Verifying Minimum Spanning Tree Algorithms with Stone Relation Algebras

Walter Guttmann

October 31, 2017

Abstract

We develop algebras for aggregation and minimisation for weight matrices and for edge weights. We verify the correctness of Prim's and Kruskal's minimum spanning tree algorithms based on these algebras. We also show numerous instances of these algebras based on linearly ordered commutative semigroups.

These theories are being prepared for the Archive of Formal Proofs. Further documentation will be added.

Contents

1 Overview **2**

2 Big sum and product over finite (non-empty) sets **2**
2.1 Generic Abelian semigroup operation over a set 2
2.2 Generalized summation over a set 13
2.2.1 Properties in more restricted classes of structures . . . 14

3 Algebras for Aggregation and Minimisation **18**

4 Matrix Algebras for Aggregation and Minimisation **21**

5 Algebras for Aggregation and Minimisation with a Linear Order **40**

6 Hoare Logic for Total Correctness **63**

7 Minimum Spanning Tree Algorithms **69**
7.1 Kruskal's Minimum Spanning Tree Algorithm 69
7.2 Prim's Minimum Spanning Tree Algorithm 82

1 Overview

This document describes the following six theory files:

* Big sums over semigroups generalises Isabelle/HOL's `Groups_Big.thy` theory of finite summation 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-Kleene-algebras 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.
* Hoare Logic extends Isabelle/HOL's `Hoare/Hoare_Logic.thy` to total-correctness proofs.
* Minimum Spanning Tree proves correctness of Prim's and Kruskal's algorithms based on m-Kleene-algebras using Hoare logic.

This is an extended version of theories developed for [1, 3, 2].

2 Big sum and product over finite (non-empty) sets

theory *Semigroups-Big*
imports *Finite-Set Power*
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 it 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 $z :: 'a$ (**1**)

assumes *z-neutral* [*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 $F :: ('b \Rightarrow 'a) \Rightarrow 'b\ set \Rightarrow 'a$
where *eq-fold*: $F\ g\ A =$ *Finite-Set.fold* $(f \circ g)\ \mathbf{1}\ A$

lemma *infinite* [*simp*]: $\neg$ *finite* $A \Longrightarrow F\ g\ A = \mathbf{1}$
by (*simp add*: *eq-fold*)

lemma *empty* [*simp*]: $F\ g\ \{\} = \mathbf{1}$
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 ‹$x \in A$› **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* $x \in A$) (*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*]: $F\ (\lambda\text{-}.\ \mathbf{1})\ A = \mathbf{1}$
by (*simp add*: *neutral*)

lemma *F-one* [*simp*]: $F\ g\ A * \mathbf{1} = F\ g\ A$
proof −
have $\bigwedge f\ b\ B.\ F\ f$ (*insert* (b::'b) B) $* \mathbf{1} = F\ f$ (*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* [*of - 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* **and** *finite B*
assumes $A \cap B = \{\}$
shows $F\ g\ (A \cup B) = F\ g\ A * F\ g\ B$
using *assms* **by** (*simp add*: *union-inter-neutral*)

lemma *union-diff2*:
assumes *finite A* **and** *finite B*
shows $F\ g\ (A \cup B) = F\ g\ (A - B) * F\ g\ (B - A) * F\ g\ (A \cap B)$
proof −
have $A \cup B = A - B \cup (B - A) \cup A \cap B$
by *auto*
with *assms* **show** *?thesis*
by *simp* (*subst union-disjoint*, *auto*)+
qed

lemma *subset-diff*:
assumes $B \subseteq A$ **and** *finite* A
shows $F\ g\ A = F\ g\ (A - B) * F\ g\ B$
proof −
from *assms* **have** *finite* $(A - B)$ **by** *auto*
moreover from *assms* **have** *finite* B **by** (*rule finite-subset*)
moreover from *assms* **have** $(A - B) \cap B = \{\}$ **by** *auto*
ultimately have $F\ g\ (A - B \cup B) = F\ g\ (A - B) * F\ g\ B$ **by** (*rule union-disjoint*)
moreover from *assms* **have** $A \cup B = A$ **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma *setdiff-irrelevant*:
assumes *finite* A
shows $F\ g\ (A - \{x.\ g\ x = z\}) = F\ g\ A$
using *assms* **by** (*induct A*) (*simp-all add*: *insert-Diff-if*)

lemma *not-neutral-contains-not-neutral*:
assumes $F\ g\ A \neq \mathbf{1}$
obtains a **where** $a \in A$ **and** $g\ a \neq \mathbf{1}$
proof −
from *assms* **have** $\exists\, a{\in}A.\ g\ a \neq \mathbf{1}$
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 $F\ g\ (h\ `\ A) = F\ (g \circ h)\ A$
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** ¬ *finite* $(h\ `\ A)$ **by** (*blast dest*: *finite-imageD*)
with *False* **show** *?thesis* **by** *simp*
qed

lemma *cong* [*fundef-cong*]:
assumes $A = B$
assumes *g-h*: $\bigwedge x.\ x \in B \Longrightarrow g\ x = h\ x$
shows $F\ g\ A = F\ h\ B$
using *g-h* **unfolding** ‹$A = B$›
by (*induct B rule*: *infinite-finite-induct*) *auto*

lemma *strong-cong* [*cong*]:
assumes $A = B$ $\bigwedge x.\ x \in B$ =*simp*=> $g\ x = h\ x$
shows $F\ (\lambda x.\ g\ x)\ A = F\ (\lambda x.\ h\ x)\ B$
by (*rule cong*) (*use assms* **in** ‹*simp-all add*: *simp-implies-def*›)

lemma *reindex-cong*:
assumes *inj-on l B*
assumes $A = l$ ‘ B
assumes $\bigwedge x.\ x \in B \Longrightarrow g\ (l\ x) = h\ x$
shows $F\ g\ A = F\ h\ B$
using *assms* **by** (*simp add*: *reindex*)

lemma *UNION-disjoint*:
assumes *finite I* **and** $\forall\, i{\in}I.$ *finite* $(A\ i)$
and $\forall\, i{\in}I.\ \forall\, j{\in}I.\ i \neq j \longrightarrow A\ i \cap A\ j = \{\}$
shows $F\ g\ (\mathit{UNION}\ I\ A) = F\ (\lambda x.\ F\ g\ (A\ x))\ I$
apply (*insert assms*)
apply (*induct rule*: *finite-induct*)
apply *simp*
apply *atomize*
apply (*subgoal-tac* $\forall\, i{\in}\mathit{Fa}.\ x \neq i$)
prefer *2* **apply** *blast*
apply (*subgoal-tac* $A\ x \cap \mathit{UNION}\ \mathit{Fa}\ A = \{\}$)
prefer *2* **apply** *blast*
apply (*simp add*: *union-disjoint*)
done

lemma *Union-disjoint*:
assumes $\forall\, A{\in}C.$ *finite* $A\ \forall\, A{\in}C.\ \forall\, B{\in}C.\ A \neq B \longrightarrow A \cap B = \{\}$
shows $F\ g\ (\bigcup C) = (F \circ F)\ g\ C$
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*: $F\ (\lambda x.\ g\ x * h\ x)\ A = F\ g\ A * F\ h\ A$
by (*induct A rule*: *infinite-finite-induct*) (*simp-all add*: *assoc commute left-commute*)

lemma *Sigma*:
finite A $\Longrightarrow \forall x{\in}A.$ *finite* $(B\ x) \Longrightarrow F\ (\lambda x.\ F\ (g\ x)\ (B\ x))\ A = F$ (*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*: $R\ \mathbf{1}\ \mathbf{1}$
and *Rop*: $\forall x1\ y1\ x2\ y2.\ R\ x1\ x2 \wedge R\ y1\ y2 \longrightarrow R\ (x1 * y1)\ (x2 * y2)$
and *fin*: *finite S*
and *R-h-g*: $\forall x{\in}S.\ R\ (h\ x)\ (g\ x)$
shows $R\ (F\ h\ S)\ (F\ g\ S)$
using *fin* **by** (*rule finite-subset-induct*) (*use assms* **in** *auto*)

lemma *mono-neutral-cong-left*:
assumes *finite T*
and $S \subseteq T$
and $\forall i \in T - S.\ h\ i = \mathbf{1}$
and $\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x$
shows $F\ g\ S = F\ h\ T$
proof−
have *eq*: $T = S \cup (T - S)$ **using** ‹$S \subseteq T$› **by** *blast*
have *d*: $S \cap (T - S) = \{\}$ **using** ‹$S \subseteq T$› **by** *blast*
from ‹*finite T*› ‹$S \subseteq T$› **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 assms*(*3*)])
qed

lemma *mono-neutral-cong-right*:
finite $T \Longrightarrow S \subseteq T \Longrightarrow \forall i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x) \Longrightarrow$
$F\ g\ T = F\ h\ S$
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 *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*:
$\bigwedge a.\ a \in S \Longrightarrow i\ (j\ a) = a$
$\bigwedge a.\ a \in S \Longrightarrow j\ a \in T$
$\bigwedge b.\ b \in T \Longrightarrow j\ (i\ b) = b$
$\bigwedge b.\ b \in T \Longrightarrow i\ b \in S$
assumes *eq*:
$\bigwedge a.\ a \in S \Longrightarrow h\ (j\ a) = g\ a$
shows $F\ g\ S = F\ h\ T$
proof −
have *bij-betw j S T*
using *bij-betw-byWitness*[**where** $A{=}S$ **and** $f{=}j$ **and** $f'{=}i$ **and** $A'{=}T$] *witness* **by** *auto*
moreover have $F\ g\ S = F\ (\lambda x.\ h\ (j\ x))\ S$
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*:
$\bigwedge a.\ a \in S' \Longrightarrow g\ (h\ a) = z$
$\bigwedge b.\ b \in T' \Longrightarrow g\ b = z$
shows $F\ (\lambda x.\ g\ (h\ x))\ S = F\ g\ T$
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** $F\ (\lambda x.\ g\ (h\ x))\ S = F\ (\lambda x.\ g\ (h\ x))\ (S - S')$
by (*intro mono-neutral-cong-right*) *auto*
also have $\ldots = F\ g\ (T - T')$
using *bij* **by** (*rule reindex-bij-betw*)
also have $\ldots = F\ g\ T$
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: ⋀x y. x ∈ A ⟹ y ∈ A ⟹ x ≠ y ⟹ h x = h y ⟹ g (h x) = 1
  shows F g (h ‘ A) = F (g ∘ h) A
proof (subst reindex-bij-betw-not-neutral [symmetric])
  show bij-betw h (A − {x ∈ A. (g ∘ h) x = 1}) (h ‘ A − h ‘ {x ∈ A. (g ∘ h) 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 ∈ S − S' ⟹ i (j a) = a
    ⋀a. a ∈ S − S' ⟹ j a ∈ T − T'
    ⋀b. b ∈ T − T' ⟹ j (i b) = b
    ⋀b. b ∈ T − T' ⟹ i b ∈ S − S'
  assumes nn:
    ⋀a. a ∈ S' ⟹ g a = z
    ⋀b. b ∈ T' ⟹ h b = z
  assumes eq:
    ⋀a. a ∈ S ⟹ h (j a) = g a
  shows F g S = F h T
proof −
  have bij: bij-betw j (S − (S' ∩ S)) (T − (T' ∩ T))
    using witness by (intro bij-betw-byWitness[where f'=i]) auto
  have F-eq: F g S = F (λx. h (j x)) S
    by (intro cong) (auto simp: eq)
  show ?thesis
    unfolding F-eq using fin nn eq
    by (intro reindex-bij-betw-not-neutral[OF - - bij]) auto
qed

lemma delta:
  assumes fS: finite S
  shows F (λk. if k = a then b k else 1) S = (if a ∈ S then b a * 1 else 1)
proof −
  let ?f = (λk. if k = a then b k else 1)
  show ?thesis
  proof (cases a ∈ S)
    case False
    then have ∀k∈S. ?f k = 1 by simp
```

```
    with False show ?thesis by simp
  next
    case True
    let ?A = S − {a}
    let ?B = {a}
    from True have eq: S = ?A ∪ ?B by blast
    have dj: ?A ∩ ?B = {} by simp
    from fS have fAB: 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 commute z-neutral by force
  qed
qed

lemma delta′:
  assumes fin: finite S
  shows F (λk. if a = k then b k else 1) S = (if a ∈ S then b a ∗ 1 else 1)
  using delta [OF fin, of a b, symmetric] by (auto intro: cong)

lemma If-cases:
  fixes P :: 'b ⇒ bool and g h :: 'b ⇒ 'a
  assumes fin: finite A
  shows F (λx. if P x then h x else g x) A = F h (A ∩ {x. P x}) ∗ F g (A ∩ −
{x. P x})
proof −
  have a: A = A ∩ {x. P x} ∪ A ∩ −{x. P x} (A ∩ {x. P x}) ∩ (A ∩ −{x. P
x}) = {}
    by blast+
  from fin have f: finite (A ∩ {x. P x}) finite (A ∩ −{x. P x}) by auto
  let ?g = λx. if P x then h x else g x
  from union-disjoint [OF f a(2), of ?g] a(1) show ?thesis
    by (subst (1 2) cong) simp-all
qed

lemma cartesian-product: F (λx. F (g x) B) A = F (case-prod g) (A × B)
  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 A*
shows $F\ g\ (A \cap B) = F\ (\lambda x.\ \textit{if}\ x \in B\ \textit{then}\ g\ x\ \textit{else}\ \mathbf{1})\ A$
proof −
let $?g = \lambda x.\ \textit{if}\ x \in A \cap B\ \textit{then}\ g\ x\ \textit{else}\ \mathbf{1}$
have $\forall\, i{\in}A - A \cap B.\ (\textit{if}\ i \in A \cap B\ \textit{then}\ g\ i\ \textit{else}\ \mathbf{1}) = \mathbf{1}$ **by** *simp*
moreover have $A \cap B \subseteq A$ **by** *blast*
ultimately have $F\ ?g\ (A \cap B) = F\ ?g\ A$
using ‹*finite A*› **by** (*intro mono-neutral-left*) *auto*
then show *?thesis* **by** *simp*
qed

lemma *inter-filter*:
finite $A \Longrightarrow F\ g\ \{x \in A.\ P\ x\} = F\ (\lambda x.\ \textit{if}\ P\ x\ \textit{then}\ g\ x\ \textit{else}\ \mathbf{1})\ A$
by (*simp add*: *inter-restrict* [*symmetric*, *of A* {*x*. *P x*} *g*, *simplified mem-Collect-eq*] *Int-def*)

lemma *Union-comp*:
assumes $\forall\, A \in B.$ *finite A*
and $\bigwedge A1\ A2\ x.\ A1 \in B \Longrightarrow A2 \in B \Longrightarrow A1 \neq A2 \Longrightarrow x \in A1 \Longrightarrow x \in A2 \Longrightarrow g\ x = \mathbf{1}$
shows $F\ g\ (\bigcup B) = (F \circ F)\ g\ B$
using *assms*
proof (*induct B rule*: *infinite-finite-induct*)
case (*infinite A*)
then have ¬ *finite* $(\bigcup A)$ **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* $(\bigcup B)$ $A \notin B$
and $\forall\, x{\in}A \cap \bigcup B.\ g\ x = \mathbf{1}$
and *H*: $F\ g\ (\bigcup B) = (F \circ F)\ g\ B$ **by** *auto*
then have $F\ g\ (A \cup \bigcup B) = F\ g\ A * F\ g\ (\bigcup B)$
by (*simp add*: *union-inter-neutral*)
with ‹*finite B*› ‹$A \notin B$› **show** *?case*
by (*simp add*: *H*)
qed

lemma *commute*: $F\ (\lambda i.\ F\ (g\ i)\ B)\ A = F\ (\lambda j.\ F\ (\lambda i.\ g\ i\ j)\ A)\ B$
unfolding *cartesian-product*
by (*rule reindex-bij-witness* [**where** $i = \lambda(i, j).\ (j, i)$ **and** $j = \lambda(i, j).\ (j, i)$]) *auto*

lemma *commute-restrict*:
finite $A \Longrightarrow$ *finite* $B \Longrightarrow$
$F\ (\lambda x.\ F\ (g\ x)\ \{y.\ y \in B \wedge R\ x\ y\})\ A = F\ (\lambda y.\ F\ (\lambda x.\ g\ x\ y)\ \{x.\ x \in A \wedge R\ x\ y\})\ B$

by (*simp add*: *inter-filter*) (*rule commute*)

lemma *Plus*:
fixes $A :: 'b\ set$ **and** $B :: 'c\ set$
assumes *fin*: *finite A finite B*
shows $F\ g\ (A <+> B) = F\ (g \circ Inl)\ A * F\ (g \circ Inr)\ B$
proof −
have $A <+> B = Inl\ `\ A \cup Inr\ `\ B$ **by** *auto*
moreover from *fin* **have** *finite* $(Inl\ `\ A)$ *finite* $(Inr\ `\ B)$ **by** *auto*
moreover have $Inl\ `\ A \cap Inr\ `\ B = \{\}$ **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*› ‹*finite* $(C - A)$› **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*› ‹*finite* $(C - B)$› **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*: $\bigwedge a.\ a \in C - A \Longrightarrow g\ a = \mathbf{1}\ \bigwedge b.\ b \in C - B \Longrightarrow h\ b = \mathbf{1}$
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

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* ($\sum$ - [*1000*] *999*)
  where $\sum A \equiv$ *sum-0* ($\lambda x.\ x$) *A*

end

context *comm-monoid-add*
begin

subclass *ab-semigroup-add-0*
  by *unfold-locales simp-all*

end

Now: lot's of fancy syntax. First, *sum* ($\lambda x.\ e$) *A* is written $\sum x{\in}A.\ e$.

syntax (*ASCII*)
  -sum :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*::*comm-monoid-add* ((*3SUM* -:-./ -) [*0*, *51*, *10*] *10*)
syntax
  -sum :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*::*comm-monoid-add* ((*2*$\sum$ -∈-./ -) [*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 $\sum x|P.\ e$.

syntax (*ASCII*)
  -qsum :: *pttrn* ⇒ *bool* ⇒ *'a* ⇒ *'a* ((*3SUM* - |/ -./ -) [*0*, *0*, *10*] *10*)
syntax
  -qsum :: *pttrn* ⇒ *bool* ⇒ *'a* ⇒ *'a* ((*2*$\sum$ - | (-)./ -) [*0*, *0*, *10*] *10*)
translations
  $\sum x|P.\ t$ => *CONST sum-0* ($\lambda x.\ t$) {$x.\ P$}

