# Aggregation Algebras 

Walter Guttmann

September 13, 2023


#### Abstract

We develop algebras for aggregation and minimisation for weight matrices and for edge weights in graphs. We show numerous instances of these algebras based on linearly ordered commutative semigroups.


## Contents

1 Overview ..... 2
2 Big Sum over Finite Sets in Abelian Semigroups ..... 3
2.1 Generic Abelian semigroup operation over a set ..... 3
2.2 Generalized summation over a set ..... 13
2.2.1 Properties in more restricted classes of structures ..... 15
3 Algebras for Aggregation and Minimisation ..... 18
4 Matrix Algebras for Aggregation and Minimisation ..... 22
4.1 Aggregation Orders and Finite Sums ..... 22
4.2 Matrix Aggregation ..... 26
4.3 Aggregation Lattices ..... 28
4.4 Matrix Minimisation ..... 33
4.5 Linear Aggregation Lattices ..... 35
5 Algebras for Aggregation and Minimisation with a Linear Order ..... 45
5.1 Linearly Ordered Commutative Semigroups ..... 45
5.2 Linearly Ordered Commutative Monoids ..... 49
5.3 Linearly Ordered Commutative Monoids with a Least Element ..... 52
5.4 Linearly Ordered Commutative Monoids with a Greatest El- ement ..... 55
5.5 Linearly Ordered Commutative Monoids with a Least Ele- ment and a Greatest Element ..... 60
5.6 Constant Aggregation ..... 63
5.7 Counting Aggregation ..... 66

## 6 An Operation to Select Components in Algebras with Minimisation

## 1 Overview

This document describes the following four theory files:

* Big sums over semigroups generalises parts of Isabelle/HOL's theory of finite summation Groups_Big.thy from commutative monoids to commutative semigroups with a unit element only on the image of the semigroup operation.
* Aggregation Algebras introduces s-algebras, m-algebras and m-Kleenealgebras with operations for aggregating the elements of a weight matrix and finding the edge with minimal weight.
* Matrix Aggregation Algebras introduces aggregation orders, aggregation lattices and linear aggregation lattices. Matrices over these structures form s-algebras and m-algebras.
* Linear Aggregation Algebras shows numerous instances based on linearly ordered commutative semigroups. They include aggregations used for the minimum weight spanning tree problem and for the minimum bottleneck spanning tree problem, as well as arbitrary t-norms and t -conorms.

Three theory files, which were originally part of this entry, have been moved elsewhere:

* A theory for total-correctness proofs in Hoare logic became part of Isabelle/HOL's theory Hoare/Hoare_Logic.thy.
* A theory with simple total-correctness proof examples became Isabelle/HOL's theory Hoare/ExamplesTC.thy.
* A theory proving total correctness of Kruskal's and Prim's minimum spanning tree algorithms based on m-Kleene-algebras using Hoare logic was split into two theories that became part of AFP entry [6].
Following a refactoring, the selection of components of graphs in m-Kleenealgebras, which was originally part of Nicolas Robinson-O'Brien's theory Relational_Minimum_Spanning_Trees/Boruvka.thy, has been moved into a new theory in this entry.

The development is based on Stone-Kleene relation algebras [3, 2]. The algebras for aggregation and minimisation, their application to weighted graphs and the verification of Prim's and Kruskal's minimum spanning tree algorithms, and various instances of aggregation are described in $[1,4,5]$. Related work is discussed in these papers.

## 2 Big Sum over Finite Sets in Abelian Semigroups

theory Semigroups-Big
imports Main
begin
This theory is based on Isabelle/HOL's Groups-Big.thy written by T. Nipkow, L. C. Paulson, M. Wenzel and J. Avigad. We have generalised a selection of its results from Abelian monoids to Abelian semigroups with an element that is a unit on the image of the semigroup operation.

### 2.1 Generic Abelian semigroup operation over a set

```
locale abel-semigroup-set \(=\) abel-semigroup +
    fixes \(z::{ }^{\prime} a\) (1)
    assumes \(z\)-neutral \([\operatorname{simp}]: x * y * \mathbf{1}=x * y\)
    assumes z-idem [simp]: \(\mathbf{1} * \mathbf{1}=\mathbf{1}\)
begin
interpretation comp-fun-commute \(f\)
    by standard (simp add: fun-eq-iff left-commute)
interpretation comp?: comp-fun-commute \(f \circ g\)
    by (fact comp-comp-fun-commute)
definition \(F::\left({ }^{\prime} b \Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} b\) set \(\Rightarrow{ }^{\prime} a\)
    where eq-fold: Fg \(A=\) Finite-Set.fold \((f \circ g) 1 A\)
lemma infinite \([\) simp \(]: \neg\) finite \(A \Longrightarrow F g A=\mathbf{1}\)
    by (simp add: eq-fold)
lemma empty \([s i m p]: F g\{ \}=\mathbf{1}\)
    by (simp add: eq-fold)
lemma insert \([\) simp \(]\) : finite \(A \Longrightarrow x \notin A \Longrightarrow F g(\) insert \(x A)=g x * F g A\)
    by (simp add: eq-fold)
lemma remove:
    assumes finite \(A\) and \(x \in A\)
    shows \(F g A=g x * F g(A-\{x\})\)
proof -
    from \(\langle x \in A\rangle\) obtain \(B\) where \(B: A=\) insert \(x B\) and \(x \notin B\)
        by (auto dest: mk-disjoint-insert)
    moreover from 〈finite \(A\) 〉 \(B\) have finite \(B\) by simp
    ultimately show ?thesis by simp
qed
lemma insert-remove: finite \(A \Longrightarrow F g(\) insert \(x A)=g x * F g(A-\{x\})\)
```

```
by (cases x }\inA)\mathrm{ (simp-all add: remove insert-absorb)
```

lemma insert-if: finite $A \Longrightarrow F g($ insert $x A)=($ if $x \in A$ then $F g A$ else $g x *$
$F g A$ )
by (cases $x \in A$ ) (simp-all add: insert-absorb)
lemma neutral: $\forall x \in A . g x=\mathbf{1} \Longrightarrow F g A=\mathbf{1}$
by (induct $A$ rule: infinite-finite-induct) simp-all
lemma neutral-const $[$ simp $]: F(\lambda$-. 1) $A=\mathbf{1}$
by (simp add: neutral)
lemma $F$-one $[$ simp $]: F g A * \mathbf{1}=F g A$
proof -
have $\Lambda f b B . F f\left(\right.$ insert $\left.\left(b::^{\prime} b\right) B\right) * \mathbf{1}=F f($ insert $b B) \vee$ infinite $B$
using insert-remove by fastforce
then show?thesis
by (metis (no-types) all-not-in-conv empty z-idem infinite insert-if)
qed
lemma one- $F[$ simp $]: \mathbf{1} * F g A=F g A$
using $F$-one commute by auto
lemma $F$-g-one $[$ simp $]: F(\lambda x . g x * \mathbf{1}) A=F g A$
apply (induct A rule: infinite-finite-induct)
apply simp
apply simp
by (metis one-F assoc insert)
lemma union-inter:
assumes finite $A$ and finite $B$
shows $F g(A \cup B) * F g(A \cap B)=F g A * F g B$
- The reversed orientation looks more natural, but LOOPS as a simprule!
using assms
proof (induct A)
case empty
then show ?case by simp
next
case (insert $x A$ )
then show? case
by (auto simp: insert-absorb Int-insert-left commute $[o f-g x]$ assoc
left-commute)
qed
corollary union-inter-neutral:
assumes finite $A$ and finite $B$
and $\forall x \in A \cap B . g x=\mathbf{1}$
shows $F g(A \cup B)=F g A * F g B$
using assms by (simp add: union-inter [symmetric] neutral)

```
corollary union-disjoint:
    assumes finite }A\mathrm{ and finite B
    assumes }A\capB={
    shows Fg(A\cupB)=FgA*Fg B
    using assms by (simp add: union-inter-neutral)
lemma union-diff2:
    assumes finite }A\mathrm{ and finite }
    shows Fg(A\cupB)=Fg(A-B)*Fg(B-A)*Fg(A\capB)
proof -
    have }A\cupB=A-B\cup(B-A)\cupA\cap
        by auto
    with assms show ?thesis
        by simp (subst union-disjoint, auto)+
qed
lemma subset-diff:
    assumes }B\subseteqA\mathrm{ and finite }
    shows FgA=Fg(A-B)*FgB
proof -
    from assms have finite (A-B) by auto
    moreover from assms have finite B by (rule finite-subset)
    moreover from assms have (A-B)\capB={} by auto
    ultimately have Fg(A-B\cupB)=Fg(A-B)*FgB by (rule
union-disjoint)
    moreover from assms have }A\cupB=A\mathrm{ by auto
    ultimately show ?thesis by simp
qed
lemma setdiff-irrelevant:
    assumes finite }
    shows Fg(A-{x.g x=z})=FgA
    using assms by (induct A) (simp-all add: insert-Diff-if)
lemma not-neutral-contains-not-neutral:
    assumes Fg A\not=1
    obtains }a\mathrm{ where }a\inA\mathrm{ and ga#=1
proof -
    from assms have \existsa\inA.g a\not=1
    proof (induct A rule: infinite-finite-induct)
        case infinite
        then show ?case by simp
    next
        case empty
        then show ?case by simp
    next
        case (insert a A)
        then show ?case by fastforce
```

```
    qed
    with that show thesis by blast
qed
lemma reindex:
    assumes inj-on h A
    shows Fg(h'A)=F(g\circh)A
proof (cases finite A)
    case True
    with assms show ?thesis
        by (simp add: eq-fold fold-image comp-assoc)
next
    case False
    with assms have \neg finite ( }\mp@subsup{h}{}{\prime}\mathrm{ 'A) by (blast dest: finite-imageD)
    with False show ?thesis by simp
qed
lemma cong [fundef-cong]:
    assumes }A=
    assumes g-h: \x. x }\inB\Longrightarrowgx=h
    shows Fg A = FhB
    using g-h unfolding <A = B>
    by (induct B rule: infinite-finite-induct) auto
lemma strong-cong [cong]:
    assumes }A=B\bigwedgex.x\inB=\operatorname{simp=> gx=hx
    shows F (\lambdax.gx) A = F (\lambdax.hx) B
    by (rule cong) (use assms in <simp-all add: simp-implies-def〉)
lemma reindex-cong:
    assumes inj-on l B
    assumes A=l`}
    assumes }\bigwedgex.x\inB\Longrightarrowg(lx)=h
    shows Fg A = Fh B
    using assms by (simp add: reindex)
lemma UNION-disjoint:
    assumes finite I and \foralli\inI. finite (A i)
        and }\foralli\inI.\forallj\inI.i\not=j\longrightarrowAi\capAj={
    shows Fg(U(A'I)) = F (\lambdax.Fg(Ax))I
    apply (insert assms)
    apply (induct rule: finite-induct)
    apply simp
    apply atomize
    apply (subgoal-tac }\foralli\inFa. x\not=i
    prefer 2 apply blast
    apply (subgoal-tac A x \cap\bigcup(A'Fa)={})
    prefer 2 apply blast
    apply (simp add: union-disjoint)
```

```
    done
lemma Union-disjoint:
    assumes }\forallA\inC\mathrm{ . finite }A\forallA\inC.\forallB\inC.A\not=B\longrightarrowA\capB={
    shows Fg(UC)=(F\circF)gC
proof (cases finite C)
    case True
    from UNION-disjoint [OF this assms] show ?thesis by simp
next
    case False
    then show ?thesis by (auto dest: finite-UnionD intro: infinite)
qed
lemma distrib: F (\lambdax.gx*hx) A=FgA*FhA
    by (induct A rule: infinite-finite-induct) (simp-all add: assoc commute
left-commute)
lemma Sigma:
    finite }A\Longrightarrow\forallx\inA. finite (Bx)\LongrightarrowF(\lambdax.F(gx)(Bx))A=F(case-prod g
(SIGMA x:A. B x)
    apply (subst Sigma-def)
    apply (subst UNION-disjoint)
        apply assumption
        apply simp
        apply blast
    apply (rule cong)
    apply rule
    apply (simp add: fun-eq-iff)
    apply (subst UNION-disjoint)
            apply simp
        apply simp
    apply blast
    apply (simp add: comp-def)
    done
lemma related:
    assumes Re: R11
        and Rop: \forallx1 y1 x2 y2. R x1 x2 ^R y1 y2 \longrightarrowR (x1* y1) (x2 * y2)
        and fin: finite S
        and }R\mathrm{ -h-g: }\forallx\inS.R(hx) (gx
    shows R (FhS) (FgS)
    using fin by (rule finite-subset-induct) (use assms in auto)
lemma mono-neutral-cong-left:
    assumes finite T
    and S\subseteqT
    and}\foralli\inT-S.hi=
    and }\x.x\inS\Longrightarrowgx=h
shows Fg S = FhT
```


## proof-

have $e q$ : $T=S \cup(T-S)$ using $\langle S \subseteq T\rangle$ by blast
have $d: S \cap(T-S)=\{ \}$ using $\langle S \subseteq T\rangle$ by blast
from 〈finite $T\rangle\langle S \subseteq T\rangle$ have $f$ : finite $S$ finite $(T-S)$
by (auto intro: finite-subset)
show ?thesis using assms(4)
by (simp add: union-disjoint [OF $f$ d, unfolded eq [symmetric]] neutral [OF $\operatorname{assms}(3)]$ )
qed
lemma mono-neutral-cong-right:
finite $T \Longrightarrow S \subseteq T \Longrightarrow \forall i \in T-S . g i=\mathbf{1} \Longrightarrow(\bigwedge x . x \in S \Longrightarrow g x=h x)$ $F g T=F h S$
by (auto intro!: mono-neutral-cong-left [symmetric])
lemma mono-neutral-left: finite $T \Longrightarrow S \subseteq T \Longrightarrow \forall i \in T-S . g i=\mathbf{1} \Longrightarrow F g$ $S=F g T$
by (blast intro: mono-neutral-cong-left)
lemma mono-neutral-right: finite $T \Longrightarrow S \subseteq T \Longrightarrow \forall i \in T-S . g i=\mathbf{1} \Longrightarrow F$ $g T=F g S$
by (blast intro!: mono-neutral-left [symmetric])
lemma mono-neutral-cong:
assumes [simp]: finite $T$ finite $S$
and $*: \bigwedge i . i \in T-S \Longrightarrow h i=\mathbf{1} \bigwedge i . i \in S-T \Longrightarrow g i=\mathbf{1}$
and $g h: \bigwedge x . x \in S \cap T \Longrightarrow g x=h x$
shows $F g S=F h T$
proof-
have $F g S=F g(S \cap T)$
by (rule mono-neutral-right)(auto intro: *)
also have $\ldots=F h(S \cap T)$ using refl gh by (rule cong)
also have $\ldots=F h T$
by (rule mono-neutral-left)(auto intro: *)
finally show? ?hesis .
qed
lemma reindex-bij-betw: bij-betw $h S T \Longrightarrow F(\lambda x . g(h x)) S=F g T$
by (auto simp: bij-betw-def reindex)
lemma reindex-bij-witness:
assumes witness:
$\bigwedge a . a \in S \Longrightarrow i(j a)=a$
$\bigwedge a . a \in S \Longrightarrow j a \in T$
$\bigwedge$ 人 $. b \in T \Longrightarrow j(i b)=b$
$\bigwedge b . b \in T \Longrightarrow i b \in S$
assumes eq:
$\bigwedge a . a \in S \Longrightarrow h(j a)=g a$

```
    shows Fg S = FhT
proof -
    have bij-betw j ST
    using bij-betw-byWitness[where }A=S\mathrm{ and }f=j\mathrm{ and }\mp@subsup{f}{}{\prime}=i\mathrm{ and }\mp@subsup{A}{}{\prime}=T]\mathrm{ witness
by auto
    moreover have Fg S=F(\lambdax.h(j x))S
    by (intro cong) (auto simp: eq)
    ultimately show ?thesis
    by (simp add: reindex-bij-betw)
qed
lemma reindex-bij-betw-not-neutral:
    assumes fin: finite S' finite T'
    assumes bij: bij-betw h(S-S')(T- T')
    assumes nn:
        \a.a\in S'\Longrightarrowg(ha)=z
    \b.b G T' \Longrightarrowgb=z
    shows}F(\lambdax.g(hx))S=Fg
proof -
    have [simp]: finite S \longleftrightarrow finite T
        using bij-betw-finite[OF bij] fin by auto
    show ?thesis
    proof (cases finite S)
        case True
        with nn have F (\lambdax.g(hx)) S=F(\lambdax.g(hx)) (S-S')
            by (intro mono-neutral-cong-right) auto
        also have ... = Fg(T- T')
            using bij by (rule reindex-bij-betw)
        also have ... = FgT
            using nn<finite S` by (intro mono-neutral-cong-left) auto
            finally show ?thesis.
    next
        case False
        then show ?thesis by simp
    qed
qed
lemma reindex-nontrivial:
    assumes finite A
    and nz: \bigwedgex y. x\inA\Longrightarrowy\inA\Longrightarrowx\not=y\Longrightarrowhx=hy\Longrightarrowg(hx)=\mathbf{1}
    shows Fg(h'A)=F(g\circh)A
proof (subst reindex-bij-betw-not-neutral [symmetric])
    show bij-betw h (A-{x\inA. (g\circh) x=1}) (h'A - h'{x\inA. (g\circh)x=
1})
    using nz by (auto intro!: inj-onI simp: bij-betw-def)
qed (use <finite A> in auto)
lemma reindex-bij-witness-not-neutral:
    assumes fin: finite S' finite T'
```

