

Catoids, Categories, Groupoids

Georg Struth

May 13, 2024

Abstract

This AFP entry formalises catoids, which are generalisations of single-set categories, and groupoids. More specifically, in catoids, the partial composition of arrows in a category is generalised to a multi-operation, which sends pairs of elements to sets of elements, and the definedness condition of arrow composition – two arrows can be composed if and only the target of the first matches the source of the second – is relaxed. Beyond a library of basic laws for catoids, single-set categories and groupoids, I formalise the facts that every catoid can be lifted to a modal powerset quantale, that every groupoid can be lifted to a Dedekind quantale and to power set relation algebras, a special case of a famous result of Jónsson and Tarski. Finally, I show that single-set categories are equivalent to a standard axiomatisation of categories based on a set of objects and a set of arrows, and compare catoids with related structures such as multimonoid and relational monoids (monoids in the monoidal category \mathbf{Rel}).

Contents

1	Introductory Remarks	2
2	Catoids	3
2.1	Multimagmas	3
2.2	Multisemigroups	4
2.3	st-Multimagmas	4
2.4	Catoids	10
2.5	Locality	13
2.6	From partial magmas to single-set categories.	15
2.7	Morphisms of multimagmas and lr-multimagmas	16
2.8	Relationship with categories	17
2.9	A Mac Lane style variant	18
2.10	Product of catoids	19

3	Groupoids	21
3.1	st-Multigroupoids	21
3.2	Groupoids	25
3.3	Axioms of relation algebra	28
4	Lifting catoids to modal powerset quantales	29
5	Lifting groupoids to powerset Dedekind quantales and powerset relation algebras	32
6	Multimonoids	33
6.1	Unital multimagnas	33
6.2	Multimonoids	36
6.3	Multimonoids and catoids	37
6.4	From multimonoids to categories	38
6.5	Multimonoids and relational monoids	39

1 Introductory Remarks

These Isabelle theories formalise results on catoids from [4, 2, 6] and groupoids from [3]. Catoids generalise single-set categories, as they can be found in Chapter XII of Mac Lane’s book [8]. One particular result, namely that catoids can be lifted to (power set) relation algebras, is due to Jónsson and Tarski [7].

A wide-ranging formalisation of category theory based on single-set categories formalised as locales can already be found in the AFP [9]. The present type-class-based alternative might lend itself to a similar programme.

The multioperation $X \times X \rightarrow \mathcal{P}X$ in the definition of catoids is obviously isomorphic to a ternary relation $X \rightarrow X \rightarrow X \rightarrow 2$. Simple mathematical components for relational monoids, which are isomorphic (as categories with suitable morphisms) to catoids, can already be found in the AFP [5]. At this stage, I do not integrate the two components. They use different formalisations of quantales with Isabelle, which remain to be consolidated.

Catoids and groupoids admit many models. Those of catoids range from shuffle monoids and generalised effect algebras to base algebras of incidence and matrix algebras [6], whereas groupoids are so ubiquitous in mathematics that some mathematicians have argued for interchanging their names with groups, see [1] for a brief history, which goes decades beyond that of category theory.

The mathematical components in this AFP entry are also stepping stones towards the formalisation of (ω, p) -catoids, strict (ω, p) -categories and (ω, p) -quantales. Components for these structures will feature in a separate AFP

entry. They contribute to a larger programme on the formalisation of higher rewriting techniques with proof assistants.

I am grateful for a fellowship at the Collegium de Lyon, Institute for Advanced Study, where this formalisation work has been done.

2 Catoids

```
theory Catoid
  imports Main
```

```
begin
```

2.1 Multimagnas

Multimagnas are sets equipped with multioperations. Multioperations are isomorphic to ternary relations.

```
class multimagma =
  fixes mcomp :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a set (infixl  $\odot$  70)
```

```
begin
```

I introduce notation for the domain of definition of the multioperation.

```
abbreviation  $\Delta$  x y  $\equiv$  (x  $\odot$  y  $\neq$  {})
```

I extend the multioperation to powersets

```
definition conv :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set (infixl  $\star$  70) where
  X  $\star$  Y = ( $\bigcup x \in X. \bigcup y \in Y. x \odot y$ )
```

```
lemma conv-exp: X  $\star$  Y = {z.  $\exists x y. z \in x \odot y \wedge x \in X \wedge y \in Y$ }
  unfolding conv-def by fastforce
```

```
lemma conv-exp2: (z  $\in$  X  $\star$  Y) = ( $\exists x y. z \in x \odot y \wedge x \in X \wedge y \in Y$ )
  by (simp add: multimagma.conv-exp)
```

```
lemma conv-distl: X  $\star$   $\bigcup \mathcal{Y}$  = ( $\bigcup Y \in \mathcal{Y}. X \star Y$ )
  unfolding conv-def by blast
```

```
lemma conv-distr:  $\bigcup \mathcal{X} \star Y$  = ( $\bigcup X \in \mathcal{X}. X \star Y$ )
  unfolding conv-def by blast
```

```
lemma conv-distl-small: X  $\star$  (Y  $\cup$  Z) = X  $\star$  Y  $\cup$  X  $\star$  Z
  unfolding conv-def by blast
```

```
lemma conv-distr-small: (X  $\cup$  Y)  $\star$  Z = X  $\star$  Z  $\cup$  Y  $\star$  Z
  unfolding conv-def by blast
```

lemma *conv-isol*: $X \subseteq Y \implies Z \star X \subseteq Z \star Y$
using *conv-exp2* **by** *fastforce*

lemma *conv-isor*: $X \subseteq Y \implies X \star Z \subseteq Y \star Z$
using *conv-exp2* **by** *fastforce*

lemma *conv-atom* [*simp*]: $\{x\} \star \{y\} = x \odot y$
by (*simp add: conv-def*)

end

2.2 Multisemigroups

Multisemigroups are associative multimagnas.

class *multisemigroup* = *multimagma* +
assumes *assoc*: $(\bigcup v \in y \odot z. x \odot v) = (\bigcup v \in x \odot y. v \odot z)$

begin

lemma *assoc-exp*: $(\exists v. w \in x \odot v \wedge v \in y \odot z) = (\exists v. v \in x \odot y \wedge w \in v \odot z)$
using *assoc* **by** *blast*

lemma *assoc-var*: $\{x\} \star (y \odot z) = (x \odot y) \star \{z\}$
unfolding *conv-def assoc-exp* **using** *local.assoc* **by** *force*

Associativity extends to powersets.

lemma *conv-assoc*: $X \star (Y \star Z) = (X \star Y) \star Z$
unfolding *conv-exp* **using** *assoc-exp* **by** *fastforce*

end

2.3 st-Multimagnas

I equip multimagnas with source and target maps.

class *st-op* =
fixes *src* :: $'a \Rightarrow 'a$ (σ)
and *tgt* :: $'a \Rightarrow 'a$ (τ)

class *st-multimagma* = *multimagma* + *st-op* +
assumes *Dst*: $x \odot y \neq \{\} \implies \tau x = \sigma y$
and *s-absorb* [*simp*]: $\sigma x \odot x = \{x\}$
and *t-absorb* [*simp*]: $x \odot \tau x = \{x\}$

The following sublocale proof sets up opposition/duality.

sublocale *st-multimagma* \subseteq *stopp*: *st-multimagma* $\lambda x y. y \odot x$ *tgt src*
rewrites *stopp.conv* $X Y = Y \star X$
by (*unfold-locales, auto simp add: local.Dst multimagma.conv-def*)

lemma (in *st-multimagma*) *ts-compat* [*simp*]: $\tau (\sigma x) = \sigma x$
by (*simp add: Dst*)

lemma (in *st-multimagma*) *ss-idem* [*simp*]: $\sigma (\sigma x) = \sigma x$
by (*metis local.stopp.ts-compat local.ts-compat*)

lemma (in *st-multimagma*) *st-fix*: $(\tau x = x) = (\sigma x = x)$
proof

assume *h1*: $\tau x = x$
hence $\sigma x = \sigma (\tau x)$
by *simp*
also have $\dots = x$
by (*metis h1 local.stopp.ts-compat*)
finally show $\sigma x = x$.

next

assume *h2*: $\sigma x = x$
hence $\tau x = \tau (\sigma x)$
by *simp*
also have $\dots = x$
by (*metis h2 ts-compat*)
finally show $\tau x = x$.

qed

lemma (in *st-multimagma*) *st-eq1*: $\sigma x = x \implies \sigma x = \tau x$
by (*simp add: local.stopp.st-fix*)

lemma (in *st-multimagma*) *st-eq2*: $\tau x = x \implies \sigma x = \tau x$
by (*simp add: local.stopp.st-fix*)

I extend source and target operations to powersets by taking images.

abbreviation (in *st-op*) *Src* :: 'a set \Rightarrow 'a set **where**
Src \equiv *image* σ

abbreviation (in *st-op*) *Tgt* :: 'a set \Rightarrow 'a set **where**
Tgt \equiv *image* τ

Fixpoints of source and target maps model source and target elements.
These correspond to units.

abbreviation (in *st-op*) *sfix* :: 'a set **where**
sfix \equiv $\{x. \sigma x = x\}$

abbreviation (in *st-op*) *tfix* :: 'a set **where**
tfix \equiv $\{x. \tau x = x\}$

lemma (in *st-multimagma*) *st-mm-rfix* [*simp*]: *tfix* = *stopp.sfix*
by *simp*

lemma (in *st-multimagma*) *st-fix-set*: $\{x. \sigma x = x\} = \{x. \tau x = x\}$
using *local.st-fix* **by** *presburger*

lemma (in *st-multimagma*) *stfix-set*: $sfix = tfix$
 using *local.st-fix-set* **by** *blast*

lemma (in *st-multimagma*) *sfix-im*: $sfix = Src\ UNIV$
by (*smt* (*verit*, *best*) *Collect-cong full-SetCompr-eq local.ss-idem*)

lemma (in *st-multimagma*) *tfix-im*: $tfix = Tgt\ UNIV$
 using *local.stopp.sfix-im* **by** *blast*

lemma (in *st-multimagma*) *ST-im*: $Src\ UNIV = Tgt\ UNIV$
 using *local.sfix-im local.stfix-set local.tfix-im* **by** *presburger*

Source and target elements are "orthogonal" idempotents.

