

# Binary Multirelations

Hitoshi Furusawa and Georg Struth

June 20, 2024

## Abstract

Binary multirelations associate elements of a set with its subsets; hence they are binary relations of type  $A \times 2^A$ . Applications include alternating automata, models and logics for games, program semantics with dual demonic and angelic nondeterministic choices and concurrent dynamic logics. This proof document supports an arXiv article that formalises the basic algebra of multirelations and proposes axiom systems for them, ranging from weak bi-monoids to weak bi-quantales.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>C-Algebras</b>	<b>2</b>
2.1	C-Monoids	2
2.2	C-Trioids	6
2.3	Results for Concurrent Dynamic Algebra	10
2.4	C-Lattices	12
2.5	Domain in C-Lattices	20
2.6	Structural Properties of C-Lattices	22
2.7	Terminal and Nonterminal Elements	29
2.8	Powers in C-Algebras	36
2.9	C-Kleene Algebras	38
2.10	C-Omega Algebras	41
2.11	C-Nabla Algebras	43
2.12	Proto-Quantales	45
<b>3</b>	<b>Multirelations</b>	<b>47</b>
3.1	Basic Definitions	47
3.2	Multirelations and Proto-Dioids	48
3.3	Simple Properties	49
3.4	Multirelations and C-Lattices	55
3.5	Terminal and Nonterminal Elements	60
3.6	Multirelations, Proto-Quantales and Iteration	62

3.7 Further Counterexamples . . . . .	62
3.8 Relationship with Up-Closed Multirelations . . . . .	63

## 1 Introduction

This proof document contains the formal proofs for an article on *Taming Multirelations* [2]. Individual cross-references to statements in [2] have been added to this document so that both can be read in parallel. The first part of this document contains algebraic axiom systems and equational proofs. Some of these proofs are presented in a human-readable style to indicate the kind of algebraic reasoning involved. The second part contains set-theoretic reasoning with concrete multirelations. Its main purpose is to justify the algebraic development and to prepare the soundness proofs of the algebraic axiomatisations with respect to the concrete multirelational model. Set-theoretic reasoning with multirelations tends to be very tedious and showing detailed proofs has not been the aim.

The algebras of multirelations proposed are based on Peleg’s multirelational semantics for concurrent dynamic logic [3]. The most basic axiom systems consider multirelations under the operations of sequential and concurrent composition with two corresponding units. These are enriched by lattice operations and various fixpoints. A main source of complexity is the set-theoretic definition of sequential composition of multirelations, which is based on higher-order logic. Its use often requires the Axiom of Choice. In addition, sequential composition is not associative.

Part of this formalisation is also relevant to a previous approach to concurrent dynamic algebra by Furusawa and Struth [1]. More material on variants of multirelations, game algebras and concurrent dynamic algebras will be added in the future.

The authors are indebted to Alasdair Armstrong and Victor Gomes for help with some tricky Isabelle proofs.

## 2 C-Algebras

```
theory C-Algebras
imports Kleene-Algebra.Dioid
begin
```

```
no-notation
times (infixl · 70)
```

### 2.1 C-Monoids

We start with the c-monoid axioms. These can be found in Section 4 of [2].

```
class proto-monoid =
```

```

fixes s-id :: 'a (1 $\sigma$ )
and s-prod :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\cdot$  80)
assumes s-prod-idl [simp]: 1 $\sigma$   $\cdot$  x = x
and s-prod-idr [simp]: x  $\cdot$  1 $\sigma$  = x

class proto-bi-monoid = proto-monoid +
  fixes c-id :: 'a (1 $\pi$ )
  and c-prod :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\parallel$  80)
  assumes c-prod-idl [simp]: 1 $\pi$   $\parallel$  x = x
  and c-prod-assoc: (x  $\parallel$  y)  $\parallel$  z = x  $\parallel$  (y  $\parallel$  z)
  and c-prod-comm: x  $\parallel$  y = y  $\parallel$  x

class c-monoid = proto-bi-monoid +
  assumes c1 [simp]: (x  $\cdot$  1 $\pi$ )  $\parallel$  x = x
  and c2 [simp]: ((x  $\cdot$  1 $\pi$ )  $\parallel$  1 $\sigma$ )  $\cdot$  y = (x  $\cdot$  1 $\pi$ )  $\parallel$  y
  and c3: (x  $\parallel$  y)  $\cdot$  1 $\pi$  = (x  $\cdot$  1 $\pi$ )  $\parallel$  (y  $\cdot$  1 $\pi$ )
  and c4: (x  $\cdot$  y)  $\cdot$  1 $\pi$  = x  $\cdot$  (y  $\cdot$  1 $\pi$ )
  and c5 [simp]: 1 $\sigma$   $\parallel$  1 $\sigma$  = 1 $\sigma$ 

```

**begin**

Next we define domain explicitly as at the beginning of Section 4 in [2] and start proving the algebraic facts from Section 4. Those involving concrete multirelations, such as Proposition 4.1, are considered in the theory file for multirelations.

**definition** (**in** *c-monoid*) *d* :: 'a  $\Rightarrow$  'a **where**

*d* x = (x  $\cdot$  1 $\pi$ )  $\parallel$  1 $\sigma$

**lemma** *c-prod-idr* [*simp*]: x  $\parallel$  1 $\pi$  = x

**by** (*simp* *add*: *local.c-prod-comm*)

We prove the retraction properties of Lemma 4.2.

**lemma** *c-idem* [*simp*]: 1 $\pi$   $\cdot$  1 $\pi$  = 1 $\pi$

**by** (*metis* *c-prod-idr* *local.c1*)

**lemma** *d-idem* [*simp*]: *d* (*d* x) = *d* x

**by** (*simp* *add*: *local.d-def*)

**lemma** *p-id-idem*: (x  $\cdot$  1 $\pi$ )  $\cdot$  1 $\pi$  = x  $\cdot$  1 $\pi$

**by** (*simp* *add*: *local.c4*)

Lemma 4.3.

**lemma** *c2-d*: *d* x  $\cdot$  y = (x  $\cdot$  1 $\pi$ )  $\parallel$  y

**by** (*simp* *add*: *local.d-def*)

**lemma** *cd-2-var*: *d* (x  $\cdot$  1 $\pi$ )  $\cdot$  y = (x  $\cdot$  1 $\pi$ )  $\parallel$  y

**by** (*simp* *add*: *c2-d* *local.c4*)

**lemma** *dc-prop1* [*simp*]:  $d x \cdot 1_\pi = x \cdot 1_\pi$   
**by** (*simp add: c2-d*)

**lemma** *dc-prop2* [*simp*]:  $d (x \cdot 1_\pi) = d x$   
**by** (*simp add: local.c4 local.d-def*)

**lemma** *ds-prop* [*simp*]:  $d x \parallel 1_\sigma = d x$   
**by** (*simp add: local.c-prod-assoc local.d-def*)

**lemma** *dc* [*simp*]:  $d 1_\pi = 1_\sigma$   
**by** (*simp add: local.d-def*)

Part (5) of this Lemma has already been verified above. The next two statements verify the two algebraic properties mentioned in the proof of Proposition 4.4.

**lemma** *dc-iso* [*simp*]:  $d (d x \cdot 1_\pi) = d x$   
**by** *simp*

**lemma** *cd-iso* [*simp*]:  $d (x \cdot 1_\pi) \cdot 1_\pi = x \cdot 1_\pi$   
**by** *simp*

Proposition 4.5.

**lemma** *d-conc6*:  $d (x \parallel y) = d x \parallel d y$

**proof** –

**have**  $d (x \parallel y) = ((x \parallel y) \cdot 1_\pi) \parallel 1_\sigma$   
**by** (*simp add: local.d-def*)

**also have**  $\dots = (x \cdot 1_\pi) \parallel (y \cdot 1_\pi) \parallel 1_\sigma$   
**by** (*simp add: local.c3*)

**finally show** *?thesis*

**by** (*metis ds-prop local.c-prod-assoc local.c-prod-comm local.d-def*)

**qed**

**lemma** *d-conc-s-prod-ax*:  $d x \parallel d y = d x \cdot d y$

**proof** –

**have**  $d x \parallel d y = (x \cdot 1_\pi) \parallel 1_\sigma \parallel d y$   
**using** *local.d-def* **by** *presburger*

**also have**  $\dots = (x \cdot 1_\pi) \parallel d y$

**using** *d-conc6 local.c3 local.c-prod-assoc local.d-def* **by** *auto*

**also have**  $\dots = ((x \cdot 1_\pi) \parallel 1_\sigma) \cdot d y$

**by** *simp*

**finally show** *?thesis*

**using** *local.d-def* **by** *auto*

**qed**

**lemma** *d-rest-ax* [*simp*]:  $d x \cdot x = x$

**by** (*simp add: c2-d*)

**lemma** *d-loc-ax* [*simp*]:  $d (x \cdot d y) = d (x \cdot y)$

**proof** –

**have**  $d(x \cdot d y) = (x \cdot d y \cdot 1_\pi) \parallel 1_\sigma$   
**by** (*simp add: local.d-def*)  
**also have**  $\dots = (x \cdot y \cdot 1_\pi) \parallel 1_\sigma$   
**by** (*simp add: local.c4*)  
**finally show** *?thesis*  
**by** (*simp add: local.d-def*)  
**qed**

**lemma** *d-exp-ax* [*simp*]:  $d(d x \cdot y) = d x \cdot d y$   
**proof** –  
**have**  $d(d x \cdot y) = d(d x \cdot d y)$   
**by** (*simp add: d-conc6*)  
**also have**  $\dots = d(d(x \parallel y))$   
**by** (*simp add: d-conc6 d-conc-s-prod-ax*)  
**also have**  $\dots = d(x \parallel y)$   
**by** *simp*  
**finally show** *?thesis*  
**by** (*simp add: d-conc6 d-conc-s-prod-ax*)  
**qed**

**lemma** *d-comm-ax*:  $d x \cdot d y = d y \cdot d x$   
**proof** –  
**have**  $(d x) \cdot (d y) = d(x \parallel y)$   
**by** (*simp add: d-conc6 d-conc-s-prod-ax*)  
**also have**  $\dots = d(y \parallel x)$   
**using** *local.c-prod-comm* **by** *auto*  
**finally show** *?thesis*  
**by** (*simp add: d-conc6 d-conc-s-prod-ax*)  
**qed**

**lemma** *d-s-id-prop* [*simp*]:  $d 1_\sigma = 1_\sigma$   
**using** *local.d-def* **by** *auto*

Next we verify the conditions of Proposition 4.6.

**lemma** *d-s-prod-closed* [*simp*]:  $d(d x \cdot d y) = d x \cdot d y$   
**by** *simp*

**lemma** *d-p-prod-closed* [*simp*]:  $d(d x \parallel d y) = d x \parallel d y$   
**using** *c2-d d-conc6* **by** *auto*

**lemma** *d-idem2* [*simp*]:  $d x \cdot d x = d x$   
**by** (*metis d-exp-ax d-rest-ax*)

**lemma** *d-assoc*:  $(d x \cdot d y) \cdot d z = d x \cdot (d y \cdot d z)$   
**proof** –  
**have**  $\bigwedge x y. d x \cdot d y = d(x \parallel y)$   
**by** (*simp add: d-conc6 d-conc-s-prod-ax*)  
**thus** *?thesis*  
**by** (*simp add: local.c-prod-assoc*)

qed

**lemma** *iso-1* [*simp*]:  $(d\ x \cdot 1_\pi) \parallel 1_\sigma = d\ x$   
**by** (*simp add: local.d-def*)

Lemma 4.7.

**lemma** *x-c-par-idem* [*simp*]:  $(x \cdot 1_\pi) \parallel (x \cdot 1_\pi) = x \cdot 1_\pi$

**proof** –

**have**  $(x \cdot 1_\pi) \parallel (x \cdot 1_\pi) = d\ x \cdot (x \cdot 1_\pi)$

**using** *c2-d* **by** *auto*

**also have**  $\dots = d\ (x \cdot 1_\pi) \cdot (x \cdot 1_\pi)$

**by** *simp*

**finally show** *?thesis*

**using** *d-rest-ax* **by** *presburger*

qed

**lemma** *d-idem-par* [*simp*]:  $d\ x \parallel d\ x = d\ x$

**by** (*simp add: d-conc-s-prod-ax*)

**lemma** *d-inter-r*:  $d\ x \cdot (y \parallel z) = (d\ x \cdot y) \parallel (d\ x \cdot z)$

**proof** –

**have**  $(d\ x) \cdot (y \parallel z) = (x \cdot 1_\pi) \parallel y \parallel z$

**using** *c2-d local.c-prod-assoc* **by** *auto*

**also have**  $\dots = (x \cdot 1_\pi) \parallel y \parallel (x \cdot 1_\pi) \parallel z$

**using** *local.c-prod-assoc local.c-prod-comm* **by** *force*

**finally show** *?thesis*

**by** (*simp add: c2-d local.c-prod-assoc*)

qed

Now we provide the counterexamples of Lemma 4.8.

**lemma**  $(x \parallel y) \cdot d\ z = (x \cdot d\ z) \parallel (y \cdot d\ z)$

**nitpick**

**oops**

**lemma**  $(x \cdot y) \cdot d\ z = x \cdot (y \cdot d\ z)$

**nitpick**

**oops**

**lemma**  $1_\pi \cdot x = 1_\pi$

**nitpick**

**oops**

end

## 2.2 C-Trioids

We can now define the class of c-trioids and prove properties in this class. This covers the algebraic material of Section 5 in [2].

```

class proto-diod = join-semilattice-zero + proto-monoid +
  assumes s-prod-distr:  $(x + y) \cdot z = x \cdot z + y \cdot z$ 
  and s-prod-subdistl:  $x \cdot y + x \cdot z \leq x \cdot (y + z)$ 
  and s-prod-annil [simp]:  $0 \cdot x = 0$ 

```

```
begin
```

```

lemma s-prod-isol:  $x \leq y \implies z \cdot x \leq z \cdot y$ 
  by (metis join.sup.boundedE order-prop s-prod-subdistl)

```

```

lemma s-prod-isor:  $x \leq y \implies x \cdot z \leq y \cdot z$ 
  using local.order-prop local.s-prod-distr by auto

```

```
end
```

```

class proto-trioid = proto-diod + proto-bi-monoid +
  assumes p-prod-distl:  $x \parallel (y + z) = x \parallel y + x \parallel z$ 
  and p-rpd-annir [simp]:  $x \parallel 0 = 0$ 

```

```

sublocale proto-trioid  $\subseteq$  ab-semigroup-mult c-prod
proof

```

```

  fix x y z
  show  $x \parallel y \parallel z = x \parallel (y \parallel z)$ 
    by (rule c-prod-assoc)
  show  $x \parallel y = y \parallel x$ 
    by (rule c-prod-comm)

```

```
qed
```

```

sublocale proto-trioid  $\subseteq$  dioid-one-zero (+) ( $\parallel$ )  $1_\pi$  0 ( $\leq$ ) ( $<$ )
proof

```

```

  fix x y z
  show  $(x + y) \parallel z = x \parallel z + y \parallel z$ 
    by (simp add: local.c-prod-comm local.p-prod-distl)
  show  $1_\pi \parallel x = x$ 
    using local.c-prod-idl by blast
  show  $x \parallel 1_\pi = x$ 
    by (simp add: local.mult-commute)
  show  $0 + x = x$ 
    by (rule add.left-neutral)
  show  $0 \parallel x = 0$ 
    by (simp add: local.mult-commute)
  show  $x \parallel 0 = 0$ 
    by (rule p-rpd-annir)
  show  $x + x = x$ 
    by (rule add-idem)
  show  $x \parallel (y + z) = x \parallel y + x \parallel z$ 
    by (rule p-prod-distl)

```

```
qed
```

```

class c-trioid = proto-trioid + c-monoid +
  assumes c6:  $x \cdot 1_\pi \leq 1_\pi$ 

```

```
begin
```

We show that every c-trioid is a c-monoid.

```
subclass c-monoid ..
```

```
subclass proto-trioid ..
```

```

lemma  $1_\pi \cdot 0 = 1_\pi$ 
  nitpick
  oops

```

```

lemma zero-p-id-prop [simp]:  $(x \cdot 0) \cdot 1_\pi = x \cdot 0$ 
  by (simp add: local.c4)

```

The following facts prove and refute properties related to sequential and parallel subidentities.

```

lemma d-subid:  $d x = x \implies x \leq 1_\sigma$ 
  by (metis local.c6 local.c-idem local.d-def local.dc local.mult-isor)

```

```

lemma  $x \leq 1_\sigma \implies d x = x$ 
  nitpick
  oops

```

```

lemma p-id-term:  $x \cdot 1_\pi = x \implies x \leq 1_\pi$ 
  by (metis local.c6)

```

```

lemma  $x \leq 1_\pi \implies x \cdot 1_\pi = x$ 
  nitpick
  oops

```

Proposition 5.1. is covered by the theory file on multirelations. We verify the remaining conditions in Proposition 5.2.

```

lemma dlp-ax:  $x \leq d x \cdot x$ 
  by simp

```

```

lemma d-add-ax:  $d (x + y) = d x + d y$ 
proof -
  have  $d (x + y) = ((x + y) \cdot 1_\pi) \parallel 1_\sigma$ 
    using local.d-def by blast
  also have  $\dots = (x \cdot 1_\pi) \parallel 1_\sigma + (y \cdot 1_\pi) \parallel 1_\sigma$ 
    by (simp add: local.distrib-right local.s-prod-distr)
  finally show ?thesis
    by (simp add: local.d-def)
qed

```



**lemma** *d-sub-id-ax*:  $d x \leq 1_\sigma$   
**proof** –  
  **have**  $d x = (x \cdot 1_\pi) \parallel 1_\sigma$   
  **by** (*simp add: local.d-def*)  
  **also have**  $\dots \leq 1_\pi \parallel 1_\sigma$   
  **using** *local.c6 local.mult-isor* **by** *blast*  
  **finally show** *?thesis*  
  **by** *simp*  
**qed**

**lemma** *d-zero-ax* [*simp*]:  $d 0 = 0$   
**by** (*simp add: local.d-def*)

We verify the algebraic conditions in Proposition 5.3.

**lemma** *d-absorb1* [*simp*]:  $d x + (d x \cdot d y) = d x$   
**proof** (*rule order.antisym*)  
  **have**  $d x + (d x \cdot d y) \leq d x + (d x \cdot 1_\sigma)$   
  **by** (*metis d-sub-id-ax c2-d d-def join.sup.bounded-iff join.sup.semilattice-axioms join.sup-ge1 s-prod-isol semilattice.idem*)  
  **thus**  $d x + (d x \cdot d y) \leq d x$   
  **by** *simp*  
  **show**  $d x \leq d x + ((d x) \cdot (d y))$   
  **using** *join.sup-ge1* **by** *blast*  
**qed**

**lemma** *d-absorb2* [*simp*]:  $d x \cdot (d x + d y) = d x$   
**proof** –  
  **have**  $x \cdot 1_\pi \parallel d x = d x$   
  **by** (*metis local.c1 local.dc-prop1*)  
  **thus** *?thesis*  
  **by** (*metis d-absorb1 local.c2-d local.p-prod-distl*)  
**qed**

**lemma** *d-dist1*:  $d x \cdot (d y + d z) = d x \cdot d y + d x \cdot d z$   
**by** (*simp add: local.c2-d local.p-prod-distl*)

**lemma** *d-dist2*:  $d x + (d y \cdot d z) = (d x + d y) \cdot (d x + d z)$   
**proof** –  
  **have**  $(d x + d y) \cdot (d x + d z) = d x \cdot d x + d x \cdot d z + d y \cdot d x + d y \cdot d z$   
  **using** *add-assoc d-dist1 local.s-prod-distr* **by** *force*  
  **also have**  $\dots = d x + d x \cdot d z + d x \cdot d y + d y \cdot d z$   
  **using** *local.d-comm-ax* **by** *auto*  
  **finally show** *?thesis*  
  **by** *simp*  
**qed**

**lemma** *d-add-prod-closed* [*simp*]:  $d (d x + d y) = d x + d y$   
**by** (*simp add: d-add-ax*)

The following properties are not covered in the article.

**lemma** *x-zero-prop*:  $(x \cdot 0) \parallel y = d(x \cdot 0) \cdot y$   
**by** (*simp add: local.c2-d*)

**lemma** *cda-add-ax*:  $d((x + y) \cdot z) = d(x \cdot z) + d(y \cdot z)$   
**by** (*simp add: d-add-ax local.s-prod-distr*)

**lemma** *d-x-zero*:  $d(x \cdot 0) = (x \cdot 0) \parallel 1_\sigma$   
**by** (*simp add: x-zero-prop*)

Lemma 5.4 is verified below because its proofs are simplified by using facts from the next subsection.

### 2.3 Results for Concurrent Dynamic Algebra

The following proofs and refutation are related to Section 6 in [2]. We do not consider those involving Kleene algebras in this section. We also do not introduce specific notation for diamond operators.

First we prove Lemma 6.1. Part (1) and (3) have already been verified above. Part (2) and (4) require additional assumptions which are present in the context of concurrent dynamic algebra [1]. We also present the counterexamples from Lemma 6.3.

**lemma**  $(x \cdot y) \cdot d z = x \cdot (y \cdot d z)$   
**nitpick**  
**oops**

**lemma**  $d((x \cdot y) \cdot z) = d(x \cdot d(y \cdot z))$   
**nitpick**  
**oops**

**lemma** *cda-ax1*:  $(x \cdot y) \cdot d z = x \cdot (y \cdot d z) \implies d((x \cdot y) \cdot z) = d(x \cdot d(y \cdot z))$   
**by** (*metis local.d-loc-ax*)

**lemma** *d-inter*:  $(x \parallel y) \cdot d z = (x \cdot d z) \parallel (y \cdot d z)$   
**nitpick**  
**oops**

**lemma**  $d((x \parallel y) \cdot z) = d(x \cdot z) \cdot d(y \cdot z)$   
**nitpick**  
**oops**

**lemma** *cda-ax2*:  
**assumes**  $(x \parallel y) \cdot d z = (x \cdot d z) \parallel (y \cdot d z)$   
**shows**  $d((x \parallel y) \cdot z) = d(x \cdot z) \cdot d(y \cdot z)$   
**by** (*metis assms local.d-conc6 local.d-conc-s-prod-ax local.d-loc-ax*)

Next we present some results that do not feature in the article.

