

# Partial Semigroups and Convolution Algebras

Brijesh Dongol, Victor B F Gomes, Ian J Hayes and Georg Struth

December 14, 2021

## Abstract

Partial Semigroups are relevant to the foundations of quantum mechanics and combinatorics as well as to interval and separation logics. Convolution algebras can be understood either as algebras of generalised binary modalities over ternary Kripke frames, in particular over partial semigroups, or as algebras of quantale-valued functions which are equipped with a convolution-style operation of multiplication that is parametrised by a ternary relation. Convolution algebras provide algebraic semantics for various substructural logics, including categorial, relevance and linear logics, for separation logic and for interval logics; they cover quantitative and qualitative applications. These mathematical components for partial semigroups and convolution algebras provide uniform foundations from which models of computation based on relations, program traces or pomsets, and verification components for separation or interval temporal logics can be built with little effort.

## Contents

<b>1</b>	<b>Introductory Remarks</b>	<b>2</b>
<b>2</b>	<b>Partial Semigroups</b>	<b>3</b>
2.1	Partial Semigroups . . . . .	3
2.2	Green's Preorders and Green's Relations . . . . .	3
2.3	Morphisms . . . . .	4
2.4	Locally Finite Partial Semigroups . . . . .	5
2.5	Cancellative Partial Semigroups . . . . .	5
2.6	Partial Monoids . . . . .	6
2.7	Cancellative Partial Monoids . . . . .	7
2.8	Positive Partial Monoids . . . . .	7
2.9	Positive Cancellative Partial Monoids . . . . .	8
2.10	From Partial Abelian Semigroups to Partial Abelian Monoids . . . . .	8
2.11	Alternative Definitions . . . . .	9
2.12	Product Constructions . . . . .	9
2.13	Partial Semigroup Actions and Semidirect Products . . . . .	11
<b>3</b>	<b>Models of Partial Semigroups</b>	<b>12</b>
3.1	Partial Monoids of Segments and Intervals . . . . .	12
3.2	Cancellative PAM's of Partial Functions . . . . .	14
3.3	PAM's of Disjoint Unions of Sets . . . . .	14

<b>4</b>	<b>Quantales</b>	<b>15</b>
4.1	Properties of Complete Lattices . . . . .	15
4.2	Families of Proto-Quantales . . . . .	15
4.3	Families of Quantales . . . . .	17
4.4	Quantales of Booleans and Complete Boolean Algebras . . . . .	18
4.5	Products of Quantales . . . . .	19
4.6	Quantale Modules and Semidirect Products . . . . .	20
<b>5</b>	<b>Binary Modalities and Relational Convolution</b>	<b>21</b>
5.1	Auxiliary Properties . . . . .	21
5.2	Binary Modalities . . . . .	22
5.3	Relational Convolution and Correspondence Theory . . . . .	23
5.4	Lifting to Function Spaces . . . . .	25
<b>6</b>	<b>Unary Modalities</b>	<b>26</b>
6.1	Forward and Backward Diamonds . . . . .	26
6.2	Forward and Backward Boxes . . . . .	27
6.3	Symmetries and Dualities . . . . .	27
<b>7</b>	<b>Liftings of Partial Semigroups</b>	<b>28</b>
7.1	Relational Semigroups and Partial Semigroups . . . . .	28
7.2	Liftings of Partial Abelian Semigroups . . . . .	29

## 1 Introductory Remarks

These mathematical components supply formal proofs for two articles on *Convolution Algebras* [3] and *Convolution as a Unifying Concept* [2]. They are sparsely documented and referenced; additional information can be found in these articles, and in particular the first one.

The approach generalises previous Isabelle components for covolution algebras that were intended for separation logic and used partial abelian semigroups and monoids for modelling store-heap pairs [1]. Due to the applications in separation logic, a detailed account of cancellative and positive partial abelian monoids has been included, as these structures characterise the heap succinctly. Isabelle verification components based on this approach will be submitted as a separate AFP entry.

Our article on convolution algebras [3] provides a detailed account of convolution-based semantics for Halpern-Shoham-style interval logics [4, 7], interval temporal logics [6] and duration calculi [8] based on partial monoids. While general approaches, including modal algebras over semi-infinite intervals, are supported by the mathematical components provided, additional work on store models and assignments of variables to values is needed in order to build verification components for such interval logics.

Convolution-based liftings of partial semigroups of graphs and partial orders allow formalisations of models of true concurrency such as pomset languages and concurrent Kleene algebras [5] in Isabelle, too. An AFP entry for these is in preparation.

In all these approaches, the main task is to construct suitable partial semigroups or monoids of the computational models intended, for instance, closed intervals over the reals under fusion product, unions of heaplets (i.e. partial functions) provided their domains are disjoint, disjoint unions of graphs as parallel products. Our approach then allows a generic lifting to convolution algebras on suitable function spaces with algebraic properties, for instance of heaplets to the assertion algebra of separation logic with separating conjunction as convolution [1, 2], or

of intervals to algebraic counterparts of interval temporal logics or duration calculi with the chop operation as convolution [3]. We believe that this general construction supports other applications as well—qualitative and quantitative ones.

We would like to thank Alasdair Armstrong for his help with some Isabelle proofs and Tony Hoare for many discussions that helped us shaping the general approach.

## 2 Partial Semigroups

```
theory Partial-Semigroups
  imports Main
```

```
begin
```

```
notation times (infixl  $\cdot$  70)
and times (infixl  $\oplus$  70)
```

### 2.1 Partial Semigroups

In this context, partiality is modelled by a definedness constraint  $D$  instead of a bottom element, which would make the algebra total. This is common practice in mathematics.

```
class partial-times = times +
  fixes  $D :: 'a \Rightarrow 'a \Rightarrow bool$ 
```

The definedness constraints for associativity state that the right-hand side of the associativity law is defined if and only if the left-hand side is and that, in this case, both sides are equal. This and slightly different constraints can be found in the literature.

```
class partial-semigroup = partial-times +
  assumes add-assocD:  $D\ y\ z \wedge D\ x\ (y \cdot z) \longleftrightarrow D\ x\ y \wedge D\ (x \cdot y)\ z$ 
  and add-assoc:  $D\ x\ y \wedge D\ (x \cdot y)\ z \Longrightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z)$ 
```

Every semigroup is a partial semigroup.

```
sublocale semigroup-mult  $\subseteq$  sg: partial-semigroup -  $\lambda x\ y.$  True
  <proof>
```

```
context partial-semigroup
begin
```

The following abbreviation is useful for sublocale statements.

```
abbreviation (input)  $R\ x\ y\ z \equiv D\ y\ z \wedge x = y \cdot z$ 
```

```
lemma add-assocD-var1:  $D\ y\ z \wedge D\ x\ (y \cdot z) \Longrightarrow D\ x\ y \wedge D\ (x \cdot y)\ z$ 
  <proof>
```

```
lemma add-assocD-var2:  $D\ x\ y \wedge D\ (x \cdot y)\ z \Longrightarrow D\ y\ z \wedge D\ x\ (y \cdot z)$ 
  <proof>
```

```
lemma add-assoc-var:  $D\ y\ z \wedge D\ x\ (y \cdot z) \Longrightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z)$ 
  <proof>
```

### 2.2 Green's Preorders and Green's Relations

We define the standard Green's preorders and Green's relations. They are usually defined on monoids. On (partial) semigroups, we only obtain transitive relations.

**definition**  $gR\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\preceq_R$  50) **where**

$$x \preceq_R y = (\exists z. D x z \wedge x \cdot z = y)$$

**definition**  $strict\text{-}gR\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\prec_R$  50) **where**

$$x \prec_R y = (x \preceq_R y \wedge \neg y \preceq_R x)$$

**definition**  $gL\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\preceq_L$  50) **where**

$$x \preceq_L y = (\exists z. D z x \wedge z \cdot x = y)$$

**definition**  $strict\text{-}gL\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\prec_L$  50) **where**

$$x \prec_L y = (x \preceq_L y \wedge \neg y \preceq_L x)$$

**definition**  $gH\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\preceq_H$  50) **where**

$$x \preceq_H y = (x \preceq_L y \wedge x \preceq_R y)$$

**definition**  $gJ\text{-rel} :: 'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\preceq_J$  50) **where**

$$x \preceq_J y = (\exists v w. D v x \wedge D (v \cdot x) w \wedge (v \cdot x) \cdot w = y)$$

**definition**  $gR x y = (x \preceq_R y \wedge y \preceq_R x)$

**definition**  $gL x y = (x \preceq_L y \wedge y \preceq_L x)$

**definition**  $gH x y = (x \preceq_H y \wedge y \preceq_H x)$

**definition**  $gJ x y = (x \preceq_J y \wedge y \preceq_J x)$

**definition**  $gR\text{-downset} :: 'a \Rightarrow 'a \text{ set}$  ( $\downarrow [100]100$ ) **where**

$$x \downarrow \equiv \{y. y \preceq_R x\}$$

The following counterexample rules out reflexivity.

