Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

December 17, 2016

Abstract

This Isabelle/HOL formalization introduces a nested multiset datatype and defines Dershowitz and Manna's nested multiset order. The order is proved well founded and linear. By removing one constructor, we transform the nested multisets into hereditary multisets. These are isomorphic to the syntactic ordinals—the ordinals can be recursively expressed in Cantor normal form. Addition, subtraction, multiplication, and linear orders are provided on this type.

Contents

1 Introduction

This Isabelle/HOL formalization introduces a nested multiset datatype and defines Dershowitz and Manna's nested multiset order. The order is proved well founded and linear. By removing one constructor, we transform the nested multisets into hereditary multisets. These are isomorphic to the syntactic ordinals—the ordinals can be recursively expressed in Cantor normal form. Addition, subtraction, multiplication, and linear orders are provided on this type.

In addition, signed (or hybrid) multisets are provided (i.e., multisets with potentially negative multiplicities), as well as signed hereditary multisets and signed ordinals (e.g., $\omega^2 - 2\omega + 1$).

2 More about Multisets

theory *Multiset_More*
imports ~~/*src/HOL/Library/Multiset_Order*
begin

Isabelle's theory of finite multisets is not as developed as other areas, such as lists and sets. The present theory introduces some missing concepts and lemmas. Some of it is expected to move to Isabelle's library.

2.1 Basic Setup

declare
diff_single_trivial [*simp*]
in_image_mset [*iff*]
image_mset.compositionality [*simp*]

mset_subset_eqD[*dest*, *intro?*]

Multiset.in_multiset_in_set[*simp*]
inter_add_left1[*simp*]
inter_add_left2[*simp*]
inter_add_right1[*simp*]
inter_add_right2[*simp*]

sum_mset_sum_list[*simp*]

lemma *add_mset_in_multiset'*:
assumes *M*: $M \in multiset$
shows $(\lambda b.\ if\ b = a\ then\ Suc\ (M\ a)\ else\ M\ b) \in multiset$
proof −
have *if_eq*: $(\lambda b.\ if\ b = a\ then\ Suc\ (M\ a)\ else\ M\ b) = (\lambda b.\ if\ b = a\ then\ Suc\ (M\ b)\ else\ M\ b)$
by *force*
show *?thesis*
by (*auto simp*: *if_eq intro*!: *add_mset_in_multiset*[*OF M*, *of a*])
qed

2.2 Lemmas about the Multiset Order

instantiation *multiset* :: (*linorder*) *distrib_lattice*
begin

definition *inf_multiset* :: $'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
inf_multiset A B = (*if A* < *B then A else B*)

definition *sup_multiset* :: $'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
sup_multiset A B = (*if B* > *A then B else A*)

instance
by *intro_classes* (*auto simp*: *inf_multiset_def sup_multiset_def*)

end

lemma *all_lt_Max_imp_lt_multiset*: $N \neq \{\#\} \Longrightarrow (\forall a \in\# M.\ a < Max\ (set\_mset\ N)) \Longrightarrow M < N$
by (*meson Max_in*[*OF finite_set_mset*] *ex_gt_imp_less_multiset set_mset_eq_empty_iff*)

lemma *lt_imp_ex_count_lt*: $M < N \Longrightarrow \exists y.\ count\ M\ y < count\ N\ y$
by (*meson less_eq_multiset*$_{HO}$ *less_le_not_le*)

lemma *subset_imp_less_multiset*:
fixes *A*
assumes *a_sub_b*: $A \subset\# B$
shows $A < B$
proof −
have *a_subeq_b*: $A \subseteq\# B$
using *a_sub_b* **by** (*rule subset_mset.less_imp_le*)
hence $A \leq B$
by (*rule subset_eq_imp_le_multiset*)
moreover have $A \neq B$
using *a_sub_b* **by** *simp*
ultimately show $A < B$
by (*rule order_neq_le_trans*[*rotated 1*])
qed

