Partial Semigroups and Convolution Algebras

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

August 16, 2018

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

1 Introductory Remarks 2

2 Partial Semigroups 3
  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

3 Models of Partial Semigroups 12
  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
# 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 convolution 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

### 4 Quantales

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Properties of Complete Lattices</td>
<td>15</td>
</tr>
<tr>
<td>4.2</td>
<td>Families of Proto-Quantales</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>Families of Quantales</td>
<td>17</td>
</tr>
<tr>
<td>4.4</td>
<td>Quantales of Booleans and Complete Boolean Algebras</td>
<td>18</td>
</tr>
<tr>
<td>4.5</td>
<td>Products of Quantales</td>
<td>19</td>
</tr>
<tr>
<td>4.6</td>
<td>Quantale Modules and Semidirect Products</td>
<td>20</td>
</tr>
</tbody>
</table>

### 5 Binary Modalities and Relational Convolution

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Auxiliary Properties</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>Binary Modalities</td>
<td>22</td>
</tr>
<tr>
<td>5.3</td>
<td>Relational Convolution and Correspondence Theory</td>
<td>23</td>
</tr>
<tr>
<td>5.4</td>
<td>Lifting to Function Spaces</td>
<td>25</td>
</tr>
</tbody>
</table>

### 6 Unary Modalities

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Forward and Backward Diamonds</td>
<td>26</td>
</tr>
<tr>
<td>6.2</td>
<td>Forward and Backward Boxes</td>
<td>27</td>
</tr>
<tr>
<td>6.3</td>
<td>Symmetries and Dualities</td>
<td>27</td>
</tr>
</tbody>
</table>

### 7 Liftings of Partial Semigroups

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Relational Semigroups and Partial Semigroups</td>
<td>28</td>
</tr>
<tr>
<td>7.2</td>
<td>Liftings of Partial Abelian Semigroups</td>
<td>29</td>
</tr>
</tbody>
</table>
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 · 70)
and times (infixl ⊕ 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 \( D :: 'a \Rightarrow 'a \Rightarrow \text{bool} \)

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: \( D y z \land D x \cdot (y \cdot z) \leftrightarrow D x y \land D (x \cdot y) z \)
and add-assoc: \( D x y \land D (x \cdot y) z \Rightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z) \)

Every semigroup is a partial semigroup.

sublocale semigroup-mult \( \subseteq sg: \text{partial-semigroup} - \lambda x y. \ True \)
⟨proof⟩

context partial-semigroup
begin

The following abbreviation is useful for sublocale statements.

abbreviation (input) \( R x y z \equiv D y z \land x = y \cdot z \)

lemma add-assocD-var1: \( D y z \land D x (y \cdot z) \Rightarrow D x y \land D (x \cdot y) z \)
⟨proof⟩

lemma add-assocD-var2: \( D x y \land D (x \cdot y) z \Rightarrow D y z \land D x (y \cdot z) \)
⟨proof⟩

lemma add-assoc-var: \( D y z \land D x (y \cdot z) \Rightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z) \)
⟨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-rel :: 'a ⇒ 'a ⇒ bool (infix ≤R 50) where
  x ≤R y = (∃ z. D x z ∧ x · z = y)

definition strict-gR-rel :: 'a ⇒ 'a ⇒ bool (infix <R 50) where
  x <R y = (x ≤R y ∧ ∼ y ≤R x)

definition gL-rel :: 'a ⇒ 'a ⇒ bool (infix ≤L 50) where
  x ≤L y = (∃ z. D z x ∧ x · z = y)

definition strict-gL-rel :: 'a ⇒ 'a ⇒ bool (infix <L 50) where
  x <L y = (x ≤L y ∧ ∼ y ≤L x)

definition gH-rel :: 'a ⇒ 'a ⇒ bool (infix ≤H 50) where
  x ≤H y = (∃ z. D z x ∧ x · z = y)

definition gJ-rel :: 'a ⇒ 'a ⇒ bool (infix ≤J 50) where
  x ≤J y = (∃ z. D z x ∧ x · z = y)

definition gR-downset :: 'a ⇒ 'a set (-↓ [100] 100) where
  x↓≡{ y. y ≤R x}

The following counterexample rules out reflexivity.

lemma x ≤R x
  ⟨proof⟩

lemma gR-rel-trans: x ≤R y ⇒ y ≤R z ⇒ x ≤R z
  ⟨proof⟩

lemma gL-rel-trans: x ≤L y ⇒ y ≤L z ⇒ x ≤L z
  ⟨proof⟩

lemma gR-add-isol: D z y ⇒ x ≤R y ⇒ z · x ≤R z · y
  ⟨proof⟩

lemma gL-add-isor: D y z ⇒ x ≤L y ⇒ x · z ≤L y · z
  ⟨proof⟩

definition annil :: 'a ⇒ bool where
  annil x = (∀ y. D x y ∧ x · y = x)

definition annir :: 'a ⇒ bool where
  annir x = (∀ y. D y x ∧ y · x = x)

end

2.3 Morphisms

definition ps-morphism :: ('a::partial-semigroup ⇒ 'b::partial-semigroup) ⇒ bool where
ps-morphism \( f = (\forall x y. \, D x y \rightarrow D (f x) (f y) \land f (x \cdot y) = (f x) \cdot (f y)) \)

definition strong-ps-morphism :: ('a::partial-semigroup \Rightarrow 'b::partial-semigroup) \Rightarrow bool where
  strong-ps-morphism \( f = (\mathrm{ps-morphism} \, f \land (\forall x y. \, D (f x) (f y) \rightarrow D x y)) \)