lemma (in *st-multimagma*) *s-idem* [*simp*]: $\sigma\ x \odot \sigma\ x = \{\sigma\ x\}$
proof–
have $\{\sigma\ x\} = \sigma\ x \odot \tau\ (\sigma\ x)$
using *local.t-absorb* **by** *presburger*
also have $\dots = \sigma\ x \odot \sigma\ x$
by *simp*
finally show *?thesis*..
qed

lemma (in *st-multimagma*) *s-ortho*:
 $\Delta\ (\sigma\ x)\ (\sigma\ y) \implies \sigma\ x = \sigma\ y$
proof–
assume $\Delta\ (\sigma\ x)\ (\sigma\ y)$
hence $\tau\ (\sigma\ x) = \sigma\ (\sigma\ y)$
using *local.Dst* **by** *blast*
thus *?thesis*
by *simp*
qed

lemma (in *st-multimagma*) *s-ortho-iff*: $\Delta\ (\sigma\ x)\ (\sigma\ y) = (\sigma\ x = \sigma\ y)$
 using *local.s-ortho* **by** *auto*

lemma (in *st-multimagma*) *st-ortho-iff*: $\Delta\ (\sigma\ x)\ (\tau\ y) = (\sigma\ x = \tau\ y)$
 using *local.Dst* **by** *fastforce*

lemma (in *st-multimagma*) *s-ortho-id*: $(\sigma\ x) \odot (\sigma\ y) = (if\ (\sigma\ x = \sigma\ y)\ then\ \{\sigma\ x\}\ else\ \{\})$
 using *local.s-ortho-iff* **by** *auto*

lemma (in *st-multimagma*) *s-absorb-var*: $(\sigma\ y \neq \sigma\ x) = (\sigma\ y \odot x = \{\})$
 using *local.Dst* **by** *force*

lemma (in *st-multimagma*) *s-absorb-var2*: $(\sigma\ y = \sigma\ x) = (\sigma\ y \odot x \neq \{\})$
 using *local.s-absorb-var* **by** *blast*

lemma (in *st-multimagma*) *s-absorb-var3*: $(\sigma y = \sigma x) = \Delta (\sigma x) y$
by (*metis local.s-absorb-var*)

lemma (in *st-multimagma*) *s-assoc*: $\{\sigma x\} \star (\sigma y \odot z) = (\sigma x \odot \sigma y) \star \{z\}$
proof –
{fix *a*
have $(a \in \{\sigma x\} \star (\sigma y \odot z)) = (\exists b. a \in \sigma x \odot b \wedge b \in \sigma y \odot z)$
by (*simp add: local.conv-exp2*)
also have $\dots = (\exists b. a \in \sigma x \odot b \wedge b \in \sigma y \odot z \wedge \sigma y = \sigma z)$
using *local.s-absorb-var* **by** *auto*
also have $\dots = (\exists b. a \in \sigma x \odot b \wedge b \in \sigma y \odot z \wedge \sigma y = \sigma z \wedge \sigma x = \sigma y)$
using *local.stopp.Dst* **by** *fastforce*
also have $\dots = (\exists b. b \in \sigma x \odot \sigma y \wedge a \in b \odot z \wedge \sigma y = \sigma z \wedge \sigma x = \sigma y)$
by *fastforce*
also have $\dots = (\exists b. b \in \sigma x \odot \sigma y \wedge a \in b \odot z)$
by (*metis equals0D local.s-absorb-var3 local.s-idem singleton-iff*)
also have $\dots = (a \in (\sigma x \odot \sigma y) \star \{z\})$
using *local.conv-exp2* **by** *auto*
finally have $(a \in \{\sigma x\} \star (\sigma y \odot z)) = (a \in (\sigma x \odot \sigma y) \star \{z\}).$
thus *?thesis*
by *blast*
qed

lemma (in *st-multimagma*) *sfix-absorb-var* [*simp*]: $(\bigcup e \in \text{sfix}. e \odot x) = \{x\}$
apply *safe*
apply (*metis local.s-absorb local.s-absorb-var2 singletonD*)
by (*smt (verit, del-insts) UNIV-I UN-iff imageI local.s-absorb local.sfix-im singletonI*)

lemma (in *st-multimagma*) *tfix-absorb-var*: $(\bigcup e \in \text{tfix}. x \odot e) = \{x\}$
using *local.stopp.sfix-absorb-var* **by** *presburger*

lemma (in *st-multimagma*) *st-comm*: $\tau x \odot \sigma y = \sigma y \odot \tau x$
using *local.Dst* **by** *fastforce*

lemma (in *st-multimagma*) *s-weak-twisted*: $(\bigcup u \in x \odot y. \sigma u \odot x) \subseteq x \odot \sigma y$
by (*safe, metis empty-iff insertI1 local.Dst local.s-absorb local.t-absorb*)

lemma (in *st-multimagma*) *s-comm*: $\sigma x \odot \sigma y = \sigma y \odot \sigma x$
using *local.Dst* **by** *force*

lemma (in *st-multimagma*) *s-export* [*simp*]: $\text{Src} (\sigma x \odot y) = \sigma x \odot \sigma y$
using *local.Dst* **by** *force*

lemma (in *st-multimagma*) *st-prop*: $(\tau x = \sigma y) = \Delta (\tau x) (\sigma y)$
by (*metis local.stopp.s-absorb-var2 local.stopp.st-comm local.ts-compat*)

lemma (in *st-multimagma*) *weak-local-var*: $\tau x \odot \sigma y = \{\} \implies x \odot y = \{\}$
using *local.Dst local.st-prop* **by** *auto*

The following facts hold by duality.

lemma (in *st-multimagma*) *st-compat*: $\sigma (\tau x) = \tau x$
by *simp*

lemma (in *st-multimagma*) *tt-idem*: $\tau (\tau x) = \tau x$
by *simp*

lemma (in *st-multimagma*) *t-idem*: $\tau x \odot \tau x = \{\tau x\}$
by *simp*

lemma (in *st-multimagma*) *t-weak-twisted*: $(\bigcup u \in y \odot x. x \odot \tau u) \subseteq \tau y \odot x$
using *local.stopp.s-weak-twisted* by *auto*

lemma (in *st-multimagma*) *t-comm*: $\tau x \odot \tau y = \tau y \odot \tau x$
by (*simp add: stopp.s-comm*)

lemma (in *st-multimagma*) *t-export*: $\text{image } \tau (x \odot \tau y) = \tau x \odot \tau y$
by *simp*

lemma (in *st-multimagma*) *tt-comp-prop*: $\Delta (\tau x) (\tau y) = (\tau x = \tau y)$
using *local.stopp.s-ortho-iff* by *force*

The set of all sources (and targets) are units at powerset level.

lemma (in *st-multimagma*) *conv-uns [simp]*: $\text{sfix} \star X = X$

proof–

{**fix** *a*
 have $(a \in \text{sfix} \star X) = (\exists b \in \text{sfix}. \exists c \in X. a \in b \odot c)$
 by (*meson local.conv-exp2*)
 also have $\dots = (\exists b. \exists c \in X. \sigma b = b \wedge a \in b \odot c)$
 by *blast*
 also have $\dots = (\exists b. \exists c \in X. a \in \sigma b \odot c)$
 by (*metis local.ss-idem*)
 also have $\dots = (\exists c \in X. a \in \sigma c \odot c)$
 by (*metis empty-iff local.s-absorb-var*)
 also have $\dots = (a \in X)$
 by *auto*
 finally have $(a \in \text{sfix} \star X) = (a \in X).$ }
 thus *?thesis*
 by *blast*

qed

lemma (in *st-multimagma*) *conv-unt*: $X \star \text{tfix} = X$
using *stopp.conv-uns* by *blast*

I prove laws of modal powerset quantales.

lemma (in *st-multimagma*) *Src-exp*: $\text{Src } X = \{\sigma x \mid x. x \in X\}$
by (*simp add: Setcompr-eq-image*)

lemma (in *st-multimagma*) *ST-compatible [simp]*: $\text{Src } (\text{Tgt } X) = \text{Tgt } X$

unfolding *image-def* **by** *fastforce*

lemma (in *st-multimagma*) *TS-compat*: $Tgt (Src X) = Src X$
by (*meson local.stopp.ST-compat*)

lemma (in *st-multimagma*) *Src-absorp* [*simp*]: $Src X \star X = X$
proof–
 {**fix** *a*
 have $(a \in Src X \star X) = (\exists b \in Src X. \exists c \in X. a \in b \odot c)$
 using *local.conv-exp2* **by** *auto*
 also have $\dots = (\exists b \in X. \exists c \in X. a \in \sigma b \odot c)$
 by *blast*
 also have $\dots = (\exists c \in X. a \in \sigma c \odot c)$
 by (*metis empty-iff local.s-absorb-var*)
 also have $\dots = (a \in X)$
 by *simp*
 finally have $(a \in Src X \star X) = (a \in X).$ }
 thus ?thesis
 by *force*
qed

lemma (in *st-multimagma*) *Tgt-absorp*: $X \star Tgt X = X$
by *simp*

lemma (in *st-multimagma*) *Src-Sup-pres*: $Src (\bigcup \mathcal{X}) = (\bigcup X \in \mathcal{X}. Src X)$
unfolding *Src-exp* **by** *auto*

lemma (in *st-multimagma*) *Tgt-Sup-pres*: $Tgt (\bigcup \mathcal{X}) = (\bigcup X \in \mathcal{X}. Tgt X)$
by *blast*

lemma (in *st-multimagma*) *ST-comm*: $Src X \star Tgt Y = Tgt Y \star Src X$
proof–
 {**fix** *a*
 have $(a \in Src X \star Tgt Y) = (\exists b \in Src X. \exists c \in Tgt Y. a \in b \odot c)$
 using *local.conv-exp2* **by** *auto*
 also have $\dots = (\exists b \in X. \exists c \in Y. a \in \sigma b \odot \tau c)$
 by *auto*
 also have $\dots = (\exists b \in X. \exists c \in Y. a \in \tau c \odot \sigma b)$
 using *local.st-comm* **by** *auto*
 also have $\dots = (a \in Tgt Y \star Src X)$
 using *multimagma.conv-exp2* **by** *fastforce*
 finally have $(a \in Src X \star Tgt Y) = (a \in Tgt Y \star Src X).$ }
 thus ?thesis
 by *force*
qed

lemma (in *st-multimagma*) *Src-comm*: $Src X \star Src Y = Src Y \star Src X$
by (*metis local.ST-comm local.TS-compat*)

lemma (in *st-multimagma*) *Tgt-comm*: $Tgt\ X \star Tgt\ Y = Tgt\ Y \star Tgt\ X$
 using *local.stopp.Src-comm* **by** *presburger*

lemma (in *st-multimagma*) *Src-subid*: $Src\ X \subseteq sfix$
by *force*

lemma (in *st-multimagma*) *Tgt-subid*: $Tgt\ X \subseteq tfix$
 using *local.stopp.Src-subid* **by** *presburger*

lemma (in *st-multimagma*) *Src-export* [*simp*]: $Src\ (Src\ X \star Y) = Src\ X \star Src\ Y$
proof–
 {**fix** *a*
 have $(a \in Src\ (Src\ X \star Y)) = (\exists b \in Src\ X \star Y. a = \sigma\ b)$
 by (*simp add: image-iff*)
 also have $\dots = (\exists b. \exists c \in Src\ X. \exists d \in Y. a = \sigma\ b \wedge b \in c \odot d)$
 by (*meson local.conv-exp2*)
 also have $\dots = (\exists b. \exists c \in X. \exists d \in Y. a = \sigma\ b \wedge b \in \sigma\ c \odot d)$
 by *simp*
 also have $\dots = (\exists c \in X. \exists d \in Y. a \in Src\ (\sigma\ c \odot d))$
 by (*metis (mono-tags, lifting) image-iff*)
 also have $\dots = (\exists c \in X. \exists d \in Y. a \in \sigma\ c \odot \sigma\ d)$
 by *auto*
 also have $\dots = (\exists c \in Src\ X. \exists d \in Src\ Y. a \in c \odot d)$
 by *force*
 also have $\dots = (a \in Src\ X \star Src\ Y)$
 using *local.conv-exp2* **by** *auto*
 finally have $(a \in Src\ (Src\ X \star Y)) = (a \in Src\ X \star Src\ Y).$
 thus ?thesis
 by *force*
qed

lemma (in *st-multimagma*) *Tgt-export* [*simp*]: $Tgt\ (X \star Tgt\ Y) = Tgt\ X \star Tgt\ Y$
by *simp*

Locality implies st-locality, which is the composition pattern of categories.

lemma (in *st-multimagma*) *locality*:
 assumes *src-local*: $Src\ (x \odot \sigma\ y) \subseteq Src\ (x \odot y)$
 and *tgt-local*: $Tgt\ (\tau\ x \odot y) \subseteq Tgt\ (x \odot y)$
 shows $\Delta\ x\ y = (\tau\ x = \sigma\ y)$
 using *local.Dst tgt-local* **by** *auto*