lemma *image_mset_mono*:
assumes
mono_f: $\forall x \in set\_mset\ M.\ \forall y \in set\_mset\ N.\ x < y \longrightarrow f\ x < f\ y$ **and**
less: $M < N$
shows *image_mset f M* < *image_mset f N*
proof −

```
  obtain Y X where
    y_nemp: Y ≠ {#} and y_sub_N: Y ⊆# N and M_eq: M = N - Y + X and
    ex_y: ∀x. x ∈# X ⟶ (∃y. y ∈# Y ∧ x < y)
    using less[unfolded less_multiset_DM] by blast

  have x_sub_M: X ⊆# M
    using M_eq by simp

  let ?fY = image_mset f Y
  let ?fX = image_mset f X

  show ?thesis
    unfolding less_multiset_DM
  proof (intro exI conjI)
    show image_mset f M = image_mset f N - ?fY + ?fX
      using M_eq[THEN arg_cong, of image_mset f] y_sub_N
      by (metis image_mset_Diff image_mset_union)
  next
    obtain y where y: ∀x. x ∈# X ⟶ y x ∈# Y ∧ x < y x
      using ex_y by moura

    show ∀fx. fx ∈# ?fX ⟶ (∃fy. fy ∈# ?fY ∧ fx < fy)
    proof (intro allI impI)
      fix fx
      assume fx ∈# ?fX
      then obtain x where fx: fx = f x and x_in: x ∈# X
        by auto
      hence y_in: y x ∈# Y and y_gt: x < y x
        using y[rule_format, OF x_in] by blast+
      hence f (y x) ∈# ?fY ∧ f x < f (y x)
        using mono_f y_sub_N x_sub_M x_in
        by (metis image_eqI in_image_mset mset_subset_eqD)
      thus ∃fy. fy ∈# ?fY ∧ fx < fy
        unfolding fx by auto
    qed
  qed (auto simp: y_nemp y_sub_N image_mset_subseteq_mono)
qed
```

2.3 Lemmas about Intersection, Union and Pointwise Inclusion

```
lemma mset_inter_single: x ∈# Σ ⟹ Σ ∩# {#x#} = {#x#}
  by simp

lemma subset_add_mset_notin_subset_mset: ‹A ⊆# add_mset b B ⟹ b ∉# A  ⟹ A ⊆# B›
  by (simp add: subset_mset.sup.absorb_iff2)
```

psubsetE is the set counterpart

```
lemma subset_msetE [elim!]:
  ⟦A ⊂# B; ⟦A ⊆# B; ~ (B⊆#A)⟧ ⟹ R⟧ ⟹ R
  unfolding subseteq_mset_def subset_mset_def by (meson mset_subset_eqI subset_mset.eq_iff)

lemma Diff_triv_mset: M ∩# N = {#} ⟹ M - N = M
  by (metis diff_intersect_left_idem diff_zero)

lemma diff_intersect_sym_diff: (A - B) ∩# (B - A) = {#}
  unfolding multiset_inter_def
proof -
  have A - (B - (B - A)) = A - B
    by (metis (full_types) diff_intersect_right_idem multiset_inter_def)
  then show A - B - (A - B - (B - A)) = {#}
    by (metis add_diff_cancel_right' cancel_ab_semigroup_add_class.diff_right_commute
      cancel_comm_monoid_add_class.diff_cancel diff_subset_eq_self diff_zero subset_mset.diff_add
      subset_mset.diff_diff_right)
qed
```

2.4 Lemmas about Size

This sections adds various lemmas about size. Most lemmas have a finite set equivalent.

lemma *size_mset_SucE*: *size A = Suc n* $\Longrightarrow$ ($\bigwedge$*a B. A = {#a#} + B* $\Longrightarrow$ *size B = n* $\Longrightarrow$ *P*) $\Longrightarrow$ *P*
by (*cases A*) (*auto simp add*: *ac_simps*)

lemma *size_Suc_Diff1*: *x* $\in\#$ Σ $\Longrightarrow$ *Suc* (*size* (Σ − {#*x*#})) = *size* Σ
using *arg_cong*[*OF insert_DiffM*, *of* _ _ *size*] **by** *simp*

lemma *size_Diff_singleton*: *x* $\in\#$ Σ $\Longrightarrow$ *size* (Σ − {#*x*#}) = *size* Σ − *1*
by (*simp add*: *size_Suc_Diff1* [*symmetric*])