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-cancl: \( D z x = \Rightarrow D z y = \Rightarrow z \cdot x = z \cdot y = \Rightarrow x = y \)
  and add-cancr: \( D x z = \Rightarrow D y z = \Rightarrow x \cdot z = y \cdot z = \Rightarrow x = y \)

begin

lemma unique-resl: \( D x z \Rightarrow D x z' \Rightarrow x \cdot z = y \Rightarrow x \cdot z' = y \Rightarrow z = z' \)
  ⟨proof⟩

lemma unique-resr: \( D z x \Rightarrow D z' x \Rightarrow z \cdot x = y \Rightarrow z' \cdot x = y \Rightarrow z = z' \)
  ⟨proof⟩

lemma gR-rel-mult: \( D x y \Rightarrow x \preceq_R x \cdot y \)
  ⟨proof⟩

lemma gL-rel-mult: \( D x y \Rightarrow y \preceq_L x \cdot y \)
  ⟨proof⟩

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

lemma quotr-unique: \( x \preceq_R y = \Rightarrow (\exists ! z. \, D x z \land y = x \cdot z) \)
  ⟨proof⟩

lemma quotl-unique: \( x \preceq_L y = \Rightarrow (\exists ! z. \, D z x \land y = z \cdot x) \)
  ⟨proof⟩

definition rquot y x = (\( \mathrm{THE} \, z. \, D x z \land x \cdot z = y \))

definition lquot y x = (\( \mathrm{THE} \, z. \, D z x \land z \cdot x = y \))

lemma rquot-prop: \( D x z \land y = x \cdot z \Rightarrow z = rquot y x \)
  ⟨proof⟩

lemma rquot-mult: \( x \preceq_R y = \Rightarrow z = rquot y x \Rightarrow x \cdot z = y \)
  ⟨proof⟩

lemma rquot-D: \( x \preceq_R y = \Rightarrow z = rquot y x = D x z \)
  ⟨proof⟩

lemma add-rquot: \( x \preceq_R y = \Rightarrow (D x z \land x \oplus z = y \leftrightarrow z = rquot y x) \)
  ⟨proof⟩

5
lemma add-canc1: $D x y \implies \text{rquot}(x \cdot y) x = y$

lemma add-canc2: $x \preceq_R y \implies x \cdot (\text{rquot} y x) = y$

lemma add-canc2-prop: $x \preceq_R y \implies \text{rquot} y x \preceq_L y$

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

lemma gR-galois-imp1: $D x z \implies x \cdot z \preceq_R y \implies z \preceq_R \text{rquot} y x$

lemma gR-galois-imp21: $x \preceq_R y \implies z \preceq_R \text{rquot} y x \implies x \cdot z \preceq_R y$

lemma gR-galois-imp22: $x \preceq_R y \implies z \preceq_R \text{rquot} y x \implies D x z$

lemma gR-galois: $x \preceq_R y \implies (D x z \land x \cdot z \preceq_R y \iff z \preceq_R \text{rquot} y x)$

lemma gR-rel-defined: $x \preceq_R y \implies D x (\text{rquot} y x)$

lemma ex-add-galois: $D x z \implies (\exists y. x \cdot z = y \iff \text{rquot} y x = z)$

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: $\exists e \in E. D e x \land e \cdot x = x$
  and unitr-ex: $\exists e \in E. D x e \land x \cdot e = x$
  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 - $\lambda x y. \text{True} \{1\}$

context partial-monoid begin

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

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

sublocale gR: preorder gR-rel strict-gR-rel
sublocale gL: preorder gL-rel strict-gL-rel
⟨proof⟩

lemma \( x \preceq R y \implies y \preceq R x \implies x = y \)
⟨proof⟩

lemma annil x = \iff annil y = \iff x = y
⟨proof⟩

lemma annir x = \iff annir y = \iff x = y
⟨proof⟩

end

Next we define partial monoid morphisms.

definition pm-morphism :: ('a::partial-monoid ⇒ 'b::partial-monoid) ⇒ bool where
  pm-morphism f = (ps-morphism f ∧ (∀ e. e ∈ E → (f e) ∈ E))

definition strong-pm-morphism :: ('a::partial-monoid ⇒ 'b::partial-monoid) ⇒ bool where
  strong-pm-morphism f = (pm-morphism f ∧ (∀ e. (f e) ∈ E → e ∈ 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
  andoner: x · 1 = x
  andonel: 1 · 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 =⇒ x · e = x =⇒ e ∈ E
⟨proof⟩

lemma canc-unitl: D e x =⇒ e · x = x =⇒ e ∈ E
⟨proof⟩

end

2.8 Positive Partial Monoids

class positive-partial-monoid = partial-monoid +
  assumes posl: D x y =⇒ x · y ∈ E =⇒ x ∈ E
and \( \text{posr}: D \ x \ y \implies x \cdot y \in E \implies y \in E \)

begin

lemma pos-unitl: \( D \ x \ y \implies e \in E \implies x \cdot y = e \implies x = e \)
\langle proof \rangle

lemma pos-unitr: \( D \ x \ y \implies e \in E \implies x \cdot y = e \implies y = e \)
\langle proof \rangle