2.4 Catoids

class *catoid* = *st-multimagma* + *multisemigroup*

sublocale *catoid* \subseteq *ts-msg*: *catoid* $\lambda x\ y. y \odot x\ tgt\ src$
by (*unfold-locales, simp add: local.assoc*)

lemma (in *catoid*) *src-comp-aur*: $v \in x \odot y \implies \sigma\ v = \sigma\ x$

by (*metis emptyE insert-iff local.assoc-exp local.s-absorb local.s-absorb-var3*)

lemma (*in catoid*) *src-comp*: $\text{Src } (x \odot y) \subseteq \{\sigma x\}$
proof–
 {fix *a*
 assume $a \in \text{Src } (x \odot y)$
 hence $\exists b \in x \odot y. a = \sigma b$
 by *auto*
 hence $\exists b. \sigma b = \sigma x \wedge a = \sigma b$
 using *local.src-comp-aux* by *blast*
 hence $a = \sigma x$
 by *blast*}
 thus ?thesis
 by *blast*
qed

lemma (*in catoid*) *src-comp-cond*: $(\Delta x y) \implies \text{Src } (x \odot y) = \{\sigma x\}$
 by (*meson image-is-empty local.src-comp subset-singletonD*)

lemma (*in catoid*) *tgt-comp-aux*: $v \in x \odot y \implies \tau v = \tau y$
 using *local.ts-msg.src-comp-aux* by *fastforce*

lemma (*in catoid*) *tgt-comp*: $\text{Tgt } (x \odot y) \subseteq \{\tau y\}$
 by (*simp add: local.ts-msg.src-comp*)

lemma (*in catoid*) *tgt-comp-cond*: $\Delta x y \implies \text{Tgt } (x \odot y) = \{\tau y\}$
 by (*simp add: local.ts-msg.src-comp-cond*)

lemma (*in catoid*) *src-weak-local*: $\text{Src } (x \odot y) \subseteq \text{Src } (x \odot \sigma y)$
proof–
 {fix *a*
 assume $a \in \text{Src } (x \odot y)$
 hence $\exists b \in x \odot y. a = \sigma b$
 by *auto*
 hence $\exists b \in x \odot y. a = \sigma b$
 by *blast*
 hence $\exists b \in x \odot y. a = \sigma b \wedge \tau x = \sigma y$
 using *local.Dst* by *auto*
 hence $\exists b \in x \odot \sigma y. a = \sigma b$
 by (*metis insertI1 local.t-absorb local.ts-msg.tgt-comp-aux*)
 hence $a \in \text{Src } (x \odot \sigma y)$
 by *force*}
 thus ?thesis
 by *force*
qed

lemma (*in catoid*) *src-local-cond*:
 $\Delta x y \implies \text{Src } (x \odot y) = \text{Src } (x \odot \sigma y)$
 by (*simp add: local.stopp.Dst local.ts-msg.tgt-comp-cond*)

lemma (in *catoid*) *tgt-weak-local*: $Tgt (x \odot y) \subseteq Tgt (\tau x \odot y)$
by (*simp add: local.ts-msg.src-weak-local*)

lemma (in *catoid*) *tgt-local-cond*:
 $\Delta x y \implies Tgt (x \odot y) = Tgt (\tau x \odot y)$
using *local.ts-msg.src-local-cond* **by** *presburger*

lemma (in *catoid*) *src-twisted-aux*:
 $u \in x \odot y \implies (x \odot \sigma y = \sigma u \odot x)$
by (*metis local.Dst local.s-absorb local.src-comp-aux local.t-absorb*)

lemma (in *catoid*) *src-twisted-cond*:
 $\Delta x y \implies x \odot \sigma y = \bigcup \{ \sigma u \odot x \mid u. u \in x \odot y \}$
using *local.stopp.Dst local.ts-msg.tgt-comp-aux* **by** *auto*

lemma (in *catoid*) *tgt-twisted-aux*:
 $u \in x \odot y \implies (\tau x \odot y = y \odot \tau u)$
by (*simp add: local.ts-msg.src-twisted-aux*)

lemma (in *catoid*) *tgt-twisted-cond*:
 $\Delta x y \implies \tau x \odot y = \bigcup \{ y \odot \tau u \mid u. u \in x \odot y \}$
by (*simp add: local.ts-msg.src-twisted-cond*)

lemma (in *catoid*) *src-funct*:
 $x \in y \odot z \implies x' \in y \odot z \implies \sigma x = \sigma x'$
using *local.src-comp-aux* **by** *force*

lemma (in *catoid*) *st-local-iff*:
 $(\forall x y. \Delta x y = (\tau x = \sigma y)) = (\forall v x y z. v \in x \odot y \longrightarrow \Delta y z \longrightarrow \Delta v z)$
apply *safe*
apply (*metis local.ts-msg.src-comp-aux*)
using *local.Dst* **apply** *blast*
by (*metis local.s-absorb-var2 local.t-absorb singleton-iff*)

Again one can lift to properties of modal semirings and quantales.

lemma (in *catoid*) *Src-weak-local*: $Src (X \star Y) \subseteq Src (X \star Src Y)$
proof –
{fix *a*
assume $a \in Src (X \star Y)$
hence $\exists b. \exists c \in X. \exists d \in Y. a = \sigma b \wedge b \in c \odot d$
by (*smt (verit) image-iff local.conv-exp2*)
hence $\exists c \in X. \exists d \in Y. a \in Src (c \odot d)$
by *auto*
hence $\exists c \in X. \exists d \in Y. a \in Src (c \odot \sigma d)$
by (*metis empty-iff image-empty local.src-local-cond*)
hence $\exists b. \exists c \in X. \exists d \in Src Y. a = \sigma b \wedge b \in c \odot d$
by *auto*
hence $a \in Src (X \star Src Y)$

```

    by (metis image-iff local.conv-exp2)}
  thus ?thesis
    by blast
qed

```

```

lemma (in catoid) Tgt-weak-local: Tgt (X ★ Y) ⊆ Tgt (Tgt X ★ Y)
  by (metis local.stopp.conv-exp local.ts-msg.Src-weak-local multimagma.conv-exp)

```

st-Locality implies locality.

```

lemma (in catoid) st-locality-l-locality:
  assumes Δ x y = (τ x = σ y)
  shows Src (x ⊙ y) = Src (x ⊙ σ y)
proof-
  {fix a
   have (a ∈ Src (x ⊙ σ y)) = (∃ b ∈ x ⊙ σ y. a = σ b)
     by auto
   also have ... = (∃ b ∈ x ⊙ σ y. a = σ b ∧ τ x = σ y)
     by (simp add: local.st-prop local.tgt-comp-aux local.tgt-twisted-aux)
   also have ... = (∃ b ∈ x ⊙ y. a = σ b)
     by (metis assms ex-in-conv local.t-absorb local.ts-msg.tgt-comp-aux singletonI)
   also have ... = (a ∈ Src (x ⊙ y))
     by auto
   finally have (a ∈ Src (x ⊙ σ y)) = (a ∈ Src (x ⊙ y)).}
  thus ?thesis
    by force
qed

```

```

lemma (in catoid) st-locality-r-locality:
  assumes lr-locality: Δ x y = (τ x = σ y)
  shows Tgt (x ⊙ y) = Tgt (τ x ⊙ y)
  by (metis local.ts-msg.st-locality-l-locality lr-locality)

```

```

lemma (in catoid) st-locality-locality:
  (Src (x ⊙ y) = Src (x ⊙ σ y) ∧ Tgt (x ⊙ y) = Tgt (τ x ⊙ y)) = (Δ x y = (τ x
= σ y))
  apply standard
  apply (simp add: local.locality)
  by (metis local.st-locality-l-locality local.ts-msg.st-locality-l-locality)

```

2.5 Locality

For st-multimagmas there are different notions of locality. I do not develop this in detail.

```

class local-catoid = catoid +
  assumes src-local: Src (x ⊙ σ y) ⊆ Src (x ⊙ y)
  and tgt-local: Tgt (τ x ⊙ y) ⊆ Tgt (x ⊙ y)

```

```

sublocale local-catoid ⊆ sts-msg: local-catoid λx y. y ⊙ x tgt src

```

apply *unfold-locale* **using** *local.tgt-local local.src-local* **by** *auto*

lemma (in *local-catoid*) *src-local-eq* [*simp*]: $\text{Src } (x \odot \sigma y) = \text{Src } (x \odot y)$
by (*simp add: local.src-local local.src-weak-local order-class.order-eq-iff*)

lemma (in *local-catoid*) *tgt-local-eq*: $\text{Tgt } (\tau x \odot y) = \text{Tgt } (x \odot y)$
by *simp*

lemma (in *local-catoid*) *src-twisted*: $x \odot \sigma y = (\bigcup u \in x \odot y. \sigma u \odot x)$
by (*metis Setcompr-eq-image Sup-empty empty-is-image local.src-twisted-cond local.sts-msg.tgt-local-eq*)

lemma (in *local-catoid*) *tgt-twisted*: $\tau x \odot y = (\bigcup u \in x \odot y. y \odot \tau u)$
using *local.sts-msg.src-twisted* **by** *auto*

lemma (in *local-catoid*) *local-var*: $\Delta x y \implies \Delta (\tau x) (\sigma y)$
by (*simp add: local.stopp.Dst*)

lemma (in *local-catoid*) *local-var-eq* [*simp*]: $\Delta (\tau x) (\sigma y) = \Delta x y$
by (*simp add: local.locality*)

I lift locality to powersets.

lemma (in *local-catoid*) *Src-local* [*simp*]: $\text{Src } (X \star \text{Src } Y) = \text{Src } (X \star Y)$
proof–
 {**fix** *a*
have $(a \in \text{Src } (X \star \text{Src } Y)) = (\exists b \in X \star \text{Src } Y. a = \sigma b)$
by (*simp add: image-iff*)
also have $\dots = (\exists b. \exists c \in X. \exists d \in \text{Src } Y. b \in c \odot d \wedge a = \sigma b)$
by (*meson local.conv-exp2*)
also have $\dots = (\exists b. \exists c \in X. \exists d \in Y. b \in c \odot \sigma d \wedge a = \sigma b)$
by *simp*
also have $\dots = (\exists c \in X. \exists d \in Y. a \in \text{Src } (c \odot \sigma d))$
by *blast*
also have $\dots = (\exists c \in X. \exists d \in Y. a \in \text{Src } (c \odot d))$
by *auto*
also have $\dots = (\exists b. \exists c \in X. \exists d \in Y. b \in c \odot d \wedge a = \sigma b)$
by *auto*
also have $\dots = (\exists b \in X \star Y. a = \sigma b)$
by (*meson local.conv-exp2*)
also have $\dots = (a \in \text{Src } (X \star Y))$
by (*simp add: image-iff*)
finally have $(a \in \text{Src } (X \star \text{Src } Y)) = (a \in \text{Src } (X \star Y)).$
thus *?thesis*
by *force*

qed

lemma (in *local-catoid*) *Tgt-local* [*simp*]: $\text{Tgt } (\text{Tgt } X \star Y) = \text{Tgt } (X \star Y)$
by (*metis local.stopp.conv-def local.sts-msg.Src-local multimagma.conv-def*)

lemma (in *local-catoid*) *st-local*: $\Delta x y = (\tau x = \sigma y)$
using *local.stopp.locality* **by** *force*

2.6 From partial magmas to single-set categories.

class *functional-magma* = *multimagma* +
assumes *functionality*: $x \in y \odot z \implies x' \in y \odot z \implies x = x'$

begin

Functional magmas could also be called partial magmas. The multioperation corresponds to a partial operation.

