blast*

lift-definition *Inf-lsu-set* :: *'a lsu-set set* ⇒ *'a lsu-set* **is** *Inf*
by (*metis Fix-lsl-lsu Inf-closed-set-def lsl-Inf-closed subset-eq*)

lift-definition *Sup-lsu-set* :: *'a lsu-set set* ⇒ *'a lsu-set* **is** *Sup*
using *Sup-closed-set-def lsu-Sup-closed subsetI* **by** *blast*

lift-definition *bot-lsu-set* :: *'a lsu-set* **is** ⊥
by *simp*

lift-definition *sup-lsu-set* :: *'a lsu-set* ⇒ *'a lsu-set* ⇒ *'a lsu-set* **is** *sup*
by (*metis Fix-lsu-iff lsu-sup-closed-var*)

lift-definition *top-lsu-set* :: *'a lsu-set* **is** ⊤
by *simp*

lift-definition *inf-lsu-set* :: *'a lsu-set* ⇒ *'a lsu-set* ⇒ *'a lsu-set* **is** *inf*
by (*metis Fix-lsu-iff lsu-inf-closed-var*)

lift-definition *less-eq-lsu-set* :: *'a lsu-set* ⇒ *'a lsu-set* ⇒ *bool* **is** *less-eq* **.**

lift-definition *less-lsu-set* :: *'a lsu-set* ⇒ *'a lsu-set* ⇒ *bool* **is** *less* **.**

instance

by (*intro-classes*; *transfer*, *simp-all add*: *mult.assoc Inf-lower Inf-greatest Sup-upper Sup-least less-le-not-le Sup-distr Sup-distl*)

end

This proof checks simply closure conditions, as one would expect for establishing a subalgebra.

instance *lsu-set* :: (*bool-ab-unital-quantale*) *distrib-ab-quantale*
apply (*intro-classes*; *transfer*)
apply (*simp add*: *mult.commute*)
using *sup-inf-distrib1* **by** *blast*

typedef (**overloaded**) *'a lsl-set* = *range* (*ν*::*'a*:: *unital-quantale* ⇒ *'a*)
by *blast*

setup-lifting *type-definition-lsl-set*

The final statement shows that lsu, as a nucleus, yields a quotient algebra.

instantiation *lsl-set* :: (*ab-unital-quantale*) *ab-unital-quantale*
begin

lift-definition *one-lsl-set* :: *'a*::*ab-unital-quantale lsl-set* **is** *ν 1*
by *blast*

lift-definition *Inf-lsl-set* :: *'a lsl-set set* ⇒ *'a lsl-set* **is** *λX*. *ν* (⨅ *X*)
by *blast*

lift-definition *Sup-lsl-set* :: *'a lsl-set set* ⇒ *'a lsl-set* **is** *λX*. *ν* (⨆ *X*)
by *blast*

lift-definition *bot-lsl-set* :: *'a lsl-set* **is** *ν* ⊥
by *blast*

lift-definition *sup-lsl-set* :: *'a lsl-set* ⇒ *'a lsl-set* ⇒ *'a lsl-set* **is** *λx y*. *ν x* ⊔ *ν y*
by *auto* (*metis lsl-sup-closed-var*)

lift-definition *top-lsl-set* :: *'a lsl-set* **is** *ν* ⊤
by *blast*

lift-definition *inf-lsl-set* :: *'a lsl-set* ⇒ *'a lsl-set* ⇒ *'a lsl-set* **is** *λx y*. *ν x* ⊓ *ν y*
by (*meson lsl-inf-closed-var range-lsl-iff*)

lift-definition *less-eq-lsl-set* :: *'a lsl-set* ⇒ *'a lsl-set* ⇒ *bool* **is** *λx y* . *ν x* ≤ *ν y* **.**

lift-definition *less-lsl-set* :: *'a lsl-set* ⇒ *'a lsl-set* ⇒ *bool* **is** *λx y*. *ν x* ≤ *ν y* ∧ *x* ≠ *y* **.**

lift-definition *times-lsl-set* :: *'a lsl-set* ⇒ *'a lsl-set* ⇒ *'a lsl-set* **is** *λ x y*. *ν* (*x* · *y*)

by *blast*

instance
apply (*standard*; *transfer*)
apply (*simp add*: *lsl-local-eq mult.commute mult.left-commute*)
apply (*simp add*: *mult.commute*)
apply *auto*[*1*]
apply *fastforce*+
apply *auto*[*1*]
apply *fastforce*+
apply (*metis Inf-lower lsl-iso-var range-lsl-iff*)
apply (*metis Inf-greatest lsl-iso-var range-lsl-iff*)
apply (*metis Sup-upper lsl-iso-var range-lsl-iff*)
apply (*metis Sup-least lsl-iso-var range-lsl-iff*)
apply *fastforce*+
apply (*smt Sup-distl image-cong lsl-local-eq mult.commute*)
apply (*smt Sup-distl image-cong lsl-local-eq mult.commute*)
apply (*simp add*: *lsl-def cong del*: *image-cong-simp*)
apply (*simp add*: *lsl-local-eq*)
done

end

This proof checks preservation properties instead of closure ones.

end

References

[1] S. Abramsky and S. Vickers. Quantales, observational logic and process semantics. *Mathematical Structures in Computer Science*, 3(2):161–227, 1993.

[2] A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013.

[3] B. Dongol, V. B. F. Gomes, I. J. Hayes, and G. Struth. Partial semigroups and convolution algebras. *Archive of Formal Proofs*, 2017.

[4] B. Dongol, I. J. Hayes, and G. Struth. Relational convolution, generalised modalities and incidence algebras. *CoRR*, abs/1702.04603, 2017.

[5] P. T. Johnstone. *Stone Spaces*. Cambridge University Press, 1982.

[6] K. I. Rosenthal. *Quantales and their Applications*. Longman Scientific & Technical, 1990.