print-translation ‹
let
  fun sum-tr' [*Abs* (*x*, *Tx*, *t*), *Const* (@{*const-syntax Collect*}, -) $ *Abs* (*y*, *Ty*, *P*)] =
    if x <> y then raise Match
    else
      let
        val x' = Syntax-Trans.mark-bound-body (*x*, *Tx*);

```
      val t' = subst-bound (x', t);
      val P' = subst-bound (x', P);
    in
     Syntax.const @{syntax-const -qsum} $ Syntax-Trans.mark-bound-abs (x,
Tx) $ P' $ t'
    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* (λ*y*. *sum-0 g* {*x*. *x* ∈ *S* ∧ *f x* = *y*}) (*f* ' *S*)
proof −
have {*y*. *y*∈ *f*'*S* ∧ *f x* = *y*} = {*f x*} **if** *x* ∈ *S* **for** *x*
using *that* **by** *auto*
then have *sum-0 g S* = *sum-0* (λ*x*. *sum-0* (λ*y*. *g x*) {*y*. *y*∈ *f*'*S* ∧ *f x* = *y*}) *S*
by *simp*
also have ... = *sum-0* (λ*y*. *sum-0 g* {*x*. *x* ∈ *S* ∧ *f x* = *y*}) (*f* ' *S*)
by (*rule sum-0.commute-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* ∪ *B*)
shows *sum-0 f* (*A* ∪ *B*) = *sum-0 f* (*A* − *B*) + *sum-0 f* (*B* − *A*) + *sum-0 f* (*A* ∩ *B*)
proof −
have *A* ∪ *B* = *A* − *B* ∪ (*B* − *A*) ∪ *A* ∩ *B*
by *auto*
with *assms* **show** *?thesis*
by *simp* (*subst sum-0.union-disjoint*, *auto*)+
qed

class *ordered-ab-semigroup-add-0* = *ab-semigroup-add-0* +
ordered-ab-semigroup-add
begin

lemma *add-nonneg-nonneg* [*simp*]: *0* ≤ *a* ⟹ *0* ≤ *b* ⟹ *0* ≤ *a* + *b*
using *add-mono*[*of 0 a 0 b*] **by** *simp*

lemma *add-nonpos-nonpos*: *a* ≤ *0* ⟹ *b* ≤ *0* ⟹ *a* + *b* ≤ *0*
using *add-mono*[*of a 0 b 0*] **by** *simp*

lemmas *add-sign-intros* =
add-pos-nonneg add-pos-pos add-nonneg-pos add-nonneg-nonneg
add-neg-nonpos add-neg-neg add-nonpos-neg add-nonpos-nonpos

end

class *strict-ordered-ab-semigroup-add-0* = *ab-semigroup-add-0* +
strict-ordered-ab-semigroup-add
begin

end

lemma (**in** *ordered-ab-semigroup-add-0*) *sum-mono*:
$(\bigwedge i.\ i{\in}K \Longrightarrow f\ i \leq g\ i) \Longrightarrow (\sum i{\in}K.\ f\ i) \leq (\sum i{\in}K.\ g\ i)$
by (*induct K rule*: *infinite-finite-induct*) (*use add-mono* **in** *auto*)

lemma (**in** *ordered-ab-semigroup-add-0*) *sum-mono-00*:
$(\bigwedge i.\ i{\in}K \Longrightarrow f\ i + 0 \leq g\ i + 0) \Longrightarrow (\sum i{\in}K.\ f\ i) \leq (\sum i{\in}K.\ g\ i)$
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*: $x \notin F$
assume *a3*: $(\bigwedge i.\ i \in F \Longrightarrow f\ i + 0 \leq g\ i + 0) \Longrightarrow sum\text{-}0\ f\ F \leq sum\text{-}0\ g\ F$
assume *a4*: $\bigwedge i.\ i \in insert\ x\ F \Longrightarrow f\ i + 0 \leq g\ i + 0$
obtain $bb :: 'b$ **where**
f5: $bb \in F \wedge \neg\ f\ bb + 0 \leq g\ bb + 0 \vee sum\text{-}0\ f\ F \leq sum\text{-}0\ g\ F$
using *a3* **by** *blast*
have $\forall b.\ x \neq b \vee f\ b + 0 \leq g\ b + 0$
using *a4* **by** *simp*
then have $\forall a\ aa.\ f\ x + 0 + a \leq g\ x + 0 + aa \vee \neg\ a \leq aa$
using *add-mono* **by** *blast*
then show *sum-0 f* (*insert x F*) ≤ *sum-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*:
$(\bigwedge i.\ i{\in}K \Longrightarrow f\ i + 0 \leq g\ i) \Longrightarrow (\sum i{\in}K.\ f\ i) \leq (\sum i{\in}K.\ g\ i)$
apply (*rule sum-mono-00*)
by (*metis add-right-mono zero-neutral*)

context *ordered-ab-semigroup-add-0*
begin

lemma *sum-nonneg*: $\forall x{\in}A.\ 0 \leq f\ x \Longrightarrow 0 \leq$ *sum-0 f 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 $0 + 0 \leq f\ x +$ *sum-0 f F* **by** (*blast intro*: *add-mono*)
with *insert* **show** *?case* **by** *simp*
qed

lemma *sum-nonpos*: $\forall x{\in}A.\ f\ x \leq 0 \Longrightarrow$ *sum-0 f A* ≤ 0
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* $\leq 0 + 0$ **by** (*blast intro*: *add-mono*)
with *insert* **show** *?case* **by** *simp*
qed

lemma *sum-mono2*:
assumes *fin*: *finite B*
and *sub*: $A \subseteq B$
and *nn*: $\bigwedge b.\ b \in B{-}A \Longrightarrow 0 \leq f\ b$
shows *sum-0 f A* $\leq$ *sum-0 f B*
proof −
have *sum-0 f A* $\leq$ *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* ∪ (*B*−*A*))
by (*simp add*: *sum-0.union-disjoint del*: *Un-Diff-cancel*)
also from *sub* **have** *A* ∪ (*B*−*A*) = *B* **by** *blast*
finally show *?thesis* **.**
qed

lemma *sum-le-included*:
assumes *finite s finite t*
and ∀ *y*∈*t*. *0* ≤ *g y* (∀ *x*∈*s*. ∃ *y*∈*t*. *i y* = *x* ∧ *f x* ≤ *g y*)
shows *sum-0 f s* ≤ *sum-0 g t*
proof −
have *sum-0 f s* ≤ *sum-0* (λ*y*. *sum-0 g* {*x*. *x*∈*t* ∧ *i x* = *y*}) *s*
proof (*rule sum-mono-0*)
fix *y*
assume *y* ∈ *s*
with *assms* **obtain** *z* **where** *z*: *z* ∈ *t y* = *i z f y* ≤ *g z* **by** *auto*
hence *f y* + *0* ≤ *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 ... ≤ *sum-0 g* {*x* ∈ *t*. *i x* = *y*}
apply (*rule sum-mono2*)
using *assms z* **by** *simp-all*
finally show *f y* + *0* ≤ *sum-0 g* {*x* ∈ *t*. *i x* = *y*} **.**
qed
also have ... ≤ *sum-0* (λ*y*. *sum-0 g* {*x*. *x*∈*t* ∧ *i x* = *y*}) (*i* ‘ *t*)
using *assms*(*2*−*4*) **by** (*auto intro*!: *sum-mono2 sum-nonneg*)
also have ... ≤ *sum-0 g t*
using *assms* **by** (*auto simp*: *sum-image-gen-0*[*symmetric*])
finally show *?thesis* **.**
qed

lemma *sum-mono3*: *finite B* ⟹ *A* ⊆ *B* ⟹ ∀ *x*∈*B* − *A*. *0* ≤ *f x* ⟹ *sum-0 f A* ≤ *sum-0 f B*
by (*rule sum-mono2*) *auto*

end

lemma *sum-comp-morphism*:
h 0 = *0* ⟹ (⋀*x y*. *h* (*x* + *y*) = *h x* + *h y*) ⟹ *sum-0* (*h* ∘ *g*) *A* = *h* (*sum-0 g A*)
by (*induct A rule*: *infinite-finite-induct*) *simp-all*

lemma *sum-cong-Suc*:
assumes *0* ∉ *A* ⋀*x*. *Suc x* ∈ *A* ⟹ *f* (*Suc x*) = *g* (*Suc x*)

shows *sum-0 f A = sum-0 g A*
proof (*rule sum-0.cong*)
fix *x*
assume *x ∈ A*
with *assms(1)* **show** *f x = g x*
by (*cases x*) (*auto intro*!: *assms(2)*)
qed *simp-all*

end

3 Algebras for Aggregation and Minimisation

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

class *sum* =
fixes *sum* :: *'a ⇒ 'a*

class *s-algebra* = *stone-relation-algebra* + *plus* + *sum* +
assumes *sum-plus-right-isotone*: *x ≠ bot ∧ sum x ≤ sum y ⟶ z + sum x ≤ z + sum y*
assumes *sum-bot*: *sum x + sum bot = sum x*
assumes *sum-plus*: *sum x + sum y = sum (x ⊔ y) + sum (x ⊓ y)*
assumes *sum-conv*: *sum (x^T) = sum x*
begin

lemma *sum-disjoint*:

assumes $x \sqcap y = bot$
shows $sum\ ((x \sqcup y) \sqcap z) = sum\ (x \sqcap z) + sum\ (y \sqcap z)$
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 $w \sqcap x = bot$
and $w \sqcap y = bot$
and $x \sqcap y = bot$
shows $sum\ ((w \sqcup x \sqcup y) \sqcap z) = sum\ (w \sqcap z) + sum\ (x \sqcap z) + sum\ (y \sqcap z)$
by (*metis assms inf-sup-distrib2 sup-idem sum-disjoint*)

lemma *sum-symmetric*:
assumes $y = y^T$
shows $sum\ (x^T \sqcap y) = sum\ (x \sqcap y)$
by (*metis assms sum-conv conv-dist-inf*)

end

class *minatom* =
fixes *minatom* :: $'a \Rightarrow 'a$

class *m-algebra* = *s-algebra* + *minatom* +
assumes *minatom-below*: $minatom\ x \leq --x$
assumes *minatom-atom*: $x \neq bot \longrightarrow atom\ (minatom\ x)$
assumes *minatom-min*: $atom\ y \wedge y \sqcap x \neq bot \longrightarrow sum\ (minatom\ x \sqcap x) \leq sum\ (y \sqcap x)$
assumes *sum-linear*: $sum\ x \leq sum\ y \vee sum\ y \leq sum\ x$
assumes *finite-regular*: $finite\ \{\ x\ .\ regular\ x\ \}$
begin

subclass *stone-relation-algebra-tarski*
proof *unfold-locales*
fix x
let $?a = minatom\ x$
assume *1*: $regular\ x$
assume $x \neq bot$
hence $atom\ ?a$
by (*simp add*: *minatom-atom*)
hence $top = top * ?a * top$
by (*simp add*: *comp-associative*)
also have $... \leq top * --x * top$
by (*simp add*: *minatom-below mult-isotone*)
finally show $top * x * top = top$
using *1 antisym* **by** *simp*
qed

lemma *minatom-bot*:
$minatom\ bot = bot$

by (*metis bot-unique minatom-below regular-closed-bot*)

lemma *minatom-bot-iff*:
minatom x = bot ⟷ x = bot
using *covector-bot-closed inf-bot-right minatom-atom vector-bot-closed minatom-bot* **by** *fastforce*

lemma *minatom-meet-bot*:
assumes *minatom x ⊓ x = bot*
shows *minatom x = bot*
proof −
have *minatom x ≤ −x*
using *assms pseudo-complement* **by** *auto*
thus *?thesis*
by (*metis minatom-below inf-absorb1 inf-import-p inf-p*)
qed

lemma *minatom-meet-bot-minatom-iff*:
minatom x ⊓ x = bot ⟷ minatom x = bot
using *comp-inf.semiring.mult-not-zero minatom-meet-bot* **by** *blast*

lemma *minatom-meet-bot-iff*:
minatom x ⊓ x = bot ⟷ x = bot
using *inf-bot-right minatom-bot-iff minatom-meet-bot* **by** *blast*

lemma *minatom-regular*:
regular (*minatom x*)
proof (*cases x = bot*)
assume *x = bot*
thus *?thesis*
by (*simp add*: *minatom-bot*)
next
assume *x ≠ bot*
thus *?thesis*
by (*simp add*: *atom-regular minatom-atom*)
qed

lemma *minatom-selection*:
selection (*minatom x ⊓ y*) *y*
using *inf-assoc minatom-regular selection-closed-id* **by** *auto*

end

class *m-kleene-algebra = m-algebra + stone-kleene-relation-algebra*

end

4 Matrix Algebras for Aggregation and Minimisation

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*)

class *aggregation-order* = *order-bot* + *ab-semigroup-add* +
assumes *add-right-isotone*: $x \neq bot \wedge x + bot \leq y + bot \longrightarrow x + z \leq y + z$
assumes *add-add-bot* [*simp*]: $x + y + bot = x + y$
assumes *add-bot*: $x + y = bot \longrightarrow x = bot$
begin

abbreviation *zero* ≡ *bot* + *bot*

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*:
$x + bot + bot + bot = x + bot$
by *simp*

end

syntax (*xsymbols*)
-sum-ab-semigroup-add-0 :: *idt* ⇒ *'a*::*bounded-semilattice-sup-bot* ⇒ *'a* ((∑- -) [*0*,*10*] *10*)

translations
$\sum_x\ t$ => *XCONST ab-semigroup-add-0.sum-0 XCONST plus* (*XCONST plus XCONST bot XCONST bot*) (λ*x* . *t*) { *x* . *CONST True* }

lemma *agg-sum-bot*:
$(\sum_k\ bot::'a::aggregation\text{-}order) = bot + bot$
proof (*induct rule*: *infinite-finite-induct*)
case (*infinite A*)
thus *?case*
by *simp*
next
case *empty*
thus *?case*

by *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*:
($\sum_k$ *bot* + *bot*::'*a*::*aggregation-order*) = *bot* + *bot*
by (*rule aggregation.sum-0.neutral-const*)

lemma *agg-sum-not-bot-1*:
fixes *f* :: '*a*::*finite* ⇒ '*b*::*aggregation-order*
assumes *f i* ≠ *bot*
shows ($\sum_k$ *f k*) ≠ *bot*
by (*metis assms add-bot aggregation.sum-0.remove finite-code mem-Collect-eq*)

lemma *agg-sum-not-bot*:
fixes *f* :: ('*a*::*finite*,'*b*::*aggregation-order*) *square*
assumes *f* (*i*,*j*) ≠ *bot*
shows ($\sum_k$ $\sum_l$ *f* (*k*,*l*)) ≠ *bot*
by (*metis assms agg-sum-not-bot-1*)

lemma *agg-sum-distrib*:
fixes *f g* :: '*a* ⇒ '*b*::*aggregation-order*
shows ($\sum_k$ *f k* + *g k*) = ($\sum_k$ *f k*) + ($\sum_k$ *g k*)
by (*rule aggregation.sum-0.distrib*)

lemma *agg-sum-distrib-2*:
fixes *f g* :: ('*a*,'*b*::*aggregation-order*) *square*
shows ($\sum_k$ $\sum_l$ *f* (*k*,*l*) + *g* (*k*,*l*)) = ($\sum_k$ $\sum_l$ *f* (*k*,*l*)) + ($\sum_k$ $\sum_l$ *g* (*k*,*l*))
proof −
have ($\sum_k$ $\sum_l$ *f* (*k*,*l*) + *g* (*k*,*l*)) = ($\sum_k$ ($\sum_l$ *f* (*k*,*l*)) + ($\sum_l$ *g* (*k*,*l*)))
by (*metis* (*no-types*) *aggregation.sum-0.distrib*)
also have ... = ($\sum_k$ $\sum_l$ *f* (*k*,*l*)) + ($\sum_k$ $\sum_l$ *g* (*k*,*l*))
by (*metis* (*no-types*) *aggregation.sum-0.distrib*)
finally show *?thesis*
.
qed

lemma *agg-sum-commute*:
fixes *f* :: ('*a*,'*b*::*aggregation-order*) *square*
shows ($\sum_k$ $\sum_l$ *f* (*k*,*l*)) = ($\sum_l$ $\sum_k$ *f* (*k*,*l*))
by (*rule aggregation.sum-0.commute*)

lemma *agg-delta*:
fixes *f* :: '*a*::*finite* ⇒ '*b*::*aggregation-order*
shows ($\sum_l$ *if l* = *j then f l else zero*) = *f j* + *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 $f :: 'a::finite \Rightarrow 'b::aggregation\text{-}order$
shows $(\sum_l \textit{if } l = j \textit{ then } f\ l \textit{ else } bot) = f\ j + bot$
proof −
let *?f* = $(\lambda l\ .\ \textit{if } l = j \textit{ then } f\ l \textit{ else } bot)$
let *?S* = $\{l::'a\ .\ True\}$
show *?thesis*
proof (*cases* $j \in$ *?S*)
case *False*
thus *?thesis* **by** *simp*
next
case *True*
let *?A* = *?S* − $\{j\}$
let *?B* = $\{j\}$
from *True* **have** *eq*: *?S* = *?A* ∪ *?B*
by *blast*
have *dj*: *?A* ∩ *?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* $(\lambda l\ .\ bot)$ *?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* + (*f j* + *zero*)
by *simp*
also have ... = *f j* + *bot*
by (*metis add.commute add.left-commute add-add-bot*)
finally show *?thesis*
.
qed
qed

lemma *agg-delta-2*:
fixes $f :: ('a::finite,'b::aggregation\text{-}order)\ square$
shows $(\sum_k \sum_l \textit{if } k = i \wedge l = j \textit{ then } f\ (k,l) \textit{ else } bot) = f\ (i,j) + bot$
proof −
have $\forall k\ .\ (\sum_l \textit{if } k = i \wedge l = j \textit{ then } f\ (k,l) \textit{ else } bot) = (\textit{if } k = i \textit{ then } f\ (k,j) + bot \textit{ else } zero)$
proof
fix *k*

have ($\sum_l$ *if* $k = i \wedge l = j$ *then* f (k,l) *else bot*) = ($\sum_l$ *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 ($\sum_l$ *if* $k = i \wedge l = j$ *then* f (k,l) *else bot*) = (*if* $k = i$ *then* f (k,j) + *bot else zero*)
by *simp*
qed
hence ($\sum_k \sum_l$ *if* $k = i \wedge l = j$ *then* f (k,l) *else bot*) = ($\sum_k$ *if* $k = i$ *then* f (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

definition *zero-matrix* :: ('*a*,'*b*::{*plus*,*bot*}) *square* (*mzero*) **where** *mzero* = (λe . *bot* + *bot*)

definition *plus-matrix* :: ('*a*,'*b*::*plus*) *square* ⇒ ('*a*,'*b*) *square* ⇒ ('*a*,'*b*) *square* (**infixl** $\oplus_M$ *65*) **where** *plus-matrix f g* = (λe . *f e* + *g e*)

definition *sum-matrix* :: ('*a*::*enum*,'*b*::{*plus*,*bot*}) *square* ⇒ ('*a*,'*b*) *square* (sum_M - [*80*] *80*) **where** *sum-matrix f* = ($\lambda(i,j)$. *if i* = *hd enum-class.enum* ∧ *j* = *i then* $\sum_k \sum_l$ *f* (k,l) *else bot* + *bot*)

lemma *bot-plus-bot*:
mbot $\oplus_M$ *mbot* = *mzero*
by (*simp add*: *plus-matrix-def bot-matrix-def zero-matrix-def*)

lemma *sum-bot*:
sum_M (*mbot* :: ('*a*::*enum*,'*b*::*aggregation-order*) *square*) = *mzero*
proof (*rule ext*, *rule prod-cases*)
fix *i j* :: '*a*
have (sum_M *mbot* :: ('*a*,'*b*) *square*) (*i*,*j*) = (*if i* = *hd enum-class.enum* ∧ *j* = *i 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 *mbot* :: ('*a*,'*b*) *square*) (*i*,*j*) = *mzero* (*i*,*j*)
.
qed

lemma *sum-plus-bot*:

fixes f :: ($'a$::*enum*,$'b$::*aggregation-order*) *square*
shows $sum_M\ f\ \oplus_M\ mbot = sum_M\ f$
proof (*rule ext*, *rule prod-cases*)
let *?h* = *hd enum-class.enum*
fix *i j*
have ($sum_M\ f\ \oplus_M\ mbot$) (*i*,*j*) = (*if i* = *?h* ∧ *j* = *i then* ($\sum_k \sum_l f\ (k,l)$) + *bot else zero* + *bot*)
by (*simp add*: *plus-matrix-def bot-matrix-def sum-matrix-def*)
also have ... = (*if i* = *?h* ∧ *j* = *i then* $\sum_k \sum_l f\ (k,l)$ *else zero*)
by (*metis* (*no-types*, *lifting*) *add-add-bot aggregation.sum-0.F-one*)
also have ... = ($sum_M\ f$) (*i*,*j*)
by (*simp add*: *sum-matrix-def*)
finally show ($sum_M\ f\ \oplus_M\ mbot$) (*i*,*j*) = ($sum_M\ f$) (*i*,*j*)
by *simp*
qed

lemma *sum-plus-zero*:
fixes f :: ($'a$::*enum*,$'b$::*aggregation-order*) *square*
shows $sum_M\ f\ \oplus_M\ mzero = sum_M\ f$
by (*rule ext*, *rule prod-cases*) (*simp add*: *plus-matrix-def zero-matrix-def sum-matrix-def*)

lemma *agg-matrix-bot*:
fixes f :: ($'a$,$'b$::*aggregation-order*) *square*
assumes ∀ *i j* . *f* (*i*,*j*) = *bot*
shows $f = mbot$
apply (*unfold bot-matrix-def*)
using *assms* **by** *auto*

definition *sum-matrix-2* :: ($'a$,$'b$::{*plus*,*bot*}) *square* ⇒ ($'a$,$'b$) *square* ($sum2_M$ - [*80*] *80*) **where** *sum-matrix-2 f* = (λ*e* . $\sum_k \sum_l f\ (k,l)$)

lemma *sum-bot-2*:
$sum2_M$ (*mbot* :: ($'a$,$'b$::*aggregation-order*) *square*) = *mzero*
proof
fix *e*
have ($sum2_M\ mbot$ :: ($'a$,$'b$) *square*) *e* = ($\sum_{(}k$::$'a$) $\sum_{(}l$::$'a$) *bot*)
by (*unfold sum-matrix-2-def bot-matrix-def*) *simp*
also have ... = *bot* + *bot*
using *agg-sum-bot aggregation.sum-0.neutral* **by** *fastforce*
also have ... = *mzero e*
by (*simp add*: *zero-matrix-def*)
finally show ($sum2_M\ mbot$ :: ($'a$,$'b$) *square*) *e* = *mzero e*
.
qed

lemma *sum-plus-bot-2*:
fixes f :: ($'a$,$'b$::*aggregation-order*) *square*
shows $sum2_M\ f\ \oplus_M\ mbot = sum2_M\ f$

proof
fix *e*
have (*sum2*$_M$ *f* $\oplus_M$ *mbot*) *e* = ($\sum_k \sum_l$ *f* (*k*,*l*)) + *bot*
by (*simp add*: *plus-matrix-def bot-matrix-def sum-matrix-2-def*)
also have ... = ($\sum_k \sum_l$ *f* (*k*,*l*))
by (*metis* (*no-types*, *lifting*) *add-add-bot aggregation.sum-0.F-one*)
also have ... = (*sum2*$_M$ *f*) *e*
by (*simp add*: *sum-matrix-2-def*)
finally show (*sum2*$_M$ *f* $\oplus_M$ *mbot*) *e* = (*sum2*$_M$ *f*) *e*
by *simp*
qed

lemma *sum-plus-zero-2*:
fixes *f* :: ('*a*,'*b*::*aggregation-order*) *square*
shows *sum2*$_M$ *f* $\oplus_M$ *mzero* = *sum2*$_M$ *f*
by (*simp add*: *plus-matrix-def zero-matrix-def sum-matrix-2-def*)

class *aggregation-lattice* = *bounded-distrib-lattice* + *dense-lattice* + *aggregation-order* +
assumes *add-lattice*: *x* + *y* = (*x* ⊔ *y*) + (*x* ⊓ *y*)

We show that matrices over linear bounded lattices with aggregation form an m-algebra.

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* ⊓ *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

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 $f\ g\ h :: ('a,'b)\ square$
show $f \neq mbot \wedge sum_M\ f \preceq sum_M\ g \longrightarrow h \oplus_M sum_M\ f \preceq h \oplus_M sum_M\ g$
proof
let *?h = hd enum-class.enum*
assume *1*: $f \neq mbot \wedge sum_M\ f \preceq sum_M\ g$
hence $\exists k\ l\ .\ f\ (k,l) \neq bot$
by (*meson agg-matrix-bot*)
hence *2*: $(\sum_k \sum_l f\ (k,l)) \neq bot$
using *agg-sum-not-bot* **by** *blast*
have $(\sum_k \sum_l f\ (k,l)) = (sum_M\ f)\ (?h,?h)$
by (*simp add*: *sum-matrix-def*)
also have ... $\leq (sum_M\ g)\ (?h,?h)$
using *1* **by** (*simp add*: *less-eq-matrix-def*)
also have ... $= (\sum_k \sum_l g\ (k,l))$
by (*simp add*: *sum-matrix-def*)
finally have $(\sum_k \sum_l f\ (k,l)) \leq (\sum_k \sum_l g\ (k,l))$
by *simp*
hence *3*: $(\sum_k \sum_l f\ (k,l)) + bot \leq (\sum_k \sum_l g\ (k,l)) + bot$
by (*metis* (*no-types*, *lifting*) *add-add-bot aggregation.sum-0.F-one*)
show $h \oplus_M sum_M\ f \preceq h \oplus_M sum_M\ g$
proof (*unfold less-eq-matrix-def*, *rule allI*, *rule prod-cases*, *unfold plus-matrix-def*)
fix *i j*
have *4*: $h\ (i,j) + (\sum_k \sum_l f\ (k,l)) \leq h\ (i,j) + (\sum_k \sum_l g\ (k,l))$
using *2 3* **by** (*metis* (*no-types*, *lifting*) *add-right-isotone add.commute*)
have $h\ (i,j) + (sum_M\ f)\ (i,j) = h\ (i,j) + (if\ i = ?h \wedge j = i\ then\ \sum_k \sum_l f\ (k,l)\ else\ zero)$
by (*simp add*: *sum-matrix-def*)
also have ... $= (if\ i = ?h \wedge j = i\ then\ h\ (i,j) + (\sum_k \sum_l f\ (k,l))\ else\ h\ (i,j) + zero)$
by *simp*
also have ... $\leq (if\ i = ?h \wedge j = i\ then\ h\ (i,j) + (\sum_k \sum_l g\ (k,l))\ else\ h\ (i,j) + zero)$
using *4 inf.eq-iff* **by** *auto*
also have ... $= h\ (i,j) + (if\ i = ?h \wedge j = i\ then\ \sum_k \sum_l g\ (k,l)\ else\ zero)$
by *simp*
finally show $h\ (i,j) + (sum_M\ f)\ (i,j) \leq h\ (i,j) + (sum_M\ g)\ (i,j)$
by (*simp add*: *sum-matrix-def*)
qed
qed
next
fix $f :: ('a,'b)\ square$
show $sum_M\ f \oplus_M sum_M\ mbot = sum_M\ f$
by (*simp add*: *sum-bot sum-plus-zero*)
next
fix $f\ g :: ('a,'b)\ square$
show $sum_M\ f \oplus_M sum_M\ g = sum_M\ (f \oplus g) \oplus_M sum_M\ (f \otimes g)$
proof (*rule ext*, *rule prod-cases*)
fix *i j*

let *?h = hd enum-class.enum*
have $(sum_M\ f \oplus_M sum_M\ g)\ (i,j) = (sum_M\ f)\ (i,j) + (sum_M\ g)\ (i,j)$
by (*simp add*: *plus-matrix-def*)
also have ... = (*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 ... = (*if* $i = ?h \wedge j = i$ *then* $(\sum_k \sum_l f\ (k,l)) + (\sum_k \sum_l g\ (k,l))$ *else zero*)
by *simp*
also have ... = (*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 ... = (*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 ... = (*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 ... = (*if* $i = ?h \wedge j = i$ *then* $(\sum_k \sum_l (f \oplus g)\ (k,l)) + (\sum_k \sum_l (f \otimes g)\ (k,l))$ *else zero*)
using *agg-sum-distrib-2* **by** (*metis* (*no-types*))
also have ... = (*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 ... = $(sum_M\ (f \oplus g))\ (i,j) + (sum_M\ (f \otimes g))\ (i,j)$
by (*simp add*: *sum-matrix-def*)
also have ... = $(sum_M\ (f \oplus g) \oplus_M sum_M\ (f \otimes g))\ (i,j)$
by (*simp add*: *plus-matrix-def*)
finally show $(sum_M\ f \oplus_M sum_M\ g)\ (i,j) = (sum_M\ (f \oplus g) \oplus_M sum_M\ (f \otimes g))\ (i,j)$
.
qed
next
fix $f :: ('a,'b)\ square$
show $sum_M\ (f^t) = sum_M\ f$
proof (*rule ext*, *rule prod-cases*)
fix *i j*
let *?h = hd enum-class.enum*
have $(sum_M\ (f^t))\ (i,j) = (if\ i = ?h \wedge j = i\ then \sum_k \sum_l (f^t)\ (k,l)\ else\ zero)$
by (*simp add*: *sum-matrix-def*)
also have ... = (*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 ... = (*if* $i = ?h \wedge j = i$ *then* $\sum_k \sum_l f\ (l,k)$ *else zero*)
by *simp*
also have ... = (*if* $i = ?h \wedge j = i$ *then* $\sum_l \sum_k f\ (l,k)$ *else zero*)
by (*metis agg-sum-commute*)
also have ... = $(sum_M\ f)\ (i,j)$
by (*simp add*: *sum-matrix-def*)
finally show $(sum_M\ (f^t))\ (i,j) = (sum_M\ f)\ (i,j)$
.

qed
qed

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 *f g h* :: ('*a*,'*b*) *square*
show $f \neq mbot \wedge sum2_M\ f \preceq sum2_M\ g \longrightarrow h \oplus_M sum2_M\ f \preceq h \oplus_M sum2_M\ g$
proof
assume *1*: $f \neq mbot \wedge sum2_M\ f \preceq sum2_M\ g$
hence $\exists k\ l\ .\ f\ (k,l) \neq bot$
by (*meson agg-matrix-bot*)
hence *2*: $(\sum_k \sum_l f\ (k,l)) \neq bot$
using *agg-sum-not-bot* **by** *blast*
obtain *c* :: '*a* **where** *True*
by *simp*
have $(\sum_k \sum_l f\ (k,l)) = (sum2_M\ f)\ (c,c)$
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 ... $= (\sum_k \sum_l g\ (k,l))$
by (*simp add*: *sum-matrix-2-def*)
finally have $(\sum_k \sum_l f\ (k,l)) \leq (\sum_k \sum_l g\ (k,l))$
by *simp*
hence *3*: $(\sum_k \sum_l f\ (k,l)) + bot \leq (\sum_k \sum_l g\ (k,l)) + bot$
by (*metis* (*no-types*, *lifting*) *add-add-bot aggregation.sum-0.F-one*)
show $h \oplus_M sum2_M\ f \preceq h \oplus_M sum2_M\ g$
proof (*unfold less-eq-matrix-def*, *rule allI*, *unfold plus-matrix-def*)
fix *e*
have $h\ e + (sum2_M\ f)\ e = h\ e + (\sum_k \sum_l f\ (k,l))$
by (*simp add*: *sum-matrix-2-def*)
also have ... $\leq h\ e + (\sum_k \sum_l g\ (k,l))$
using *2 3* **by** (*metis* (*no-types*, *lifting*) *add-right-isotone add.commute*)
finally show $h\ e + (sum2_M\ f)\ e \leq h\ e + (sum2_M\ g)\ e$
by (*simp add*: *sum-matrix-2-def*)
qed
qed
next
fix *f* :: ('*a*,'*b*) *square*
show $sum2_M\ f \oplus_M sum2_M\ mbot = sum2_M\ f$
by (*simp add*: *sum-bot-2 sum-plus-zero-2*)
next
fix *f g* :: ('*a*,'*b*) *square*
show $sum2_M\ f \oplus_M sum2_M\ g = sum2_M\ (f \oplus g) \oplus_M sum2_M\ (f \otimes g)$
proof