**lemma**  $x \preceq_R x$

$\langle \text{proof} \rangle$

**lemma**  $gR\text{-rel-trans}: x \preceq_R y \Longrightarrow y \preceq_R z \Longrightarrow x \preceq_R z$

$\langle \text{proof} \rangle$

**lemma**  $gL\text{-rel-trans}: x \preceq_L y \Longrightarrow y \preceq_L z \Longrightarrow x \preceq_L z$

$\langle \text{proof} \rangle$

**lemma**  $gR\text{-add-isol}: D z y \Longrightarrow x \preceq_R y \Longrightarrow z \cdot x \preceq_R z \cdot y$

$\langle \text{proof} \rangle$

**lemma**  $gL\text{-add-isol}: D y z \Longrightarrow x \preceq_L y \Longrightarrow x \cdot z \preceq_L y \cdot z$

$\langle \text{proof} \rangle$

**definition**  $annil :: 'a \Rightarrow \text{bool}$  **where**

$$annil x = (\forall y. D x y \wedge x \cdot y = x)$$

**definition**  $annir :: 'a \Rightarrow \text{bool}$  **where**

$$annir x = (\forall y. D y x \wedge y \cdot x = x)$$

**end**

## 2.3 Morphisms

**definition**  $ps\text{-morphism} :: ('a::\text{partial-semigroup} \Rightarrow 'b::\text{partial-semigroup}) \Rightarrow \text{bool}$  **where**

*ps-morphism*  $f = (\forall x y. D x y \longrightarrow D (f x) (f y) \wedge f (x \cdot y) = (f x) \cdot (f y))$

**definition** *strong-ps-morphism*  $:: ('a::\text{partial-semigroup} \Rightarrow 'b::\text{partial-semigroup}) \Rightarrow \text{bool}$  **where**  
*strong-ps-morphism*  $f = (\text{ps-morphism } f \wedge (\forall x y. D (f x) (f y) \longrightarrow D x y))$

## 2.4 Locally Finite Partial Semigroups

In locally finite partial semigroups, elements can only be split in finitely many ways.

**class** *locally-finite-partial-semigroup* = *partial-semigroup* +  
**assumes** *loc-fin*: *finite* ( $x \downarrow$ )

## 2.5 Cancellative Partial Semigroups

**class** *cancellative-partial-semigroup* = *partial-semigroup* +  
**assumes** *add-cancell*:  $D z x \Longrightarrow D z y \Longrightarrow z \cdot x = z \cdot y \Longrightarrow x = y$   
**and** *add-cancell*:  $D x z \Longrightarrow D y z \Longrightarrow x \cdot z = y \cdot z \Longrightarrow x = y$

**begin**

**lemma** *unique-resl*:  $D x z \Longrightarrow D x z' \Longrightarrow x \cdot z = y \Longrightarrow x \cdot z' = y \Longrightarrow z = z'$   
 $\langle \text{proof} \rangle$

**lemma** *unique-resr*:  $D z x \Longrightarrow D z' x \Longrightarrow z \cdot x = y \Longrightarrow z' \cdot x = y \Longrightarrow z = z'$   
 $\langle \text{proof} \rangle$

**lemma** *gR-rel-mult*:  $D x y \Longrightarrow x \preceq_R x \cdot y$   
 $\langle \text{proof} \rangle$

**lemma** *gL-rel-mult*:  $D x y \Longrightarrow y \preceq_L x \cdot y$   
 $\langle \text{proof} \rangle$

By cancellation, the element  $z$  is uniquely defined for each pair  $x y$ , provided it exists. In both cases,  $z$  is therefore a function of  $x$  and  $y$ ; it is a quotient or residual of  $x y$ .

**lemma** *quotr-unique*:  $x \preceq_R y \Longrightarrow (\exists! z. D x z \wedge y = x \cdot z)$   
 $\langle \text{proof} \rangle$

**lemma** *quotl-unique*:  $x \preceq_L y \Longrightarrow (\exists! z. D z x \wedge y = z \cdot x)$   
 $\langle \text{proof} \rangle$

**definition** *rquot*  $y x = (\text{THE } z. D x z \wedge x \cdot z = y)$

**definition** *lquot*  $y x = (\text{THE } z. D z x \wedge z \cdot x = y)$

**lemma** *rquot-prop*:  $D x z \wedge y = x \cdot z \Longrightarrow z = \text{rquot } y x$   
 $\langle \text{proof} \rangle$

**lemma** *rquot-mult*:  $x \preceq_R y \Longrightarrow z = \text{rquot } y x \Longrightarrow x \cdot z = y$   
 $\langle \text{proof} \rangle$

**lemma** *rquot-D*:  $x \preceq_R y \Longrightarrow z = \text{rquot } y x \Longrightarrow D x z$   
 $\langle \text{proof} \rangle$

**lemma** *add-rquot*:  $x \preceq_R y \Longrightarrow (D x z \wedge x \oplus z = y \longleftrightarrow z = \text{rquot } y x)$   
 $\langle \text{proof} \rangle$

**lemma** *add-canc1*:  $D x y \implies rquot (x \cdot y) x = y$   
 ⟨proof⟩

**lemma** *add-canc2*:  $x \preceq_R y \implies x \cdot (rquot y x) = y$   
 ⟨proof⟩

**lemma** *add-canc2-prop*:  $x \preceq_R y \implies rquot y x \preceq_L y$   
 ⟨proof⟩

The next set of lemmas establishes standard Galois connections for cancellative partial semi-groups.

**lemma** *gR-galois-imp1*:  $D x z \implies x \cdot z \preceq_R y \implies z \preceq_R rquot y x$   
 ⟨proof⟩

**lemma** *gR-galois-imp21*:  $x \preceq_R y \implies z \preceq_R rquot y x \implies x \cdot z \preceq_R y$   
 ⟨proof⟩

**lemma** *gR-galois-imp22*:  $x \preceq_R y \implies z \preceq_R rquot y x \implies D x z$   
 ⟨proof⟩

**lemma** *gR-galois*:  $x \preceq_R y \implies (D x z \wedge x \cdot z \preceq_R y \longleftrightarrow z \preceq_R rquot y x)$   
 ⟨proof⟩

**lemma** *gR-rel-defined*:  $x \preceq_R y \implies D x (rquot y x)$   
 ⟨proof⟩

**lemma** *ex-add-galois*:  $D x z \implies (\exists y. x \cdot z = y \longleftrightarrow rquot y x = z)$   
 ⟨proof⟩

**end**

## 2.6 Partial Monoids

We allow partial monoids with multiple units. This is similar to and inspired by small categories.

```
class partial-monoid = partial-semigroup +
  fixes E :: 'a set
  assumes unitl-ex:  $\exists e \in E. D e x \wedge e \cdot x = x$ 
  and unitr-ex:  $\exists e \in E. D x e \wedge x \cdot e = x$ 
  and units-eq:  $e1 \in E \implies e2 \in E \implies D e1 e2 \implies e1 = e2$ 
```

Every monoid is a partial monoid.

**sublocale** *monoid-mult*  $\subseteq$  *mon*: *partial-monoid* -  $\lambda x y. True \{1\}$   
 ⟨proof⟩

**context** *partial-monoid*  
**begin**

**lemma** *units-eq-var*:  $e1 \in E \implies e2 \in E \implies e1 \neq e2 \implies \neg D e1 e2$   
 ⟨proof⟩

In partial monoids, Green's relations become preorders, but need not be partial orders.

**sublocale** *gR*: *preorder gR-rel strict-gR-rel*

*<proof>*

**sublocale** *gL: preorder gL-rel strict-gL-rel*

*<proof>*

**lemma**  $x \preceq_R y \implies y \preceq_R x \implies x = y$

*<proof>*

**lemma**  $\text{annil } x \implies \text{annil } y \implies x = y$

*<proof>*

**lemma**  $\text{annir } x \implies \text{annir } y \implies x = y$

*<proof>*

**end**

Next we define partial monoid morphisms.

**definition** *pm-morphism* :: (*a::partial-monoid*  $\Rightarrow$  *b::partial-monoid*)  $\Rightarrow$  *bool* **where**  
*pm-morphism* *f* = (*ps-morphism* *f*  $\wedge$  ( $\forall e. e \in E \longrightarrow (f\ e) \in E$ ))

**definition** *strong-pm-morphism* :: (*a::partial-monoid*  $\Rightarrow$  *b::partial-monoid*)  $\Rightarrow$  *bool* **where**  
*strong-pm-morphism* *f* = (*pm-morphism* *f*  $\wedge$  ( $\forall e. (f\ e) \in E \longrightarrow e \in E$ ))

Partial Monoids with a single unit form a special case.

**class** *partial-monoid-one* = *partial-semigroup* + *one* +  
**assumes** *oneDl*:  $D\ x\ 1$   
**and** *oneDr*:  $D\ 1\ x$   
**and** *oner*:  $x \cdot 1 = x$   
**and** *onel*:  $1 \cdot x = x$

**begin**

**sublocale** *pmo*: *partial-monoid* - - {1}  
*<proof>*

**end**

## 2.7 Cancellative Partial Monoids

**class** *cancellative-partial-monoid* = *cancellative-partial-semigroup* + *partial-monoid*

**begin**

**lemma** *canc-unitr*:  $D\ x\ e \implies x \cdot e = x \implies e \in E$   
*<proof>*

**lemma** *canc-unittl*:  $D\ e\ x \implies e \cdot x = x \implies e \in E$   
*<proof>*

**end**

## 2.8 Positive Partial Monoids

**class** *positive-partial-monoid* = *partial-monoid* +  
**assumes** *posl*:  $D\ x\ y \implies x \cdot y \in E \implies x \in E$

**and** *posr*:  $D x y \implies x \cdot y \in E \implies y \in E$

**begin**

**lemma** *pos-unittl*:  $D x y \implies e \in E \implies x \cdot y = e \implies x = e$   
*<proof>*

**lemma** *pos-unitr*:  $D x y \implies e \in E \implies x \cdot y = e \implies y = e$   
*<proof>*

**end**

## 2.9 Positive Cancellative Partial Monoids

**class** *positive-cancellative-partial-monoid* = *positive-partial-monoid* + *cancellative-partial-monoid*

**begin**

In positive cancellative monoids, the Green's relations are partial orders.