assumes witness:
\a. $a \in S-S^{\prime} \Longrightarrow i(j a)=a$
$\bigwedge a . a \in S-S^{\prime} \Longrightarrow j a \in T-T^{\prime}$
^b. $b \in T-T^{\prime} \Longrightarrow j(i b)=b$
$\bigwedge b . b \in T-T^{\prime} \Longrightarrow i b \in S-S^{\prime}$
assumes $n n$ :
$\bigwedge a . a \in S^{\prime} \Longrightarrow g a=z$
$\bigwedge b . b \in T^{\prime} \Longrightarrow h b=z$
assumes eq:
$\bigwedge a . a \in S \Longrightarrow h(j a)=g a$
shows $F g S=F h T$
proof -
have bij: bij-betw $j\left(S-\left(S^{\prime} \cap S\right)\right)\left(T-\left(T^{\prime} \cap T\right)\right)$
using witness by (intro bij-betw-byWitness $\left[\right.$ where $\left.f^{\prime}=i\right]$ ) auto
have $F$-eq: $F g S=F(\lambda x . h(j x)) S$
by (intro cong) (auto simp: eq)
show ?thesis
unfolding $F$-eq using fin $n n e q$
by (intro reindex-bij-betw-not-neutral $[O F-b i j]$ ) auto
qed
lemma delta-remove:
assumes fS: finite $S$
shows $F(\lambda k$. if $k=a$ then $b k$ else $c k) S=($ if $a \in S$ then $b a * F c(S-\{a\})$
else $F c(S-\{a\}))$
proof -
let ?f $=(\lambda k$. if $k=a$ then $b k$ else $c k)$
show ?thesis
proof (cases $a \in S$ )
case False
then have $\forall k \in S$. ?f $k=c k$ by $\operatorname{simp}$
with False show ?thesis by simp
next
case True
let ? $A=S-\{a\}$
let $? B=\{a\}$
from True have eq: $S=$ ? $A \cup$ ?B by blast
have $d j: ? A \cap ? B=\{ \}$ by simp
from $f S$ have $f A B$ : finite ?A finite ?B by auto
have $F$ ?f $S=F$ ?f ? $A * F$ ?f ? $B$
using union-disjoint [OF fAB dj, of ?f, unfolded eq [symmetric]] by simp with True show ?thesis
using abel-semigroup-set.remove abel-semigroup-set-axioms fS by fastforce
qed
qed
lemma delta [simp]:
assumes fS: finite $S$
shows $F(\lambda k$. if $k=a$ then $b k$ else $\mathbf{1}) S=($ if $a \in S$ then $b a * \mathbf{1}$ else $\mathbf{1})$

```
    by (simp add: delta-remove [OF assms])
lemma delta' [simp]:
    assumes fin: finite S
    shows F ( }\lambdak\mathrm{ . if }a=k\mathrm{ then b k else 1) S = (if a GS then b a* 1 else 1)
    using delta [OF fin, of a b, symmetric] by (auto intro: cong)
lemma If-cases:
    fixes P :: 'b b bool and gh :: 'b > 'a
    assumes fin: finite A
    shows F(\lambdax. if P x then hx else g x) A=Fh(A\cap{x.P x})*Fg(A\cap-
{x.P x})
proof -
    have a:A=A\cap{x.Px}\cupA\cap-{x.P x} (A\cap{x.P x})\cap(A\cap-{x.P
x})={}
    by blast+
    from fin have f: finite ( }A\cap{x.Px})\mathrm{ finite ( }A\cap-{x.Px})\mathrm{ by auto
    let ?g=\lambdax. if P x then hx else g x
    from union-disjoint [OF fa(2), of ?g] a(1) show ?thesis
    by (subst (1 2) cong) simp-all
qed
lemma cartesian-product: F (\lambdax. F (gx) B) A = F (case-prod g) (A\timesB)
    apply (rule sym)
    apply (cases finite A)
    apply (cases finite B)
    apply (simp add: Sigma)
    apply (cases A = {})
    apply simp
    apply simp
    apply (auto intro: infinite dest: finite-cartesian-productD2)
    apply (cases B={})
    apply (auto intro: infinite dest: finite-cartesian-productD1)
    done
lemma inter-restrict:
    assumes finite }
    shows Fg(A\capB)=F(\lambdax. if x\inB then g x else 1) A
proof -
    let ?g = \lambdax. if }x\inA\capB\mathrm{ then g x else 1
    have }\foralli\inA-A\capB\mathrm{ . (if }i\inA\capB\mathrm{ then g i else 1)=1 by simp
    moreover have }A\capB\subseteqA\mathrm{ by blast
    ultimately have F?g (A\capB)=F?g A
        using <finite A〉 by (intro mono-neutral-left) auto
    then show ?thesis by simp
qed
lemma inter-filter:
    finite }A\LongrightarrowFg{x\inA.Px}=F(\lambdax. if P x then g x else 1) A
```

```
    by (simp add: inter-restrict [symmetric, of A {x. P x} g, simplified
mem-Collect-eq] Int-def)
lemma Union-comp:
    assumes }\forallA\inB\mathrm{ . finite }
```



```
"gx=1
    shows Fg(UB)=(F\circF)gB
    using assms
proof (induct B rule: infinite-finite-induct)
    case (infinite A)
    then have }\neg\mathrm{ finite ( }\bigcup\mathrm{ A) by (blast dest: finite-UnionD)
    with infinite show ?case by simp
next
    case empty
    then show ?case by simp
next
    case (insert A B)
    then have finite A finite B finite ( }\bigcupB)A\not\in
        and }\forallx\inA\cap\bigcupB.gx=
        and H:Fg(\bigcupB)=(F\circF)gB by auto
    then have Fg(A\cup\bigcupB)=FgA*Fg(\bigcupB)
        by (simp add: union-inter-neutral)
    with 〈finite B\rangle\langleA\not\inB\rangle\mathrm{ show ?case}
        by (simp add: H)
qed
lemma swap:F(\lambdai.F(gi)B)A=F(\lambdaj.F(\lambdai.gij)A)B
    unfolding cartesian-product
    by (rule reindex-bij-witness [where i=\lambda(i,j). (j,i) and j=\lambda(i,j). (j,i)])
auto
lemma swap-restrict:
    finite }A\Longrightarrow\mathrm{ finite }B
        F(\lambdax.F(gx) {y. y\inB\wedgeRxy}) A=F(\lambday.F(\lambdax.gxy){x.x\inA\wedgeR
x y}) B
    by (simp add: inter-filter) (rule swap)
lemma Plus:
    fixes }A\mathrm{ :: 'b set and B :: 'c set
    assumes fin: finite }A\mathrm{ finite }
    shows Fg(A<+>B)=F(g\circInl)A*F(g\circInr)B
proof -
    have }A<+>B=Inl`A\cupInr ' B by aut
    moreover from fin have finite (Inl'A) finite (Inr ` B) by auto
    moreover have Inl ' }A\cap\mathrm{ Inr ' }B={}\mathrm{ by auto
    moreover have inj-on Inl A inj-on Inr B by (auto intro: inj-onI)
    ultimately show ?thesis
        using fin by (simp add: union-disjoint reindex)
```

qed
lemma same－carrier：
assumes finite $C$
assumes subset：$A \subseteq C B \subseteq C$
assumes trivial：$\bigwedge a . a \in C-A \Longrightarrow g a=\mathbf{1} \bigwedge b . b \in C-B \Longrightarrow h b=\mathbf{1}$
shows $F g A=F h B \longleftrightarrow F g C=F h C$
proof－
have finite $A$ and finite $B$ and finite $(C-A)$ and finite $(C-B)$
using 〈finite $C$ 〉 subset by（auto elim：finite－subset）
from subset have $[$ simp $]: A-(C-A)=A$ by auto
from subset have $[$ simp］：$B-(C-B)=B$ by auto
from subset have $C=A \cup(C-A)$ by auto
then have $F g C=F g(A \cup(C-A))$ by simp
also have $\ldots=F g(A-(C-A)) * F g(C-A-A) * F g(A \cap(C-A))$
using 〈finite $A\rangle\langle$ finite $(C-A)\rangle$ by（simp only：union－diff2）
finally have $*: F g C=F g A$ using trivial by simp
from subset have $C=B \cup(C-B)$ by auto
then have $F h C=F h(B \cup(C-B))$ by simp
also have $\ldots=F h(B-(C-B)) * F h(C-B-B) * F h(B \cap(C-B))$
using 〈finite $B\rangle\langle$ finite $(C-B)\rangle$ by（simp only：union－diff2）
finally have $F h C=F h B$
using trivial by simp
with $*$ show ？thesis by simp
qed
lemma same－carrierI：
assumes finite $C$
assumes subset：$A \subseteq C B \subseteq C$
assumes trivial：$\bigwedge a . a \in C-A \Longrightarrow g a=1 \bigwedge b . b \in C-B \Longrightarrow h b=1$
assumes $F g C=F h C$
shows $F g A=F h B$
using assms same－carrier［of C A B］by simp

## end

## 2．2 Generalized summation over a set

```
no-notation Sum(\sum)
```

class ab-semigroup-add-0 $=$ zero + ab-semigroup-add +
assumes zero-neutral $[$ simp $]: x+y+0=x+y$
assumes zero-idem [simp]: $0+0=0$
begin
sublocale sum-0: abel-semigroup-set plus 0
defines sum-0 $=$ sum-0.F
by unfold－locales simp－all

```
abbreviation Sum-0 (\sum)
    where }\sum\equiv\operatorname{sum-0}(\lambdax.x
end
context comm-monoid-add
begin
subclass ab-semigroup-add-0
    by unfold-locales simp-all
end
```

    Now: lots of fancy syntax. First, sum-0 \((\lambda x . e) A\) is written \(\sum x \in A . e\).
    syntax (ASCII)
-sum :: pttrn $\Rightarrow$ ' $a$ set $\Rightarrow$ ' $b \Rightarrow{ }^{\prime} b::$ comm-monoid-add ((3SUM (-/:-)./ -) [0, 51,
10] 10)
syntax
-sum $::$ pttrn $\Rightarrow{ }^{\prime} a$ set $\Rightarrow{ }^{\prime} b \Rightarrow{ }^{\prime} b::$ comm-monoid-add ((2 $\left.\sum(-/ \in-) . /-\right)[0,51$,
10] 10)
translations - Beware of argument permutation!
$\sum i \in A . b \rightleftharpoons$ CONST sum-0 ( $\lambda i$. b) $A$
Instead of $\sum x \in\{x . P\} . e$ we introduce the shorter $\sum x \mid P . e$.
syntax (ASCII)
-qsum :: pttrn $\Rightarrow$ bool $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a((3 S U M-\mid /-. /-)[0,0,10] 10)$
syntax
-qsum :: pttrn $\Rightarrow$ bool $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\left(\left(2 \sum-\mid(-) . /-\right)[0,0,10] 10\right)$
translations
$\sum x \mid P . t=>$ CONST sum-0 $(\lambda x . t)\{x . P\}$
print-translation <
let
fun sum-tr' $[$ Abs $(x, T x, t)$, Const $(@\{$ const-syntax Collect $\},-) \$ A b s(y, T y$,
$P)]=$
if $x<>y$ then raise Match
else
let
val $x^{\prime}=$ Syntax-Trans.mark-bound-body $(x, T x)$;
val $t^{\prime}=$ subst-bound $\left(x^{\prime}, t\right)$;
val $P^{\prime}=$ subst-bound $\left(x^{\prime}, P\right)$;
in
Syntax.const @\{syntax-const-qsum\} \$ Syntax-Trans.mark-bound-abs (x,
$T x) \$ P^{\prime} \$ t^{\prime}$
end
| sum-tr' - = raise Match;
in $[(@\{$ const-syntax sum-0 $\}, K$ sum-tr' $)]$ end

```
lemma (in ab-semigroup-add-0) sum-image-gen-0:
    assumes fin: finite S
    shows sum-0 g S = sum-0 (\lambday. sum-0 g {x. x \inS^fx=y})(f'S)
proof -
    have {y.y\in f'S\wedgefx=y}={fx} if }x\inS\mathrm{ for }
        using that by auto
    then have sum-0 g S = sum-0(\lambdax. sum-0 (\lambday.gx) {y. y\inf`S\wedgefx=y})S
        by simp
    also have ... = sum-0 ( }\lambday.\operatorname{sum-0}g{x.x\inS\wedgefx=y})(f'S
        by (rule sum-0.swap-restrict [OF fin finite-imageI [OF fin]])
    finally show ?thesis.
qed
```


### 2.2.1 Properties in more restricted classes of structures

```
lemma sum-Un2:
    assumes finite \((A \cup B)\)
    shows sum-0 \(f(A \cup B)=\) sum-0 \(f(A-B)+\) sum-0 \(f(B-A)+\) sum-0 \(f(A\)
\(\cap B\) )
proof -
    have \(A \cup B=A-B \cup(B-A) \cup A \cap B\)
        by auto
    with assms show ?thesis
        by \(\operatorname{simp}\) (subst sum-0.union-disjoint, auto)+
qed
class ordered-ab-semigroup-add- \(0=a b\)-semigroup-add- \(0+\)
ordered-ab-semigroup-add
begin
lemma add-nonneg-nonneg [simp]: \(0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a+b\)
    using add-mono[of 0 a 0 b] by simp
lemma add-nonpos-nonpos: \(a \leq 0 \Longrightarrow b \leq 0 \Longrightarrow a+b \leq 0\)
    using add-mono[of a 0 blal by simp
end
lemma (in ordered-ab-semigroup-add-0) sum-mono:
    \((\bigwedge i . i \in K \Longrightarrow f i \leq g i) \Longrightarrow\left(\sum i \in K . f i\right) \leq\left(\sum i \in K . g i\right)\)
    by (induct \(K\) rule: infinite-finite-induct) (use add-mono in auto)
lemma (in ordered-ab-semigroup-add-0) sum-mono-00:
    \((\bigwedge i . i \in K \Longrightarrow f i+0 \leq g i+0) \Longrightarrow\left(\sum i \in K . f i\right) \leq\left(\sum i \in K . g i\right)\)
proof (induct \(K\) rule: infinite-finite-induct)
    case (infinite \(A\) )
    then show ?case by simp
next
    case empty
```

```
    then show ?case by simp
next
    case (insert x F)
    then show ?case
    proof -
        fix x :: 'b and F :: 'b set
        assume a1: finite F
    assume a2: x\not\inF
    assume a3:(\bigwedgei. i }\\F\Longrightarrowfi+0\leqgi+0)\Longrightarrowsum-0 f F \leqsum-0 g F
    assume a4: \bigwedgei. i\in insert x F\Longrightarrowfi+0\leqgi+0
    obtain bb :: 'b where
        f5:bb\inF^\negf bb+0\leqgbb+0\vee sum-0 f F\leqsum-0 g F
        using a3 by blast
    have }\forallb.x\not=b\veefb+0\leqgb+
        using a& by simp
    then have }\foralla\mathrm{ aa. f x+0 +a}\leqgx+0+aa\vee\nega\leqa
        using add-mono by blast
    then show sum-0 f(insert x F)\leqsum-0 g (insert x F)
        using f5 a4 a2 a1 by (metis (no-types) add-assoc insert-iff sum-0.insert
sum-0.one-F)
    qed
qed
lemma (in ordered-ab-semigroup-add-0) sum-mono-0:
    (\bigwedgei. i\inK\Longrightarrowfi+0\leqgi)\Longrightarrow(\sumi\inK.fi)\leq(\sumi\inK.gi)
    apply (rule sum-mono-00)
    by (metis add-right-mono zero-neutral)
context ordered-ab-semigroup-add-0
begin
lemma sum-nonneg:(\x. x \inA\Longrightarrow0\leqfx)\Longrightarrow0\leqsum-0f A
proof (induct A rule: infinite-finite-induct)
    case infinite
    then show ?case by simp
next
    case empty
    then show ?case by simp
next
    case (insert x F)
    then have 0 + 0\leqfx+ sum-0 f F by (blast intro:add-mono)
    with insert show ?case by simp
qed
lemma sum-nonpos:(\bigwedgex. x \inA\Longrightarrowfx\leq0)\Longrightarrowsum-0 f A\leq0
proof (induct A rule: infinite-finite-induct)
    case infinite
    then show ?case by simp
next
```

```
    case empty
    then show ?case by simp
next
    case (insert x F)
    then have f x + sum-0 f F\leq0+0 by (blast intro: add-mono)
    with insert show ?case by simp
qed
lemma sum-mono2:
    assumes fin: finite B
    and sub: A\subseteqB
    and nn:\bigwedgeb.b\inB-A\Longrightarrow0\leqfb
    shows sum-0 f A \leq sum-0 f B
proof -
    have sum-0 f A sum-0 f A + sum-0 f ( }B-A
    by (metis add-left-mono sum-0.F-one nn sum-nonneg)
    also from fin finite-subset[OF sub fin] have ... = sum-0 f (A\cup (B-A))
    by (simp add: sum-0.union-disjoint del:Un-Diff-cancel)
    also from sub have }A\cup(B-A)=B\mathrm{ by blast
    finally show ?thesis .
qed
lemma sum-le-included:
    assumes finite s finite t
    and }\forally\int.0\leqgy(\forallx\ins.\existsy\int.i y=x\wedgefx\leqgy
    shows sum-0 f s}\leq\mathrm{ sum-0 g t
proof -
    have sum-0 fs\leq sum-0(\lambday. sum-0 g {x. x\int ^ix=y})s
    proof (rule sum-mono-0)
        fix }
        assume y\ins
        with assms obtain z where z:z\inty=izfy\leqgz by auto
        hence f y + 0 \leq sum-0 g{z}
            by (metis Diff-eq-empty-iff add-commute finite.simps add-left-mono
sum-0.empty sum-0.insert-remove subset-insertI)
            also have ... \leq sum-0 g {x\int.ix=y}
            apply (rule sum-mono2)
            using assms z by simp-all
        finally show fy+0\leqsum-0 g{x\int.ix=y}.
    qed
    also have ... \leq sum-0 (\lambday. sum-0 g {x. x\int ^ix=y})(i't)
        using assms(2-4) by (auto intro!: sum-mono2 sum-nonneg)
    also have ... s sum-0 g t
        using assms by (auto simp: sum-image-gen-0[symmetric])
    finally show ?thesis.
qed
end
```

lemma sum-comp-morphism:

```
\(h 0=0 \Longrightarrow(\bigwedge x y . h(x+y)=h x+h y) \Longrightarrow s u m-0(h \circ g) A=h(s u m-0 g\)
A)
    by (induct A rule: infinite-finite-induct) simp-all
lemma sum-cong-Suc:
    assumes \(0 \notin A \bigwedge x\). Suc \(x \in A \Longrightarrow f(\) Suc \(x)=g(\) Suc \(x)\)
    shows sum-0 \(f A=\) sum-0 \(g A\)
proof (rule sum-0.cong)
    fix \(x\)
    assume \(x \in A\)
    with \(\operatorname{assms}(1)\) show \(f x=g x\)
        by (cases \(x\) ) (auto intro!: assms(2))
qed simp-all
end
```


## 3 Algebras for Aggregation and Minimisation

This theory gives algebras with operations for aggregation and minimisation. In the weighted-graph model of matrices over (extended) numbers, the operations have the following meaning. The binary operation + adds the weights of corresponding edges of two graphs. Addition does not have to be the standard addition on numbers, but can be any aggregation satisfying certain basic properties as demonstrated by various models of the algebras in another theory. The unary operation sum adds the weights of all edges of a graph. The result is a single aggregated weight using the same aggregation as + but applied internally to the edges of a single graph. The unary operation minarc finds an edge with a minimal weight in a graph. It yields the position of such an edge as a regular element of a Stone relation algebra.

We give axioms for these operations which are sufficient to prove the correctness of Prim's and Kruskal's minimum spanning tree algorithms. The operations have been proposed and axiomatised first in [1] with simplified axioms given in [4]. The present version adds two axioms to prove total correctness of the spanning tree algorithms as discussed in [5].
theory Aggregation-Algebras
imports Stone-Kleene-Relation-Algebras.Kleene-Relation-Algebras
begin
context sup
begin

```
no-notation
    sup (infixl + 65)
```


## end

## context plus

begin

## notation

plus (infixl +65 )
end
We first introduce s-algebras as a class with the operations + and sum. Axiom sum-plus-right-isotone states that for non-empty graphs, the operation + is $\leq$-isotone in its second argument on the image of the aggregation operation sum. Axiom sum-bot expresses that the empty graph contributes no weight. Axiom sum-plus generalises the inclusion-exclusion principle to sets of weights. Axiom sum-conv specifies that reversing edge directions does not change the aggregated weight. In instances of s-algebra, aggregated weights can be partially ordered.

```
class sum =
    fixes sum :: 'a }\mp@subsup{|}{}{\prime}
class s-algebra = stone-relation-algebra + plus + sum +
    assumes sum-plus-right-isotone: x 
sumz + sum y
    assumes sum-bot: sum x + sum bot = sum x
    assumes sum-plus: sum x + sum y = sum (x\sqcupy) + sum (x\sqcapy)
    assumes sum-conv: sum (x }\mp@subsup{x}{}{T})=\operatorname{sum}
begin
lemma sum-disjoint:
    assumes }x\sqcapy=bo
        shows sum ((x\sqcupy)\sqcapz)=\operatorname{sum}(x\sqcapz)+\operatorname{sum}(y\sqcapz)
    by (subst sum-plus) (metis assms inf.sup-monoid.add-assoc
inf.sup-monoid.add-commute inf-bot-left inf-sup-distrib2 sum-bot)
lemma sum-disjoint-3:
    assumes w\sqcapx=bot
        and w\sqcapy=bot
        and }x\sqcapy=bo
        shows sum ((w\sqcupx\sqcupy)\sqcapz)=\operatorname{sum}(w\sqcapz)+\operatorname{sum}(x\sqcapz)+\operatorname{sum}(y\sqcapz)
    by (metis assms inf-sup-distrib2 sup-idem sum-disjoint)
lemma sum-symmetric:
    assumes }y=\mp@subsup{y}{}{T
        shows sum ( }\mp@subsup{x}{}{T}\sqcapy)=\operatorname{sum}(x\sqcapy
    by (metis assms sum-conv conv-dist-inf)
lemma sum-commute:
```

```
sum x + sum y = sum }y+\operatorname{sum}
by (metis inf-commute sum-plus sup-commute)
end
```

We next introduce the operation minarc. Axiom minarc-below expresses that the result of minarc is contained in the graph ignoring the weights. Axiom minarc-arc states that the result of minarc is a single unweighted edge if the graph is not empty. Axiom minarc-min specifies that any edge in the graph weighs at least as much as the edge at the position indicated by the result of minarc, where weights of edges between different nodes are compared by applying the operation sum to single-edge graphs. Axiom sum-linear requires that aggregated weights are linearly ordered, which is necessary for both Prim's and Kruskal's minimum spanning tree algorithms. Axiom finite-regular ensures that there are only finitely many unweighted graphs, and therefore only finitely many edges and nodes in a graph; again this is necessary for the minimum spanning tree algorithms we consider.

```
class minarc \(=\)
    fixes minarc :: ' \(a \Rightarrow{ }^{\prime} a\)
class \(m\)-algebra \(=s\)-algebra + minarc +
    assumes minarc-below: minarc \(x \leq--x\)
    assumes minarc-arc: \(x \neq b o t \longrightarrow \operatorname{arc}(\) minarc \(x)\)
    assumes minarc-min: arc \(y \wedge y \sqcap x \neq b o t \longrightarrow \operatorname{sum}(\) minarc \(x \sqcap x) \leq \operatorname{sum}(y\)
    \(7 x)\)
    assumes sum-linear: sum \(x \leq \operatorname{sum} y \vee \operatorname{sum} y \leq \operatorname{sum} x\)
    assumes finite-regular: finite \(\{x\). regular \(x\}\)
begin
```

Axioms minarc-below and minarc-arc suffice to derive the Tarski rule in Stone relation algebras.

```
subclass stone-relation-algebra-tarski
proof unfold-locales
    fix }
    let ?a = minarc x
    assume 1: regular x
    assume x}\not=bo
    hence arc ?a
        by (simp add: minarc-arc)
    hence top = top * ?a * top
        by (simp add: comp-associative)
    also have .. \leqtop * --x* top
    by (simp add: minarc-below mult-isotone)
    finally show top*x* top = top
    using 1 order.antisym by simp
qed
lemma minarc-bot:
```

```
    minarc bot = bot
    by (metis bot-unique minarc-below regular-closed-bot)
lemma minarc-bot-iff:
    minarc }x=\mathrm{ bot }\longleftrightarrowx=\mathrm{ bot
    using covector-bot-closed inf-bot-right minarc-arc vector-bot-closed minarc-bot
by fastforce
lemma minarc-meet-bot:
    assumes minarc x }\sqcapx=bo
        shows minarc x bot
proof -
    have minarc x }\leq-
        using assms pseudo-complement by auto
    thus ?thesis
        by (metis minarc-below inf-absorb1 inf-import-p inf-p)
qed
lemma minarc-meet-bot-minarc-iff:
    minarc }x\sqcapx=\mathrm{ bot }\longleftrightarrow\mathrm{ minarc }x=\mathrm{ bot
    using comp-inf.semiring.mult-not-zero minarc-meet-bot by blast
lemma minarc-meet-bot-iff:
    minarc }x\sqcapx=\mathrm{ bot }\longleftrightarrowx=\mathrm{ bot
    using inf-bot-right minarc-bot-iff minarc-meet-bot by blast
lemma minarc-regular:
    regular (minarc x)
proof (cases x = bot)
    assume x = bot
    thus ?thesis
        by (simp add: minarc-bot)
next
    assume x\not= bot
    thus ?thesis
        by (simp add: arc-regular minarc-arc)
qed
lemma minarc-selection:
    selection (minarc x}\sqcapy)
    using inf-assoc minarc-regular selection-closed-id by auto
lemma minarc-below-regular:
    regular }x\Longrightarrow\mathrm{ minarc }x\leq
    by (metis minarc-below)
end
```

class $m$-kleene-algebra $=m$-algebra + stone-kleene-relation-algebra
end

## 4 Matrix Algebras for Aggregation and Minimisation

This theory formalises aggregation orders and lattices as introduced in [4]. Aggregation in these algebras is an associative and commutative operation satisfying additional properties related to the partial order and its least element. We apply the aggregation operation to finite matrices over the aggregation algebras, which shows that they form an s-algebra. By requiring the aggregation algebras to be linearly ordered, we also obtain that the matrices form an m-algebra.