```
    fix e
    have (sum2_M f ⊕_M sum2_M g) e = (sum2_M f) e + (sum2_M g) e
      by (simp add: plus-matrix-def)
    also have ... = (∑_k ∑_l f (k,l)) + (∑_k ∑_l g (k,l))
      by (simp add: sum-matrix-2-def)
    also have ... = (∑_k ∑_l f (k,l) + g (k,l))
      using agg-sum-distrib-2 by (metis (no-types))
    also have ... = (∑_k ∑_l (f (k,l) ⊔ g (k,l)) + (f (k,l) ⊓ g (k,l)))
      using add-lattice aggregation.sum-0.cong by (metis (no-types, lifting))
    also have ... = (∑_k ∑_l (f ⊕ g) (k,l) + (f ⊗ g) (k,l))
      by (simp add: sup-matrix-def inf-matrix-def)
    also have ... = (∑_k ∑_l (f ⊕ g) (k,l)) + (∑_k ∑_l (f ⊗ g) (k,l))
      using agg-sum-distrib-2 by (metis (no-types))
    also have ... = (sum2_M (f ⊕ g)) e + (sum2_M (f ⊗ g)) e
      by (simp add: sum-matrix-2-def)
    also have ... = (sum2_M (f ⊕ g) ⊕_M sum2_M (f ⊗ g)) e
      by (simp add: plus-matrix-def)
    finally show (sum2_M f ⊕_M sum2_M g) e = (sum2_M (f ⊕ g) ⊕_M sum2_M (f
⊗ g)) e
      .
  qed
next
  fix f :: ('a,'b) square
  show sum2_M (f^t) = sum2_M f
  proof
    fix e
    have (sum2_M (f^t)) e = (∑_k ∑_l (f^t) (k,l))
      by (simp add: sum-matrix-2-def)
    also have ... = (∑_k ∑_l (f (l,k))^T)
      by (simp add: conv-matrix-def)
    also have ... = (∑_k ∑_l f (l,k))
      by simp
    also have ... = (∑_l ∑_k f (l,k))
      by (metis agg-sum-commute)
    also have ... = (sum2_M f) e
      by (simp add: sum-matrix-2-def)
    finally show (sum2_M (f^t)) e = (sum2_M f) e
      .
  qed
qed

primrec enum-pos' :: 'a list ⇒ 'a::enum ⇒ nat where
  enum-pos' Nil x = 0
| enum-pos' (y#ys) x = (if x = y then 0 else 1 + enum-pos' ys x)

lemma enum-pos'-inverse:
  List.member xs x ⟹ xs!(enum-pos' xs x) = x
  apply (induct xs)
  apply (simp add: member-rec(2))
```

by (*metis diff-add-inverse enum-pos'.simps*(*2*) *less-one member-rec*(*1*) *not-add-less1 nth-Cons'*)

fun *enum-pos* :: *'a*::*enum* ⇒ *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* ⟹ *x* = *y*
by (*metis enum-pos-inverse*)

abbreviation *enum-pos-less-eq* :: *'a*::*enum* ⇒ *'a* ⇒ *bool* **where** *enum-pos-less-eq x y* ≡ (*enum-pos x* ≤ *enum-pos y*)
abbreviation *enum-pos-less* :: *'a*::*enum* ⇒ *'a* ⇒ *bool* **where** *enum-pos-less x y* ≡ (*enum-pos x* < *enum-pos y*)

sublocale *enum* < *enum-order*: *order* **where** *less-eq* = λ*x y* . *enum-pos-less-eq x y* **and** *less* = λ*x y* . *enum-pos x* < *enum-pos y*
apply *unfold-locales*
by *auto*

abbreviation *enum-lex-less* :: *'a*::*enum* × *'a* ⇒ *'a* × *'a* ⇒ *bool* **where** *enum-lex-less* ≡ (λ(*i*,*j*) (*k*,*l*) . *enum-pos-less i k* ∨ (*i* = *k* ∧ *enum-pos-less j l*))
abbreviation *enum-lex-less-eq* :: *'a*::*enum* × *'a* ⇒ *'a* × *'a* ⇒ *bool* **where** *enum-lex-less-eq* ≡ (λ(*i*,*j*) (*k*,*l*) . *enum-pos-less i k* ∨ (*i* = *k* ∧ *enum-pos-less-eq j l*))

definition *minatom-matrix* :: (*'a*::*enum*,*'b*::{*bot*,*ord*,*plus*,*top*}) *square* ⇒ (*'a*,*'b*) *square* (*minatom*$_M$ - [*80*] *80*) **where** *minatom-matrix f* = (λ*e* . *if f e* ≠ *bot* ∧ (∀ *d* . (*f d* ≠ *bot* ⟶ (*f e* + *bot* ≤ *f d* + *bot* ∧ (*enum-lex-less d e* ⟶ *f e* + *bot* ≠ *f d* + *bot*)))) *then top else bot*)

lemma *minatom-at-most-one*:
fixes *f* :: (*'a*::*enum*,*'b*::{*aggregation-order*,*top*}) *square*
assumes (*minatom*$_M$ *f*) *e* ≠ *bot*
and (*minatom*$_M$ *f*) *d* ≠ *bot*
shows *e* = *d*
proof −
have *1*: *f e* + *bot* ≤ *f d* + *bot*
by (*metis assms minatom-matrix-def*)
have *f d* + *bot* ≤ *f e* + *bot*
by (*metis assms minatom-matrix-def*)
hence *f e* + *bot* = *f d* + *bot*
using *1* **by** *simp*

hence ¬ *enum-lex-less d e* ∧ ¬ *enum-lex-less e d*
using *assms* **by** (*unfold minatom-matrix-def*) (*metis* (*lifting*))
thus *?thesis*
using *enum-pos-injective less-linear* **by** *auto*
qed

class *linear-aggregation-lattice* = *linear-bounded-lattice* + *aggregation-order*
begin

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

end

lemma *least-order*:
assumes *transitive*: ∀ *x y z* . *le x y* ∧ *le y z* ⟶ *le x z*
and *total*: ∀ *x y* . *le x y* ∨ *le y x*
shows *finite A* ⟹ *A* ≠ {} ⟹ ∃ *x* . *x* ∈ *A* ∧ (∀ *y* . *y* ∈ *A* ⟶ *le 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 *minatom-at-least-one*:
fixes *f* :: (*'a*::*enum*,*'b*::*linear-aggregation-lattice*) *square*
assumes *f* ≠ *mbot*
shows ∃ *e* . (*minatom*$_M$ *f*) *e* = *top*
proof −
let *?nbe* = { (*e*,*f e*) | *e* . *f e* ≠ *bot* }
have *1*: *finite ?nbe*
using *finite-code finite-image-set* **by** *blast*
have *2*: *?nbe* ≠ {}
using *assms agg-matrix-bot* **by** *fastforce*
let *?le* = λ(*e*::*'a* × *'a*,*fe*::*'b*) (*d*::*'a* × *'a*,*fd*) . *fe* + *bot* ≤ *fd* + *bot*
have *3*: ∀ *x y z* . *?le x y* ∧ *?le y z* ⟶ *?le x z*
by *auto*
have *4*: ∀ *x y* . *?le x y* ∨ *?le y x*
by (*simp add*: *linear*)
have ∃ *x* . *x* ∈ *?nbe* ∧ (∀ *y* . *y* ∈ *?nbe* ⟶ *?le x y*)
by (*rule least-order*, *rule 3*, *rule 4*, *rule 1*, *rule 2*)
then obtain *e fe* **where** *5*: (*e*,*fe*) ∈ *?nbe* ∧ (∀ *y* . *y* ∈ *?nbe* ⟶ *?le* (*e*,*fe*) *y*)
by *auto*
let *?me* = { *e* . *f e* ≠ *bot* ∧ *f e* + *bot* = *fe* + *bot* }

have *6*: *finite ?me*
using *finite-code finite-image-set* **by** *blast*
have *7*: *?me* $\neq$ {}
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* $\in$ *?me* $\wedge$ ($\forall$ *y* . *y* $\in$ *?me* $\longrightarrow$ *enum-lex-less-eq x y*)
by (*rule least-order*, *rule 8*, *rule 9*, *rule 6*, *rule 7*)
then obtain *m* **where** *10*: *m* $\in$ *?me* $\wedge$ ($\forall$ *y* . *y* $\in$ *?me* $\longrightarrow$ *enum-lex-less-eq m y*)
by *auto*
have *11*: *f m* $\neq$ *bot*
using *10 5* **by** *auto*
have *12*: $\forall$ *d*. *f d* $\neq$ *bot* $\longrightarrow$ *f m* + *bot* $\leq$ *f d* + *bot*
using *10 5* **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 (*minatom*$_M$ *f*) *m* = *top*
using *11 12* **by** (*simp add*: *minatom-matrix-def*)
thus *?thesis*
by *blast*
qed

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* $\sqcap$ *y*
assumes *conv-def-2* [*simp*]: x^T = *x*
begin

subclass *aggregation-algebra*
apply *unfold-locales*
using *inf-dense* **by** *auto*

lemma *regular-bot-top-2*:
regular x $\longleftrightarrow$ *x* = *bot* $\vee$ *x* = *top*
by *simp*

end
declare[[*show-sorts*,*show-types*]]
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**

minatom = minatom-matrix
proof
  fix *f* :: (*'a*,*'b*) *square*
  show *minatom*$_M$ *f* $\preceq$ $\ominus\ominus$*f*
  proof (*unfold less-eq-matrix-def*, *rule allI*)
    fix *e* :: *'a* × *'a*
    have (*minatom*$_M$ *f*) *e* ≤ (*if f e* ≠ *bot then top else* −−(*f e*))
      by (*simp add*: *minatom-matrix-def*)
    also have ... = −−(*f e*)
      by *simp*
    also have ... = ($\ominus\ominus$*f*) *e*
      by (*simp add*: *uminus-matrix-def*)
    finally show (*minatom*$_M$ *f*) *e* ≤ ($\ominus\ominus$*f*) *e*
      .
  qed
next
  fix *f* :: (*'a*,*'b*) *square*
  let *?at* = *relation-algebra-signature.atom mone times-matrix less-eq-matrix mtop conv-matrix*
  show *f* ≠ *mbot* ⟶ *?at* (*minatom*$_M$ *f*)
  proof
    assume *1*: *f* ≠ *mbot*
    have *minatom*$_M$ *f* ⊙ *mtop* ⊙ (*minatom*$_M$ *f* ⊙ *mtop*)^t = *minatom*$_M$ *f* ⊙ *mtop* ⊙ (*minatom*$_M$ *f*)^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 (*minatom*$_M$ *f* ⊙ *mtop* ⊙ (*minatom*$_M$ *f*)^t) (*i*,*j*) = ($\bigsqcup_l$ ($\bigsqcup_k$ (*minatom*$_M$ *f*) (*i*,*k*) ∗ *mtop* (*k*,*l*)) ∗ ((*minatom*$_M$ *f*)^t) (*l*,*j*))
        by (*simp add*: *times-matrix-def*)
      also have ... = ($\bigsqcup_l$ ($\bigsqcup_k$ (*minatom*$_M$ *f*) (*i*,*k*) ∗ *top*) ∗ ((*minatom*$_M$ *f*) (*j*,*l*))^T)
        by (*simp add*: *top-matrix-def conv-matrix-def*)
      also have ... = ($\bigsqcup_l$ $\bigsqcup_k$ (*minatom*$_M$ *f*) (*i*,*k*) ∗ *top* ∗ ((*minatom*$_M$ *f*) (*j*,*l*))^T)
        by (*metis comp-right-dist-sum*)
      also have ... = ($\bigsqcup_l$ $\bigsqcup_k$ *if i* = *j* ∧ *l* = *k then* (*minatom*$_M$ *f*) (*i*,*k*) ∗ *top* ∗ ((*minatom*$_M$ *f*) (*j*,*l*))^T *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 minatom-at-most-one*)
      also have ... = (*if i* = *j then* ($\bigsqcup_l$ $\bigsqcup_k$ *if l* = *k then* (*minatom*$_M$ *f*) (*i*,*k*) ∗ *top* ∗ ((*minatom*$_M$ *f*) (*j*,*l*))^T *else bot*) *else bot*)
        by *auto*
      also have ... ≤ (*if i* = *j then top else bot*)
        by *simp*

also have ... = *mone* (*i*,*j*)
by (*simp add*: *one-matrix-def*)
finally show (*minatom*$_M$ *f* ⊙ *mtop* ⊙ (*minatom*$_M$ *f*)^t) (*i*,*j*) ≤ *mone* (*i*,*j*)
.
qed
finally have *2*: *minatom*$_M$ *f* ⊙ *mtop* ⊙ (*minatom*$_M$ *f* ⊙ *mtop*)^t ≼ *mone*
.
have *3*: *mtop* ⊙ (*minatom*$_M$ *f* ⊙ *mtop*) = *mtop*
proof (*rule ext*, *rule prod-cases*)
fix *i j*
from *minatom-at-least-one* **obtain** *ei ej* **where** (*minatom*$_M$ *f*) (*ei*,*ej*) = *top*
using *1* **by** *force*
hence *4*: *top* ∗ *top* ≤ (⨆$_l$ (*minatom*$_M$ *f*) (*ei*,*l*) ∗ *top*)
by (*metis comp-inf.ub-sum*)
have *top* ∗ (⨆$_l$ (*minatom*$_M$ *f*) (*ei*,*l*) ∗ *top*) ≤ (⨆$_k$ *top* ∗ (⨆$_l$ (*minatom*$_M$ *f*) (*k*,*l*) ∗ *top*))
by (*rule comp-inf.ub-sum*)
hence *top* ≤ (⨆$_k$ *top* ∗ (⨆$_l$ (*minatom*$_M$ *f*) (*k*,*l*) ∗ *top*))
using *4* **by** *auto*
also have ... = (⨆$_k$ *mtop* (*i*,*k*) ∗ (⨆$_l$ (*minatom*$_M$ *f*) (*k*,*l*) ∗ *mtop* (*l*,*j*)))
by (*simp add*: *top-matrix-def*)
also have ... = (*mtop* ⊙ (*minatom*$_M$ *f* ⊙ *mtop*)) (*i*,*j*)
by (*simp add*: *times-matrix-def*)
finally show (*mtop* ⊙ (*minatom*$_M$ *f* ⊙ *mtop*)) (*i*,*j*) = *mtop* (*i*,*j*)
by (*simp add*: *eq-iff top-matrix-def*)
qed
have (*minatom*$_M$ *f*)^t ⊙ *mtop* ⊙ ((*minatom*$_M$ *f*)^t ⊙ *mtop*)^t = (*minatom*$_M$ *f*)^t ⊙ *mtop* ⊙ (*minatom*$_M$ *f*)
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 ... ≼ *mone*
proof (*unfold less-eq-matrix-def*, *rule allI*, *rule prod-cases*)
fix *i j*
have ((*minatom*$_M$ *f*)^t ⊙ *mtop* ⊙ *minatom*$_M$ *f*) (*i*,*j*) = (⨆$_l$ (⨆$_k$ ((*minatom*$_M$ *f*)^t) (*i*,*k*) ∗ *mtop* (*k*,*l*)) ∗ (*minatom*$_M$ *f*) (*l*,*j*))
by (*simp add*: *times-matrix-def*)
also have ... = (⨆$_l$ (⨆$_k$ ((*minatom*$_M$ *f*) (*k*,*i*))^T ∗ *top*) ∗ (*minatom*$_M$ *f*) (*l*,*j*))
by (*simp add*: *top-matrix-def conv-matrix-def*)
also have ... = (⨆$_l$ ⨆$_k$ ((*minatom*$_M$ *f*) (*k*,*i*))^T ∗ *top* ∗ (*minatom*$_M$ *f*) (*l*,*j*))
by (*metis comp-right-dist-sum*)
also have ... = (⨆$_l$ ⨆$_k$ *if i* = *j* ∧ *l* = *k then* ((*minatom*$_M$ *f*) (*k*,*i*))^T ∗ *top* ∗ (*minatom*$_M$ *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 minatom-at-most-one)
      also have ... = (if i = j then (⨆l ⨆k if l = k then ((minatomM f) (k,i))T
* top * (minatomM f) (l,j) else bot) else bot)
        by auto
      also have ... ≤ (if i = j then top else bot)
        by simp
      also have ... = mone (i,j)
        by (simp add: one-matrix-def)
      finally show ((minatomM f)t ⊙ mtop ⊙ (minatomM f)) (i,j) ≤ mone (i,j)
        .
    qed
    finally have 5: (minatomM f)t ⊙ mtop ⊙ ((minatomM f)t ⊙ mtop)t ⪯ mone
      .
    have mtop ⊙ ((minatomM f)t ⊙ mtop) = mtop
      using 3 by (metis matrix-monoid.mult-assoc
matrix-stone-relation-algebra.conv-dist-comp
matrix-stone-relation-algebra.conv-top)
    thus ?at (minatomM f)
      using 2 3 5 by blast
  qed
next
  fix f g :: ('a,'b) square
  let ?at = relation-algebra-signature.atom mone times-matrix less-eq-matrix
mtop conv-matrix
  show ?at g ∧ g ⊗ f ≠ mbot ⟶ summ (minatomM f ⊗ f) ⪯ summ (g ⊗ f)
  proof
    assume 1: ?at g ∧ g ⊗ f ≠ mbot
    hence 2: g = ⊖⊖g
      using matrix-stone-relation-algebra.atom-regular by blast
    show summ (minatomM f ⊗ f) ⪯ summ (g ⊗ f)
    proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
      fix i j
      from minatom-at-least-one obtain ei ej where 3: (minatomM f) (ei,ej) =
top
        using 1 by force
      hence 4: ∀ k l . ¬(k = ei ∧ l = ej) ⟶ (minatomM f) (k,l) = bot
        by (metis (mono-tags, hide-lams) bot.extremum inf.bot-unique prod.inject
minatom-at-most-one)
      from agg-matrix-bot obtain di dj where 5: (g ⊗ f) (di,dj) ≠ bot
        using 1 by force
      hence 6: g (di,dj) ≠ bot
        by (metis inf-bot-left inf-matrix-def)
      hence 7: g (di,dj) = top
        using 2 by (metis uminus-matrix-def uminus-def)
      hence 8: (g ⊗ f) (di,dj) = f (di,dj)
        by (metis inf-matrix-def inf-top.left-neutral)
      have 9: ∀ k l . k ≠ di ⟶ g (k,l) = bot
      proof (intro allI, rule impI)
        fix k l
```