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
\langle proof \rangle

sublocale pcpmL: order gL-rel strict-gL-rel
\langle proof \rangle

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: \( D \ x \ y \implies D \ y \ x \land x \oplus y = y \oplus x \)

begin

lemma D-comm: \( D \ x \ y \iff D \ y \ x \)
\langle proof \rangle

lemma add-comm': \( D \ x \ y \implies x \oplus y = y \oplus x \)
\langle proof \rangle

lemma gL-gH-rel: \( ( x \preceq_L y ) = ( x \preceq_H y ) \)
\langle proof \rangle

lemma gR-gH-rel: \( ( x \preceq_R y ) = ( x \preceq_H y ) \)
\langle proof \rangle

lemma annilr: annil x = annir x
\langle proof \rangle

lemma anni-unique: annil x \implies annil y \implies x = y
\langle proof \rangle

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

\[ \text{class} \ \text{locally-finite-pas} = \text{pas} + \text{locally-finite-partial-semigroup} \]

\[ \text{class} \ \text{pam} = \text{pas} + \text{partial-monoid} \]

\[ \text{class} \ \text{cancellative-pam} = \text{pam} + \text{cancellative-partial-semigroup} \]

\[ \text{class} \ \text{positive-pam} = \text{pam} + \text{positive-partial-monoid} \]

\[ \text{class} \ \text{positive-cancellative-pam} = \text{positive-pam} + \text{cancellative-partial-semigroup} \]

\[ \text{class} \ \text{generalised-effect-algebra} = \text{pas} + \text{partial-monoid-one} \]

\[ \text{class} \ \text{cancellative-pam-one} = \text{cancellative-pam} + \text{partial-monoid-one} \]

\[ \text{class} \ \text{positive-cancellative-pam-one} = \text{positive-cancellative-pam} + \text{cancellative-pam-one} \]

\[ \text{context} \ \text{cancellative-pam-one} \]

\[ \text{begin} \]

\[ \text{lemma} \ E\text{-eq-one}: E = \{1\} \]

\[ \langle \text{proof} \rangle \]

\[ \text{lemma} \ \text{one-in-E}: 1 \in E \]

\[ \langle \text{proof} \rangle \]

\[ \text{end} \]

2.11 Alternative Definitions

PAS’s can be axiomatised more compactly as follows.

\[ \text{class} \ \text{pas-alt} = \text{partial-times} + \]

\[ \text{assumes} \ \text{pas-alt-assoc}: D x y \land D (x \oplus y) z \implies D y z \land D x (y \oplus z) \land (x \oplus y) \oplus z = x \oplus (y \oplus z) \]

\[ \text{and} \ \text{pas-alt-comm}: D x y \implies D y x \land x \oplus y = y \oplus x \]

\[ \text{sublocale} \ \text{pas-alt} \subseteq \text{palt}: \text{pas} \]

\[ \langle \text{proof} \rangle \]

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

\[ \text{class} \ \text{pam-pos-alt} = \text{pam} + \]

\[ \text{assumes} \ \text{pos-alt}: D x y \implies e \in E \implies x \oplus y = e \implies x = e \]

\[ \text{sublocale} \ \text{pam-pos-alt} \subseteq \text{ppalt}: \text{positive-pam} \]

\[ \langle \text{proof} \rangle \]

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.

\[ \text{instantiation} \ \text{prod} :: (\text{type}, \text{partial-semigroup}) \text{ partial-semigroup} \]

\[ \text{begin} \]
\textbf{definition} \( D\)-prod \( x \ y = \ (\text{fst} \ x = \text{fst} \ y \ \land \ D \ (\text{snd} \ x) \ (\text{snd} \ y)) \)

\textbf{for} \( x \ y :: \ 'a \times \ 'b \)

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

\textbf{instance} \( \langle \text{proof} \rangle \) \end

\textbf{instantiation} prod :: (\text{type}, \text{partial-monoid}) \text{ partial-monoid} \begin{align*}
\textbf{definition} & \quad E\text{-prod} :: (\ 'a \times \ 'b) \ \text{set} \ \textbf{where} \\
& \quad E\text{-prod} = \{ x. \ \text{snd} \ x \in E \}
\end{align*}

\textbf{instance} \( \langle \text{proof} \rangle \)
\end

\textbf{instance} prod :: (\text{type}, \text{pas}) \text{ pas} \langle \text{proof} \rangle

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

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

\textbf{lemma} prod-div-eq: \( (x1, y1) \preceq_R (x2, y2) \iff x1 = x2 \land y1 \preceq_R y2 \) \langle \text{proof} \rangle

\textbf{instance} prod :: (\text{type}, \text{pam}) \text{ pam} \langle \text{proof} \rangle

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

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

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

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

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

\textbf{Next we consider products of two partial semigroups.}

\textbf{definition} ps-prod-D \:: \('a :: \text{partial-semigroup} \times \ 'b :: \text{partial-semigroup} \Rightarrow 'a \times 'b \Rightarrow \text{bool} \)
\textbf{where} \( \text{ps-prod-D} \ x \ y \equiv D \ (\text{fst} \ x) \ (\text{fst} \ y) \land D \ (\text{snd} \ x) \ (\text{snd} \ y) \)
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 (p ⊕ q) = (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 q ∧ Dla x (p ⊕ q) ⇔ Dla p q ∧ Dla x q ∧ D (α x p) (α x q)
  and act-distrib: D p q ∧ Dla 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 x) ∧ D (snd y) (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⟩