**sublocale** *pcpmR*: *order gR-rel strict-gR-rel*  
*<proof>*

**sublocale** *pcpmL*: *order gL-rel strict-gL-rel*  
*<proof>*

**end**

## 2.10 From Partial Abelian Semigroups to Partial Abelian Monoids

Next we define partial abelian semigroups. These are interesting, e.g., for the foundations of quantum mechanics and as resource monoids in separation logic.

**class** *pas* = *partial-semigroup* +  
**assumes** *add-comm*:  $D x y \implies D y x \wedge x \oplus y = y \oplus x$

**begin**

**lemma** *D-comm*:  $D x y \longleftrightarrow D y x$   
*<proof>*

**lemma** *add-comm'*:  $D x y \implies x \oplus y = y \oplus x$   
*<proof>*

**lemma** *gL-gH-rel*:  $(x \preceq_L y) = (x \preceq_H y)$   
*<proof>*

**lemma** *gR-gH-rel*:  $(x \preceq_R y) = (x \preceq_H y)$   
*<proof>*

**lemma** *annilr*:  $\text{annil } x = \text{annir } x$   
*<proof>*

**lemma** *anni-unique*:  $\text{annil } x \implies \text{annil } y \implies x = y$   
*<proof>*

**end**



The following classes collect families of partially ordered abelian semigroups and monoids.

**class** *locally-finite-pas* = *pas* + *locally-finite-partial-semigroup*

**class** *pam* = *pas* + *partial-monoid*

**class** *cancellative-pam* = *pam* + *cancellative-partial-semigroup*

**class** *positive-pam* = *pam* + *positive-partial-monoid*

**class** *positive-cancellative-pam* = *positive-pam* + *cancellative-pam*

**class** *generalised-effect-algebra* = *pas* + *partial-monoid-one*

**class** *cancellative-pam-one* = *cancellative-pam* + *partial-monoid-one*

**class** *positive-cancellative-pam-one* = *positive-cancellative-pam* + *cancellative-pam-one*

**context** *cancellative-pam-one*

**begin**

**lemma** *E-eq-one*:  $E = \{1\}$

*<proof>*

**lemma** *one-in-E*:  $1 \in E$

*<proof>*

**end**

## 2.11 Alternative Definitions

PAS's can be axiomatised more compactly as follows.

**class** *pas-alt* = *partial-times* +

**assumes** *pas-alt-assoc*:  $D x y \wedge D (x \oplus y) z \implies D y z \wedge D x (y \oplus z) \wedge (x \oplus y) \oplus z = x \oplus (y \oplus z)$

**and** *pas-alt-comm*:  $D x y \implies D y x \wedge x \oplus y = y \oplus x$

**sublocale** *pas-alt*  $\subseteq$  *palt*: *pas*

*<proof>*

Positive abelian PAM's can be axiomatised more compactly as well.

**class** *pam-pos-alt* = *pam* +

**assumes** *pos-alt*:  $D x y \implies e \in E \implies x \oplus y = e \implies x = e$

**sublocale** *pam-pos-alt*  $\subseteq$  *ppalt*: *positive-pam*

*<proof>*

## 2.12 Product Constructions

We consider two kinds of product construction. The first one combines partial semigroups with sets, the second one partial semigroups with partial semigroups. The first one is interesting for Separation Logic. Semidirect product constructions are considered later.

**instantiation** *prod* :: (*type*, *partial-semigroup*) *partial-semigroup*

**begin**

**definition**  $D\text{-prod } x y = (fst x = fst y \wedge D (snd x) (snd y))$   
**for**  $x y :: 'a \times 'b$

**definition**  $times\text{-prod} :: 'a \times 'b \Rightarrow 'a \times 'b \Rightarrow 'a \times 'b$  **where**  
 $times\text{-prod } x y = (fst x, snd x \cdot snd y)$

**instance**  
 $\langle proof \rangle$

**end**

**instantiation**  $prod :: (type, partial\text{-monoid}) partial\text{-monoid}$   
**begin**

**definition**  $E\text{-prod} :: ('a \times 'b) set$  **where**  
 $E\text{-prod} = \{x. snd x \in E\}$

**instance**  
 $\langle proof \rangle$

**end**

**instance**  $prod :: (type, pas) pas$   
 $\langle proof \rangle$

**lemma**  $prod\text{-div1}: (x1 :: 'a, y1 :: 'b :: pas) \preceq_R (x2 :: 'a, y2 :: 'b :: pas) \Longrightarrow x1 = x2$   
 $\langle proof \rangle$

**lemma**  $prod\text{-div2}: (x1, y1) \preceq_R (x2, y2) \Longrightarrow y1 \preceq_R y2$   
 $\langle proof \rangle$

**lemma**  $prod\text{-div\text{-}eq}: (x1, y1) \preceq_R (x2, y2) \longleftrightarrow x1 = x2 \wedge y1 \preceq_R y2$   
 $\langle proof \rangle$

**instance**  $prod :: (type, pam) pam$   
 $\langle proof \rangle$

**instance**  $prod :: (type, cancellative\text{-}pam) cancellative\text{-}pam$   
 $\langle proof \rangle$

**lemma**  $prod\text{-res\text{-}eq}: (x1, y1) \preceq_R (x2 :: 'a, y2 :: 'b :: cancellative\text{-}pam)$   
 $\Longrightarrow rquot (x2, y2) (x1, y1) = (x1, rquot y2 y1)$   
 $\langle proof \rangle$

**instance**  $prod :: (type, positive\text{-}pam) positive\text{-}pam$   
 $\langle proof \rangle$

**instance**  $prod :: (type, positive\text{-}cancellative\text{-}pam) positive\text{-}cancellative\text{-}pam$   $\langle proof \rangle$

**instance**  $prod :: (type, locally\text{-}finite\text{-}pas) locally\text{-}finite\text{-}pas$   
 $\langle proof \rangle$

Next we consider products of two partial semigroups.

**definition**  $ps\text{-prod}\text{-}D :: 'a :: partial\text{-semigroup} \times 'b :: partial\text{-semigroup} \Rightarrow 'a \times 'b \Rightarrow bool$   
**where**  $ps\text{-prod}\text{-}D x y \equiv D (fst x) (fst y) \wedge D (snd x) (snd y)$

**definition** *ps-prod-times* :: 'a :: partial-semigroup × 'b :: partial-semigroup ⇒ 'a × 'b ⇒ 'a × 'b  
**where** *ps-prod-times* x y = (fst x · fst y, snd x · snd y)

**interpretation** *ps-prod*: partial-semigroup *ps-prod-times* *ps-prod-D*  
⟨proof⟩

**interpretation** *pas-prod*: pas *ps-prod-times* *ps-prod-D* :: 'a :: pas × 'b :: pas ⇒ 'a × 'b ⇒ bool  
⟨proof⟩

**definition** *pm-prod-E* :: ('a :: partial-monoid × 'b :: partial-monoid) set **where**  
*pm-prod-E* = {x. fst x ∈ E ∧ snd x ∈ E}

**interpretation** *pm-prod*: partial-monoid *ps-prod-times* *ps-prod-D* *pm-prod-E*  
⟨proof⟩

**interpretation** *pam-prod*: pam *ps-prod-times* *ps-prod-D* *pm-prod-E* :: ('a :: pam × 'a :: pam) set ⟨proof⟩

## 2.13 Partial Semigroup Actions and Semidirect Products

(Semi)group actions are a standard mathematical construction. We generalise this to partial semigroups and monoids. We use it to define semidirect products of partial semigroups. A generalisation to wreath products might be added in the future.

First we define the (left) action of a partial semigroup on a set. A right action could be defined in a similar way, but we do not pursue this at the moment.

**locale** *partial-sg-laction* =  
**fixes** *Dla* :: 'a::partial-semigroup ⇒ 'b ⇒ bool  
**and** *act* :: 'a::partial-semigroup ⇒ 'b ⇒ 'b (α)  
**assumes** *act-assocD*: D x y ∧ *Dla* (x · y) p ⇔ *Dla* y p ∧ *Dla* x (α y p)  
**and** *act-assoc*: D x y ∧ *Dla* (x · y) p ⇒ α (x · y) p = α x (α y p)

Next we define the action of a partial semigroup on another partial semigroup. In the tradition of semigroup theory we use addition as a non-commutative operation for the second semigroup.

**locale** *partial-sg-sg-laction* = *partial-sg-laction* +  
**assumes** *act-distribD*: D (p::'b::partial-semigroup) q ∧ *Dla* (x::'a::partial-semigroup) (p ⊕ q) ⇔ *Dla* x p ∧ *Dla* x q ∧ D (α x p) (α x q)  
**and** *act-distrib*: D p q ∧ *Dla* x (p ⊕ q) ⇒ α x (p ⊕ q) = (α x p) ⊕ (α x q)

**begin**

Next we define the semidirect product as a partial operation and show that the semidirect product of two partial semigroups forms a partial semigroup.