**lemma**  $(x \cdot y) \cdot 0 = x \cdot (y \cdot 0)$

**nitpick**  
**oops**

**lemma** *d-x-zero-prop* [*simp*]:  $d (x \cdot 0) \cdot 1_\pi = x \cdot 0$   
**by** *simp*

**lemma**  $x \leq 1_\sigma \wedge y \leq 1_\sigma \longrightarrow x \cdot y = x \parallel y$   
**nitpick**  
**oops**

**lemma**  $x \cdot (y \parallel z) \leq (x \cdot y) \parallel (x \cdot z)$   
**nitpick**  
**oops**

**lemma**  $x \leq x \parallel x$   
**nitpick**  
**oops**

Lemma 5.4

**lemma** *d-lb1*:  $d x \cdot d y \leq d x$   
**by** (*simp add: less-eq-def add-commute*)

**lemma** *d-lb2*:  $d x \cdot d y \leq d y$   
**using** *d-lb1 local.d-comm-ax* **by** *fastforce*

**lemma** *d-glb*:  $d z \leq d x \wedge d z \leq d y \implies d z \leq d x \cdot d y$   
**by** (*simp add: d-dist2 local.less-eq-def*)

**lemma** *d-glb-iff*:  $d z \leq d x \wedge d z \leq d y \iff d z \leq d x \cdot d y$   
**using** *d-glb d-lb1 d-lb2 local.order-trans* **by** *blast*

**lemma** *x-zero-le-c*:  $x \cdot 0 \leq 1_\pi$   
**by** (*simp add: p-id-term*)

**lemma** *p-subid-lb1*:  $(x \cdot 0) \parallel (y \cdot 0) \leq x \cdot 0$   
**using** *local.mult-isol x-zero-le-c* **by** *fastforce*

**lemma** *p-subid-lb2*:  $(x \cdot 0) \parallel (y \cdot 0) \leq y \cdot 0$   
**using** *local.mult-commute p-subid-lb1* **by** *fastforce*

**lemma** *p-subid-idem* [*simp*]:  $(x \cdot 0) \parallel (x \cdot 0) = x \cdot 0$   
**by** (*metis local.c1 zero-p-id-prop*)

**lemma** *p-subid-glb*:  $z \cdot 0 \leq x \cdot 0 \wedge z \cdot 0 \leq y \cdot 0 \implies z \cdot 0 \leq (x \cdot 0) \parallel (y \cdot 0)$   
**using** *local.mult-isol-var* **by** *force*

**lemma** *p-subid-glb-iff*:  $z \cdot 0 \leq x \cdot 0 \wedge z \cdot 0 \leq y \cdot 0 \iff z \cdot 0 \leq (x \cdot 0) \parallel (y \cdot 0)$   
**using** *local.order-trans p-subid-glb p-subid-lb1 p-subid-lb2* **by** *blast*

```

lemma x-c-glb:  $z \cdot 1_\pi \leq x \cdot 1_\pi \wedge z \cdot 1_\pi \leq y \cdot 1_\pi \implies z \cdot 1_\pi \leq (x \cdot 1_\pi) \parallel (y \cdot 1_\pi)$ 
  using local.mult-isol-var by force

lemma x-c-lb1:  $(x \cdot 1_\pi) \parallel (y \cdot 1_\pi) \leq x \cdot 1_\pi$ 
  using local.c6 local.mult-isol-var by force

lemma x-c-lb2:  $(x \cdot 1_\pi) \parallel (y \cdot 1_\pi) \leq y \cdot 1_\pi$ 
  using local.mult-commute x-c-lb1 by fastforce

lemma x-c-glb-iff:  $z \cdot 1_\pi \leq x \cdot 1_\pi \wedge z \cdot 1_\pi \leq y \cdot 1_\pi \iff z \cdot 1_\pi \leq (x \cdot 1_\pi) \parallel (y \cdot 1_\pi)$ 
  by (meson local.order.trans x-c-glb x-c-lb1 x-c-lb2)

end

```

## 2.4 C-Lattices

We can now define c-lattices and prove the results from Section 7 in [2].

```

class pbl-monoid = proto-trioid +
  fixes U :: 'a
  fixes meet :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\sqcap$  70)
  assumes U-def:  $x \leq U$ 
  and meet-assoc:  $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$ 
  and meet-comm:  $x \sqcap y = y \sqcap x$ 
  and meet-idem [simp]:  $x \sqcap x = x$ 
  and absorp1:  $x \sqcap (x + y) = x$ 
  and absorp2:  $x + (x \sqcap y) = x$ 

begin

sublocale lattice ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) ( $+$ )
proof
  show a:  $\bigwedge x y. x \sqcap y \leq x$ 
    by (simp add: local.absorp2 local.less-eq-def add-commute)
  show b:  $\bigwedge x y. x \sqcap y \leq y$ 
    using a local.meet-comm by fastforce
  show  $\bigwedge x y z. x \leq y \implies x \leq z \implies x \leq y \sqcap z$ 
    by (metis b local.absorp1 local.less-eq-def local.meet-assoc)
qed

lemma meet-glb:  $z \leq x \wedge z \leq y \implies z \leq x \sqcap y$ 
  by simp

lemma meet-prop:  $z \leq x \wedge z \leq y \iff z \leq x \sqcap y$ 
  by simp

end

```

```

class pddl-monoid = pbl-monoid +
  assumes lat-dist1:  $x + (y \sqcap z) = (x + y) \sqcap (x + z)$ 

begin

lemma lat-dist2:  $(x \sqcap y) + z = (x + z) \sqcap (y + z)$ 
  by (simp add: local.lat-dist1 add-commute)

lemma lat-dist3:  $x \sqcap (y + z) = (x \sqcap y) + (x \sqcap z)$ 
proof -
  have  $\bigwedge x y z. x \sqcap ((x + y) \sqcap z) = x \sqcap z$ 
    by (metis local.absorp1 local.meet-assoc)
  thus ?thesis
    using lat-dist2 local.absorp2 add-commute by force
qed

lemma lat-dist4:  $(x + y) \sqcap z = (x \sqcap z) + (y \sqcap z)$ 
  using lat-dist3 local.meet-comm by auto

lemma d-equiv-prop:  $(\forall z. z + x = z + y \wedge z \sqcap x = z \sqcap y) \implies x = y$ 
  by (metis local.add-zero1)

end

```

The symbol  $\bar{1}_\pi$  from [2] is written `nc` in this theory file.

```

class c-lattice = pddl-monoid +
  fixes nc :: 'a
  assumes cl1 [simp]:  $x \cdot 1_\pi + x \cdot nc = x \cdot U$ 
  and cl2 [simp]:  $1_\pi \sqcap (x + nc) = x \cdot 0$ 
  and cl3:  $x \cdot (y \parallel z) \leq (x \cdot y) \parallel (x \cdot z)$ 
  and cl4:  $z \parallel z \leq z \implies (x \parallel y) \cdot z = (x \cdot z) \parallel (y \cdot z)$ 
  and cl5:  $x \cdot (y \cdot (z \cdot 0)) = (x \cdot y) \cdot (z \cdot 0)$ 
  and cl6 [simp]:  $(x \cdot 0) \cdot z = x \cdot 0$ 
  and cl7 [simp]:  $1_\sigma \parallel 1_\sigma = 1_\sigma$ 
  and cl8 [simp]:  $((x \cdot 1_\pi) \parallel 1_\sigma) \cdot y = (x \cdot 1_\pi) \parallel y$ 
  and cl9 [simp]:  $((x \sqcap 1_\sigma) \cdot 1_\pi) \parallel 1_\sigma = x \sqcap 1_\sigma$ 
  and cl10:  $((x \sqcap nc) \cdot 1_\pi) \parallel 1_\sigma = 1_\sigma \sqcap (x \sqcap nc) \cdot nc$ 
  and cl11 [simp]:  $((x \sqcap nc) \cdot 1_\pi) \parallel nc = (x \sqcap nc) \cdot nc$ 

```

begin

We show that every c-lattice is a c-trioid (Proposition 7.1) Proposition 7.2 is again covered by the theory for multirelations.

```

subclass c-trioid
proof
  fix x y
  show  $x \cdot 1_\pi \parallel 1_\sigma \cdot y = x \cdot 1_\pi \parallel y$ 
    by auto
  show  $x \parallel y \cdot 1_\pi = x \cdot 1_\pi \parallel (y \cdot 1_\pi)$ 

```

```

    by (simp add: local.cl4)
  show  $x \cdot y \cdot 1_\pi = x \cdot (y \cdot 1_\pi)$ 
    by (metis local.absorp1 local.cl2 local.cl5)
  show  $1_\sigma \parallel 1_\sigma = 1_\sigma$ 
    by (meson local.cl7)
  show  $x \cdot 1_\pi \leq 1_\pi$ 
    by (metis local.absorp1 local.cl2 local.cl5 local.inf-le1 local.s-prod-idl)
  show  $x \cdot 1_\pi \parallel x = x$ 
    by (metis x order.eq-iff local.cl3 local.mult-1-right local.mult-commute local.mult-isol
local.s-prod-idr)
qed

```

First we verify the complementation conditions after the definition of c-lattices.

```

lemma c-nc-comp1 [simp]:  $1_\pi + nc = U$ 
  by (metis local.cl1 local.s-prod-idl)

```

```

lemma c-nc-comp2 [simp]:  $1_\pi \sqcap nc = 0$ 
  by (metis local.add-zero-l local.cl2 local.s-prod-annil)

```

```

lemma c-0:  $x \sqcap 1_\pi = x \cdot 0$ 
  by (metis c-nc-comp2 local.add-zero-l local.cl2 local.lat-dist3 local.meet-comm)

```

Next we verify the conditions in Proposition 7.2.

```

lemma d-s-subid:  $d x = x \iff x \leq 1_\sigma$ 
  by (metis local.cl9 local.d-def local.d-subid local.inf.absorb-iff1)

```

```

lemma term-p-subid:  $x \cdot 1_\pi = x \iff x \leq 1_\pi$ 
  by (metis c-0 local.cl6 local.inf.absorb-iff1 local.p-id-term)

```

```

lemma term-p-subid-var:  $x \cdot 0 = x \iff x \leq 1_\pi$ 
  using c-0 local.inf.absorb-iff1 by auto

```

```

lemma vec-iff:  $d x \cdot U = x \iff (x \cdot 1_\pi) \parallel U = x$ 
  by (simp add: local.c2-d)

```

```

lemma nc-iff1:  $x \leq nc \iff x \sqcap 1_\pi = 0$ 

```

**proof**

```

  fix x

```

```

  assume assm:  $x \leq nc$ 

```

```

  hence  $x = x \sqcap nc$ 

```

```

    by (simp add: local.inf.absorb-iff1)

```

```

  hence  $x \sqcap 1_\pi = x \sqcap nc \sqcap 1_\pi$ 

```

```

    by auto

```

```

  then show  $x \sqcap 1_\pi = 0$ 

```

```

    by (metis assm c-0 c-nc-comp2 local.cl2 local.less-eq-def)

```

**next**

```

  fix x

```

```

  assume assm:  $x \sqcap 1_\pi = 0$ 

```

**have**  $x = (x \sqcap nc) + (x \sqcap 1_\pi)$   
**by** (*metis c-nc-comp1 local.U-def local.add-comm local.lat-dist3 local.inf.absorb-iff1*)  
**hence**  $x = x \sqcap nc$   
**using** *assm* **by** *auto*  
**thus**  $x \leq nc$   
**using** *local.inf.absorb-iff1* **by** *auto*  
**qed**

**lemma** *nc-iff2*:  $x \leq nc \longleftrightarrow x \cdot 0 = 0$   
**using** *c-0 nc-iff1* **by** *auto*

The results of Lemma 7.3 are again at the multirelational level. Hence we continue with Lemma 7.4.

**lemma** *assoc-p-subid*:  $(x \cdot y) \cdot (z \cdot 1_\pi) = x \cdot (y \cdot (z \cdot 1_\pi))$   
**by** (*metis c-0 local.c6 local.cl5 local.inf.absorb-iff1*)

**lemma** *zero-assoc3*:  $(x \cdot y) \cdot 0 = x \cdot (y \cdot 0)$   
**by** (*metis local.cl5 local.s-prod-annil*)

**lemma** *x-zero-interr*:  $(x \cdot 0) \parallel (y \cdot 0) = (x \parallel y) \cdot 0$   
**by** (*simp add: local.cl4*)

**lemma** *p-subid-interr*:  $(x \cdot z \cdot 1_\pi) \parallel (y \cdot z \cdot 1_\pi) = (x \parallel y) \cdot z \cdot 1_\pi$   
**by** (*simp add: local.c4 local.cl4*)

**lemma** *d-interr*:  $(x \cdot d z) \parallel (y \cdot d z) = (x \parallel y) \cdot d z$   
**by** (*simp add: local.cl4*)

**lemma** *subidem-par*:  $x \leq x \parallel x$

**proof** –  
**have**  $x = x \cdot 1_\sigma$   
**by** *auto*  
**also have**  $\dots = x \cdot (1_\sigma \parallel 1_\sigma)$   
**by** *auto*  
**finally show** *?thesis*  
**by** (*metis local.cl3 local.cl7*)  
**qed**

**lemma** *meet-le-par*:  $x \sqcap y \leq x \parallel y$

**proof** –  
**have**  $x \sqcap y = (x \sqcap y) \sqcap (x \sqcap y)$   
**using** *local.meet-idem* **by** *presburger*  
**thus** *?thesis*  
**using** *local.inf-le1 local.inf-le2 local.mult-isol-var local.order-trans subidem-par*  
**by** *blast*  
**qed**

Next we verify Lemma 7.5 and prove some related properties.

**lemma** *x-split [simp]*:  $(x \sqcap nc) + (x \sqcap 1_\pi) = x$

**proof** –  
**have**  $x = x \sqcap U$   
**using** *local.U-def local.inf.absorb-iff1* **by** *auto*  
**also have**  $\dots = x \sqcap (nc + 1_\pi)$   
**by** (*simp add: add-commute*)  
**finally show** *?thesis*  
**by** (*metis local.lat-dist3*)  
**qed**

**lemma** *x-split-var [simp]: (x \sqcap nc) + (x \cdot 0) = x*  
**by** (*metis local.c-0 x-split*)

**lemma** *s-subid-closed [simp]: x \sqcap nc \sqcap 1\_\sigma = x \sqcap 1\_\sigma*

**proof** –  
**have**  $x \sqcap 1_\sigma = ((x \sqcap nc) + (x \sqcap 1_\pi)) \sqcap 1_\sigma$   
**using** *x-split* **by** *presburger*  
**also have**  $\dots = (x \sqcap nc \sqcap 1_\sigma) + (x \sqcap 1_\pi \sqcap 1_\sigma)$   
**by** (*simp add: local.lat-dist3 local.meet-comm*)  
**also have**  $\dots = (x \sqcap nc \sqcap 1_\sigma) + (x \sqcap 0)$   
**by** (*metis c-0 local.meet-assoc local.meet-comm local.s-prod-idl*)  
**finally show** *?thesis*  
**by** (*metis local.absorp1 local.add-zeror local.lat-dist1 local.meet-comm*)  
**qed**

**lemma** *sub-id-le-nc: x \sqcap 1\_\sigma \leq nc*  
**by** (*metis local.inf.absorb-iff2 local.inf-left-commute local.meet-comm s-subid-closed*)

**lemma** *s-x-c [simp]: 1\_\sigma \sqcap (x \cdot 1\_\pi) = 0*

**proof** –  
**have**  $1_\sigma \sqcap 1_\pi = 0$   
**using** *c-0 local.s-prod-idl* **by** *presburger*  
**hence**  $1_\sigma \sqcap x \cdot 1_\pi \leq 0$   
**using** *local.c6 local.inf-le1 local.inf-le2 local.meet-prop local.order.trans* **by** *blast*  
**thus** *?thesis*  
**using** *local.less-eq-def local.no-trivial-inverse* **by** *blast*  
**qed**

**lemma** *s-x-zero [simp]: 1\_\sigma \sqcap (x \cdot 0) = 0*  
**by** (*metis local.cl6 s-x-c*)

**lemma** *c-nc [simp]: (x \cdot 1\_\pi) \sqcap nc = 0*

**proof** –  
**have**  $x \cdot 1_\pi \sqcap nc \leq 1_\pi$   
**by** (*meson local.c6 local.dual-order.trans local.inf-le1*)  
**thus** *?thesis*  
**by** (*metis local.inf-le2 nc-iff2 term-p-subid-var*)  
**qed**

**lemma** *zero-nc [simp]: (x \cdot 0) \sqcap nc = 0*



by (metis c-nc local.cl6)

**lemma** nc-zero [simp]:  $(x \sqcap nc) \cdot 0 = 0$   
by (meson local.inf-le2 nc-iff2)

Lemma 7.6.

**lemma** c-def [simp]:  $U \cdot 0 = 1_\pi$   
by (metis c-nc-comp1 c-0 local.absorp1 local.meet-comm)

**lemma** c-x-prop [simp]:  $1_\pi \cdot x = 1_\pi$   
using c-def local.cl6 by blast

**lemma** U-idem-s-prod [simp]:  $U \cdot U = U$   
by (metis local.U-def order.eq-iff local.s-prod-idl local.s-prod-isor)

**lemma** U-idem-p-prod [simp]:  $U \parallel U = U$   
using local.U-def order.eq-iff subidem-par by presburger

**lemma** U-c [simp]:  $U \cdot 1_\pi = 1_\pi$   
by (metis U-idem-s-prod local.c-def zero-assoc3)

**lemma** s-le-nc:  $1_\sigma \leq nc$   
by (metis local.meet-idem sub-id-le-nc)

**lemma** nc-c [simp]:  $nc \cdot 1_\pi = 1_\pi$   
**proof** (rule order.antisym)  
have  $nc \cdot 1_\pi = nc \cdot 1_\pi \cdot 0$   
by (simp add: zero-assoc3)  
**also have**  $\dots = nc \cdot 1_\pi \sqcap 1_\pi$   
by (simp add: c-0)  
**finally show**  $nc \cdot 1_\pi \leq 1_\pi$   
using local.c6 by blast  
**show**  $1_\pi \leq nc \cdot 1_\pi$   
using local.s-prod-isor s-le-nc by fastforce  
**qed**

**lemma** nc-nc [simp]:  $nc \cdot nc = nc$   
**proof** –  
have  $nc \cdot nc = (nc \cdot 1_\pi) \parallel nc$   
by (metis local.cl11 local.meet-idem)  
**thus** ?thesis  
by simp  
**qed**

**lemma** U-nc [simp]:  $U \cdot nc = U$   
**proof** –  
have  $U \cdot nc = (1_\pi + nc) \cdot nc$   
by force  
**also have**  $\dots = 1_\pi \cdot nc + nc \cdot nc$

**using** *local.s-prod-distr* **by** *blast*  
**also have**  $\dots = 1_\pi + nc$   
**by** *simp*  
**finally show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *nc-U* [*simp*]:  $nc \cdot U = U$   
**proof** –  
**have**  $nc \cdot U = nc \cdot 1_\pi + nc \cdot nc$   
**using** *local.cl1* **by** *presburger*  
**thus** *?thesis*  
**by** *simp*  
**qed**

**lemma** *nc-nc-par* [*simp*]:  $nc \parallel nc = nc$   
**proof** –  
**have**  $nc \parallel nc = (nc \parallel nc \sqcap nc) + (nc \parallel nc) \cdot 0$   
**by** *simp*  
**also have**  $\dots = nc + (nc \cdot 0) \parallel (nc \cdot 0)$   
**by** (*metis local.meet-comm local.inf.absorb-iff1 subidem-par x-zero-interr*)  
**also have**  $\dots = nc + 0 \parallel 0$   
**by** (*metis local.absorp1 local.meet-comm nc-zero*)  
**finally show** *?thesis*  
**by** (*metis add-commute local.add-zero local.annil*)  
**qed**

**lemma** *U-nc-par* [*simp*]:  $U \parallel nc = nc$   
**proof** –  
**have**  $U \parallel nc = nc \parallel nc + 1_\pi \parallel nc$   
**by** (*metis c-nc-comp1 local.add-comm local.distrib-right*)  
**also have**  $\dots = nc + nc$   
**by** *force*  
**finally show** *?thesis*  
**by** *simp*  
**qed**

We prove Lemma 7.8 and related properties.