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 ⇒ 'b::partial-semigroup ⇒ bool +
  assumes act-unitl: e ∈ E ⇒ Dla e p ∧ α e p = p

locale partial-mon-mon-laction = partial-mon-sg-laction - Dla
  for Dla :: 'a::partial-monoid ⇒ 'b::partial-monoid ⇒ bool +
  assumes act-annir: e ∈ Ea ⇒ Dla x e ∧ α x e = e
```

```
begin
definition sd-E :: ('a × 'b) set where
  sd-E = {x. fst x ∈ E ∧ snd x ∈ 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 ⇒ 'a dprod ⇒ bool is λx y. (snd x = fst y) ⟨proof⟩
```

12
lift-definition times-dprod :: 'a dprod ⇒ 'a dprod ⇒ 'a dprod is λx 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 × 'a::order). fst x ≤ 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 ⇒ 'a segment ⇒ bool
is λx y. (snd x = fst y) ⟨proof⟩

lift-definition times-segment :: 'a::order segment ⇒ 'a segment ⇒ 'a segment
is λx y. if snd x = fst y then (fst x, snd y) 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 ⇒ 'a set where
  segm x = \{ y. fst (Rep-segment x) ≤ y ∧ y ≤ snd (Rep-segment x)\}

  thm Rep-segment

lemma segm-sub-morph: snd (Rep-segment x) = fst (Rep-segment y) ⇒ segm x ∪ segm y ≤ segm (x · y)
⟨proof⟩

The function segm is not generally a morphism.
lemma \( \text{snd\ (Rep-segment\ } x) = \text{fst\ (Rep-segment\ } y) \implies \text{segm\ } x \cup \text{segm\ } y = \text{segm\ } (x \cdot y) \)

(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: \( x \leq y \implies (\forall v\ w. \ (x \leq v \wedge v \leq y \wedge x \leq w \wedge w \leq y \implies v \leq w \vee w \leq v)) \)

The function \( \text{segm} \) is now a morphism.

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

(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 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-set = {{}}

instance ⟨proof⟩
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 ⊔ 60)
  and inf (infixl ⊓ 55)
  and top (⊤)
  and bot (⊥)
  and relcomp (infixl ; 70)
  and times (infixl · 70)
  and Sup (⨆ · [900] 900)
  and Inf (⨅ · [900] 900)

4.1 Properties of Complete Lattices

lemma (in complete-lattice) Sup-sup-pred: x ⊔ ∪ {y. P y} = ∪ {y. y = x ∨ P y}
  ⟨proof⟩

lemma (in complete-lattice) sup-Sup: x ⊔ y = ∪ {x,y}
  ⟨proof⟩

lemma (in complete-lattice) sup-Sup-var: x ⊔ y = ∪ {z. z ∈ {x,y}}
  ⟨proof⟩

lemma (in complete-boolean-algebra) shunt1: x ∩ y ≤ z ⟷ x ≤ −y ∪ z
  ⟨proof⟩

lemma (in complete-boolean-algebra) meet-shunt: x ∩ y = ⊥ ⟷ x ≤ −y
  ⟨proof⟩

lemma (in complete-boolean-algebra) join-shunt: x ⊔ y = ⊤ ⟷ −x ≤ y
  ⟨proof⟩

4.2 Families of Proto-Quantales

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

class proto-near-quantale = complete-lattice + times +
  assumes Sup-distr: X · y = ∪ {x · y | x. x ∈ X}
begin

lemma mult-botl [simp]: \( \bot \cdot x = \bot \)
\langle proof \rangle

lemma sup-distr: \((x \sqcup y) \cdot z = (x \cdot z) \sqcup (y \cdot z)\)
\langle proof \rangle

lemma mult-isor: \(x \leq y \Rightarrow x \cdot z \leq y \cdot z\)
\langle proof \rangle

definition bres :: 'a ⇒ 'a ⇒ 'a (infixr → 60)
where
\(x \rightarrow z = \bigsqcup \{ y \cdot x \cdot y \leq z \}\)

definition fres :: 'a ⇒ 'a ⇒ 'a (infixl ← 60)
where
\(z ← y = \bigsqcup \{ x \cdot x \cdot y \leq z \}\)

lemma bres-galois-imp: \(x \cdot y \leq z \Rightarrow y \leq x \rightarrow z\)
\langle proof \rangle

lemma fres-galois: \(x \cdot y \leq z \iff x \leq z ← y\)
\langle proof \rangle

end

class proto-pre-quantale = proto-near-quantale +
assumes Sup-subdistl: \(\bigsqcup \{ x \cdot y \mid y \cdot y \in Y \} \leq x \cdot \bigsqcup Y\)
begin

lemma sup-subdistl: \((x \cdot y) \sqcup (x \cdot z) \leq x \cdot (y \sqcup z)\)
\langle proof \rangle\n
lemma mult-isol: \(x \leq y \Rightarrow z \cdot x \leq z \cdot y\)
\langle proof \rangle

end

class weak-proto-quantale = proto-near-quantale +
assumes weak-Sup-distl: \(Y \neq \{ \} \Rightarrow x \cdot \bigsqcup Y = \bigsqcup \{ x \cdot y \mid y \cdot y \in Y \}\)
begin

subclass proto-pre-quantale
\langle proof \rangle

lemma sup-distl: \(x \cdot (y \sqcup z) = (x \cdot y) \sqcup (x \cdot z)\)
\langle proof \rangle\n
lemma y ≤ x → z → x · y ≤ z
\langle proof \rangle

end

class proto-quantale = proto-near-quantale +
assumes \( \text{Sup-distl}: x \cdot \bigcup Y = \bigcup \{x \cdot y \mid y \in Y\} \)

begin

subclass weak-proto-quantale ⟨proof⟩

lemma bres-galois: \( x \cdot y \leq z \iff y \leq x \rightarrow z \) ⟨proof⟩