lemma *partial-card*: $\text{card } (x \odot y) \leq 1$
by (*metis One-nat-def card.infinite card-le-Suc0-iff-eq local.functionality zero-less-one-class.zero-le-one*)

lemma *fun-in-sgl*: $(x \in y \odot z) = (\{x\} = y \odot z)$
using *local.functionality* **by** *fastforce*

I introduce a partial operation.

definition *pcomp* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** \otimes 70) **where**
 $x \otimes y = (\text{THE } z. z \in x \odot y)$

lemma *functionality-var*: $\Delta x y \implies (\exists! z. z \in x \odot y)$
using *local.functionality* **by** *auto*

lemma *functionality-lem*: $(\exists! z. z \in x \odot y) \vee (x \odot y = \{\})$
using *functionality-var* **by** *blast*

lemma *functionality-lem-var*: $\Delta x y = (\exists z. \{z\} = x \odot y)$
using *functionality-lem* **by** *blast*

lemma *pcomp-def-var*: $(\Delta x y \wedge x \otimes y = z) = (z \in x \odot y)$
unfolding *pcomp-def* **by** (*smt (verit, del-insts) all-not-in-conv functionality-lem theI-unique*)

lemma *pcomp-def-var2*: $\Delta x y \implies ((x \otimes y = z) = (z \in x \odot y))$
using *pcomp-def-var* **by** *blast*

lemma *pcomp-def-var3*: $\Delta x y \implies ((x \otimes y = z) = (\{z\} = x \odot y))$
by (*simp add: fun-in-sgl pcomp-def-var2*)

end

class *functional-st-magma* = *functional-magma* + *st-multimagma*

class *functional-semigroup* = *functional-magma* + *multisemigroup*

begin

```

lemma pcomp-assoc-defined:  $(\Delta u v \wedge \Delta (u \otimes v) w) = (\Delta u (v \otimes w) \wedge \Delta v w)$ 
proof –
  have  $(\Delta u v \wedge \Delta (u \otimes v) w) = (\exists x. \Delta u v \wedge \Delta x w \wedge x = u \otimes v)$ 
    by simp
  also have  $\dots = (\exists x. x \in u \odot v \wedge \Delta x w)$ 
    by (metis local.pcomp-def-var)
  also have  $\dots = (\exists x. x \in v \odot w \wedge \Delta u x)$ 
    using local.assoc-exp by blast
  also have  $\dots = (\exists x. \Delta v w \wedge x = v \otimes w \wedge \Delta u x)$ 
    by (metis local.pcomp-def-var)
  also have  $\dots = (\Delta u (v \otimes w) \wedge \Delta v w)$ 
    by auto
  finally show ?thesis.
qed

```

```

lemma pcomp-assoc:  $\Delta x y \wedge \Delta (x \otimes y) z \implies (x \otimes y) \otimes z = x \otimes (y \otimes z)$ 
  by (metis (full-types) local.assoc-var local.conv-atom local.fun-in-sgl local.pcomp-def-var2
    pcomp-assoc-defined)
end

```

```

class functional-catoid = functional-semigroup + catoid

```

Finally, here comes the definition of single-set categories as in Chapter 12 of Mac Lane’s book, but with partiality of arrow composition modelled using a multioperation, or a partial operation based on it.

```

class single-set-category = functional-catoid + local-catoid

```

```

begin

```

```

lemma st-assoc:  $\tau x = \sigma y \implies \tau y = \sigma z \implies (x \otimes y) \otimes z = x \otimes (y \otimes z)$ 
  by (metis local.st-local local.pcomp-assoc local.pcomp-def-var2 local.tgt-comp-aux)

```

```

end

```

2.7 Morphisms of multimagnas and lr-multimagnas

In the context of single-set categories, these morphisms are functors. Bounded morphisms are functional bisimulations. They are known as zig-zag morphisms or p-morphism in modal and substructural logics.

```

definition mm-morphism :: ('a::multimagma  $\Rightarrow$  'b::multimagma)  $\Rightarrow$  bool where
  mm-morphism f =  $(\forall x y. \text{image } f (x \odot y) \subseteq f x \odot f y)$ 

```

```

definition bounded-mm-morphism :: ('a::multimagma  $\Rightarrow$  'b::multimagma)  $\Rightarrow$  bool
where
  bounded-mm-morphism f =  $(\text{mm-morphism } f \wedge (\forall x u v. f x \in u \odot v \longrightarrow (\exists y z. u = f y \wedge v = f z \wedge x \in y \odot z)))$ 

```


definition *st-mm-morphism* :: ('a::st-multimagma \Rightarrow 'b::st-multimagma) \Rightarrow bool
where

st-mm-morphism f = (mm-morphism f \wedge f \circ σ = $\sigma \circ$ f \wedge f \circ τ = $\tau \circ$ f)

definition *bounded-st-mm-morphism* :: ('a::st-multimagma \Rightarrow 'b::st-multimagma) \Rightarrow bool **where**

bounded-st-mm-morphism f = (bounded-mm-morphism f \wedge *st-mm-morphism* f)

2.8 Relationship with categories

Next I add a standard definition of a category following Moerdijk and Mac Lane's book and, for good measure, show that categories form single set categories and vice versa.

locale *category* =

fixes *id* :: 'objects \Rightarrow 'arrows

and *dom* :: 'arrows \Rightarrow 'objects

and *cod* :: 'arrows \Rightarrow 'objects

and *comp* :: 'arrows \Rightarrow 'arrows \Rightarrow 'arrows (**infixl** \cdot 70)

assumes *dom-id* [*simp*]: *dom* (*id* X) = X

and *cod-id* [*simp*]: *cod* (*id* X) = X

and *id-dom* [*simp*]: *id* (*dom* f) \cdot f = f

and *id-cod* [*simp*]: f \cdot *id* (*cod* f) = f

and *dom-loc* [*simp*]: *cod* f = *dom* g \Longrightarrow *dom* (f \cdot g) = *dom* f

and *cod-loc* [*simp*]: *cod* f = *dom* g \Longrightarrow *cod* (f \cdot g) = *cod* g

and *assoc*: *cod* f = *dom* g \Longrightarrow *cod* g = *dom* h \Longrightarrow (f \cdot g) \cdot h = f \cdot (g \cdot h)

begin

lemma *cod* f = *dom* g \Longrightarrow *dom* (f \cdot g) = *dom* (f \cdot *id* (*dom* g))

by *simp*

abbreviation *LL* f \equiv *id* (*dom* f)

abbreviation *RR* f \equiv *id* (*cod* f)

abbreviation *Comp* \equiv $\lambda f g. (if\ RR\ f = LL\ g\ then\ \{f \cdot g\}\ else\ \{\})$

end

typedef (**overloaded**) 'a::single-set-category *st-objects* = {x::'a::single-set-category.
 $\sigma\ x = x$ }

using *stopp.tt-idem* **by** *blast*

setup-lifting *type-definition-st-objects*

lemma *Sfix-coerce* [*simp*]: *Abs-st-objects* (σ (*Rep-st-objects* X)) = X

by (*smt* (*verit*, *best*) *Rep-st-objects* *Rep-st-objects-inverse* *mem-Collect-eq*)

lemma *Rfix-coerce* [*simp*]: *Abs-st-objects* (τ (*Rep-st-objects* X)) = X

```

by (smt (verit, best) Rep-st-objects Rep-st-objects-inverse mem-Collect-eq stopp.st-fix)

sublocale single-set-category  $\subseteq$  sscatcat: category Rep-st-objects Abs-st-objects  $\circ$ 
 $\sigma$  Abs-st-objects  $\circ \tau$  ( $\otimes$ )
  apply unfold-locales
    apply simp-all
      apply (metis (mono-tags, lifting) Abs-st-objects-inverse empty-not-insert func-
tional-magma-class.pcomp-def-var2 insertI1 mem-Collect-eq st-multimagma-class.s-absorb
st-multimagma-class.ss-idem)
        apply (metis (mono-tags, lifting) Abs-st-objects-inverse functional-magma-class.pcomp-def-var
insert-iff mem-Collect-eq st-multimagma-class.stopp.s-absorb st-multimagma-class.stopp.ts-compatible)
          apply (metis (mono-tags, lifting) Abs-st-objects-inject catoid-class.ts-msg.tgt-comp-aux
functional-magma-class.pcomp-def-var2 local-catoid-class.sts-msg.st-local mem-Collect-eq
st-multimagma-class.stopp.ts-compatible st-multimagma-class.stopp.tt-idem)
            apply (metis (mono-tags, lifting) Abs-st-objects-inject functional-semigroup-class.pcomp-assoc-defined
local-catoid-class.sts-msg.st-local mem-Collect-eq st-multimagma-class.stopp.s-absorb-var
st-multimagma-class.stopp.st-compatible)
              by (metis (mono-tags, lifting) Abs-st-objects-inverse mem-Collect-eq single-set-category-class.st-assoc
st-multimagma-class.stopp.st-compatible st-multimagma-class.stopp.ts-compatible)

sublocale category  $\subseteq$  catlrm: st-multimagma Comp LL RR
  by unfold-locales auto

sublocale category  $\subseteq$  catsscat: single-set-category Comp LL RR
  apply unfold-locales
    apply (smt (verit, ccfv-threshold) Sup-empty category.assoc category-axioms
ccpo-Sup-singleton cod-id cod-loc dom-loc image-empty image-insert)
      apply (metis empty-iff singletonD)
        apply (smt (verit, best) category.dom-id category-axioms dom-loc image-insert
set-eq-subset)
          by (smt (z3) category.cod-id category-axioms cod-loc image-insert subsetI)

```

2.9 A Mac Lane style variant

Next I present an axiomatisation of single-set categories that follows Mac Lane's axioms in Chapter I of his textbook more closely, but still uses a multioperation for arrow composition.

```

class mlss-cat = functional-magma +
  fixes l0 :: 'a  $\Rightarrow$  'a
  fixes r0 :: 'a  $\Rightarrow$  'a
  assumes comp0-def:  $(x \odot y \neq \{\}) = (r0\ x = l0\ y)$ 
  assumes r0l0 [simp]:  $r0\ (l0\ x) = l0\ x$ 
  assumes l0r0 [simp]:  $l0\ (r0\ x) = r0\ x$ 
  assumes l0-absorb [simp]:  $l0\ x \otimes x = x$ 
  assumes r0-absorb [simp]:  $x \otimes r0\ x = x$ 
  assumes assoc-defined:  $(u \odot v \neq \{\}) \wedge (u \otimes v) \odot w \neq \{\}) = (u \odot (v \otimes w) \neq \{\}) \wedge v \odot w \neq \{\})$ 
  assumes comp0-assoc:  $r0\ x = l0\ y \implies r0\ y = l0\ z \implies x \otimes (y \otimes z) = (x \otimes y) \otimes z$ 