**lemma** *x-y-split* [*simp*]:  $(x \sqcap nc) \cdot y + x \cdot 0 = x \cdot y$   
**by** (*metis c-0 local.cl6 local.s-prod-distr x-split*)

**lemma** *x-y-prop*:  $1_\sigma \sqcap (x \sqcap nc) \cdot y = 1_\sigma \sqcap x \cdot y$   
**proof** –  
**have**  $1_\sigma \sqcap x \cdot y = 1_\sigma \sqcap ((x \sqcap nc) \cdot y + x \cdot 0)$   
**using** *x-y-split* **by** *presburger*  
**also have**  $\dots = (1_\sigma \sqcap (x \sqcap nc) \cdot y) + (1_\sigma \sqcap x \cdot 0)$   
**by** (*simp add: local.lat-dist3 add-commute*)  
**finally show** *?thesis*  
**by** (*metis local.add-zero s-x-zero*)

qed

**lemma** *s-nc-U*:  $1_\sigma \sqcap x \cdot nc = 1_\sigma \sqcap x \cdot U$

**proof** –

**have**  $1_\sigma \sqcap x \cdot U = 1_\sigma \sqcap (x \cdot nc + x \cdot 1_\pi)$

**by** (*simp add: add-commute*)

**also have**  $\dots = (1_\sigma \sqcap x \cdot nc) + (1_\sigma \sqcap x \cdot 1_\pi)$

**using** *local.lat-dist3* **by** *blast*

**finally show** *?thesis*

**by** (*metis local.add-zero s-x-c*)

qed

**lemma** *sid-le-nc-var*:  $1_\sigma \sqcap x \leq 1_\sigma \sqcap x \parallel nc$

**proof** –

**have**  $1_\sigma \sqcap x = x \sqcap (1_\sigma \sqcap nc)$

**by** (*metis (no-types) local.inf.absorb1 local.inf.commute s-le-nc*)

**hence**  $1_\sigma \sqcap x \parallel nc + 1_\sigma \sqcap x = (x \parallel nc + x \sqcap nc) \sqcap 1_\sigma$

**using** *local.inf.commute local.inf.left-commute local.lat-dist4* **by** *auto*

**thus** *?thesis*

**by** (*metis (no-types) local.inf.commute local.join.sup.absorb-iff1 meet-le-par*)

qed

**lemma** *s-nc-par-U*:  $1_\sigma \sqcap x \parallel nc = 1_\sigma \sqcap x \parallel U$

**proof** –

**have**  $1_\sigma \sqcap x \parallel U = 1_\sigma \sqcap (x \parallel nc + x)$

**by** (*metis c-nc-comp1 local.add-comm local.distrib-left local.mult-oner*)

**also have**  $\dots = (1_\sigma \sqcap x \parallel nc) + (x \sqcap 1_\sigma)$

**by** (*metis local.lat-dist3 local.meet-comm*)

**also have**  $\dots = 1_\sigma \sqcap x \parallel nc$

**by** (*metis local.add-comm local.less-eq-def local.meet-comm sid-le-nc-var*)

**finally show** *?thesis*

**by** *metis*

qed

**lemma** *x-c-nc-split*:  $(x \cdot 1_\pi) \parallel nc = (x \sqcap nc) \cdot nc + (x \cdot 0) \parallel nc$

**by** (*metis local.cl11 local.mult-commute local.p-prod-dist1 x-y-split*)

**lemma** *x-c-U-split*:  $(x \cdot 1_\pi) \parallel U = x \cdot U + (x \cdot 0) \parallel U$

**proof** –

**have**  $x \cdot U + (x \cdot 0) \parallel U = (x \sqcap nc) \cdot U + (x \cdot 0) \parallel U$

**by** (*metis U-c U-idem-s-prod U-nc local.add-assoc' local.cl1 local.distrib-left local.mult-oner x-y-split*)

**also have**  $\dots = (x \sqcap nc) \cdot nc + (x \sqcap nc) \cdot 1_\pi + (x \cdot 0) \parallel nc + x \cdot 0$

**by** (*metis add-commute c-nc-comp1 local.cl1 local.combine-common-factor local.mult-1-right local.mult-commute*)

**also have**  $\dots = (x \cdot 1_\pi) \parallel nc + x \cdot 1_\pi$

**by** (*metis local.add-ac(1) local.add-commute x-c-nc-split x-y-split*)

**thus** *?thesis*

**by** (*metis c-nc-comp1 calculation local.add-comm local.distrib-left local.mult-oner*)

qed

## 2.5 Domain in C-Lattices

We now prove variants of the domain axioms and verify the properties of Section 8 in [2].

**lemma** *cl9-d* [simp]:  $d(x \sqcap 1_\sigma) = x \sqcap 1_\sigma$   
by (simp add: local.d-def)

**lemma** *cl10-d*:  $d(x \sqcap nc) = 1_\sigma \sqcap (x \sqcap nc) \cdot nc$   
using local.cl10 local.d-def by auto

**lemma** *cl11-d* [simp]:  $d(x \sqcap nc) \cdot nc = (x \sqcap nc) \cdot nc$   
using local.c2-d by force

**lemma** *cl10-d-var1*:  $d(x \sqcap nc) = 1_\sigma \sqcap x \cdot nc$   
by (simp add: cl10-d x-y-prop)

**lemma** *cl10-d-var2*:  $d(x \sqcap nc) = 1_\sigma \sqcap (x \sqcap nc) \cdot U$   
by (simp add: cl10-d s-nc-U)

**lemma** *cl10-d-var3*:  $d(x \sqcap nc) = 1_\sigma \sqcap x \cdot U$   
by (simp add: cl10-d-var1 s-nc-U)

We verify the remaining properties of Lemma 8.1.

**lemma** *d-U* [simp]:  $d U = 1_\sigma$   
by (simp add: local.d-def)

**lemma** *d-nc* [simp]:  $d nc = 1_\sigma$   
using local.d-def by auto

**lemma** *alt-d-def-nc-nc*:  $d(x \sqcap nc) = 1_\sigma \sqcap ((x \sqcap nc) \cdot 1_\pi) \parallel nc$   
by (simp add: cl10-d-var1 x-y-prop)

**lemma** *alt-d-def-nc-U*:  $d(x \sqcap nc) = 1_\sigma \sqcap ((x \sqcap nc) \cdot 1_\pi) \parallel U$   
by (metis alt-d-def-nc-nc local.c2-d s-nc-U)

We verify the identity before Lemma 8.2 of [2] together with variants.

**lemma** *d-def-split* [simp]:  $d(x \sqcap nc) + d(x \cdot 0) = d x$   
by (metis local.d-add-ax x-split-var)

**lemma** *d-def-split-var* [simp]:  $d(x \sqcap nc) + (x \cdot 0) \parallel 1_\sigma = d x$   
by (metis d-def-split local.d-x-zero)

**lemma** *ax7* [simp]:  $(1_\sigma \sqcap x \cdot U) + (x \cdot 0) \parallel 1_\sigma = d x$   
by (metis cl10-d-var3 d-def-split-var)

Lemma 8.2.

**lemma** *dom12-d*:  $d x = 1_\sigma \sqcap (x \cdot 1_\pi) \parallel nc$   
**proof** –  
  **have**  $1_\sigma \sqcap (x \cdot 1_\pi) \parallel nc = 1_\sigma \sqcap ((x \sqcap nc) \cdot 1_\pi + x \cdot 0) \parallel nc$   
  **using** *x-y-split* **by** *presburger*  
  **also have**  $\dots = (1_\sigma \sqcap ((x \sqcap nc) \cdot 1_\pi) \parallel nc) + (1_\sigma \sqcap (x \cdot 0) \parallel nc)$   
  **by** (*simp add: local.lat-dist3 local.mult-commute local.p-prod-distl add-commute*)  
  **also have**  $\dots = d (x \sqcap nc) + d (x \cdot 0)$   
  **by** (*metis add-commute c-0 cl10-d-var1 local.add-zero1 local.annil local.c2-d local.d-def local.mult-commute local.mult-onel local.zero-p-id-prop x-split*)  
  **finally show** *?thesis*  
  **by** (*metis d-def-split*)  
**qed**

**lemma** *dom12-d-U*:  $d x = 1_\sigma \sqcap (x \cdot 1_\pi) \parallel U$   
**by** (*simp add: dom12-d s-nc-par-U*)

**lemma** *dom-def-var*:  $d x = (x \cdot U \sqcap 1_\pi) \parallel 1_\sigma$   
**by** (*simp add: c-0 local.d-def zero-assoc3*)

Lemma 8.3.

**lemma** *ax5-d* [*simp*]:  $d (x \sqcap nc) \cdot U = (x \sqcap nc) \cdot U$   
**proof** –  
  **have**  $d (x \sqcap nc) \cdot U = d (x \sqcap nc) \cdot nc + d (x \sqcap nc) \cdot 1_\pi$   
  **using** *add-commute local.cl1* **by** *presburger*  
  **also have**  $\dots = (x \sqcap nc) \cdot nc + (x \sqcap nc) \cdot 1_\pi$   
  **by** *simp*  
  **finally show** *?thesis*  
  **by** (*simp add: add-commute*)  
**qed**

**lemma** *ax5-0* [*simp*]:  $d (x \cdot 0) \cdot U = (x \cdot 0) \parallel U$   
**using** *local.x-zero-prop* **by** *presburger*

**lemma** *x-c-U-split2*:  $d x \cdot nc = (x \sqcap nc) \cdot nc + (x \cdot 0) \parallel nc$   
**by** (*simp add: local.c2-d x-c-nc-split*)

**lemma** *x-c-U-split3*:  $d x \cdot U = (x \sqcap nc) \cdot U + (x \cdot 0) \parallel U$   
**by** (*metis d-def-split local.s-prod-distr ax5-0 ax5-d*)

**lemma** *x-c-U-split-d*:  $d x \cdot U = x \cdot U + (x \cdot 0) \parallel U$   
**using** *local.c2-d x-c-U-split* **by** *presburger*

**lemma** *x-U-prop2*:  $x \cdot nc = d (x \sqcap nc) \cdot nc + x \cdot 0$   
**by** (*metis local.c2-d local.cl11 x-y-split*)

**lemma** *x-U-prop3*:  $x \cdot U = d (x \sqcap nc) \cdot U + x \cdot 0$   
**by** (*metis ax5-d x-y-split*)

**lemma** *d-x-nc* [*simp*]:  $d (x \cdot nc) = d x$

**using** *local.c4 local.d-def* **by** *auto*

**lemma** *d-x-U [simp]*:  $d (x \cdot U) = d x$   
**by** (*simp add: local.c4 local.d-def*)

The next properties of domain are important, but do not feature in [2].  
Proofs can be found in [1].

**lemma** *d-llp1*:  $d x \leq d y \implies x \leq d y \cdot x$   
**by** (*metis local.d-rest-ax local.s-prod-isor*)

**lemma** *d-llp2*:  $x \leq d y \cdot x \implies d x \leq d y$

**proof** –

**assume** *a1*:  $x \leq d y \cdot x$

**have**  $\forall x y. d (x \parallel y) = x \cdot 1_\pi \parallel d y$

**using** *local.c2-d local.d-conc6 local.d-conc-s-prod-ax* **by** *presburger*

**hence**  $d x \leq d (y \cdot 1_\pi)$

**using** *a1* **by** (*metis (no-types) local.c2-d local.c6 local.c-prod-comm order.eq-iff local.mult-isol local.mult-oner*)

**thus** *?thesis*

**by** *simp*

**qed**

**lemma** *demod1*:  $d (x \cdot y) \leq d z \implies x \cdot d y \leq d z \cdot x$

**proof** –

**assume**  $d (x \cdot y) \leq d z$

**hence**  $\forall v. x \cdot y \cdot 1_\pi \parallel v \leq z \cdot 1_\pi \parallel v$

**by** (*metis (no-types) local.c2-d local.s-prod-isor*)

**hence**  $\forall v. x \cdot (y \cdot 1_\pi \parallel v) \leq z \cdot 1_\pi \parallel (x \cdot v)$

**by** (*metis local.c4 local.cl3 local.dual-order.trans*)

**thus** *?thesis*

**by** (*metis local.c2-d local.s-prod-idr*)

**qed**

**lemma** *demod2*:  $x \cdot d y \leq d z \cdot x \implies d (x \cdot y) \leq d z$

**proof** –

**assume**  $x \cdot d y \leq d z \cdot x$

**hence**  $d (x \cdot y) \leq d (d z \cdot x)$

**by** (*metis local.d-def local.d-loc-ax local.mult-isor local.s-prod-isor*)

**thus** *?thesis*

**using** *local.d-conc6 local.d-conc-s-prod-ax local.d-glb-iff* **by** *fastforce*

**qed**

## 2.6 Structural Properties of C-Lattices

Now we consider the results from Section 9 and 10 in [2]. First we verify the conditions for Proposition 9.1.

**lemma** *d-meet-closed [simp]*:  $d (d x \sqcap d y) = d x \sqcap d y$

**using** *d-s-subid local.d-sub-id-ax local.inf-le1 local.order-trans* **by** *blast*

**lemma** *d-s-prod-eq-meet*:  $d x \cdot d y = d x \sqcap d y$   
**apply** (*rule order.antisym*)  
**apply** (*metis local.d-lb1 local.d-lb2 local.meet-glb*)  
**by** (*metis d-meet-closed local.inf-le1 local.inf-le2 local.d-glb*)

**lemma** *d-p-prod-eq-meet*:  $d x \parallel d y = d x \sqcap d y$   
**by** (*simp add: d-s-prod-eq-meet local.d-conc-s-prod-ax*)

**lemma** *s-id-par-s-prod*:  $(x \sqcap 1_\sigma) \parallel (y \sqcap 1_\sigma) = (x \sqcap 1_\sigma) \cdot (y \sqcap 1_\sigma)$   
**by** (*metis cl9-d local.d-conc-s-prod-ax*)

**lemma** *s-id-par [simp]*:  $x \sqcap 1_\sigma \parallel x \sqcap 1_\sigma = x \sqcap 1_\sigma$   
**using** *local.meet-assoc local.meet-comm local.inf.absorb-iff1 meet-le-par* **by** *auto*

We verify the remaining conditions in Proposition 9.2.

**lemma** *p-subid-par-eq-meet*:  $(x \cdot 0) \parallel (y \cdot 0) = (x \cdot 0) \sqcap (y \cdot 0)$   
**by** (*simp add: local.meet-glb local.order.antisym local.p-subid-lb1 local.p-subid-lb2 meet-le-par*)

**lemma** *p-subid-par-eq-meet-var*:  $(x \cdot 1_\pi) \parallel (y \cdot 1_\pi) = (x \cdot 1_\pi) \sqcap (y \cdot 1_\pi)$   
**by** (*metis c-x-prop p-subid-par-eq-meet zero-assoc3*)

**lemma** *x-zero-add-closed*:  $x \cdot 0 + y \cdot 0 = (x + y) \cdot 0$   
**by** (*simp add: local.s-prod-distr*)

**lemma** *x-zero-meet-closed*:  $(x \cdot 0) \sqcap (y \cdot 0) = (x \sqcap y) \cdot 0$   
**by** (*metis c-0 local.cl6 local.meet-assoc local.meet-comm*)

The following set of lemmas investigates the closure properties of vectors, including Lemma 9,3.

**lemma** *U-par-zero [simp]*:  $(0 \cdot c) \parallel U = 0$   
**by** *fastforce*

**lemma** *U-par-s-id [simp]*:  $(1_\sigma \cdot 1_\pi) \parallel U = U$   
**by** *auto*

**lemma** *U-par-p-id [simp]*:  $(1_\pi \cdot 1_\pi) \parallel U = U$   
**by** *auto*

**lemma** *U-par-nc [simp]*:  $(nc \cdot 1_\pi) \parallel U = U$   
**by** *auto*

**lemma** *d-add-var*:  $d x \cdot z + d y \cdot z = d (x + y) \cdot z$   
**by** (*simp add: local.d-add-ax local.s-prod-distr*)

**lemma** *d-interr-U*:  $(d x \cdot U) \parallel (d y \cdot U) = d (x \parallel y) \cdot U$   
**by** (*simp add: local.cl4 local.d-conc6*)

**lemma** *d-meet*:

**assumes**  $\bigwedge x y z. (x \sqcap y \sqcap 1_\sigma) \cdot z = (x \sqcap 1_\sigma) \cdot z \sqcap (y \sqcap 1_\sigma) \cdot z$

**shows**  $d x \cdot z \sqcap d y \cdot z = (d x \sqcap d y) \cdot z$

**proof** –

**have**  $(d x \sqcap d y) \cdot z = (d x \sqcap d y \sqcap 1_\sigma) \cdot z$

**using** *local.d-sub-id-ax local.meet-assoc local.inf.absorb-iff1* **by** *fastforce*

**also have**  $\dots = (d x \sqcap 1_\sigma) \cdot z \sqcap (d y \sqcap 1_\sigma) \cdot z$

**using** *assms* **by** *auto*

**finally show** *?thesis*

**by** (*metis local.d-sub-id-ax local.inf.absorb-iff1*)

**qed**

Proposition 9.4

**lemma** *nc-zero-closed* [*simp*]:  $0 \sqcap nc = 0$

**by** (*simp add: local.inf commute local.inf-absorb2*)

**lemma** *nc-s* [*simp*]:  $1_\sigma \sqcap nc = 1_\sigma$

**using** *local.inf.absorb-iff1 s-le-nc* **by** *blast*

**lemma** *nc-add-closed*:  $(x \sqcap nc) + (y \sqcap nc) = (x + y) \sqcap nc$

**using** *local.lat-dist4* **by** *force*

**lemma** *nc-meet-closed*:  $(x \sqcap nc) \sqcap (y \sqcap nc) = x \sqcap y \sqcap nc$

**using** *local.meet-assoc local.meet-comm local.inf-le1 local.inf.absorb-iff1* **by** *fastforce*

**lemma** *nc-scomp-closed*:  $((x \sqcap nc) \cdot (y \sqcap nc)) \leq nc$

**by** (*simp add: c-0 nc-iff1 zero-assoc3*)

**lemma** *nc-scomp-closed-alt* [*simp*]:  $((x \sqcap nc) \cdot (y \sqcap nc)) \sqcap nc = (x \sqcap nc) \cdot (y \sqcap nc)$

**using** *local.inf.absorb-iff1 nc-scomp-closed* **by** *blast*

**lemma** *nc-ccomp-closed*:  $(x \sqcap nc) \parallel (y \sqcap nc) \leq nc$

**proof** –

**have**  $(x \sqcap nc) \parallel (y \sqcap nc) \leq nc \parallel nc$

**by** (*meson local.inf-le2 local.mult-isol-var*)

**thus** *?thesis*

**by** *auto*

**qed**

**lemma** *nc-ccomp-closed-alt* [*simp*]:  $(x \parallel (y \sqcap nc)) \sqcap nc = x \parallel (y \sqcap nc)$

**by** (*metis U-nc-par local.U-def local.inf-le2 local.mult-isol-var local.inf.absorb-iff1*)

Lemma 9.6.

**lemma** *tarski-prod*:

**assumes**  $\bigwedge x. x \sqcap nc \neq 0 \implies nc \cdot ((x \sqcap nc) \cdot nc) = nc$

**and**  $\bigwedge x y z. d x \cdot (y \cdot z) = (d x \cdot y) \cdot z$

**shows**  $((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc) = (if (y \sqcap nc) = 0 then 0 else (x \sqcap nc))$



$\cdot nc)$   
**proof** (*cases*  $y \sqcap nc = 0$ )  
  **fix**  $x y$   
  **assume**  $assm: y \sqcap nc = 0$   
  **show**  $(x \sqcap nc) \cdot nc \cdot ((y \sqcap nc) \cdot nc) = (if\ y \sqcap nc = 0\ then\ 0\ else\ (x \sqcap nc) \cdot nc)$   
  **by** (*metis* *assm* *c-0* *local.cl6* *local.meet-comm* *nc-zero* *zero-assoc3*)  
**next**  
  **fix**  $x y$   
  **assume**  $assm: y \sqcap nc \neq 0$   
  **have**  $((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc) = (d\ (x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)$   
  **by** *simp*  
  **also have**  $\dots = d\ (x \sqcap nc) \cdot (nc \cdot ((y \sqcap nc) \cdot nc))$   
  **by** (*simp* *add: assms(2)*)  
  **also have**  $\dots = d\ (x \sqcap nc) \cdot nc$   
  **by** (*simp* *add: assm* *assms(1)*)  
  **finally show**  $(x \sqcap nc) \cdot nc \cdot ((y \sqcap nc) \cdot nc) = (if\ y \sqcap nc = 0\ then\ 0\ else\ (x \sqcap nc) \cdot nc)$   
  **by** (*simp* *add: assm*)  
**qed**

We show the remaining conditions of Proposition 9.8.

**lemma** *nc-prod-aux* [*simp*]:  $((x \sqcap nc) \cdot nc) \cdot nc = (x \sqcap nc) \cdot nc$

**proof** –

**have**  $((x \sqcap nc) \cdot nc) \cdot nc = (d\ (x \sqcap nc) \cdot nc) \cdot nc$

**by** *simp*

**also have**  $\dots = d\ (x \sqcap nc) \cdot (nc \cdot nc)$

**by** (*metis* *cl11-d* *d-x-nc* *local.cl11* *local.meet-idem* *nc-ccomp-closed-alt* *nc-nc*)

**also have**  $\dots = d\ (x \sqcap nc) \cdot nc$

**by** *auto*

**finally show** *?thesis*

**by** *simp*

**qed**

**lemma** *nc-vec-add-closed*:  $((x \sqcap nc) \cdot nc + (y \sqcap nc) \cdot nc) \cdot nc = (x \sqcap nc) \cdot nc + (y \sqcap nc) \cdot nc$

**by** (*simp* *add: local.s-prod-distr*)

**lemma** *nc-vec-par-closed*:  $((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc) \cdot nc = ((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc)$

**by** (*simp* *add: local.cl4*)

**lemma** *nc-vec-par-is-meet*:

**assumes**  $\bigwedge x\ y\ z. (d\ x \sqcap d\ y) \cdot z = d\ x \cdot z \sqcap d\ y \cdot z$

**shows**  $((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc) = ((x \sqcap nc) \cdot nc) \sqcap ((y \sqcap nc) \cdot nc)$

**proof** –

**have**  $((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc) = (d\ (x \sqcap nc) \cdot nc) \parallel (d\ (y \sqcap nc) \cdot nc)$

**by** *auto*

**also have**  $\dots = (d\ (x \sqcap nc) \parallel d\ (y \sqcap nc)) \cdot nc$

**by** (*simp* *add: local.cl4*)

**also have**  $\dots = (d (x \sqcap nc) \sqcap d (y \sqcap nc)) \cdot nc$   
**by** (*simp add: d-p-prod-eq-meet*)  
**finally show** *?thesis*  
**by** (*simp add: assms*)  
**qed**

**lemma** *nc-vec-meet-closed*:  
**assumes**  $\bigwedge x y z. (d x \sqcap d y) \cdot z = d x \cdot z \sqcap d y \cdot z$   
**shows**  $((x \sqcap nc) \cdot nc \sqcap (y \sqcap nc) \cdot nc) \cdot nc = (x \sqcap nc) \cdot nc \sqcap (y \sqcap nc) \cdot nc$   
**proof** –  
**have**  $((x \sqcap nc) \cdot nc \sqcap (y \sqcap nc) \cdot nc) \cdot nc = (((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc)) \cdot nc$   
**by** (*simp add: assms nc-vec-par-is-meet*)  
**also have**  $\dots = ((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc)$   
**by** (*simp add: nc-vec-par-closed*)  
**finally show** *?thesis*  
**by** (*simp add: assms nc-vec-par-is-meet*)  
**qed**

**lemma** *nc-vec-seq-closed*:  
**assumes**  $\bigwedge x. x \sqcap nc \neq 0 \implies nc \cdot ((x \sqcap nc) \cdot nc) = nc$   
**and**  $\bigwedge x y z. d x \cdot (y \cdot z) = (d x \cdot y) \cdot z$   
**shows**  $((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc) \cdot nc = ((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)$   
**proof** –  
**have**  $one : y \sqcap nc = 0 \implies (((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)) \cdot nc = ((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)$   
**by** *simp*  
**have**  $y \sqcap nc \neq 0 \implies (((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)) \cdot nc = ((x \sqcap nc) \cdot nc) \cdot ((y \sqcap nc) \cdot nc)$   
**by** (*simp add: assms(1) assms(2) tarski-prod*)  
**thus** *?thesis*  
**using one by blast**  
**qed**

Proposition 10.1 and 10.2.