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 ⇒ 'a where
iter x ≡ \( \exists i. y = x \cdot i \) \}

lemma iter-ref [simp]: iter x \leq 1 ⟨proof⟩

lemma le-top: \( x \leq \top \cdot x \) ⟨proof⟩

end

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

subclass (in pre-quantale) near-quantale ⟨proof⟩

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

subclass (in unital-pre-quantale) unital-near-quantale ⟨proof⟩

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

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

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

lemma (in weak-quantale) \( x \cdot \bot = \bot \) ⟨proof⟩

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

lemma (in unital-weak-quantale) \( x \cdot \bot = \bot \) ⟨proof⟩

subclass (in unital-weak-quantale) unital-pre-quantale ⟨proof⟩
class quantale = proto-quantale + semigroup-mult
begin
subclass weak-quantale (proof)
lemma mult-botr [simp]: \( x \cdot \bot = \bot \)
(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 Quantales of Booleans and Complete Boolean Algebras
instantiation bool :: bool-ab-unital-quantale
begin
definition one-bool = True
definition times-bool = (\( \lambda x \ y. \ x \land y \))
instance
(proof)
end
context complete-distrib-lattice
begin

interpretation cdl-quantale: distrib-quantale inf

end

context complete-boolean-algebra
begin

interpretation cba-quantale: bool-ab-unital-quantale inf top

end

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

lemma cba-bres1: x \cap y \leq z \iff x \leq cba-quantale.bres y z

lemma cba-bres2: x \leq -y \cup z \iff x \leq cba-quantale.bres y z

lemma cba-bres-prop: cba-quantale.bres x y = -x \cup y

end

Other models will follow.

4.5 Products of Quantales

definition Inf-prod X = (\{fst x | x \in X\}, \{snd x | x \in X\})
definition inf-prod x y = (fst x \cap fst y, snd x \cap snd y)
definition bot-prod = (bot, bot)
definition Sup-prod X = (\{fst x | x \in X\}, \{snd x | x \in X\})
definition sup-prod x y = (fst x \cup fst y, snd x \cup snd y)
definition top-prod = (top, top)
definition less-eq-prod x y :\equiv less-eq (fst x) (fst y) \land less-eq (snd x) (snd y)
definition less-prod x y :\equiv less-eq (fst x) (fst y) \land less-eq (snd x) (snd y) \land 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 \times 'b::complete-lattice)

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 \times 'b::proto-near-quantale) times-prod
interpretation prod: proto-quantale Inf-prod Sup-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 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 y) ⊔ (α y p)
⟨proof⟩

lemma act-sup-distr-var: α x (y ⊔ p) = (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:
\[ \bigcup \{ \text{snd} \ x \mid x. x \in X \} \sqcup \bigcup \{ \alpha (\text{fst} \ x) \ p \mid x. x \in X \} = \bigcup \{ \text{snd} \ x \sqcup \alpha (\text{fst} \ x) \ p \mid x. x \in X \} \]
(\text{proof})

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

lemma sd-distl-aux: \(Y \neq \{} \implies \ p \sqcup (\bigcup \{ \alpha \ x \ \text{snd} \ y \mid y. y \in Y \}) = \bigcup \{ p \sqcup \alpha \ x \ \text{snd} \ y \mid y. y \in Y \} \)
(\text{proof})

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

definition sd-unit = \((1, \bot)\)

lemma sd-unitl [simp]: sd-prod \ sd-unit \ x = x
(\text{proof})

lemma sd-unitr [simp]: sd-prod \ x \ sd-unit = x
(\text{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 \ (\text{Sup-prod} \ Y) = \text{Sup-prod} \{ sd-prod \ x \ y \mid y. y \in Y \}
(\text{proof})

lemma sd-prod \ x \ \text{bot-prod} = \text{bot-prod}
(\text{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 \ \text{Inf-prod} \ \text{Sup-prod} \ \text{inf-prod} \ \text{less-eq-prod} \ \text{less-prod} \ \text{sup-prod} \ \text{bot-prod} \ \text{top-prod}
(\text{proof})

interpretation dpw-quantale: \unital-weak-quantale \ sd-unit \ sd-prod \ \text{Inf-prod} \ \text{Sup-prod} \ \text{inf-prod} \ \text{less-eq-prod} \ \text{less-prod} \ \text{sup-prod} \ \text{bot-prod} \ \text{top-prod}
(\text{proof})

end
end

5 Binary Modalities and Relational Convolution

theory Binary-Modalities
  imports Quantales
begin

5.1 Auxiliary Properties

lemma SUP-is-Sup: \((\text{SUP} f. f \ y) = \bigcup \{ \text{\texttt{\`a}} \Rightarrow \text{\texttt{\`b}} : \text{proto-quantale} \} y \mid f. f \in F \}\)