```

```

assumes locall-var:  $r0\ x = l0\ y \implies l0\ (x \otimes y) = l0\ x$ 
assumes localr-var:  $r0\ x = l0\ y \implies r0\ (x \otimes y) = r0\ y$ 

begin

lemma ml-locall [simp]:  $l0\ (x \otimes l0\ y) = l0\ (x \otimes y)$ 
by (metis local.comp0-def local.l0-absorb local.locall-var local.pcomp-def local.r0l0)

lemma ml-localr [simp]:  $r0\ (r0\ x \otimes y) = r0\ (x \otimes y)$ 
by (metis local.comp0-def local.l0r0 local.localr-var local.pcomp-def local.r0l0)

lemma ml-locall-im [simp]:  $image\ l0\ (x \odot l0\ y) = image\ l0\ (x \odot y)$ 
by (metis (no-types, lifting) image-insert image-is-empty local.comp0-def local.fun-in-sgl
local.l0r0 local.pcomp-def-var local.r0-absorb local.r0l0 ml-locall)

lemma ml-localr-im [simp]:  $image\ r0\ (r0\ x \odot y) = image\ r0\ (x \odot y)$ 
by (smt (verit, best) image-insert local.comp0-def local.fun-in-sgl local.functionality-lem
local.l0-absorb local.l0r0 local.pcomp-def-var local.r0-absorb ml-localr)

end

sublocale single-set-category  $\subseteq$  sscatml: mlss-cat  $(\odot)\ \sigma\ \tau$ 
apply unfold-locales
apply (simp-all add: st-local pcomp-def-var2)
using local.pcomp-assoc-defined local.st-local apply force
using pcomp-assoc-defined st-assoc local.pcomp-def-var2 local.st-local local.src-comp-aux
tgt-comp-aux by fastforce+

sublocale mlss-cat  $\subseteq$  mlsscat: single-set-category  $(\odot)\ l0\ r0$ 
apply unfold-locales
apply (simp-all add: comp0-def)
apply standard
apply (clarsimp, smt (verit, ccfv-SIG) local.assoc-defined local.comp0-assoc
local.comp0-def local.fun-in-sgl local.pcomp-def-var)
apply (clarsimp, metis local.assoc-defined local.comp0-assoc local.comp0-def
local.pcomp-def-var)
apply (metis local.comp0-def local.fun-in-sgl local.l0-absorb local.pcomp-def-var2
local.r0l0)
using local.comp0-def local.fun-in-sgl local.l0r0 local.pcomp-def-var2 local.r0-absorb
by presburger

```

2.10 Product of catoids

Finally I formalise products of categories as an exercise.

instantiation *prod* :: (*catoid*, *catoid*) *catoid*
begin

definition *src-prod* $x = (\sigma\ (fst\ x), \sigma\ (snd\ x))$
for $x :: 'a \times 'b$

definition $\text{tgt-prod } x = (\tau \text{ (fst } x), \tau \text{ (snd } x))$
for $x :: 'a \times 'b$

definition $\text{mcomp-prod } x \ y = \{(u, v) \mid u \ v. \ u \in \text{fst } x \odot \text{fst } y \wedge v \in \text{snd } x \odot \text{snd } y\}$
for $x \ y :: 'a \times 'b$

instance

proof

fix $x \ y \ z :: 'a \times 'b$
show $(\bigcup v \in y \odot z. x \odot v) = (\bigcup v \in x \odot y. v \odot z)$
proof–
 {fix $a \ b$
 have $((a, b) \in (\bigcup v \in y \odot z. x \odot v)) = (\exists v. (a, b) \in x \odot v \wedge v \in y \odot z)$
 by *blast*
 also have $\dots = (\exists v \ w. a \in \text{fst } x \odot v \wedge v \in \text{fst } y \odot \text{fst } z \wedge b \in \text{snd } x \odot w \wedge$
 $w \in \text{snd } y \odot \text{snd } z)$
 using *mcomp-prod-def* **by** *auto*
 also have $\dots = (\exists v \ w. a \in v \odot \text{fst } z \wedge v \in \text{fst } x \odot \text{fst } y \wedge b \in w \odot \text{snd } z \wedge$
 $w \in \text{snd } x \odot \text{snd } y)$
 by (*meson ts-msg.assoc-exp*)
 also have $\dots = (\exists v. (a, b) \in v \odot z \wedge v \in x \odot y)$
 using *mcomp-prod-def* **by** *auto*
 also have $\dots = ((a, b) \in (\bigcup v \in x \odot y. v \odot z))$
 by *blast*
 finally have $((a, b) \in (\bigcup v \in y \odot z. x \odot v)) = ((a, b) \in (\bigcup v \in x \odot y. v \odot$
 $z)).\}$
 thus *?thesis*
 by (*meson pred-equals-eq2*)
qed
show $x \odot y \neq \{\} \implies \tau \ x = \sigma \ y$
 by (*simp add: Catoid.mcomp-prod-def Dst src-prod-def tgt-prod-def*)
show $\sigma \ x \odot x = \{x\}$
 unfolding *src-prod-def mcomp-prod-def* **by** *simp*
show $x \odot \tau \ x = \{x\}$
 unfolding *tgt-prod-def mcomp-prod-def* **by** *simp*
qed
end

instantiation $\text{prod} :: (\text{single-set-category}, \text{single-set-category}) \ \text{single-set-category}$
begin

instance

proof

fix $x \ y \ z \ x' :: 'a \times 'b$
show $x \in y \odot z \implies x' \in y \odot z \implies x = x'$
 unfolding *mcomp-prod-def* **by** (*smt (verit, best) functionality mem-Collect-eq*)

```

show  $a: \text{stopp.Tgt } (x \odot \sigma y) \subseteq \text{stopp.Tgt } (x \odot y)$ 
proof–
  {fix  $a b$ 
    have  $((a,b) \in \text{stopp.Tgt } (x \odot \sigma y)) = ((a,b) \in \text{Src } \{(c,d) \mid c d. c \in \text{fst } x \odot \sigma (\text{fst } y) \wedge d \in \text{snd } x \odot \sigma (\text{snd } y)\})$ 
    by (simp add: mcomp-prod-def src-prod-def)
    also have  $\dots = (a \in \text{Src } (\text{fst } x \odot \sigma (\text{fst } y)) \wedge b \in \text{Src } (\text{snd } x \odot \sigma (\text{snd } y)))$ 
    by (smt (z3) Setcompr-eq-image fst-conv mem-Collect-eq snd-conv src-prod-def stopp.tt-idem)
    also have  $\dots = (a \in \text{Src } (\text{fst } x \odot \text{fst } y) \wedge b \in \text{Src } (\text{snd } x \odot \text{snd } y))$ 
    by simp
    also have  $\dots = ((a,b) \in \text{Src } \{(c,d) \mid c d. c \in (\text{fst } x \odot \text{fst } y) \wedge d \in (\text{snd } x \odot \text{snd } y)\})$ 
    by (smt (z3) Setcompr-eq-image fst-conv mem-Collect-eq snd-conv src-prod-def stopp.tt-idem)
    also have  $\dots = ((a,b) \in \text{stopp.Tgt } (x \odot y))$ 
    by (simp add: mcomp-prod-def src-prod-def)
    finally have  $((a,b) \in \text{stopp.Tgt } (x \odot \sigma y)) = ((a,b) \in \text{stopp.Tgt } (x \odot y)).$ 
  }
thus ?thesis
by auto
qed
show  $\text{Tgt } (\tau x \odot y) \subseteq \text{Tgt } (x \odot y)$ 
by (metis (no-types, lifting) a bot.extremum-uniqueI empty-is-image stopp.s-absorb-var3 tgt-local-cond tgt-weak-local ts-msg.st-locality-l-locality)
qed

end

end

```

3 Groupoids

```

theory Groupoid
imports Catoid

```

```

begin

```

3.1 st-Multigroupoids

I define multigroupoids, extending the standard definition. I equip catoids with an operation of inversion.

```

class inv-op = fixes inv :: 'a  $\Rightarrow$  'a

```

```

class st-multigroupoid = catoid + inv-op +
  assumes invl:  $\sigma x \in x \odot \text{inv } x$ 
  and invr:  $\tau x \in \text{inv } x \odot x$ 

```

```

sublocale st-multigroupoid  $\subseteq$  st-mgpd: st-multigroupoid  $\lambda x y. y \odot x \text{ tgt src inv}$ 

```

by *unfold-locales* (*simp-all add: local.invr local.invl*)

Every multigroupoid is local.

lemma (in *st-multigroupoid*) *st-mgpd-local*:

assumes $\tau x = \sigma y$

shows $\Delta x y$

proof –

have $x \in x \odot \sigma y$

by (*metis assms local.t-absorb singletonI*)

hence $x \in \{x\} \star (y \odot \text{inv } y)$

using *local.conv-exp2 local.invl* by *auto*

hence $x \in (x \odot y) \star \{\text{inv } y\}$

using *local.assoc-var* by *force*

hence $\exists u v. x \in u \odot v \wedge u \in x \odot y \wedge v = \text{inv } y$

by (*metis multimagma.conv-exp2 singletonD*)

hence $\exists u. x \in u \odot \text{inv } y \wedge u \in x \odot y$

by *presburger*

hence $\exists u. u \in x \odot y$

by *fastforce*

thus *?thesis*

by *force*

qed

sublocale *st-multigroupoid* \subseteq *stmgpd: local-catoid* (\odot) *src tgt*

apply *unfold-locales*

apply (*metis local.Dst local.st-locality-locality local.st-mgpd-local set-eq-subset*)

by (*metis local.Dst local.st-locality-locality local.st-mgpd-local subset-refl*)

lemma (in *st-multigroupoid*) *tgt-inv* [*simp*]: $\tau (\text{inv } x) = \sigma x$

using *local.Dst local.invr* by *fastforce*

lemma (in *st-multigroupoid*) *src-inv*: $\sigma (\text{inv } x) = \tau x$

by *simp*

The following lemma is from Theorem 5.2 of Jónsson and Tarski’s Boolean Algebras with Operators II article.

lemma (in *st-multigroupoid*) *ba03*:

assumes $x \odot y = \{\sigma x\}$

shows $\text{inv } x = y$

proof –

have $\tau x = \sigma y$

using *assms local.Dst* by *force*

hence $\{y\} = \tau x \odot y$

by *simp*

hence $y \in \{\text{inv } x\} \star \{x\} \star \{y\}$

using *local.conv-exp2 local.invr* by *fastforce*

hence $y \in \text{inv } x \odot \sigma x$

by (*metis assms local.assoc-var local.conv-atom*)

hence $y \in \text{inv } x \odot \tau (\text{inv } x)$

```

    by simp
  thus ?thesis
    by (metis local.t-absorb singletonD)
qed

```

```

lemma (in st-multigroupoid) inv-s [simp]: inv (σ x) = σ x
proof-
  have σ x ⊙ σ x = {σ x}
  by simp
  thus inv (σ x) = σ x
    by (simp add: local.st-mgpd.bao3)
qed

```

```

lemma (in st-multigroupoid) srcfunct-inv:
  σ x ∈ x ⊙ inv x ⟹ σ y ∈ x ⊙ inv x ⟹ σ x = σ y
using local.ts-msg.src-funct by fastforce

```

```

lemma (in st-multigroupoid) tgtfunct-inv:
  τ x ∈ inv x ⊙ x ⟹ τ y ∈ inv x ⊙ x ⟹ τ x = τ y
by (metis local.ts-msg.src-comp-aux local.tt-idem)

```

As for catoids, I prove quantalic properties, lifting to powersets.

```

abbreviation (in st-multigroupoid) Inv :: 'a set ⇒ 'a set where
  Inv ≡ image inv

```

```

lemma (in st-multigroupoid) Inv-exp: Inv X = {inv x | x. x ∈ X}
by blast

```

```

lemma (in st-multigroupoid) Inv-un: Inv (X ∪ Y) = Inv X ∪ Inv Y
by (simp add: image-Un)

```

```

lemma (in st-multigroupoid) Inv-Un: Inv (⋃ X) = (⋃ X ∈ X. Inv X)
unfolding Inv-exp by auto

```

```

lemma (in st-multigroupoid) Invl: Src X ⊆ X ★ Inv X
unfolding Inv-exp conv-exp using local.invl by fastforce