lemma *size_Diff_singleton_if*: *size* (*A* − {#*x*#}) = (*if x* $\in\#$ *A then size A* − *1 else size A*)
by (*simp add*: *size_Diff_singleton*)

lemma *size_Un_Int*: *size A* + *size B* = *size* (*A* $\cup\#$ *B*) + *size* (*A* $\cap\#$ *B*)
proof −
have ∗: *A* + *B* = *B* + (*A* − *B* + (*A* − (*A* − *B*)))
by (*simp add*: *subset_mset.add_diff_inverse union_commute*)
have *size B* + *size* (*A* − *B*) = *size A* + *size* (*B* − *A*)
unfolding *size_union*[*symmetric*] *sup_subset_mset_def*[*symmetric*]
by (*subst subset_mset.sup_commute*) (*rule refl*)
then show *?thesis* **unfolding** *multiset_inter_def size_union*[*symmetric*] ∗
by (*simp add*: *sup_subset_mset_def del*: *subset_mset.add_diff_assoc*)
qed

lemma *size_Un_disjoint*:
assumes *A* $\cap\#$ *B* = {#}
shows *size* (*A* $\cup\#$ *B*) = *size A* + *size B*
using *assms size_Un_Int* [*of A B*] **by** *simp*

lemma *size_Diff_subset_Int*:
shows *size* (Σ − Σ') = *size* Σ − *size* (Σ $\cap\#$ Σ')
proof −
have ∗: Σ − Σ' = Σ − Σ $\cap\#$ Σ' **by** (*auto simp add*: *multiset_eq_iff*)
show *?thesis* **unfolding** ∗ **using** *size_Diff_submset subset_mset.inf.cobounded1* **by** *blast*
qed

lemma *diff_size_le_size_Diff*: *size* (Σ:: _ *multiset*) − *size* Σ' ≤ *size* (Σ − Σ')
proof−
have *size* Σ − *size* Σ' ≤ *size* Σ − *size* (Σ $\cap\#$ Σ')
using *size_mset_mono diff_le_mono2 subset_mset.inf_le2* **by** *metis*
also have ... = *size*(Σ−Σ') **by**(*simp add*: *size_Diff_subset_Int*)
finally show *?thesis* **.**
qed

lemma *size_Diff1_less*: *x*$\in\#$ Σ $\Longrightarrow$ *size* (Σ − {#*x*#}) < *size* Σ
by (*rule Suc_less_SucD*) (*simp add*: *size_Suc_Diff1*)

lemma *size_Diff2_less*: *x*$\in\#$ Σ $\Longrightarrow$ *y*$\in\#$ Σ $\Longrightarrow$ *size* (Σ − {#*x*#} − {#*y*#}) < *size* Σ
by (*metis less_imp_diff_less size_Diff1_less size_Diff_subset_Int*)

lemma *size_Diff1_le*: *size* (Σ − {#*x*#}) ≤ *size* Σ
by (*cases x* $\in\#$ Σ) (*simp_all add*: *size_Diff1_less less_imp_le*)

lemma *size_psubset*: Σ $\subseteq\#$ Σ' $\Longrightarrow$ *size* Σ < *size* Σ' $\Longrightarrow$ Σ $\subset\#$ Σ'
using *less_irrefl subset_mset_def* **by** *blast*

2.5 Lemmas about Filter and Image

lemma *count_image_mset_ge_count*:
count (*image_mset f A*) (*f b*) ≥ *count A b*
by (*induction A*) *auto*

lemma *count_ image_ mset_ inj*:
assumes ‹*inj f*›
shows ‹*count* (*image_ mset f M*) (*f x*) = *count M x*›
by (*induction M*) (*use assms* **in** ‹*auto simp*: *inj_ on_ def*›)

lemma *mset_ filter_ compl*: *mset* (*filter p xs*) + *mset* (*filter* (*Not* ∘ *p*) *xs*) = *mset xs*
by (*induction xs*) (*auto simp*: *mset_ filter ac_ simps*)

lemma *comprehension_ mset_ False*[*simp*]: {# *L* ∈# *A*. *False*#} = {#}
by (*auto simp*: *multiset_ eq_ iff*)