assume *10*: $k \neq di$
have $top * (g\ (k,l))^T = g\ (di,dj) * top * (g^t)\ (l,k)$
using *7* **by** (*simp add*: *conv-matrix-def*)
also have ... $\leq (\bigsqcup_n\ g\ (di,n) * top) * (g^t)\ (l,k)$
by (*metis comp-inf.ub-sum comp-right-dist-sum*)
also have ... $\leq (\bigsqcup_m\ (\bigsqcup_n\ g\ (di,n) * top) * (g^t)\ (m,k))$
by (*metis comp-inf.ub-sum*)
also have ... $= (g \odot mtop \odot g^t)\ (di,k)$
by (*simp add*: *times-matrix-def top-matrix-def*)
also have ... $\leq mone\ (di,k)$
using *1* **by** (*metis matrix-stone-relation-algebra.atom-expanded less-eq-matrix-def*)
also have ... $= bot$
apply (*unfold one-matrix-def*)
using *10* **by** *auto*
finally have $g\ (k,l) \neq top$
using *5* **by** (*metis bot.extremum conv-def inf.bot-unique mult.left-neutral one-def*)
thus $g\ (k,l) = bot$
using *2* **by** (*metis uminus-def uminus-matrix-def*)
qed
have $\forall\, k\ l\ .\ l \neq dj \longrightarrow g\ (k,l) = bot$
proof (*intro allI*, *rule impI*)
fix *k l*
assume *11*: $l \neq dj$
have $(g\ (k,l))^T * top = (g^t)\ (l,k) * top * g\ (di,dj)$
using *7* **by** (*simp add*: *comp-associative conv-matrix-def*)
also have ... $\leq (\bigsqcup_n\ (g^t)\ (l,n) * top) * g\ (di,dj)$
by (*metis comp-inf.ub-sum comp-right-dist-sum*)
also have ... $\leq (\bigsqcup_m\ (\bigsqcup_n\ (g^t)\ (l,n) * top) * g\ (m,dj))$
by (*metis comp-inf.ub-sum*)
also have ... $= (g^t \odot mtop \odot g)\ (l,dj)$
by (*simp add*: *times-matrix-def top-matrix-def*)
also have ... $\leq mone\ (l,dj)$
using *1* **by** (*metis matrix-stone-relation-algebra.atom-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 $g\ (k,l) = bot$
using *2* **by** (*metis uminus-def uminus-matrix-def*)
qed
hence *12*: $\forall\, k\ l\ .\ \neg(k = di \wedge l = dj) \longrightarrow (g \otimes f)\ (k,l) = bot$
using *9* **by** (*metis inf-bot-left inf-matrix-def*)
have $(\sum_k \sum_l\ (minatom_M\ f \otimes f)\ (k,l)) = (\sum_k \sum_l\ if\ k = ei \wedge l = ej\ then\ (minatom_M\ f \otimes f)\ (k,l)\ else\ (minatom_M\ f \otimes f)\ (k,l))$

```
      by simp
    also have ... = (∑ k ∑ l if k = ei ∧ l = ej then (minatom_M f ⊗ f) (k,l)
else (minatom_M f) (k,l) ⊓ f (k,l))
      by (unfold inf-matrix-def) simp
    also have ... = (∑ k ∑ l if k = ei ∧ l = ej then (minatom_M f ⊗ f) (k,l)
else bot)
      apply (rule aggregation.sum-0.cong)
      apply simp
      using 4 by (metis inf-bot-left)
    also have ... = (minatom_M f ⊗ f) (ei,ej) + bot
      by (unfold agg-delta-2) simp
    also have ... = f (ei,ej) + bot
      using 3 by (simp add: inf-matrix-def)
    also have ... ≤ (g ⊗ f) (di,dj) + bot
      using 3 5 6 7 8 by (metis minatom-matrix-def)
    also have ... = (∑ k ∑ l if k = di ∧ l = dj then (g ⊗ f) (k,l) else bot)
      by (unfold agg-delta-2) simp
    also have ... = (∑ k ∑ l if k = di ∧ l = dj then (g ⊗ f) (k,l) else (g ⊗ f)
(k,l))
      apply (rule aggregation.sum-0.cong)
      apply simp
      using 12 by metis
    also have ... = (∑ k ∑ l (g ⊗ f) (k,l))
      by simp
    finally show (sum_M (minatom_M f ⊗ f)) (i,j) ≤ (sum_M (g ⊗ f)) (i,j)
      by (simp add: sum-matrix-def)
  qed
 qed
next
 fix f g :: ('a,'b) square
 let ?h = hd enum-class.enum
 show sum_M f ⪯ sum_M g ∨ sum_M g ⪯ sum_M f
 proof (cases (sum_M f) (?h,?h) ≤ (sum_M g) (?h,?h))
  case 1: True
  have sum_M f ⪯ sum_M g
   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_M g) (?h,?h) ≤ (sum_M f) (?h,?h)
   by (simp add: linear)
  have sum_M g ⪯ 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* . (∀ *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

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** *minatom* = *minatom-matrix*
proof
fix *f* :: ('*a*,'*b*) *square*
show $minatom_M\ f \preceq \ominus\ominus f$
by (*simp add*: *agg-square-m-algebra.minatom-below*)
next
fix *f* :: ('*a*,'*b*) *square*
let *?at* = *relation-algebra-signature.atom mone times-matrix less-eq-matrix mtop conv-matrix*
show *f* ≠ *mbot* ⟶ *?at* ($minatom_M\ f$)
by (*simp add*: *agg-square-m-algebra.minatom-atom*)
next
fix *f g* :: ('*a*,'*b*) *square*
let *?at* = *relation-algebra-signature.atom mone times-matrix less-eq-matrix mtop conv-matrix*
show *?at g* ∧ *g* ⊗ *f* ≠ *mbot* ⟶ $sum2_M\ (minatom_M\ f \otimes f) \preceq sum2_M\ (g \otimes f)$
proof
let *?h* = *hd enum-class.enum*
assume *?at g* ∧ *g* ⊗ *f* ≠ *mbot*
hence $sum_M\ (minatom_M\ f \otimes f) \preceq sum_M\ (g \otimes f)$
by (*simp add*: *agg-square-m-algebra.minatom-min*)
hence ($sum_M\ (minatom_M\ f \otimes f)$) (*?h*,*?h*) ≤ ($sum_M\ (g \otimes f)$) (*?h*,*?h*)
by (*simp add*: *less-eq-matrix-def*)
thus $sum2_M\ (minatom_M\ f \otimes f) \preceq sum2_M\ (g \otimes f)$
by (*simp add*: *sum-matrix-def sum-matrix-2-def less-eq-matrix-def*)
qed
next
fix *f g* :: ('*a*,'*b*) *square*
let *?h* = *hd enum-class.enum*
have $sum_M\ f \preceq sum_M\ g \vee sum_M\ g \preceq sum_M\ f$
by (*simp add*: *agg-square-m-algebra.sum-linear*)
hence ($sum_M\ f$) (*?h*,*?h*) ≤ ($sum_M\ g$) (*?h*,*?h*) ∨ ($sum_M\ g$) (*?h*,*?h*) ≤ (sum_M *f*) (*?h*,*?h*)
using *less-eq-matrix-def* **by** *auto*
thus $sum2_M\ f \preceq sum2_M\ g \vee sum2_M\ g \preceq sum2_M\ f$
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

class *aggregation-kleene-algebra* = *aggregation-algebra* + *star* +
assumes *star-def* [*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 *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** *minatom* = *minatom-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** *minatom* = *minatom-matrix* **..**

end

5 Algebras for Aggregation and Minimisation with a Linear Order

theory *Linear-Aggregation-Algebras*

imports *Matrix-Aggregation-Algebras Real*

begin

no-notation
  inf (**infixl** ⊓ *70*)
  and *uminus* (− - [*81*] *80*)

datatype *'a ext* =
    Bot
  | *Val 'a*
  | *Top*

instantiation *ext* :: (*linordered-ab-semigroup-add*)
linear-aggregation-kleene-algebra
begin

fun *plus-ext* :: *'a ext* ⇒ *'a ext* ⇒ *'a ext* **where**
  plus-ext Bot x = *x*
| *plus-ext* (*Val x*) *Bot* = *Val x*
| *plus-ext* (*Val x*) (*Val y*) = *Val* (*x* + *y*)
| *plus-ext* (*Val* -) *Top* = *Top*
| *plus-ext Top* - = *Top*

fun *sup-ext* :: *'a ext* ⇒ *'a ext* ⇒ *'a ext* **where**
  sup-ext Bot x = *x*
| *sup-ext* (*Val x*) *Bot* = *Val x*
| *sup-ext* (*Val x*) (*Val y*) = *Val* (*max x y*)
| *sup-ext* (*Val* -) *Top* = *Top*
| *sup-ext Top* - = *Top*

fun *inf-ext* :: *'a ext* ⇒ *'a ext* ⇒ *'a ext* **where**
  inf-ext Bot - = *Bot*
| *inf-ext* (*Val* -) *Bot* = *Bot*
| *inf-ext* (*Val x*) (*Val y*) = *Val* (*min x y*)
| *inf-ext* (*Val x*) *Top* = *Val x*
| *inf-ext Top x* = *x*

fun *times-ext* :: *'a ext* ⇒ *'a ext* ⇒ *'a ext* **where** *times-ext x y* = *x* ⊓ *y*

fun *uminus-ext* :: *'a ext* ⇒ *'a ext* **where**
  uminus-ext Bot = *Top*
| *uminus-ext* (*Val* -) = *Bot*
| *uminus-ext Top* = *Bot*

fun *star-ext* :: *'a ext* ⇒ *'a ext* **where** *star-ext* - = *Top*

fun *conv-ext* :: *'a ext* ⇒ *'a ext* **where** *conv-ext x* = *x*

definition *bot-ext* :: *'a ext* **where** *bot-ext* ≡ *Bot*

definition *one-ext* :: *'a ext* **where** *one-ext* ≡ *Top*
definition *top-ext* :: *'a ext* **where** *top-ext* ≡ *Top*

fun *less-eq-ext* :: *'a ext* ⇒ *'a ext* ⇒ *bool* **where**
  less-eq-ext Bot - = True
| *less-eq-ext* (*Val -*) *Bot* = *False*
| *less-eq-ext* (*Val x*) (*Val y*) = (*x* ≤ *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* ⇒ *'a ext* ⇒ *bool* **where** *less-ext x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *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* ≤ *y* ∧ ¬ *y* ≤ *x*)
    by *simp*
  show *x* ≤ *x*
    using *less-eq-ext.elims*(*3*) **by** *fastforce*
  show *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *x*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ≤ *y* ⟹ *x* ≤ *z* ⟹ *x* ≤ *y* ⊓ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⟹ *z* ≤ *x* ⟹ *y* ⊔ *z* ≤ *x*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *bot* ≤ *x*
    by (*simp add*: *bot-ext-def*)
  show *x* ≤ *top*
    by (*cases x*) (*simp-all add*: *top-ext-def*)
  show *x* ≠ *bot* ∧ *x* + *bot* ≤ *y* + *bot* ⟶ *x* + *z* ≤ *y* + *z*
    by (*cases x*; *cases y*; *cases z*) (*simp-all add*: *bot-ext-def add-right-mono*)
  show *x* + *y* + *bot* = *x* + *y*
    by (*cases x*; *cases y*) (*simp-all add*: *bot-ext-def*)

show $x + y = bot \longrightarrow x = bot$
by (*cases x*; *cases y*) (*simp-all add*: *bot-ext-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-def top-ext-def*)
show $(1::'a\ ext) = top$
by (*simp add*: *one-ext-def top-ext-def*)
show $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = top$
by (*simp add*: *top-ext-def*)
qed

end

lemma *example-real-ext-matrix*:
fixes $x :: ('a::enum, real\ ext)\ square$
shows $minatom_M\ x \preceq \ominus\ominus x$
by (*rule agg-square-m-algebra.minatom-below*)

datatype *real-max* = *Rmax real*

instantiation *real-max* :: *linordered-ab-semigroup-add*
begin

fun *less-eq-real-max* **where** *less-eq-real-max* (*Rmax x*) (*Rmax y*) = $(x \leq y)$
fun *less-real-max* **where** *less-real-max* (*Rmax x*) (*Rmax y*) = $(x < y)$
fun *plus-real-max* **where** *plus-real-max* (*Rmax x*) (*Rmax y*) = *Rmax* (*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 $x :: ('a::enum, real\text{-}max\ ext)\ square$
shows $minatom_M\ x \preceq \ominus\ominus x$
by (*rule agg-square-m-algebra.minatom-below*)

datatype *real-min* = *Rmin real*

instantiation *real-min* :: *linordered-ab-semigroup-add*
begin

fun *less-eq-real-min* **where** *less-eq-real-min* (*Rmin x*) (*Rmin y*) = $(x \leq y)$
fun *less-real-min* **where** *less-real-min* (*Rmin x*) (*Rmin y*) = $(x < y)$
fun *plus-real-min* **where** *plus-real-min* (*Rmin x*) (*Rmin y*) = *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 $x :: ('a::enum, real\text{-}min\ ext)\ square$
shows $minatom_M\ x \preceq \ominus\ominus x$
by (*rule agg-square-m-algebra.minatom-below*)

There is a zero.

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* ⇒ *'a ext0* ⇒ *'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*) = *Val* (*x* + *y*)
| *plus-ext0* (*Val* -) *Top* = *Top*
| *plus-ext0 Top* - = *Top*

fun *sup-ext0* :: *'a ext0* ⇒ *'a ext0* ⇒ *'a ext0* **where**
  sup-ext0 Bot x = *x*
| *sup-ext0* (*Val x*) *Bot* = *Val x*
| *sup-ext0* (*Val x*) (*Val y*) = *Val* (*max x y*)
| *sup-ext0* (*Val* -) *Top* = *Top*
| *sup-ext0 Top* - = *Top*

fun *inf-ext0* :: *'a ext0* ⇒ *'a ext0* ⇒ *'a ext0* **where**
  inf-ext0 Bot - = *Bot*
| *inf-ext0* (*Val* -) *Bot* = *Bot*
| *inf-ext0* (*Val x*) (*Val y*) = *Val* (*min x y*)
| *inf-ext0* (*Val x*) *Top* = *Val x*
| *inf-ext0 Top x* = *x*

fun *times-ext0* :: *'a ext0* ⇒ *'a ext0* ⇒ *'a ext0* **where** *times-ext0 x y* = *x* ⊓ *y*

fun *uminus-ext0* :: *'a ext0* ⇒ *'a ext0* **where**
  uminus-ext0 Bot = *Top*
| *uminus-ext0* (*Val* -) = *Bot*
| *uminus-ext0 Top* = *Bot*

fun *star-ext0* :: *'a ext0* ⇒ *'a ext0* **where** *star-ext0* - = *Top*

fun *conv-ext0* :: *'a ext0* ⇒ *'a ext0* **where** *conv-ext0 x* = *x*

definition *bot-ext0* :: *'a ext0* **where** *bot-ext0* ≡ *Bot*
definition *one-ext0* :: *'a ext0* **where** *one-ext0* ≡ *Top*
definition *top-ext0* :: *'a ext0* **where** *top-ext0* ≡ *Top*

fun *less-eq-ext0* :: *'a ext0* ⇒ *'a ext0* ⇒ *bool* **where**
  less-eq-ext0 Bot - = *True*
| *less-eq-ext0* (*Val* -) *Bot* = *False*
| *less-eq-ext0* (*Val x*) (*Val y*) = (*x* ≤ *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* ⇒ *'a ext0* ⇒ *bool* **where** *less-ext0 x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)

instance
proof
  fix *x y z* :: *'a 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* ≤ *y* ∧ ¬ *y* ≤ *x*)
    by *simp*
  show *x* ≤ *x*
    using *less-eq-ext0.elims*(*3*) **by** *fastforce*
  show *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *x*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ≤ *y* ⟹ *x* ≤ *z* ⟹ *x* ≤ *y* ⊓ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⟹ *z* ≤ *x* ⟹ *y* ⊔ *z* ≤ *x*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *bot* ≤ *x*
    by (*simp add*: *bot-ext0-def*)
  show *x* ≤ *top*
    by (*cases x*) (*simp-all add*: *top-ext0-def*)
  show *x* ≠ *bot* ∧ *x* + *bot* ≤ *y* + *bot* ⟶ *x* + *z* ≤ *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* + *bot* = *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 $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = top$
by (*simp add*: *top-ext0-def*)
qed

end

instantiation *real* :: *linordered-comm-monoid-add*
begin

instance ..

end

There is a least element, which is also a zero.

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* ⇒ *'a extT* **where**
plus-extT (*Val x*) (*Val y*) = *Val* (*x* + *y*)
| *plus-extT* (*Val* -) *Top* = *Top*
| *plus-extT Top* - = *Top*

fun *sup-extT* :: *'a extT* ⇒ *'a extT* ⇒ *'a extT* **where**
  sup-extT (*Val x*) (*Val y*) = *Val* (*max x y*)
| *sup-extT* (*Val -*) *Top* = *Top*
| *sup-extT Top -* = *Top*

fun *inf-extT* :: *'a extT* ⇒ *'a extT* ⇒ *'a extT* **where**
  inf-extT (*Val x*) (*Val y*) = *Val* (*min x y*)
| *inf-extT* (*Val x*) *Top* = *Val x*
| *inf-extT Top x* = *x*

fun *times-extT* :: *'a extT* ⇒ *'a extT* ⇒ *'a extT* **where** *times-extT x y* = *x* ⊓ *y*

fun *uminus-extT* :: *'a extT* ⇒ *'a extT* **where** *uminus-extT x* = (*if x* = *Val bot then Top else Val bot*)

fun *star-extT* :: *'a extT* ⇒ *'a extT* **where** *star-extT -* = *Top*

fun *conv-extT* :: *'a extT* ⇒ *'a extT* **where** *conv-extT x* = *x*

definition *bot-extT* :: *'a extT* **where** *bot-extT* ≡ *Val bot*
definition *one-extT* :: *'a extT* **where** *one-extT* ≡ *Top*
definition *top-extT* :: *'a extT* **where** *top-extT* ≡ *Top*

fun *less-eq-extT* :: *'a extT* ⇒ *'a extT* ⇒ *bool* **where**
  less-eq-extT (*Val x*) (*Val y*) = (*x* ≤ *y*)
| *less-eq-extT Top* (*Val -*) = *False*
| *less-eq-extT -* *Top* = *True*

fun *less-extT* :: *'a extT* ⇒ *'a extT* ⇒ *bool* **where** *less-extT x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)

instance
proof
  fix *x y z* :: *'a extT*
  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* ≤ *y* ∧ ¬ *y* ≤ *x*)
    by *simp*
  show *x* ≤ *x*
    by (*cases x*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *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 (*cases x*) (*simp-all add*: *bot-extT-def*)
show $x \leq top$
by (*cases x*) (*simp-all add*: *top-extT-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-extT-def add-right-mono*)
show $x + y + bot = x + y$
by (*cases x*; *cases y*) (*simp-all add*: *bot-extT-def*)
show $x + y = bot \longrightarrow x = bot$
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 $x \leq y \vee y \leq x$
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-extT-def top-extT-def*)
show $(1::'a\ extT) = top$
by (*simp add*: *one-extT-def top-extT-def*)
show $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = top$
by (*simp add*: *top-extT-def*)
qed

end

datatype *real-max-bot* =
MInfty
| *R real*

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

definition *bot-real-max-bot* ≡ *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 ≤ y)

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 + x
    by (cases x; cases y) simp-all
  show (x < y) = (x ≤ y ∧ ¬ y ≤ x)
    by (cases x; cases y) auto
  show x ≤ x
    by (cases x) simp-all
  show x ≤ y ⟹ y ≤ z ⟹ x ≤ z
    by (cases x; cases y; cases z) simp-all
  show x ≤ y ⟹ y ≤ x ⟹ x = y
    by (cases x; cases y) simp-all
  show x ≤ y ⟹ z + x ≤ z + y
    by (cases x; cases y; cases z) simp-all
  show x ≤ y ∨ y ≤ x
    by (cases x; cases y) auto
  show bot ≤ x
    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
```