This is an intermediate step in demonstrating that weighted graphs form an s-algebra and an m-algebra. The present theory specifies abstract properties for the edge weights and shows that matrices over such weights form an instance of s-algebras and m-algebras. A second step taken in a separate theory gives concrete instances of edge weights satisfying the abstract properties introduced here.

Lifting the aggregation to matrices requires finite sums over elements taken from commutative semigroups with an element that is a unit on the image of the semigroup operation. Because standard sums assume a commutative monoid we have to derive a number of properties of these generalised sums as their statements or proofs are different.
theory Matrix-Aggregation-Algebras
imports Stone-Kleene-Relation-Algebras.Matrix-Kleene-Algebras
Aggregation-Algebras Semigroups-Big
begin
no-notation
inf (infixl $\sqcap 70$ )
and uminus ( - - [81] 80)

### 4.1 Aggregation Orders and Finite Sums

An aggregation order is a partial order with a least element and an associative commutative operation satisfying certain properties. Axiom add-add-bot introduces almost a commutative monoid; we require that bot is a unit only on the image of the aggregation operation. This is necessary since it is not a unit of a number of aggregation operations we are interested in. Axiom add-right-isotone states that aggregation is $\leq$-isotone on the image of the
aggregation operation. Its assumption $x \neq b o t$ is necessary because the introduction of new edges can decrease the aggregated value. Axiom add-bot expresses that aggregation is zero-sum-free.

```
class aggregation-order \(=\) order-bot + ab-semigroup-add +
    assumes add-right-isotone: \(x \neq b\) ot \(\wedge x+\) bot \(\leq y+b o t \longrightarrow x+z \leq y+z\)
    assumes add-add-bot [simp]: \(x+y+b o t=x+y\)
    assumes add-bot: \(x+y=b o t \longrightarrow x=\) bot
begin
abbreviation zero \(\equiv b o t+b o t\)
sublocale aggregation: ab-semigroup-add- 0 where plus \(=\) plus and zero \(=\) zero
    apply unfold-locales
    using add-assoc add-add-bot by auto
lemma add-bot-bot-bot:
    \(x+b o t+b o t+b o t=x+b o t\)
    by \(\operatorname{simp}\)
end
```

We introduce notation for finite sums over aggregation orders. The index variable of the summation ranges over the finite universe of its type. Finite sums are defined recursively using the binary aggregation and $\perp+\perp$ for the empty sum.

```
syntax (xsymbols)
```

    -sum-ab-semigroup-add-0 :: idt \(\Rightarrow{ }^{\prime} a:\) :bounded-semilattice-sup-bot \(\Rightarrow{ }^{\prime} a\left(\left(\sum--\right)\right.\)
    [0,10] 10)

## translations

$\sum_{x} t=>$ XCONST ab-semigroup-add-0.sum-0 XCONST plus (XCONST plus XCONST bot XCONST bot) $(\lambda x . t)\{x$. CONST True $\}$

The following are basic properties of such sums.
lemma agg-sum-bot:
$\left(\sum_{k}\right.$ bot::'a::aggregation-order $)=$ bot + bot
proof (induct rule: infinite-finite-induct)
case (infinite $A$ )
thus ?case
by $\operatorname{simp}$
next
case empty
thus?case
by $\operatorname{simp}$
next
case (insert x F)
thus ?case
by (metis add.commute add-add-bot aggregation.sum-0.insert)

## qed

lemma agg-sum-bot-bot:
$\left(\sum_{k}\right.$ bot + bot::'a::aggregation-order $)=b o t+b o t$
by (rule aggregation.sum-0.neutral-const)
lemma agg-sum-not-bot-1:
fixes $f::{ }^{\prime} a::$ finite $\Rightarrow$ ' $b::$ aggregation-order
assumes $f i \neq b o t$ shows $\left(\sum_{k} f k\right) \neq$ bot
by (metis assms add-bot aggregation.sum-0.remove finite-code mem-Collect-eq)

## lemma agg-sum-not-bot:

fixes $f::\left(' a:: f i n i t e,{ }^{\prime} b::\right.$ aggregation-order) square
assumes $f(i, j) \neq b o t$
shows $\left(\sum_{k} \sum_{l} f(k, l)\right) \neq b o t$
by (metis assms agg-sum-not-bot-1)
lemma agg-sum-distrib:
fixes $f g$ :: ' $a \Rightarrow$ 'b::aggregation-order
shows $\left(\sum_{k} f k+g k\right)=\left(\sum_{k} f k\right)+\left(\sum_{k} g k\right)$
by (rule aggregation.sum-0.distrib)
lemma agg-sum-distrib-2:
fixes $f g::$ ('a,'b::aggregation-order) square
shows $\left(\sum_{k} \sum_{l} f(k, l)+g(k, l)\right)=\left(\sum_{k} \sum_{l} f(k, l)\right)+\left(\sum_{k} \sum_{l} g(k, l)\right)$
proof -
have $\left(\sum_{k} \sum_{l} f(k, l)+g(k, l)\right)=\left(\sum_{k}\left(\sum_{l} f(k, l)\right)+\left(\sum_{l} g(k, l)\right)\right)$
by (metis (no-types) aggregation.sum-0.distrib)
also have $\ldots=\left(\sum_{k} \sum_{l} f(k, l)\right)+\left(\sum_{k} \sum_{l} g(k, l)\right)$
by (metis (no-types) aggregation.sum-0.distrib)
finally show ?thesis
qed
lemma agg-sum-add-bot:
fixes $f::{ }^{\prime} a \Rightarrow{ }^{\prime} b::$ aggregation-order
shows $\left(\sum_{k} f k\right)=\left(\sum_{k} f k\right)+b o t$
by (metis (no-types) add-add-bot aggregation.sum-0.F-one)
lemma agg-sum-add-bot-2:
fixes $f::{ }^{\prime} a \Rightarrow{ }^{\prime} b::$ aggregation-order
shows $\left(\sum_{k} f k+b o t\right)=\left(\sum_{k} f k\right)$
proof -
have $\left(\sum_{k} f k+b o t\right)=\left(\sum_{k} f k\right)+\left(\sum_{k}::^{\prime} a\right.$ bot: : $\left.' b\right)$
using agg-sum-distrib by simp
also have $\ldots=\left(\sum_{k} f k\right)+(b o t+b o t)$
by (metis agg-sum-bot)
also have $\ldots=\left(\sum_{k} f k\right)$

```
    by simp
    finally show ?thesis
    by simp
qed
lemma agg-sum-commute:
    fixes f :: ('a,'b::aggregation-order) square
    shows}(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}f(k,l))=(\mp@subsup{\sum}{l}{}\mp@subsup{\sum}{k}{}f(k,l)
    by (rule aggregation.sum-0.swap)
lemma agg-delta:
    fixes f :: 'a::finite = 'b::aggregation-order
    shows ( }\mp@subsup{\sum}{l}{}\mathrm{ if }l=j\mathrm{ then fl else zero) = fj + bot
    apply (subst aggregation.sum-0.delta)
    apply simp
    by (metis add.commute add.left-commute add-add-bot mem-Collect-eq)
lemma agg-delta-1:
    fixes }f:: 'a::finite => 'b::aggregation-order
    shows ( }\mp@subsup{\sum}{l}{}\mathrm{ if }l=j\mathrm{ then fl else bot) =fj+ bot
proof -
    let ?f = ( }\lambdal.\mathrm{ . if l = j then f l else bot)
    let ?S = {l::'a.True }
    show ?thesis
    proof (cases j ? ?S)
        case False
        thus ?thesis by simp
    next
        case True
    let ?A = ?S - {j}
    let ?B={j}
    from True have eq: ?S = ?A \cup?B
        by blast
    have dj:?A\cap?B={}
        by simp
    have fAB: finite ?A finite ?B
            by auto
    have aggregation.sum-0 ?f ?S = aggregation.sum-0 ?f ?A + aggregation.sum-0
?f ?B
            using aggregation.sum-0.union-disjoint[OF fAB dj, of ?f, unfolded eq
[symmetric]] by simp
    also have ... = aggregation.sum-0 ( }\lambdal.\mathrm{ bot) ?A + aggregation.sum-0 ?f ?B
            by (subst aggregation.sum-0.cong[where ?B=?A]) simp-all
    also have ... = zero + aggregation.sum-0 ?f ?B
            by (metis (no-types, lifting) add.commute add-add-bot
aggregation.sum-0.F-g-one aggregation.sum-0.neutral)
    also have ... = zero + (fj + zero )
            by simp
    also have ... =fj+bot
```

```
            by (metis add.commute add.left-commute add-add-bot)
    finally show ?thesis
    qed
qed
lemma agg-delta-2:
    fixes f ::('a::finite,'b::aggregation-order) square
    shows (\sum }\mp@subsup{k}{k}{}\mp@subsup{\sum}{l}{l}\mathrm{ if }k=i\wedgel=j\mathrm{ then f(k,l) else bot) =f(i,j)+bot
proof -
    have }\forallk.(\mp@subsup{\sum}{l}{l}\mathrm{ if }k=i\wedgel=j\mathrm{ then f (k,l) else bot ) = (if k=i then f (k,j)+
bot else zero)
    proof
        fix }
            have (\mp@subsup{\sum}{l}{}}\mathrm{ if }k=i\wedgel=j\mathrm{ then f (k,l) else bot ) = ( ( }\mp@subsup{l}{l}{}\mathrm{ if l = j then if k = i
then f(k,l) else bot else bot)
            by meson
    also have ... = (if k=i then f (k,j) else bot) + bot
        by (rule agg-delta-1)
    finally show ( }\mp@subsup{\sum}{l}{l}\mathrm{ if }k=i\wedgel=j\mathrm{ then f(k,l) else bot) = (if k=i then f
(k,j) + bot else zero)
        by simp
    qed
```



```
(k,j) + bot else zero)
    using aggregation.sum-0.cong by auto
    also have ... =f(i,j)+bot
        apply (subst agg-delta)
        by simp
    finally show ?thesis
qed
```


### 4.2 Matrix Aggregation

The following definitions introduce the matrix of unit elements, componentwise aggregation and aggregation on matrices. The aggregation of a matrix is a single value, but because s-algebras are single-sorted the result has to be encoded as a matrix of the same type (size) as the input. We store the aggregated matrix value in the 'first' entry of a matrix, setting all other entries to the unit value. The first entry is determined by requiring an enumeration of indices. It does not have to be the first entry; any fixed location in the matrix would work as well.
definition zero-matrix :: ('a,'b::\{plus,bot\}) square (mzero) where mzero $=(\lambda e$. $b o t+b o t)$
definition plus-matrix :: ('a,'b::plus) square $\Rightarrow$ ('a,'b) square $\Rightarrow$ ('a,'b) square
(infixl $\left.\oplus_{M} 65\right)$ where plus-matrix $f g=(\lambda e . f e+g e)$
definition sum-matrix :: ('a::enum,'b::\{plus,bot\}) square $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ square (sum ${ }_{M}$ - [80] 80) where sum-matrix $f=(\lambda(i, j)$. if $i=h d$ enum-class.enum $\wedge j=i$ then $\sum_{k} \sum_{l} f(k, l)$ else bot + bot $)$

Basic properties of these operations are given in the following.

```
lemma bot-plus-bot:
    mbot }\mp@subsup{\oplus}{M}{}\mathrm{ mbot = mzero
    by (simp add: plus-matrix-def bot-matrix-def zero-matrix-def)
lemma sum-bot:
    sum}\mp@subsup{M}{M}{(mbot :: ('a::enum,'b::aggregation-order) square) = mzero
proof (rule ext, rule prod-cases)
    fix ij :: 'a
    have (sum}\mp@subsup{M}{M}{mbot :: ('a,'b) square) (i,j)=(if i=hd enum-class.enum }\wedgej=
then \sum(k::'a) \sum(l::'a) bot else bot + bot)
    by (unfold sum-matrix-def bot-matrix-def) simp
    also have ... = bot + bot
    using agg-sum-bot aggregation.sum-0.neutral by fastforce
    also have ... = mzero (i,j)
    by (simp add: zero-matrix-def)
    finally show (sum}M\mathrm{ mbot :: ('a,'b) square) (i,j)=mzero (i,j)
qed
lemma sum-plus-bot:
    fixes f :: ('a::enum,'b::aggregation-order) square
    shows sum}M\mp@code{f}\mp@subsup{\oplus}{M}{}\mathrm{ mbot = sum}\mp@subsup{M}{}{\prime}
proof (rule ext, rule prod-cases)
    let ?h = hd enum-class.enum
    fix ij
```



```
bot else zero + bot)
    by (simp add: plus-matrix-def bot-matrix-def sum-matrix-def)
    also have ... = (if i=?h}\wedgej=i then \mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}\mp@subsup{l}{}{\prime}f(k,l)\mathrm{ else zero)
    by (metis (no-types, lifting) add-add-bot aggregation.sum-0.F-one)
    also have ... = (sum}M|)(i,j
    by (simp add: sum-matrix-def)
    finally show (sum}M|\mp@subsup{\oplus}{M}{}\mathrm{ mbot) (i,j)=(sum}M|)(i,j
    by simp
qed
lemma sum-plus-zero:
    fixes f :: ('a::enum,'b::aggregation-order) square
    shows sum}MM f\mp@subsup{\oplus}{M}{}\mathrm{ mzero = sum
    by (rule ext, rule prod-cases) (simp add: plus-matrix-def zero-matrix-def
sum-matrix-def)
lemma agg-matrix-bot:
```

fixes $f::\left({ }^{\prime} a,{ }^{\prime} b::\right.$ aggregation-order) square
assumes $\forall i j . f(i, j)=$ bot
shows $f=$ mbot
apply (unfold bot-matrix-def)
using assms by auto
We consider a different implementation of matrix aggregation which stores the aggregated value in all entries of the matrix instead of a particular one. This does not require an enumeration of the indices. All results continue to hold using this alternative implementation.
definition sum-matrix-2 :: ('a,'b::\{plus,bot\}) square $\Rightarrow$ ('a,'b) square (sum2 ${ }_{M}$ [80] 80) where sum-matrix-2 $f=\left(\lambda e \cdot \sum_{k} \sum_{l} f(k, l)\right)$
lemma sum-bot-2:
sum2 $_{M}($ mbot :: ('a,'b::aggregation-order $)$ square $)=$ mzero
proof
fix $e$
have $\left(\right.$ sum2 $_{M}$ mbot $::\left({ }^{\prime} a,^{\prime} b\right)$ square $) e=\left(\sum_{\left(k::^{\prime} a\right)} \sum_{\left(l::^{\prime} a\right)}\right.$ bot)
by (unfold sum-matrix-2-def bot-matrix-def) simp
also have $\ldots=b o t+b o t$
using agg-sum-bot aggregation.sum-0.neutral by fastforce
also have $\ldots=$ mzero $e$
by (simp add: zero-matrix-def)
finally show $\left(\right.$ sum2 $_{M}$ mbot $::\left({ }^{\prime} a,{ }^{\prime} b\right)$ square $) e=m z e r o ~ e$
qed
lemma sum-plus-bot-2:
fixes $f::\left({ }^{\prime} a, ' b::\right.$ aggregation-order) square
shows sum2 $_{M} f \oplus_{M}$ mbot $=\operatorname{sum2}_{M} f$
proof
fix $e$
have $\left(\operatorname{sum2}_{M} f \oplus_{M}\right.$ mbot) $e=\left(\sum_{k} \sum_{l} f(k, l)\right)+$ bot
by (simp add: plus-matrix-def bot-matrix-def sum-matrix-2-def)
also have $\ldots=\left(\sum_{k} \sum_{l} f(k, l)\right)$
by (metis (no-types, lifting) add-add-bot aggregation.sum-0.F-one)
also have $\ldots=\left(\operatorname{sum2}_{M} f\right) e$
by (simp add: sum-matrix-2-def)
finally show $\left(\operatorname{sum}^{2}{ }_{M} f \oplus_{M}\right.$ mbot) $e=\left(\operatorname{sum2}_{M} f\right) e$ by $\operatorname{simp}$
qed
lemma sum-plus-zero-2:
fixes $f::\left({ }^{\prime} a\right.$, 'b::aggregation-order) square
shows sum2 ${ }_{M} f \oplus_{M}$ mzero $=\operatorname{sum2}_{M} f$
by (simp add: plus-matrix-def zero-matrix-def sum-matrix-2-def)

### 4.3 Aggregation Lattices

We extend aggregation orders to dense bounded distributive lattices. Axiom add-lattice implements the inclusion-exclusion principle at the level of edge weights.

```
class aggregation-lattice \(=\) bounded-distrib-lattice + dense-lattice +
aggregation-order +
    assumes add-lattice: \(x+y=(x \sqcup y)+(x \sqcap y)\)
```

Aggregation lattices form a Stone relation algebra by reusing the meet operation as composition, using identity as converse and a standard implementation of pseudocomplement.

```
class aggregation-algebra \(=\) aggregation-lattice + uminus + one + times + conv
\(+\)
    assumes uminus-def \([\) simp \(]:-x=\) (if \(x=\) bot then top else bot \()\)
    assumes one-def [simp]: \(1=\) top
    assumes times-def [simp]: \(x * y=x \sqcap y\)
    assumes conv-def [simp]: \(x^{T}=x\)
begin
subclass stone-algebra
    apply unfold-locales
    using bot-meet-irreducible bot-unique by auto
subclass stone-relation-algebra
    apply unfold-locales
    prefer 9 using bot-meet-irreducible apply auto[1]
    by (simp-all add: inf.assoc le-infI2 inf-sup-distrib1 inf-sup-distrib2 inf.commute
inf.left-commute)
end
```