Near duplicate of *filter_ eq_ replicate_ mset*: {#*y* ∈# *?D*. *y* = *?x*#} = *replicate_ mset* (*count ?D ?x*) *?x*.

lemma *filter_ mset_ eq*: *filter_ mset* (*op* = *L*) *A* = *replicate_ mset* (*count A L*) *L*
by (*auto simp*: *multiset_ eq_ iff*)

See *filter_ cong* for the set version. Mark as [*fundef_ cong*] too?

lemma *filter_ mset_ cong*:
assumes [*simp*]: *M* = *M′* **and** [*simp*]: ⋀*a*. *a* ∈# *M* ⟹ *P a* = *Q a*
shows *filter_ mset P M* = *filter_ mset Q M*
proof −
have *M* − *filter_ mset Q M* = *filter_ mset* (λ*a*. ¬*Q a*) *M*
by (*subst multiset_ partition*[*of* _ *Q*]) *simp*
then show *?thesis*
by (*auto simp*: *filter_ mset_ eq_ conv*)
qed

lemma *filter_ mset_ filter_ mset*: *filter_ mset Q* (*filter_ mset P M*) = {#*x* ∈# *M*. *P x* ∧ *Q x*#}
by (*auto simp*: *multiset_ eq_ iff*)

lemma *image_ mset_ const*: {#*c*. *x* ∈# *M*#} = *replicate_ mset* (*size M*) *c*
by (*induction M*) *auto*

lemma *image_ mset_ filter_ swap*: *image_ mset f* {# *x* ∈# *M*. *P* (*f x*)#} = {# *x* ∈# *image_ mset f M*. *P x*#}
by (*induction M*) *auto*

lemma *image_ mset_ cong2*[*cong*]:
(⋀*x*. *x* ∈# *M* ⟹ *f x* = *g x*) ⟹ *M* = *N* ⟹ *image_ mset f M* = *image_ mset g N*
by (*hypsubst*, *rule image_ mset_ cong*)

lemma *filter_ mset_ empty_ conv*: ‹(*filter_ mset P M* = {#}) = (∀ *L*∈#*M*. ¬ *P L*)›
by (*induction M*) *auto*

lemma *multiset_ filter_ mono2*: ‹*filter_ mset P A* ⊆# *filter_ mset Q A* ⟷
(∀ *a*∈#*A*. *P a* ⟶ *Q a*)›
by (*induction A*) (*auto intro*: *subset_ mset.order.trans*)

lemma *image_ filter_ cong*:
assumes ‹⋀*C*. *C* ∈# *M* ⟹ *P C* ⟹ *f C* = *g C*›
shows
‹{#*f C*. *C* ∈# {#*C* ∈# *M*. *P C*#}#} = {#*g C*|*C*∈# *M*. *P C*#}›
using *assms* **by** (*induction M*) *auto*

lemma *image_ mset_ filter_ swap2*: ‹{#*C* ∈# {#*P x*. *x* ∈# *D*#}. *Q C* #} = {#*P x*. *x* ∈# {#*C*| *C* ∈# *D*. *Q* (*P C*)#}#}›
by (*simp add*: *image_ mset_ filter_ swap*)

2.6 Lemma about Sum

lemma *count_ sum_ mset_ if_ 1_ 0*: ‹*count M a* = (∑ *x*∈#*M*. *if x* = *a then 1 else 0*)›
by (*induction M*) *auto*

2.7 Lemmas about Remove

lemma *set_mset_minus_replicate_mset*[*simp*]:
  $n \geq count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A - \{a\}$
  $n < count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A$
  unfolding *set_mset_def* **by** (*auto split*: *if_split simp*: *not_in_iff*)

abbreviation *removeAll_mset* :: $'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
removeAll_mset C M $\equiv$ *M − replicate_mset* (*count M C*) *C*

lemma *mset_removeAll*[*simp*, *code*]:
  removeAll_mset C (*mset L*) = *mset* (*removeAll C L*)
  by (*induction L*) (*auto simp*: *ac_simps multiset_eq_iff split*: *if_split_asm*)

lemma *removeAll_mset_filter_mset*:
  removeAll_mset C M = *filter_mset* (*op* ≠ *C*) *M*
  by (*induction M*) (*auto simp*: *ac_simps multiset_eq_iff*)