There is a greatest element, which is also a zero.

```
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 *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 extB* =
Bot
| *Val 'a*

instantiation *extB* :: (*linordered-comm-monoid-add-top*)
linear-aggregation-kleene-algebra
begin

fun *plus-extB* :: *'a extB* ⇒ *'a extB* ⇒ *'a extB* **where**
plus-extB Bot Bot = *Val top*
| *plus-extB Bot* (*Val x*) = *Val x*
| *plus-extB* (*Val x*) *Bot* = *Val x*
| *plus-extB* (*Val x*) (*Val y*) = *Val* (*x* + *y*)

fun *sup-extB* :: *'a extB* ⇒ *'a extB* ⇒ *'a extB* **where**
sup-extB Bot x = *x*
| *sup-extB* (*Val x*) *Bot* = *Val x*
| *sup-extB* (*Val x*) (*Val y*) = *Val* (*max x y*)

fun *inf-extB* :: *'a extB* ⇒ *'a extB* ⇒ *'a extB* **where**
inf-extB Bot - = *Bot*
| *inf-extB* (*Val* -) *Bot* = *Bot*
| *inf-extB* (*Val x*) (*Val y*) = *Val* (*min x y*)

fun *times-extB* :: *'a extB* ⇒ *'a extB* ⇒ *'a extB* **where** *times-extB x y* = *x* ⊓ *y*

fun *uminus-extB* :: *'a extB* ⇒ *'a extB* **where**
uminus-extB Bot = *Val top*
| *uminus-extB* (*Val* -) = *Bot*

fun *star-extB* :: *'a extB* ⇒ *'a extB* **where** *star-extB* - = *Val top*

fun *conv-extB* :: *'a extB* ⇒ *'a extB* **where** *conv-extB x* = *x*

definition *bot-extB* :: *'a extB* **where** *bot-extB* ≡ *Bot*
definition *one-extB* :: *'a extB* **where** *one-extB* ≡ *Val top*
definition *top-extB* :: *'a extB* **where** *top-extB* ≡ *Val top*

fun *less-eq-extB* :: *'a extB* ⇒ *'a extB* ⇒ *bool* **where**

less-eq-extB Bot - = True
| *less-eq-extB* (*Val -*) *Bot = False*
| *less-eq-extB* (*Val x*) (*Val y*) = (*x* ≤ *y*)

fun *less-extB* :: *'a extB* ⇒ *'a extB* ⇒ *bool* **where** *less-extB x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)

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* + *x*
by (*cases x*; *cases y*) (*simp-all add*: *add.commute*)
show (*x* < *y*) = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)
by *simp*
show *x* ≤ *x*
by (*cases x*) *simp-all*
show *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*
by (*cases x*; *cases y*; *cases z*) *simp-all*
show *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y*
by (*cases x*; *cases y*) *simp-all*
show *x* ⊓ *y* ≤ *x*
by (*cases x*; *cases y*) *simp-all*
show *x* ⊓ *y* ≤ *y*
by (*cases x*; *cases y*) *simp-all*
show *x* ≤ *y* ⟹ *x* ≤ *z* ⟹ *x* ≤ *y* ⊓ *z*
by (*cases x*; *cases y*; *cases z*) *simp-all*
show *x* ≤ *x* ⊔ *y*
by (*cases x*; *cases y*) *simp-all*
show *y* ≤ *x* ⊔ *y*
by (*cases x*; *cases y*) *simp-all*
show *y* ≤ *x* ⟹ *z* ≤ *x* ⟹ *y* ⊔ *z* ≤ *x*
by (*cases x*; *cases y*; *cases z*) *simp-all*
show *bot* ≤ *x*
by (*simp add*: *bot-extB-def*)
show *1*: *x* ≤ *top*
by (*cases x*) (*simp-all add*: *top-extB-def*)
show *x* ≠ *bot* ∧ *x* + *bot* ≤ *y* + *bot* ⟶ *x* + *z* ≤ *y* + *z*
apply (*cases x*; *cases y*; *cases z*)
prefer *6* **using** *1* **apply** (*metis* (*mono-tags*, *lifting*) *plus-extB.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* + *y*
by (*cases x*; *cases y*) (*simp-all add*: *bot-extB-def*)
show *x* + *y* = *bot* ⟶ *x* = *bot*
by (*cases x*; *cases y*) (*simp-all add*: *bot-extB-def*)
show *x* ≤ *y* ∨ *y* ≤ *x*
by (*cases x*; *cases y*) (*simp-all add*: *linear*)

show $-x = (\mathit{if}\ x = \mathit{bot}\ \mathit{then}\ \mathit{top}\ \mathit{else}\ \mathit{bot})$
by $(\mathit{cases}\ x)\ (\mathit{simp\text{-}all}\ \mathit{add}\colon \mathit{bot\text{-}extB\text{-}def}\ \mathit{top\text{-}extB\text{-}def})$
show $(1{::}'a\ \mathit{extB}) = \mathit{top}$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{one\text{-}extB\text{-}def}\ \mathit{top\text{-}extB\text{-}def})$
show $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = \mathit{top}$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{top\text{-}extB\text{-}def})$
qed

end

datatype *real-min-top* =
R real
| *PInfty*

instantiation *real-min-top* :: *linordered-comm-monoid-add-top*
begin

definition *top-real-min-top* ≡ *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*) (*R y*) = *R* (*min x y*)

instance
proof
fix *x y z* :: *real-min-top*
show $(x + y) + z = x + (y + z)$
by $(\mathit{cases}\ x;\ \mathit{cases}\ y;\ \mathit{cases}\ z)\ \mathit{simp\text{-}all}$
show $x + y = y + x$
by $(\mathit{cases}\ x;\ \mathit{cases}\ y)\ \mathit{simp\text{-}all}$
show $(x < y) = (x \leq y \wedge \neg\ y \leq x)$
by $(\mathit{cases}\ x;\ \mathit{cases}\ y)\ \mathit{auto}$
show $x \leq x$
by $(\mathit{cases}\ x)\ \mathit{simp\text{-}all}$

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 $x \leq top$
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

Ideally, we would like to show that the unit interval of real numbers 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 *simp*

lift-definition *top-unit* :: *unit* **is** *1*
by *simp*

lift-definition *less-eq-unit* :: *unit* ⇒ *unit* ⇒ *bool* **is** *less-eq* **.**

lift-definition *less-unit* :: *unit* ⇒ *unit* ⇒ *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

instantiation *unit* :: *linordered-comm-monoid-add-top*
begin

abbreviation *tl* :: *real* ⇒ *real* ⇒ *real* **where**
tl x y ≡ *max* (*x* + *y* − *1*) *0*

lemma *tl-assoc*:
x ∈ {*0..1*} ⟹ *z* ∈ {*0..1*} ⟹ *tl* (*tl x y*) *z* = *tl x* (*tl y z*)
by *auto*

lemma *tl-top-zero*:
x ∈ {*0..1*} ⟹ *tl 1 x* = *x*
by *auto*

lift-definition *plus-unit* :: *unit* ⇒ *unit* ⇒ *unit* **is** *tl*
by *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

There is a least element, which is also a zero, and a greatest element.

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* = λ*x* . *top* **and** *uminus* = λ*x* . *if x* = *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* ≤ *x* + *y*
using *add-left-mono bot.extremum* **by** *fastforce*

lemma *t-max*: $max\ x\ y \leq x + y$
using *add-commute add-increasing* **by** *force*

end

For the same reason as 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 *simp*

lift-definition *top-unit2* :: *unit2* **is** *1*
by *simp*

lift-definition *less-eq-unit2* :: *unit2* ⇒ *unit2* ⇒ *bool* **is** *less-eq* **.**

lift-definition *less-unit2* :: *unit2* ⇒ *unit2* ⇒ *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

instantiation *unit2* :: *linordered-bounded-comm-monoid-add-bot*
begin

abbreviation *sp* :: *real* ⇒ *real* ⇒ *real* **where**
$sp\ x\ y \equiv x + y - x * y$

lemma *sp-assoc*:
$sp\ (sp\ x\ y)\ z = sp\ x\ (sp\ y\ z)$
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 $sp\ z\ x \leq sp\ z\ y$

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* ⇒ *unit2* ⇒ *unit2* **is** *sp*
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 *1 2* **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 *simp*
have $y * (x - 1) \leq 0$
using *1 2* **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

Constant aggregation.

class *pointed-linorder* = *linorder* +
fixes *const* :: *'a*

datatype *'a extC* =
Bot
| *Val 'a*
| *Top*

instantiation *extC* :: (*pointed-linorder*) *linear-aggregation-kleene-algebra*
begin

fun *plus-extC* :: *'a extC* ⇒ *'a extC* ⇒ *'a extC* **where** *plus-extC x y* = *Val const*

fun *sup-extC* :: *'a extC* ⇒ *'a extC* ⇒ *'a extC* **where**
  sup-extC Bot x = *x*
| *sup-extC* (*Val x*) *Bot* = *Val x*
| *sup-extC* (*Val x*) (*Val y*) = *Val* (*max x y*)
| *sup-extC* (*Val -*) *Top* = *Top*
| *sup-extC Top -* = *Top*

fun *inf-extC* :: *'a extC* ⇒ *'a extC* ⇒ *'a extC* **where**
  inf-extC Bot - = *Bot*
| *inf-extC* (*Val -*) *Bot* = *Bot*
| *inf-extC* (*Val x*) (*Val y*) = *Val* (*min x y*)
| *inf-extC* (*Val x*) *Top* = *Val x*
| *inf-extC Top x* = *x*

fun *times-extC* :: *'a extC* ⇒ *'a extC* ⇒ *'a extC* **where** *times-extC x y* = *x* ⊓ *y*

fun *uminus-extC* :: *'a extC* ⇒ *'a extC* **where**
  uminus-extC Bot = *Top*
| *uminus-extC* (*Val -*) = *Bot*
| *uminus-extC Top* = *Bot*

fun *star-extC* :: *'a extC* ⇒ *'a extC* **where** *star-extC -* = *Top*

fun *conv-extC* :: *'a extC* ⇒ *'a extC* **where** *conv-extC x* = *x*

definition *bot-extC* :: *'a extC* **where** *bot-extC* ≡ *Bot*
definition *one-extC* :: *'a extC* **where** *one-extC* ≡ *Top*
definition *top-extC* :: *'a extC* **where** *top-extC* ≡ *Top*

fun *less-eq-extC* :: *'a extC* ⇒ *'a extC* ⇒ *bool* **where**
  less-eq-extC Bot - = *True*
| *less-eq-extC* (*Val -*) *Bot* = *False*
| *less-eq-extC* (*Val x*) (*Val y*) = (*x* ≤ *y*)
| *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-extC* :: *'a extC* ⇒ *'a extC* ⇒ *bool* **where** *less-extC x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)

instance
proof
  fix *x y z* :: *'a extC*
  show (*x* + *y*) + *z* = *x* + (*y* + *z*)
    by *simp*
  show *x* + *y* = *y* + *x*

by *simp*
show $(x < y) = (x \leq y \wedge \neg\ y \leq x)$
by *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-extC-def*)
show $x \leq top$
by (*cases x*) (*simp-all add*: *top-extC-def*)
show $x \neq bot \wedge x + bot \leq y + bot \longrightarrow x + z \leq y + z$
by *simp*
show $x + y + bot = x + y$
by *simp*
show $x + y = bot \longrightarrow x = bot$
by (*simp add*: *bot-extC-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-extC-def top-extC-def*)
show $(1::'a\ extC) = top$
by (*simp add*: *one-extC-def top-extC-def*)
show $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = top$
by (*simp add*: *top-extC-def*)
qed

end

instantiation *real* :: *pointed-linorder*
begin

instance ..

end

Alpha-Median

fun *median3* :: *'a*::*ord* ⇒ *'a* ⇒ *'a* ⇒ *'a* **where**
median3 x y z =
(*if x* ≤ *y* ∧ *y* ≤ *z then y else*
if x ≤ *z* ∧ *z* ≤ *y then z else*
if y ≤ *x* ∧ *x* ≤ *z then x else*
if y ≤ *z* ∧ *z* ≤ *x then z else*
if z ≤ *x* ∧ *x* ≤ *y 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* :: *'a*
show *median3 const* (*median3 const a b*) *c* = *median3 const a* (*median3 const b c*)
by (*cases const* ≤ *a*; *cases const* ≤ *b*; *cases const* ≤ *c*; *cases a* ≤ *b*; *cases a* ≤ *c*; *cases b* ≤ *c*) *auto*
show *median3 const a b* = *median3 const b a*
by (*cases const* ≤ *a*; *cases const* ≤ *b*; *cases a* ≤ *b*) *auto*
assume *a* ≤ *b*
thus *median3 const c a* ≤ *median3 const c b*
by (*cases const* ≤ *a*; *cases const* ≤ *b*; *cases const* ≤ *c*; *cases a* ≤ *c*; *cases b* ≤ *c*) *auto*
qed

Counting aggregation.

datatype *'a extN* =
Bot
| *Val 'a*
| *N nat*
| *Top*

instantiation *extN* :: (*linorder*) *linear-aggregation-kleene-algebra*
begin

fun *plus-extN* :: *'a extN* ⇒ *'a extN* ⇒ *'a extN* **where**
plus-extN Bot Bot = *N 0*
| *plus-extN Bot* (*Val* -) = *N 1*
| *plus-extN Bot* (*N y*) = *N y*
| *plus-extN Bot Top* = *N 1*
| *plus-extN* (*Val* -) *Bot* = *N 1*
| *plus-extN* (*Val* -) (*Val* -) = *N 2*
| *plus-extN* (*Val* -) (*N y*) = *N* (*y* + *1*)
| *plus-extN* (*Val* -) *Top* = *N 2*