We show that matrices over aggregation lattices form an s-algebra using the above operations.
interpretation agg-square-s-algebra: s-algebra where sup = sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::enum,'b::aggregation-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix proof
fix $f g h::\left({ }^{\prime} a,{ }^{\prime} b\right)$ square
show $f \neq$ mbot $\wedge \operatorname{sum}_{M} f \preceq \operatorname{sum}_{M} g \longrightarrow h \oplus_{M} \operatorname{sum}_{M} f \preceq h \oplus_{M} \operatorname{sum}_{M} g$
proof
let $? h=h d$ enum-class.enum
assume 1: $f \neq \operatorname{mbot} \wedge \operatorname{sum}_{M} f \preceq \operatorname{sum}_{M} g$
hence $\exists k l . f(k, l) \neq b o t$
by (meson agg-matrix-bot)
hence 2: $\left(\sum_{k} \sum_{l} f(k, l)\right) \neq$ bot
using agg-sum-not-bot by blast
have $\left(\sum_{k} \sum_{l} f(k, l)\right)=\left(s u m_{M} f\right)(? h, ? h)$
by (simp add: sum-matrix-def)
also have $\ldots \leq\left(\operatorname{sum}_{M} g\right)(? h, ? h)$
using 1 by (simp add: less-eq-matrix-def)
also have $\ldots=\left(\sum_{k} \sum_{l} g(k, l)\right)$
by (simp add: sum-matrix-def)
finally have $\left(\sum_{k} \sum_{l} f(k, l)\right) \leq\left(\sum_{k} \sum_{l} g(k, l)\right)$
by simp
hence 3: $\left(\sum_{k} \sum_{l} f(k, l)\right)+$ bot $\leq\left(\sum_{k} \sum_{l} g(k, l)\right)+$ bot
by (metis (no-types, lifting) add-add-bot aggregation.sum-0.F-one)
show $h \oplus_{M} \operatorname{sum}_{M} f \preceq h \oplus_{M} \operatorname{sum}_{M} g$
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases, unfold
plus-matrix-def)
fix $i j$
have $4: h(i, j)+\left(\sum_{k} \sum_{l} f(k, l)\right) \leq h(i, j)+\left(\sum_{k} \sum_{l} g(k, l)\right)$
using 23 by (metis (no-types, lifting) add-right-isotone add.commute)
have $h(i, j)+\left(\operatorname{sum}_{M} f\right)(i, j)=h(i, j)+\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l} f$ ( $k, l$ ) else zero)
by (simp add: sum-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $h(i, j)+\left(\sum_{k} \sum_{l} f(k, l)\right)$ else $h$
$(i, j)+$ zero $)$
by $\operatorname{simp}$
also have $\ldots \leq\left(\right.$ if $i=? h \wedge j=i$ then $h(i, j)+\left(\sum_{k} \sum_{l} g(k, l)\right)$ else $h$ $(i, j)+z e r o)$
using 4 order.eq-iff by auto
also have $\ldots=h(i, j)+\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l} g(k, l)$ else zero $)$
by $\operatorname{simp}$
finally show $h(i, j)+\left(\operatorname{sum}_{M} f\right)(i, j) \leq h(i, j)+\left(\operatorname{sum}_{M} g\right)(i, j)$
by (simp add: sum-matrix-def)
qed
qed
next
fix $f::\left({ }^{\prime} a,{ }^{\prime} b\right)$ square
show $\operatorname{sum}_{M} f \oplus_{M} \operatorname{sum}_{M}$ mbot $=\operatorname{sum}_{M} f$
by (simp add: sum-bot sum-plus-zero)
next
fix $f g::\left({ }^{\prime} a,{ }^{\prime} b\right)$ square
show $\operatorname{sum}_{M} f \oplus_{M} \operatorname{sum}_{M} g=\operatorname{sum}_{M}(f \oplus g) \oplus_{M} \operatorname{sum}_{M}(f \otimes g)$
proof (rule ext, rule prod-cases)
fix $i j$
let $? h=h d$ enum-class.enum
have $\left(\operatorname{sum}_{M} f \oplus_{M}\right.$ sum $\left._{M} g\right)(i, j)=\left(\right.$ sum $\left._{M} f\right)(i, j)+\left(\operatorname{sum}_{M} g\right)(i, j)$
by (simp add: plus-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l} f(k, l)$ else zero $)+($ if $i=? h$
$\wedge j=i$ then $\sum_{k} \sum_{l} g(k, l)$ else zero $)$
by (simp add: sum-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\left(\sum_{k} \sum_{l} f(k, l)\right)+\left(\sum_{k} \sum_{l} g(k, l)\right)$ else zero)
by $\operatorname{simp}$
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l} f(k, l)+g(k, l)$ else zero $)$
using agg-sum-distrib-2 by (metis (no-types))
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l}(f(k, l) \sqcup g(k, l))+(f(k, l)$ $\sqcap g(k, l))$ else zero)
using add-lattice aggregation.sum-0.cong by (metis (no-types, lifting))
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l}(f \oplus g)(k, l)+(f \otimes g)(k, l)$ else zero)
by (simp add: sup-matrix-def inf-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\left(\sum_{k} \sum_{l}(f \oplus g)(k, l)\right)+\left(\sum_{k} \sum_{l}(f\right.$ $\otimes g)(k, l))$ else zero $)$
using agg-sum-distrib-2 by (metis (no-types))
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l}(f \oplus g)(k, l)$ else zero $)+($ if $i$ $=? h \wedge j=i$ then $\sum_{k} \sum_{l}(f \otimes g)(k, l)$ else zero $)$
by simp
also have $\ldots=\left(\operatorname{sum}_{M}(f \oplus g)\right)(i, j)+\left(\operatorname{sum}_{M}(f \otimes g)\right)(i, j)$
by (simp add: sum-matrix-def)
also have $\ldots=\left(\operatorname{sum}_{M}(f \oplus g) \oplus_{M} \operatorname{sum}_{M}(f \otimes g)\right)(i, j)$
by (simp add: plus-matrix-def)
finally show $\left(\operatorname{sum}_{M} f \oplus_{M} \operatorname{sum}_{M} g\right)(i, j)=\left(\operatorname{sum}_{M}(f \oplus g) \oplus_{M} \operatorname{sum}_{M}(f \otimes\right.$ g)) $(i, j)$
qed
next
fix $f::\left({ }^{\prime} a,{ }^{\prime} b\right)$ square
show $\operatorname{sum}_{M}\left(f^{t}\right)=\operatorname{sum}_{M} f$
proof (rule ext, rule prod-cases)
fix $i j$
let $? h=h d$ enum-class.enum
have $\left(\operatorname{sum}_{M}\left(f^{t}\right)\right)(i, j)=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l}\left(f^{t}\right)(k, l)$ else zero $)$
by (simp add: sum-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l}(f(l, k))^{T}$ else zero $)$
by (simp add: conv-matrix-def)
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{k} \sum_{l} f(l, k)$ else zero $)$
by $\operatorname{simp}$
also have $\ldots=\left(\right.$ if $i=? h \wedge j=i$ then $\sum_{l} \sum_{k} f(l, k)$ else zero $)$
by (metis agg-sum-commute)
also have $\ldots=\left(\operatorname{sum}_{M} f\right)(i, j)$
by (simp add: sum-matrix-def)
finally show $\left(\operatorname{sum}_{M}\left(f^{t}\right)\right)(i, j)=\left(\operatorname{sum}_{M} f\right)(i, j)$
qed
qed
We show the same for the alternative implementation that stores the result of aggregation in all elements of the matrix.
interpretation agg-square-s-algebra-2: s-algebra where sup $=$ sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::finite,'b::aggregation-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix- 2

```
proof
    fix fgh :: ('a,'b) square
```



```
g
    proof
        assume 1: f}=\textrm{mbot}\wedge\mp@subsup{\operatorname{sum2}}{M}{}f\preceq\mp@subsup{\operatorname{sum2}}{M}{}
        hence }\existskl.f(k,l)\not=bo
            by (meson agg-matrix-bot)
    hence 2: (\sum k 峟f 
        using agg-sum-not-bot by blast
    obtain c:: ' }a\mathrm{ where True
        by simp
    have}(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{l}f(k,l))=(\mp@subsup{\operatorname{sum2}}{M}{}f)(c,c
        by (simp add: sum-matrix-2-def)
        also have .. \leq (sum2 M g) (c,c)
        using 1 by (simp add: less-eq-matrix-def)
    also have ... = (\sum k \sum 存g(k,l))
        by (simp add: sum-matrix-2-def)
    finally have (\sum}\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}f(k,l))\leq(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}\mp@subsup{|}{g}{}(k,l)
        by simp
    hence 3: (\sum k \mp@subsup{\sum}{l}{}f(k,l))+bot\leq(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}\mp@subsup{l}{g}{}(k,l))+bot
        by (metis (no-types, lifting) add-add-bot aggregation.sum-0.F-one)
    show h}\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}f\preceqh\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}
    proof (unfold less-eq-matrix-def, rule allI, unfold plus-matrix-def)
        fix e
        have he+(sum2}\mp@subsup{M}{M}{}f)e=he+(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}f(k,l)
            by (simp add: sum-matrix-2-def)
```



```
            using 2 3 by (metis (no-types, lifting) add-right-isotone add.commute)
        finally show he+(sum2 M}f)e\leqhe+(sum2 M g) 
            by (simp add: sum-matrix-2-def)
        qed
    qed
next
    fix f :: ('a,'b) square
    show sum2}\mp@subsup{M}{}{\prime}f\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}\mathrm{ mbot }=\mp@subsup{\operatorname{sum2}}{M}{}
        by (simp add: sum-bot-2 sum-plus-zero-2)
next
    fix fg :: ('a,'b) square
    show sum2}\mp@subsup{M}{}{\prime}f\mp@subsup{\oplus}{M}{}\mp@subsup{\mathrm{ sum2 }}{M}{}g=\mp@subsup{\operatorname{sum2}}{M}{}(f\oplusg)\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}(f\otimesg
    proof
        fix e
        have (sum2}\mp@subsup{M}{}{\prime}f\mp@subsup{\oplus}{M}{\prime}\mp@subsup{\operatorname{sum2}}{M}{}g)e=(\mp@subsup{\operatorname{sum2}}{M}{}f)e+(\mp@subsup{\operatorname{sum2}}{M}{}g)
        by (simp add: plus-matrix-def)
    also have ... = (\sum \mp@subsup{k}{k}{}\mp@subsup{\sum}{l}{}f}\mp@subsup{f}{(k,l))}{
        by (simp add: sum-matrix-2-def)
```



```
        using agg-sum-distrib-2 by (metis (no-types))
    also have ... = (\sum k \sum l (f (k,l) \sqcupg(k,l)) + (f (k,l) \sqcapg(k,l)))
```

```
        using add-lattice aggregation.sum-0.cong by (metis (no-types, lifting))
    also have ... = (\sum 庳汭 (f\oplusg)(k,l)+(f\otimesg)(k,l))
        by (simp add: sup-matrix-def inf-matrix-def)
    also have ... = (\sum \mp@subsup{k}{k}{}\mp@subsup{\sum}{l}{}(f\oplusg)(k,l))+(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{}(f\otimesg)(k,l))
        using agg-sum-distrib-2 by (metis (no-types))
    also have }\ldots=(\mp@subsup{\operatorname{sum2}}{M}{}(f\oplusg))e+(sum\mp@subsup{2}{M}{\prime}(f\otimesg))
        by (simp add: sum-matrix-2-def)
    also have ... = (sum2}\mp@subsup{M}{M}{}(f\oplusg)\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}(f\otimesg))
        by (simp add: plus-matrix-def)
    finally show (sum2}\mp@subsup{M}{M}{}f\mp@subsup{\oplus}{M}{M sum2}\mp@subsup{M}{M}{}g)e=(\mp@subsup{\operatorname{sum2}}{M}{}(f\oplusg)\mp@subsup{\oplus}{M}{}\mp@subsup{\operatorname{sum2}}{M}{}(
\otimesg))}
    qed
next
    fix f :: ('a,'b) square
    show sum2}\mp@subsup{M}{}{\prime}(\mp@subsup{f}{}{t})=\mp@subsup{\operatorname{sum2}}{M}{}
    proof
        fix }
        have (sum2}\mp@subsup{M}{M}{(ft))e=(\mp@subsup{\sum}{k}{}\mp@subsup{\sum}{l}{l}(\mp@subsup{f}{}{t})(k,l))
            by (simp add: sum-matrix-2-def)
        also have ... = (\sum \mp@subsup{k}{k}{}\mp@subsup{\sum}{l}{}(f(l,k)\mp@subsup{)}{}{T})
            by (simp add: conv-matrix-def)
```



```
        by simp
    also have ... = (\sum l \mp@subsup{\sum}{k}{}f(l,k))
            by (metis agg-sum-commute)
        also have ... = (sum2}\mp@subsup{M}{M}{}f)
            by (simp add: sum-matrix-2-def)
    finally show (sum2 }\mp@subsup{M}{}{\prime}(\mp@subsup{f}{}{t}))e=(sum\mp@subsup{2}{M}{}f)
    qed
qed
```


### 4.4 Matrix Minimisation

We construct an operation that finds the minimum entry of a matrix. Because a matrix can have several entries with the same minimum value, we introduce a lexicographic order on the indices to make the operation deterministic. The order is obtained by enumerating the universe of the index.
primrec enum-pos' $::$ ' $a$ list $\Rightarrow$ ' $a::$ enum $\Rightarrow$ nat where
enum-pos ${ }^{\prime}$ Nil $x=0$
|enum-pos' $(y \# y s) x=\left(\right.$ if $x=y$ then 0 else $1+$ enum-pos $\left.^{\prime} y s x\right)$
lemma enum-pos'-inverse:
List.member xs $x \Longrightarrow x s!($ enum-pos' $x s x)=x$
apply (induct $x s$ )
apply (simp add: member-rec(2))
by (metis diff-add-inverse enum-pos'. $\operatorname{simps}(2)$ less-one member-rec(1)
not-add-less1 nth-Cons')

The following function finds the position of an index in the enumerated universe.

```
fun enum-pos :: ' \(a::\) enum \(\Rightarrow\) nat where enum-pos \(x=\) enum-pos'
(enum-class.enum::'a list) \(x\)
lemma enum-pos-inverse [simp]:
    enum-class.enum!(enum-pos \(x)=x\)
    apply (unfold enum-pos.simps)
    apply (rule enum-pos'-inverse)
    by (metis in-enum List.member-def)
lemma enum-pos-injective [simp]:
    enum-pos \(x=\) enum-pos \(y \Longrightarrow x=y\)
    by (metis enum-pos-inverse)
```

The position in the enumerated universe determines the order.
abbreviation enum-pos-less-eq :: ' $a::$ enum $\Rightarrow$ ' $a \Rightarrow$ bool where enum-pos-less-eq $x y \equiv$ (enum-pos $x \leq$ enum-pos $y$ )
abbreviation enum-pos-less :: 'a::enum $\Rightarrow$ ' $a \Rightarrow$ bool where enum-pos-less $x y$ $\equiv($ enum-pos $x<$ enum-pos $y$ )
sublocale enum <enum-order: order where less-eq $=\lambda x y$.enum-pos-less-eq $x$ $y$ and less $=\lambda x y$. enum-pos $x<$ enum-pos $y$
apply unfold-locales
by auto
Based on this, a lexicographic order is defined on pairs, which represent locations in a matrix.
abbreviation enum-lex-less :: 'a::enum $\times{ }^{\prime} a \Rightarrow{ }^{\prime} a \times{ }^{\prime} a \Rightarrow$ bool where enum-lex-less $\equiv(\lambda(i, j)(k, l)$. enum-pos-less $i k \vee(i=k \wedge$ enum-pos-less $j l))$ abbreviation enum-lex-less-eq :: 'a::enum $\times{ }^{\prime} a \Rightarrow{ }^{\prime} a \times{ }^{\prime} a \Rightarrow$ bool where enum-lex-less-eq $\equiv(\lambda(i, j)(k, l)$. enum-pos-less $i k \vee(i=k \wedge$ enum-pos-less-eq $j$ l))

The $m$-operation determines the location of the non- $\perp$ minimum element which is first in the lexicographic order. The result is returned as a regular matrix with $\top$ at that location and $\perp$ everywhere else. In the weighted-graph model, this represents a single unweighted edge of the graph.

```
definition minarc-matrix :: ('a::enum,'b::\{bot,ord,plus,top\}) square \(\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)\)
square \(\left(\operatorname{minarc}_{M}-[80] 80\right)\) where minarc-matrix \(f=(\lambda e\). if \(f e \neq\) bot \(\wedge(\forall d\).
\((f d \neq\) bot \(\longrightarrow(f e+\) bot \(\leq f d+\) bot \(\wedge(\) enum-lex-less \(d e \longrightarrow f e+\) bot \(\neq f d+\)
bot)))) then top else bot)
lemma minarc-at-most-one:
    fixes \(f::\) ('a::enum, 'b::\{aggregation-order,top\}) square
    assumes \(\left(\operatorname{minarc}_{M} f\right) e \neq\) bot
        and \(\left(\operatorname{minarc}_{M} f\right) d \neq\) bot
    shows \(e=d\)
```

```
proof -
    have 1: \(f e+b o t \leq f d+b o t\)
    by (metis assms minarc-matrix-def)
    have \(f d+b o t \leq f e+b o t\)
        by (metis assms minarc-matrix-def)
    hence \(f e+b o t=f d+b o t\)
        using 1 by simp
    hence \(\neg\) enum-lex-less \(d e \wedge \neg\) enum-lex-less e \(d\)
        using assms by (unfold minarc-matrix-def) (metis (lifting))
    thus ?thesis
        using enum-pos-injective less-linear by auto
qed
```


### 4.5 Linear Aggregation Lattices

We now assume that the aggregation order is linear and forms a bounded lattice. It follows that these structures are aggregation lattices. A linear order on matrix entries is necessary to obtain a unique minimum entry.
class linear-aggregation-lattice $=$ linear-bounded-lattice + aggregation-order begin

```
subclass aggregation-lattice
    apply unfold-locales
    by (metis add-commute sup-inf-selective)
```

sublocale heyting: bounded-heyting-lattice where implies $=\lambda x y$. if $x \leq y$ then
top else $y$
apply unfold-locales
by (simp add: inf-less-eq)
end

Every non-empty set with a transitive total relation has a least element with respect to this relation.

```
lemma least-order:
    assumes transitive: \(\forall x y z\). le \(x y \wedge l e y z \longrightarrow l e x z\)
            and total: \(\forall x y\). le \(x y \vee\) le \(y x\)
        shows finite \(A \Longrightarrow A \neq\{ \} \Longrightarrow \exists x . x \in A \wedge(\forall y . y \in A \longrightarrow l e x y)\)
proof (induct A rule: finite-ne-induct)
    case singleton
    thus ?case
        using total by auto
next
    case insert
    thus ?case
        by (metis insert-iff transitive total)
qed
lemma minarc-at-least-one:
```

```
    fixes \(f::(' a::\) enum,'b::linear-aggregation-lattice) square
    assumes \(f \neq\) mbot
        shows \(\exists e \cdot\left(\operatorname{minarc}_{M} f\right) e=\) top
proof -
    let ?nbe \(=\{(e, f e) \mid e . f e \neq b o t\}\)
    have 1: finite ?nbe
        using finite-code finite-image-set by blast
    have 2: ? \(n b e \neq\{ \}\)
        using assms agg-matrix-bot by fastforce
    let ?le \(=\lambda\left(e::^{\prime} a \times\right.\) ' \(\left.a, f e:: ' b\right)\left(d::^{\prime} a \times\right.\) 'a,fd \() . f e+b o t \leq f d+b o t\)
    have 3: \(\forall x y z\). ?le \(x y \wedge\) ?le \(y z \longrightarrow\) ?le \(x z\)
        by auto
    have \(4: \forall x y\). ?le \(x y \vee\) ?le \(y x\)
    by (simp add: linear)
    have \(\exists x . x \in\) ? nbe \(\wedge(\forall y . y \in\) ? nbe \(\longrightarrow\) ?le \(x y)\)
    by (rule least-order, rule 3, rule 4, rule 1, rule 2)
    then obtain \(e f e\) where \(5:(e, f e) \in\) ?nbe \(\wedge(\forall y . y \in ? n b e \longrightarrow\) ?le \((e, f e) y)\)
    by auto
    let \(? m e=\{e . f e \neq b o t \wedge f e+b o t=f e+b o t\}\)
    have 6: finite ?me
        using finite-code finite-image-set by blast
    have 7: ? \(m e \neq\{ \}\)
        using 5 by auto
    have \(8: \forall x y z\). enum-lex-less-eq \(x\) y \(\wedge\) enum-lex-less-eq \(y z \longrightarrow\)
enum-lex-less-eq \(x\) z
    by auto
    have 9: \(\forall x y\). enum-lex-less-eq \(x y \vee\) enum-lex-less-eq \(y x\)
        by auto
    have \(\exists x . x \in\) ? \(m e \wedge(\forall y . y \in\) ? \(m e \longrightarrow\) enum-lex-less-eq \(x y)\)
    by (rule least-order, rule 8 , rule 9 , rule 6 , rule 7)
    then obtain \(m\) where 10: \(m \in\) ? \(m e \wedge(\forall y . y \in\) ? \(m e \longrightarrow\) enum-lex-less-eq \(m\)
y)
    by auto
    have 11: \(f m \neq b o t\)
        using 105 by auto
    have 12: \(\forall d . f d \neq\) bot \(\longrightarrow f m+\) bot \(\leq f d+\) bot
        using 105 by simp
    have \(\forall d . f d \neq\) bot \(\wedge\) enum-lex-less \(d m \longrightarrow f m+\) bot \(\neq f d+\) bot
        using 10 by fastforce
    hence \(\left(\operatorname{minarc}_{M} f\right) m=\) top
    using 1112 by (simp add: minarc-matrix-def)
    thus ?thesis
    by blast
qed
```