**definition** *sd-D* :: ('a × 'b) ⇒ ('a × 'b) ⇒ bool **where**  
*sd-D* x y ≡ D (fst x) (fst y) ∧ *Dla* (fst x) (snd y) ∧ D (snd x) (α (fst x) (snd y))

**definition** *sd-prod* :: ('a × 'b) ⇒ ('a × 'b) ⇒ ('a × 'b) **where**  
*sd-prod* x y = ((fst x) · (fst y), (snd x) ⊕ (α (fst x) (snd y)))

**sublocale** *dp-semigroup*: partial-semigroup *sd-prod* *sd-D*  
⟨proof⟩

**end**

Finally we define the semigroup action for two partial monoids and show that the semidirect product of two partial monoids is a partial monoid.

```
locale partial-mon-sg-laction = partial-sg-sg-laction Dla
for Dla :: 'a::partial-monoid  $\Rightarrow$  'b::partial-semigroup  $\Rightarrow$  bool +
assumes act-unitl:  $e \in E \Longrightarrow Dla\ e\ p \wedge \alpha\ e\ p = p$ 
```

```
locale partial-mon-mon-laction = partial-mon-sg-laction - Dla
for Dla :: 'a::partial-monoid  $\Rightarrow$  'b::partial-monoid  $\Rightarrow$  bool +
assumes act-annir:  $e \in Ea \Longrightarrow Dla\ x\ e \wedge \alpha\ x\ e = e$ 
```

**begin**

```
definition sd-E :: ('a  $\times$  'b) set where
sd-E = {x. fst x  $\in E \wedge$  snd x  $\in E$ }
```

```
sublocale dp-semigroup : partial-monoid sd-prod sd-D sd-E
  <proof>
```

**end**

**end**

### 3 Models of Partial Semigroups

```
theory Partial-Semigroup-Models
imports Partial-Semigroups
```

**begin**

So far this section collects three models that we need for applications. Other interesting models might be added in the future. These might include binary relations, formal power series and matrices, paths in graphs under fusion, program traces with alternating state and action symbols under fusion, partial orders under series and parallel products.

#### 3.1 Partial Monoids of Segments and Intervals

Segments of a partial order are sub partial orders between two points. Segments generalise intervals in that intervals are segments in linear orders. We formalise segments and intervals as pairs, where the first coordinate is smaller than the second one. Algebras of segments and intervals are interesting in Rota's work on the foundations of combinatorics as well as for interval logics and duration calculi.

First we define the subtype of ordered pairs of one single type.

```
typedef 'a dprod = {(x::'a, y::'a). True}
  <proof>
```

```
setup-lifting type-definition-dprod
```

Such pairs form partial semigroups and partial monoids with respect to fusion.

```
instantiation dprod :: (type) partial-semigroup
begin
```

```
lift-definition D-dprod :: 'a dprod  $\Rightarrow$  'a dprod  $\Rightarrow$  bool is  $\lambda x\ y. (snd\ x = fst\ y)$  <proof>
```

**lift-definition** *times-dprod* :: 'a dprod  $\Rightarrow$  'a dprod  $\Rightarrow$  'a dprod **is**  $\lambda x y. (fst\ x, snd\ y)$   
 <proof>

**instance**  
 <proof>

**end**

**instantiation** *dprod* :: (type) *partial-monoid*  
**begin**

**lift-definition** *E-dprod* :: 'a dprod **set is**  $\{x. fst\ x = snd\ x\}$   
 <proof>

**instance**  
 <proof>

**end**

Next we define the type of segments.

**typedef** (overloaded) 'a *segment* =  $\{x::('a::order \times 'a::order). fst\ x \leq snd\ x\}$   
 <proof>

**setup-lifting** *type-definition-segment*

Segments form partial monoids as well.

**instantiation** *segment* :: (order) *partial-monoid*  
**begin**

**lift-definition** *E-segment* :: 'a *segment* **set is**  $\{x. fst\ x = snd\ x\}$   
 <proof>

**lift-definition** *D-segment* :: 'a::order *segment*  $\Rightarrow$  'a *segment*  $\Rightarrow$  bool  
**is**  $\lambda x y. (snd\ x = fst\ y)$  <proof>

**lift-definition** *times-segment* :: 'a::order *segment*  $\Rightarrow$  'a *segment*  $\Rightarrow$  'a *segment*  
**is**  $\lambda x y. \text{if } snd\ x = fst\ y \text{ then } (fst\ x, snd\ y) \text{ else } x$   
 <proof>

**instance**  
 <proof>

**end**

Next we define the function *segm* that maps segments-as-pairs to segments-as-sets.

**definition** *segm* :: 'a::order *segment*  $\Rightarrow$  'a *set* **where**  
*segm*  $x = \{y. fst\ (Rep\text{-segment}\ x) \leq y \wedge y \leq snd\ (Rep\text{-segment}\ x)\}$

**thm** *Rep-segment*

**lemma** *segm-sub-morph*:  $snd\ (Rep\text{-segment}\ x) = fst\ (Rep\text{-segment}\ y) \implies segm\ x \cup segm\ y \leq segm\ (x \cdot y)$   
 <proof>

The function *segm* is not generally a morphism.

**lemma** *snd* (*Rep-segment*  $x$ ) = *fst* (*Rep-segment*  $y$ )  $\implies$  *segm*  $x \cup$  *segm*  $y$  = *segm* ( $x \cdot y$ )  
 ⟨*proof*⟩

Intervals are segments over orders that satisfy Halpern and Shoham’s linear order property. This is still more general than linearity of the poset.

**class** *lip-order* = *order* +  
**assumes** *lip*:  $x \leq y \implies (\forall v w. (x \leq v \wedge v \leq y \wedge x \leq w \wedge w \leq y \longrightarrow v \leq w \vee w \leq v))$

The function *segm* is now a morphism.

**lemma** *segm-morph*: *snd* (*Rep-segment*  $x :: ('a :: \text{lip-order} \times 'a :: \text{lip-order})$ ) = *fst* (*Rep-segment*  $y$ )  
 $\implies$  *segm*  $x \cup$  *segm*  $y$  = *segm* ( $x \cdot y$ )  
 ⟨*proof*⟩

### 3.2 Cancellative PAM’s of Partial Functions

We show that partial functions under disjoint union form a positive cancellative PAM. This is interesting for modeling the heap in separation logic.

**type-synonym** *'a pfun* = *'a*  $\Rightarrow$  *'a option*

**definition** *ortho* :: *'a pfun*  $\Rightarrow$  *'a pfun*  $\Rightarrow$  *bool*  
**where** *ortho*  $f g \equiv \text{dom } f \cap \text{dom } g = \{\}$

**lemma** *pfun-comm*: *ortho*  $x y \implies x ++ y = y ++ x$   
 ⟨*proof*⟩

**lemma** *pfun-canc*: *ortho*  $z x \implies \text{ortho } z y \implies z ++ x = z ++ y \implies x = y$   
 ⟨*proof*⟩

**interpretation** *pfun*: *positive-cancellative-pam-one map-add ortho* {*Map.empty*} *Map.empty*  
 ⟨*proof*⟩

### 3.3 PAM’s of Disjoint Unions of Sets

This simple disjoint union construction underlies important compositions of graphs or partial orders, in particular in the context of complete joins and disjoint unions of graphs and of series and parallel products of partial orders.

**instantiation** *set* :: (*type*) *pas*  
**begin**

**definition** *D-set* :: *'a set*  $\Rightarrow$  *'a set*  $\Rightarrow$  *bool* **where**  
*D-set*  $x y \equiv x \cap y = \{\}$

**definition** *times-set* :: *'a set*  $\Rightarrow$  *'a set*  $\Rightarrow$  *'a set* **where**  
*times-set*  $x y = x \cup y$

**instance**  
 ⟨*proof*⟩

**end**

**instantiation** *set* :: (*type*) *pam*  
**begin**

**definition** *E-set* :: 'a set set where

$E\text{-set} = \{\{\}\}$

**instance**

$\langle \text{proof} \rangle$

**end**

**end**

## 4 Quantales

This entry will be merged eventually with other quantale entries and become a standalone one.

**theory** *Quantales*

**imports** *Main*

**begin**

**notation** *sup* (**infixl**  $\sqcup$  60)

**and** *inf* (**infixl**  $\sqcap$  55)

**and** *top* ( $\top$ )

**and** *bot* ( $\perp$ )

**and** *relcomp* (**infixl** ; 70)

**and** *times* (**infixl**  $\cdot$  70)

**and** *Sup* ( $\sqcup$ - [900] 900)

**and** *Inf* ( $\sqcap$ - [900] 900)

### 4.1 Properties of Complete Lattices

**lemma** (**in** *complete-lattice*) *Sup-sup-pred*:  $x \sqcup \sqcup \{y. P y\} = \sqcup \{y. y = x \vee P y\}$

$\langle \text{proof} \rangle$

**lemma** (**in** *complete-lattice*) *sup-Sup*:  $x \sqcup y = \sqcup \{x, y\}$

$\langle \text{proof} \rangle$

**lemma** (**in** *complete-lattice*) *sup-Sup-var*:  $x \sqcup y = \sqcup \{z. z \in \{x, y\}\}$

$\langle \text{proof} \rangle$

**lemma** (**in** *complete-boolean-algebra*) *shunt1*:  $x \sqcap y \leq z \iff x \leq -y \sqcup z$

$\langle \text{proof} \rangle$

**lemma** (**in** *complete-boolean-algebra*) *meet-shunt*:  $x \sqcap y = \perp \iff x \leq -y$

$\langle \text{proof} \rangle$

**lemma** (**in** *complete-boolean-algebra*) *join-shunt*:  $x \sqcup y = \top \iff -x \leq y$

$\langle \text{proof} \rangle$

### 4.2 Families of Proto-Quantales

Proto-Quantales are complete lattices equipped with an operation of composition or multiplication that need not be associative.