**lemma** *iso3* [*simp*]:  $d (d x \cdot U) = d x$   
**by** *simp*

**lemma** *iso4* [*simp*]:  $d ((x \cdot 1_\pi) \parallel U) \cdot U = (x \cdot 1_\pi) \parallel U$   
**by** (*simp add: local.c3 local.c4 vec-iff*)

**lemma** *iso5* [*simp*]:  $((x \cdot 1_\pi) \parallel U) \cdot 1_\pi = x \cdot 1_\pi$   
**by** (*simp add: local.c3 local.c4*)

**lemma** *iso6* [*simp*]:  $((x \cdot 1_\pi) \parallel U) \cdot 1_\pi \parallel U = (x \cdot 1_\pi) \parallel U$   
**by** *simp*

**lemma** *iso3-sharp* [*simp*]:  $d (d (x \sqcap nc) \cdot nc) = d (x \sqcap nc)$   
**using** *d-s-subid local.c4 local.d-def local.inf-le1* **by** *auto*

**lemma** *iso4-sharp* [*simp*]:  $d((x \sqcap nc) \cdot nc) \cdot nc = (x \sqcap nc) \cdot nc$   
**by** (*simp add: local.c2-d local.c4*)

**lemma** *iso5-sharp* [*simp*]:  $((x \sqcap nc) \cdot 1_\pi) \parallel nc \cdot 1_\pi = (x \sqcap nc) \cdot 1_\pi$   
**by** (*simp add: local.c3 local.c4*)

**lemma** *iso6-sharp* [*simp*]:  $((x \sqcap nc) \cdot nc) \cdot 1_\pi \parallel nc = (x \sqcap nc) \cdot nc$   
**using** *local.c4 local.cl11 nc-c* **by** *presburger*

We verify Lemma 15.2 at this point, because it is helpful for the following proofs.

**lemma** *uc-par-meet*:  $x \parallel U \sqcap y \parallel U = x \parallel U \parallel y \parallel U$   
**apply** (*rule order.antisym*)  
**apply** (*metis local.c-prod-assoc meet-le-par*)  
**by** (*metis U-idem-p-prod local.U-def local.c-prod-assoc local.meet-prop local.mult.left-commute local.mult-double-iso*)

**lemma** *uc-unc* [*simp*]:  $x \parallel U \parallel x \parallel U = x \parallel U$   
**by** (*metis local.meet-idem uc-par-meet*)

**lemma** *uc-interr*:  $(x \parallel y) \cdot (z \parallel U) = (x \cdot (z \parallel U)) \parallel (y \cdot (z \parallel U))$

**proof** –

**have**  $(z \parallel U) \parallel (z \parallel U) = z \parallel U$   
**by** (*metis local.c-prod-assoc uc-unc*)  
**thus** *?thesis*  
**by** (*simp add: local.cl4*)

**qed**

We verify the remaining cases of Proposition 10.3.

**lemma** *sc-hom-meet*:  $(d x \sqcap d y) \cdot 1_\pi = (d x) \cdot 1_\pi \sqcap (d y) \cdot 1_\pi$   
**by** (*metis d-p-prod-eq-meet local.c3 p-subid-par-eq-meet-var*)

**lemma** *sc-hom-seq*:  $(d x \cdot d y) \cdot 1_\pi = (d x \sqcap d y) \cdot 1_\pi$   
**by** (*simp add: d-s-prod-eq-meet*)

**lemma** *cs-hom-meet*:  $d(x \cdot 1_\pi \sqcap y \cdot 1_\pi) = d(x \cdot 1_\pi) \sqcap d(y \cdot 1_\pi)$   
**by** (*metis d-p-prod-eq-meet local.d-conc6 p-subid-par-eq-meet-var*)

**lemma** *sv-hom-meet*:  $(d x \sqcap d y) \cdot U = (d x) \cdot U \sqcap (d y) \cdot U$

**proof** –

**have**  $(d x \sqcap d y) \cdot U = ((d x) \cdot U) \parallel ((d y) \cdot U)$   
**by** (*simp add: d-interr-U d-p-prod-eq-meet local.d-conc6*)  
**thus** *?thesis*  
**by** (*simp add: local.c2-d local.c-prod-assoc uc-par-meet*)

**qed**

**lemma** *sv-hom-par*:  $(x \parallel y) \cdot U = (x \cdot U) \parallel (y \cdot U)$   
**by** (*simp add: local.cl4*)

**lemma** *vs-hom-meet*:  $d ((x \cdot 1_\pi) \parallel U) \sqcap ((y \cdot 1_\pi) \parallel U) = d ((x \cdot 1_\pi) \parallel U) \sqcap d ((y \cdot 1_\pi) \parallel U)$

**proof** –

**have**  $f1: \bigwedge x y. x \cdot 1_\pi \parallel 1_\sigma \sqcap y \cdot 1_\pi \parallel 1_\sigma = x \parallel y \cdot 1_\pi \parallel 1_\sigma$   
**using** *d-p-prod-eq-meet local.d-conc6 local.d-def* **by** *auto*  
**hence**  $\bigwedge x y. x \cdot 1_\pi \parallel U \sqcap y \cdot 1_\pi \parallel U = x \parallel y \cdot 1_\pi \parallel U$   
**using** *local.d-def sv-hom-meet* **by** *force*  
**thus** *?thesis*  
**using**  $f1$  **by** (*simp add: local.d-def*)

**qed**

**lemma** *cv-hom-meet*:  $(x \cdot 1_\pi \sqcap y \cdot 1_\pi) \parallel U = (x \cdot 1_\pi) \parallel U \sqcap (y \cdot 1_\pi) \parallel U$

**proof** –

**have**  $d (x \parallel y) \cdot U = x \cdot 1_\pi \parallel U \sqcap y \cdot 1_\pi \parallel U$   
**by** (*simp add: d-p-prod-eq-meet local.c2-d local.d-conc6 sv-hom-meet*)  
**thus** *?thesis*  
**using** *local.c2-d local.c3 p-subid-par-eq-meet-var* **by** *auto*

**qed**

**lemma** *cv-hom-par [simp]*:  $x \parallel U \parallel y \parallel U = (x \parallel y) \parallel U$

**by** (*metis U-idem-p-prod local.mult.left-commute local.mult-assoc*)

**lemma** *vc-hom-meet*:  $((x \cdot 1_\pi) \parallel U \sqcap (y \cdot 1_\pi) \parallel U) \cdot 1_\pi = ((x \cdot 1_\pi) \parallel U) \cdot 1_\pi \sqcap ((y \cdot 1_\pi) \parallel U) \cdot 1_\pi$

**by** (*metis cv-hom-meet iso5 local.c3 p-subid-par-eq-meet-var*)

**lemma** *vc-hom-seq*:  $((x \cdot 1_\pi) \parallel U) \cdot ((y \cdot 1_\pi) \parallel U) \cdot 1_\pi = (((x \cdot 1_\pi) \parallel U) \cdot 1_\pi) \cdot (((y \cdot 1_\pi) \parallel U) \cdot 1_\pi)$

**proof** –

**have**  $((x \cdot 1_\pi) \parallel U) \cdot ((y \cdot 1_\pi) \parallel U) \cdot 1_\pi = ((x \cdot 1_\pi) \parallel U) \cdot (y \cdot 1_\pi)$   
**by** (*simp add: local.c4*)  
**also have**  $\dots = (x \cdot 1_\pi) \parallel (U \cdot (y \cdot 1_\pi))$   
**by** (*metis assoc-p-subid local.cl8*)  
**also have**  $\dots = (x \cdot 1_\pi) \parallel (nc \cdot (y \cdot 1_\pi) + 1_\pi \cdot (y \cdot 1_\pi))$   
**by** (*metis add-commute c-nc-comp1 local.s-prod-distr*)  
**also have**  $\dots = (x \cdot 1_\pi) \parallel 1_\pi$

**by** (*metis add-commute c-x-prop local.absorp2 local.c4 local.meet-comm local.mult-oner p-subid-par-eq-meet-var*)

**thus** *?thesis*

**by** (*simp add: assoc-p-subid calculation*)

**qed**

Proposition 10.4.

**lemma** *nsv-hom-meet*:  $(d x \sqcap d y) \cdot nc = (d x) \cdot nc \sqcap (d y) \cdot nc$

**proof** (*rule order.antisym*)

**have**  $(d x \sqcap d y) \cdot nc \leq (d x) \cdot nc$

**by** (*simp add: local.s-prod-isor*)

**hence**  $(d x \sqcap d y) \cdot nc \leq (d x) \cdot nc$

**by** *blast*  
**thus**  $(d\ x \sqcap d\ y) \cdot nc \leq (d\ x) \cdot nc \sqcap (d\ y) \cdot nc$   
**by** (*simp add: local.s-prod-isor*)  
**have**  $(d\ x) \cdot nc \sqcap (d\ y) \cdot nc \leq ((d\ x) \cdot nc) \parallel ((d\ y) \cdot nc)$   
**by** (*simp add: meet-le-par*)  
**also have**  $\dots = (d\ x \parallel d\ y) \cdot nc$   
**by** (*metis local.cl4 nc-nc-par subidem-par*)  
**finally show**  $(d\ x) \cdot nc \sqcap (d\ y) \cdot nc \leq (d\ x \sqcap d\ y) \cdot nc$   
**by** (*simp add: d-p-prod-eq-meet*)  
**qed**

**lemma** *nsv-hom-par*:  $(x \parallel y) \cdot nc = (x \cdot nc) \parallel (y \cdot nc)$   
**by** (*simp add: local.cl4*)

**lemma** *vec-p-prod-meet*:  $((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc) = ((x \sqcap nc) \cdot nc) \sqcap ((y \sqcap nc) \cdot nc)$

**proof** –

**have**  $((x \sqcap nc) \cdot nc) \parallel ((y \sqcap nc) \cdot nc) = (d\ (x \sqcap nc) \cdot nc) \parallel (d\ (y \sqcap nc) \cdot nc)$   
**by** (*metis cl11-d*)  
**also have**  $\dots = (d\ (x \sqcap nc) \parallel d\ (y \sqcap nc)) \cdot nc$   
**by** (*simp add: nsv-hom-par*)  
**also have**  $\dots = (d\ (x \sqcap nc) \sqcap d\ (y \sqcap nc)) \cdot nc$   
**by** (*simp add: d-p-prod-eq-meet*)  
**also have**  $\dots = (d\ (x \sqcap nc) \cdot nc) \sqcap (d\ (y \sqcap nc) \cdot nc)$   
**by** (*simp add: nsv-hom-meet*)  
**thus** *?thesis*  
**by** (*simp add: calculation*)  
**qed**

**lemma** *nvs-hom-meet*:  $d\ (((x \sqcap nc) \cdot nc) \sqcap ((y \sqcap nc) \cdot nc)) = d\ ((x \sqcap nc) \cdot nc) \sqcap d\ ((y \sqcap nc) \cdot nc)$   
**by** (*metis d-p-prod-eq-meet local.d-conc6 vec-p-prod-meet*)

**lemma** *ncv-hom-meet*:  $(x \cdot 1_\pi \sqcap y \cdot 1_\pi) \parallel nc = (x \cdot 1_\pi) \parallel nc \sqcap (y \cdot 1_\pi) \parallel nc$   
**by** (*metis d-p-prod-eq-meet local.c2-d local.c3 local.d-conc6 nsv-hom-meet p-subid-par-eq-meet-var*)

**lemma** *ncv-hom-par*:  $(x \parallel y) \parallel nc = x \parallel nc \parallel y \parallel nc$   
**by** (*metis local.mult-assoc local.mult-commute nc-nc-par*)

**lemma** *nvc-hom-meet*:  $((x \sqcap nc) \cdot nc \sqcap (y \sqcap nc) \cdot nc) \cdot 1_\pi = ((x \sqcap nc) \cdot nc) \cdot 1_\pi \sqcap ((y \sqcap nc) \cdot nc) \cdot 1_\pi$   
**by** (*metis local.c3 p-subid-par-eq-meet-var vec-p-prod-meet*)

## 2.7 Terminal and Nonterminal Elements

Now we define the projection functions on terminals and nonterminal parts and verify the properties of Section 11 in [2].

**definition** *tau* ::  $'a \Rightarrow 'a$  ( $\tau$ ) **where**  
 $\tau\ x = x \cdot 0$

**definition**  $nu :: 'a \Rightarrow 'a (\nu)$  **where**  
 $\nu x = x \sqcap nc$

Lemma 11.1.

**lemma**  $tau-int$ :  $\tau x \leq x$   
**by** ( $metis\ c-0\ local.inf-le1\ tau-def$ )

**lemma**  $nu-int$ :  $\nu x \leq x$   
**by** ( $simp\ add:\ nu-def$ )

**lemma**  $tau-ret$  [ $simp$ ]:  $\tau (\tau x) = \tau x$   
**by** ( $simp\ add:\ tau-def$ )

**lemma**  $nu-ret$  [ $simp$ ]:  $\nu (\nu x) = \nu x$   
**by** ( $simp\ add:\ local.meet-assoc\ nu-def$ )

**lemma**  $tau-iso$ :  $x \leq y \implies \tau x \leq \tau y$   
**using**  $local.order-prop\ local.s-prod-distr\ tau-def$  **by**  $auto$

**lemma**  $nu-iso$ :  $x \leq y \implies \nu x \leq \nu y$   
**using**  $local.inf-mono\ nu-def$  **by**  $auto$

Lemma 11.2.

**lemma**  $tau-zero$  [ $simp$ ]:  $\tau 0 = 0$   
**by** ( $simp\ add:\ tau-def$ )

**lemma**  $nu-zero$  [ $simp$ ]:  $\nu 0 = 0$   
**using**  $nu-def$  **by**  $auto$

**lemma**  $tau-s$  [ $simp$ ]:  $\tau 1_\sigma = 0$   
**using**  $tau-def$  **by**  $auto$

**lemma**  $nu-s$  [ $simp$ ]:  $\nu 1_\sigma = 1_\sigma$   
**using**  $nu-def$  **by**  $auto$

**lemma**  $tau-c$  [ $simp$ ]:  $\tau 1_\pi = 1_\pi$   
**using**  $c-x-prop\ tau-def$  **by**  $presburger$

**lemma**  $nu-c$  [ $simp$ ]:  $\nu 1_\pi = 0$   
**using**  $c-nc-comp2\ nu-def$  **by**  $presburger$

**lemma**  $tau-nc$  [ $simp$ ]:  $\tau nc = 0$   
**using**  $nc-iff2\ tau-def$  **by**  $auto$

**lemma**  $nu-nc$  [ $simp$ ]:  $\nu nc = nc$   
**using**  $nu-def$  **by**  $auto$

**lemma**  $tau-U$  [ $simp$ ]:  $\tau U = 1_\pi$

**using** *c-def tau-def* **by** *presburger*

**lemma** *nu-U* [*simp*]:  $\nu U = nc$   
**using** *local.U-def local.meet-comm local.inf.absorb-iff1 nu-def* **by** *fastforce*

Lemma 11.3.

**lemma** *tau-add* [*simp*]:  $\tau (x + y) = \tau x + \tau y$   
**by** (*simp add: tau-def x-zero-add-closed*)

**lemma** *nu-add* [*simp*]:  $\nu (x + y) = \nu x + \nu y$   
**by** (*simp add: local.lat-dist3 local.meet-comm nu-def*)

**lemma** *tau-meet* [*simp*]:  $\tau (x \sqcap y) = \tau x \sqcap \tau y$   
**using** *tau-def x-zero-meet-closed* **by** *auto*

**lemma** *nu-meet* [*simp*]:  $\nu (x \sqcap y) = \nu x \sqcap \nu y$   
**using** *nc-meet-closed nu-def* **by** *auto*

**lemma** *tau-seq*:  $\tau (x \cdot y) = \tau x + \nu x \cdot \tau y$   
**using** *local.add-comm nu-def tau-def x-y-split zero-assoc3* **by** *presburger*

**lemma** *tau-par* [*simp*]:  $\tau (x \parallel y) = \tau x \parallel \tau y$   
**using** *tau-def x-zero-interr* **by** *presburger*

**lemma** *nu-par-aux1*:  $x \parallel \tau y = d (\tau y) \cdot x$   
**by** (*simp add: local.c2-d local.mult-commute tau-def*)

**lemma** *nu-par-aux2* [*simp*]:  $\nu (\nu x \parallel \nu y) = \nu x \parallel \nu y$   
**by** (*simp add: nu-def*)

**lemma** *nu-par-aux3* [*simp*]:  $\nu (\nu x \parallel \tau y) = \nu x \parallel \tau y$   
**by** (*metis local.mult-commute nc-ccomp-closed-alt nu-def*)

**lemma** *nu-par-aux4* [*simp*]:  $\nu (\tau x \parallel \tau y) = 0$   
**by** (*metis nu-def tau-def tau-par zero-nc*)

**lemma** *nu-par*:  $\nu (x \parallel y) = d (\tau x) \cdot \nu y + d (\tau y) \cdot \nu x + \nu x \parallel \nu y$

**proof** –

**have**  $\nu (x \parallel y) = \nu (\nu x \parallel \nu y) + \nu (\nu x \parallel \tau y) + \nu (\tau x \parallel \nu y) + \nu (\tau x \parallel \tau y)$   
**by** (*metis local.distrib-left local.distrib-right nu-add nu-def tau-def x-split-var join.sup commute join.sup.left-commute*)

**also have**  $\nu (x \parallel y) = \nu x \parallel \nu y + \nu x \parallel \tau y + \tau x \parallel \nu y$   
**by** (*simp add: calculation local.c-prod-comm*)

**thus** *?thesis*

**using** *local.join.sup-assoc local.join.sup-commute local.mult-commute nu-par-aux1*

**by** *auto*

**qed**

Lemma 11.5.

**lemma** *sprod-tau-nu*:  $x \cdot y = \tau x + \nu x \cdot y$   
**by** (*metis local.add-comm nu-def tau-def x-y-split*)

**lemma** *pprod-tau-nu*:  $x \parallel y = \nu x \parallel \nu y + d (\tau x) \cdot \nu y + d (\tau y) \cdot \nu x + \tau x \parallel \tau y$

**proof** –

**have**  $x \parallel y = \nu (x \parallel y) + \tau (x \parallel y)$

**by** (*simp add: nu-def tau-def*)

**also have**  $\dots = (d (\tau x) \cdot \nu y + d (\tau y) \cdot \nu x + \nu x \parallel \nu y) + \tau x \parallel \tau y$

**by** (*simp add: nu-par*)

**thus** *?thesis*

**using** *add-assoc add-commute calculation* **by force**

**qed**

We now verify some additional properties which are not mentioned in the paper.

**lemma** *tau-idem* [*simp*]:  $\tau x \cdot \tau x = \tau x$   
**by** (*simp add: tau-def*)

**lemma** *tau-interr*:  $(x \parallel y) \cdot \tau z = (x \cdot \tau z) \parallel (y \cdot \tau z)$   
**by** (*simp add: local.cl4 tau-def*)

**lemma** *tau-le-c*:  $\tau x \leq 1_\pi$   
**by** (*simp add: local.x-zero-le-c tau-def*)

**lemma** *c-le-tauc*:  $1_\pi \leq \tau 1_\pi$   
**using** *local.eq-refl tau-c* **by presburger**

**lemma** *x-alpha-tau* [*simp*]:  $\nu x + \tau x = x$   
**using** *nu-def tau-def x-split-var* **by presburger**

**lemma** *alpha-tau-zero* [*simp*]:  $\nu (\tau x) = 0$   
**by** (*simp add: nu-def tau-def*)

**lemma** *tau-alpha-zero* [*simp*]:  $\tau (\nu x) = 0$   
**by** (*simp add: nu-def tau-def*)

**lemma** *sprod-tau-nu-var* [*simp*]:  $\nu (\nu x \cdot y) = \nu (x \cdot y)$

**proof** –

**have**  $\nu (x \cdot y) = \nu (\tau x) + \nu (\nu x \cdot y)$

**by** (*metis nu-add sprod-tau-nu*)

**thus** *?thesis*

**by** *simp*

**qed**

**lemma** *tau-s-prod* [*simp*]:  $\tau (x \cdot y) = x \cdot \tau y$   
**by** (*simp add: tau-def zero-assoc3*)

**lemma** *alpha-fp*:  $\nu x = x \longleftrightarrow x \cdot 0 = 0$



by (*metis local.add-zero tau-alpha-zero tau-def x-alpha-tau*)

**lemma** *alpha-prod-closed* [*simp*]:  $\nu (\nu x \cdot \nu y) = \nu x \cdot \nu y$   
by (*simp add: nu-def*)

**lemma** *alpha-par-prod* [*simp*]:  $\nu (x \parallel \nu y) = x \parallel \nu y$   
by (*simp add: nu-def*)

**lemma** *p-prod-tau-alpha*:  $x \parallel y = x \parallel \nu y + \nu x \parallel y + \tau x \parallel \tau y$   
**proof** –  
**have**  $x \parallel y = (\nu x + \tau x) \parallel (\nu y + \tau y)$   
**using** *x-alpha-tau* **by** *simp*  
**also have**  $\dots = \nu x \parallel \nu y + \nu x \parallel \tau y + \tau x \parallel \nu y + \tau x \parallel \tau y$   
**by** (*metis add-commute local.combine-common-factor local.p-prod-distl*)  
**also have**  $\dots = (\nu x \parallel \nu y + \nu x \parallel \tau y) + (\nu x \parallel \nu y + \tau x \parallel \nu y) + \tau x \parallel \tau y$   
**by** (*simp add: add-ac*)  
**thus** *?thesis*  
**by** (*metis calculation local.add-comm local.distrib-left local.distrib-right x-alpha-tau*)  
**qed**

**lemma** *p-prod-tau-alpha-var*:  $x \parallel y = x \parallel \nu y + \nu x \parallel y + \tau (x \parallel y)$   
by (*metis p-prod-tau-alpha tau-par*)

**lemma** *alpha-par*:  $\nu (x \parallel y) = \nu x \parallel y + x \parallel \nu y$   
**proof** –  
**have**  $\nu (x \parallel y) = \nu (x \parallel \nu y) + \nu (\nu x \parallel y) + \nu (\tau (x \parallel y))$   
**by** (*metis nu-add p-prod-tau-alpha-var*)  
**thus** *?thesis*  
**by** (*simp add: local.mult-commute add-ac*)  
**qed**

**lemma** *alpha-tau* [*simp*]:  $\nu (x \cdot \tau y) = 0$   
by (*metis alpha-tau-zero tau-s-prod*)

**lemma** *nu-par-prop*:  $\nu x = x \implies \nu (x \parallel y) = x \parallel y$   
by (*metis alpha-par-prod local.mult-commute*)

**lemma** *tau-seq-prop*:  $\tau x = x \implies x \cdot y = x$   
by (*metis local.cl6 tau-def*)

**lemma** *tau-seq-prop2*:  $\tau y = y \implies \tau (x \cdot y) = x \cdot y$   
by *auto*

**lemma** *d-nu*:  $\nu (d x \cdot y) = d x \cdot \nu y$   
**proof** –  
**have**  $\nu (d x \cdot y) = \nu ((x \cdot 1_\pi) \parallel y)$   
**by** (*simp add: local.c2-d*)  
**also have**  $\dots = d (\tau (x \cdot 1_\pi)) \cdot \nu y + d (\tau y) \cdot \nu (x \cdot 1_\pi) + \nu (x \cdot 1_\pi) \parallel \nu y$   
**by** (*simp add: nu-par*)

**thus** *?thesis*  
**using** *alpha-par local.c2-d nu-def* **by** *force*  
**qed**

Lemma 11.6 and 11.7.