```
| plus-extN (N x) Bot = N x
| plus-extN (N x) (Val -) = N (x + 1)
| plus-extN (N x) (N y) = N (x + y)
| plus-extN (N x) Top = N (x + 1)
| plus-extN Top Bot = N 1
| plus-extN Top (Val -) = N 2
| plus-extN Top (N y) = N (y + 1)
| plus-extN Top Top = N 2

fun sup-extN :: 'a extN ⇒ 'a extN ⇒ 'a extN where
  sup-extN Bot x = x
| sup-extN (Val x) Bot = Val x
| sup-extN (Val x) (Val y) = Val (max x y)
| sup-extN (Val -) (N y) = N y
| sup-extN (Val -) Top = Top
| sup-extN (N x) Bot = N x
| sup-extN (N x) (Val -) = N x
| sup-extN (N x) (N y) = N (max x y)
| sup-extN (N -) Top = Top
| sup-extN Top - = Top

fun inf-extN :: 'a extN ⇒ 'a extN ⇒ 'a extN where
  inf-extN Bot - = Bot
| inf-extN (Val -) Bot = Bot
| inf-extN (Val x) (Val y) = Val (min x y)
| inf-extN (Val x) (N -) = Val x
| inf-extN (Val x) Top = Val x
| inf-extN (N -) Bot = Bot
| inf-extN (N -) (Val y) = Val y
| inf-extN (N x) (N y) = N (min x y)
| inf-extN (N x) Top = N x
| inf-extN Top y = y

fun times-extN :: 'a extN ⇒ 'a extN ⇒ 'a extN where times-extN x y = x ⊓ y

fun uminus-extN :: 'a extN ⇒ 'a extN where
  uminus-extN Bot = Top
| uminus-extN (Val -) = Bot
| uminus-extN (N -) = Bot
| uminus-extN Top = Bot

fun star-extN :: 'a extN ⇒ 'a extN where star-extN - = Top

fun conv-extN :: 'a extN ⇒ 'a extN where conv-extN x = x

definition bot-extN :: 'a extN where bot-extN ≡ Bot
definition one-extN :: 'a extN where one-extN ≡ Top
definition top-extN :: 'a extN where top-extN ≡ Top
```

fun *less-eq-extN* :: *'a extN* ⇒ *'a extN* ⇒ *bool* **where**
  less-eq-extN Bot - = *True*
| *less-eq-extN* (*Val* -) *Bot* = *False*
| *less-eq-extN* (*Val x*) (*Val y*) = (*x* ≤ *y*)
| *less-eq-extN* (*Val* -) (*N* -) = *True*
| *less-eq-extN* (*Val* -) *Top* = *True*
| *less-eq-extN* (*N* -) *Bot* = *False*
| *less-eq-extN* (*N* -) (*Val* -) = *False*
| *less-eq-extN* (*N x*) (*N y*) = (*x* ≤ *y*)
| *less-eq-extN* (*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 extN* ⇒ *'a extN* ⇒ *bool* **where** *less-extN x y* = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*)

instance
proof
  fix *x y z* :: *'a extN*
  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* ≤ *y* ∧ ¬ *y* ≤ *x*)
    by *simp*
  show *x* ≤ *x*
    by (*cases x*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *x*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ⊓ *y* ≤ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *x* ≤ *y* ⟹ *x* ≤ *z* ⟹ *x* ≤ *y* ⊓ *z*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *x* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⊔ *y*
    by (*cases x*; *cases y*) *simp-all*
  show *y* ≤ *x* ⟹ *z* ≤ *x* ⟹ *y* ⊔ *z* ≤ *x*
    by (*cases x*; *cases y*; *cases z*) *simp-all*
  show *bot* ≤ *x*
    by (*simp add*: *bot-extN-def*)
  show *x* ≤ *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-extN-def*)
show $x + y = bot \longrightarrow x = bot$
by (*cases x*; *cases y*) (*simp-all add*: *bot-extN-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-extN-def top-extN-def*)
show $(1::'a\ extN) = top$
by (*simp add*: *one-extN-def top-extN-def*)
show $x * y = x \sqcap y$
by *simp*
show $x^T = x$
by *simp*
show $x^\star = top$
by (*simp add*: *top-extN-def*)
qed

end

end

6 Hoare Logic for Total Correctness

theory *Hoare-Logic*
imports *Main*
begin

This theory is based on Isabelle/HOL's *Hoare/Hoare-Logic.thy* written by L. P. Nieto and T. Nipkow. We have extended it to total-correctness proofs. We added corresponding modifications to *hoare-syntax.ML* and *hoare-tac.ML*.

type-synonym *'a bexp* = *'a set*
type-synonym *'a assn* = *'a set*
type-synonym *'a var* = *'a* ⇒ *nat*

datatype *'a com* =
Basic 'a ⇒ *'a*
| *Seq 'a com 'a com* ((-;/ -) [61,60] 60)
| *Cond 'a bexp 'a com 'a com* ((1IF -/ THEN - / ELSE -/ FI) [0,0,0] 61)
| *While 'a bexp 'a assn 'a var 'a com* ((1WHILE -/ INV {-} / VAR {-} //DO - /OD) [0,0,0,0] 61)

syntax (*xsymbols*)
-whilePC :: *'a bexp* ⇒ *'a assn* ⇒ *'a var* ⇒ *'a com* ⇒ *'a com* ((1WHILE -/ INV {-} //DO - /OD) [0,0,0] 61)

translations
WHILE b INV {x} DO c OD => WHILE b INV {x} VAR {0} DO c OD

abbreviation *annskip* (*SKIP*) **where** *SKIP == Basic id*

type-synonym *'a sem = 'a => 'a => bool*

inductive *Sem :: 'a com ⇒ 'a sem*
where
Sem (Basic f) s (f s)
| *Sem c1 s s'' ⟹ Sem c2 s'' s' ⟹ Sem (c1;c2) s s'*
| *s ∈ b ⟹ Sem c1 s s' ⟹ Sem (IF b THEN c1 ELSE c2 FI) s s'*
| *s ∉ b ⟹ Sem c2 s s' ⟹ Sem (IF b THEN c1 ELSE c2 FI) s s'*
| *s ∉ b ⟹ Sem (While b x y c) s s*
| *s ∈ b ⟹ Sem c s s'' ⟹ Sem (While b x y c) s'' s' ⟹*
Sem (While b x y c) s s'

inductive-cases [*elim*!]:
Sem (Basic f) s s' Sem (c1;c2) s s'
Sem (IF b THEN c1 ELSE c2 FI) s s'

lemma *Sem-deterministic*:
assumes *Sem c s s1*
and *Sem c s s2*
shows *s1 = s2*
proof −
have *Sem c s s1 ⟹ (∀ s2. Sem c s s2 ⟶ s1 = s2)*
by (*induct rule*: *Sem.induct*) (*subst Sem.simps*, *blast*)+
thus *?thesis*
using *assms* **by** *simp*
qed

definition *Valid :: 'a bexp ⇒ 'a com ⇒ 'a bexp ⇒ bool*
where *Valid p c q ⟷ (!s s'. Sem c s s' −−> s : p −−> s' : q)*

definition *ValidTC :: 'a bexp ⇒ 'a com ⇒ 'a bexp ⇒ bool*
where *ValidTC p c q ≡ ∀ s . s ∈ p ⟶ (∃ t . Sem c s t ∧ t ∈ q)*

lemma *tc-implies-pc*:
ValidTC p c q ⟹ Valid p c q
by (*metis Sem-deterministic Valid-def ValidTC-def*)

lemma *tc-extract-function*:
ValidTC p c q ⟹ ∃ f . ∀ s . s ∈ p ⟶ f s ∈ q
by (*metis ValidTC-def*)

syntax
-assign :: idt => 'b => 'a com ((*2- :=/ -*) [*70*, *65*] *61*)

syntax
-hoare-vars :: [*idts*, *'a assn*,*'a com*,*'a assn*] => *bool*
(*VARS -// {-} // - // {-} [0,0,55,0] 50*)
syntax (**output**)
-hoare :: [*'a assn*,*'a com*,*'a assn*] => *bool*
({-} // - // {-} [*0*,*55*,*0*] *50*)

ML-file *hoare-syntax.ML*
parse-translation ‹[(@{*syntax-const -hoare-vars*}, *K Hoare-Syntax.hoare-vars-tr*)]›
print-translation ‹[(@{*const-syntax Valid*}, *K* (*Hoare-Syntax.spec-tr' @{syntax-const -hoare}*))]›

syntax
-hoare-tc-vars :: [*idts*, *'a assn*,*'a com*,*'a assn*] => *bool*
(*VARS -// [-] // - // [-] [0,0,55,0] 50*)
syntax (**output**)
-hoare-tc :: [*'a assn*,*'a com*,*'a assn*] => *bool*
([-] // - // [-] [*0*,*55*,*0*] *50*)

parse-translation ‹[(@{*syntax-const -hoare-tc-vars*}, *K Hoare-Syntax.hoare-tc-vars-tr*)]›
print-translation ‹[(@{*const-syntax ValidTC*}, *K* (*Hoare-Syntax.spec-tr' @{syntax-const -hoare-tc}*))]›

lemma *SkipRule*: $p \subseteq q \Longrightarrow$ *Valid p* (*Basic id*) *q*
by (*auto simp*:*Valid-def*)

lemma *BasicRule*: $p \subseteq \{s.\ f\ s \in q\} \Longrightarrow$ *Valid p* (*Basic f*) *q*
by (*auto simp*:*Valid-def*)

lemma *SeqRule*: *Valid P c1 Q* $\Longrightarrow$ *Valid Q c2 R* $\Longrightarrow$ *Valid P* (*c1*;*c2*) *R*
by (*auto simp*:*Valid-def*)

lemma *CondRule*:
$p \subseteq \{s.\ (s \in b \longrightarrow s \in w) \wedge (s \notin b \longrightarrow s \in w')\}$
$\Longrightarrow$ *Valid w c1 q* $\Longrightarrow$ *Valid w' c2 q* $\Longrightarrow$ *Valid p* (*Cond b c1 c2*) *q*
by (*auto simp*:*Valid-def*)

lemma *While-aux*:
assumes *Sem* (*WHILE b INV* {*i*} *VAR* {*v*} *DO c OD*) *s s'*
shows $\forall s\ s'.$ *Sem c s s'* $\longrightarrow s \in I \wedge s \in b \longrightarrow s' \in I \Longrightarrow$
$s \in I \Longrightarrow s' \in I \wedge s' \notin b$
using *assms*
by (*induct WHILE b INV* {*i*} *VAR* {*v*} *DO c OD s s'*) *auto*

lemma *WhileRule*:

$p \subseteq i \Longrightarrow Valid\ (i \cap b)\ c\ i \Longrightarrow i \cap (-b) \subseteq q \Longrightarrow Valid\ p\ (While\ b\ i\ v\ c)\ q$
apply (*clarsimp simp*:*Valid-def*)
apply(*drule While-aux*)
apply *assumption*
apply *blast*
apply *blast*
done

lemma *SkipRuleTC*:
assumes $p \subseteq q$
shows *ValidTC p* (*Basic id*) *q*
by (*metis assms Sem.intros*(*1*) *ValidTC-def id-apply set-mp*)

lemma *BasicRuleTC*:
assumes $p \subseteq \{s.\ f\ s \in q\}$
shows *ValidTC p* (*Basic f*) *q*
by (*metis assms Ball-Collect Sem.intros*(*1*) *ValidTC-def*)

lemma *SeqRuleTC*:
assumes *ValidTC p c1 q*
and *ValidTC q c2 r*
shows *ValidTC p* (*c1*;*c2*) *r*
by (*meson assms Sem.intros*(*2*) *ValidTC-def*)

lemma *CondRuleTC*:
assumes $p \subseteq \{s.\ (s \in b \longrightarrow s \in w) \wedge (s \notin b \longrightarrow s \in w')\}$
and *ValidTC w c1 q*
and *ValidTC w' c2 q*
shows *ValidTC p* (*Cond b c1 c2*) *q*
proof (*unfold ValidTC-def*, *rule allI*)
fix *s*
show $s \in p \longrightarrow (\exists t\ .\ Sem\ (Cond\ b\ c1\ c2)\ s\ t \wedge t \in q)$
apply (*cases* $s \in b$)
apply (*metis* (*mono-tags*, *lifting*) *assms*(*1*,*2*) *Ball-Collect Sem.intros*(*3*) *ValidTC-def*)
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*,*3*) *Ball-Collect Sem.intros*(*4*) *ValidTC-def*)
qed

lemma *WhileRuleTC*:
assumes $p \subseteq i$
and $\bigwedge n{::}nat\ .\ ValidTC\ (i \cap b \cap \{s\ .\ v\ s = n\})\ c\ (i \cap \{s\ .\ v\ s < n\})$
and $i \cap uminus\ b \subseteq q$
shows *ValidTC p* (*While b i v c*) *q*
proof −
{
fix *s n*
have $s \in i \wedge v\ s = n \longrightarrow (\exists t\ .\ Sem\ (While\ b\ i\ v\ c)\ s\ t \wedge t \in q)$
proof (*induction n arbitrary*: *s rule*: *less-induct*)

fix $n :: nat$
fix $s :: 'a$
assume *1*: $\bigwedge(m::nat)\ s::'a\ .\ m < n \Longrightarrow s \in i \wedge v\ s = m \longrightarrow (\exists\, t\ .\ Sem\ (While\ b\ i\ v\ c)\ s\ t \wedge t \in q)$
show $s \in i \wedge v\ s = n \longrightarrow (\exists\, t\ .\ Sem\ (While\ b\ i\ v\ c)\ s\ t \wedge t \in q)$
proof (*rule impI*, *cases* $s \in b$)
assume *2*: $s \in b$ **and** $s \in i \wedge v\ s = n$
hence $s \in i \cap b \cap \{s\ .\ v\ s = n\}$
using *assms(1)* **by** *auto*
hence $\exists\, t\ .\ Sem\ c\ s\ t \wedge t \in i \cap \{s\ .\ v\ s < n\}$
by (*metis assms(2) ValidTC-def*)
from *this* **obtain** *t* **where** *3*: $Sem\ c\ s\ t \wedge t \in i \cap \{s\ .\ v\ s < n\}$
by *auto*
hence $\exists\, u\ .\ Sem\ (While\ b\ i\ v\ c)\ t\ u \wedge u \in q$
using *1* **by** *auto*
thus $\exists\, t\ .\ Sem\ (While\ b\ i\ v\ c)\ s\ t \wedge t \in q$
using *2 3 Sem.intros(6)* **by** *force*
next
assume $s \notin b$ **and** $s \in i \wedge v\ s = n$
thus $\exists\, t\ .\ Sem\ (While\ b\ i\ v\ c)\ s\ t \wedge t \in q$
using *Sem.intros(5) assms(3)* **by** *fastforce*
qed
qed
}
thus *?thesis*
using *assms(1) ValidTC-def* **by** *force*
qed

lemma *Compl-Collect*: $-(Collect\ b) = \{x.\ ^\sim(b\ x)\}$
by *blast*

lemmas *AbortRule* = *SkipRule* — dummy version
lemmas *AbortRuleTC* = *SkipRuleTC* — dummy version
ML-file *hoare-tac.ML*

method-setup *vcg* = ‹
Scan.succeed (*fn ctxt => SIMPLE-METHOD'* (*Hoare.hoare-tac ctxt* (*K all-tac*)))›
verification condition generator

method-setup *vcg-simp* = ‹
Scan.succeed (*fn ctxt =>*
SIMPLE-METHOD' (*Hoare.hoare-tac ctxt* (*asm-full-simp-tac ctxt*)))›
verification condition generator plus simplification

method-setup *vcg-tc* = ‹
Scan.succeed (*fn ctxt => SIMPLE-METHOD'* (*Hoare.hoare-tc-tac ctxt* (*K all-tac*)))›
verification condition generator

```
method-setup vcg-tc-simp = ‹
  Scan.succeed (fn ctxt =>
    SIMPLE-METHOD' (Hoare.hoare-tc-tac ctxt (asm-full-simp-tac ctxt)))›
  verification condition generator plus simplification

lemma multiply-by-add: VARS m s a b
  {a=A & b=B}
  m := 0; s := 0;
  WHILE m~=a
  INV {s=m*b & a=A & b=B & m≤a}
  DO s := s+b; m := m+(1::nat) OD
  {s = A*B}
by vcg-simp

lemma power: VARS (x::nat) n p i
  { 0 ≤ n }
  p := 1;
  i := 0;
  WHILE i < n
    INV { p = x^i ∧ i ≤ n }
     DO p := p * x;
        i := i + 1
     OD
  { p = x^n }
  apply vcg-simp
  by auto

lemma multiply-by-add-tc: VARS m s a b
  [a=A & b=B]
  m := 0; s := 0;
  WHILE m~=a
  INV {s=m*b & a=A & b=B & m≤a}
  VAR {a-m}
  DO s := s+b; m := m+(1::nat) OD
  [s = A*B]
apply vcg-tc-simp
by auto

lemma power-tc: VARS (x::nat) n p i
  [ 0 ≤ n ]
  p := 1;
  i := 0;
  WHILE i < n
    INV { p = x^i ∧ i ≤ n }
    VAR { n - i }
     DO p := p * x;
        i := i + 1
     OD
```

[$p = x\hat{}n$]
apply *vcg-tc*
by *auto*

end

7 Minimum Spanning Tree Algorithms

theory *Minimum-Spanning-Tree*

imports *Hoare-Logic Aggregation-Algebras*

begin

no-notation
trancl ((-$^+$) [*1000*] *999*)

context *m-kleene-algebra*
begin

7.1 Kruskal's Minimum Spanning Tree Algorithm

The total-correctness proof of Kruskal's minimum spanning tree algorithm uses the following steps. We first establish that the algorithm terminates and constructs a spanning tree. This is a constructive proof of the existence of a spanning tree; any spanning tree algorithm could be used for this. We then conclude that a minimum spanning tree exists. This is necessary to establish the invariant for the actual correctness proof, which shows that Kruskal's algorithm produces a minimum spanning tree.

definition *spanning-forest f g* $\equiv$ *forest f* $\wedge$ *f* $\leq$ $--g$ $\wedge$ *components g* $\leq$ *forest-components f* $\wedge$ *regular f*
definition *minimum-spanning-forest f g* $\equiv$ *spanning-forest f g* $\wedge$ ($\forall$ *u* . *spanning-forest u g* $\longrightarrow$ *sum* (*f* $\sqcap$ *g*) $\leq$ *sum* (*u* $\sqcap$ *g*))
definition *kruskal-spanning-invariant f g h* $\equiv$ *symmetric g* $\wedge$ *h* $= h^T$ $\wedge$ *g* $\sqcap$ $--h$ $=$ *h* $\wedge$ *spanning-forest f* ($-h$ $\sqcap$ *g*)
definition *kruskal-invariant f g h* $\equiv$ *kruskal-spanning-invariant f g h* $\wedge$ ($\exists$ *w* . *minimum-spanning-forest w g* $\wedge$ *f* $\leq$ *w* $\sqcup$ w^T)

We first show two verification conditions which are used in both correctness proofs.

lemma *kruskal-vc-1*:
assumes *symmetric g*
shows *kruskal-spanning-invariant bot g g*
proof (*unfold kruskal-spanning-invariant-def*, *intro conjI*)
show *symmetric g*
using *assms* **by** *simp*
next

show $g = g^T$
using *assms* **by** *simp*
next
show $g \sqcap --g = g$
using *inf.sup-monoid.add-commute selection-closed-id* **by** *simp*
next
show *spanning-forest bot* $(-g \sqcap g)$
using *star.circ-transitive-equal spanning-forest-def* **by** *simp*
qed

lemma *kruskal-vc-2*:
assumes *kruskal-spanning-invariant f g h*
and $h \neq bot$
and *card* $\{ x$. *regular* $x \wedge x \leq --h \}$ $= n$
shows (*minatom* $h \leq -$*forest-components* $f \longrightarrow$ *kruskal-spanning-invariant* $((f \sqcap -(top * minatom\ h * f^{T\star})) \sqcup (f \sqcap top * minatom\ h * f^{T\star})^T \sqcup minatom\ h)\ g\ (h \sqcap -minatom\ h \sqcap -minatom\ h^T)$
$\wedge$ *card* $\{ x$. *regular* $x \wedge x \leq --h \wedge x \leq -minatom\ h \wedge x \leq -minatom\ h^T \} < n) \wedge$
$(\neg$ *minatom* $h \leq -$*forest-components* $f \longrightarrow$ *kruskal-spanning-invariant* $f\ g\ (h \sqcap -minatom\ h \sqcap -minatom\ h^T)$
$\wedge$ *card* $\{ x$. *regular* $x \wedge x \leq --h \wedge x \leq -minatom\ h \wedge x \leq -minatom\ h^T \} < n)$
proof −
let *?e* $=$ *minatom h*
let *?f* $= (f \sqcap -(top * ?e * f^{T\star})) \sqcup (f \sqcap top * ?e * f^{T\star})^T \sqcup ?e$
let *?h* $= h \sqcap -?e \sqcap -?e^T$
let *?F* $=$ *forest-components f*
let *?u* $=$ *card* $\{ x$. *regular* $x \wedge x \leq --h \}$
let *?v* $=$ *card* $\{ x$. *regular* $x \wedge x \leq --h \wedge x \leq -?e \wedge x \leq -?e^T \}$
have *1*: *regular f* $\wedge$ *regular ?e*
by (*metis assms*(*1*) *kruskal-spanning-invariant-def spanning-forest-def minatom-regular*)
hence *2*: *regular ?f* $\wedge$ *regular ?F* $\wedge$ *regular* $(?e^T)$
using *regular-closed-star regular-conv-closed regular-mult-closed* **by** *simp*
have *3*: $\neg$ *?e* $\leq -?e$
using *assms*(*2*) *inf.orderE minatom-bot-iff* **by** *fastforce*
have *?v* $<$ *?u*
apply (*rule psubset-card-mono*)
using *finite-regular* **apply** *simp*
using *1 3 kruskal-spanning-invariant-def minatom-below* **by** *auto*
hence *4*: *?v* $< n$
using *assms*(*3*) **by** *simp*
show (*?e* $\leq -?F \longrightarrow$ *kruskal-spanning-invariant ?f g ?h* $\wedge$ *?v* $< n) \wedge (\neg$ *?e* $\leq -?F \longrightarrow$ *kruskal-spanning-invariant f g ?h* $\wedge$ *?v* $< n)$
proof (*rule conjI*)
have *5*: *injective ?f*
apply (*rule kruskal-injective-inv*)
using *assms*(*1*) *kruskal-spanning-invariant-def spanning-forest-def* **apply**

simp
apply (*simp add*: *covector-mult-closed*)
apply (*simp add*: *comp-associative comp-isotone star.right-plus-below-circ*)
apply (*meson mult-left-isotone order-lesseq-imp star-outer-increasing top.extremum*)
using *assms*(*1*,*2*) *kruskal-spanning-invariant-def kruskal-injective-inv-2 minatom-atom spanning-forest-def* **apply** *simp*
using *assms*(*2*) *atom-injective minatom-atom* **apply** *blast*
using *assms*(*1*,*2*) *kruskal-spanning-invariant-def kruskal-injective-inv-3 minatom-atom spanning-forest-def* **by** *simp*
show *?e* $\leq -$*?F* $\longrightarrow$ *kruskal-spanning-invariant ?f g ?h* $\wedge$ *?v* $<$ *n*
proof
assume *6*: *?e* $\leq -$*?F*
have *7*: *equivalence ?F*
using *assms*(*1*) *kruskal-spanning-invariant-def forest-components-equivalence spanning-forest-def* **by** *simp*
have $?e^T * top * ?e^T = ?e^T$
using *assms*(*2*) **by** (*simp add*: *atom-top-atom minatom-atom*)
hence $?e^T * top * ?e^T \leq -?F$
using *6 7 conv-complement conv-isotone* **by** *fastforce*
hence *8*: *?e* $*$ *?F* $*$ *?e* $=$ *bot*
using *le-bot triple-schroeder-p* **by** *simp*
show *kruskal-spanning-invariant ?f g ?h* $\wedge$ *?v* $<$ *n*
proof (*unfold kruskal-spanning-invariant-def*, *intro conjI*)
show *symmetric g*
using *assms*(*1*) *kruskal-spanning-invariant-def* **by** *simp*
next
show $?h = ?h^T$
using *assms*(*1*) **by** (*simp add*: *conv-complement conv-dist-inf inf-commute inf-left-commute kruskal-spanning-invariant-def*)
next
show *g* $\sqcap$ $--$*?h* $=$ *?h*
using *1 2* **by** (*metis* (*hide-lams*) *assms*(*1*) *kruskal-spanning-invariant-def inf-assoc pp-dist-inf*)
next
show *spanning-forest ?f* ($-$*?h* $\sqcap$ *g*)
proof (*unfold spanning-forest-def*, *intro conjI*)
show *injective ?f*
using *5* **by** *simp*
next
show *acyclic ?f*
apply (*rule kruskal-acyclic-inv*)
using *assms*(*1*) *kruskal-spanning-invariant-def spanning-forest-def* **apply** *simp*
apply (*simp add*: *covector-mult-closed*)
using *8 assms*(*1*) *kruskal-spanning-invariant-def spanning-forest-def kruskal-acyclic-inv-1* **apply** *simp*
using *8* **apply** (*metis comp-associative mult-left-sub-dist-sup-left star.circ-loop-fixpoint sup-commute le-bot*)