21
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** \( \textup{bmod-comp} :: (a \Rightarrow b \Rightarrow c \Rightarrow \text{bool}) \Rightarrow (b \Rightarrow d::\text{proto-near-quantale}) \Rightarrow (c \Rightarrow d) \Rightarrow a \Rightarrow \text{'}d \text{ ~ (\otimes)} \text{ where} \)
\[
\otimes R f g x = \bigcup \{f y \cdot g z \mid y z. R x y z\}
\]

**Definition** \( \textup{bmod-bres} :: (c \Rightarrow b \Rightarrow a \Rightarrow \text{bool}) \Rightarrow (b \Rightarrow d::\text{proto-near-quantale}) \Rightarrow (c \Rightarrow d) \Rightarrow a \Rightarrow \text{'}d \text{ ~ (\lhd)} \text{ where} \)
\[
\lhd R f g x = \bigcap \{f y \hookrightarrow (g z) \mid y z. R y x z\}
\]

**Definition** \( \textup{bmod-fres} :: (b \Rightarrow a \Rightarrow c \Rightarrow \text{bool}) \Rightarrow (b \Rightarrow d::\text{proto-near-quantale}) \Rightarrow (c \Rightarrow d) \Rightarrow a \Rightarrow \text{'}d \text{ ~ (\rightarrow)} \text{ where} \)
\[
\rightarrow R f g x = \bigcap \{f y \leftarrow (g z) \mid y z. R y x z\}
\]

**Lemma** \( \textup{bmod-un-rel}: \otimes (R \sqcup S) = \otimes R \sqcup \otimes S \)
\[
\text{proof}\]

**Lemma** \( \textup{bmod-Un-rel}: \otimes (\bigcup \mathcal{R}) f g x = \bigcup \{\otimes R f g x \mid R. R \in \mathcal{R}\} \)
\[
\text{proof}\]

**Lemma** \( \textup{bmod-sup-fun1}: \otimes R (f \sqcup g) = \otimes R f \sqcup \otimes R g \)
\[
\text{proof}\]

**Lemma** \( \textup{bmod-Sup-fun1}: \otimes R (\bigcup \mathcal{F}) g x = \bigcup \{\otimes R f g x \mid f. f \in \mathcal{F}\} \)
\[
\text{proof}\]

**Lemma** \( \textup{bmod-sup-fun2}: \otimes R (f::'a \Rightarrow b::\text{weak-quantale}) (g \sqcup h) = \otimes R f g \sqcup \otimes R f h \)
\[
\text{proof}\]
lemma bmod-Sup-fun2-weak:
assumes \( G \neq \{\} \)
shows \( \otimes R f (\bigsqcup G) x = \bigsqcup \{ \otimes R f (g :: \text{'b::weak-quantale}) x \mid g, g \in G \} \)
(proof)

lemma bmod-Sup-fun2: \( \otimes R f (\bigsqcup G) x = \bigsqcup \{ \otimes R f (g :: \text{'b::proto-quantale}) x \mid g, g \in G \} \)
(proof)

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

The following Galois connection requires functions into proto-quantales.

lemma bmod-comp-bres-galois: \((\forall x. \otimes R (f :: \text{'b::proto-quantale}) g x \leq h x) \iff (\forall x. g x \leq \triangleleft R f h x)\)
(proof)

lemma bmod-comp-fres-galois: \((\forall x. \otimes R f g x \leq h x) \iff (\forall x. f x \leq \triangledown R h g x)\)
(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 =
  fixed \( \rho :: \text{'a} \Rightarrow \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool} \)
begin

definition times-rel-fun :: \((\text{'a} \Rightarrow \text{'b::proto-near-quantale}) \Rightarrow (\text{'a} \Rightarrow \text{'b::proto-near-quantale}) \Rightarrow (\text{'a} \Rightarrow \text{'b::proto-near-quantale})\) where
  \( f \ast g = \otimes \rho f g \)

lemma rel-fun-Sup-distl-weak:
  \( G \neq \{\} \implies (f :: \text{'a} \Rightarrow \text{'b::weak-quantale}) \ast \bigsqcup G = \bigsqcup \{ f \ast g \mid g, g \in G \} \)
(proof)

lemma rel-fun-Sup-distl: \((f :: \text{'a} \Rightarrow \text{'b::proto-quantale}) \ast \bigsqcup G = \bigsqcup \{ f \ast g \mid g, g \in G \} \)
(proof)

lemma rel-fun-Sup-distr: \(\bigsqcup G \ast (f :: \text{'a} \Rightarrow \text{'b::proto-near-quantale}) = \bigsqcup \{ g \ast f \mid g, g \in G \} \)
(proof)
end

class rel-semigroup = rel-magma +
  assumes rel-assoc: \((\exists y. g y u v \land g x y w) \iff (\exists z. g z v w \land g x u z)\)
begin

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

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

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

23
proof

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

end

lemma \(\otimes R (\otimes R f)\) f x = \(\otimes R f (\otimes R f)\) x
}(proof)

class rel-monoid = rel-semigroup +
fixes \(\xi :: 'a set\)
assumes unitl-ex: \(\exists e \in \xi. \rho x e x\)
and unitr-ex: \(\exists e \in \xi. \rho x x e\)
and unitl-eq: \(e \in \xi \Rightarrow \rho x e y \Rightarrow x = y\)
and unitr-eq: \(e \in \xi \Rightarrow y e x \Rightarrow x = y\)

begin

lemma xi-prop: e1 \(\in \xi \Rightarrow e2 \in \xi \Rightarrow e1 \neq e2 \Rightarrow \neg \rho x e1 e2 \wedge \neg \rho x e2 e1\)