**lemma** *nu-ideal1*:  $\llbracket \nu x = x; y \leq x \rrbracket \implies \nu y = y$   
**by** (*metis local.meet-prop local.inf.absorb-iff1 nu-def*)

**lemma** *tau-ideal1*:  $\llbracket \tau x = x; y \leq x \rrbracket \implies \tau y = y$   
**by** (*metis local.dual-order.trans tau-def term-p-subid-var*)

**lemma** *nu-ideal2*:  $\llbracket \nu x = x; \nu y = y \rrbracket \implies \nu (x + y) = x + y$   
**by** (*simp add: local.lat-dist3 local.meet-comm*)

**lemma** *tau-ideal2*:  $\llbracket \tau x = x; \tau y = y \rrbracket \implies \tau (x + y) = x + y$   
**by** *simp*

**lemma** *tau-ideal3*:  $\tau x = x \implies \tau (x \cdot y) = x \cdot y$   
**by** (*simp add: tau-seq-prop*)

We prove the precongruence properties of Lemma 11.9.

**lemma** *tau-add-precong*:  $\tau x \leq \tau y \implies \tau (x + z) \leq \tau (y + z)$

**proof** –

**assume**  $\tau x \leq \tau y$

**hence**  $(x + y) \cdot 0 = y \cdot 0$  **using** *local.less-eq-def local.s-prod-distr tau-def*  
**by** *auto*

**hence**  $(x + z + y) \cdot 0 = (y + z) \cdot 0$

**by** (*metis (no-types) add-assoc add-commute local.s-prod-distr*)

**thus**  $\tau (x + z) \leq \tau (y + z)$  **using** *local.order-prop local.s-prod-distr tau-def*  
**by** *metis*

**qed**

**lemma** *tau-meet-precong*:  $\tau x \leq \tau y \implies \tau (x \sqcap z) \leq \tau (y \sqcap z)$

**proof** –

**assume**  $\tau x \leq \tau y$

**hence**  $\bigwedge z. (x \sqcap y \sqcap z) \cdot 0 = (x \sqcap z) \cdot 0$

**by** (*metis local.le-iff-inf tau-def x-zero-meet-closed*)

**thus** *?thesis*

**using** *local.inf-left-commute local.le-iff-inf local.meet-comm tau-def x-zero-meet-closed*

**by** *fastforce*

**qed**

**lemma** *tau-par-precong*:  $\tau x \leq \tau y \implies \tau (x \parallel z) \leq \tau (y \parallel z)$

**proof** –

**assume**  $\tau x \leq \tau y$

**hence**  $x \parallel z \cdot 0 \leq y \cdot 0$

**by** (*metis (no-types) local.dual-order.trans local.p-subid-lb1 tau-def tau-par*)

**thus**  $\tau (x \parallel z) \leq \tau (y \parallel z)$

**by** (*simp add:  $\langle \tau x \leq \tau y \rangle$  local.mult-isor*)

qed

**lemma** *tau-seq-precongl*:  $\tau x \leq \tau y \implies \tau (z \cdot x) \leq \tau (z \cdot y)$   
by (*simp add: local.s-prod-isol*)

**lemma** *nu-add-precong*:  $\nu x \leq \nu y \implies \nu (x + z) \leq \nu (y + z)$

**proof** –

assume  $\nu x \leq \nu y$

hence  $\nu x = \nu x \sqcap \nu y$

using *local.inf.absorb-iff1* by *auto*

hence  $\forall a. \nu (x + a) = \nu (x + a) \sqcap \nu (y + a)$

by (*metis (no-types) local.lat-dist2 nu-add*)

thus *?thesis*

using *local.inf.absorb-iff1* by *presburger*

qed

**lemma** *nu-meet-precong*:  $\nu x \leq \nu y \implies \nu (x \sqcap z) \leq \nu (y \sqcap z)$

**proof** –

assume  $\nu x \leq \nu y$

hence  $\nu y = \nu x + \nu y$

using *local.less-eq-def* by *auto*

hence  $\nu (y \sqcap z) = \nu (x \sqcap z) + \nu (y \sqcap z)$

by (*metis (no-types) local.lat-dist4 nu-meet*)

thus *?thesis*

using *local.less-eq-def* by *presburger*

qed

**lemma** *nu-seq-precongr*:  $\nu x \leq \nu y \implies \nu (x \cdot z) \leq \nu (y \cdot z)$

**proof** –

assume  $a: \nu x \leq \nu y$

have  $\nu (x \cdot z) = \nu (\nu x \cdot z)$

by *simp*

also have  $\dots \leq \nu (\nu y \cdot z)$

by (*metis a local.less-eq-def local.s-prod-distr nu-iso*)

thus *?thesis*

by *simp*

qed

We prove the congruence properties of Corollary 11.11.

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

$tcg\ x\ y = (\tau\ x \leq \tau\ y \wedge \tau\ y \leq \tau\ x)$

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

$ncg\ x\ y = (\nu\ x \leq \nu\ y \wedge \nu\ y \leq \nu\ x)$

**lemma** *tcg-refl*:  $tcg\ x\ x$

by (*simp add: tcg-def*)

**lemma** *tcg-trans*:  $\llbracket tcg\ x\ y; tcg\ y\ z \rrbracket \implies tcg\ x\ z$

```

using tcg-def by force

lemma tcg-sym: tcg x y  $\implies$  tcg y x
  by (simp add: tcg-def)

lemma ncg-refl: ncg x x
  using ncg-def by blast

lemma ncg-trans:  $\llbracket$ ncg x y; ncg y z $\rrbracket \implies$  ncg x z
  using ncg-def by force

lemma ncg-sym: ncg x y  $\implies$  ncg y x
  using ncg-def by auto

lemma tcg-alt: tcg x y = ( $\tau$  x =  $\tau$  y)
  using tcg-def by auto

lemma ncg-alt: ncg x y = ( $\nu$  x =  $\nu$  y)
  by (simp add: order.eq-iff ncg-def)

lemma tcg-add:  $\tau$  x =  $\tau$  y  $\implies$   $\tau$  (x + z) =  $\tau$  (y + z)
  by simp

lemma tcg-meet:  $\tau$  x =  $\tau$  y  $\implies$   $\tau$  (x  $\sqcap$  z) =  $\tau$  (y  $\sqcap$  z)
  by simp

lemma tcg-par:  $\tau$  x =  $\tau$  y  $\implies$   $\tau$  (x  $\parallel$  z) =  $\tau$  (y  $\parallel$  z)
  by simp

lemma tcg-seq1:  $\tau$  x =  $\tau$  y  $\implies$   $\tau$  (z  $\cdot$  x) =  $\tau$  (z  $\cdot$  y)
  by simp

lemma ncg-add:  $\nu$  x =  $\nu$  y  $\implies$   $\nu$  (x + z) =  $\nu$  (y + z)
  by simp

lemma ncg-meet:  $\nu$  x =  $\nu$  y  $\implies$   $\nu$  (x  $\sqcap$  z) =  $\nu$  (y  $\sqcap$  z)
  by simp

lemma ncg-seqr:  $\nu$  x =  $\nu$  y  $\implies$   $\nu$  (x  $\cdot$  z) =  $\nu$  (y  $\cdot$  z)
  by (simp add: order.eq-iff nu-seq-precongr)

end

```

## 2.8 Powers in C-Algebras

We define the power functions from Section 6 in [2] after Lemma 12.4.

```

context proto-dioid
begin

```

**primrec** *p-power* :: 'a ⇒ nat ⇒ 'a **where**  
*p-power* *x* 0 = 1<sub>σ</sub> |  
*p-power* *x* (Suc *n*) = *x* · *p-power* *x* *n*

**primrec** *power-rd* :: 'a ⇒ nat ⇒ 'a **where**  
*power-rd* *x* 0 = 0 |  
*power-rd* *x* (Suc *n*) = 1<sub>σ</sub> + *x* · *power-rd* *x* *n*

**primrec** *power-sq* :: 'a ⇒ nat ⇒ 'a **where**  
*power-sq* *x* 0 = 1<sub>σ</sub> |  
*power-sq* *x* (Suc *n*) = 1<sub>σ</sub> + *x* · *power-sq* *x* *n*

Lemma 12.5

**lemma** *power-rd-chain*: *power-rd* *x* *n* ≤ *power-rd* *x* (*n* + 1)  
**by** (*induct* *n*, *simp*, *metis* *Suc-eq-plus1* *local.add-comm* *local.add-iso* *local.s-prod-isol* *power-rd.simps*(2))

**lemma** *power-sq-chain*: *power-sq* *x* *n* ≤ *power-sq* *x* (*n* + 1)  
**by** (*induct* *n*, *clarsimp*, *metis* *Suc-eq-plus1* *local.add-comm* *local.add-iso* *local.s-prod-isol* *power-sq.simps*(2))

**lemma** *pow-chain*: *p-power* (1<sub>σ</sub> + *x*) *n* ≤ *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)  
**by** (*induct* *n*, *simp*, *metis* *Suc-eq-plus1* *local.p-power.simps*(2) *local.s-prod-isol*)

**lemma** *pow-prop*: *p-power* (1<sub>σ</sub> + *x*) (*n* + 1) = 1<sub>σ</sub> + *x* · *p-power* (1<sub>σ</sub> + *x*) *n*

**proof** (*induct* *n*)

**case** 0

**show** ?*case* **by** *simp*

**next**

**case** (Suc *n*)

**have** *f1*: *p-power* (1<sub>σ</sub> + *x*) (Suc *n* + 1) = 1<sub>σ</sub> + *x* · *p-power* (1<sub>σ</sub> + *x*) *n* + *x* · *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)

**proof** –

**have** *p-power* (1<sub>σ</sub> + *x*) (Suc (*n* + 1)) = (1<sub>σ</sub> + *x*) · *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)

**using** *local.p-power.simps*(2) **by** *blast*

**also have** ... = *p-power* (1<sub>σ</sub> + *x*) (*n* + 1) + *x* · *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)

**by** (*metis* *local.s-prod-distr* *local.s-prod-idl*)

**also have** ... = 1<sub>σ</sub> + *x* · *p-power* (1<sub>σ</sub> + *x*) *n* + *x* · *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)

**using** *Suc.hyps* **by** *auto*

**finally show** ?*thesis*

**by** *simp*

**qed**

**have** *x* · *p-power* (1<sub>σ</sub> + *x*) (Suc *n*) = *x* · *p-power* (1<sub>σ</sub> + *x*) *n* + *x* · *p-power* (1<sub>σ</sub> + *x*) (*n* + 1)

**using** *Suc-eq-plus1* *local.less-eq-def* *local.s-prod-isol* *pow-chain* **by** *simp*

**with** *f1* **show** ?*case* **by** (*simp* *add: add-ac*)

**qed**

Next we verify facts from the proofs of Lemma 12.6.

**lemma** *power-rd-le-sq*:  $\text{power-rd } x \ n \leq \text{power-sq } x \ n$   
**by** (*induct n, simp, simp add: local.join.le-supI2 local.s-prod-isol*)

**lemma** *power-sq-le-rd*:  $\text{power-sq } x \ n \leq \text{power-rd } x \ (\text{Suc } n)$   
**by** (*induct n, simp, simp add: local.join.le-supI2 local.s-prod-isol*)

**lemma** *power-sq-power*:  $\text{power-sq } x \ n = \text{p-power } (1_\sigma + x) \ n$   
**apply** (*induct n*)  
**apply** (*simp*)  
**using** *Suc-eq-plus1 pow-prop power-sq.simps(2)* **by** *simp*

**end**

## 2.9 C-Kleene Algebras

The definition of c-Kleene algebra is slightly different from that in Section 6 of [2]. It is used to prove properties from Section 6 and Section 12.

**class** *c-kleene-algebra* = *c-lattice + star-op +*  
**assumes** *star-unfold*:  $1_\sigma + x \cdot x^* \leq x^*$   
**and** *star-induct*:  $1_\sigma + x \cdot y \leq y \implies x^* \leq y$

**begin**

**lemma** *star-irr*:  $1_\sigma \leq x^*$   
**using** *local.star-unfold* **by** *auto*

**lemma** *star-unfold-part*:  $x \cdot x^* \leq x^*$   
**using** *local.star-unfold* **by** *auto*

**lemma** *star-ext-aux*:  $x \leq x \cdot x^*$   
**using** *local.s-prod-isol star-irr* **by** *fastforce*

**lemma** *star-ext*:  $x \leq x^*$   
**using** *local.order-trans star-ext-aux star-unfold-part* **by** *blast*

**lemma** *star-co-trans*:  $x^* \leq x^* \cdot x^*$   
**using** *local.s-prod-isol star-irr* **by** *fastforce*

**lemma** *star-iso*:  $x \leq y \implies x^* \leq y^*$

**proof** –

**assume** *a1*:  $x \leq y$

**have** *f2*:  $y \cdot y^* + y^* = y^*$

**by** (*meson local.less-eq-def star-unfold-part*)

**have**  $x + y = y$

**using** *a1* **by** (*meson local.less-eq-def*)

**hence**  $x \cdot y^* + y^* = y^*$

**using** *f2* **by** (*metis (no-types) local.add-assoc' local.s-prod-distr*)

**thus** *?thesis*

**using** *local.add-assoc' local.less-eq-def local.star-induct star-irr* **by** *presburger*

qed

**lemma** *star-unfold-eq* [*simp*]:  $1_\sigma + x \cdot x^* = x^*$

**proof** (*rule order.antisym*)

**show**  $a: 1_\sigma + x \cdot x^* \leq x^*$

**using** *local.star-unfold* **by** *blast*

**have**  $1_\sigma + x \cdot (1_\sigma + x \cdot x^*) \leq 1_\sigma + x \cdot x^*$

**by** (*meson local.eq-refl local.join.sup-mono local.s-prod-isol local.star-unfold*)

**thus**  $x^* \leq 1_\sigma + x \cdot x^*$

**by** (*simp add: local.star-induct*)

qed

Lemma 12.2.

**lemma** *nu-star1*:

**assumes**  $\bigwedge x y z. x \cdot (y \cdot z) = (x \cdot y) \cdot z$

**shows**  $x^* \leq (\nu x)^* \cdot (1_\sigma + \tau x)$

**proof** –

**have**  $1_\sigma + x \cdot ((\nu x)^* \cdot (1_\sigma + \tau x)) = 1_\sigma + \tau x + \nu x \cdot ((\nu x)^* \cdot (1_\sigma + \tau x))$

**by** (*metis add-assoc local.sprod-tau-nu*)

**also have**  $\dots = (1_\sigma + \nu x \cdot (\nu x)^*) \cdot (1_\sigma + \tau x)$

**using** *assms local.s-prod-distr local.s-prod-idl* **by** *presburger*

**also have**  $\dots \leq (\nu x)^* \cdot (1_\sigma + \tau x)$

**using** *local.s-prod-isol local.star-unfold* **by** *auto*

**thus** *?thesis*

**by** (*simp add: calculation local.star-induct*)

qed

**lemma** *nu-star2*:

**assumes**  $\bigwedge x. x^* \cdot x^* \leq x^*$

**shows**  $(\nu x)^* \cdot (1_\sigma + \tau x) \leq x^*$

**proof** –

**have**  $(\nu x)^* \cdot (1_\sigma + \tau x) \leq x^* \cdot (1_\sigma + \tau x)$

**using** *local.nu-int local.s-prod-isol star-iso* **by** *blast*

**also have**  $\dots \leq x^* \cdot (1_\sigma + x)$

**using** *local.s-prod-isol local.join.sup-mono local.tau-int* **by** *blast*

**also have**  $\dots \leq x^* \cdot x^*$

**by** (*simp add: local.s-prod-isol star-ext star-irr*)

**finally show** *?thesis*

**using** *assms local.order-trans* **by** *blast*

qed

**lemma** *nu-star*:

**assumes**  $\bigwedge x. x^* \cdot x^* \leq x^*$

**and**  $\bigwedge x y z. x \cdot (y \cdot z) = (x \cdot y) \cdot z$

**shows**  $(\nu x)^* \cdot (1_\sigma + \tau x) = x^*$

**by** (*simp add: assms(1) assms(2) local.dual-order.antisym nu-star1 nu-star2*)

Lemma 12.3.

**lemma** *tau-star*:  $(\tau x)^* = 1_\sigma + \tau x$

by (metis local.cl6 local.tau-def star-unfold-eq)

**lemma tau-star-var:**

assumes  $\bigwedge x y z. x \cdot (y \cdot z) = (x \cdot y) \cdot z$

and  $\bigwedge x. x^* \cdot x^* \leq x^*$

shows  $\tau (x^*) = (\nu x)^* \cdot \tau x$

by (metis (no-types, lifting) assms(1) assms(2) local.add-0-right local.add-comm local.s-prod-distr local.s-prod-idl local.tau-def local.tau-zero nu-star)

**lemma nu-star-sub:**  $(\nu x)^* \leq \nu (x^*)$

by (metis add-commute local.less-eq-def local.meet-prop local.nc-nc local.nu-def local.order-refl local.s-le-nc local.star-induct star-iso)

**lemma nu-star-nu [simp]:**  $\nu ((\nu x)^*) = (\nu x)^*$

using local.nu-ideal1 local.nu-ret nu-star-sub by blast

**lemma nu-star-tau [simp]:**  $\nu ((\tau x)^*) = 1_\sigma$

using tau-star by fastforce

**lemma tau-star-tau [simp]:**  $\tau ((\tau x)^*) = \tau x$

using local.s-prod-distr tau-star by auto

**lemma tau-star-nu [simp]:**  $\tau ((\nu x)^*) = 0$

using local.alpha-fp local.tau-def nu-star-nu by presburger

Finally we verify Lemma 6.2. Proofs can be found in [1].

**lemma d-star-unfold [simp]:**

assumes  $\bigwedge x y z. (x \cdot y) \cdot d z = x \cdot (y \cdot d z)$

shows  $d y + d (x \cdot d (x^* \cdot y)) = d (x^* \cdot y)$

proof –

have  $d y + d (x \cdot d (x^* \cdot y)) = d y + d (x \cdot (x^* \cdot d y))$

by (metis local.c4 local.d-def local.dc-prop1)

moreover have  $\dots = d (1_\sigma \cdot d y + (x \cdot (x^* \cdot d y)))$

by (simp add: local.d-add-ax)

moreover have  $\dots = d (1_\sigma \cdot d y + (x \cdot x^*) \cdot d y)$

by (simp add: assms)

moreover have  $\dots = d ((1_\sigma + x \cdot x^*) \cdot d y)$

using local.s-prod-distr by presburger

ultimately show ?thesis

by simp

qed

**lemma d-star-sim1:**

assumes  $\bigwedge x y z. d z + x \cdot y \leq y \implies x^* \cdot d z \leq y$

and  $\bigwedge x y z. (x \cdot d y) \cdot z = x \cdot (d y \cdot z)$

and  $\bigwedge x y z. (d x \cdot y) \cdot z = d x \cdot (y \cdot z)$

shows  $x \cdot d z \leq d z \cdot y \implies x^* \cdot d z \leq d z \cdot y^*$

proof –

fix  $x y z$



```

assume  $a: x \cdot d z \leq d z \cdot y$ 
have  $b: x \cdot (d z \cdot y^*) \leq d z \cdot (y \cdot y^*)$ 
  by (metis a assms(2) assms(3) local.s-prod-isol)
hence  $x \cdot (d z \cdot y^*) \leq d z \cdot y^*$ 
proof –
  have  $f1: x \cdot (y^* \parallel (z \cdot 1_\pi)) \leq z \cdot 1_\pi \parallel (y \cdot y^*)$ 
    using  $b$  local.c2-d local.mult-commute by auto
  have  $\exists a. (a + z \cdot 1_\pi) \parallel (y \cdot y^*) \leq y^* \parallel (z \cdot 1_\pi)$ 
    by (metis (no-types) local.eq-refl local.mult-commute local.mult-isol-var local.join.sup-idem star-unfold-part)
  hence  $x \cdot (y^* \parallel (z \cdot 1_\pi)) \leq y^* \parallel (z \cdot 1_\pi)$ 
    using  $f1$  by (metis (no-types) local.distrib-right' local.dual-order.trans local.join.sup.cobounded2)
  thus ?thesis
    using local.c2-d local.mult-commute by auto
qed
hence  $d z + x \cdot (d z \cdot y^*) \leq d z \cdot y^*$ 
  using local.s-prod-isol star-irr by fastforce
thus  $x^* \cdot d z \leq d z \cdot y^*$ 
  using assms(1) by force
qed

```

**lemma** *d-star-induct*:

```

assumes  $\bigwedge x y z. d z + x \cdot y \leq y \implies x^* \cdot d z \leq y$ 
and  $\bigwedge x y z. (x \cdot d y) \cdot z = x \cdot (d y \cdot z)$ 
and  $\bigwedge x y z. (d x \cdot y) \cdot z = d x \cdot (y \cdot z)$ 
shows  $d (x \cdot y) \leq d y \implies d (x^* \cdot y) \leq d y$ 

```

```

proof –
  fix  $x y$ 
  assume  $d (x \cdot y) \leq d y$ 
  hence  $x \cdot d y \leq d y \cdot x$ 
    by (simp add: demod1)
  hence  $x^* \cdot d y \leq d y \cdot x^*$ 
    using assms(1) assms(2) assms(3) d-star-sim1 by blast
  thus  $d (x^* \cdot y) \leq d y$ 
    by (simp add: demod2)
qed

```

**end**

## 2.10 C-Omega Algebras

These structures do not feature in [2], but in fact, many lemmas from Section 13 can be proved in this setting. The proto-quantales and c-quantales using in [2] provide a more expressive setting in which least and greatest fixpoints need not be postulated; they exist due to properties of sequential composition and addition over complete lattices.

```

class c-omega-algebra = c-kleene-algebra + omega-op +

```

**assumes** *om-unfold*:  $x^\omega \leq x \cdot x^\omega$   
**and** *om-coinduct*:  $y \leq x \cdot y \implies y \leq x^\omega$

**begin**

Lemma 13.4.