Linear aggregation lattices form a Stone relation algebra by reusing the meet operation as composition, using identity as converse and a standard implementation of pseudocomplement.
class linear-aggregation-algebra $=$ linear-aggregation-lattice + uminus + one +

```
times + conv +
    assumes uminus-def-2 [simp]: -x= (if x = bot then top else bot)
    assumes one-def-2 [simp]: 1 = top
    assumes times-def-2 [simp]: x*y=x\sqcapy
    assumes conv-def-2 [simp]: x }\mp@subsup{x}{}{T}=
begin
subclass aggregation-algebra
    apply unfold-locales
    using inf-dense by auto
lemma regular-bot-top-2:
    regular }x\longleftrightarrowx=\mathrm{ bot }\veex=\mathrm{ top
    by simp
sublocale heyting: heyting-stone-algebra where implies = \lambdax y. if x}\leqy\mathrm{ then
top else y
    apply unfold-locales
    apply (simp add: order.antisym)
    by auto
end
```

We show that matrices over linear aggregation lattices form an m-algebra using the above operations.
interpretation agg-square-m-algebra: m-algebra where sup $=$ sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::enum,'b::linear-aggregation-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix and minarc $=$ minarc-matrix
proof
fix $f::\left({ }^{\prime} a, ' b\right)$ square
show $\operatorname{minarc}_{M} f \preceq \ominus \ominus f$
proof (unfold less-eq-matrix-def, rule allI)
fix $e::{ }^{\prime} a \times{ }^{\prime} a$
have $\left(\operatorname{minarc}_{M} f\right) e \leq($ if $f e \neq b$ bot then top else $--(f e))$
by (simp add: minarc-matrix-def)
also have $\ldots=--(f e)$
by $\operatorname{simp}$
also have $\ldots=(\ominus \ominus f) e$
by (simp add: uminus-matrix-def)
finally show $\left(\operatorname{minarc}_{M} f\right) e \leq(\ominus \ominus f) e$
qed
next
fix $f::\left({ }^{\prime} a, ' b\right)$ square
let $? a t=$ bounded-distrib-allegory-signature.arc mone times-matrix
less-eq-matrix mtop conv-matrix

$$
\text { show } f \neq \text { mbot } \longrightarrow \text { ?at }\left(\operatorname{minarc}_{M} f\right)
$$

## proof

assume $1: f \neq$ mbot
have $\operatorname{minarc}_{M} f \odot$ mtop $\odot\left(\operatorname{minarc}_{M} f \odot m t o p\right)^{t}=\operatorname{minarc}_{M} f \odot$ mtop $\odot$ $\left(\text { minarc }_{M} f\right)^{t}$
by (metis matrix-bounded-idempotent-semiring.surjective-top-closed matrix-monoid.mult-assoc matrix-stone-relation-algebra.conv-dist-comp matrix-stone-relation-algebra.conv-top)
also have ... $\preceq$ mone
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
fix $i j$
have $\left(\operatorname{minarc}_{M} f \odot\right.$ mtop $\left.\odot\left(\operatorname{minarc}_{M} f\right)^{t}\right)(i, j)=\left(\bigsqcup_{l}\left(\bigsqcup_{k}\left(\operatorname{minarc}_{M} f\right)\right.\right.$ $(i, k) *$ mtop $\left.(k, l)) *\left(\left(\operatorname{minarc}_{M} f\right)^{t}\right)(l, j)\right)$
by (simp add: times-matrix-def)
also have $\ldots=\left(\bigsqcup_{l}\left(\bigsqcup_{k}\left(\operatorname{minarc}_{M} f\right)(i, k) *\right.\right.$ top $\left.) *\left(\left(\operatorname{minarc}_{M} f\right)(j, l)\right)^{T}\right)$ by (simp add: top-matrix-def conv-matrix-def)
also have $\ldots=\left(\bigsqcup_{l} \bigsqcup_{k}\left(\operatorname{minarc}_{M} f\right)(i, k) *\right.$ top $\left.*\left(\left(\operatorname{minarc}_{M} f\right)(j, l)\right)^{T}\right)$ by (metis comp-right-dist-sum)
also have $\ldots=\left(\bigsqcup_{l} \bigsqcup_{k}\right.$ if $i=j \wedge l=k$ then $\left(\operatorname{minarc}_{M} f\right)(i, k) *$ top * $\left(\left(\operatorname{minarc}_{M} f\right)(j, l)\right)^{T}$ else bot)
apply (rule sup-monoid.sum.cong)
apply $\operatorname{simp}$
by (metis (no-types, lifting) comp-left-zero comp-right-zero conv-bot prod.inject minarc-at-most-one)
also have $\ldots=\left(\right.$ if $i=j$ then $\left(\bigsqcup_{l} \bigsqcup_{k}\right.$ if $l=k$ then $\left(\operatorname{minarc}_{M} f\right)(i, k) *$ top * $\left(\left(\operatorname{minarc}_{M} f\right)(j, l)\right)^{T}$ else bot) else bot $)$ by auto
also have $\ldots \leq($ if $i=j$ then top else bot $)$ by $\operatorname{simp}$ also have $\ldots=$ mone $(i, j)$ by (simp add: one-matrix-def) finally show $\left(\operatorname{minarc}_{M} f \odot\right.$ mtop $\left.\odot\left(\operatorname{minarc}_{M} f\right)^{t}\right)(i, j) \leq$ mone $(i, j)$
qed
finally have 2: $\operatorname{minarc}_{M} f \odot m t o p \odot\left(\text { minarc }_{M} f \odot m t o p\right)^{t} \preceq$ mone
have 3: mtop $\odot\left(\right.$ minarc $_{M} f \odot$ mtop $)=$ mtop
proof (rule ext, rule prod-cases)
fix $i j$
from minarc-at-least-one obtain ei ej where $\left(\operatorname{minarc}_{M} f\right)(e i, e j)=t o p$ using 1 by force
hence 4: top $*$ top $\leq\left(\bigsqcup_{l}\left(\operatorname{minarc}_{M} f\right)(e i, l) *\right.$ top $)$ by (metis comp-inf.ub-sum)
have top $*\left(\bigsqcup_{l}\left(\operatorname{minarc}_{M} f\right)(e i, l) * t o p\right) \leq\left(\bigsqcup_{k}\right.$ top $*\left(\bigsqcup_{l}\left(\operatorname{minarc}_{M} f\right)\right.$
$(k, l) * t o p))$ by (rule comp-inf.ub-sum)
hence top $\leq\left(\bigsqcup_{k}\right.$ top $*\left(\bigsqcup_{l}\left(\operatorname{minarc}_{M} f\right)(k, l) *\right.$ top $\left.)\right)$ using 4 by auto
also have $\ldots=\left(\bigsqcup_{k} \operatorname{mtop}(i, k) *\left(\bigsqcup_{l}\left(\operatorname{minarc}_{M} f\right)(k, l) * \operatorname{mtop}(l, j)\right)\right)$

```
        by (simp add: top-matrix-def)
    also have ... =(mtop \odot (minarc}M|\odotmtop))(i,j
        by (simp add: times-matrix-def)
    finally show (mtop \odot (minarc}\mp@subsup{M}{M}{}f\odotmtop))(i,j)=mtop (i,j
    by (simp add: eq-iff top-matrix-def)
    qed
```



```
mtop \odot (minarc}Mf
    by (metis matrix-stone-relation-algebra.comp-associative
matrix-stone-relation-algebra.conv-dist-comp
matrix-stone-relation-algebra.conv-involutive
matrix-stone-relation-algebra.conv-top
matrix-bounded-idempotent-semiring.surjective-top-closed)
    also have ... \preceq mone
    proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
    fix ij
```



```
(i,k)*mtop (k,l)) * (minarc}\mp@subsup{M}{M}{\prime})(l,j)
    by (simp add: times-matrix-def)
    also have ... =( \ l ( \ ل k (( }\mp@subsup{\operatorname{minarc}}{M}{}f)(k,i)\mp@subsup{)}{}{T}*top)*(\mp@subsup{\operatorname{minarc}}{M}{}f)(l,j)
        by (simp add: top-matrix-def conv-matrix-def)
    also have ... =(\\\ \ \ (( }\mp@subsup{\operatorname{minarc}}{M}{}f)(k,i)\mp@subsup{)}{}{T}*\operatorname{top}*(\mp@subsup{\operatorname{minarc}}{M}{}f)(l,j)
        by (metis comp-right-dist-sum)
    also have ... = (\l \ \k if i=j^l=k then ((\mp@subsup{\operatorname{minarc}}{M}{}f)(k,i))}\mp@subsup{)}{}{T}*\mathrm{ top *
(minarc⿱}|\mp@code{f) (l,j) else bot)
    apply (rule sup-monoid.sum.cong)
    apply simp
    by (metis (no-types, lifting) comp-left-zero comp-right-zero conv-bot
prod.inject minarc-at-most-one)
    also have ... = (if i i=j then ( }\mp@subsup{\bigsqcup}{l}{}\mp@subsup{\bigsqcup}{k}{}\mathrm{ if }l=k\mathrm{ then (( (minarc}M\mp@code{f})(k,i)\mp@subsup{)}{}{T}
top * (minarc}\mp@subsup{M}{M}{f)(l,j) else bot) else bot)
            by auto
            also have ... \leq(if i=j then top else bot)
            by simp
            also have ... = mone (i,j)
            by (simp add: one-matrix-def)
            finally show ((\mp@subsup{\operatorname{minarc}}{M}{}f\mp@subsup{)}{}{t}\odot mtop \odot ( (\mp@subsup{\operatorname{marc}}{M}{}f))}(i,j)\leqmone (i,j
    qed
    finally have 5:(minarc}\mp@subsup{M}{M}{}f\mp@subsup{)}{}{t}\odot\mathrm{ mtop }\odot ((\mp@subsup{\operatorname{minarc}}{M}{}f\mp@subsup{)}{}{t}\odotmtop)t \preceq mon
    have mtop \odot ((minarc}M|)t\odotmtop)=mto
        using 3 by (metis matrix-monoid.mult-assoc
matrix-stone-relation-algebra.conv-dist-comp
matrix-stone-relation-algebra.conv-top)
    thus ?at ( minarc}\mp@subsup{M}{M}{}f
        using 2 3 5 by blast
    qed
next
```

fix $f g::\left({ }^{\prime} a, ' b\right)$ square
let ?at $=$ bounded-distrib-allegory-signature.arc mone times-matrix less-eq-matrix mtop conv-matrix

$$
\text { show ?at } g \wedge g \otimes f \neq \operatorname{mbot} \longrightarrow \operatorname{sum}_{M}\left(\operatorname{minarc}_{M} f \otimes f\right) \preceq \operatorname{sum}_{M}(g \otimes f)
$$

proof
assume 1: ? at $g \wedge g \otimes f \neq$ mbot
hence 2: $g=\ominus \ominus g$
using matrix-stone-relation-algebra.arc-regular by blast
show $\operatorname{sum}_{M}\left(\operatorname{minarc}_{M} f \otimes f\right) \preceq \operatorname{sum}_{M}(g \otimes f)$
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
fix $i j$
from minarc-at-least-one obtain ei ej where 3: $\left(\operatorname{minarc}_{M} f\right)(e i, e j)=t o p$ using 1 by force
hence 4: $\forall k l . \neg(k=e i \wedge l=e j) \longrightarrow\left(\operatorname{minarc}_{M} f\right)(k, l)=b o t$ by (metis (mono-tags, opaque-lifting) bot.extremum inf.bot-unique prod.inject minarc-at-most-one)
from agg-matrix-bot obtain $d i d j$ where $5:(g \otimes f)(d i, d j) \neq b o t$ using 1 by force
hence $6: g(d i, d j) \neq$ bot
by (metis inf-bot-left inf-matrix-def)
hence 7: $g(d i, d j)=t o p$ using 2 by (metis uminus-matrix-def uminus-def)
hence $8:(g \otimes f)(d i, d j)=f(d i, d j)$ by (metis inf-matrix-def inf-top.left-neutral)
have $9: \forall k l . k \neq d i \longrightarrow g(k, l)=b o t$ proof (intro allI, rule impI)
fix $k l$
assume 10: $k \neq d i$
have top * $(g(k, l))^{T}=g(d i, d j) * t o p *\left(g^{t}\right)(l, k)$
using 7 by (simp add: conv-matrix-def)
also have $\ldots \leq\left(\bigsqcup_{n} g(d i, n) * t o p\right) *\left(g^{t}\right)(l, k)$
by (metis comp-inf.ub-sum comp-right-dist-sum)
also have $\ldots \leq\left(\bigsqcup_{m}\left(\bigsqcup_{n} g(d i, n) * t o p\right) *\left(g^{t}\right)(m, k)\right)$
by (metis comp-inf.ub-sum)
also have $\ldots=\left(g \odot\right.$ mtop $\left.\odot g^{t}\right)(d i, k)$
by (simp add: times-matrix-def top-matrix-def)
also have $\ldots \leq$ mone ( $d i, k$ )
using 1 by (metis matrix-stone-relation-algebra.arc-expanded
less-eq-matrix-def)
also have...$=$ bot
apply (unfold one-matrix-def)
using 10 by auto
finally have $g(k, l) \neq t o p$
using 5 by (metis bot.extremum conv-def inf.bot-unique mult.left-neutral
one-def)
thus $g(k, l)=b o t$
using 2 by (metis uminus-def uminus-matrix-def)
qed
have $\forall k l . l \neq d j \longrightarrow g(k, l)=b o t$
proof (intro allI, rule impI)
fix $k l$
assume $11: l \neq d j$
have $(g(k, l))^{T} *$ top $=\left(g^{t}\right)(l, k) *$ top $* g(d i, d j)$
using 7 by (simp add: comp-associative conv-matrix-def)
also have $\ldots \leq\left(\bigsqcup_{n}\left(g^{t}\right)(l, n) * t o p\right) * g(d i, d j)$
by (metis comp-inf.ub-sum comp-right-dist-sum)
also have $\ldots \leq\left(\bigsqcup_{m}\left(\bigsqcup_{n}\left(g^{t}\right)(l, n) * t o p\right) * g(m, d j)\right)$
by (metis comp-inf.ub-sum)
also have $\ldots=\left(g^{t} \odot m t o p \odot g\right)(l, d j)$
by (simp add: times-matrix-def top-matrix-def)
also have $\ldots \leq$ mone $(l, d j)$
using 1 by (metis matrix-stone-relation-algebra.arc-expanded
less-eq-matrix-def)
also have ... $=$ bot
apply (unfold one-matrix-def)
using 11 by auto
finally have $g(k, l) \neq$ top
using 5 by (metis bot.extremum comp-right-one conv-def one-def top.extremum-unique)
thus $g(k, l)=b o t$
using 2 by (metis uminus-def uminus-matrix-def)
qed
hence 12: $\forall k l . \neg(k=d i \wedge l=d j) \longrightarrow(g \otimes f)(k, l)=b o t$
using 9 by (metis inf-bot-left inf-matrix-def)
have $\left(\sum_{k} \sum_{l}\left(\operatorname{minarc}_{M} f \otimes f\right)(k, l)\right)=\left(\sum_{k} \sum_{l}\right.$ if $k=e i \wedge l=e j$ then $\left(\operatorname{minarc}_{M} f \otimes f\right)(k, l)$ else $\left(\right.$ minarc $\left.\left._{M} f \otimes f\right)(k, l)\right)$ by $\operatorname{simp}$
also have $\ldots=\left(\sum_{k} \sum_{l}\right.$ if $k=e i \wedge l=e j$ then $\left(\operatorname{minarc}_{M} f \otimes f\right)(k, l)$ else $\left.\left(\operatorname{minarc}_{M} f\right)(k, l) \sqcap f(k, l)\right)$
by (unfold inf-matrix-def) simp
also have $\ldots=\left(\sum_{k} \sum_{l}\right.$ if $k=e i \wedge l=e j$ then $\left(\operatorname{minarc}_{M} f \otimes f\right)(k, l)$ else
$b o t)$
apply (rule aggregation.sum-0.cong)
apply $\operatorname{simp}$
using 4 by (metis inf-bot-left)
also have $\ldots=\left(\operatorname{minarc}_{M} f \otimes f\right)(e i, e j)+b o t$
by (unfold agg-delta-2) simp
also have $\ldots=f(e i, e j)+b o t$
using 3 by (simp add: inf-matrix-def)
also have $\ldots \leq(g \otimes f)(d i, d j)+b o t$
using 35678 by (metis minarc-matrix-def)
also have $\ldots=\left(\sum_{k} \sum_{l}\right.$ if $k=d i \wedge l=d j$ then $(g \otimes f)(k, l)$ else bot $)$
by (unfold agg-delta-2) simp
also have $\ldots=\left(\sum_{k} \sum_{l}\right.$ if $k=d i \wedge l=d j$ then $(g \otimes f)(k, l)$ else $(g \otimes f)$
$(k, l))$
apply (rule aggregation.sum-0.cong)
apply simp
using 12 by metis


```
            by simp
        finally show (sum}M(\mp@subsup{\operatorname{minarc}}{M}{}f\otimesf))(i,j)\leq(sum, (g (g\otimesf))(i,j
            by (simp add: sum-matrix-def)
    qed
    qed
next
    fix fg :: ('a,'b) square
    let ?h = hd enum-class.enum
    show sum}\mp@code{M f\preceq sum}\mp@subsup{M}{M}{}g\vee\mp@subsup{\operatorname{sum}}{M}{}g\preceq\mp@subsup{\operatorname{sum}}{M}{}
    proof (cases (sum}\mp@subsup{M}{M}{\prime})(?h,?h)\leq(sum\mp@subsup{M}{M}{}g)(?h,?h)
        case 1:True
        have sum}M|{\mp@code{sum}\mp@subsup{M}{M}{
            apply (unfold less-eq-matrix-def, rule allI, rule prod-cases)
            using 1 by (unfold sum-matrix-def) auto
            thus ?thesis
            by simp
    next
        case False
        hence 2: (sum}Mg)(?h,?h)\leq(sum M f) (?h,?h
            by (simp add: linear)
    have sum}M\ g\preceq sum M f
            apply (unfold less-eq-matrix-def, rule allI, rule prod-cases)
            using 2 by (unfold sum-matrix-def) auto
    thus ?thesis
        by simp
    qed
next
    have finite {f ::('a,'b) square. ( }\forall\mathrm{ e . regular (f e)) }
    by (unfold regular-bot-top-2, rule finite-set-of-finite-funs-pred) auto
    thus finite {f ::('a,'b) square . matrix-p-algebra.regular f }
        by (unfold uminus-matrix-def) meson
qed
```

We show the same for the alternative implementation that stores the result of aggregation in all elements of the matrix.
interpretation agg-square-m-algebra-2: m-algebra where sup $=$ sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::enum,'b::linear-aggregation-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix-2 and minarc $=$ minarc-matrix
proof
fix $f$ :: ( ${ }^{\prime} a$, ' $b$ ) square
show $\operatorname{minarc}_{M} f \preceq \ominus \ominus f$
by (simp add: agg-square-m-algebra.minarc-below)
next
fix $f::\left({ }^{\prime} a, ' b\right)$ square
let ?at $=$ bounded-distrib-allegory-signature.arc mone times-matrix

```
less-eq-matrix mtop conv-matrix
    show }f\not==\mathrm{ mbot }\longrightarrow\mathrm{ ?at ( minarc}M|
        by (simp add: agg-square-m-algebra.minarc-arc)
next
    fix fg :: ('a,'b) square
    let ?at = bounded-distrib-allegory-signature.arc mone times-matrix
less-eq-matrix mtop conv-matrix
    show ?at g ^g\otimesf\not= mbot \longrightarrow \mp@subsup{\operatorname{sum2}}{M}{}(\mp@subsup{\operatorname{minarc}}{M}{}f\otimesf)\preceq\mp@subsup{\operatorname{sum2}}{M}{}(g\otimesf)
    proof
        let ?h = hd enum-class.enum
        assume ?at g}\wedgeg\otimesf\not=mbo
        hence sum}M(\mp@subsup{\operatorname{minarc}}{M}{}f\otimesf)\preceq\mp@subsup{\operatorname{sum}}{M}{}(g\otimesf
            by (simp add: agg-square-m-algebra.minarc-min)
    hence (sum}M=(\mp@subsup{\operatorname{minarc}}{M}{}f\otimesf))(?h,?h)\leq(sum M (g\otimesf))(?h,?h
                by (simp add: less-eq-matrix-def)
    thus sum2}\mp@subsup{M}{M}{(minarc}\mp@subsup{M}{M}{}f\otimesf)\preceq\mp@subsup{\operatorname{sum2}}{M}{}(g\otimesf
            by (simp add: sum-matrix-def sum-matrix-2-def less-eq-matrix-def)
        qed
next
    fix fg :: ('a,'b) square
    let ?h = hd enum-class.enum
    have sum}M|{\mp@subsup{\operatorname{sum}}{M}{}g\vee\mp@subsup{\operatorname{sum}}{M}{}g\preceq\mp@subsup{\operatorname{sum}}{M}{}
        by (simp add: agg-square-m-algebra.sum-linear)
    hence }(\mp@subsup{\operatorname{sum}}{M}{}f)(?h,?h)\leq(\mp@subsup{\operatorname{sum}}{M}{}g)(?h,?h)\vee(\mp@subsup{\operatorname{sum}}{M}{}g)(?h,?h)\leq(\mp@subsup{sum}{M}{
f) (?h,?h)
    using less-eq-matrix-def by auto
    thus sum2}\mp@subsup{M}{}{\prime}f\preceq\mp@subsup{\mathrm{ sum2}}{M}{}g\vee\mp@subsup{\operatorname{sum2}}{M}{}g\preceq\mp@subsup{\operatorname{sum2}}{M}{}
        by (simp add: sum-matrix-def sum-matrix-2-def less-eq-matrix-def)
next
    show finite {f :: ('a,'b) square . matrix-p-algebra.regular f }
        by (simp add: agg-square-m-algebra.finite-regular)
qed
```