}(proof)

definition pid :: 'a \(\Rightarrow 'b::unital-weak-quantale\) (\(\delta\)) where
\(\delta x = (if x \in \xi then 1 else \bot)\)

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

lemma bmod-onel: \(\otimes \rho f (\delta :: 'a \Rightarrow 'b::unital-weak-quantale)\) x = f x
}(proof)

A unital quantale is required for this lifting.

lemma bmod-onel: \(\otimes \rho f (\delta :: 'a \Rightarrow 'b::unital-quantale)\) x = f x
}(proof)

lemma bmod-oner: \(\otimes \rho \delta f x = f x\)
}(proof)

lemma pid-unitl [simp]: \(\delta * f = f\)
}(proof)

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

lemma bmod-assoc-weak-aux:
\(f u \cdot \bigcup \{(g v \cdot h z \mid v z, \rho y v z\} = \bigcup \{(f :: 'a \Rightarrow 'b::weak-quantale) u \cdot g v \cdot h z \mid v z, \rho y v z\}\)
}(proof)

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

lemma rel-fan-assoc-weak: ((\(f :: 'a \Rightarrow 'b::weak-quantale\)) * g) * h = f * (g * h)
}(proof)

end
lemma (in rel-semigroup) \( \exists \text{id}. \forall f x. (\otimes g \text{id} x = f x \land \otimes \text{id} f x = f x) \)

\quad \langle \text{proof} \rangle

class rel-ab-semigroup = rel-semigroup +

\quad assumes \text{rel-comm}: g x y z \Longrightarrow g x z y

begin

lemma bmod-comm: \( \otimes (f :: 'a \Rightarrow 'b::ab-quantale) g = \otimes g f \)

\quad \langle \text{proof} \rangle

lemma \( \otimes g f g = \otimes g f \)

\quad \langle \text{proof} \rangle

lemma bmod-bres-fres-eq: \( \sqcup g (f :: 'a \Rightarrow 'b::ab-quantale) g = \sqcup g f \)

\quad \langle \text{proof} \rangle

lemma rel-fun-comm: (f :: 'a \Rightarrow 'b::ab-quantale) * g = g * f

\quad \langle \text{proof} \rangle

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

\quad \langle \text{proof} \rangle

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

\quad \langle \text{proof} \rangle

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

\quad \langle \text{proof} \rangle

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

\quad \langle \text{proof} \rangle
interpretation rel-fun2: quantale times-rel-fun Inf Sup inf less-eq less sup bot top::'a::rel-semigroup ⇒ 'b::quantale
⟨proof⟩

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

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

interpretation rel-fun2: unital-quantale Inf Sup inf less-eq less sup bot top::'a::rel-monoid ⇒ 'b::unital-quantale
⟨proof⟩

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

interpretation rel-fun2: bool-unital-quantale Inf Sup inf less-eq less sup bot top::'a::rel-monoid ⇒ 'b::bool-unital-quantale minus uminus 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 ⇒ '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 ⇒ '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 ⇒ '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 ⇒ '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 × 'b) set ⇒ ('b ⇒ 'c::complete-lattice) ⇒ 'a ⇒ 'c (( | - ) - -) [61,81] 82) where
  ( |R) f x) = ∪ {f y | y. (x,y) ∈ R}

definition bdia :: ('a × 'b) set ⇒ ('a ⇒ 'c::complete-lattice) ⇒ 'b ⇒ 'c (( - | - ) - -) [61,81] 82) where
  (⟨ R) f y) = ∪ {f x | x. (x,y) ∈ R}

definition c1 :: 'a ⇒ 'b::unital-quantale where
\[ c1 \ x = 1 \]

The relationship with binary modalities is as follows.

**Lemma** (\texttt{fdia-bmod-comp}): \((|R| \ f \ x) = \otimes (\lambda x y z. (x,y) \in R) \ f \ c1 \ x\)

\textit{Proof}

**Lemma** (\texttt{bdia-bmod-comp}): \((|R| \ f \ x) = \otimes (\lambda y x z. (x,y) \in R) \ f \ c1 \ x\)

\textit{Proof}

**Lemma** (\texttt{bmod-fdia-comp-var}): \(
\otimes R \ f \ g \ x = \{((x,y,z) \mid x y z. \ R x y z)\} \ (\lambda(x,y). (f x) \cdot (g y)) \ x
\)

\textit{Proof}

**Lemma** (\texttt{fdia-im}): \((|R| \ f \ x) = \bigcup \{f \cdot R \ " \ \{x\}\}

\textit{Proof}

**Lemma** (\texttt{fdia-an-rel}): \texttt{fdia} (\(R \cup S\)) = \texttt{fdia} R \cup \texttt{fdia} S

\textit{Proof}

**Lemma** (\texttt{fdia-Un-rel}): \((\bigcup R) \ f x = \bigcup \{|R| \ f x \mid R. \ R \in R\}

\textit{Proof}

**Lemma** (\texttt{fdia-sup-fun}): \texttt{fdia} (\(f \cup g\)) = \texttt{fdia} R \ f \cup \texttt{fdia} R \ g

\textit{Proof}

**Lemma** (\texttt{fdia-Sup-fun}): \((|R|) \ (\bigcup F) \ x = \bigcup \{|R| \ f x \mid f. \ f \in F\}

\textit{Proof}