**lemma** *om-unfold-eq* [*simp*]:  $x \cdot x^\omega = x^\omega$   
**apply** (*rule order.antisym*)  
**using** *local.om-coinduct local.om-unfold local.s-prod-isol* **by** *auto*

**lemma** *om-iso*:  $x \leq y \implies x^\omega \leq y^\omega$   
**by** (*metis local.om-coinduct local.s-prod-isol om-unfold-eq*)

Lemma 13.5.

**lemma** *zero-om* [*simp*]:  $0^\omega = 0$   
**by** (*metis local.s-prod-annil om-unfold-eq*)

**lemma** *s-id-om* [*simp*]:  $1_\sigma^\omega = U$   
**by** (*simp add: local.U-def order.eq-iff local.om-coinduct*)

**lemma** *p-id-om* [*simp*]:  $1_\pi^\omega = 1_\pi$   
**by** (*metis local.c-x-prop om-unfold-eq*)

**lemma** *nc-om* [*simp*]:  $nc^\omega = U$   
**using** *local.U-def order.eq-iff local.s-le-nc om-iso s-id-om* **by** *blast*

**lemma** *U-om* [*simp*]:  $U^\omega = U$   
**by** (*simp add: local.U-def order.eq-iff local.om-coinduct*)

Lemma 13.6.

**lemma** *tau-om1*:  $\tau x \leq \tau (x^\omega)$   
**using** *local.om-coinduct local.s-prod-isol local.tau-def local.tau-int* **by** *fastforce*

**lemma** *tau-om2* [*simp*]:  $\tau x^\omega = \tau x$   
**by** (*metis local.cl6 local.tau-def om-unfold-eq*)

**lemma** *tau-om3*:  $(\tau x)^\omega \leq \tau (x^\omega)$   
**by** (*simp add: tau-om1*)

Lemma 13.7.

**lemma** *om-nu-tau*:  $(\nu x)^\omega + (\nu x)^\star \cdot \tau x \leq x^\omega$

**proof** –

**have**  $(\nu x)^\omega + (\nu x)^\star \cdot \tau x = (\nu x)^\omega + (1_\sigma + \nu x \cdot (\nu x)^\star) \cdot \tau x$   
**by** *auto*

**also have**  $\dots = (\nu x)^\omega + \tau x + \nu x \cdot (\nu x)^\star \cdot \tau x$

**using** *add-assoc local.s-prod-distr local.s-prod-idl* **by** *presburger*

**also have**  $\dots = \tau x + \nu x \cdot (\nu x)^\omega + \nu x \cdot (\nu x)^\star \cdot \tau x$

**by** (*simp add: add-ac*)

```

also have ...  $\leq \tau x + \nu x \cdot ((\nu x)^\omega + (\nu x)^* \cdot \tau x)$ 
by (metis add-assoc local.cl5 local.lat-dist1 local.inf.absorb-iff1 local.s-prod-subdistl
local.tau-def)
also have ...  $= x \cdot ((\nu x)^\omega + (\nu x)^* \cdot \tau x)$ 
by (metis local.sprod-tau-nu)
finally show ?thesis
using local.om-coinduct by blast
qed

end

```

## 2.11 C-Nabla Algebras

Nabla-algebras provide yet another way of formalising non-terminating behaviour in Section 13.

```

class c-nabla-algebra = c-omega-algebra +
fixes nabla :: 'a  $\Rightarrow$  'a ( $\nabla$ )
assumes nabla-unfold:  $\nabla x \leq d (x \cdot \nabla x)$ 
and nabla-coinduct:  $d y \leq d (x \cdot y) \Longrightarrow d y \leq \nabla x$ 

```

**begin**

```

lemma nabla-unfold-eq [simp]:  $\nabla x = d (x \cdot \nabla x)$ 
proof (rule order.antisym)
show  $\nabla x \leq d (x \cdot \nabla x)$ 
using local.nabla-unfold by blast
have  $d (x \cdot \nabla x) \leq d (x \cdot d (x \cdot \nabla x))$ 
by (metis local.d-def local.mult-commute local.mult-isol local.nabla-unfold local.s-prod-isol)
also have ...  $= d (x \cdot (x \cdot \nabla x))$ 
using local.d-loc-ax by blast
finally show  $d (x \cdot \nabla x) \leq \nabla x$ 
by (simp add: local.nabla-coinduct)
qed

```

```

lemma nabla-le-s:  $\nabla x \leq 1_\sigma$ 
by (metis local.d-sub-id-ax nabla-unfold-eq)

```

```

lemma nabla-nu [simp]:  $\nu (\nabla x) = \nabla x$ 
using local.nu-ideal1 local.nu-s nabla-le-s by blast

```

Proposition 13.9.

```

lemma nabla-omega-U:
assumes  $\bigwedge x y z. x \cdot (d y \cdot z) = (x \cdot d y) \cdot z$ 
shows  $(\nu x)^\omega = \nabla (\nu x) \cdot U$ 
proof (rule order.antisym)
have  $d ((\nu x)^\omega) \leq \nabla (\nu x)$ 
using local.nabla-coinduct local.om-unfold-eq local.order-refl by presburger

```

**hence**  $(\nu x)^\omega \leq \nabla (\nu x) \cdot (\nu x)^\omega$   
**using** *local.dlp-ax local.dual-order.trans local.s-prod-isor* **by** *blast*  
**thus**  $(\nu x)^\omega \leq \nabla (\nu x) \cdot U$   
**using** *local.U-def local.dual-order.trans local.s-prod-isol* **by** *blast*  
**have**  $\nu x \cdot (\nabla (\nu x) \cdot U) = (\nu x \cdot d (\nabla (\nu x))) \cdot U$   
**by** (*metis assms local.d-s-subid nabla-le-s*)  
**also have**  $\dots = (\nu (\nu x \cdot \nu (\nabla (\nu x)))) \cdot U$   
**by** (*metis local.d-s-subid nabla-le-s nabla-nu local.alpha-prod-closed*)  
**also have**  $\dots = d (\nu (\nu x \cdot \nu (\nabla (\nu x)))) \cdot U$   
**using** *local.ax5-d local.nu-def* **by** *presburger*  
**also have**  $\dots = d (\nu x \cdot \nabla (\nu x)) \cdot U$   
**by** (*metis local.alpha-prod-closed nabla-nu*)  
**finally show**  $\nabla (\nu x) \cdot U \leq (\nu x)^\omega$   
**using** *local.nabla-unfold local.om-coinduct local.s-prod-isor* **by** *presburger*  
**qed**

Corollary 13.10.

**lemma** *nabla-omega-U-cor*:  
**assumes**  $\bigwedge x y z. x \cdot (d y \cdot z) = (x \cdot d y) \cdot z$   
**shows**  $\nabla (\nu x) \cdot U + (\nu x)^* \cdot \tau x \leq x^\omega$   
**by** (*metis assms nabla-omega-U local.om-nu-tau*)

Lemma 13.11.

**lemma** *nu-om-nu*:  
**assumes**  $\bigwedge x y z. x \cdot (d y \cdot z) = (x \cdot d y) \cdot z$   
**shows**  $\nu ((\nu x)^\omega) = \nabla (\nu x) \cdot nc$   
**proof** –  
**have**  $\nu ((\nu x)^\omega) = \nu (\nabla (\nu x) \cdot U)$   
**using** *assms nabla-omega-U* **by** *presburger*  
**also have**  $\dots = \nu (d (\nabla (\nu x)) \cdot U)$   
**by** (*metis local.d-s-subid nabla-le-s*)  
**also have**  $\dots = (\nabla (\nu x)) \cdot \nu U$   
**by** (*metis local.d-nu local.d-s-subid nabla-le-s*)  
**finally show** *?thesis*  
**using** *local.nu-U* **by** *presburger*  
**qed**

**lemma** *tau-om-nu*:  
**assumes**  $\bigwedge x y z. x \cdot (d y \cdot z) = (x \cdot d y) \cdot z$   
**shows**  $\tau ((\nu x)^\omega) = \nabla (\nu x) \cdot 1_\pi$   
**proof** –  
**have**  $\tau ((\nu x)^\omega) = \tau (\nabla (\nu x) \cdot U)$   
**by** (*metis assms nabla-omega-U*)  
**also have**  $\dots = \nabla (\nu x) \cdot \tau U$   
**using** *local.tau-s-prod* **by** *blast*  
**finally show** *?thesis*  
**using** *local.tau-U* **by** *blast*  
**qed**

Proposition 13.12.

**lemma** *wf-eq-defl*:  $(\forall y. d\ y \leq d\ (x \cdot y) \longrightarrow d\ y = 0) \longleftrightarrow (\forall y. y \leq x \cdot y \longrightarrow y = 0)$

**apply** *standard*

**apply** (*metis local.d-add-ax local.d-rest-ax local.less-eq-def local.s-prod-annil*)

**by** (*metis local.c2-d local.c4 local.d-def local.mult-commute local.mult-onel local.p-rpd-annir local.s-prod-isor*)

**lemma** *defl-eq-om-trivial*:  $x^\omega = 0 \longleftrightarrow (\forall y. y \leq x \cdot y \longrightarrow y = 0)$

**using** *local.join.bot-unique local.om-coinduct* **by** *auto*

**lemma** *wf-eq-om-trivial*:  $x^\omega = 0 \longleftrightarrow (\forall y. d\ y \leq d\ (x \cdot y) \longrightarrow d\ y = 0)$

**by** (*simp add: defl-eq-om-trivial wf-eq-defl*)

**end**

## 2.12 Proto-Quantales

Finally we define the class of proto-quantales and prove some of the remaining facts from the article. Full c-quantales, as defined there, are not needed for these proofs.

**class** *proto-quantale* = *complete-lattice* + *proto-monoid* +  
**assumes** *Sup-mult-distr*:  $Sup\ X \cdot y = Sup\ \{x \cdot y \mid x. x \in X\}$   
**and** *isol*:  $x \leq y \implies z \cdot x \leq z \cdot y$

**begin**

**sublocale** *pd?*: *proto-diod*  $1_\sigma$   $(\cdot)$  *sup*  $(\leq)$   $(<)$  *Sup*  $\{\}$

**proof**

**show**  $\bigwedge x\ y. (x \leq y) = (sup\ x\ y = y)$

**by** (*simp add: local.le-iff-sup*)

**show**  $\bigwedge x\ y. (x < y) = (x \leq y \wedge x \neq y)$

**by** (*simp add: local.order.strict-iff-order*)

**show**  $\bigwedge x\ y\ z. sup\ (sup\ x\ y)\ z = sup\ x\ (sup\ y\ z)$

**by** (*simp add: local.sup-assoc*)

**show**  $\bigwedge x\ y. sup\ x\ y = sup\ y\ x$

**by** (*simp add: local.sup-commute*)

**show**  $\bigwedge x. sup\ x\ x = x$

**by** (*simp add: insert-commute*)

**show**  $\bigwedge x. sup\ (Sup\ \{x\})\ x = x$

**by** *simp*

**show**  $\bigwedge x\ y\ z. sup\ (x \cdot y)\ (x \cdot z) \leq x \cdot (sup\ y\ z)$

**by** (*simp add: local.isol*)

**show**  $\bigwedge x. Sup\ \{x\} \cdot x = Sup\ \{x\}$

**proof** –

**fix**  $x :: 'a$

**have**  $\forall A\ a. \{x\} \neq A \vee \{x\} = \{aa. \exists ab. (aa::'a) = ab \cdot a \wedge ab \in A\}$

**by** *fastforce*

**thus**  $Sup\ \{x\} \cdot x = Sup\ \{x\}$

**using** *local.Sup-mult-distr* **by** *presburger*

```

qed
show  $\bigwedge x y z. (sup\ x\ y) \cdot z = sup\ (x \cdot z)\ (y \cdot z)$ 
proof -
  fix x y z
  have  $(sup\ x\ y) \cdot z = Sup\ \{x,y\} \cdot z$ 
  by simp
  moreover have  $\dots = sup\ (x \cdot z)\ (y \cdot z)$ 
  by (subst Sup-mult-distr, rule Sup-eqI, auto)
  thus  $(sup\ x\ y) \cdot z = sup\ (x \cdot z)\ (y \cdot z)$ 
  using calculation by presburger
qed
qed

```

**definition** *star-rd* :: 'a  $\Rightarrow$  'a **where**  
*star-rd* x = Sup {power-rd x i | i. i  $\in$   $\mathbf{N}$ }

**definition** *star-sq* :: 'a  $\Rightarrow$  'a **where**  
*star-sq* x = Sup {power-sq x i | i. i  $\in$   $\mathbf{N}$ }

Now we prove Lemma 12.6.

**lemma** *star-rd-le-sq*: *star-rd* x  $\leq$  *star-sq* x  
**apply** (auto simp: star-rd-def star-sq-def)  
**apply** (rule Sup-mono)  
**using** pd.power-rd-le-sq **by** auto

**lemma** *star-sq-le-rd*: *star-sq* x  $\leq$  *star-rd* x  
**apply** (auto simp: star-rd-def star-sq-def)  
**apply** (rule Sup-mono)  
**apply** auto  
**by** (metis Nats-1 Nats-add Suc-eq-plus1 local.Sup-empty pd.power-sq-le-rd)

**lemma** *star-rd-sq*: *star-rd* x = *star-sq* x  
**by** (simp add: local.dual-order.antisym star-rd-le-sq star-sq-le-rd)

**lemma** *star-sq-power*: *star-sq* x = Sup {pd.p-power (sup 1 <sub>$\sigma$</sub>  x) i | i. i  $\in$   $\mathbf{N}$ }  
**by** (auto simp: star-sq-def pd.power-sq-power [symmetric] intro: Sup-eqI)

The following lemma should be somewhere close to complete lattices.

**end**

**lemma** *mono-aux*: mono ( $\lambda y. sup\ (z :: 'a :: proto-quantale)\ (x \cdot y)$ )  
**by** (meson mono-def order-refl pd.s-prod-isol sup.mono)

**lemma** *gfp-lfp-prop*:  $sup\ (gfp\ (\lambda(y :: 'a :: proto-quantale). x \cdot y))\ (lfp\ (\lambda y. sup\ z\ (x \cdot y))) \leq gfp\ (\lambda y. sup\ z\ (x \cdot y))$   
**apply** (simp, rule conjI)  
**apply** (simp add: gfp-mono)  
**by** (simp add: lfp-le-gfp mono-aux)

end

### 3 Multirelations

**theory** *Multirelations*  
**imports** *C-Algebras*  
**begin**

#### 3.1 Basic Definitions

We define a type synonym for multirelations.

**type-synonym** *'a mrel* = (*'a* \* (*'a set*)) *set*

**no-notation** *s-prod* (**infixl** · 80)

**no-notation** *s-id* ( $1_\sigma$ )

**no-notation** *c-prod* (**infixl** || 80)

**no-notation** *c-id* ( $1_\pi$ )

Now we start with formalising the multirelational model. First we define sequential composition and parallel composition of multirelations, their units and the universal multirelation as in Section 2 of the article.

**definition** *s-prod* :: *'a mrel*  $\Rightarrow$  *'a mrel*  $\Rightarrow$  *'a mrel* (**infixl** · 70) **where**  
 $R \cdot S = \{(a,A). (\exists B. (a,B) \in R \wedge (\exists f. (\forall b \in B. (b,f b) \in S) \wedge A = \bigcup \{f b \mid b. b \in B\}))\}$

**definition** *s-id* :: *'a mrel* ( $1_\sigma$ ) **where**

$$1_\sigma \equiv \bigcup a. \{(a, \{a\})\}$$

**definition** *p-prod* :: *'a mrel*  $\Rightarrow$  *'a mrel*  $\Rightarrow$  *'a mrel* (**infixl** || 70) **where**

$$R \parallel S = \{(a,A). (\exists B C. A = B \cup C \wedge (a,B) \in R \wedge (a,C) \in S)\}$$

**definition** *p-id* :: *'a mrel* ( $1_\pi$ ) **where**

$$1_\pi \equiv \bigcup a. \{(a, \{\})\}$$

**definition** *U* :: *'a mrel* **where**

$$U \equiv \{(a,A) \mid a A. a \in UNIV \wedge A \subseteq UNIV\}$$

**abbreviation** *NC*  $\equiv U - 1_\pi$

We write NC where  $\overline{1_\pi}$  is written in [2].

Next we prove some basic set-theoretic properties.

**lemma** *s-prod-im*:  $R \cdot S = \{(a,A). (\exists B. (a,B) \in R \wedge (\exists f. (\forall b \in B. (b,f b) \in S) \wedge A = \bigcup ((\lambda x. f x) ' B)))\}$   
**by** (*auto simp: s-prod-def*)

**lemma** *s-prod-iff*:  $(a,A) \in (R \cdot S) \iff (\exists B. (a,B) \in R \wedge (\exists f. (\forall b \in B. (b,f b) \in S) \wedge A = \bigcup ((\lambda x. f x) ' B)))$

by (*unfold s-prod-in, auto*)

**lemma** *s-id-iff*:  $(a, A) \in 1_\sigma \iff A = \{a\}$   
 by (*simp add: s-id-def*)

**lemma** *p-prod-iff*:  $(a, A) \in R \parallel S \iff (\exists B C. A = B \cup C \wedge (a, B) \in R \wedge (a, C) \in S)$   
 by (*clarsimp simp add: p-prod-def*)

**named-theorems** *mr-simp*

**declare** *s-prod-in* [*mr-simp*] *p-prod-def* [*mr-simp*] *s-id-def* [*mr-simp*] *p-id-def* [*mr-simp*] *U-def* [*mr-simp*]

### 3.2 Multirelations and Proto-Dioids

We can now show that multirelations form proto-trioids. This is Proposition 5.1, and it subsumes Proposition 4.1,

**interpretation** *mrel-proto-trioid*: *proto-trioid*  $1_\sigma$   $(\cdot)$   $1_\pi$   $(\parallel)$   $(\cup)$   $(\subseteq)$   $(\subset)$   $\{\}$   
**proof**

show  $\bigwedge x. 1_\sigma \cdot x = x$

by (*auto simp: mr-simp*)

show  $\bigwedge x. x \cdot 1_\sigma = x$

by (*auto simp add: mr-simp*) (*metis UN-singleton*)

show  $\bigwedge x. 1_\pi \parallel x = x$

by (*simp add: mr-simp*)

show  $\bigwedge x y z. x \parallel y \parallel z = x \parallel (y \parallel z)$

apply (*rule antisym*)

apply (*clarsimp simp: mr-simp Un-assoc, metis*)

by (*clarsimp simp: mr-simp, metis (no-types) Un-assoc*)

show  $\bigwedge x y. x \parallel y = y \parallel x$

by (*auto simp: mr-simp*)

show  $\bigwedge x y. (x \subseteq y) = (x \cup y = y)$

by *blast*

show  $\bigwedge x y. (x \subset y) = (x \subseteq y \wedge x \neq y)$

by (*simp add: psubset-eq*)

show  $\bigwedge x y z. x \cup y \cup z = x \cup (y \cup z)$

by (*simp add: Un-assoc*)

show  $\bigwedge x y. x \cup y = y \cup x$

by *blast*

show  $\bigwedge x. x \cup x = x$

by *auto*

show  $\bigwedge x. \{\} \cup x = x$

by *blast*

show  $\bigwedge x y z. (x \cup y) \cdot z = x \cdot z \cup y \cdot z$

by (*auto simp: mr-simp*)

show  $\bigwedge x y z. x \cdot y \cup x \cdot z \subseteq x \cdot (y \cup z)$

by (*auto simp: mr-simp*)

show  $\bigwedge x. \{\} \cdot x = \{\}$

by (*auto simp: mr-simp*)



```

show  $\bigwedge x y z. x \parallel (y \cup z) = x \parallel y \cup x \parallel z$ 
  by (auto simp: mr-simp)
show  $\bigwedge x. x \parallel \{\} = \{\}$ 
  by (simp add: mr-simp)
qed

```

### 3.3 Simple Properties

This covers all the identities in the display before Lemma 2.1 except the two following ones.

```

lemma s-prod-assoc1:  $(R \cdot S) \cdot T \subseteq R \cdot (S \cdot T)$ 
  by (clarsimp simp: mr-simp, metis)

```

```

lemma seq-conc-subdistr:  $(R \parallel S) \cdot T \subseteq (R \cdot T) \parallel (S \cdot T)$ 
  by (clarsimp simp: mr-simp UnI1 UnI2, blast)

```

Next we provide some counterexamples. These do not feature in [2].

```

lemma  $R \cdot \{\} = \{\}$ 
  nitpick
  oops

```

```

lemma  $R \cdot (S \cup T) = R \cdot S \cup R \cdot T$ 
  apply (auto simp: s-prod-im)
  nitpick
  oops