By defining the Kleene star as $\top$ aggregation lattices form a Kleene algebra.
class aggregation-kleene-algebra $=$ aggregation-algebra + star +
assumes star-def $[\operatorname{simp}]: x^{\star}=$ top
begin

```
subclass stone-kleene-relation-algebra
    apply unfold-locales
    by (simp-all add: inf.assoc le-infI2 inf-sup-distrib1 inf-sup-distrib2)
```

end
class linear-aggregation-kleene-algebra $=$ linear-aggregation-algebra + star +
assumes star-def-2 [simp]: $x^{\star}=$ top
begin
subclass aggregation-kleene-algebra
apply unfold-locales
by $\operatorname{simp}$
end
interpretation agg-square-m-kleene-algebra: $m$-kleene-algebra where sup $=$ sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::enum,'b::linear-aggregation-kleene-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and star $=$ star-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix and minarc $=$ minarc-matrix ..
interpretation agg-square-m-kleene-algebra-2: m-kleene-algebra where sup $=$ sup-matrix and inf $=$ inf-matrix and less-eq $=$ less-eq-matrix and less $=$ less-matrix and bot $=$ bot-matrix::('a::enum,'b::linear-aggregation-kleene-algebra) square and top $=$ top-matrix and uminus $=$ uminus-matrix and one $=$ one-matrix and times $=$ times-matrix and conv $=$ conv-matrix and star $=$ star-matrix and plus $=$ plus-matrix and sum $=$ sum-matrix- 2 and minarc $=$ minarc-matrix ..
class linorder-stone-relation-algebra-plus-expansion $=$ linorder-stone-relation-algebra-expansion + plus + assumes plus-def: plus $=$ sup

## begin

subclass linear-aggregation-algebra
apply unfold-locales
using plus-def sup-monoid.add-assoc apply blast
using plus-def sup-monoid.add-commute apply blast
using comp-inf.semiring.add-mono plus-def apply auto[1]
using plus-def apply force
using bot-eq-sup-iff plus-def apply blast
apply simp
apply simp
using times-inf apply presburger
by $\operatorname{simp}$
end
class linorder-stone-kleene-relation-algebra-plus-expansion $=$
linorder-stone-kleene-relation-algebra-expansion +
linorder-stone-relation-algebra-plus-expansion
begin
subclass linear-aggregation-kleene-algebra
apply unfold-locales
by $\operatorname{simp}$
end
class linorder-stone-kleene-relation-algebra-tarski-consistent-plus-expansion $=$
linorder-stone-kleene-relation-algebra-tarski-consistent-expansion +
linorder-stone-kleene-relation-algebra-plus-expansion
end

## 5 Algebras for Aggregation and Minimisation with a Linear Order

This theory gives several classes of instances of linear aggregation lattices as described in [4]. Each of these instances can be used as edge weights and the resulting graphs will form s-algebras and $m$-algebras as shown in a separate theory.
theory Linear-Aggregation-Algebras
imports Matrix-Aggregation-Algebras HOL.Real
begin
no-notation
inf (infixl $\sqcap 70$ )
and uminus (- - [81] 80)

### 5.1 Linearly Ordered Commutative Semigroups

Any linearly ordered commutative semigroup extended by new least and greatest elements forms a linear aggregation lattice. The extension is done so that the new least element is a unit of aggregation and the new greatest element is a zero of aggregation.

```
datatype ' \(a\) ext \(=\)
    Bot
    | Val 'a
    | Top
instantiation ext :: (linordered-ab-semigroup-add)
linear-aggregation-kleene-algebra
begin
fun plus-ext :: ' \(a\) ext \(\Rightarrow\) ' \(a\) ext \(\Rightarrow\) 'a ext where
    plus-ext Bot \(x=x\)
| plus-ext \((\) Val \(x)\) Bot \(=\) Val \(x\)
\(\mid\) plus-ext \((\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(x+y)\)
| plus-ext (Val -) Top \(=\) Top
| plus-ext Top - = Top
```

```
fun sup-ext :: ' \(a\) ext \(\Rightarrow\) 'a ext \(\Rightarrow{ }^{\prime} a\) ext where
    sup-ext Bot \(x=x\)
| sup-ext (Val x) Bot \(=\) Val \(x\)
\(\mid \sup -\operatorname{ext}(\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(\max x y)\)
| sup-ext (Val -) Top = Top
| sup-ext Top - = Top
fun inf-ext :: 'a ext \(\Rightarrow{ }^{\prime} a\) ext \(\Rightarrow\) 'a ext where
    inf-ext Bot - = Bot
| inf-ext (Val -) Bot = Bot
\(\mid \inf -\operatorname{ext}(\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(\min x y)\)
inf-ext (Val \(x)\) Top \(=\) Val \(x\)
inf-ext Top \(x=x\)
fun times-ext :: 'a ext \(\Rightarrow{ }^{\prime} a\) ext \(\Rightarrow{ }^{\prime} a\) ext where times-ext \(x y=x \sqcap y\)
fun uminus-ext :: 'a ext \(\Rightarrow\) ' \(a\) ext where
    uminus-ext Bot \(=\) Top
| uminus-ext (Val -) = Bot
| uminus-ext Top \(=\) Bot
fun star-ext :: 'a ext \(\Rightarrow\) 'a ext where star-ext \(-=\) Top
fun conv-ext :: ' \(a\) ext \(\Rightarrow\) 'a ext where conv-ext \(x=x\)
definition bot-ext :: 'a ext where bot-ext \(\equiv\) Bot
definition one-ext :: ' \(a\) ext where one-ext \(\equiv\) Top
definition top-ext :: 'a ext where top-ext \(\equiv\) Top
fun less-eq-ext :: 'a ext \(\Rightarrow\) 'a ext \(\Rightarrow\) bool where
    less-eq-ext Bot \(-=\) True
| less-eq-ext (Val -) Bot = False
|less-eq-ext \((\) Val \(x)(\) Val \(y)=(x \leq y)\)
less-eq-ext (Val -) Top = True
| less-eq-ext Top Bot = False
| less-eq-ext Top (Val -) = False
less-eq-ext Top Top = True
```

fun less-ext :: 'a ext $\Rightarrow{ }^{\prime}$ 'a ext $\Rightarrow$ bool where less-ext $x y=(x \leq y \wedge \neg y \leq x)$
instance
proof
fix $x y z::$ 'a ext
show $(x+y)+z=x+(y+z)$
by (cases $x$; cases $y$; cases $z$ ) (simp-all add: add.assoc)
show $x+y=y+x$
by (cases $x$; cases $y$ ) (simp-all add: add.commute)
show $(x<y)=(x \leq y \wedge \neg y \leq x)$

```
    by simp
show }x\leq
    using less-eq-ext.elims(3) by fastforce
show }x\leqy\Longrightarrowy\leqz\Longrightarrowx\leq
    by (cases x; cases y; cases z) simp-all
show }x\leqy\Longrightarrowy\leqx\Longrightarrowx=
    by (cases x; cases y) simp-all
show }x\sqcapy\leq
    by (cases x; cases y) simp-all
show }x\sqcapy\leq
    by (cases x; cases y) simp-all
show }x\leqy\Longrightarrowx\leqz\Longrightarrowx\leqy\sqcap
    by (cases x; cases y; cases z) simp-all
show }x\leqx\sqcup
    by (cases x; cases y) simp-all
show }y\leqx\sqcup
    by (cases x; cases y) simp-all
show }y\leqx\Longrightarrowz\leqx\Longrightarrowy\sqcupz\leq
    by (cases }x\mathrm{ ; cases y; cases z) simp-all
show bot }\leq
    by (simp add: bot-ext-def)
show }x\leqto
    by (cases x) (simp-all add: top-ext-def)
show }x\not=\mathrm{ bot }\wedgex+\mathrm{ bot }\leqy+\mathrm{ bot }\longrightarrowx+z\leqy+
    by (cases x; cases y; cases z) (simp-all add: bot-ext-def add-right-mono)
show }x+y+bot=x+
    by (cases x; cases y) (simp-all add: bot-ext-def)
show }x+y=bot\longrightarrowx=bo
    by (cases x; cases y) (simp-all add: bot-ext-def)
show }x\leqy\veey\leq
    by (cases x; cases y) (simp-all add: linear)
show -x = (if }x=\mathrm{ bot then top else bot)
    by (cases x) (simp-all add: bot-ext-def top-ext-def)
show (1::'a ext) = top
    by (simp add: one-ext-def top-ext-def)
show }x*y=x\sqcap
    by simp
show }\mp@subsup{x}{}{T}=
    by simp
show }\mp@subsup{x}{}{\star}= to
    by (simp add: top-ext-def)
qed
end
```

An example of a linearly ordered commutative semigroup is the set of real numbers with standard addition as aggregation.

```
lemma example-real-ext-matrix:
fixes x :: ('a::enum,real ext) square
```

```
shows minarc}M\\\`\ominus\ominus
by (rule agg-square-m-algebra.minarc-below)
```

Another example of a linearly ordered commutative semigroup is the set of real numbers with maximum as aggregation.

```
datatype real-max = Rmax real
```

instantiation real-max :: linordered-ab-semigroup-add
begin
fun less-eq-real-max where less-eq-real-max $(\operatorname{Rmax} x)(R \max y)=(x \leq y)$
fun less-real-max where less-real-max $(\operatorname{Rmax} x)(R \max y)=(x<y)$
fun plus-real-max where plus-real-max $(\operatorname{Rmax} x)(R \max y)=R \max (\max x y)$

## instance

## proof

    fix \(x\) y \(z\) :: real-max
    show \((x+y)+z=x+(y+z)\)
        by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x+y=y+x\)
    by (cases \(x\); cases \(y\) ) simp
    show \((x<y)=(x \leq y \wedge \neg y \leq x)\)
    by (cases \(x\); cases \(y\) ) auto
    show \(x \leq x\)
    by (cases \(x\) ) simp
    show \(x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z\)
    by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x \leq y \Longrightarrow y \leq x \Longrightarrow x=y\)
    by (cases \(x\); cases \(y\) ) simp
    show \(x \leq y \Longrightarrow z+x \leq z+y\)
    by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x \leq y \vee y \leq x\)
    by (cases \(x\); cases \(y\) ) auto
    qed
end
lemma example-real-max-ext-matrix:
fixes $x$ :: ('a::enum,real-max ext) square
shows $\operatorname{minarc}_{M} x \preceq \ominus \ominus x$
by (rule agg-square-m-algebra.minarc-below)

A third example of a linearly ordered commutative semigroup is the set of real numbers with minimum as aggregation.
datatype real-min $=$ Rmin real
instantiation real-min :: linordered-ab-semigroup-add
begin

```
fun less-eq-real-min where less-eq-real-min \((\operatorname{Rmin} x)(R \min y)=(x \leq y)\)
fun less-real-min where less-real-min \((\operatorname{Rmin} x)(\operatorname{Rmin} y)=(x<y)\)
fun plus-real-min where plus-real-min \((\operatorname{Rmin} x)(\operatorname{Rmin} y)=\operatorname{Rmin}(\min x y)\)
instance
proof
    fix \(x y z\) :: real-min
    show \((x+y)+z=x+(y+z)\)
    by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x+y=y+x\)
    by (cases \(x\); cases \(y\) ) simp
    show \((x<y)=(x \leq y \wedge \neg y \leq x)\)
    by (cases \(x\); cases \(y\) ) auto
    show \(x \leq x\)
    by (cases \(x\) ) simp
    show \(x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z\)
    by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x \leq y \Longrightarrow y \leq x \Longrightarrow x=y\)
    by (cases \(x\); cases \(y\) ) simp
    show \(x \leq y \Longrightarrow z+x \leq z+y\)
    by (cases \(x\); cases \(y\); cases \(z\) ) simp
    show \(x \leq y \vee y \leq x\)
    by (cases \(x\); cases \(y\) ) auto
qed
end
lemma example-real-min-ext-matrix:
    fixes \(x::\) ('a::enum,real-min ext) square
    shows \(\operatorname{minarc}_{M} x \preceq \ominus \ominus x\)
    by (rule agg-square-m-algebra.minarc-below)
```


### 5.2 Linearly Ordered Commutative Monoids

Any linearly ordered commutative monoid extended by new least and greatest elements forms a linear aggregation lattice. This is similar to linearly ordered commutative semigroups except that the aggregation $\perp+\perp$ produces the unit of the monoid instead of the least element. Applied to weighted graphs, this means that the aggregation of the empty graph will be the unit of the monoid (for example, 0 for real numbers under standard addition, instead of $\perp$ ).
class linordered-comm-monoid-add $=$ linordered-ab-semigroup-add + comm-monoid-add

```
datatype 'a ext0 =
    Bot
    | Val 'a
    | Top
```

```
instantiation ext0 :: (linordered-comm-monoid-add)
linear-aggregation-kleene-algebra
begin
fun plus-ext0 :: 'a ext0 \(\Rightarrow\) ' \(a\) ext0 \(\Rightarrow\) ' \(a\) ext0 where
    plus-ext0 Bot Bot = Val 0
| plus-ext0 Bot \(x=x\)
plus-ext0 (Val x) Bot = Val x
| plus-ext0 \((\) Val \(x)(\) Val \(y)=\operatorname{Val}(x+y)\)
| plus-ext0 (Val -) Top = Top
| plus-ext0 Top - = Top
fun sup-ext0 :: ' \(a\) ext \(0 \Rightarrow\) ' \(a\) ext \(0 \Rightarrow\) ' \(a\) ext0 where
    sup-ext0 Bot \(x=x\)
| sup-ext0 (Val x) Bot = Val x
\(\mid \sup -\operatorname{ext0}(\operatorname{Val} x)(\) Val \(y)=\operatorname{Val}(\max x y)\)
| sup-ext0 (Val -) Top \(=\) Top
| sup-ext0 Top - = Top
fun inf-ext0 :: ' \(a \operatorname{ext0} \Rightarrow{ }^{\prime} a \operatorname{ext0} \Rightarrow{ }^{\prime} a \operatorname{ext0}\) where
    inf-ext0 Bot - = Bot
| inf-ext0 (Val -) Bot \(=\) Bot
|inf-ext0 \((\) Val \(x)(\) Val \(y)=\operatorname{Val}(\min x y)\)
inf-ext0 (Val x) Top \(=\) Val \(x\)
| inf-ext0 Top \(x=x\)
fun times-ext0 :: 'a ext0 \(\Rightarrow{ }^{\prime} a\) ext0 \(\Rightarrow{ }^{\prime} a\) ext0 where times-ext0 \(x y=x \sqcap y\)
fun uminus-ext0 :: ' \(a\) ext0 \(\Rightarrow{ }^{\prime} a\) ext0 where
    uminus-ext0 Bot \(=\) Top
| uminus-ext0 (Val -) = Bot
| uminus-ext0 Top = Bot
fun star-ext0 :: ' \(a \operatorname{ext} 0 \Rightarrow{ }^{\prime} a\) ext0 where star-ext0 \(-=\) Top
fun conv-ext0 :: ' \(a\) ext \(0 \Rightarrow\) ' \(a\) ext 0 where conv-ext0 \(x=x\)
definition bot-ext0 :: ' \(a\) ext0 where bot-ext0 \(\equiv\) Bot
definition one-ext0 :: ' \(a\) ext0 where one-ext0 \(\equiv\) Top
definition top-ext0 :: ' \(a\) ext 0 where top-ext0 \(\equiv\) Top
fun less-eq-ext0 :: 'a ext0 \(\Rightarrow\) ' \(a\) ext0 \(\Rightarrow\) bool where
    less-eq-ext0 Bot - = True
| less-eq-ext0 (Val -) Bot = False
| less-eq-ext0 \((\) Val \(x)(\) Val \(y)=(x \leq y)\)
| less-eq-ext0 (Val -) Top = True
| less-eq-ext0 Top Bot = False
| less-eq-ext0 Top (Val -) = False
```

| less-eq-ext0 Top Top $=$ True
fun less-ext0 :: 'a ext0 $\Rightarrow$ 'a ext0 $\Rightarrow$ bool where less-ext0 $x y=(x \leq y \wedge \neg y \leq$ $x)$
instance
proof
fix $x y z::$ ' $a \operatorname{ext0}$
show $(x+y)+z=x+(y+z)$
by (cases $x$; cases $y$; cases $z$ ) (simp-all add: add.assoc)
show $x+y=y+x$
by (cases $x$; cases $y$ ) (simp-all add: add.commute)
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
by $\operatorname{simp}$
show $x \leq x$
using less-eq-ext0.elims(3) by fastforce
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
by (cases $x$; cases $y$ ) simp-all
show $x \sqcap y \leq x$
by (cases $x$; cases $y$ ) simp-all
show $x \sqcap y \leq y$
by (cases $x$; cases $y$ ) simp-all
show $x \leq y \Longrightarrow x \leq z \Longrightarrow x \leq y \sqcap z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq x \sqcup y$
by (cases $x$; cases $y$ ) simp-all
show $y \leq x \sqcup y$
by (cases $x$; cases y) simp-all
show $y \leq x \Longrightarrow z \leq x \Longrightarrow y \sqcup z \leq x$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show bot $\leq x$
by (simp add: bot-ext0-def)
show $x \leq t o p$
by (cases $x$ ) (simp-all add: top-ext0-def)
show $x \neq$ bot $\wedge x+$ bot $\leq y+$ bot $\longrightarrow x+z \leq y+z$
apply (cases $x$; cases $y$; cases $z$ )
prefer 11 using add-right-mono bot-ext0-def apply fastforce
by (simp-all add: bot-ext0-def add-right-mono)
show $x+y+b o t=x+y$
by (cases $x$; cases y) (simp-all add: bot-ext0-def)
show $x+y=$ bot $\longrightarrow x=$ bot
by (cases $x$; cases y) (simp-all add: bot-ext0-def)
show $x \leq y \vee y \leq x$
by (cases $x$; cases $y$ ) (simp-all add: linear)
show $-x=$ (if $x=$ bot then top else bot)
by (cases $x$ ) (simp-all add: bot-ext0-def top-ext0-def)
show $(1:: ' a$ ext0 $)=$ top

```
    by (simp add: one-ext0-def top-ext0-def)
    show }x*y=x\sqcap
    by simp
    show }\mp@subsup{x}{}{T}=
    by simp
    show }\mp@subsup{x}{}{\star}=to
    by (simp add: top-ext0-def)
qed
end
```

An example of a linearly ordered commutative monoid is the set of real numbers with standard addition and unit 0 .
instantiation real :: linordered-comm-monoid-add
begin
instance ..
end