```

```

lemma (in st-multigroupoid) Invr: Tgt X ⊆ Inv X ★ X
by (meson imageI image-subsetI local.invr local.stopp.conv-exp2)

```

```

lemma (in st-multigroupoid) Inv-strong-gelfand: X ⊆ X ★ Inv X ★ X
proof-
  have X = Src X ★ X
  by simp
  also have ... ⊆ X ★ Inv X ★ X
  using local.Invl local.conv-isor by presburger
  finally show ?thesis.
qed

```

At powerset level, one can define domain and codomain operations explicitly

as in relation algebras.

lemma (in *st-multigroupoid*) *dom-def*: $\text{Src } X = \text{sfix} \cap (X \star \text{Inv } X)$

proof–

{**fix** a

have $(a \in \text{sfix} \cap (X \star \text{Inv } X)) = (\sigma a = a \wedge \sigma a \in X \star \text{Inv } X)$

by *fastforce*

also have $\dots = (\sigma a = a \wedge (\exists b \in X. \exists c \in \text{Inv } X. \sigma a \in b \odot c))$

using *local.conv-exp2* **by** *auto*

also have $\dots = (\sigma a = a \wedge (\exists b \in X. \sigma a = \sigma b))$

by (*metis imageI local.invl local.ts-msg.tgt-comp-aux*)

also have $\dots = (a \in \text{Src } X)$

by *auto*

finally have $(a \in \text{sfix} \cap (X \star \text{Inv } X)) = (a \in \text{Src } X).$

thus *?thesis*

by *blast*

qed

lemma (in *st-multigroupoid*) *cod-def*: $\text{Tgt } X = \text{sfix} \cap (\text{Inv } X \star X)$

by (*metis local.st-mgpd.dom-def local.stfix-set local.stopp.conv-def multimagma.conv-def*)

lemma (in *st-multigroupoid*) *dom-def-var*: $\text{Src } X = \text{sfix} \cap (X \star \text{UNIV})$

proof–

{**fix** a

have $(a \in \text{sfix} \cap (X \star \text{UNIV})) = (\sigma a = a \wedge \sigma a \in X \star \text{UNIV})$

by *fastforce*

also have $\dots = (\sigma a = a \wedge (\exists b \in X. \exists c. \sigma a \in b \odot c))$

using *local.conv-exp2* **by** *auto*

also have $\dots = (\sigma a = a \wedge (\exists b \in X. \sigma a = \sigma b))$

by (*metis local.invl local.ts-msg.tgt-comp-aux*)

also have $\dots = (a \in \text{Src } X)$

by *auto*

finally have $(a \in \text{sfix} \cap (X \star \text{UNIV})) = (a \in \text{Src } X).$

thus *?thesis*

by *blast*

qed

lemma (in *st-multigroupoid*) *cod-def-var*: $\text{Tgt } X = \text{sfix} \cap (\text{UNIV} \star X)$

by (*metis local.ST-im local.sfix-im local.st-mgpd.dom-def-var local.stopp.conv-def local.tfix-im multimagma.conv-def*)

lemma (in *st-multigroupoid*) *dom-univ*: $X \star \text{UNIV} = \text{Src } X \star \text{UNIV}$

proof–

have $X \star \text{UNIV} = \text{Src } X \star X \star \text{UNIV}$

using *local.Src-absorp* **by** *presburger*

also have $\dots \subseteq \text{Src } X \star \text{UNIV} \star \text{UNIV}$

by (*meson local.conv-isol local.conv-isor subset-UNIV*)

finally have $a: X \star \text{UNIV} \subseteq \text{Src } X \star \text{UNIV}$

using *local.conv-assoc local.conv-isol subset-UNIV* **by** *blast*

have $\text{Src } X \star \text{UNIV} \subseteq X \star \text{Inv } X \star \text{UNIV}$


```

    using local.Invl local.conv-isol by presburger
  also have ...  $\subseteq X \star UNIV \star UNIV$ 
    by (simp add: local.conv-isol local.conv-isol)
  finally have  $Src\ X \star UNIV \subseteq X \star UNIV$ 
    by (metis dual-order.trans local.conv-assoc local.conv-isol subset-UNIV)
  thus ?thesis
    using a by force
qed

```

```

lemma (in st-multigroupoid) cod-univ:  $UNIV \star X = UNIV \star Tgt\ X$ 
  by (metis local.st-mgpd.dom-univ local.stopp.conv-def multimagma.conv-def)

```

3.2 Groupoids

Groupoids are simply functional multigroupoids. I start with a somewhat indirect axiomatisation.

```

class groupoid-var = st-multigroupoid + functional-catoid

```

```

begin

```

```

lemma invl [simp]:  $x \odot inv\ x = \{\sigma\ x\}$ 
  using local.fun-in-sgl local.invl by force

```

```

lemma invr [simp]:  $inv\ x \odot x = \{\tau\ x\}$ 
  using local.fun-in-sgl local.invr by force

```

```

end

```

Next, I provide a more direct axiomatisation.

```

class groupoid = catoid + inv-op +
  assumes invs [simp]:  $x \odot inv\ x = \{\sigma\ x\}$ 
  and invt [simp]:  $inv\ x \odot x = \{\tau\ x\}$ 

```

```

subclass (in groupoid) st-multigroupoid
  by unfold-locales simp-all

```

```

sublocale groupoid  $\subseteq$  lrgpd: groupoid  $\lambda x\ y. y \odot x\ tgt\ src\ inv$ 
  by unfold-locales simp-all

```

```

lemma (in groupoid) bao4 [simp]:  $inv\ (inv\ x) = x$ 
proof-
  have  $inv\ x \odot x = \{\sigma\ (inv\ x)\}$ 
    by simp
  thus ?thesis
    using local.bao3 by blast
qed

```

```

lemma (in groupoid) rev1:

```

$x \in y \odot z \implies y \in x \odot \text{inv } z$
proof –
 assume $h: x \in y \odot z$
 hence $x \odot \text{inv } z \subseteq y \odot z \star \{\text{inv } z\}$
 using *multimagma.conv-exp2* **by** *fastforce*
 hence $x \odot \text{inv } z \subseteq \{y\} \star (z \odot \text{inv } z)$
 using *local.assoc-var* **by** *presburger*
 hence $x \odot \text{inv } z \subseteq y \odot \sigma z$
 by *simp*
 hence $x \odot \text{inv } z \subseteq y \odot \tau y$
 using *h local.src-comp-aux local.src-twisted-aux* **by** *auto*
 hence $a: x \odot \text{inv } z \subseteq \{y\}$
 by *simp*
 have $\tau x = \tau z$
 using *h local.tgt-comp-aux* **by** *auto*
 hence $x \odot \text{inv } z \neq \{\}$
 by (*simp add: local.st-mgpd-local*)
 hence $x \odot \text{inv } z = \{y\}$
 using *a* **by** *auto*
 thus *?thesis*
 by *force*
qed

lemma (*in groupoid*) *rev2*:
 $x \in y \odot z \implies z \in \text{inv } y \odot x$
by (*simp add: local.lrgpd.rev1*)

lemma (*in groupoid*) *rev1-eq*: $(y \in x \odot (\text{inv } z)) = (x \in y \odot z)$
using *local.lrgpd.rev2* **by** *force*

lemma (*in groupoid*) *rev2-eq*: $(z \in (\text{inv } y) \odot x) = (x \in y \odot z)$
by (*simp add: local.lrgpd.rev1-eq*)

The following fact show that the axiomatisation above captures indeed groupoids.

lemma (*in groupoid*) *lr-mgpd-partial*:
 assumes $x \in y \odot z$
 and $x' \in y \odot z$
 shows $x = x'$
proof –
 have $z \in \text{inv } y \odot x$
 by (*simp add: assms(1) rev2*)
 hence $x' \in \{y\} \star (\text{inv } y \odot x)$
 using *assms(2) local.conv-exp2* **by** *auto*
 hence $x' \in (y \odot \text{inv } y) \star \{x\}$
 by (*simp add: local.assoc-var*)
 hence $x' \in \sigma y \odot x$
 by (*simp add: multimagma.conv-atom*)
 hence $x' \in \sigma x \odot x$

```

    using assms(1) local.ts-msg.tgt-comp-aux by auto
  thus ?thesis
    by simp
qed

```

```

subclass (in groupoid) single-set-category
  by unfold-locales (simp add: local.lr-mgpd-partial)

```

Hence st-groupoids are indeed single-set categories in which all arrows are isomorphisms.

```

lemma (in groupoid) src-canc1:
  assumes  $\tau z = \sigma x$ 
  and  $\tau z = \sigma y$ 
  and  $z \otimes x = z \otimes y$ 
shows  $x = y$ 
proof-
  have  $\text{inv } z \otimes (z \otimes x) = \text{inv } z \otimes (z \otimes y)$ 
    by (simp add: assms(3))
  hence  $(\text{inv } z \otimes z) \otimes x = (\text{inv } z \otimes z) \otimes y$ 
    using assms(1) assms(2) local.sscatml.comp0-assoc by auto
  hence  $\tau z \otimes x = \tau z \otimes y$ 
    by (simp add: local.pcomp-def)
  thus ?thesis
    by (metis assms(1) assms(2) local.sscatml.l0-absorb)
qed

```

```

lemma (in groupoid) tgt-canc1:
  assumes  $\tau x = \sigma z$ 
  and  $\tau y = \sigma z$ 
  and  $x \otimes z = y \otimes z$ 
shows  $x = y$ 
  by (metis assms local.lrgpd.pcomp-def-var local.lrgpd.src-canc1 local.pcomp-def-var
    local.st-mgpd.st-mgpd-local)

```

The following lemmas are from Theorem 5.2 of Jónsson and Tarski's BAO II article.

```

lemma (in groupoid) bao1 [simp]:  $x \otimes (\text{inv } x \otimes x) = x$ 
  by (simp add: local.pcomp-def)

```

```

lemma (in groupoid) bao2 [simp]:  $(x \otimes \text{inv } x) \otimes x = x$ 
  by (simp add: local.st-assoc)

```

```

lemma (in groupoid) bao5:
   $\tau x = \sigma y \implies \text{inv } x \otimes x = y \otimes \text{inv } y$ 
  using local.invs local.invt local.pcomp-def by auto

```

```

lemma (in groupoid) bao6:  $\text{Inv } (x \odot y) = \text{inv } y \odot \text{inv } x$ 
  apply (rule antisym)
  using rev1-eq rev2-eq apply force

```

by (clarsimp, metis imageI local.bao4 local.rev1-eq local.rev2-eq)

3.3 Axioms of relation algebra

I formalise a special case of a famous theorem of Jónsson and Tarski, showing that groupoids lift to relation algebras at powerset level. All axioms not related to converse have already been considered previously.

lemma (in groupoid) *Inv-invol* [simp]: $\text{Inv} (\text{Inv } X) = X$

proof–

have $\text{Inv} (\text{Inv } X) = \{\text{inv} (\text{inv } x) \mid x. x \in X\}$

by (simp add: image-image)

also have $\dots = X$

by simp

finally show ?thesis.

qed

lemma (in groupoid) *Inv-contrav*: $\text{Inv} (X \star Y) = \text{Inv } Y \star \text{Inv } X$

proof–

have $\text{Inv} (X \star Y) = (\bigcup x \in X. \bigcup y \in Y. \text{Inv} (x \odot y))$

unfolding conv-def image-def by blast

also have $\dots = (\bigcup x \in X. \bigcup y \in Y. \text{inv } y \odot \text{inv } x)$

by (simp add: local.bao6)

also have $\dots = \text{Inv } Y \star \text{Inv } X$

unfolding conv-def image-def by blast

finally show ?thesis.

qed

lemma (in groupoid) *residuation*: $\text{Inv } X \star -(X \star Y) \subseteq -Y$

using local.lrgpd.rev1 local.stopp.conv-exp2 by fastforce

lemma (in groupoid) *modular-law*: $(X \star Y) \cap Z \subseteq (X \cap (Z \star \text{Inv } Y)) \star Y$

using local.lrgpd.rev2 local.stopp.conv-exp2 by fastforce

lemma (in groupoid) *dedekind*: $(X \star Y) \cap Z \subseteq (X \cap (Z \star \text{Inv } Y)) \star (Y \cap (\text{Inv } X \star Z))$

unfolding Inv-exp conv-exp

apply clarsimp

using local.rev1 local.rev2 by blast

In sum, this shows that the powerset lifting of a groupoid is a relation algebra. I link this formally with relations in an interpretation statement in another component.