```

```

lemma  $R \cdot (S \cdot T) \subseteq (R \cdot S) \cdot T$ 
  apply (auto simp: s-prod-im)
  nitpick
  oops

```

```

lemma  $(R \parallel R) \cdot T = (R \cdot T) \parallel (R \cdot T)$ 
  quickcheck
  oops

```

Next we prove the distributivity and associativity laws for sequential subidentities mentioned before Lemma 2.1

```

lemma subid-aux2:
assumes  $R \subseteq 1_\sigma$  and  $(a, A) \in R$ 
shows  $A = \{a\}$ 
  using assms by (auto simp: mr-simp)

```

```

lemma s-prod-test-aux1:
assumes  $S \subseteq 1_\sigma$ 
and  $(a, A) \in R \cdot S$ 
shows  $((a, A) \in R \wedge (\forall a \in A. (a, \{a\}) \in S))$ 
  using assms apply (clarsimp simp: s-prod-im)

```

**by** (*metis* *assms*(2) *mrel-proto-trioid.s-prod-idr* *mrel-proto-trioid.s-prod-isol* *singletonD* *subid-aux2* *subset-eq*)

**lemma** *s-prod-test-aux2*:

**assumes**  $(a,A) \in R$

**and**  $\forall a \in A. (a,\{a\}) \in S$

**shows**  $(a,A) \in R \cdot S$

**using** *assms* **by** (*auto* *simp*: *mr-simp*, *fastforce*)

**lemma** *s-prod-test*:

**assumes**  $P \subseteq 1_\sigma$

**shows**  $(a,A) \in R \cdot P \longleftrightarrow (a,A) \in R \wedge (\forall a \in A. (a,\{a\}) \in P)$

**by** (*meson* *assms* *s-prod-test-aux1* *s-prod-test-aux2*)

**lemma** *test-s-prod-aux1*:

**assumes**  $P \subseteq 1_\sigma$

**and**  $(a,A) \in P \cdot R$

**shows**  $(a,\{a\}) \in P \wedge (a,A) \in R$

**by** (*metis* *assms* *mrel-proto-trioid.s-prod-idl* *s-id-iff* *s-prod-iff* *subid-aux2*)

**lemma** *test-s-prod-aux2*:

**assumes**  $(a,A) \in R$

**and**  $(a,\{a\}) \in P$

**shows**  $(a,A) \in P \cdot R$

**using** *assms* *s-prod-iff* **by** *fastforce*

**lemma** *test-s-prod*:

**assumes**  $P \subseteq 1_\sigma$

**shows**  $(a,A) \in P \cdot R \longleftrightarrow (a,\{a\}) \in P \wedge (a,A) \in R$

**by** (*meson* *assms* *test-s-prod-aux1* *test-s-prod-aux2*)

**lemma** *test-assoc1*:

**assumes**  $P \subseteq 1_\sigma$

**shows**  $(R \cdot P) \cdot S = R \cdot (P \cdot S)$

**proof** (*rule antisym*)

**show**  $(R \cdot P) \cdot S \subseteq R \cdot (P \cdot S)$

**by** (*metis* *s-prod-assoc1*)

**next**

**show**  $R \cdot (P \cdot S) \subseteq (R \cdot P) \cdot S$  **using** *assms*

**proof** *clarify*

**fix**  $a A$

**assume**  $(a,A) \in R \cdot (P \cdot S)$

**hence**  $\exists B.(a,B) \in R \wedge (\exists f. (\forall b \in B. (b,f b) \in P \cdot S) \wedge A = \bigcup((\lambda x. f x) ` B))$

**by** (*clarsimp* *simp*: *mr-simp*)

**hence**  $\exists B.(a,B) \in R \wedge (\exists f. (\forall b \in B. (b,\{b\}) \in P \wedge (b,f b) \in S) \wedge A = \bigcup((\lambda x. f x) ` B))$

**by** (*metis* *assms* *test-s-prod*)

**hence**  $\exists B.(a,B) \in R \wedge (\forall b \in B. (b,\{b\}) \in P) \wedge (\exists f. (\forall b \in B. (b,f b) \in S) \wedge$

$A = \bigcup((\lambda x. f x) \text{ ‘ } B)$   
**by** *auto*  
**hence**  $\exists B. (a, B) \in R \cdot P \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge A = \bigcup((\lambda x. f x) \text{ ‘ } B)$   
**by** (*metis assms s-prod-test*)  
**thus**  $(a, A) \in (R \cdot P) \cdot S$   
**by** (*clarsimp simp: mr-simp*)  
**qed**  
**qed**

**lemma** *test-assoc2*:

**assumes**  $P \subseteq 1_\sigma$   
**shows**  $(P \cdot R) \cdot S = P \cdot (R \cdot S)$   
**proof** (*rule antisym*)  
**show**  $(P \cdot R) \cdot S \subseteq P \cdot (R \cdot S)$   
**by** (*metis s-prod-assoc1*)  
**show**  $P \cdot (R \cdot S) \subseteq (P \cdot R) \cdot S$  **using** *assms*  
**proof** *clarify*  
**fix**  $a A$   
**assume**  $(a, A) \in P \cdot (R \cdot S)$   
**hence**  $(a, \{a\}) \in P \wedge (a, A) \in R \cdot S$   
**by** (*metis assms test-s-prod*)  
**hence**  $(a, \{a\}) \in P \wedge (\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge A = \bigcup((\lambda x. f x) \text{ ‘ } B))$   
**by** (*clarsimp simp: mr-simp*)  
**hence**  $\exists B. (a, \{a\}) \in P \wedge (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge A = \bigcup((\lambda x. f x) \text{ ‘ } B)$   
**by** (*clarsimp simp: mr-simp*)  
**hence**  $\exists B. (a, B) \in P \cdot R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge A = \bigcup((\lambda x. f x) \text{ ‘ } B)$   
**by** (*metis assms test-s-prod*)  
**thus**  $(a, A) \in (P \cdot R) \cdot S$   
**by** (*clarsimp simp: mr-simp*)  
**qed**  
**qed**

**lemma** *test-assoc3*:

**assumes**  $P \subseteq 1_\sigma$   
**shows**  $(R \cdot S) \cdot P = R \cdot (S \cdot P)$   
**proof** (*rule antisym*)  
**show**  $(R \cdot S) \cdot P \subseteq R \cdot (S \cdot P)$   
**by** (*metis s-prod-assoc1*)  
**show**  $R \cdot (S \cdot P) \subseteq (R \cdot S) \cdot P$  **using** *assms*  
**proof** *clarify*  
**fix**  $a A$   
**assume** *hyp1*:  $(a, A) \in R \cdot (S \cdot P)$   
**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S \cdot P) \wedge A = \bigcup((\lambda x. f x) \text{ ‘ } B)$   
**by** (*simp add: s-prod-test s-prod-im*)  
**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S \wedge (\forall a \in f b. (a, \{a\}) \in P)) \wedge$

$A = \bigcup((\lambda x. f x) \cdot B)$   
**by** (*simp add: s-prod-test assms*)  
**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge (\forall a \in \bigcup((\lambda x. f x) \cdot B). (a, \{a\}) \in P) \wedge A = \bigcup((\lambda x. f x) \cdot B))$   
**by** *auto*  
**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge (\forall a \in A. (a, \{a\}) \in P) \wedge A = \bigcup((\lambda x. f x) \cdot B))$   
**by** *auto blast*  
**hence**  $(a, A) \in R \cdot S \wedge (\forall a \in A. (a, \{a\}) \in P)$   
**by** (*auto simp: mr-simp*)  
**thus**  $(a, A) \in (R \cdot S) \cdot P$   
**by** (*simp add: s-prod-test assms*)  
**qed**  
**qed**

**lemma** *s-distl-test*:  
**assumes**  $R \subseteq 1_\sigma$   
**shows**  $R \cdot (S \cup T) = R \cdot S \cup R \cdot T$   
**apply** (*clarsimp simp: mr-simp*) **using** *assms subid-aux2* **by** *fastforce*

Next we verify Lemma 2.1.

**lemma** *subid-par-idem*:  
**assumes**  $R \subseteq 1_\sigma$   
**shows**  $R \parallel R = R$   
**by** (*rule set-eqI, clarsimp simp: mr-simp, metis Un-absorb assms subid-aux2*)

**lemma** *term-par-idem*:  
**assumes**  $R \subseteq 1_\pi$   
**shows**  $R \parallel R = R$   
**using** *assms* **by** (*auto simp: mr-simp*)

**lemma** *U-par-idem*:  $U \parallel U = U$   
**by** (*auto simp: mr-simp*)

**lemma** *nc-par-idem*:  $NC \parallel NC = NC$   
**by** (*auto simp: mr-simp*)

Next we prove the properties of Lemma 2.2 and 3.2. First we prepare to show that multirelations form c-lattices.

We define the domain operation on multirelations and verify the explicit definition from Section 3.

**definition**  $d :: 'a \text{ mrel} \Rightarrow 'a \text{ mrel}$  **where**  
 $d R \equiv \{(a, \{a\}) \mid a. \exists B. (a, B) \in R\}$

**named-theorems** *mrd-simp*  
**declare** *mr-simp* [*mrd-simp*] *d-def* [*mrd-simp*]

**lemma** *d-def-expl*:  $d R = (R \cdot 1_\pi) \parallel 1_\sigma$

**apply** (*simp add: mrd-simp*) **using** *set-eqI* **by** *force*

**interpretation** *mrel-pbdl-monoid: pbdl-monoid*  $1_\sigma (\cdot) 1_\pi (\parallel) (\cup) (\subseteq) (\subset) \{\} U$   
 $(\cap)$   
**by** (*unfold-locales, auto simp: mrd-simp*)

Here come the properties of Lemma 2.2.

**lemma** *c1*:  $(R \cdot 1_\pi) \parallel R = R$   
**apply** (*rule set-eqI*)  
**apply** (*clarsimp simp: mr-simp*)  
**by** (*metis (no-types, lifting) SUP-bot SUP-bot-conv(2) sup-bot.left-neutral*)

**lemma** *t-aux*:  $T \parallel T \subseteq T \implies (\forall a B C. (a,B) \in T \wedge (a,C) \in T \implies (a,B \cup C) \in T)$   
**by** (*clarsimp simp: mr-simp*)

**lemma** *cl4*:

**assumes**  $T \parallel T \subseteq T$

**shows**  $(R \cdot T) \parallel (S \cdot T) \subseteq (R \parallel S) \cdot T$

**proof** *clarify*

**fix**  $a A$

**assume**  $(a,A) \in (R \cdot T) \parallel (S \cdot T)$

**hence**  $\exists B C. A = B \cup C \wedge (\exists D. (a,D) \in R \wedge (\exists f. (\forall d \in D. (d,f d) \in T) \wedge B = \bigcup ((\lambda x. f x) \text{' } D))) \wedge (\exists E. (a,E) \in S \wedge (\exists g. (\forall e \in E. (e,g e) \in T) \wedge C = \bigcup ((\lambda x. g x) \text{' } E)))$

**by** (*simp add: mr-simp*)

**hence**  $\exists D E. (a,D \cup E) \in R \parallel S \wedge (\exists f g. (\forall d \in D. (d,f d) \in T) \wedge (\forall e \in E. (e,g e) \in T) \wedge A = (\bigcup ((\lambda x. f x) \text{' } D)) \cup (\bigcup ((\lambda x. g x) \text{' } E)))$

**by** (*auto simp: mr-simp*)

**hence**  $\exists D E. (a,D \cup E) \in R \parallel S \wedge (\exists f g. (\forall d \in D-E. (d,f d) \in T) \wedge (\forall e \in E-D. (e,g e) \in T) \wedge (\forall x \in D \cap E. (x,f x) \in T \wedge (x,g x) \in T) \wedge A = (\bigcup ((\lambda x. f x) \text{' } (D-E))) \cup (\bigcup ((\lambda x. g x) \text{' } (E-D))) \cup (\bigcup ((\lambda y. f y \cup g y) \text{' } (D \cap E))))$

**by** *auto blast*

**hence**  $\exists D E. (a,D \cup E) \in R \parallel S \wedge (\exists f g. (\forall d \in D-E. (d,f d) \in T) \wedge (\forall e \in E-D. (e,g e) \in T) \wedge (\forall x \in D \cap E. (x,f x \cup g x) \in T) \wedge A = (\bigcup ((\lambda x. f x) \text{' } (D-E))) \cup (\bigcup ((\lambda x. g x) \text{' } (E-D))) \cup (\bigcup ((\lambda y. f y \cup g y) \text{' } (D \cap E))))$

**apply** *clarify*

**apply** (*rule-tac x = D in exI*)

**apply** (*rule-tac x = E in exI*)

**apply** *clarify*

**apply** (*rule-tac x = f in exI*)

**apply** (*rule-tac x = g in exI*)

**using** *assms* **by** (*auto simp: p-prod-def p-prod-iff, blast*)

**hence**  $\exists D E. (a,D \cup E) \in R \parallel S \wedge (\exists h. (\forall d \in D-E. (d,h d) \in T) \wedge (\forall e \in E-D. (e,h e) \in T) \wedge (\forall x \in D \cap E. (x,h x) \in T) \wedge A = (\bigcup ((\lambda x. h x) \text{' } (D-E))) \cup (\bigcup ((\lambda x. h x) \text{' } (E-D))) \cup (\bigcup ((\lambda y. h y) \text{' } (D \cap E))))$

**apply** *clarify*

**apply** (*rule-tac x = D in exI*)

**apply** (*rule-tac x = E in exI*)

**apply clarify**  
**apply** (*rule-tac*  $x = \lambda x. \text{if } x \in (D - E) \text{ then } f x \text{ else (if } x \in D \cap E \text{ then } (f x \cup g x) \text{ else } g x)$  **in**  $exI$ )  
**by auto**  
**hence**  $(\exists B. (a, B) \in R \parallel S \wedge (\exists h. (\forall b \in B. (b, h b) \in T) \wedge A = \bigcup((\lambda x. h x) \text{ ' } B)))$   
**by auto blast**  
**thus**  $(a, A) \in (R \parallel S) \cdot T$   
**by (simp add: mr-simp)**  
**qed**

**lemma cl3:**  $R \cdot (S \parallel T) \subseteq (R \cdot S) \parallel (R \cdot T)$

**proof clarify**

**fix**  $a A$

**assume**  $(a, A) \in R \cdot (S \parallel T)$

**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. \exists C D. f b = C \cup D \wedge (b, C) \in S \wedge (b, D) \in T) \wedge A = \bigcup((\lambda x. f x) \text{ ' } B))$

**by (clarsimp simp: mr-simp)**

**hence**  $\exists B. (a, B) \in R \wedge (\exists f g h. (\forall b \in B. f b = g b \cup h b \wedge (b, g b) \in S \wedge (b, h b) \in T) \wedge A = \bigcup((\lambda x. f x) \text{ ' } B))$

**by (clarsimp simp: bchoice, metis)**

**hence**  $\exists B. (a, B) \in R \wedge (\exists g h. (\forall b \in B. (b, g b) \in S \wedge (b, h b) \in T) \wedge A = (\bigcup((\lambda x. g x) \text{ ' } B)) \cup (\bigcup((\lambda x. h x) \text{ ' } B)))$

**by blast**

**hence**  $\exists C D. (\exists B. (a, B) \in R \wedge (\exists g. (\forall b \in B. (b, g b) \in S) \wedge C = \bigcup((\lambda x. g x) \text{ ' } B))) \wedge (\exists B. (a, B) \in R \wedge (\exists h. (\forall b \in B. (b, h b) \in T) \wedge D = \bigcup((\lambda x. h x) \text{ ' } B))) \wedge A = C \cup D$

**by blast**

**thus**  $(a, A) \in (R \cdot S) \parallel (R \cdot T)$

**by (auto simp: mr-simp)**

**qed**

**lemma cl5:**  $(R \cdot S) \cdot (T \cdot \{\}) = R \cdot (S \cdot (T \cdot \{\}))$

**proof (rule antisym)**

**show**  $(R \cdot S) \cdot (T \cdot \{\}) \subseteq R \cdot (S \cdot (T \cdot \{\}))$

**by (metis s-prod-assoc1)**

**show**  $R \cdot (S \cdot (T \cdot \{\})) \subseteq (R \cdot S) \cdot (T \cdot \{\})$

**proof clarify**

**fix**  $a A$

**assume**  $(a, A) \in R \cdot (S \cdot (T \cdot \{\}))$

**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (\exists C. (b, C) \in S \wedge (\exists g. (\forall x \in C. (x, g x) \in T \cdot \{\}) \wedge f b = \bigcup((\lambda x. g x) \text{ ' } C)))) \wedge A = \bigcup((\lambda x. f x) \text{ ' } B))$

**by (clarsimp simp: mr-simp)**

**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (\exists C. (b, C) \in S \wedge (\forall x \in C. (x, \{\}) \in T \cdot \{\}) \wedge f b = \{\})) \wedge A = \bigcup((\lambda x. f x) \text{ ' } B))$

**by (clarsimp simp: mr-simp) fastforce**

**hence**  $\exists B. (a, B) \in R \wedge (\forall b \in B. (\exists C. (b, C) \in S \wedge (\forall x \in C. (x, \{\}) \in T \cdot \{\}))) \wedge A = \{\}$

**by (metis (erased, opaque-lifting) SUP-bot-conv(2))**

**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S \wedge (\forall x \in f b. (x, \{\}) \in T \cdot \{\}))) \wedge A = \{\}$   
**by** *metis*  
**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge (\forall x \in \bigcup((\lambda x. f x) \cdot B). (x, \{\}) \in T \cdot \{\}))) \wedge A = \{\}$   
**by** (*metis UN-E*)  
**hence**  $\exists C B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge C = \bigcup((\lambda x. f x) \cdot B) \wedge (\forall x \in C. (x, \{\}) \in T \cdot \{\}))) \wedge A = \{\}$   
**by** *metis*  
**hence**  $\exists C. (a, C) \in R \cdot S \wedge (\forall x \in C. (x, \{\}) \in T \cdot \{\}) \wedge A = \{\}$   
**by** (*auto simp: mr-simp*)  
**thus**  $(a, A) \in (R \cdot S) \cdot (T \cdot \{\})$   
**by** (*clarsimp simp: mr-simp*) *blast*  
**qed**  
**qed**

We continue verifying other c-lattice axioms

**lemma** *cl8-var*:  $d R \cdot S = (R \cdot 1_\pi) \parallel S$   
**apply** (*rule set-eqI*)  
**apply** (*clarsimp simp: mrd-simp*)  
**apply** *standard*  
**apply** (*metis SUP-bot sup commute sup-bot.right-neutral*)  
**by** *auto*

**lemma** *cl9-var*:  $d (R \cap 1_\sigma) = R \cap 1_\sigma$   
**by** (*auto simp: mrd-simp*)

**lemma** *cl10-var*:  $d (R - 1_\pi) = 1_\sigma \cap ((R - 1_\pi) \cdot NC)$   
**apply** (*rule set-eqI*)  
**apply** (*clarsimp simp: d-def p-id-def s-id-def U-def s-prod-im*)  
**by** (*metis UN-constant insert-not-empty*)

### 3.4 Multirelations and C-Lattices

Next we show that multirelations form c-lattices (Proposition 7.3) and prove further facts in this setting.

**interpretation** *mrel-c-lattice*: *c-lattice*  $1_\sigma (\cdot) 1_\pi (\parallel) (\cup) (\subseteq) (\subset) \{\} U (\cap) NC$   
**proof**

**fix**  $x y z :: ('b \times 'b \text{ set}) \text{ set}$   
**show**  $x \cdot 1_\pi \cup x \cdot NC = x \cdot U$   
**apply** (*rule set-eqI*)  
**apply** (*clarsimp simp: mr-simp*)  
**using** *UN-constant all-not-in-conv* **by** *metis*  
**show**  $1_\pi \cap (x \cup NC) = x \cdot \{\}$   
**by** (*auto simp: mr-simp*)  
**show**  $x \cdot (y \parallel z) \subseteq x \cdot y \parallel (x \cdot z)$   
**by** (*rule cl3*)  
**show**  $z \parallel z \subseteq z \implies x \parallel y \cdot z = x \cdot z \parallel (y \cdot z)$

by (*metis cl4 seq-conc-subdistr subset-antisym*)  
**show**  $x \cdot (y \cdot (z \cdot \{\})) = x \cdot y \cdot (z \cdot \{\})$   
 by (*metis cl5*)  
**show**  $x \cdot \{\} \cdot z = x \cdot \{\}$   
 by (*clarsimp simp: mr-simp*)  
**show**  $1_\sigma \parallel 1_\sigma = 1_\sigma$   
 by (*auto simp: mr-simp*)  
**show**  $x \cdot 1_\pi \parallel 1_\sigma \cdot y = x \cdot 1_\pi \parallel y$   
 by (*metis cl8-var d-def-expl*)  
**show**  $x \cap 1_\sigma \cdot 1_\pi \parallel 1_\sigma = x \cap 1_\sigma$   
 by (*auto simp: mr-simp*)  
**show**  $x \cap NC \cdot 1_\pi \parallel 1_\sigma = 1_\sigma \cap (x \cap NC \cdot NC)$   
 by (*metis Int-Diff cl10-var d-def-expl*)  
**show**  $x \cap NC \cdot 1_\pi \parallel NC = x \cap NC \cdot NC$   
 apply (*rule set-eqI*)  
 apply (*clarsimp simp: d-def U-def p-id-def p-prod-def s-prod-im*)  
 apply *standard*  
 apply (*metis (no-types, lifting) UN-extend-simps(2) Un-empty*)  
**proof** –  
**fix**  $a :: 'b$  **and**  $b :: 'b$  *set*  
**assume**  $a1: \exists B. (a, B) \in x \wedge B \neq \{\} \wedge (\exists f. (\forall b \in B. f b \neq \{\}) \wedge b = (\bigcup_{x \in B. f x}))$   
**{** **fix**  $bb :: 'b$  *set*  $\Rightarrow 'b$  *set*  $\Rightarrow 'b$  *set*  $\Rightarrow ('b \Rightarrow 'b$  *set*)  $\Rightarrow 'b$   
**obtain**  $BB :: 'b$  *set* **and**  $BBa :: 'b \Rightarrow 'b$  *set* **where**  
 $ff1: (a, BB) \in x \wedge \{\} \neq BB \wedge (\forall b. b \notin BB \vee \{\} \neq BBa b) \wedge \bigcup (BBa ` BB) = b$   
 by (*metis (full-types) a1*)  
**hence**  $\forall B. (\bigcup_{b \in BB. (B :: 'b$  *set*)) =  $B$   
 by *force*  
**hence**  $\exists B Ba. B \cup Ba = b \wedge (\exists Bb. (a, Bb) \in x \wedge \{\} \neq Bb \wedge (\exists f. (bb B Ba Bb f) \notin Bb \vee \{\} = f (bb B Ba Bb f)) \wedge \bigcup (f ` Bb) = B)) \wedge \{\} \neq Ba$   
 by (*metis ff1 SUP-bot-conv(2) sup-bot.left-neutral*) }  
**thus**  $\exists B Ba. b = B \cup Ba \wedge (\exists Ba. (a, Ba) \in x \wedge Ba \neq \{\} \wedge (\exists f. (\forall b \in Ba. f b = \{\}) \wedge B = (\bigcup_{b \in Ba. f b})) \wedge Ba \neq \{\})$   
 by *metis*  
**qed**  
**qed**

The following facts from Lemma 2.2 remain to be shown.