abbreviation *remove1_mset* :: $'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
  remove1_mset C M $\equiv$ *M − {#C#}*

lemma *in_remove1_mset_neq*:
  assumes *ab*: $a \neq b$
  shows $a \in\# remove1\_mset\ b\ C \longleftrightarrow a \in\# C$
proof −
  have *count {#b#} a* = *0*
    using *ab* **by** *simp*
  then show *?thesis*
    by (*metis* (*no_types*) *count_diff diff_zero mem_Collect_eq set_mset_def*)
qed

lemma *size_mset_removeAll_mset_le_iff*: *size* (*removeAll_mset x M*) < *size M* $\longleftrightarrow x \in\# M$
  apply (*rule iffI*)
    apply (*force intro*: *count_inI*)
  apply (*rule mset_subset_size*)
  apply (*auto simp*: *subset_mset_def multiset_eq_iff*)
  done

lemma *size_remove1_mset_If*: ‹*size* (*remove1_mset x M*) = *size M* − (*if x* ∈# *M then 1 else 0*)›
  by (*auto simp*: *size_Diff_subset_Int mset_inter_single*)

lemma *size_mset_remove1_mset_le_iff*: *size* (*remove1_mset x M*) < *size M* $\longleftrightarrow x \in\# M$
  apply (*rule iffI*)
    using *less_irrefl* **apply** *fastforce*
  apply (*rule mset_subset_size*)
  by (*auto elim*: *in_countE simp*: *subset_mset_def multiset_eq_iff*)

lemma *single_remove1_mset_eq*:
  add_mset a (*remove1_mset a M*) = *M* $\longleftrightarrow a \in\# M$
  by (*cases count M a*) (*auto simp*: *multiset_eq_iff count_eq_zero_iff count_inI*)

lemma *remove_1_mset_id_iff_notin*:
  remove1_mset a M = *M* $\longleftrightarrow a \notin\# M$
  by (*meson diff_single_trivial multi_drop_mem_not_eq*)

lemma *id_remove_1_mset_iff_notin*:
  M = *remove1_mset a M* $\longleftrightarrow a \notin\# M$
  using *remove_1_mset_id_iff_notin* **by** *metis*

lemma *remove1_mset_eqE*:
  remove1_mset L x1 = *M* $\Longrightarrow$
    (*L* ∈# *x1* $\Longrightarrow$ *x1* = *M* + {#*L*#} $\Longrightarrow$ *P*) $\Longrightarrow$
    (*L* ∉# *x1* $\Longrightarrow$ *x1* = *M* $\Longrightarrow$ *P*) $\Longrightarrow$
  P

by (*cases L ∈# x1*) *auto*

lemma *image_filter_ne_mset*[*simp*]:
image_mset f {#*x* ∈# *M*. *f x* ≠ *y*#} = *removeAll_mset y* (*image_mset f M*)
by (*induction M*) *simp_all*

lemma *image_mset_remove1_mset_if*:
image_mset f (*remove1_mset a M*) =
(*if a* ∈# *M then remove1_mset* (*f a*) (*image_mset f M*) *else image_mset f M*)
by (*auto simp*: *image_mset_Diff*)

lemma *filter_mset_neq*: {#*x* ∈# *M*. *x* ≠ *y*#} = *removeAll_mset y M*
by (*metis add_diff_cancel_left' filter_eq_replicate_mset multiset_partition*)

lemma *filter_mset_neq_cond*: {#*x* ∈# *M*. *P x* ∧ *x* ≠ *y*#} = *removeAll_mset y* {# *x*∈#*M*. *P x*#}
by (*metis add_diff_cancel_left' filter_eq_replicate_mset filter_mset_filter_mset*
multiset_partition)

lemma *remove1_mset_add_mset_If*:
remove1_mset L (*add_mset L' C*) = (*if L* = *L' then C else remove1_mset L C* + {#*L'*#})
by (*auto simp*: *multiset_eq_iff*)

lemma *minus_remove1_mset_if*:
A − *remove1_mset b B* = (*if b* ∈# *B* ∧ *b* ∈# *A* ∧ *count A b* ≥ *count B b then* {#*b*#} + (*A* − *B*) *else A* − *B*)
by (*auto simp*: *multiset_eq_iff count_greater_zero_iff*[*symmetric*]
simp del: *count_greater_zero_iff*)

lemma *add_mset_eq_add_mset_ne*:
a ≠ *b* ⟹ *add_mset a A* = *add_mset b B* ⟷ *a* ∈# *B* ∧ *b* ∈# *A* ∧ *A* = *add_mset b* (*B* − {#*a*#})
by (*metis* (*no_types*, *lifting*) *diff_single_eq_union diff_union_swap multi_self_add_other_not_self*
remove_1_mset_id_iff_notin union_single_eq_diff)