Jónsson and Tarski's axioms of relation algebra are slightly different. It is routine to related them formally with those used here. It might also be interested to use their partiality-by-closure approach to defining groupoids in a setting with explicit carrier sets in another Isabelle formalisation.

lemma (in groupoid) *Inv-compl*: $\text{Inv} (-X) = -(\text{Inv } X)$

by (metis UNIV-I bij-def bij-image-Compl-eq equalityI image-eqI inj-def local.bao4 subsetI)

lemma (in groupoid) *Inv-inter*: $Inv (X \cap Y) = Inv X \cap Inv Y$
 using local.Inv-compl by auto

lemma (in groupoid) *Inv-Un*: $Inv (\bigcap \mathcal{X}) = (\bigcap X \in \mathcal{X}. Inv X)$

proof–

have $Inv (\bigcap \mathcal{X}) = Inv (\neg(\bigcup X \in \mathcal{X}. \neg X))$

by (simp add: Setcompr-eq-image)

also have $\dots = \neg (Inv (\bigcup X \in \mathcal{X}. \neg X))$

using local.Inv-compl by presburger

also have $\dots = \neg(\bigcup X \in \mathcal{X}. Inv (\neg X))$

by blast

also have $\dots = \neg(\bigcup X \in \mathcal{X}. \neg(Inv X))$

using local.Inv-compl by presburger

also have $\dots = (\bigcap X \in \mathcal{X}. Inv X)$

by blast

finally show $Inv (\bigcap \mathcal{X}) = (\bigcap X \in \mathcal{X}. Inv X)$.

qed

end

4 Lifting catoids to modal powerset quantales

theory *Catoid-Lifting*

imports *Catoid Quantales-Converse.Modal-Quantale*

begin

instantiation *set* :: (catoid) monoid-mult

begin

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

$1 = \text{sfix}$

definition *times-set* :: 'a set \Rightarrow 'a set \Rightarrow 'a set **where**

$X * Y = X \star Y$

instance

apply *intro-classes*

unfolding *times-set-def one-set-def*

apply (simp add: conv-assoc)

using stopp.conv-unt apply blast

by (metis stfix-set stopp.conv-uns)

end

```

instantiation set :: (catoid) semiring-one-zero

begin

definition zero-set :: 'a set where
  zero-set = {}

definition plus-set :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set where
   $X + Y = X \cup Y$ 

instance
  apply intro-classes
  unfolding times-set-def one-set-def zero-set-def plus-set-def conv-exp
    apply safe
    apply blast
    apply blast
    apply blast
    apply blast
    apply (metis Dst empty-iff singletonD stopp.st-compat stopp.t-absorb)
    apply (metis (mono-tags) insertI1 mem-Collect-eq stopp.t-absorb stopp.tt-idem)
  using Dst singletonD apply fastforce
    apply (metis (mono-tags) insertI1 mem-Collect-eq stopp.s-absorb stopp.ts-compat)
    apply blast
    apply blast
  apply blast
  by blast

end

instantiation set :: (catoid) dioid

begin

instance
  by intro-classes (auto simp: plus-set-def)

end

instantiation set :: (local-catoid) domain-semiring

begin

definition domain-op-set :: 'a set  $\Rightarrow$  'a set where
  dom X = Src X

instance
  apply intro-classes
    apply (simp add: Catoid-Lifting.domain-op-set-def times-set-def)
    apply (simp add: domain-op-set-def times-set-def)

```

```

    apply (metis (full-types) domain-op-set-def less-eq-def one-set-def stopp.Tgt-subid)
    apply (simp add: Catoid-Lifting.domain-op-set-def zero-set-def)
    by (simp add: Catoid-Lifting.domain-op-set-def image-Un plus-set-def)

end

instantiation set :: (local-catoid) range-semiring

begin

definition range-op-set :: 'a set  $\Rightarrow$  'a set where
  cod X = Tgt X

instance
  apply intro-classes
    apply (simp add: Catoid-Lifting.range-op-set-def times-set-def)
    apply (simp add: range-op-set-def times-set-def)
    apply (metis (mono-tags, lifting) Catoid-Lifting.range-op-set-def boolean-algebra.disj-one-right
image-Un one-set-def plus-set-def stfix-set stopp.sfix-im)
    apply (simp add: range-op-set-def zero-set-def)
    by (simp add: image-Un plus-set-def range-op-set-def)

end

instantiation set :: (local-catoid) dr-modal-semiring

begin

instance
  by intro-classes (auto simp add: domain-op-set-def range-op-set-def)

end

instantiation set :: (catoid) quantale

begin

instance
  by (intro-classes, auto simp: times-set-def conv-exp)

end

instantiation set :: (local-catoid) domain-quantale

begin

instance
  by intro-classes (simp-all, auto simp add: domain-op-set-def image-Un)

```

```

end

instantiation set :: (local-catoid) codomain-quantale

begin

instance
  by intro-classes (simp-all, auto simp add: range-op-set-def image-Un)

end

instantiation set :: (local-catoid) dc-modal-quantale

begin

instance
  by intro-classes simp-all

end

end

```

5 Lifting groupoids to powerset Dedekind quantales and powerset relation algebras

```

theory Groupoid-Lifting
  imports Groupoid Quantales-Converse.Quantale-Converse Catoid-Lifting Relation-Algebra.Relation-Algebra

begin

instantiation set :: (groupoid) dedekind-quantale
begin

definition invol-set :: 'a set  $\Rightarrow$  'a set where
  invol = Inv

instance
  apply intro-classes
    apply (simp add: invol-set-def)
    apply (simp add: Inv-contrav invol-set-def times-set-def)
    apply (simp add: Groupoid-Lifting.invol-set-def image-Union)
    by (simp add: groupoid-class.modular-law invol-set-def times-set-def)

end

instantiation set :: (groupoid) boolean-dedekind-quantale

```



```

begin

instance..

end

instantiation set :: (groupoid) relation-algebra

begin

definition composition-set :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set where
  composition-set x y = x  $\star$  y

definition converse-set :: 'a set  $\Rightarrow$  'a set where
  converse = Inv

definition unit-set :: 'a set where
  unit-set = sfix

instance
  apply intro-classes
    apply (simp add: composition-set-def conv-assoc)
    apply (smt (verit) composition-set-def stfix-set stopp.conv-uns unit-set-def)
    apply (simp add: composition-set-def stopp.conv-distl-small)
    apply (simp add: converse-set-def)
    apply (simp add: converse-set-def st-mgpd.Inv-un)
    apply (simp add: Inv-contrav composition-set-def converse-set-def)
    by (simp add: composition-set-def converse-set-def groupoid-class.residuation)

end

end

```

6 Multimonoids

```

theory Multimonoid
  imports Catoid

```

```

begin

```

```

context multimagma
begin

```

6.1 Unital multimagnas

This component presents an alternative approach to catoids, as multisemi-groups with many units. This is more akin to the formalisation of single-set

categories in Chapter I of Mac Lane's book, but in fact this approach to axiomatising categories goes back to the middle of the twentieth century.

Units can already be defined in multimagnas.

definition $\text{munitl } e = ((\exists x. x \in e \odot x) \wedge (\forall x y. y \in e \odot x \longrightarrow y = x))$

definition $\text{munitr } e = ((\exists x. x \in x \odot e) \wedge (\forall x y. y \in x \odot e \longrightarrow y = x))$

abbreviation $\text{munit } e \equiv (\text{munitl } e \vee \text{munitr } e)$

end

A multimagma is unital if every element has a left and a right unit.

class $\text{unital-multimagma-var} = \text{multimagma} +$
assumes $\text{munitl-ex}: \forall x. \exists e. \text{munitl } e \wedge \Delta e x$
assumes $\text{munitr-ex}: \forall x. \exists e. \text{munitr } e \wedge \Delta x e$

begin

lemma $\text{munitl-ex-var}: \forall x. \exists e. \text{munitl } e \wedge x \in e \odot x$
by $(\text{metis equals0I local.munitl-def local.munitl-ex})$

lemma $\text{unitl}: \bigcup \{e \odot x \mid e. \text{munitl } e\} = \{x\}$
apply safe
apply $(\text{simp add: multimagma.munitl-def})$
by $(\text{simp, metis munitl-ex-var})$

lemma $\text{munitr-ex-var}: \forall x. \exists e. \text{munitr } e \wedge x \in x \odot e$
by $(\text{metis equals0I local.munitr-def local.munitr-ex})$

lemma $\text{unitr}: \bigcup \{x \odot e \mid e. \text{munitr } e\} = \{x\}$
apply safe
apply $(\text{simp add: multimagma.munitr-def})$
by $(\text{simp, metis munitr-ex-var})$

end

Here is an alternative definition.

class $\text{unital-multimagma} = \text{multimagma} +$
fixes $E :: 'a \text{ set}$
assumes $E1: \bigcup \{e \odot x \mid e. e \in E\} = \{x\}$
and $E2: \bigcup \{x \odot e \mid e. e \in E\} = \{x\}$

begin

lemma $E1: \forall e \in E. (\forall x y. y \in e \odot x \longrightarrow y = x)$
using local.E1 **by** fastforce

lemma $E2: \forall e \in E. (\forall x y. y \in x \odot e \longrightarrow y = x)$

```

using local.Er by fastforce

lemma El11:  $\forall x. \exists e \in E. x \in e \odot x$ 
  using local.El by fastforce

lemma El12:  $\forall x. \exists e \in E. e \odot x = \{x\}$ 
  using E1 El11 by fastforce

lemma Er11:  $\forall x. \exists e \in E. x \in x \odot e$ 
  using local.Er by fastforce

lemma Er12:  $\forall x. \exists e \in E. x \odot e = \{x\}$ 
  using Er Er11 by fastforce

Units are "orthogonal" idempotents.

lemma unit-id:  $\forall e \in E. e \in e \odot e$ 
  using E1 local.Er by fastforce

lemma unit-id-eq:  $\forall e \in E. e \odot e = \{e\}$ 
  by (simp add: E1 equalityI subsetI unit-id)

lemma unit-comp:
  assumes  $e_1 \in E$ 
  and  $e_2 \in E$ 
  and  $\Delta e_1 e_2$ 
  shows  $e_1 = e_2$ 
  proof-
    obtain  $x$  where  $a: x \in e_1 \odot e_2$ 
      using assms(3) by auto
    hence  $b: x = e_1$ 
      using E2 assms(2) by blast
    hence  $x = e_2$ 
      using E1 a assms(1) by blast
    thus  $e_1 = e_2$ 
      by (simp add: b)
  qed

lemma unit-comp-iff:  $e_1 \in E \implies e_2 \in E \implies (\Delta e_1 e_2 = (e_1 = e_2))$ 
  using unit-comp unit-id by fastforce

lemma  $\forall e \in E. \exists x. x \in e \odot x$ 
  using unit-id by force

lemma  $\forall e \in E. \exists x. x \in x \odot e$ 
  using unit-id by force

sublocale unital-multimagma-var
  apply unfold-locales
  apply (metis E1 El12 empty-not-insert insertI1 local.munitl-def)

```

by (*metis E2 Er12 empty-not-insert insertI1 local.munitr-def*)

Now it is clear that the two definitions are equivalent.