```
          using 6 by (simp add: p-antitone-iff)
        next
          show ?f ≤ −−(−?h ⊓ g)
            apply (rule kruskal-subgraph-inv)
            using assms(1) kruskal-spanning-invariant-def spanning-forest-def
apply simp
            using assms(1) apply (metis kruskal-spanning-invariant-def
minatom-below order.trans pp-isotone-inf)
            using assms(1) kruskal-spanning-invariant-def apply simp
            using assms(1) kruskal-spanning-invariant-def by simp
        next
          show components (−?h ⊓ g) ≤ forest-components ?f
            apply (rule kruskal-spanning-inv)
            using 5 apply simp
            using 1 regular-closed-star regular-conv-closed regular-mult-closed
apply simp
            using 1 apply simp
            using assms(1) kruskal-spanning-invariant-def spanning-forest-def by
simp
        next
          show regular ?f
            using 2 by simp
        qed
      next
        show ?v < n
          using 4 by simp
      qed
    qed
  next
    show ¬ ?e ≤ −?F ⟶ kruskal-spanning-invariant f g ?h ∧ ?v < n
    proof
      assume ¬ ?e ≤ −?F
      hence 9: ?e ≤ ?F
        using 2 assms(2) atom-in-partition minatom-atom by fastforce
      show kruskal-spanning-invariant f g ?h ∧ ?v < n
      proof (unfold kruskal-spanning-invariant-def, intro conjI)
        show symmetric g
          using assms(1) kruskal-spanning-invariant-def by simp
      next
        show ?h = ?h^T
          using assms(1) by (simp add: conv-complement conv-dist-inf
inf-commute inf-left-commute kruskal-spanning-invariant-def)
      next
        show g ⊓ −−?h = ?h
          using 1 2 by (metis (hide-lams) assms(1) kruskal-spanning-invariant-def
inf-assoc pp-dist-inf)
      next
        show spanning-forest f (−?h ⊓ g)
        proof (unfold spanning-forest-def, intro conjI)
```

```
        show injective f
          using assms(1) kruskal-spanning-invariant-def spanning-forest-def by
simp
      next
        show acyclic f
          using assms(1) kruskal-spanning-invariant-def spanning-forest-def by
simp
      next
        have f ≤ −−(−h ⊓ g)
          using assms(1) kruskal-spanning-invariant-def spanning-forest-def by
simp
        also have ... ≤ −−(−?h ⊓ g)
          using comp-inf.mult-right-isotone inf.sup-monoid.add-commute
inf-left-commute p-antitone-inf pp-isotone by presburger
        finally show f ≤ −−(−?h ⊓ g)
          by simp
      next
        show components (−?h ⊓ g) ≤ ?F
          apply (rule kruskal-spanning-inv-1)
          using 9 apply simp
          using 1 apply simp
          using assms(1) kruskal-spanning-invariant-def spanning-forest-def
apply simp
          using assms(1) kruskal-spanning-invariant-def
forest-components-equivalence spanning-forest-def by simp
      next
        show regular f
          using 1 by simp
      qed
    next
      show ?v < n
        using 4 by simp
    qed
  qed
 qed
qed
```

The following result shows that Kruskal's algorithm terminates and constructs a spanning tree. We cannot yet show that this is a minimum spanning tree.

```
theorem kruskal-spanning:
  VARS e f h
  [ symmetric g ]
  f := bot;
  h := g;
  WHILE h ≠ bot
    INV { kruskal-spanning-invariant f g h }
    VAR { card { x . regular x ∧ x ≤ −−h } }
     DO e := minatom h;
```

```
      IF e ≤ −forest-components f THEN
        f := (f ⊓ −(top ∗ e ∗ f^T⋆)) ⊔ (f ⊓ top ∗ e ∗ f^T⋆)^T ⊔ e
      ELSE
        SKIP
      FI;
      h := h ⊓ −e ⊓ −e^T
    OD
  [ spanning-forest f g ]
  apply vcg-tc-simp
  using kruskal-vc-1 apply simp
  using kruskal-vc-2 apply blast
  using kruskal-spanning-invariant-def by auto
```

Because we have shown total correctness, we conclude that a spanning tree exists.

```
lemma kruskal-exists-spanning:
  symmetric g ⟹ ∃f . spanning-forest f g
  using tc-extract-function kruskal-spanning by blast
```

This implies that a minimum spanning tree exists, which is used in the subsequent correctness proof.

```
lemma kruskal-exists-minimal-spanning:
  assumes symmetric g
    shows ∃f . minimum-spanning-forest f g
proof −
  let ?s = { f . spanning-forest f g }
  have ∃m∈?s . ∀z∈?s . sum (m ⊓ g) ≤ sum (z ⊓ g)
    apply (rule finite-set-minimal)
    using finite-regular spanning-forest-def apply simp
    using assms kruskal-exists-spanning apply simp
    using sum-linear by simp
  thus ?thesis
    using minimum-spanning-forest-def by simp
qed
```

Kruskal's minimum spanning tree algorithm terminates and is correct. This is the same algorithm that is used in the previous correctness proof, with the same precondition and variant, but with different invariant and postcondition.

```
theorem kruskal:
  VARS e f h
  [ symmetric g ]
  f := bot;
  h := g;
  WHILE h ≠ bot
    INV { kruskal-invariant f g h }
    VAR { card { x . regular x ∧ x ≤ −−h } }
    DO e := minatom h;
      IF e ≤ −forest-components f THEN
```

$f := (f \sqcap -(top * e * f^{T\star})) \sqcup (f \sqcap top * e * f^{T\star})^T \sqcup e$
ELSE
SKIP
FI;
$h := h \sqcap -e \sqcap -e^T$
OD
[*minimum-spanning-forest f g*]
proof *vcg-tc-simp*
assume *symmetric g*
thus *kruskal-invariant bot g g*
using *kruskal-vc-1 kruskal-exists-minimal-spanning kruskal-invariant-def* **by** *simp*
next
fix *n f h*
let *?e* = *minatom h*
let *?f* = $(f \sqcap -(top * ?e * f^{T\star})) \sqcup (f \sqcap top * ?e * f^{T\star})^T \sqcup ?e$
let *?h* = $h \sqcap -?e \sqcap -?e^T$
let *?F* = *forest-components f*
let *?u* = *card* { *x* . *regular x* $\wedge$ $x \leq --h$ }
let *?v* = *card* { *x* . *regular x* $\wedge$ $x \leq --h \wedge x \leq -?e \wedge x \leq -?e^T$ }
assume *1*: *kruskal-invariant f g h* $\wedge$ $h \neq bot \wedge ?u = n$
from *1* **obtain** *w* **where** *2*: *minimum-spanning-forest w g* $\wedge$ $f \leq w \sqcup w^T$
using *kruskal-invariant-def* **by** *auto*
hence *3*: *regular f* $\wedge$ *regular w* $\wedge$ *regular ?e*
using *1* **by** (*metis kruskal-invariant-def kruskal-spanning-invariant-def minimum-spanning-forest-def spanning-forest-def minatom-regular*)
show ($?e \leq -?F \longrightarrow$ *kruskal-invariant ?f g ?h* $\wedge$ $?v < n$) $\wedge$ ($\neg$ $?e \leq -?F \longrightarrow$ *kruskal-invariant f g ?h* $\wedge$ $?v < n$)
proof (*rule conjI*)
show $?e \leq -?F \longrightarrow$ *kruskal-invariant ?f g ?h* $\wedge$ $?v < n$
proof
assume *4*: $?e \leq -?F$
have *5*: *equivalence ?F*
using *1 kruskal-invariant-def kruskal-spanning-invariant-def forest-components-equivalence spanning-forest-def* **by** *simp*
have $?e^T * top * ?e^T = ?e^T$
using *1* **by** (*simp add*: *atom-top-atom minatom-atom*)
hence $?e^T * top * ?e^T \leq -?F$
using *4 5 conv-complement conv-isotone* **by** *fastforce*
hence *6*: $?e * ?F * ?e = bot$
using *le-bot triple-schroeder-p* **by** *simp*
show *kruskal-invariant ?f g ?h* $\wedge$ $?v < n$
proof (*unfold kruskal-invariant-def*, *intro conjI*)
show *kruskal-spanning-invariant ?f g ?h*
using *1 4 kruskal-vc-2 kruskal-invariant-def* **by** *simp*
next
show $\exists\, w$. *minimum-spanning-forest w g* $\wedge$ $?f \leq w \sqcup w^T$
proof
let *?p* = $w \sqcap top * ?e * w^{T\star}$

let *?v* = $(w \sqcap -(top * ?e * w^{T\star})) \sqcup ?p^T$
have *7*: *regular ?p*
using *3 regular-closed-star regular-conv-closed regular-mult-closed* **by** *simp*
have *8*: *injective ?v*
apply (*rule kruskal-exchange-injective-inv-1*)
using *2 minimum-spanning-forest-def spanning-forest-def* **apply** *simp*
apply (*simp add*: *covector-mult-closed*)
apply (*simp add*: *comp-associative comp-isotone star.right-plus-below-circ*)
using *1 2 kruskal-injective-inv-3 minatom-atom minimum-spanning-forest-def spanning-forest-def* **by** *simp*
have *9*: *components g* $\leq$ *forest-components ?v*
apply (*rule kruskal-exchange-spanning-inv-1*)
using *8* **apply** *simp*
using *7* **apply** *simp*
using *2 minimum-spanning-forest-def spanning-forest-def* **by** *simp*
have *10*: *spanning-forest ?v g*
proof (*unfold spanning-forest-def*, *intro conjI*)
show *injective ?v*
using *8* **by** *simp*
next
show *acyclic ?v*
apply (*rule kruskal-exchange-acyclic-inv-1*)
using *2 minimum-spanning-forest-def spanning-forest-def* **apply** *simp*
by (*simp add*: *covector-mult-closed*)
next
show *?v* $\leq --g$
apply (*rule sup-least*)
using *2 inf.coboundedI1 minimum-spanning-forest-def spanning-forest-def* **apply** *simp*
using *1 2* **by** (*metis kruskal-invariant-def kruskal-spanning-invariant-def conv-complement conv-dist-inf order.trans inf.absorb2 inf.cobounded1 minimum-spanning-forest-def spanning-forest-def*)
next
show *components g* $\leq$ *forest-components ?v*
using *9* **by** *simp*
next
show *regular ?v*
using *3 regular-closed-star regular-conv-closed regular-mult-closed* **by** *simp*
qed
have *11*: *sum* (*?v* $\sqcap$ *g*) = *sum* (*w* $\sqcap$ *g*)
proof −
have *sum* (*?v* $\sqcap$ *g*) = *sum* ($w \sqcap -(top * ?e * w^{T\star}) \sqcap g$) + *sum* ($?p^T \sqcap g$)
using *2* **by** (*metis conv-complement conv-top epm-8 inf-import-p inf-top-right regular-closed-top vector-top-closed minimum-spanning-forest-def spanning-forest-def sum-disjoint*)

also have ... = *sum* $(w \sqcap -(top * ?e * w^{T\star}) \sqcap g) + sum\ (?p \sqcap g)$
using *1 kruskal-invariant-def kruskal-spanning-invariant-def sum-symmetric* **by** *simp*
also have ... = *sum* $(((w \sqcap -(top * ?e * w^{T\star})) \sqcup ?p) \sqcap g)$
using *inf-commute inf-left-commute sum-disjoint* **by** *simp*
also have ... = *sum* $(w \sqcap g)$
using *3 7 maddux-3-11-pp* **by** *simp*
finally show *?thesis*
by *simp*
qed
have *12*: $?v \sqcup ?v^T = w \sqcup w^T$
proof −
have $?v \sqcup ?v^T = (w \sqcap -?p) \sqcup ?p^T \sqcup (w^T \sqcap -?p^T) \sqcup ?p$
using *conv-complement conv-dist-inf conv-dist-sup inf-import-p sup-assoc* **by** *simp*
also have ... = $w \sqcup w^T$
using *3 7 conv-complement conv-dist-inf inf-import-p maddux-3-11-pp sup-monoid.add-assoc sup-monoid.add-commute* **by** *simp*
finally show *?thesis*
by *simp*
qed
have *13*: $?v * ?e^T = bot$
apply (*rule kruskal-reroot-edge*)
using *1* **apply** (*simp add*: *minatom-atom*)
using *2 minimum-spanning-forest-def spanning-forest-def* **by** *simp*
have $?v \sqcap ?e \leq ?v \sqcap top * ?e$
using *inf.sup-right-isotone top-left-mult-increasing* **by** *simp*
also have ... $\leq ?v * (top * ?e)^T$
using *covector-restrict-comp-conv covector-mult-closed vector-top-closed* **by** *simp*
finally have *14*: $?v \sqcap ?e = bot$
using *13* **by** (*metis conv-dist-comp mult-assoc le-bot mult-left-zero*)
let $?d = ?v \sqcap top * ?e^T * ?v^{T\star} \sqcap ?F * ?e^T * top \sqcap top * ?e * -?F$
let $?w = (?v \sqcap -?d) \sqcup ?e$
have *15*: *regular ?d*
using *3 regular-closed-star regular-conv-closed regular-mult-closed* **by** *simp*
have *16*: $?F \leq -?d$
apply (*rule kruskal-edge-between-components-1*)
using *5* **apply** *simp*
using *1 conv-dist-comp minatom-atom mult-assoc* **by** *simp*
have *17*: $f \sqcup f^T \leq (?v \sqcap -?d \sqcap -?d^T) \sqcup (?v^T \sqcap -?d \sqcap -?d^T)$
apply (*rule kruskal-edge-between-components-2*)
using *16* **apply** *simp*
using *1 kruskal-invariant-def kruskal-spanning-invariant-def spanning-forest-def* **apply** *simp*
using *2 12* **by** (*metis conv-dist-sup conv-involutive conv-isotone le-supI sup-commute*)
show *minimum-spanning-forest ?w g* $\wedge$ $?f \leq ?w \sqcup ?w^T$

```
        proof (intro conjI)
          have 18: ?e^T ≤ ?v^⋆
            apply (rule kruskal-edge-atom-1[where g=g and h=h])
            using minatom-below apply simp
            using 1 apply (metis kruskal-invariant-def
kruskal-spanning-invariant-def inf-le1)
            using 1 kruskal-invariant-def kruskal-spanning-invariant-def apply
simp
            using 9 apply simp
            using 13 by simp
          have 19: atom ?d
            apply (rule kruskal-edge-atom)
            using 5 apply simp
            using 10 spanning-forest-def apply blast
            using 1 apply (simp add: minatom-atom)
            using 3 apply (metis conv-complement pp-dist-star
regular-mult-closed)
            using 2 8 12 apply (simp add: kruskal-forest-components-inf)
            using 10 spanning-forest-def apply simp
            using 13 apply simp
            using 6 apply simp
            using 18 by simp
          show minimum-spanning-forest ?w g
          proof (unfold minimum-spanning-forest-def, intro conjI)
            have (?v ⊓ −?d) ∗ ?e^T ≤ ?v ∗ ?e^T
              using inf-le1 mult-left-isotone by simp
            hence (?v ⊓ −?d) ∗ ?e^T = bot
              using 13 le-bot by simp
            hence 20: ?e ∗ (?v ⊓ −?d)^T = bot
              using conv-dist-comp conv-involutive conv-bot by force
            have 21: injective ?w
              apply (rule injective-sup)
              using 8 apply (simp add: injective-inf-closed)
              using 20 apply simp
              using 1 atom-injective minatom-atom by blast
            show spanning-forest ?w g
            proof (unfold spanning-forest-def, intro conjI)
              show injective ?w
                using 21 by simp
              next
              show acyclic ?w
                apply (rule kruskal-exchange-acyclic-inv-2)
                using 10 spanning-forest-def apply blast
                using 8 apply simp
                using inf.coboundedI1 apply simp
                using 19 apply simp
                using 1 apply (simp add: minatom-atom)
                using inf.cobounded2 inf.coboundedI1 apply simp
                using 13 by simp
```

```
      next
        have ?w ≤ ?v ⊔ ?e
          using inf-le1 sup-left-isotone by simp
        also have ... ≤ −−g ⊔ ?e
          using 10 sup-left-isotone spanning-forest-def by blast
        also have ... ≤ −−g ⊔ −−h
          by (simp add: le-supI2 minatom-below)
        also have ... = −−g
          using 1 by (metis kruskal-invariant-def
kruskal-spanning-invariant-def pp-isotone-inf sup.orderE)
        finally show ?w ≤ −−g
          by simp
      next
        have 22: ?d ≤ (?v ⊓ −?d)^T⋆ ∗ ?e^T ∗ top
          apply (rule kruskal-exchange-spanning-inv-2)
          using 8 apply simp
          using 13 apply (metis semiring.mult-not-zero star-absorb
star-simulation-right-equal)
          using 17 apply simp
          by (simp add: inf.coboundedI1)
        have components g ≤ forest-components ?v
          using 10 spanning-forest-def by auto
        also have ... ≤ forest-components ?w
          apply (rule kruskal-exchange-forest-components-inv)
          using 21 apply simp
          using 15 apply simp
          using 1 apply (simp add: atom-top-atom minatom-atom)
          apply (simp add: inf.coboundedI1)
          using 13 apply simp
          using 8 apply simp
          apply (simp add: le-infI1)
          using 22 by simp
        finally show components g ≤ forest-components ?w
          by simp
      next
        show regular ?w
          using 3 7 regular-conv-closed by simp
      qed
    next
      have 23: ?e ⊓ g ≠ bot
        using 1 by (metis kruskal-invariant-def
kruskal-spanning-invariant-def comp-inf.semiring.mult-zero-right
inf.sup-monoid.add-assoc inf.sup-monoid.add-commute minatom-bot-iff
minatom-meet-bot)
      have g ⊓ −h ≤ (g ⊓ −h)⋆
        using star.circ-increasing by simp
      also have ... ≤ (−−(g ⊓ −h))⋆
        using pp-increasing star-isotone by blast
      also have ... ≤ ?F
```

using *1 kruskal-invariant-def kruskal-spanning-invariant-def inf.sup-monoid.add-commute spanning-forest-def* **by** *simp*
finally have *24*: $g \sqcap -h \leq ?F$
by *simp*
have $?d \leq --g$
using *10 inf.coboundedI1 spanning-forest-def* **by** *blast*
hence $?d \leq --g \sqcap -?F$
using *16 inf.boundedI p-antitone-iff* **by** *simp*
also have $... = --(g \sqcap -?F)$
by *simp*
also have $... \leq --h$
using *24 p-shunting-swap pp-isotone* **by** *fastforce*
finally have *25*: $?d \leq --h$
by *simp*
have $?d = bot \longrightarrow top = bot$
using *19* **by** (*metis mult-left-zero mult-right-zero*)
hence $?d \neq bot$
using *1 le-bot* **by** *auto*
hence *26*: $?d \sqcap h \neq bot$
using *25* **by** (*metis inf.absorb-iff2 inf-commute pseudo-complement*)
have $sum\ (?e \sqcap g) = sum\ (?e \sqcap --h \sqcap g)$
by (*simp add*: *inf.absorb1 minatom-below*)
also have $... = sum\ (?e \sqcap h)$
using *1* **by** (*metis kruskal-invariant-def kruskal-spanning-invariant-def inf.left-commute inf.sup-monoid.add-commute*)
also have $... \leq sum\ (?d \sqcap h)$
using *19 26 minatom-min* **by** *simp*
also have $... = sum\ (?d \sqcap (--h \sqcap g))$
using *1 kruskal-invariant-def kruskal-spanning-invariant-def inf-commute* **by** *simp*
also have $... = sum\ (?d \sqcap g)$
using *25* **by** (*simp add*: *inf.absorb2 inf-assoc inf-commute*)
finally have *27*: $sum\ (?e \sqcap g) \leq sum\ (?d \sqcap g)$
by *simp*
have $?v \sqcap ?e \sqcap -?d = bot$
using *14* **by** *simp*
hence $sum\ (?w \sqcap g) = sum\ (?v \sqcap -?d \sqcap g) + sum\ (?e \sqcap g)$
using *sum-disjoint inf-commute inf-assoc* **by** *simp*
also have $... \leq sum\ (?v \sqcap -?d \sqcap g) + sum\ (?d \sqcap g)$
using *23 27 sum-plus-right-isotone* **by** *simp*
also have $... = sum\ (((?v \sqcap -?d) \sqcup ?d) \sqcap g)$
using *sum-disjoint inf-le2 pseudo-complement* **by** *simp*
also have $... = sum\ ((?v \sqcup ?d) \sqcap (-?d \sqcup ?d) \sqcap g)$
by (*simp add*: *sup-inf-distrib2*)
also have $... = sum\ ((?v \sqcup ?d) \sqcap g)$
using *15* **by** (*metis inf-top-right stone*)
also have $... = sum\ (?v \sqcap g)$
by (*simp add*: *inf.sup-monoid.add-assoc*)
finally have $sum\ (?w \sqcap g) \leq sum\ (?v \sqcap g)$