**class** *proto-near-quantale* = *complete-lattice* + *times* +

**assumes** *Sup-distr*:  $\sqcup X \cdot y = \sqcup \{x \cdot y \mid x \in X\}$

**begin**

**lemma** *mult-botl* [*simp*]:  $\perp \cdot x = \perp$   
*<proof>*

**lemma** *sup-distr*:  $(x \sqcup y) \cdot z = (x \cdot z) \sqcup (y \cdot z)$   
*<proof>*

**lemma** *mult-isor*:  $x \leq y \implies x \cdot z \leq y \cdot z$   
*<proof>*

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

**definition** *fres* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (**infixl**  $\leftarrow$  60) **where**  
 $z \leftarrow y = \sqcup \{x. x \cdot y \leq z\}$

**lemma** *bres-galois-imp*:  $x \cdot y \leq z \longrightarrow y \leq x \rightarrow z$   
*<proof>*

**lemma** *fres-galois*:  $x \cdot y \leq z \longleftrightarrow x \leq z \leftarrow y$   
*<proof>*

**end**

**class** *proto-pre-quantale* = *proto-near-quantale* +  
**assumes** *Sup-subdistl*:  $\sqcup \{x \cdot y \mid y \cdot y \in Y\} \leq x \cdot \sqcup Y$

**begin**

**lemma** *sup-subdistl*:  $(x \cdot y) \sqcup (x \cdot z) \leq x \cdot (y \sqcup z)$   
*<proof>*

**lemma** *mult-isol*:  $x \leq y \implies z \cdot x \leq z \cdot y$   
*<proof>*

**end**

**class** *weak-proto-quantale* = *proto-near-quantale* +  
**assumes** *weak-Sup-distl*:  $Y \neq \{\} \implies x \cdot \sqcup Y = \sqcup \{x \cdot y \mid y \cdot y \in Y\}$

**begin**

**subclass** *proto-pre-quantale*  
*<proof>*

**lemma** *sup-distl*:  $x \cdot (y \sqcup z) = (x \cdot y) \sqcup (x \cdot z)$   
*<proof>*

**lemma**  $y \leq x \rightarrow z \longrightarrow x \cdot y \leq z$   
*<proof>*

**end**

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



**assumes** *Sup-distl*:  $x \cdot \bigsqcup Y = \bigsqcup \{x \cdot y \mid y. y \in Y\}$

**begin**

**subclass** *weak-proto-quantale*  
   $\langle$ *proof* $\rangle$

**lemma** *bres-galois*:  $x \cdot y \leq z \iff y \leq x \rightarrow z$   
   $\langle$ *proof* $\rangle$

**end**

### 4.3 Families of Quantales

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

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

**begin**

**definition** *iter* :: 'a  $\Rightarrow$  'a **where**  
  *iter*  $x \equiv \bigsqcap \{y. \exists i. y = x \wedge i\}$

**lemma** *iter-ref* [*simp*]: *iter*  $x \leq 1$   
   $\langle$ *proof* $\rangle$

**lemma** *le-top*:  $x \leq \top \cdot x$   
   $\langle$ *proof* $\rangle$

**end**

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

**subclass** (**in** *pre-quantale*) *near-quantale*  $\langle$ *proof* $\rangle$

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

**subclass** (**in** *unital-pre-quantale*) *unital-near-quantale*  $\langle$ *proof* $\rangle$

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

**subclass** (**in** *weak-quantale*) *pre-quantale*  $\langle$ *proof* $\rangle$

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

**lemma** (**in** *weak-quantale*)  $x \cdot \perp = \perp$   
   $\langle$ *proof* $\rangle$

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

**lemma** (**in** *unital-weak-quantale*)  $x \cdot \perp = \perp$   
   $\langle$ *proof* $\rangle$

**subclass** (**in** *unital-weak-quantale*) *unital-pre-quantale*  $\langle$ *proof* $\rangle$

```

class quantale = proto-quantale + semigroup-mult
begin
subclass weak-quantale ⟨proof⟩

lemma mult-botr [simp]:  $x \cdot \perp = \perp$ 
  ⟨proof⟩

end

class unital-quantale = quantale + monoid-mult

subclass (in unital-quantale) unital-weak-quantale ⟨proof⟩

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

begin

lemma bres-fres-eq:  $x \rightarrow y = y \leftarrow x$ 
  ⟨proof⟩

end

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

class distrib-quantale = quantale + complete-distrib-lattice

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

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

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

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

```

#### 4.4 Quantaes of Booleans and Complete Boolean Algebras

```

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

definition one-bool = True

definition times-bool = ( $\lambda x y. x \wedge y$ )

instance
  ⟨proof⟩

end

```

**context** *complete-distrib-lattice*  
**begin**

**interpretation** *cdl-quantale: distrib-quantale - - - - - inf*  
 ⟨*proof*⟩

**end**

**context** *complete-boolean-algebra*  
**begin**

**interpretation** *cba-quantale: bool-ab-unital-quantale inf - - - - - top*  
 ⟨*proof*⟩

In this setting, residuation can be translated like classical implication.

**lemma** *cba-bres1:  $x \sqcap y \leq z \iff x \leq \text{cba-quantale.bres } y \ z$*   
 ⟨*proof*⟩

**lemma** *cba-bres2:  $x \leq -y \sqcup z \iff x \leq \text{cba-quantale.bres } y \ z$*   
 ⟨*proof*⟩

**lemma** *cba-bres-prop:  $\text{cba-quantale.bres } x \ y = -x \sqcup y$*   
 ⟨*proof*⟩

**end**

Other models will follow.

## 4.5 Products of Quantales

**definition** *Inf-prod*  $X = (\sqcap \{fst \ x \mid x. \ x \in X\}, \sqcap \{snd \ x \mid x. \ x \in X\})$

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

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

**definition** *Sup-prod*  $X = (\sqcup \{fst \ x \mid x. \ x \in X\}, \sqcup \{snd \ x \mid x. \ x \in X\})$

**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*  $= (1, 1)$

**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)*  
 ⟨*proof*⟩

**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'*

*<proof>*

**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'*

*<proof>*

**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)*

*<proof>*

## 4.6 Quantale Modules and Semidirect Products

Quantale modules are extensions of semigroup actions in that a quantale acts on a complete lattice.

**locale** *unital-quantale-module =*

**fixes** *act :: 'a::unital-quantale ⇒ 'b::complete-lattice ⇒ '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 = ⊔ {α x p | x. x ∈ X}*

**and** *act4: α x (⊔ P) = ⊔ {α x p | p. p ∈ 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)*

*<proof>*

**lemma** *act-morph2: α 1 = id*

*<proof>*

**lemma** *emp-act: α (⊔ {}) p = ⊥*

*<proof>*

**lemma** *emp-act-var: α ⊥ p = ⊥*

*<proof>*

**lemma** *act-emp: α x (⊔ {}) = ⊥*

*<proof>*

**lemma** *act-emp-var: α x ⊥ = ⊥*

*<proof>*

**lemma** *act-sup-distl: α x (p ⊔ q) = (α x p) ⊔ (α x q)*

*<proof>*

**lemma** *act-sup-distr: α (x ⊔ y) p = (α x p) ⊔ (α y p)*

*<proof>*

**lemma** *act-sup-distr-var: α (x ⊔ y) = (α x) ⊔ (α y)*

*<proof>*

Next we define the semidirect product of a unital quantale and a complete lattice.

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

**lemma** *sd-distr-aux*:

$$\bigsqcup \{snd\ x \mid x. x \in X\} \sqcup \bigsqcup \{\alpha\ (fst\ x)\ p \mid x. x \in X\} = \bigsqcup \{snd\ x \sqcup \alpha\ (fst\ x)\ p \mid x. x \in X\}$$

*<proof>*

**lemma** *sd-distr*:  $sd\text{-prod}\ (Sup\text{-prod}\ X)\ y = Sup\text{-prod}\ \{sd\text{-prod}\ x\ y \mid x. x \in X\}$

*<proof>*

**lemma** *sd-distl-aux*:  $Y \neq \{\}\ \Longrightarrow\ p \sqcup (\bigsqcup \{\alpha\ x\ (snd\ y) \mid y. y \in Y\}) = \bigsqcup \{p \sqcup \alpha\ x\ (snd\ y) \mid y. y \in Y\}$

*<proof>*

**lemma** *sd-distl*:  $Y \neq \{\}\ \Longrightarrow\ sd\text{-prod}\ x\ (Sup\text{-prod}\ Y) = Sup\text{-prod}\ \{sd\text{-prod}\ x\ y \mid y. y \in Y\}$

*<proof>*

**definition** *sd-unit* =  $(1, \perp)$

**lemma** *sd-unitl* [*simp*]:  $sd\text{-prod}\ sd\text{-unit}\ x = x$

*<proof>*

**lemma** *sd-unitr* [*simp*]:  $sd\text{-prod}\ x\ sd\text{-unit} = x$

*<proof>*

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}*

*<proof>*

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

*<proof>*

However we can show that semidirect products of (unital) quantales with complete lattices form weak (unital) quantales.