**lemma** *p-id-assoc1*:  $(1_\pi \cdot R) \cdot S = 1_\pi \cdot (R \cdot S)$   
 by (*clarsimp simp: mr-simp*)

**lemma** *p-id-assoc2*:  $(R \cdot 1_\pi) \cdot T = R \cdot (1_\pi \cdot T)$   
 by (*auto simp add: mr-simp cong del: SUP-cong-simp, blast+*)

**lemma** *seq-conc-subdistr*:

**assumes**  $P \subseteq 1_\sigma$

**shows**  $P \cdot (S \parallel T) = (P \cdot S) \parallel (P \cdot T)$

by (*metis assms mrel-c-lattice.d-inter-r mrel-c-lattice.d-s-subid*)



**lemma** *test-s-prod-is-meet* [*simp*]:  
**assumes**  $R \subseteq 1_\sigma$   
**and**  $S \subseteq 1_\sigma$   
**shows**  $R \cdot S = R \cap S$   
**using** *assms* **by** (*auto simp: mr-simp, force+*)

**lemma** *test-p-prod-is-meet*:  
**assumes**  $R \subseteq 1_\sigma$   
**and**  $S \subseteq 1_\sigma$   
**shows**  $R \parallel S = R \cap S$   
**apply** *standard*  
**using** *assms*  
**apply** (*auto simp: mr-simp, force+*)  
**done**

**lemma** *test-multiplicativer*:  
**assumes**  $R \subseteq 1_\sigma$   
**and**  $S \subseteq 1_\sigma$   
**shows**  $(R \cap S) \cdot T = (R \cdot T) \cap (S \cdot T)$   
**using** *assms* **by** (*clarsimp simp: set-eqI mr-simp subid-aux2, force*)

Next we verify the remaining fact from Lemma 2.2; in fact it follows from the corresponding theorem of c-lattices.

**lemma** *c6*:  $R \cdot 1_\pi \subseteq 1_\pi$   
**by** (*clarsimp simp: mr-simp*)

Next we verify Lemma 3.1.

**lemma** *p-id-st*:  $R \cdot 1_\pi = \{(a, \{\}) \mid a. \exists B. (a, B) \in R\}$   
**by** (*auto simp: mr-simp*)

**lemma** *p-id-zero*:  $R \cap 1_\pi = R \cdot \{\}$   
**by** (*auto simp: mr-simp*)

**lemma** *p-id-zero-st*:  $R \cap 1_\pi = \{(a, \{\}) \mid a. (a, \{\}) \in R\}$   
**by** (*auto simp: mr-simp*)

**lemma** *s-id-st*:  $R \cap 1_\sigma = \{(a, \{a\}) \mid a. (a, \{a\}) \in R\}$   
**by** (*auto simp: mr-simp*)

**lemma** *U-seq-st*:  $(a, A) \in R \cdot U \iff (A = \{\} \wedge (a, \{\}) \in R) \vee (\exists B. B \neq \{\} \wedge (a, B) \in R)$   
**by** (*clarsimp simp: s-prod-im U-def,metis SUP-constant SUP-empty*)

**lemma** *U-par-st*:  $(a, A) \in R \parallel U \iff (\exists B. B \subseteq A \wedge (a, B) \in R)$   
**by** (*auto simp: mr-simp*)

Next we verify the relationships after Lemma 3.1.

**lemma** *s-subid-iff1*:  $R \subseteq 1_\sigma \iff R \cap 1_\sigma = R$

by *blast*

**lemma** *s-subid-iff2*:  $R \subseteq 1_\sigma \longleftrightarrow d R = R$   
by (*auto simp: mrd-simp*)

**lemma** *p-subid-iff*:  $R \subseteq 1_\pi \longleftrightarrow R \cdot 1_\pi = R$   
by (*simp add: mrel-c-lattice.term-p-subid*)

**lemma** *vec-iff1*:  
**assumes**  $\forall a. (\exists A. (a,A) \in R) \longrightarrow (\forall A. (a,A) \in R)$   
**shows**  $(R \cdot 1_\pi) \parallel U = R$   
using *assms* by (*auto simp: mr-simp*)

**lemma** *vec-iff2*:  
**assumes**  $(R \cdot 1_\pi) \parallel U = R$   
**shows**  $(\forall a. (\exists A. (a,A) \in R) \longrightarrow (\forall A. (a,A) \in R))$   
using *assms* apply (*clarsimp simp: mr-simp*)  
**proof** –  
  **fix**  $a :: 'a$  **and**  $A :: 'a$  **set** **and**  $Aa :: 'a$  **set**  
  **assume**  $a1: (a, A) \in R$   
  **obtain**  $AA :: ('a \times 'a \text{ set}) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \Rightarrow 'a \text{ set}$  **where**  
     $\forall x0\ x1\ x2. (\exists v3 \subseteq x1. (x2, v3) \in x0) = (AA\ x0\ x1\ x2 \subseteq x1 \wedge (x2, AA\ x0\ x1\ x2) \in x0)$   
  by *moura*  
  **hence**  $f2: AA\ (R \cdot 1_\pi)\ A\ a \subseteq A \wedge (a, AA\ (R \cdot 1_\pi)\ A\ a) \in R \cdot 1_\pi$   
  by (*metis a1 U-par-st assms*)  
  **hence**  $\exists aa. (a, AA\ (R \cdot 1_\pi)\ A\ a) = (aa, \{\}) \wedge (\exists A. (aa, A) \in R)$   
  by (*simp add: p-id-st*)  
  **hence**  $AA\ (R \cdot 1_\pi)\ A\ a \subseteq Aa$   
  by *blast*  
  **thus**  $(a, Aa) \in R$   
  using  $f2$  by (*metis (no-types) U-par-st assms*)  
**qed**

**lemma** *vec-iff*:  $(\forall a. (\exists A. (a,A) \in R) \longrightarrow (\forall A. (a,A) \in R)) \longleftrightarrow (R \cdot 1_\pi) \parallel U = R$   
by (*metis vec-iff1 vec-iff2*)

**lemma** *ucl-iff*:  $(\forall a\ A\ B. (a,A) \in R \wedge A \subseteq B \longrightarrow (a,B) \in R) \longleftrightarrow R \parallel U = R$   
by (*clarsimp simp: mr-simp, blast*)

**lemma** *nt-iff*:  $R \subseteq NC \longleftrightarrow R \cap NC = R$   
by *blast*

Next we provide a counterexample for the final paragraph of Section 3.

**lemma**  $1_\sigma \cap R \cdot U = R$   
**nitpick**  
**oops**

Next we present a counterexample for vectors mentioned before Lemma 9.3.

**lemma**  $d (d R \cdot U) \cdot (d S \cdot U) \cdot U = (d R \cdot U) \cdot (d S \cdot U)$   
**nitpick**  
**oops**

Next we prove Tarski' rule (Lemma 9.3).

**lemma** *tarski-aux*:  
**assumes**  $R - 1_\pi \neq \{\}$   
**and**  $(a, A) \in NC$   
**shows**  $(a, A) \in NC \cdot ((R - 1_\pi) \cdot NC)$   
**proof** –  
**have**  $(\exists B. B \neq \{\} \wedge (\forall x \in B. (x, \{x\}) \in d (R - 1_\pi)))$   
**using** *assms(1)* **by** (*auto simp: mrd-simp*)  
**hence**  $(\exists B. B \neq \{\} \wedge (\forall x \in B. (x, \{x\}) \in d (R - 1_\pi))) \wedge A \neq \{\}$   
**using** *assms(2)* **by** (*clarsimp simp: mr-simp*)  
**hence**  $(\exists B. B \neq \{\} \wedge (\exists f. (\forall x \in B. (x, \{x\}) \in d (R - 1_\pi) \wedge f x \neq \{\})) \wedge A =$   
 $\bigcup ((\lambda x. (f x)) ` B))$   
**by** (*metis UN-constant*)  
**hence**  $(a, A) \in NC \cdot (d (R - 1_\pi) \cdot NC)$   
**by** (*clarsimp simp: mrd-simp metis*)  
**thus** *?thesis*  
**by** (*clarsimp simp: mrd-simp, metis UN-constant*)  
**qed**

**lemma** *tarski*:  
**assumes**  $R - 1_\pi \neq \{\}$   
**shows**  $NC \cdot ((R - 1_\pi) \cdot NC) = NC$   
**by** *standard (simp add: U-def p-id-def s-prod-im, force, metis assms tarski-aux subrelI)*

Next we verify the assumptions of Proposition 9.8.

**lemma** *d-assoc1*:  $d R \cdot (S \cdot T) = (d R \cdot S) \cdot T$   
**by** (*metis d-def-expl mrel-c-lattice.d-def mrel-c-lattice.d-sub-id-ax test-assoc2*)

**lemma** *d-meet-distr-var*:  $(d R \cap d S) \cdot T = (d R \cdot T) \cap (d S \cdot T)$   
**by** (*auto simp: mrd-simp*)

Lemma 10.5.

**lemma**  $((R \cap 1_\sigma) \cdot (S \cap 1_\sigma)) \cdot 1_\pi = ((R \cap 1_\sigma) \cdot 1_\pi) \cdot ((S \cap 1_\sigma) \cdot 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $d ((R \cdot 1_\pi) \cdot (S \cdot 1_\pi)) = d (R \cdot 1_\pi) \cdot d (S \cdot 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $((R \cap 1_\sigma) \cdot (S \cap 1_\sigma)) \cdot U = ((R \cap 1_\sigma) \cdot U) \cdot ((S \cap 1_\sigma) \cdot U)$   
**nitpick**  
**oops**

**lemma**  $d(((R \cdot 1_\pi) \parallel U) \cdot ((S \cdot 1_\pi) \parallel U)) = d((R \cdot 1_\pi) \parallel U) \cdot d((S \cdot 1_\pi) \parallel U)$   
**nitpick**  
**oops**

**lemma**  $((R \cdot 1_\pi) \cdot (S \cdot 1_\pi)) \parallel U = ((R \cdot 1_\pi) \parallel U) \cdot ((S \cdot 1_\pi) \parallel U)$   
**nitpick**  
**oops**

**lemma**  $((R - 1_\pi) \cap 1_\sigma) \cdot ((S - 1_\pi) \cap 1_\sigma) \cdot 1_\pi = (((R - 1_\pi) \cap 1_\sigma) \cdot 1_\pi) \cdot (((S - 1_\pi) \cap 1_\sigma) \cdot 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $d(((R - 1_\pi) \cdot 1_\pi) \cdot ((S - 1_\pi) \cdot 1_\pi)) = d((R - 1_\pi) \cdot 1_\pi) \cdot d((S - 1_\pi) \cdot 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $((R - 1_\pi) \cap 1_\sigma) \cdot ((S - 1_\pi) \cap 1_\sigma) \cdot NC = (((R - 1_\pi) \cap 1_\sigma) \cdot NC) \cdot (((S - 1_\pi) \cap 1_\sigma) \cdot NC)$   
**apply** (*auto simp: U-def p-id-def s-id-def s-prod-im*)  
**defer**  
**nitpick**  
**oops**

**lemma**  $d((((R - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot (((S - 1_\pi) \cdot 1_\pi) \parallel NC)) = d(((R - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot d(((S - 1_\pi) \cdot 1_\pi) \parallel NC)$   
**apply** (*simp add: U-def p-id-def s-prod-im p-prod-def d-def*)  
**nitpick**  
**oops**

**lemma**  $((R - 1_\pi) \cdot 1_\pi) \cdot ((S - 1_\pi) \cdot 1_\pi) \parallel NC = (((R - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot (((S - 1_\pi) \cdot 1_\pi) \parallel NC)$   
**nitpick**  
**oops**

**lemma**  $((((R - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot (((S - 1_\pi) \cdot 1_\pi) \parallel NC)) \cdot 1_\pi = (((R - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot 1_\pi \cdot (((S - 1_\pi) \cdot 1_\pi) \parallel NC) \cdot 1_\pi$   
**nitpick**  
**oops**

### 3.5 Terminal and Nonterminal Elements

Lemma 11.4

**lemma**  $(R \cdot S) \cdot \{\} = (R \cdot \{\}) \cdot (S \cdot \{\})$   
**nitpick**  
**oops**

**lemma**  $(R \cdot S) - 1_\pi = (R - 1_\pi) \cdot (S - 1_\pi)$

**apply** (*auto simp: s-prod-im p-id-def*)  
**nitpick**  
**oops**

**lemma**  $(R \parallel S) - 1_\pi = (R - 1_\pi) \parallel (S - 1_\pi)$   
**nitpick**  
**oops**

Lemma 11.8.

**lemma**  $((R \cdot 1_\pi) \cdot (S - 1_\pi)) - 1_\pi = (R \cdot 1_\pi) \cdot (S - 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $((S - 1_\pi) \cdot (R \cdot 1_\pi)) - 1_\pi = (S - 1_\pi) \cdot (R \cdot 1_\pi)$   
**nitpick**  
**oops**

**lemma**  $((R \cdot 1_\pi) \parallel (S - 1_\pi)) \cdot 1_\pi = (R \cdot 1_\pi) \parallel (S - 1_\pi)$   
**nitpick**  
**oops**

Lemma 11.10.

**lemma**  $R \cdot \{\} \subseteq S \cdot \{\} \implies (R \cdot T) \cdot \{\} \subseteq (S \cdot T) \cdot \{\}$   
**nitpick**  
**oops**

**lemma**  $R - 1_\pi \subseteq S - 1_\pi \implies (R \parallel T) - 1_\pi \subseteq (S \parallel T) - 1_\pi$   
**nitpick**  
**oops**

**lemma**  $R - 1_\pi \subseteq S - 1_\pi \implies (T \cdot R) - 1_\pi \subseteq (T \cdot S) - 1_\pi$   
**apply** (*auto simp: p-id-def s-prod-im*)  
**nitpick**  
**oops**

Corollary 11.12

**lemma**  $R \cdot \{\} = S \cdot \{\} \implies (R \cdot T) \cdot \{\} = (S \cdot T) \cdot \{\}$   
**nitpick**  
**oops**

**lemma**  $R - 1_\pi = S - 1_\pi \implies (R \parallel T) - 1_\pi = (S \parallel T) - 1_\pi$   
**nitpick**  
**oops**

**lemma**  $R - 1_\pi = S - 1_\pi \implies (T \cdot R) - 1_\pi = (T \cdot S) - 1_\pi$   
**apply** (*auto simp: p-id-def s-prod-im*)  
**nitpick**  
**oops**

### 3.6 Multirelations, Proto-Quantales and Iteration

**interpretation** *mrel-proto-quantale*: *proto-quantale*  $1_\sigma$   $(\cdot)$  *Inter Union*  $(\cap)$   $(\subseteq)$   
 $(\subset)$   $(\cup)$   $\{\}$   $U$   
**by** (*unfold-locales*, *auto simp*: *mr-simp*)

We reprove Corollary 13.2. because Isabelle does not pick it up from the quantale level.

**lemma** *iso-prop: mono*  $(\lambda X. S \cup R \cdot X)$   
**by** (*rule monoI*, (*clarsimp simp*: *mr-simp*), *blast*)

**lemma** *gfp-lfp-prop*:  $\text{gfp } (\lambda X. R \cdot X) \cup \text{lfp } (\lambda X. S \cup R \cdot X) \subseteq \text{gfp } (\lambda X. S \cup R \cdot X)$   
**by** (*simp add*: *lfp-le-gfp* *gfp-mono* *iso-prop*)

### 3.7 Further Counterexamples

Lemma 14,1. and 14.2

**lemma**  $R \parallel R \subseteq R$   
**nitpick**  
**oops**

**lemma**  $R \subseteq R \parallel S$   
**nitpick**  
**oops**

**lemma**  $R \parallel S \cap R \parallel T \subseteq R \parallel (S \cap T)$   
**nitpick**  
**oops**

**lemma**  $R \cdot (S \parallel T) = (R \cdot S) \parallel (R \cdot T)$   
**nitpick**  
**oops**

**lemma**  $R \cdot (S \cdot T) \subseteq (R \cdot S) \cdot T$   
**apply** (*auto simp*: *s-prod-im*)  
**nitpick**  
**oops**

**lemma**  $\llbracket R \parallel R = R; S \parallel S = S; T \parallel T = T \rrbracket \implies R \cdot (S \parallel T) = (R \cdot S) \parallel (R \cdot T)$   
**nitpick**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \rrbracket \implies R \cdot S \subseteq R \parallel S$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \rrbracket \implies R \parallel S \subseteq R \cdot S$

**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; T \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \cup T \rrbracket \implies (R \parallel S) \cdot T \subseteq R \parallel (S \cdot T)$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; T \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \cup T \rrbracket \implies R \parallel (S \cdot T) \subseteq (R \parallel S) \cdot T$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; T \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \cup T \rrbracket \implies R \cdot (S \parallel T) \subseteq (R \cdot S) \parallel T$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; T \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \cup T \rrbracket \implies (R \cdot S) \parallel T \subseteq R \cdot (S \parallel T)$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \rrbracket \implies (R \parallel S) \cdot (R \parallel S) \subseteq (R \cdot R) \parallel (S \cdot S)$   
**quickcheck**  
**oops**

**lemma**  $\llbracket R \neq \{\}; S \neq \{\}; \forall a. (a, \{\}) \notin R \cup S \rrbracket \implies (R \cdot R) \parallel (S \cdot S) \subseteq (R \parallel S) \cdot (R \parallel S)$   
**quickcheck**  
**oops**

### 3.8 Relationship with Up-Closed Multirelations

We now define Parikh's sequential composition.

**definition**  $s\text{-prod-pa} :: 'a \text{ mrel} \Rightarrow 'a \text{ mrel} \Rightarrow 'a \text{ mrel}$  (**infixl**  $\otimes$  70) **where**  
 $R \otimes S = \{(a, A). (\exists B. (a, B) \in R \wedge (\forall b \in B. (b, A) \in S))\}$

We show that Parikh's definition doesn't preserve up-closure.

**lemma**  $up\text{-closed-prop}: ((R \parallel U) \cdot (S \parallel U)) \parallel U = (R \parallel U) \cdot (S \parallel U)$   
**apply** (*auto simp: p-prod-def s-prod-pa-def U-def*)  
**nitpick**  
**oops**

Lemma 15.1.

**lemma**  $onelem: (R \cdot S) \parallel U \subseteq R \otimes (S \parallel U)$   
**by** (*auto simp: s-prod-im p-prod-def U-def s-prod-pa-def*)

**lemma** *twolem*:  $R \otimes (S \parallel U) \subseteq (R \cdot S) \parallel U$

**proof** *clarify*

**fix**  $a A$

**assume**  $(a, A) \in R \otimes (S \parallel U)$

**hence**  $\exists B. (a, B) \in R \wedge (\forall b \in B. (b, A) \in S \parallel U)$

**by** (*auto simp: s-prod-pa-def*)

**hence**  $\exists B. (a, B) \in R \wedge (\forall b \in B. \exists C. C \subseteq A \wedge (b, C) \in S)$

**by** (*metis U-par-st*)

**hence**  $\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. f b \subseteq A \wedge (b, f b) \in S))$

**by** *metis*

**hence**  $\exists C. C \subseteq A \wedge (\exists B. (a, B) \in R \wedge (\exists f. (\forall b \in B. (b, f b) \in S) \wedge C = \bigcup ((\lambda x. f x) \text{ ` } B)))$

**by** *clarsimp blast*

**hence**  $\exists C. C \subseteq A \wedge (a, C) \in R \cdot S$

**by** (*clarsimp simp: mr-simp*)

**thus**  $(a, A) \in (R \cdot S) \parallel U$

**by** (*simp add: U-par-st*)

**qed**

**lemma** *pe-pa-sim*:  $(R \cdot S) \parallel U = R \otimes (S \parallel U)$

**by** (*metis antisym onelem twolem*)

**lemma** *pe-pa-sim-var*:  $((R \parallel U) \cdot (S \parallel U)) \parallel U = (R \parallel U) \otimes (S \parallel U)$

**by** (*simp add: mrel-prototrioid.mult-assoc pe-pa-sim*)

**lemma** *pa-assoc1*:  $((R \parallel U) \otimes (S \parallel U)) \otimes (T \parallel U) \subseteq (R \parallel U) \otimes ((S \parallel U) \otimes (T \parallel U))$

**by** (*clarsimp simp: p-prod-def s-prod-pa-def U-def, metis*)

The converse direction of associativity remains to be proved.

Corollary 15.3.

**lemma** *up-closed-par-is-meet*:  $(R \parallel U) \parallel (S \parallel U) = (R \parallel U) \cap (S \parallel U)$

**by** (*auto simp: mr-simp*)

**end**

## References

- [1] H. Furusawa and G. Struth. Concurrent dynamic algebra. *ACM Transactions on Computational Logic*, 2015. (In Press).
- [2] H. Furusawa and G. Struth. Taming multirelations. *CoRR*, abs/1501.05147, 2015.
- [3] D. Peleg. Concurrent dynamic logic. *J. ACM*, 34(2):450–479, 1987.