The next two lemmas show that the set of units is a left and right unit of composition at powerset level.

lemma *conv-unl*: $E \star X = X$
unfolding *conv-def*
apply *safe*
using *E1* **apply** *blast*
using *El12* **by** *fastforce*

lemma *conv-unr*: $X \star E = X$
unfolding *conv-def*
apply *safe*
using *E2* **apply** *blast*
using *Er12* **by** *fastforce*

end

6.2 Multimonoids

A multimonoid is a unital multisemigroup.

class *multimonoid* = *multisemigroup* + *unital-multimagma*

begin

In a multimonoid, left and right units are unique for each element.

lemma *munits-unique*: $\forall x. \exists! e. e \in E \wedge e \odot x = \{x\}$
proof –
{fix *x*
obtain *e* **where** *a*: $e \in E \wedge e \odot x = \{x\}$
using *local.El12* **by** *blast*
{fix *e'*
assume *b*: $e' \in E \wedge e' \odot x = \{x\}$
hence $\{e\} \star (e' \odot x) = \{x\}$
by (*simp add: a multimagma.conv-atom*)
hence $(e \odot e') \star \{x\} = \{x\}$
by (*simp add: local.assoc-var*)
hence $\Delta e e'$
using *local.conv-exp2* **by** *auto*
hence $e = e'$
by (*simp add: a b local.unit-comp-iff*)
hence $\exists e \in E. e \odot x = \{x\} \wedge (\forall e' \in E. e' \odot x = \{x\} \longrightarrow e = e)$
using *a* **by** *blast*
thus *?thesis*
by (*metis emptyE local.assoc-exp local.unit-comp singletonI*)
qed

lemma *munits-unique*: $\forall x. \exists ! e. e \in E \wedge x \odot e = \{x\}$
apply *safe*
apply (*meson local.Er12*)
by (*metis insertI1 local.E1 local.E2 local.assoc-var local.conv-exp2*)

In a monoid, there is of course one single unit, and my definition of many units reduces to this one.

lemma *units-unique*: $(\forall x y. \Delta x y) \implies \exists ! e. e \in E$
apply *safe*
using *local.El11* **apply** *blast*
using *local.unit-comp-iff* **by** *presburger*

lemma *units-rm2l*: $e_1 \in E \implies e_2 \in E \implies \Delta e_1 x \implies \Delta e_2 x \implies e_1 = e_2$
by (*smt (verit, del-insts) ex-in-conv local.E1 local.assoc-exp local.unit-comp*)

lemma *units-rm2r*: $e_1 \in E \implies e_2 \in E \implies \Delta x e_1 \implies \Delta x e_2 \implies e_1 = e_2$
by (*metis (full-types) ex-in-conv local.E2 local.assoc-exp local.unit-comp*)

One can therefore express the functional relationship between elements and their units in terms of explicit (source and target) maps – as in catoids.

definition *so* :: $'a \Rightarrow 'a$ **where**
 $so\ x = (THE\ e. e \in E \wedge e \odot x = \{x\})$

definition *ta* :: $'a \Rightarrow 'a$ **where**
 $ta\ x = (THE\ e. e \in E \wedge x \odot e = \{x\})$

abbreviation *So* :: $'a\ set \Rightarrow 'a\ set$ **where**
 $So\ X \equiv image\ so\ X$

abbreviation *Ta* :: $'a\ set \Rightarrow 'a\ set$ **where**
 $Ta\ X \equiv image\ ta\ X$

end

6.3 Multimonoids and catoids

It is now easy to show that every catoid is a multimonoid and vice versa.

One cannot have both sublocale statements at the same time.

The converse direction requires some preparation.

lemma (*in multimonoid*) *so-unit*: $so\ x \in E$
unfolding *so-def* **by** (*metis (mono-tags, lifting) local.munits-unique theI'*)

lemma (*in multimonoid*) *ta-unit*: $ta\ x \in E$
unfolding *ta-def* **by** (*metis (mono-tags, lifting) local.munits-unique theI'*)

```

lemma (in multimonoid) so-absorbl:  $so\ x \odot x = \{x\}$ 
  unfolding so-def by (metis (mono-tags, lifting) local.munits-unique the-equality)

lemma (in multimonoid) ta-absorbr:  $x \odot ta\ x = \{x\}$ 
  unfolding ta-def by (metis (mono-tags, lifting) local.munits-unique the-equality)

lemma (in multimonoid) semi-locality:  $\Delta\ x\ y \implies ta\ x = so\ y$ 
  by (smt (verit, best) local.assoc-var local.conv-atom local.so-absorbl local.so-unit
local.ta-absorbr local.ta-unit local.units-rm2l local.units-rm2r)

sublocale multimonoid  $\subseteq$  monlr: catoid ( $\odot$ ) so ta
  by (unfold-locales, simp-all add: local.semi-locality local.so-absorbl local.ta-absorbr)

```

6.4 From multimonoids to categories

Single-set categories are precisely local partial monoids, that is, object-free categories as in Chapter I of Mac Lane's book.

```

class local-multimagma = multimagma +
  assumes locality:  $v \in x \odot y \implies \Delta\ y\ z \implies \Delta\ v\ z$ 

class local-multisemigroup = multisemigroup + local-multimagma

In this context, a semicategory is an object-free category without identity
arrows

class of-semicategory = local-multisemigroup + functional-semigroup

begin

lemma part-locality:  $\Delta\ x\ y \implies \Delta\ y\ z \implies \Delta\ (x \otimes y)\ z$ 
  by (meson local.locality local.pcomp-def-var2)

lemma part-locality-var:  $\Delta\ x\ y \implies \Delta\ y\ z \implies (x \odot y) \star \{z\} \neq \{\}$ 
  by (smt (z3) ex-in-conv local.locality multimagma.conv-exp2 singleton-iff)

lemma locality-iff:  $(\Delta\ x\ y \wedge \Delta\ y\ z) = (\Delta\ x\ y \wedge \Delta\ (x \otimes y)\ z)$ 
  by (meson local.pcomp-assoc-defined part-locality)

lemma locality-iff-var:  $(\Delta\ x\ y \wedge \Delta\ y\ z) = (\Delta\ x\ y \wedge (x \odot y) \star \{z\} \neq \{\})$ 
  by (metis ex-in-conv local.assoc-var local.conv-exp2 part-locality-var)

end

class partial-monoid = multimonoid + functional-magma

class local-multimonoid = multimonoid + local-multimagma

begin

```

```

lemma sota-locality:  $ta\ x = so\ y \implies \Delta\ x\ y$ 
  using local.locality monlr.st-local-iff by blast

lemma So-local:  $So\ (x \odot so\ y) = So\ (x \odot y)$ 
  using local.locality monlr.st-local-iff monlr.st-locality-locality by presburger

lemma Ta-local:  $Ta\ (ta\ x \odot y) = Ta\ (x \odot y)$ 
  using local.locality monlr.st-local-iff monlr.st-locality-locality by presburger

sublocale locmm: local-catoid ( $\odot$ ) so ta
  by (unfold-locales, simp-all add: So-local Ta-local)

The following statements formalise compatibility properties.

lemma local-conv:  $v \in x \odot y \implies (\Delta\ v\ z = \Delta\ y\ z)$ 
  by (metis ex-in-conv local.assoc-exp local.locality)

lemma local-alt:  $e \in E \implies x \in x \odot e \implies y \in e \odot y \implies \Delta\ x\ y$ 
  using local-conv by blast

lemma local-iff:  $\Delta\ x\ y = (\exists e \in E. \Delta\ x\ e \wedge \Delta\ e\ y)$ 
  by (smt (verit, best) local.Er11 local.units-rm2l local-alt local-conv)

lemma local-iff2:  $(ta\ x = so\ y) = \Delta\ x\ y$ 
  by (simp add: locmm.st-local)

end

```

Finally I formalise object-free categories. The axioms are essentially Mac Lane's, but a multioperation is used for arrow composition, to capture partiality.

```

class of-category = of-semicategory + partial-monoid

```

The next statements show that single-set categories based on catoids and object-free categories based on multimonooids are the same (we can only have one direction as a sublocale statement). It then follows from results about catoids and single-set categories that object-free categories are indeed categories. These results can be found in the catoid component. I do not present explicit proofs for object-free categories here.

```

sublocale of-category  $\subseteq$  ofss-cat: single-set-category - so ta
  apply unfold-locales
  using local.locality monlr.st-local-iff monlr.st-locality-locality apply auto[1]
  using local.locality monlr.st-local-iff monlr.st-locality-locality monlr.tgt-weak-local
by presburger

```

6.5 Multimonooids and relational monooids

Relational monooids are monooids in the category Rel. They have been used previously to construct convolution algebras in another AFP entry. Here I

show that relational monoids are isomorphic to multimonoids, but I do not integrate the AFP entry with relational monoids because it uses a historic quantale component, which is different from the quantale component in the AFP. Instead, I simply copy in the definitions leading to relational monoids and leave the consolidation of Isabelle theories to the future.

```

class rel-magma =
  fixes  $\varrho :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ 

class rel-semigroup = rel-magma +
  assumes rel-assoc:  $(\exists y. \varrho y u v \wedge \varrho x y w) = (\exists z. \varrho z v w \wedge \varrho x u z)$ 

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

```

Once again, only one of the two sublocale statements compiles.

```

sublocale multimonoid  $\subseteq$  rel-monoid  $\lambda x y z. x \in y \odot z E$ 
  apply unfold-locales
  using local.assoc-exp apply blast
  using local.El11 apply blast
  apply (simp add: local.Er11)
  using local.E1 apply blast
  by (simp add: local.E2)

end

```

References

- [1] R. Brown. From groups to groupoids: A brief survey. *Bulletin of the London Mathematical Society*, 19:113–134, 1987.
- [2] C. Calk, U. Fahrenberg, C. Johansen, G. Struth, and K. Ziemiański. *lr*-multisemigroups, modal quantales and the origin of locality. In *RAMiCS 2021*, volume 13027 of *LNCS*, pages 90–107. Springer, 2021.
- [3] C. Calk, P. Malbos, D. Pous, and G. Struth. Higher catoids, higher quantales and their correspondences. *arXiv*, 2307.09253, 2023.
- [4] J. Cranch, S. Doherty, and G. Struth. Relational semigroups and object-free categories. *arXiv*, 2001.11895, 2020.
- [5] B. Dongol, V. B. F. Gomes, I. J. Hayes, and G. Struth. Partial semigroups and convolution algebras. *Archive of Formal Proofs*, 2017.

- [6] U. Fahrenberg, C. Johansen, G. Struth, and K. Ziemiański. Catoids and modal convolution algebras. *Algebra Universalis*, 84:10, 2023.
- [7] B. Jónsson and A. Tarski. Boolean algebras with operators. Part II. *American Journal of Mathematics*, 74(1):127–162, 1952.
- [8] S. Mac Lane. *Categories for the Working Mathematician*, volume 5. Springer, second edition, 1998.
- [9] E. W. Stark. Category theory with adjunctions and limits. *Archive of Formal Proofs*, 2016.