### 5.3 Linearly Ordered Commutative Monoids with a Least Element

If a linearly ordered commutative monoid already contains a least element which is a unit of aggregation, only a new greatest element has to be added to obtain a linear aggregation lattice.

```
class linordered-comm-monoid-add-bot = linordered-ab-semigroup-add +
order-bot +
    assumes bot-zero [simp]: bot +x=x
begin
sublocale linordered-comm-monoid-add where zero = bot
    apply unfold-locales
    by simp
end
datatype 'a extT =
        Val 'a
    | Top
instantiation extT :: (linordered-comm-monoid-add-bot)
linear-aggregation-kleene-algebra
begin
fun plus-extT :: 'a extT = 'a extT व ' a extT where
    plus-extT (Val x) (Val y) = Val (x+y)
|plus-extT (Val -) Top = Top
```

```
| plus-extT Top - = Top
fun sup-ext \(T::\) ' \(a \operatorname{ext} T \Rightarrow\) ' \(a \operatorname{ext} T \Rightarrow\) ' \(a \operatorname{ext} T\) where
    sup-ext \(T(\operatorname{Val} x)(\) Val \(y)=\operatorname{Val}(\max x y)\)
    sup-extT (Val -) Top \(=\) Top
| sup-extT Top - = Top
fun \(\inf -\operatorname{ext} T::\) ' \(a \operatorname{ext} T \Rightarrow{ }^{\prime} a \operatorname{ext} T \Rightarrow{ }^{\prime} a \operatorname{ext} T\) where
    \(\inf -\operatorname{ext} T(\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(\min x y)\)
\(\mid \inf -\operatorname{ext} T(\) Val \(x)\) Top \(=\) Val \(x\)
\(\mid\) inf-extT Top \(x=x\)
fun times-ext \(T::\) ' \(a \operatorname{ext} T \Rightarrow{ }^{\prime} a \operatorname{ext} T \Rightarrow{ }^{\prime} a \operatorname{ext} T\) where times-ext \(T x y=x \sqcap y\)
fun uminus-ext \(T\) :: ' \(a\) ext \(T \Rightarrow\) 'a ext \(T\) where uminus-ext \(T x=(\) if \(x=\) Val bot then Top else Val bot)
fun star-ext \(T\) :: ' \(a\) ext \(T \Rightarrow\) 'a ext \(T\) where star-ext \(T-=\) Top
fun conv-ext \(T::\) ' \(a \operatorname{ext} T \Rightarrow{ }^{\prime} a \operatorname{ext} T\) where conv-ext \(T x=x\)
definition bot-ext \(T\) :: 'a ext \(T\) where bot-ext \(T \equiv\) Val bot
definition one-ext \(T\) :: 'a ext \(T\) where one-ext \(T \equiv\) Top
definition top-ext \(T\) :: 'a ext \(T\) where top-ext \(T \equiv\) Top
fun less-eq-ext \(T\) :: ' \(a\) ext \(T \Rightarrow\) 'a ext \(T \Rightarrow\) bool where
less-eq-ext \(T(\) Val \(x)(\) Val \(y)=(x \leq y)\)
| less-eq-extT Top (Val -) = False
less-eq-extT - Top \(=\) True
fun less-ext \(T::\) ' \(a\) ext \(T \Rightarrow\) ' \(a\) ext \(T \Rightarrow\) bool where less-extT \(x y=(x \leq y \wedge \neg y\) \(\leq x\) )
```

```
instance
```

instance
proof
proof
fix $x y z::$ ' $a$ ext $T$
fix $x y z::$ ' $a$ ext $T$
show $(x+y)+z=x+(y+z)$
show $(x+y)+z=x+(y+z)$
by (cases $x$; cases $y$; cases $z$ ) (simp-all add: add.assoc)
by (cases $x$; cases $y$; cases $z$ ) (simp-all add: add.assoc)
show $x+y=y+x$
show $x+y=y+x$
by (cases $x$; cases $y$ ) (simp-all add: add.commute)
by (cases $x$; cases $y$ ) (simp-all add: add.commute)
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
by $\operatorname{simp}$
by $\operatorname{simp}$
show $x \leq x$
show $x \leq x$
by (cases $x$ ) simp-all
by (cases $x$ ) simp-all
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
by (cases $x$; cases $y$ ) simp-all
by (cases $x$; cases $y$ ) simp-all
show $x \sqcap y \leq x$

```
    show \(x \sqcap y \leq x\)
```

```
    by (cases x; cases y) simp-all
show }x\sqcapy\leq
    by (cases x; cases y) simp-all
show }x\leqy\Longrightarrowx\leqz\Longrightarrowx\leqy\sqcap
    by (cases }x\mathrm{ ; cases }y\mathrm{ ; cases z) simp-all
show }x\leqx\sqcup
    by (cases x; cases y) simp-all
show }y\leqx\sqcup
    by (cases x; cases y) simp-all
show }y\leqx\Longrightarrowz\leqx\Longrightarrowy\sqcupz\leq
    by (cases x; cases y; cases z) simp-all
show bot \leqx
    by (cases x) (simp-all add: bot-extT-def)
show }x\leqto
    by (cases x) (simp-all add: top-extT-def)
show }x\not=\mathrm{ bot }\wedgex+bot\leqy+bot \longrightarrowx+z\leqy+
    by (cases x; cases y; cases z) (simp-all add: bot-extT-def add-right-mono)
show }x+y+bot=x+
    by (cases x; cases y) (simp-all add: bot-extT-def)
show }x+y=bot\longrightarrowx=bo
    apply (cases x; cases y)
    apply (metis (mono-tags) add.commute add-right-mono bot.extremum
bot.extremum-uniqueI bot-zero extT.inject plus-extT.simps(1) bot-extT-def)
    by (simp-all add: bot-extT-def)
show }x\leqy\veey\leq
    by (cases x; cases y) (simp-all add: linear)
show -x = (if }x=\mathrm{ bot then top else bot)
    by (cases x) (simp-all add: bot-extT-def top-extT-def)
show (1::'a extT) = top
    by (simp add: one-extT-def top-extT-def)
show }x*y=x\sqcap
    by simp
show \mp@subsup{x}{}{T}=x
    by simp
show }\mp@subsup{x}{}{\star}= to
    by (simp add: top-extT-def)
qed
end
```

An example of a linearly ordered commutative monoid with a least element is the set of real numbers extended by minus infinity with maximum as aggregation.

```
datatype real-max-bot =
    MInfty
    | real
```

instantiation real-max-bot :: linordered-comm-monoid-add-bot begin

```
definition bot-real-max-bot \equiv MInfty
fun less-eq-real-max-bot where
    less-eq-real-max-bot MInfty - = True
| less-eq-real-max-bot (R -) MInfty = False
| less-eq-real-max-bot (R x) (R y) = (x\leqy)
fun less-real-max-bot where
    less-real-max-bot - MInfty = False
| less-real-max-bot MInfty (R -) = True
| less-real-max-bot (R x) (R y) =(x<y)
fun plus-real-max-bot where
    plus-real-max-bot MInfty y = y
| plus-real-max-bot x MInfty = x
| plus-real-max-bot (R x) (R y) =R (max x y)
```


## instance

```
proof
    fix x y z :: real-max-bot
    show }(x+y)+z=x+(y+z
        by (cases x; cases y; cases z) simp-all
    show }x+y=y+
        by (cases x; cases y) simp-all
    show }(x<y)=(x\leqy^\negy\leqx
        by (cases x; cases y) auto
    show }x\leq
        by (cases x) simp-all
    show }x\leqy\Longrightarrowy\leqz\Longrightarrowx\leq
        by (cases x; cases y; cases z) simp-all
    show }x\leqy\Longrightarrowy\leqx\Longrightarrowx=
        by (cases x; cases y) simp-all
    show }x\leqy\Longrightarrowz+x\leqz+
        by (cases x; cases y; cases z) simp-all
    show }x\leqy\veey\leq
        by (cases x; cases y) auto
    show bot }\leq
        by (cases x) (simp-all add: bot-real-max-bot-def)
    show bot + x = x
        by (cases x) (simp-all add: bot-real-max-bot-def)
qed
end
```


### 5.4 Linearly Ordered Commutative Monoids with a Greatest Element

If a linearly ordered commutative monoid already contains a greatest element which is a unit of aggregation, only a new least element has to be added to obtain a linear aggregation lattice.

```
class linordered-comm-monoid-add-top \(=\) linordered-ab-semigroup-add +
order-top +
    assumes top-zero \([\) simp \(]\) : top \(+x=x\)
begin
sublocale linordered-comm-monoid-add where zero \(=\) top
    apply unfold-locales
    by \(\operatorname{simp}\)
lemma add-decreasing: \(x+y \leq x\)
    using add-left-mono top.extremum by fastforce
lemma \(t\)-min: \(x+y \leq \min x y\)
    using add-commute add-decreasing by force
end
datatype ' \(a \operatorname{ext} B=\)
    Bot
    | Val 'a
instantiation extB :: (linordered-comm-monoid-add-top)
linear-aggregation-kleene-algebra
begin
fun plus-ext \(B::\) ' \(a\) ext \(B \Rightarrow{ }^{\prime} a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B\) where
    plus-extB Bot Bot = Val top
| plus-extB Bot \((\) Val \(x)=\) Val \(x\)
| plus-extB \((\) Val \(x)\) Bot \(=\) Val \(x\)
\(\mid\) plus-extB \((\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(x+y)\)
fun sup-ext \(B\) :: ' \(a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B\) where
    sup-extB Bot \(x=x\)
\(\mid \sup -\operatorname{extB}(\) Val \(x)\) Bot \(=\) Val \(x\)
\(\mid \sup -\operatorname{ext} B(\operatorname{Val} x)(\) Val \(y)=\operatorname{Val}(\max x y)\)
fun \(\inf -\operatorname{ext} B::\) ' \(a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B\) where
    inf-extB Bot - = Bot
|inf-extB (Val-) Bot = Bot
\(\mid \operatorname{inf-extB}(\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(\min x y)\)
fun times-ext \(B::\) ' \(a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B \Rightarrow{ }^{\prime} a\) ext \(B\) where times-extB \(x y=x \sqcap y\)
fun uminus-ext \(B::\) ' \(a \operatorname{ext} B \Rightarrow{ }^{\prime} a \operatorname{ext} B\) where
    uminus-extB Bot = Val top
\(\mid\) uminus-extB (Val-) \(=\) Bot
```



```
fun conv-extB :: 'a extB # ' }a\mathrm{ extB where conv-extB }x=
definition bot-extB :: ' }a\mathrm{ extB where bot-ext B 三 Bot
definition one-extB :: ' }a\mathrm{ extB where one-extB इ Val top
definition top-extB :: 'a extB where top-extB\equiv Val top
fun less-eq-extB :: 'a extB => 'a extB }=>\mathrm{ bool where
    less-eq-extB Bot - = True
| less-eq-extB (Val -) Bot = False
| less-eq-extB (Val x) (Val y) = (x\leqy)
```



```
\leqx)
```


## instance

```
proof
    fix x y z :: 'a extB
    show }(x+y)+z=x+(y+z
        by (cases x; cases y; cases z) (simp-all add: add.assoc)
    show }x+y=y+
    by (cases }x\mathrm{ ; cases y) (simp-all add: add.commute)
    show }(x<y)=(x\leqy^\negy\leqx
    by simp
    show }x\leq
    by (cases x) simp-all
    show }x\leqy\Longrightarrowy\leqz\Longrightarrowx\leq
    by (cases x; cases y; cases z) simp-all
    show }x\leqy\Longrightarrowy\leqx\Longrightarrowx=
    by (cases x; cases y) simp-all
    show }x\sqcapy\leq
    by (cases x; cases y) simp-all
    show }x\sqcapy\leq
    by (cases x; cases y) simp-all
    show }x\leqy\Longrightarrowx\leqz\Longrightarrowx\leqy\sqcap
    by (cases }x\mathrm{ ; cases }y\mathrm{ ; cases z) simp-all
    show }x\leqx\sqcup
    by (cases x; cases y) simp-all
show }y\leqx\sqcup
    by (cases x; cases y) simp-all
    show }y\leqx\Longrightarrowz\leqx\Longrightarrowy\sqcupz\leq
    by (cases }x\mathrm{ ; cases }y\mathrm{ ; cases z) simp-all
show bot }\leq
    by (simp add: bot-extB-def)
show 1:x\leq top
    by (cases x) (simp-all add: top-extB-def)
show }x\not=\mathrm{ bot }\wedgex+\mathrm{ bot }\leqy+\mathrm{ bot }\longrightarrowx+z\leqy+
```

```
    apply (cases x; cases y; cases z)
    prefer 6 using 1 apply (metis (mono-tags, lifting) plus-extB.\operatorname{simps}(2,4)
top-extB-def add-right-mono less-eq-extB.simps(3) top-zero)
    by (simp-all add: bot-extB-def add-right-mono)
    show }x+y+bot=x+
    by (cases x; cases y) (simp-all add: bot-extB-def)
    show }x+y=\mathrm{ bot }\longrightarrowx=bo
    by (cases x; cases y) (simp-all add: bot-extB-def)
    show }x\leqy\veey\leq
    by (cases x; cases y) (simp-all add: linear)
    show -x = (if }x=\mathrm{ bot then top else bot)
    by (cases x) (simp-all add: bot-extB-def top-extB-def)
show (1::'a extB) = top
    by (simp add: one-extB-def top-extB-def)
show }x*y=x\sqcap
    by simp
show }\mp@subsup{x}{}{T}=
    by simp
show }\mp@subsup{x}{}{\star}= to
    by (simp add: top-extB-def)
qed
end
```

An example of a linearly ordered commutative monoid with a greatest element is the set of real numbers extended by infinity with minimum as aggregation.

```
datatype real-min-top \(=\)
    \(R\) real
    PInfty
instantiation real-min-top :: linordered-comm-monoid-add-top
begin
definition top-real-min-top \(\equiv\) PInfty
fun less-eq-real-min-top where
    less-eq-real-min-top - PInfty \(=\) True
| less-eq-real-min-top PInfty \((R-)=\) False
less-eq-real-min-top \((R x)(R y)=(x \leq y)\)
fun less-real-min-top where
    less-real-min-top PInfty - = False
|less-real-min-top ( \(R\)-) PInfty \(=\) True
|less-real-min-top \((R x)(R y)=(x<y)\)
fun plus-real-min-top where
    plus-real-min-top PInfty \(y=y\)
| plus-real-min-top x PInfty \(=x\)
```

```
| plus-real-min-top (R x) (Ry)=R(min x y)
```

```
instance
proof
    fix \(x\) y \(z\) :: real-min-top
    show \((x+y)+z=x+(y+z)\)
        by (cases \(x\); cases \(y\); cases \(z\) ) simp-all
    show \(x+y=y+x\)
        by (cases \(x\); cases y) simp-all
    show \((x<y)=(x \leq y \wedge \neg y \leq x)\)
        by (cases \(x\); cases \(y\) ) auto
    show \(x \leq x\)
        by (cases \(x\) ) simp-all
    show \(x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z\)
        by (cases \(x\); cases \(y\); cases \(z\) ) simp-all
    show \(x \leq y \Longrightarrow y \leq x \Longrightarrow x=y\)
        by (cases \(x\); cases \(y\) ) simp-all
    show \(x \leq y \Longrightarrow z+x \leq z+y\)
        by (cases \(x\); cases \(y\); cases \(z\) ) simp-all
    show \(x \leq y \vee y \leq x\)
        by (cases \(x\); cases \(y\) ) auto
    show \(x \leq t o p\)
        by (cases \(x\) ) (simp-all add: top-real-min-top-def)
    show top \(+x=x\)
        by (cases \(x\) ) (simp-all add: top-real-min-top-def)
qed
```

end

Another example of a linearly ordered commutative monoid with a greatest element is the unit interval of real numbers with any triangular norm (t-norm) as aggregation. Ideally, we would like to show that the unit interval is an instance of linordered-comm-monoid-add-top. However, this class has an addition operation, so the instantiation would require dependent types. We therefore show only the order property in general and a particular instance of the class.
typedef (overloaded) unit $=\{0 . .1\}$ :: real set
by auto
setup-lifting type-definition-unit
instantiation unit :: bounded-linorder
begin

```
lift-definition bot-unit :: unit is 0
    by \(\operatorname{simp}\)
```

lift-definition top-unit :: unit is 1
by $\operatorname{simp}$

```
lift-definition less-eq-unit :: unit }=>\mathrm{ unit }=>\mathrm{ bool is less-eq .
lift-definition less-unit :: unit }=>\mathrm{ unit }=>\mathrm{ bool is less .
instance
    apply intro-classes
    using bot-unit.rep-eq top-unit.rep-eq less-eq-unit.rep-eq less-unit.rep-eq
unit.Rep-unit-inject unit.Rep-unit by auto
end
```

    We give the Łukasiewicz t-norm as a particular instance.
    instantiation unit :: linordered-comm-monoid-add-top
begin
abbreviation $t l::$ real $\Rightarrow$ real $\Rightarrow$ real where
$t l x y \equiv \max (x+y-1) 0$
lemma tl-assoc:
$x \in\{0 . .1\} \Longrightarrow z \in\{0 . .1\} \Longrightarrow t l(t l x y) z=t l x(t l y z)$
by auto
lemma tl-top-zero:
$x \in\{0 . .1\} \Longrightarrow$ tl $1 x=x$
by auto
lift-definition plus-unit :: unit $\Rightarrow$ unit $\Rightarrow$ unit is $t l$
by $\operatorname{simp}$
instance
apply intro-classes
apply (metis (mono-tags, lifting) plus-unit.rep-eq unit.Rep-unit-inject
unit.Rep-unit tl-assoc)
using unit.Rep-unit-inject plus-unit.rep-eq apply fastforce
apply (simp add: less-eq-unit.rep-eq plus-unit.rep-eq)
by (metis (mono-tags, lifting) top-unit.rep-eq unit.Rep-unit-inject unit.Rep-unit
plus-unit.rep-eq tl-top-zero)
end

### 5.5 Linearly Ordered Commutative Monoids with a Least Element and a Greatest Element

If a linearly ordered commutative monoid already contains a least element which is a unit of aggregation and a greatest element, it forms a linear aggregation lattice.
class linordered-bounded-comm-monoid-add-bot $=$ linordered-comm-monoid-add-bot + order-top

```
begin
subclass bounded-linorder ..
subclass aggregation-order
    apply unfold-locales
    apply (simp add: add-right-mono)
    apply simp
    by (metis add-0-right add-left-mono bot.extremum bot.extremum-unique)
sublocale linear-aggregation-kleene-algebra where sup = max and inf = min
and times = min and conv =id and one = top and star = \lambdax.top and
uminus = \lambdax. if }x=\mathrm{ bot then top else bot
    apply unfold-locales
    by simp-all
lemma t-top: x + top = top
    by (metis add-right-mono bot.extremum bot-zero top-unique)
lemma add-increasing: }x\leqx+
    using add-left-mono bot.extremum by fastforce
lemma t-max: max x y \leqx+y
    using add-commute add-increasing by force
end
```

An example of a linearly ordered commutative monoid with a least and a greatest element is the unit interval of real numbers with any triangular conorm (t-conorm) as aggregation. For the reason outlined above, we show just a particular instance of linordered-bounded-comm-monoid-add-bot. Because the plus functions in the two instances given for the unit type are different, we work on a copy of the unit type.
typedef (overloaded) unit2 $=\{0 . .1\}::$ real set
by auto
setup-lifting type-definition-unit2
instantiation unit2 :: bounded-linorder
begin
lift-definition bot-unit2 :: unit2 is 0
by $\operatorname{simp}$
lift-definition top-unit2 :: unit2 is 1
by simp
lift-definition less-eq-unit2 :: unit2 $\Rightarrow$ unit2 $\Rightarrow$ bool is less-eq.
lift-definition less-unit2 :: unit2 $\Rightarrow$ unit2 $\Rightarrow$ bool is less .