```
            by simp
          thus ∀ u . spanning-forest u g ⟶ sum (?w ⊓ g) ≤ sum (u ⊓ g)
            using 2 11 minimum-spanning-forest-def by auto
        qed
      next
        have ?f ≤ f ⊔ f^T ⊔ ?e
          using conv-dist-inf inf-le1 sup-left-isotone sup-mono by presburger
        also have ... ≤ (?v ⊓ −?d ⊓ −?d^T) ⊔ (?v^T ⊓ −?d ⊓ −?d^T) ⊔ ?e
          using 17 sup-left-isotone by simp
        also have ... ≤ (?v ⊓ −?d) ⊔ (?v^T ⊓ −?d ⊓ −?d^T) ⊔ ?e
          using inf.cobounded1 sup-inf-distrib2 by presburger
        also have ... = ?w ⊔ (?v^T ⊓ −?d ⊓ −?d^T)
          by (simp add: sup-assoc sup-commute)
        also have ... ≤ ?w ⊔ (?v^T ⊓ −?d^T)
          using inf.sup-right-isotone inf-assoc sup-right-isotone by simp
        also have ... ≤ ?w ⊔ ?w^T
          using conv-complement conv-dist-inf conv-dist-sup sup-right-isotone
by simp
        finally show ?f ≤ ?w ⊔ ?w^T
          by simp
      qed
    qed
  next
    show ?v < n
      using 1 kruskal-vc-2 kruskal-invariant-def by auto
  qed
 qed
next
 show ¬ ?e ≤ −?F ⟶ kruskal-invariant f g ?h ∧ ?v < n
  using 1 kruskal-vc-2 kruskal-invariant-def by auto
qed
next
 fix f g h
 assume 28: kruskal-invariant f g h ∧ h = bot
 hence 29: spanning-forest f g
  using kruskal-invariant-def kruskal-spanning-invariant-def by auto
 from 28 obtain w where 30: minimum-spanning-forest w g ∧ f ≤ w ⊔ w^T
  using kruskal-invariant-def by auto
 hence w = w ⊓ −−g
  by (simp add: inf.absorb1 minimum-spanning-forest-def spanning-forest-def)
 also have ... ≤ w ⊓ components g
  by (metis inf.sup-right-isotone star.circ-increasing)
 also have ... ≤ w ⊓ f^T⋆ ∗ f⋆
  using 29 spanning-forest-def inf.sup-right-isotone by simp
 also have ... ≤ f ⊔ f^T
  apply (rule cancel-separate-6[where z=w and y=w^T])
  using 30 minimum-spanning-forest-def spanning-forest-def apply simp
  using 30 apply (metis conv-dist-inf conv-dist-sup conv-involutive
inf.cobounded2 inf.orderE)
```

using *30* **apply** (*simp add*: *sup-commute*)
using *30 minimum-spanning-forest-def spanning-forest-def* **apply** *simp*
using *30* **by** (*metis acyclic-star-below-complement comp-inf.mult-right-isotone inf-p le-bot minimum-spanning-forest-def spanning-forest-def*)
finally have *31*: $w \leq f \sqcup f^T$
by *simp*
have *sum* $(f \sqcap g)$ = *sum* $((w \sqcup w^T) \sqcap (f \sqcap g))$
using *30* **by** (*metis inf-absorb2 inf.assoc*)
also have ... = *sum* $(w \sqcap (f \sqcap g))$ + *sum* $(w^T \sqcap (f \sqcap g))$
using *30 inf.commute acyclic-asymmetric sum-disjoint minimum-spanning-forest-def spanning-forest-def* **by** *simp*
also have ... = *sum* $(w \sqcap (f \sqcap g))$ + *sum* $(w \sqcap (f^T \sqcap g^T))$
by (*metis conv-dist-inf conv-involutive sum-conv*)
also have ... = *sum* $(f \sqcap (w \sqcap g))$ + *sum* $(f^T \sqcap (w \sqcap g))$
using *28 inf.commute inf.assoc kruskal-invariant-def kruskal-spanning-invariant-def* **by** *simp*
also have ... = *sum* $((f \sqcup f^T) \sqcap (w \sqcap g))$
using *29 acyclic-asymmetric inf.sup-monoid.add-commute sum-disjoint spanning-forest-def* **by** *simp*
also have ... = *sum* $(w \sqcap g)$
using *31* **by** (*metis inf-absorb2 inf.assoc*)
finally show *minimum-spanning-forest f g*
using *29 30 minimum-spanning-forest-def* **by** *simp*
qed

7.2 Prim's Minimum Spanning Tree Algorithm

abbreviation *component g r* $\equiv r^T * (--g)^\star$
definition *spanning-tree t g r* $\equiv$ *forest t* $\wedge$ $t \leq$ (*component g r*)^T $*$ (*component g r*) $\sqcap$ $--g$ $\wedge$ *component g r* $\leq r^T * t^\star$ $\wedge$ *regular t*
definition *minimum-spanning-tree t g r* $\equiv$ *spanning-tree t g r* $\wedge$ $(\forall u$. *spanning-tree u g r* $\longrightarrow$ *sum* $(t \sqcap g) \leq$ *sum* $(u \sqcap g))$
definition *precondition g r* $\equiv g = g^T$ $\wedge$ *injective r* $\wedge$ *vector r* $\wedge$ *regular r*
definition *invariant t v g r* $\equiv$ *precondition g r* $\wedge$ $v^T = r^T * t^\star$ $\wedge$ *spanning-tree t* $(v * v^T \sqcap g)$ *r* $\wedge$ $(\exists w$. *minimum-spanning-tree w g r* $\wedge$ $t \leq w)$

lemma *span-tree-split*:
assumes *vector r*
shows $t \leq$ (*component g r*)^T $*$ (*component g r*) $\sqcap$ $--g \longleftrightarrow$ ($t \leq$ (*component g r*)^T $\wedge$ $t \leq$ *component g r* $\wedge$ $t \leq --g$)
proof −
have (*component g r*)^T $*$ (*component g r*) = (*component g r*)^T $\sqcap$ *component g r*
by (*metis assms conv-involutive covector-mult-closed vector-conv-covector vector-covector*)
thus *?thesis*
by *simp*
qed

lemma *span-tree-component*:

assumes *spanning-tree t g r*
shows *component g r = component t r*
using *assms* **by** (*simp add*: *antisym mult-right-isotone star-isotone spanning-tree-def*)

Prim's minimum spanning tree algorithm is correct.

theorem *prim*:
VARS t v e
{ *precondition g r* ∧ (∃ *w* . *minimum-spanning-tree w g r*) }
t := *bot*;
v := *r*;
WHILE v ∗ −v^T ⊓ *g* ≠ *bot*
INV { *invariant t v g r* }
DO e := *minatom* (*v* ∗ −v^T ⊓ *g*);
t := *t* ⊔ *e*;
v := *v* ⊔ e^T ∗ *top*
OD
{ *minimum-spanning-tree t g r* }
proof *vcg-simp*
let *?ss* = *r* ∗ r^T ⊓ *g*
assume *1*: *precondition g r* ∧ (∃ *w* . *minimum-spanning-tree w g r*)
show *invariant bot r g r*
proof (*unfold invariant-def*, *intro conjI*)
show *precondition g r*
using *1* **by** *simp*
next
show r^T = r^T ∗ $bot^\star$
by (*simp add*: *star-absorb*)
next
show *spanning-tree bot ?ss r*
proof (*unfold spanning-tree-def*, *intro conjI*)
show *injective bot*
by *simp*
next
show *acyclic bot*
by *simp*
next
show *bot* ≤ (*component ?ss r*)^T ∗ (*component ?ss r*) ⊓ −− *?ss*
by *simp*
next
show *component ?ss r* ≤ r^T ∗ $bot^\star$
proof −
have *component ?ss r* ≤ *component* (*r* ∗ r^T) *r*
by (*simp add*: *mult-right-isotone star-isotone*)
also have ... ≤ r^T ∗ $1^\star$
using *1* **by** (*metis inf.eq-iff p-antitone regular-one-closed star-sub-one precondition-def*)
also have ... = r^T ∗ $bot^\star$
by (*simp add*: *star.circ-zero star-one*)

```
      finally show ?thesis
        .
    qed
  next
    show regular bot
      by simp
  qed
 next
  show (∃ w . minimum-spanning-tree w g r ∧ bot ≤ w)
    using 1 by simp
 qed
next
 fix t v
 let ?vcv = v ∗ −v^T ⊓ g
 let ?vv = v ∗ v^T ⊓ g
 let ?e = minatom ?vcv
 let ?c = component g r
 let ?g = −−g
 assume 2: invariant t v g r ∧ ?vcv ≠ bot
 have 3: regular v ∧ regular (v ∗ v^T) ∧ regular ((r^T ∗ ?g^⋆)^T ∗ (r^T ∗ ?g^⋆)) ∧
regular ((v ⊔ ?e^T ∗ top) ∗ (v ⊔ ?e^T ∗ top)^T)
   using 2 by (metis invariant-def spanning-tree-def precondition-def
regular-conv-closed regular-closed-star regular-mult-closed conv-involutive
regular-closed-pp regular-closed-top regular-closed-sup minatom-regular)
 have 4: t ≤ v ∗ v^T ⊓ ?g
   using 2 3 by (metis invariant-def spanning-tree-def inf-pp-commute
inf.boundedE)
 hence 5: t ≤ v ∗ v^T
   by simp
 have 6: t ≤ ?g
   using 4 by simp
 have 7: ?e ≤ v ∗ −v^T ⊓ ?g
   using 3 by (metis minatom-below pp-dist-inf regular-mult-closed
regular-closed-p)
 hence 8: ?e ≤ v ∗ −v^T
   by simp
 have 9: vector v
   using 2 invariant-def precondition-def by (simp add: covector-mult-closed
vector-conv-covector)
 have 10: ?e ∗ t = bot
   using 5 8 9 et(1) by blast
 have 11: ?e ∗ t^T = bot
   using 5 8 9 et(2) by simp
 have 12: atom ?e
   using 2 minatom-atom by simp
 have r^T ≤ r^T ∗ t^⋆
   by (metis mult-right-isotone order-refl semiring.mult-not-zero
star.circ-separate-mult-1 star-absorb)
 hence 13: r^T ≤ v^T
```

using *2* **by** (*simp add*: *invariant-def*)
from *2* **obtain** *w* **where** *14*: *minimum-spanning-tree w g r* $\wedge$ $t \leq w$
by (*metis invariant-def*)
have *15*: *vector r* $\wedge$ *injective r* $\wedge$ $v^T = r^T * t^\star$ $\wedge$ *forest w* $\wedge$ $t \leq w \wedge w \leq (r^T * ?g^\star)^T * (r^T * ?g^\star) \sqcap ?g \wedge r^T * ((r^T * ?g^\star)^T * (r^T * ?g^\star) \sqcap ?g)^\star \leq r^T * w^\star$
using *2 3 14 invariant-def precondition-def minimum-spanning-tree-def spanning-tree-def reachable-restrict* **by** *simp*
hence *16*: $w * v \leq v$
using *predecessors-reachable reachable-restrict* **by** *auto*
have *17*: $g = g^T$
using *2 invariant-def precondition-def* **by** *simp*
hence *18*: $?g^T = ?g$
using *conv-complement* **by** *simp*
show *invariant* $(t \sqcup ?e)$ $(v \sqcup ?e^T * top)$ *g r*
proof (*unfold invariant-def*, *intro conjI*)
show *precondition g r*
using *2 invariant-def* **by** *simp*
next
show $(v \sqcup ?e^T * top)^T = r^T * (t \sqcup ?e)^\star$
using *2 8 9 10* **by** (*simp add*: *reachable-inv invariant-def precondition-def spanning-tree-def*)
next
let $?v = v \sqcup ?e^T * top$
let $?G = ?v * ?v^T \sqcap g$
show *spanning-tree* $(t \sqcup ?e)$ *?G r*
proof (*unfold spanning-tree-def*, *intro conjI*)
show *injective* $(t \sqcup ?e)$
using *2 11 12* **by** (*simp add*: *injective-inv invariant-def spanning-tree-def*)
next
show *acyclic* $(t \sqcup ?e)$
using *2 5 8 9 acyclic-inv invariant-def spanning-tree-def* **by** *simp*
next
show $t \sqcup ?e \leq$ (*component ?G r*)$^T * ($*component ?G r*$) \sqcap --?G$
using *3 4 7 9 15 prim-subgraph-inv inf-pp-commute mst-subgraph-inv-2* **by** *auto*
next
show *component* $((v \sqcup ?e^T * top) * (v \sqcup ?e^T * top)^T \sqcap g)$ $r \leq r^T * (t \sqcup ?e)^\star$
proof −
have *19*: $r^T * ((v * v^T) \sqcap ?g)^\star \leq r^T * t^\star$
using *2 3* **by** (*metis invariant-def spanning-tree-def inf-pp-commute*)
have *component* $((v \sqcup ?e^T * top) * (v \sqcup ?e^T * top)^T \sqcap g)$ $r = r^T * ((v \sqcup ?e^T * top) * (v \sqcup ?e^T * top)^T \sqcap ?g)^\star$
using *3* **by** *simp*
also have $... \leq r^T * (t \sqcup ?e)^\star$
using *4 8 9 12 13 15 18 19* **by** (*metis span-inv*)
finally show *?thesis*
.
qed

```
    next
      show regular (t ⊔ ?e)
        using 2 by (metis invariant-def spanning-tree-def regular-closed-sup
minatom-regular)
    qed
  next
    show ∃w . minimum-spanning-tree w g r ∧ t ⊔ ?e ≤ w
    proof
      let ?f = w ⊓ v ∗ −v^T ⊓ top ∗ ?e ∗ w^T⋆
      let ?p = w ⊓ −v ∗ −v^T ⊓ top ∗ ?e ∗ w^T⋆
      let ?fp = w ⊓ −v^T ⊓ top ∗ ?e ∗ w^T⋆
      let ?w = (w ⊓ −?fp) ⊔ ?p^T ⊔ ?e
      have 20: regular ?f ∧ regular ?fp ∧ regular ?w
        using 3 14 by (metis regular-conv-closed regular-closed-star
regular-mult-closed regular-closed-top regular-closed-inf regular-closed-sup
minatom-regular minimum-spanning-tree-def spanning-tree-def)
      show minimum-spanning-tree ?w g r ∧ t ⊔ ?e ≤ ?w
      proof (intro conjI)
        show minimum-spanning-tree ?w g r
        proof (unfold minimum-spanning-tree-def, intro conjI)
          show spanning-tree ?w g r
          proof (unfold spanning-tree-def, intro conjI)
            show injective ?w
              using 8 9 12 15 exchange-injective by blast
          next
            show acyclic ?w
              using 8 9 15 16 exchange-acyclic by blast
          next
            show ?w ≤ ?c^T ∗ ?c ⊓ −−g
            proof −
              have 21: w ⊓ −?fp ≤ ?c^T ∗ ?c ⊓ −−g
                using 14 by (simp add: le-infI1 minimum-spanning-tree-def
spanning-tree-def)
              have ?p^T ≤ w^T
                by (simp add: conv-isotone inf.sup-monoid.add-assoc)
              also have ... ≤ (?c^T ∗ ?c ⊓ −−g)^T
                using 15 conv-order by simp
              also have ... = ?c^T ∗ ?c ⊓ −−g
                using 3 18 conv-dist-comp conv-dist-inf by simp
              finally have 22: ?p^T ≤ ?c^T ∗ ?c ⊓ −−g
                .
              have ?e ≤ (r^T ∗ ?g⋆)^T ∗ (r^T ∗ ?g⋆) ⊓ ?g
                using 6 7 15 mst-subgraph-inv by auto
              thus ?thesis
                using 21 22 by simp
            qed
          next
            show ?c ≤ r^T ∗ ?w⋆
            proof −
```

```
      have ?c ≤ r^T * w^⋆
        using 14 minimum-spanning-tree-def spanning-tree-def by simp
      also have ... ≤ r^T * ?w^⋆
        using 5 8 9 14 15 16 20 by (metis mst-reachable-inv)
      finally show ?thesis
        .
    qed
  next
    show regular ?w
      using 20 by simp
  qed
next
  have 23: ?f ⊔ ?p = ?fp
    using 3 9 epm-1 by fastforce
  have atom (w ⊓ −−v * −v^T ⊓ top * ?e * w^T⋆)
    using 6 7 9 12 15 16 reachable-restrict atom-edge by auto
  hence 24: atom ?f
    using 3 by simp
  hence ?f = bot ⟶ top = bot
    by (metis mult-left-zero mult-right-zero)
  hence ?f ≠ bot
    using 2 le-bot by auto
  hence ?f ⊓ v * −v^T ⊓ ?g ≠ bot
    using 3 15 by (simp add: inf.absorb1 le-infI1)
  hence g ⊓ (?f ⊓ v * −v^T) ≠ bot
    using inf-commute pp-inf-bot-iff by simp
  hence 25: ?f ⊓ ?vcv ≠ bot
    by (simp add: inf-assoc inf-commute)
  hence 26: ?f ⊓ g = ?f ⊓ ?vcv
    using 3 by (simp add: inf-assoc inf-commute inf-left-commute)
  have 27: ?e ⊓ g = minatom ?vcv ⊓ ?vcv
    using 8 by (simp add: inf.absorb2 inf.assoc inf.commute)
  hence 28: sum (?e ⊓ g) ≤ sum (?f ⊓ g)
    using 20 24 25 26 by (simp add: minatom-min)
  have ?e ≠ bot
    using 25 comp-inf.semiring.mult-not-zero semiring.mult-not-zero by
blast
  hence 29: ?e ⊓ g ≠ bot
    using 27 minatom-meet-bot by auto
  have sum (?w ⊓ g) = sum (w ⊓ −?fp ⊓ g) + sum (?p^T ⊓ g) + sum (?e
⊓ g)
    using 8 9 14 by (metis sum-disjoint-3 epm-8 epm-9 epm-10
minimum-spanning-tree-def spanning-tree-def)
  also have ... ≤ sum (w ⊓ −?fp ⊓ g) + sum (?p ⊓ g) + sum (?f ⊓ g)
    using 17 28 29 by (metis sum-plus-right-isotone sum-symmetric)
  also have ... = sum (((w ⊓ −?fp) ⊔ ?p ⊔ ?f) ⊓ g)
    using 3 9 by (metis sum-disjoint-3 epm-11 epm-12 epm-13)
  also have ... = sum (w ⊓ g)
    using 3 9 20 23 epm-2 by force
```

```
        finally have sum (?w ⊓ g) ≤ sum (w ⊓ g)
          .
        thus ∀ u . spanning-tree u g r ⟶ sum (?w ⊓ g) ≤ sum (u ⊓ g)
          using 14 order-lesseq-imp minimum-spanning-tree-def by auto
      qed
    next
      show t ⊔ ?e ≤ ?w
        using 5 9 14 mst-extends-new-tree by simp
    qed
  qed
qed
next
  fix t v
  let ?g = −−g
  assume 30: invariant t v g r ∧ v ∗ −v^T ⊓ g = bot
  have 31: regular v ∧ regular (v ∗ v^T)
    using 30 by (metis invariant-def spanning-tree-def precondition-def
regular-conv-closed regular-closed-star regular-mult-closed conv-involutive)
  have 32: v ∗ −v^T ⊓ ?g = bot
    using 30 pp-inf-bot-iff pp-pp-inf-bot-iff by simp
  have 33: v^T = r^T ∗ t^⋆ ∧ vector v
    using 30 by (simp add: covector-mult-closed invariant-def
vector-conv-covector precondition-def)
  have 34: t ≤ v ∗ v^T ⊓ ?g
    using 30 31 by (metis invariant-def inf-pp-commute spanning-tree-def
inf.boundedE)
  have r^T ∗ (v ∗ v^T ⊓ ?g)^⋆ ≤ r^T ∗ t^⋆
    using 30 31 by (metis invariant-def inf-pp-commute spanning-tree-def)
  hence 35: component g r = v^T
    using 31 32 33 34 by (metis span-post)
  from 30 obtain w where 36: minimum-spanning-tree w g r ∧ t ≤ w
    by (metis invariant-def)
  have 37: w ≤ v ∗ v^T
    using 31 35 36 by (simp add: minimum-spanning-tree-def spanning-tree-def
inf-pp-commute)
  have vector r ∧ injective r ∧ forest w
    using 30 36 by (simp add: invariant-def precondition-def
minimum-spanning-tree-def spanning-tree-def)
  hence w = t
    using 33 36 37 mst-post by auto
  thus minimum-spanning-tree t g r
    using 36 by simp
qed

end

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. An algebraic framework for minimum spanning tree problems. Submitted, 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.