**interpretation** *dp-quantale*: *weak-quantale sd-prod Inf-prod Sup-prod inf-prod less-eq-prod less-prod sup-prod bot-prod top-prod*

*<proof>*

**interpretation** *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*

*<proof>*

end

end

## 5 Binary Modalities and Relational Convolution

**theory** *Binary-Modalities*

**imports** *Quantales*

**begin**

### 5.1 Auxiliary Properties

**lemma** *SUP-is-Sup*:  $(SUP\ f \in F. f\ y) = \bigsqcup \{(f::'a \Rightarrow 'b)::proto\text{-near-quantale}\ y \mid f. f \in F\}$

*<proof>*

**lemma** *bmod-aux1*:  $\{x \cdot g z \mid x. \exists f. x = f y \wedge f \in F\} = \{f y \cdot g z \mid f. f \in F\}$   
*<proof>*

**lemma** *bmod-aux2*:  $\{f y \cdot x \mid x. \exists g. x = g z \wedge g \in G\} = \{f y \cdot g z \mid g. g \in G\}$   
*<proof>*

**lemma** *bmod-assoc-aux1*:

$$\begin{aligned} & \sqcup \{ \sqcup \{ (f :: 'a \Rightarrow 'b :: \text{proto-near-quantale}) u \cdot g v \cdot h w \mid u v. R y u v \} \mid y w. R x y w \} \\ &= \sqcup \{ (f u \cdot g v) \cdot h w \mid u v y w. R y u v \wedge R x y w \} \end{aligned}$$

*<proof>*

**lemma** *bmod-assoc-aux2*:

$$\begin{aligned} & \sqcup \{ \sqcup \{ (f :: 'a \Rightarrow 'b :: \text{proto-near-quantale}) u \cdot g v \cdot h w \mid v w. R y v w \} \mid u y. R x u y \} \\ &= \sqcup \{ f u \cdot g v \cdot h w \mid u v w y. R y v w \wedge R x u y \} \end{aligned}$$

*<proof>*

## 5.2 Binary Modalities

Most of the development in the papers mentioned in the introduction generalises to proto-near-quantales. Binary modalities are interesting for various substructural logics over ternary Kripke frames. They also arise, e.g., as chop modalities in interval logics or as separation conjunction in separation logic. Binary modalities can be understood as a convolution operation parametrised by a ternary operation. Our development yields a unifying framework.

We would prefer a notation that is more similar to our articles, that is,  $f *_R g$ , but we don't know how we could index an infix operator by a variable in Isabelle.

**definition** *bmod-comp* ::  $('a \Rightarrow 'b \Rightarrow 'c \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'd :: \text{proto-near-quantale}) \Rightarrow ('c \Rightarrow 'd) \Rightarrow 'a \Rightarrow 'd$  ( $\otimes$ ) **where**  
 $\otimes R f g x = \sqcup \{ f y \cdot g z \mid y z. R x y z \}$

**definition** *bmod-bres* ::  $('c \Rightarrow 'b \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'd :: \text{proto-near-quantale}) \Rightarrow ('c \Rightarrow 'd) \Rightarrow 'a \Rightarrow 'd$  ( $\triangleleft$ ) **where**  
 $\triangleleft R f g x = \sqcap \{ (f y) \rightarrow (g z) \mid y z. R z y x \}$

**definition** *bmod-fres* ::  $('b \Rightarrow 'a \Rightarrow 'c \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'd :: \text{proto-near-quantale}) \Rightarrow ('c \Rightarrow 'd) \Rightarrow 'a \Rightarrow 'd$  ( $\triangleright$ ) **where**  
 $\triangleright R f g x = \sqcap \{ (f y) \leftarrow (g z) \mid y z. R y x z \}$

**lemma** *bmod-un-rel*:  $\otimes (R \sqcup S) = \otimes R \sqcup \otimes S$   
*<proof>*

**lemma** *bmod-Un-rel*:  $\otimes (\sqcup \mathcal{R}) f g x = \sqcup \{ \otimes R f g x \mid R. R \in \mathcal{R} \}$   
*<proof>*

**lemma** *bmod-sup-fun1*:  $\otimes R (f \sqcup g) = \otimes R f \sqcup \otimes R g$   
*<proof>*

**lemma** *bmod-Sup-fun1*:  $\otimes R (\sqcup \mathcal{F}) g x = \sqcup \{ \otimes R f g x \mid f. f \in \mathcal{F} \}$   
*<proof>*

**lemma** *bmod-sup-fun2*:  $\otimes R (f :: 'a \Rightarrow 'b :: \text{weak-proto-quantale}) (g \sqcup h) = \otimes R f g \sqcup \otimes R f h$   
*<proof>*

**lemma** *bmod-Sup-fun2-weak*:

**assumes**  $\mathcal{G} \neq \{\}$

**shows**  $\otimes R f (\bigsqcup \mathcal{G}) x = \bigsqcup \{\otimes R f (g::'a \Rightarrow 'b::\text{weak-proto-quantale}) x \mid g. g \in \mathcal{G}\}$

*<proof>*

**lemma** *bmod-Sup-fun2*:  $\otimes R f (\bigsqcup \mathcal{G}) x = \bigsqcup \{\otimes R f (g::'a \Rightarrow 'b::\text{proto-quantale}) x \mid g. g \in \mathcal{G}\}$

*<proof>*

**lemma** *bmod-comp-bres-galois*:  $(\forall x. \otimes R f g x \leq h x) \longleftrightarrow (\forall x. g x \leq \triangleleft R f h x)$

*<proof>*

The following Galois connection requires functions into proto-quantales.

**lemma** *bmod-comp-bres-galois*:  $(\forall x. \otimes R (f::'a \Rightarrow 'b::\text{proto-quantale}) g x \leq h x) \longleftrightarrow (\forall x. g x \leq \triangleleft R f h x)$

*<proof>*

**lemma** *bmod-comp-fres-galois*:  $(\forall x. \otimes R f g x \leq h x) \longleftrightarrow (\forall x. f x \leq \triangleright R h g x)$

*<proof>*

### 5.3 Relational Convolution and Correspondence Theory

We now fix a ternary relation  $\rho$  and can then hide the parameter in a convolution-style notation.

**class** *rel-magma* =

**fixes**  $\rho :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$

**begin**

**definition** *times-rel-fun* ::  $('a \Rightarrow 'b::\text{proto-near-quantale}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$  (**infix**  $\star$  70) **where**

$f \star g = \otimes \rho f g$

**lemma** *rel-fun-Sup-distl-weak*:

$G \neq \{\} \implies (f::'a \Rightarrow 'b::\text{weak-proto-quantale}) \star \bigsqcup G = \bigsqcup \{f \star g \mid g. g \in G\}$

*<proof>*

**lemma** *rel-fun-Sup-distl*:  $(f::'a \Rightarrow 'b::\text{proto-quantale}) \star \bigsqcup G = \bigsqcup \{f \star g \mid g. g \in G\}$

*<proof>*

**lemma** *rel-fun-Sup-distr*:  $\bigsqcup G \star (f::'a \Rightarrow 'b::\text{proto-near-quantale}) = \bigsqcup \{g \star f \mid g. g \in G\}$

*<proof>*

**end**

**class** *rel-semigroup* = *rel-magma* +

**assumes** *rel-assoc*:  $(\exists y. \rho y u v \wedge \rho x y w) \longleftrightarrow (\exists z. \rho z v w \wedge \rho x u z)$

**begin**

Nitpick produces counterexamples even for weak quantales. Hence one cannot generally lift functions into weak quantales to weak quantales.

**lemma** *bmod-assoc*:  $\otimes \rho (\otimes \rho (f::'a \Rightarrow 'b::\text{weak-quantale}) g) h x = \otimes \rho f (\otimes \rho g h) x$

*<proof>*

**lemma** *bmod-assoc*:  $\otimes \rho (\otimes \rho (f::'a \Rightarrow 'b::\text{quantale}) g) h x = \otimes \rho f (\otimes \rho g h) x$

*<proof>*

**lemma** *rel-fun-assoc*:  $((f :: 'a \Rightarrow 'b::\text{quantale}) \star g) \star h = f \star (g \star h)$

*<proof>*

**end**

**lemma**  $\otimes R (\otimes R f f) f x = \otimes R f (\otimes R f f) x$

*<proof>*

**class** *rel-monoid* = *rel-semigroup* +  
  **fixes**  $\xi :: 'a \text{ set}$   
  **assumes** *unitl-ex*:  $\exists e \in \xi. \varrho x e x$   
  **and** *unitr-ex*:  $\exists e \in \xi. \varrho x x e$   
  **and** *unitl-eq*:  $e \in \xi \Longrightarrow \varrho x e y \Longrightarrow x = y$   
  **and** *unitr-eq*:  $e \in \xi \Longrightarrow \varrho x y e \Longrightarrow x = y$

**begin**

**lemma** *xi-prop*:  $e1 \in \xi \Longrightarrow e2 \in \xi \Longrightarrow e1 \neq e2 \Longrightarrow \neg \varrho x e1 e2 \wedge \neg \varrho x e2 e1$

*<proof>*

**definition** *pid* ::  $'a \Rightarrow 'b::\text{unital-weak-quantale } (\delta)$  **where**

$\delta x = (\text{if } x \in \xi \text{ then } 1 \text{ else } \perp)$

Due to the absence of right annihilation, the right unit law fails for functions into weak quantales.

**lemma** *bmod-onel*:  $\otimes \varrho f (\delta::'a \Rightarrow 'b::\text{unital-weak-quantale}) x = f x$

*<proof>*

A unital quantale is required for this lifting.