## instance

apply intro-classes
using bot-unit2.rep-eq top-unit2.rep-eq less-eq-unit2.rep-eq less-unit2.rep-eq unit2.Rep-unit2-inject unit2.Rep-unit2 by auto
end
We give the product t-conorm as a particular instance.
instantiation unit2 :: linordered-bounded-comm-monoid-add-bot begin
abbreviation $s p::$ real $\Rightarrow$ real $\Rightarrow$ real where

$$
s p x y \equiv x+y-x * y
$$

lemma sp-assoc:
$s p(s p x y) z=s p x(s p y z)$
by (unfold left-diff-distrib right-diff-distrib distrib-left distrib-right) simp
lemma sp-mono:
assumes $z \in\{0 . .1\}$
and $x \leq y$
shows $s p z x \leq s p z y$
proof -
have $z+(1-z) * x \leq z+(1-z) * y$
using assms mult-left-mono by fastforce
thus ?thesis
by (unfold left-diff-distrib right-diff-distrib distrib-left distrib-right) simp
qed
lift-definition plus-unit2 :: unit2 $\Rightarrow$ unit2 $\Rightarrow$ unit2 is $s p$
proof -
fix $x y$ :: real
assume 1: $x \in\{0 . .1\}$
assume 2: $y \in\{0 . .1\}$
have $x-x * y \leq 1-y$
using 12 by (metis (full-types) atLeastAtMost-iff diff-ge-0-iff-ge
left-diff-distrib' mult.commute mult.left-neutral mult-left-le)
hence $3: x+y-x * y \leq 1$
by $\operatorname{simp}$
have $y *(x-1) \leq 0$
using 12 by (meson atLeastAtMost-iff le-iff-diff-le-0 mult-nonneg-nonpos)
hence $x+y-x * y \geq 0$
using 1 by (metis (no-types) atLeastAtMost-iff diff-diff-eq2 diff-ge-0-iff-ge
left-diff-distrib mult.commute mult.left-neutral order-trans)
thus $x+y-x * y \in\{0 . .1\}$
using 3 by simp
qed

```
instance
    apply intro-classes
    apply (metis (mono-tags, lifting) plus-unit2.rep-eq unit2.Rep-unit2-inject
sp-assoc)
    using unit2.Rep-unit2-inject plus-unit2.rep-eq apply fastforce
    using sp-mono unit2.Rep-unit2 less-eq-unit2.rep-eq plus-unit2.rep-eq apply
simp
    using bot-unit2.rep-eq unit2.Rep-unit2-inject plus-unit2.rep-eq by fastforce
end
```


### 5.6 Constant Aggregation

Any linear order with a constant element extended by new least and greatest elements forms a linear aggregation lattice where the aggregation returns the given constant.

```
class pointed-linorder = linorder +
    fixes const :: 'a
datatype 'a extC=
    Bot
    Val 'a
    Top
```

instantiation ext $C$ :: (pointed-linorder) linear-aggregation-kleene-algebra
begin
fun plus-ext $C$ :: ' $a$ ext $C \Rightarrow{ }^{\prime} a \operatorname{ext} C \Rightarrow$ ' $a$ ext $C$ where plus-ext $C$ x $y=$ Val const
fun sup-ext $C$ :: 'a ext $C \Rightarrow$ ' $a$ ext $C \Rightarrow{ }^{\prime} a$ ext $C$ where
sup-extC Bot $x=x$
$\mid \sup -e x t C($ Val $x)$ Bot $=$ Val $x$
$\mid \sup -\operatorname{ext} C(\operatorname{Val} x)($ Val $y)=\operatorname{Val}(\max x y)$
| sup-extC (Val -) Top = Top
| sup-extC Top - = Top
fun $\inf -\operatorname{ext} C::$ ' $a \operatorname{ext} C \Rightarrow$ 'a ext $C \Rightarrow$ ' $a \operatorname{ext} C$ where
inf-extC Bot - = Bot
|inf-extC (Val -) Bot = Bot
$\mid \operatorname{inf-extC}(\operatorname{Val} x)(\operatorname{Val} y)=\operatorname{Val}(\min x y)$
inf-extC (Val x) Top $=$ Val $x$
$\mid \inf -$ ext $C$ Top $x=x$
fun times-ext $C$ :: 'a ext $C \Rightarrow{ }^{\prime} a \operatorname{ext} C \Rightarrow$ ' $a$ ext $C$ where times-ext $C x y=x \sqcap y$
fun uminus-ext $C$ :: ' $a$ ext $C \Rightarrow$ ' $a$ ext $C$ where
uminus-ext C Bot $=$ Top
$\mid$ uminus-ext $C$ (Val -$)=$ Bot

```
| uminus-extC Top = Bot
fun star-extC :: 'a extC = ''a extC where star-extC - = Top
```



```
definition bot-extC :: ' a extC where bot-extC \equiv Bot
definition one-extC :: 'a extC where one-extC \equiv Top
definition top-extC :: ' a extC where top-extC \equiv Top
fun less-eq-extC :: 'a extC = 'a ext C => bool where
    less-eq-extC Bot - = True
| less-eq-extC (Val -) Bot = False
|less-eq-extC (Val x) (Val y) = (x\leqy)
|less-eq-extC (Val -) Top = True
less-eq-extC Top Bot = False
| less-eq-extC Top (Val -) = False
| less-eq-extC Top Top = True
fun less-ext \(C\) :: ' \(a\) ext \(C \Rightarrow{ }^{\prime} a\) ext \(C \Rightarrow\) bool where less-ext \(C x y=(x \leq y \wedge \neg y\) \(\leq x\) )
```

```
instance
```

instance
proof
proof
fix $x y z::$ 'a ext $C$
fix $x y z::$ 'a ext $C$
show $(x+y)+z=x+(y+z)$
show $(x+y)+z=x+(y+z)$
by $\operatorname{simp}$
by $\operatorname{simp}$
show $x+y=y+x$
show $x+y=y+x$
by $\operatorname{simp}$
by $\operatorname{simp}$
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
by $\operatorname{simp}$
by $\operatorname{simp}$
show $x \leq x$
show $x \leq x$
by (cases $x$ ) simp-all
by (cases $x$ ) simp-all
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
by (cases $x$; cases $y$ ) simp-all
by (cases $x$; cases $y$ ) simp-all
show $x \sqcap y \leq x$
show $x \sqcap y \leq x$
by (cases $x$; cases y) simp-all
by (cases $x$; cases y) simp-all
show $x \sqcap y \leq y$
show $x \sqcap y \leq y$
by (cases $x$; cases y) simp-all
by (cases $x$; cases y) simp-all
show $x \leq y \Longrightarrow x \leq z \Longrightarrow x \leq y \sqcap z$
show $x \leq y \Longrightarrow x \leq z \Longrightarrow x \leq y \sqcap z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq x \sqcup y$
show $x \leq x \sqcup y$
by (cases $x$; cases $y$ ) simp-all
by (cases $x$; cases $y$ ) simp-all
show $y \leq x \sqcup y$
show $y \leq x \sqcup y$
by (cases $x$; cases $y$ ) simp-all
by (cases $x$; cases $y$ ) simp-all
show $y \leq x \Longrightarrow z \leq x \Longrightarrow y \sqcup z \leq x$
show $y \leq x \Longrightarrow z \leq x \Longrightarrow y \sqcup z \leq x$
by (cases $x$; cases $y$; cases $z$ ) simp-all

```
    by (cases \(x\); cases \(y\); cases \(z\) ) simp-all
```

```
show bot \leqx
    by (simp add: bot-extC-def)
show }x\leqto
    by (cases x) (simp-all add: top-extC-def)
show }x\not=\mathrm{ bot }\wedgex+bot\leqy+bot \longrightarrowx+z\leqy+
    by simp
show }x+y+bot=x+
    by simp
show }x+y=\mathrm{ bot }\longrightarrowx=bo
    by (simp add: bot-extC-def)
show }x\leqy\veey\leq
    by (cases x; cases y) (simp-all add: linear)
show -x=(if x=bot then top else bot)
    by (cases x) (simp-all add: bot-extC-def top-extC-def)
show (1::'a extC) = top
    by (simp add: one-extC-def top-extC-def)
show }x*y=x\sqcap
    by simp
show }\mp@subsup{x}{}{T}=
    by simp
show }\mp@subsup{x}{}{\star}= to
    by (simp add: top-extC-def)
qed
end
```

An example of a linear order is the set of real numbers. Any real number can be chosen as the constant.

```
instantiation real :: pointed-linorder
begin
```

instance ..
end

The following instance shows that any linear order with a constant forms a linearly ordered commutative semigroup with the alpha-median operation as aggregation. The alpha-median of two elements is the median of these elements and the given constant.

```
fun median3 :: 'a::ord }=>\mp@subsup{}{}{\prime}a=>\mp@subsup{}{}{\prime}a=>'a wher
    median3 x y z =
    (if }x\leqy\wedgey\leqz\mathrm{ then }y\mathrm{ else
    if }x\leqz\wedgez\leqy\mathrm{ then z else
    if }y\leqx\wedgex\leqz\mathrm{ then x else
    if }y\leqz\wedgez\leqx\mathrm{ then z else
    if z\leqx^x\leqy then x else y)
```

interpretation alpha-median: linordered-ab-semigroup-add where plus $=$ median3 const and less-eq $=$ less-eq and less $=$ less

## proof

fix $a b c::{ }^{\prime} a$
show median3 const (median3 const $a b) c=$ median3 const $a$ (median3 const $b$ c)
by (cases const $\leq a$; cases const $\leq b$; cases const $\leq c$; cases $a \leq b$; cases $a \leq$ $c$; cases $b \leq c$ ) auto
show median3 const $a b=$ median3 const $b a$
by (cases const $\leq a$; cases const $\leq b$; cases $a \leq b$ ) auto
assume $a \leq b$
thus median3 const $c$ a median3 const $c b$
by (cases const $\leq a ;$ cases const $\leq b$; cases const $\leq c$; cases $a \leq c$; cases $b \leq$ c) auto
qed

### 5.7 Counting Aggregation

Any linear order extended by new least and greatest elements and a copy of the natural numbers forms a linear aggregation lattice where the aggregation counts non- $\perp$ elements using the copy of the natural numbers.

```
datatype ' \(a \operatorname{ext} N=\)
    Bot
    Val 'a
    \| nat
    | Top
instantiation ext \(N\) :: (linorder) linear-aggregation-kleene-algebra
begin
```

```
fun plus-ext \(N::\) ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N\) where
```

fun plus-ext $N::$ ' $a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N$ where
plus-extN Bot Bot $=N 0$
plus-extN Bot Bot $=N 0$
plus-extN Bot (Val -) = N 1
plus-extN Bot (Val -) = N 1
plus-ext $N \operatorname{Bot}(N y)=N y$
plus-ext $N \operatorname{Bot}(N y)=N y$
plus-extN Bot Top $=$ N 1
plus-extN Bot Top $=$ N 1
plus-extN (Val -) Bot = N 1
plus-extN (Val -) Bot = N 1
| plus-extN (Val -) (Val -) = N 2
| plus-extN (Val -) (Val -) = N 2
| plus-ext $N($ Val -) $(N y)=N(y+1)$
| plus-ext $N($ Val -) $(N y)=N(y+1)$
|plus-ext $N($ Val -) Top $=N 2$
|plus-ext $N($ Val -) Top $=N 2$
plus-ext $N(N x)$ Bot $=N x$
plus-ext $N(N x)$ Bot $=N x$
$\mid$ plus-ext $N(N x)($ Val -) $)=N(x+1)$
$\mid$ plus-ext $N(N x)($ Val -) $)=N(x+1)$
plus-ext $N(N x)(N y)=N(x+y)$
plus-ext $N(N x)(N y)=N(x+y)$
plus-ext $N(N x)$ Top $=N(x+1)$
plus-ext $N(N x)$ Top $=N(x+1)$
plus-extN Top Bot = N 1
plus-extN Top Bot = N 1
plus-extN Top (Val -) = N 2
plus-extN Top (Val -) = N 2
plus-ext $N$ Top $(N y)=N(y+1)$
plus-ext $N$ Top $(N y)=N(y+1)$
| plus-ext $N$ Top Top = N 2
| plus-ext $N$ Top Top = N 2
fun sup-ext $N::$ ' $a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N$ where
fun sup-ext $N::$ ' $a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N$ where
sup-extN Bot $x=x$
sup-extN Bot $x=x$
sup-extN $($ Val $x)$ Bot $=\operatorname{Val} x$

```
sup-extN \((\) Val \(x)\) Bot \(=\operatorname{Val} x\)
```

```
\(\mid \sup -\operatorname{ext} N(\operatorname{Val} x)(\) Val \(y)=\operatorname{Val}(\max x y)\)
| sup-ext \(N(\) Val -) \((N y)=N y\)
|sup-extN (Val -) Top \(=\) Top
| sup-extN \((N x)\) Bot \(=N x\)
\(\mid \sup -\operatorname{ext} N(N x)(\) Val -) \(=N x\)
\(\mid \sup -\operatorname{ext} N(N x)(N y)=N(\max x y)\)
| sup-extN (N -) Top = Top
| sup-extN Top - = Top
fun \(\inf -\operatorname{ext} N::\) ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow\) ' \(a \operatorname{ext} N\) where
    inf-extN Bot - = Bot
| inf-extN (Val -) Bot = Bot
\(\mid \inf -\operatorname{ext} N(\operatorname{Val} x)(\) Val \(y)=\operatorname{Val}(\min x y)\)
inf-extN \((\) Val \(x)(N-)=\) Val \(x\)
inf-extN (Val x) Top \(=\) Val \(x\)
|inf-extN (N -) Bot = Bot
\(\mid \operatorname{inf-extN}(N-)(\) Val \(y)=\) Val \(y\)
\(\inf -\operatorname{ext} N(N x)(N y)=N(\min x y)\)
\(\mid \inf -\operatorname{ext} N(N x)\) Top \(=N x\)
\(\mid \inf -\operatorname{ext} N\) Top \(y=y\)
fun times-ext \(N::\) ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N\) where times-ext \(N x y=x \sqcap y\)
fun uminus-ext \(N\) :: ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N\) where
    uminus-extN Bot \(=\) Top
| uminus-extN (Val-) = Bot
| uminus-ext \(N\left(N_{-}\right)=\)Bot
| uminus-extN Top = Bot
fun star-ext \(N::\) ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N\) where \(\operatorname{star-ext} N-=\) Top
fun conv-extN :: ' \(a \operatorname{ext} N \Rightarrow{ }^{\prime} a \operatorname{ext} N\) where conv-ext \(N x=x\)
definition bot-ext \(N\) :: 'a ext \(N\) where bot-ext \(N \equiv\) Bot
definition one-ext \(N\) :: 'a ext \(N\) where one-ext \(N \equiv \operatorname{Top}\)
definition top-ext \(N\) :: 'a ext \(N\) where top-ext \(N \equiv\) Top
fun less-eq-ext \(N\) :: ' \(a \operatorname{ext} N \Rightarrow\) 'a ext \(N \Rightarrow\) bool where
    less-eq-extN Bot - = True
| less-eq-ext N (Val -) Bot \(=\) False
|less-eq-ext \(N(\) Val \(x)(\) Val \(y)=(x \leq y)\)
| less-eq-ext \(N(\) Val -) \((N-)=\) True
|less-eq-extN (Val -) Top = True
|less-eq-ext \(N(N-)\) Bot \(=\) False
|less-eq-extN ( \(N\)-) (Val -) \(=\) False
|less-eq-ext \(N(N x)(N y)=(x \leq y)\)
| less-eq-ext \(N(N-)\) Top \(=\) True
| less-eq-extN Top Bot = False
|less-eq-extN Top (Val -) = False
```

```
| less-eq-extN Top (N -) = False
| less-eq-extN Top Top = True
```

fun less-extN :: 'a ext $N \Rightarrow{ }^{\prime} a$ ext $N \Rightarrow$ bool where less-ext $N x y=(x \leq y \wedge \neg y$ $\leq x$ )

## instance

## proof

fix $x y z::$ ' $a$ ext $N$
show $(x+y)+z=x+(y+z)$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x+y=y+x$
by (cases $x$; cases $y$ ) simp-all
show $(x<y)=(x \leq y \wedge \neg y \leq x)$
by $\operatorname{simp}$
show $x \leq x$
by (cases $x$ ) simp-all
show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq y \Longrightarrow y \leq x \Longrightarrow x=y$
by (cases $x$; cases y) simp-all
show $x \sqcap y \leq x$
by (cases $x$; cases $y$ ) simp-all
show $x \sqcap y \leq y$
by (cases $x$; cases $y$ ) simp-all
show $x \leq y \Longrightarrow x \leq z \Longrightarrow x \leq y \sqcap z$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show $x \leq x \sqcup y$
by (cases $x$; cases $y$ ) simp-all
show $y \leq x \sqcup y$
by (cases $x$; cases y) simp-all
show $y \leq x \Longrightarrow z \leq x \Longrightarrow y \sqcup z \leq x$
by (cases $x$; cases $y$; cases $z$ ) simp-all
show bot $\leq x$
by (simp add: bot-ext $N$-def)
show $x \leq$ top
by (cases $x$ ) (simp-all add: top-extN-def)
show $x \neq$ bot $\wedge x+$ bot $\leq y+$ bot $\longrightarrow x+z \leq y+z$
by (cases $x$; cases $y$; cases $z$ ) (simp-all add: bot-extN-def)
show $x+y+$ bot $=x+y$
by (cases $x$; cases $y$ ) (simp-all add: bot-ext $N$-def)
show $x+y=$ bot $\longrightarrow x=$ bot
by (cases $x$; cases $y$ ) (simp-all add: bot-ext $N$-def)
show $x \leq y \vee y \leq x$
by (cases $x$; cases y) (simp-all add: linear)
show $-x=$ (if $x=$ bot then top else bot)
by (cases $x$ ) (simp-all add: bot-ext $N$-def top-ext $N$-def)
show $\left(1::^{\prime} a \operatorname{extN}\right)=$ top
by (simp add: one-ext $N$-def top-ext $N$-def)

```
        show }x*y=x\sqcap
    by simp
show }\mp@subsup{x}{}{T}=
    by simp
show }\mp@subsup{x}{}{\star}=to
    by (simp add: top-extN-def)
qed
end
end
```


## 6 An Operation to Select Components in Algebras with Minimisation

In this theory we show that an operation to select components of a graph can be defined in m-Kleene Algebras. This work is by Nicolas Robinson-O'Brien.
theory M-Choose-Component

## imports

Stone-Relation-Algebras.Choose-Component
Matrix-Aggregation-Algebras

## begin

Every m-kleene-algebra is an instance of choose-component-algebra when the choose-component operation is defined as follows:

```
context m-kleene-algebra
begin
definition m-choose-component e v\equiv
    if vector-classes e v then
        e* minarc(v) * top
    else
        bot
```

sublocale m-choose-component-algebra: choose-component-algebra where
choose-component $=m$-choose-component
proof
fix $e v$
show $m$-choose-component e $v \leq--v$
proof (cases vector-classes e v)
case True
hence $m$-choose-component e $v=e * \operatorname{minarc}(v) *$ top
by (simp add: m-choose-component-def)
also have $\ldots \leq e *--v *$ top
by (simp add: comp-isotone minarc-below)
also have $\ldots=e * v *$ top
using True vector-classes-def by auto
also have $\ldots \leq v *$ top
using True vector-classes-def mult-assoc by auto
finally show ?thesis
using True vector-classes-def by auto
next
case False
hence $m$-choose-component e $v=$ bot
using False m-choose-component-def by auto
thus ?thesis
by $\operatorname{simp}$
qed
next
fix $e v$
show vector (m-choose-component ev)
proof (cases vector-classes ev)
case True
thus ?thesis
by (simp add: mult-assoc m-choose-component-def)
next
case False
thus ?thesis
by ( $\operatorname{simp}$ add: m-choose-component-def)
qed
next
fix $e v$
show regular ( $m$-choose-component e $v$ )
using minarc-regular regular-mult-closed vector-classes-def
m-choose-component-def by auto
next
fix $e v$
show m-choose-component ev*(m-choose-component ev) $)^{T} \leq e$
proof (cases vector-classes e $v$ )
case True
assume 1: vector-classes e $v$
hence $m$-choose-component ev*(m-choose-component ev) $=e * \operatorname{minarc}(v)$

* top $*(e * \operatorname{minarc}(v) * t o p)^{T}$
by (simp add: m-choose-component-def)
also have $\ldots=e * \operatorname{minarc}(v) * \operatorname{top} * \operatorname{top}^{T} * \operatorname{minarc}(v)^{T} * e^{T}$
by (metis comp-associative conv-dist-comp)
also have $\ldots=e * \operatorname{minarc}(v) * \operatorname{top} * \operatorname{top} * \operatorname{minarc}(v)^{T} * e$
using True vector-classes-def by auto
also have $\ldots=e * \operatorname{minarc}(v) * \operatorname{top} * \operatorname{minarc}(v)^{T} * e$
by (simp add: comp-associative)
also have $\ldots \leq e$
proof (cases $\bar{v}=b o t)$
case True
thus ?thesis
by (simp add: True minarc-bot)


## next

case False
assume $3: v \neq$ bot
hence $e * \operatorname{minarc}(v) *$ top $* \operatorname{minarc}(v)^{T} \leq e * 1$
using 3 minarc-arc arc-expanded comp-associative mult-right-isotone by
fastforce
hence $e * \operatorname{minarc}(v) *$ top $* \operatorname{minarc}(v)^{T} * e \leq e * 1 * e$
using mult-left-isotone by auto
also have $\ldots=e$
using True preorder-idempotent vector-classes-def by auto
thus ?thesis
using calculation by auto
qed
thus ?thesis
by (simp add: calculation)
next
case False
thus ?thesis
by (simp add: m-choose-component-def)
qed
next
fix $e v$
show $e * m$-choose-component $e v \leq m$-choose-component $e v$
proof (cases vector-classes e v)
case True
thus ?thesis
using comp-right-one dual-order.eq-iff mult-isotone vector-classes-def
$m$-choose-component-def mult-assoc by metis
next
case False
thus ?thesis
by (simp add: m-choose-component-def)
qed
next
fix $e v$
show vector-classes e $v \longrightarrow m$-choose-component e $v \neq b o t$
proof (cases vector-classes e $v$ )
case True
hence $m$-choose-component e $v \geq \operatorname{minarc}(v) *$ top
using vector-classes-def m-choose-component-def comp-associative
minarc-arc shunt-bijective by fastforce
also have $\ldots \geq \operatorname{minarc}(v)$
using calculation dual-order.trans top-right-mult-increasing by blast
thus ?thesis
using le-bot minarc-bot-iff vector-classes-def by fastforce
next
case False
thus ?thesis
by blast

```
    qed
qed
sublocale m-choose-component-algebra-tarski: choose-component-algebra-tarski
where choose-component = m-choose-component
end
class m-kleene-algebra-choose-component = m-kleene-algebra +
choose-component-algebra
end
```


## References

[1] W. Guttmann. Relation-algebraic verification of Prim's minimum spanning tree algorithm. In A. Sampaio and F. Wang, editors, Theoretical Aspects of Computing - ICTAC 2016, volume 9965 of Lecture Notes in Computer Science, pages 51-68. Springer, 2016.
[2] W. Guttmann. Stone-Kleene relation algebras. Archive of Formal Proofs, 2017.
[3] W. Guttmann. Stone relation algebras. In P. Höfner, D. Pous, and G. Struth, editors, Relational and Algebraic Methods in Computer Science, volume 10226 of Lecture Notes in Computer Science, pages 127-143. Springer, 2017.
[4] W. Guttmann. An algebraic framework for minimum spanning tree problems. Theoretical Computer Science, 2018. https://doi.org/10.1016/ j.tcs.2018.04.012.
[5] W. Guttmann. Verifying minimum spanning tree algorithms with Stone relation algebras. Journal of Logical and Algebraic Methods in Programming, 2018. https://doi.org/10.1016/j.jlamp.2018.09.005.
[6] W. Guttmann and N. Robinson-O'Brien. Relational minimum spanning tree algorithms. Archive of Formal Proofs, 2020.