**Lemma** (\texttt{fdia-seq}): \texttt{fdia} (\(R ; S\)) \ f x = \texttt{fdia} R (\texttt{fdia} S \ f) \ x

\textit{Proof}

**Lemma** (\texttt{fdia-Id simp}): \((|Id| \ f x) = f x

\textit{Proof}

### 6.2 Forward and Backward Boxes

**Definition** \texttt{fbox} :: \((a \times b) \ \text{set} \Rightarrow (b \Rightarrow 'c::complete-lattice) \Rightarrow 'a \Rightarrow 'c ([\cdot] - - [61,81] 82) \ where

\((|R| \ f x) = \prod \{f \ y \cdot (x,y) \in R\}\)

**Definition** \texttt{bbox} :: \((a \times b) \ \text{set} \Rightarrow ('a \Rightarrow 'c::complete-lattice) \Rightarrow 'b \Rightarrow 'c ([\cdot] - - [61,81] 82) \ where

\((|R| \ f y) = \prod \{f x \mid x. \ (x,y) \in R\}\)

### 6.3 Symmetries and Dualities

**Lemma** (\texttt{fdia-fbox-demorgan}): \((|R|) (f::{'b \Rightarrow 'c::complete-boolean-algebra}) \ x) = - |R| (\lambda y. \ f y) x

\textit{Proof}

**Lemma** (\texttt{bbox-fdia-demorgan}): \((|R|) (f::{'b \Rightarrow 'c::complete-boolean-algebra}) \ x) = - |R| (\lambda y. \ f y) x

\textit{Proof}

**Lemma** (\texttt{bdia-bbox-demorgan}): \((|R|) (f::{'b \Rightarrow 'c::complete-boolean-algebra}) \ x) = - |R| (\lambda y. \ f y) x

\textit{Proof}
lemma bbox-bdia-demorgan: \( [R] (f::'b \Rightarrow 'c::complete-boolean-algebra) x) = - [R] (\lambda y. - f y) x \)
(proof)

lemma fdia-bdia-conv: \( [R] f x = [converse R] f x \)
(proof)

lemma fbox-bbox-conv: \( [R] f x = [converse R] f x \)
(proof)

lemma fdia-bbox-galois: \( \forall x. ([R] f x) \leq g x \leftrightarrow (\forall x. f x \leq [R] g x) \)
(proof)

lemma bdia-fbox-galois: \( \forall x. ([R] f x) \leq g x \leftrightarrow (\forall x. f x \leq [R] g x) \)
(proof)

lemma dia-conjugate:
\( \forall x. (|R| f::'b \Rightarrow 'c::complete-boolean-algebra) x) \sqcap g x = \bot \leftrightarrow (\forall x. f x \sqcap ([R] g x) = \bot) \)
(proof)

lemma box-conjugate:
\( \forall x. (|R| f::'b \Rightarrow 'c::complete-boolean-algebra) x) \sqcup g x = \top \leftrightarrow (\forall x. f x \sqcup ([R] g x) = \top) \)
(proof)

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
(proof)
end

Every partial monoid is a relational monoid.
context partial-monoid
begin
sublocale rel-partial-monoid: rel-monoid R E
(proof)
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
Functions from partial monoids into unital quantales form unital quantales.

\textbf{instantiation }fun :: (partial-monoid, unital-quantale) unital-quantale

\begin{proof}
\begin{definition}
  \textbf{one-fun} :: 'a ⇒ 'b where
  \textbf{one-fun} ≡ \text{rel-partial-monoid.pid}
\end{definition}
\end{proof}

\textbf{instance}

\begin{proof}
These lifting results extend to PASs and PAMs as expected.

\textbf{instance }fun :: (pam, ab-quantale) ab-quantale
\begin{proof}
\textbf{instance }fun :: (pam, bool-ab-quantale) bool-ab-quantale
\begin{proof}
\textbf{instance }fun :: (pam, bool-ab-unital-quantale) bool-ab-unital-quantale
\begin{proof}
\textbf{sublocale} ab-quantale < abq: pas (∗) λ - -. True
\begin{proof}
Finally we prove some identities that hold in function spaces.

\textbf{lemma }times-fun-var: \((f * g) x = \bigcup \{f y * g z \mid y z. R x y z\}\)
\begin{proof}
\textbf{lemma }times-fun-var2: \((f * g) = (\lambda x. \bigcup \{f y * g z \mid y z. R x y z\})\)
\begin{proof}
\textbf{lemma }one-fun-var1 [simp]: \(x \in E \implies 1 x = 1\)
\begin{proof}
\textbf{lemma }one-fun-var2 [simp]: \(x \notin E \implies 1 x = \bot\)
\begin{proof}
\textbf{lemma }times-fun-canc: \((f * g) x = \bigcup \{f y * g (rquot x y) \mid y. y \preceq_R x\}\)
\begin{proof}
\textbf{lemma }times-fun-prod: \((f * g) = (\lambda (x, y). \bigcup \{f (x, y1) * g (x, y2) \mid y1 y2. R y y1 y2\})\)
\begin{proof}
\textbf{lemma }one-fun-prod1 [simp]: \(y \in E \implies 1 (x, y) = 1\)
\begin{proof}
\textbf{lemma }one-fun-prod2 [simp]: \(y \notin E \implies 1 (x, y) = \bot\)
\begin{proof}
\textbf{lemma }fres-galois-funI: \(\forall x. (f * g) x \preceq h x \implies f x \preceq (h \leftarrow g) x\)
\begin{proof}
\textbf{lemma }times-fun-prod-canc: \((f * g) (x, y) = \bigcup \{f (x, z) * g (x, rquot y z) \mid z. z \preceq_R y\}\)
\begin{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{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.

\[ \text{lemma fres-lift: } (\text{fres } f \ g) (x : 'b::cancellative-pam) = \{ (f y) \leftarrow (g z) \mid y z . z \preceq_R y \land x = rquot y z \} \]

References