**lemma** *bmod-onel*:  $\otimes \varrho f (\delta::'a \Rightarrow 'b::\text{unital-quantale}) x = f x$

*<proof>*

**lemma** *bmod-oner*:  $\otimes \varrho \delta f x = f x$

*<proof>*

**lemma** *pid-unitl [simp]*:  $\delta \star f = f$

*<proof>*

**lemma** *pid-unitr [simp]*:  $f \star (\delta::'a \Rightarrow 'b::\text{unital-quantale}) = f$

*<proof>*

**lemma** *bmod-assoc-weak-aux*:

$f u \cdot \bigsqcup \{g v \cdot h z \mid v z. \varrho y v z\} = \bigsqcup \{(f::'a \Rightarrow 'b::\text{weak-quantale}) u \cdot g v \cdot h z \mid v z. \varrho y v z\}$

*<proof>*

**lemma** *bmod-assoc-weak*:  $\otimes \varrho (\otimes \varrho (f::'a \Rightarrow 'b::\text{weak-quantale}) g) h x = \otimes \varrho f (\otimes \varrho g h) x$

*<proof>*

**lemma** *rel-fun-assoc-weak*:  $((f :: 'a \Rightarrow 'b::\text{weak-quantale}) \star g) \star h = f \star (g \star h)$

*<proof>*

**end**



**lemma** (in *rel-semigroup*)  $\exists id. \forall f x. (\otimes \varrho f id x = f x \wedge \otimes \varrho id f x = f x)$

*<proof>*

**class** *rel-ab-semigroup* = *rel-semigroup* +  
**assumes** *rel-comm*:  $\varrho x y z \implies \varrho x z y$

**begin**

**lemma** *bmod-comm*:  $\otimes \varrho (f :: 'a \Rightarrow 'b::ab-quantale) g = \otimes \varrho g f$   
*<proof>*

**lemma**  $\otimes \varrho f g = \otimes \varrho g f$   
*<proof>*

**lemma** *bmod-bres-fres-eq*:  $\triangleleft \varrho (f :: 'a \Rightarrow 'b::ab-quantale) g = \triangleright \varrho g f$   
*<proof>*

**lemma** *rel-fun-comm*:  $(f :: 'a \Rightarrow 'b::ab-quantale) \star g = g \star f$   
*<proof>*

**end**

**class** *rel-ab-monoid* = *rel-ab-semigroup* + *rel-monoid*

## 5.4 Lifting to Function Spaces

We lift by interpretation, since we need sort instantiations to be used for functions from PAM's to Quantales. Both instantiations cannot be used in Isabelle at the same time.

**interpretation** *rel-fun*: *weak-proto-quantale Inf Sup inf less-eq less sup bot top* :: *'a::rel-magma*  $\Rightarrow$  *'b::weak-proto-quantale times-rel-fun*  
*<proof>*

**interpretation** *rel-fun*: *proto-quantale Inf Sup inf less-eq less sup bot top* :: *'a::rel-magma*  $\Rightarrow$  *'b::proto-quantale times-rel-fun*  
*<proof>*

Nitpick shows that the lifting of weak quantales to weak quantales does not work for relational semigroups, because associativity fails.

**interpretation** *rel-fun*: *weak-quantale times-rel-fun Inf Sup inf less-eq less sup bot top*::*'a::rel-semigroup*  $\Rightarrow$  *'b::weak-quantale*

*<proof>*

Associativity is obtained when lifting from relational monoids, but the right unit law doesn't hold in the quantale on the function space, according to our above results. Hence the lifting results into a non-unital quantale, in which only the left unit law holds, as shown above. We don't provide a special class for such quantales. Hence we lift only to non-unital quantales.

**interpretation** *rel-fun*: *weak-quantale times-rel-fun Inf Sup inf less-eq less sup bot top*::*'a::rel-monoid*  $\Rightarrow$  *'b::unital-weak-quantale*  
*<proof>*

**interpretation** *rel-fun2*: *quantale times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-semigroup*  $\Rightarrow$  *'b::quantale*  
 ⟨*proof*⟩

**interpretation** *rel-fun2*: *distrib-quantale Inf Sup inf less-eq less sup bot top::'a::rel-semigroup*  $\Rightarrow$  *'b::distrib-quantale times-rel-fun* ⟨*proof*⟩

**interpretation** *rel-fun2*: *bool-quantale minus uminus inf less-eq less sup bot top::'a::rel-semigroup*  $\Rightarrow$  *'b::bool-quantale* *Inf Sup times-rel-fun* ⟨*proof*⟩

**interpretation** *rel-fun2*: *unital-quantale pid times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-monoid*  $\Rightarrow$  *'b::unital-quantale*  
 ⟨*proof*⟩

**interpretation** *rel-fun2*: *distrib-unital-quantale Inf Sup inf less-eq less sup bot top::'a::rel-monoid*  $\Rightarrow$  *'b::distrib-unital-quantale pid times-rel-fun* ⟨*proof*⟩

**interpretation** *rel-fun2*: *bool-unital-quantale minus uminus inf less-eq less sup bot top::'a::rel-monoid*  $\Rightarrow$  *'b::bool-unital-quantale* *Inf Sup pid times-rel-fun* ⟨*proof*⟩

**interpretation** *rel-fun*: *ab-quantale times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-ab-semigroup*  $\Rightarrow$  *'b::ab-quantale*  
 ⟨*proof*⟩

**interpretation** *rel-fun*: *ab-unital-quantale times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-ab-monoid*  $\Rightarrow$  *'b::ab-unital-quantale pid* ⟨*proof*⟩

**interpretation** *rel-fun2*: *distrib-ab-unital-quantale Inf Sup inf less-eq less sup bot top::'a::rel-ab-monoid*  $\Rightarrow$  *'b::distrib-ab-unital-quantale times-rel-fun pid* ⟨*proof*⟩

**interpretation** *rel-fun2*: *bool-ab-unital-quantale times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-ab-monoid*  $\Rightarrow$  *'b::bool-ab-unital-quantale minus uminus pid* ⟨*proof*⟩

end

## 6 Unary Modalities

**theory** *Unary-Modalities*  
**imports** *Binary-Modalities*  
**begin**

Unary modalities arise as specialisations of the binary ones; and as generalisations of the standard (multi-)modal operators from predicates to functions into complete lattices. They are interesting, for instance, in combination with partial semigroups or monoids, for modelling the Halpern-Shoham modalities in interval logics.

### 6.1 Forward and Backward Diamonds

**definition** *fdia* :: (*'a*  $\times$  *'b*) *set*  $\Rightarrow$  (*'b*  $\Rightarrow$  *'c::complete-lattice*)  $\Rightarrow$  *'a*  $\Rightarrow$  *'c* ((*|*-) - -) [*61,81*] *82*) **where**  
 (*|R*) *f x* =  $\sqcup \{f y \mid y. (x,y) \in R\}$

**definition** *bdia* :: (*'a*  $\times$  *'b*) *set*  $\Rightarrow$  (*'a*  $\Rightarrow$  *'c::complete-lattice*)  $\Rightarrow$  *'b*  $\Rightarrow$  *'c* ((<*|*-) - -) [*61,81*] *82*) **where**  
 (<*R*) *f y* =  $\sqcup \{f x \mid x. (x,y) \in R\}$

**definition** *c1* :: *'a*  $\Rightarrow$  *'b::unital-quantale* **where**

$$c1 \ x = 1$$

The relationship with binary modalities is as follows.

**lemma** *fdia-bmod-comp*:  $(\langle R \rangle f \ x) = \otimes (\lambda x \ y \ z. (x,y) \in R) f \ c1 \ x$   
 $\langle proof \rangle$

**lemma** *bdia-bmod-comp*:  $(\langle R \mid f \ x) = \otimes (\lambda y \ x \ z. (x,y) \in R) f \ c1 \ x$   
 $\langle proof \rangle$

**lemma** *bmod-fdia-comp*:  $\otimes R \ f \ g \ x = |\{(x,(y,z)) \mid x \ y \ z. R \ x \ y \ z\}| (\lambda(x,y). (f \ x) \cdot (g \ y)) \ x$   
 $\langle proof \rangle$

**lemma** *bmod-fdia-comp-var*:

$$\otimes R \ f \ g \ x = |\{(x,(y,z)) \mid x \ y \ z. R \ x \ y \ z\}| (\lambda(x,y). (\lambda(v,w).(v \cdot w)) (f \ x, g \ y)) \ x$$
 $\langle proof \rangle$

**lemma** *fdia-im*:  $(\langle R \rangle f \ x) = \sqcup (f \ ' R \ ' \{x\})$   
 $\langle proof \rangle$

**lemma** *fdia-un-rel*:  $fdia \ (R \cup \ S) = fdia \ R \sqcup fdia \ S$   
 $\langle proof \rangle$

**lemma** *fdia-Un-rel*:  $(\langle \bigcup \mathcal{R} \rangle f \ x) = \sqcup \{ \langle R \rangle f \ x \mid R. R \in \mathcal{R} \}$   
 $\langle proof \rangle$

**lemma** *fdia-sup-fun*:  $fdia \ R \ (f \sqcup g) = fdia \ R \ f \sqcup fdia \ R \ g$   
 $\langle proof \rangle$

**lemma** *fdia-Sup-fun*:  $(\langle R \rangle (\sqcup \mathcal{F}) \ x) = \sqcup \{ \langle R \rangle f \ x \mid f. f \in \mathcal{F} \}$   
 $\langle proof \rangle$

**lemma** *fdia-seq*:  $fdia \ (R \ ; \ S) \ f \ x = fdia \ R \ (fdia \ S \ f) \ x$   
 $\langle proof \rangle$

**lemma** *fdia-Id [simp]*:  $(\langle Id \rangle f \ x) = f \ x$   
 $\langle proof \rangle$

## 6.2 Forward and Backward Boxes

**definition** *fbox* ::  $('a \times 'b) \ set \Rightarrow ('b \Rightarrow 'c :: complete-lattice) \Rightarrow 'a \Rightarrow 'c$  ( $[-]$  - - [61,81] 82) **where**  
 $(\langle R \rangle f \ x) = \sqcap \{ f \ y \mid y. (x,y) \in R \}$

**definition** *bbbox* ::  $('a \times 'b) \ set \Rightarrow ('a \Rightarrow 'c :: complete-lattice) \Rightarrow 'b \Rightarrow 'c$  ( $[-]$  - - [61,81] 82) **where**  
 $(\langle R \mid f \ y) = \sqcap \{ f \ x \mid x. (x,y) \in R \}$

## 6.3 Symmetries and Dualities

**lemma** *fdia-fbox-demorgan*:  $(\langle R \rangle (f :: 'b \Rightarrow 'c :: complete-boolean-algebra) \ x) = - \langle R \rangle (\lambda y. -f \ y) \ x$   
 $\langle proof \rangle$

**lemma** *fbox-fdia-demorgan*:  $(\langle R \rangle (f :: 'b \Rightarrow 'c :: complete-boolean-algebra) \ x) = - \langle R \rangle (\lambda y. -f \ y) \ x$   
 $\langle proof \rangle$

**lemma** *bdia-bbox-demorgan*:  $(\langle R \mid (f :: 'b \Rightarrow 'c :: complete-boolean-algebra) \ x) = - \langle R \mid (\lambda y. -f \ y) \ x$   
 $\langle proof \rangle$

**lemma** *bbbox-bdia-demorgan*:  $( [R] (f :: 'b \Rightarrow 'c :: \text{complete-boolean-algebra}) x ) = - \langle R | (\lambda y. -f y) x$   
 $\langle \text{proof} \rangle$

**lemma** *fdia-bdia-conv*:  $( |R | f x ) = \langle \text{converse } R | f x$   
 $\langle \text{proof} \rangle$

**lemma** *fbbox-bbbox-conv*:  $( |R | f x ) = [ \text{converse } R | f x$   
 $\langle \text{proof} \rangle$

**lemma** *fdia-bbbox-galois*:  $( \forall x. ( |R | f x ) \leq g x ) \longleftrightarrow ( \forall x. f x \leq [R | g x )$   
 $\langle \text{proof} \rangle$

**lemma** *bdia-fbbox-galois*:  $( \forall x. ( \langle R | f x ) \leq g x ) \longleftrightarrow ( \forall x. f x \leq |R | g x )$   
 $\langle \text{proof} \rangle$

**lemma** *dia-conjugate*:  
 $( \forall x. ( |R | (f :: 'b \Rightarrow 'c :: \text{complete-boolean-algebra}) x ) \sqcap g x = \perp ) \longleftrightarrow ( \forall x. f x \sqcap ( \langle R | g x ) = \perp )$   
 $\langle \text{proof} \rangle$

**lemma** *box-conjugate*:  
 $( \forall x. ( |R | (f :: 'b \Rightarrow 'c :: \text{complete-boolean-algebra}) x ) \sqcup g x = \top ) \longleftrightarrow ( \forall x. f x \sqcup ( [R | g x ) = \top )$   
 $\langle \text{proof} \rangle$

**end**

## 7 Liftings of Partial Semigroups

**theory** *Partial-Semigroup-Lifting*

**imports** *Partial-Semigroups Binary-Modalities*

**begin**

First we show that partial semigroups are instances of relational semigroups. Then we extend the lifting results for relational semigroups to partial semigroups.

### 7.1 Relational Semigroups and Partial Semigroups

Every partial semigroup is a relational partial semigroup.

**context** *partial-semigroup*

**begin**

**sublocale** *rel-partial-semigroup*: *rel-semigroup*  $R$   
 $\langle \text{proof} \rangle$

**end**

Every partial monoid is a relational monoid.

**context** *partial-monoid*

**begin**

**sublocale** *rel-partial-monoid*: *rel-monoid*  $R E$   
 $\langle \text{proof} \rangle$

**end**

Every PAS is a relational abelian semigroup.

**context** *pas*  
**begin**

**sublocale** *rel-pas: rel-ab-semigroup R*  
*⟨proof⟩*

**end**

Every PAM is a relational abelian monoid.

**context** *pam*  
**begin**

**sublocale** *rel-pam: rel-ab-monoid R E* *⟨proof⟩*

**end**

## 7.2 Liftings of Partial Abelian Semigroups

Functions from partial semigroups into weak quantales form weak proto-quantales.

**instantiation** *fun :: (partial-semigroup, weak-quantale) weak-proto-quantale*  
**begin**

**definition** *times-fun :: ('a ⇒ 'b) ⇒ ('a ⇒ 'b) ⇒ 'a ⇒ 'b* **where**  
*times-fun ≡ rel-partial-semigroup.times-rel-fun*

The following counterexample shows that the associativity law may fail in convolution algebras of functions from partial semigroups into weak quantales.

**lemma** *(rel-partial-semigroup.times-rel-fun (rel-partial-semigroup.times-rel-fun f f) f) x =*  
*(rel-partial-semigroup.times-rel-fun (f::'a::partial-semigroup ⇒ 'b::weak-quantale) (rel-partial-semigroup.times-rel-fun*  
*f f)) x*

*⟨proof⟩*

**lemma** *rel-partial-semigroup.times-rel-fun (rel-partial-semigroup.times-rel-fun f g) h =*  
*rel-partial-semigroup.times-rel-fun (f::'a::partial-semigroup ⇒ 'b::weak-quantale) (rel-partial-semigroup.times-rel-fun*  
*g h)*

*⟨proof⟩*

**instance**  
*⟨proof⟩*

**end**

Functions from partial semigroups into quantales form quantales.

**instance** *fun :: (partial-semigroup, quantale) quantale*  
*⟨proof⟩*

The following counterexample shows that the right unit law may fail in convolution algebras of functions from partial monoids into weak unital quantales.

**lemma** *(rel-partial-semigroup.times-rel-fun (f::'a::partial-monoid ⇒ 'b::unital-weak-quantale) rel-partial-monoid.pid)*  
*x = f x*

$\langle proof \rangle$

Functions from partial monoids into unital quantales form unital quantales.

**instantiation**  $fun :: (partial-monoid, unital-quantale) unital-quantale$   
**begin**

**definition**  $one-fun :: 'a \Rightarrow 'b$  **where**  
 $one-fun \equiv rel-partial-monoid.pid$

**instance**  
 $\langle proof \rangle$

**end**

These lifting results extend to PASs and PAMs as expected.

**instance**  $fun :: (pam, ab-quantale) ab-quantale$   
 $\langle proof \rangle$

**instance**  $fun :: (pam, bool-ab-quantale) bool-ab-quantale \langle proof \rangle$

**instance**  $fun :: (pam, bool-ab-unital-quantale) bool-ab-unital-quantale \langle proof \rangle$

**sublocale**  $ab-quantale < abq: pas (*) \lambda - . True$   
 $\langle proof \rangle$

Finally we prove some identities that hold in function spaces.

**lemma**  $times-fun-var: (f * g) x = \sqcup \{f y * g z \mid y z. R x y z\}$   
 $\langle proof \rangle$

**lemma**  $times-fun-var2: (f * g) = (\lambda x. \sqcup \{f y * g z \mid y z. R x y z\})$   
 $\langle proof \rangle$

**lemma**  $one-fun-var1 [simp]: x \in E \Longrightarrow 1 x = 1$   
 $\langle proof \rangle$

**lemma**  $one-fun-var2 [simp]: x \notin E \Longrightarrow 1 x = \perp$   
 $\langle proof \rangle$

**lemma**  $times-fun-canc: (f * g) x = \sqcup \{f y * g (\text{rquot } x y) \mid y. y \preceq_R x\}$   
 $\langle proof \rangle$

**lemma**  $times-fun-prod: (f * g) = (\lambda(x, y). \sqcup \{f(x, y1) * g(x, y2) \mid y1 y2. R y y1 y2\})$   
 $\langle proof \rangle$

**lemma**  $one-fun-prod1 [simp]: y \in E \Longrightarrow 1(x, y) = 1$   
 $\langle proof \rangle$

**lemma**  $one-fun-prod2 [simp]: y \notin E \Longrightarrow 1(x, y) = \perp$   
 $\langle proof \rangle$

**lemma**  $fres-galois-funI: \forall x. (f * g) x \leq h x \Longrightarrow f x \leq (h \leftarrow g) x$   
 $\langle proof \rangle$

**lemma**  $times-fun-prod-canc: (f * g)(x, y) = \sqcup \{f(x, z) * g(x, \text{rquot } y z) \mid z. z \preceq_R y\}$

*<proof>*

The following statement shows, in a generalised setting, that the magic wand operator of separation logic can be lifted from the heap subtraction operation generalised to a cancellative PAM.

**lemma** *fres-lift*:  $(fres\ f\ g)\ (x::'b::cancellative-pam) = \sqcap \{(f\ y) \leftarrow (g\ z) \mid y\ z . z \preceq_R y \wedge x = rquot\ y\ z\}$   
*<proof>*

**end**

## References

- [1] B. Dongol, V. B. F. Gomes, and G. Struth. A program construction and verification tool for separation logic. In *MPC 2015*, volume 9129 of *LNCSS*, pages 137–158. Springer, 2015.
- [2] B. Dongol, I. J. Hayes, and G. Struth. Convolution as a unifying concept: Applications in separation logic, interval calculi, and concurrency. *ACM TOCL*, 17(3):15, 2016.
- [3] B. Dongol, I. J. Hayes, and G. Struth. Relational convolution, generalised modalities and incidence algebras. *CoRR*, abs/1702.04603, 2017.
- [4] J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *J. ACM*, 38(4):935–962, 1991.
- [5] T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *J. Logic and Algebraic Programming*, 80(6):266–296, 2011.
- [6] B. C. Moszkowski. A complete axiomatization of interval temporal logic with infinite time. In *LICS 2000*, pages 241–252. IEEE Computer Society, 2000.
- [7] Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991.
- [8] C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer, 2004.