This lemma allows to split equality like {#*K*, *K'*#} = {#*L*, *L'*#} into *K* = *L* ∧ *L* = *L'* ∨ *K* ≠ *L* ∧ *K* = *L'* ∧ *K'* = *L*. It can lead to a explosion of the numbers of cases.

lemma *add_mset_eq_add_mset*: ‹*add_mset a M* = *add_mset b M'* ⟷
(*a* = *b* ∧ *M* = *M'*) ∨ (*a* ≠ *b* ∧ *b* ∈# *M* ∧ *add_mset a* (*M* − {#*b*#}) = *M'*)›
by (*metis add_mset_eq_add_mset_ne add_mset_remove_trivial union_single_eq_member*)

lemma *add_mset_remove_trivial_iff*: ‹*N* = *add_mset a* (*N* − {#*b*#}) ⟷ *a* ∈# *N* ∧ *a* = *b*›
by (*metis add_left_cancel add_mset_remove_trivial insert_DiffM2 single_eq_single*
size_mset_remove1_mset_le_iff union_single_eq_member)

lemma *trivial_add_mset_remove_iff*: ‹*add_mset a* (*N* − {#*b*#}) = *N* ⟷ *a* ∈# *N* ∧ *a* = *b*›
by (*subst eq_commute*) (*fact add_mset_remove_trivial_iff*)

lemma *remove1_single_empty_iff*[*simp*]: ‹*remove1_mset L* {#*L'*#} = {#} ⟷ *L* = *L'*›
using *add_mset_remove_trivial_iff* **by** *fastforce*

2.8 Lemmas about Replicate

lemma *replicate_mset_minus_replicate_mset_same*[*simp*]:
replicate_mset m x − *replicate_mset n x* = *replicate_mset* (*m* − *n*) *x*
proof (*induct m arbitrary*: *n*, *simp_all*)
fix *ma* :: *nat* **and** *na* :: *nat*
assume *a1*: ⋀*n*. *replicate_mset ma x* − *replicate_mset n x* = *replicate_mset* (*ma* − *n*) *x*
have *f2*: ⋀*n a*. *0* < *n* ⟹ *add_mset a* (*replicate_mset* (*n* − *Suc 0*) *a*) = *replicate_mset n a*
by (*metis* (*full_types*) *Suc_pred replicate_mset_Suc*)
have *replicate_mset* (*Suc ma*) *x* = *add_mset x* (*replicate_mset ma x*)
by *auto*
then show *add_mset x* (*replicate_mset ma x*) − *replicate_mset na x* = *replicate_mset* (*Suc ma* − *na*) *x*
using *f2 a1* **by** (*metis* (*no_types*) *Suc_pred add_mset_diff_bothsides*
cancel_comm_monoid_add_class.diff_cancel diff_Suc_Suc diff_zero neq0_conv)

qed

lemma *replicate_mset_subset_iff_lt*[*simp*]: *replicate_mset m x* $\subset\#$ *replicate_mset n x* $\longleftrightarrow$ $m < n$
by (*induct n m rule*: *diff_induct*) (*auto intro*: *subset_mset.gr_zeroI*)

lemma *replicate_mset_subseteq_iff_le*[*simp*]: *replicate_mset m x* $\subseteq\#$ *replicate_mset n x* $\longleftrightarrow$ $m \leq n$
by (*induct n m rule*: *diff_induct*) *auto*

lemma *replicate_mset_lt_iff_lt*[*simp*]: *replicate_mset m x* $<$ *replicate_mset n x* $\longleftrightarrow$ $m < n$
by (*induct n m rule*: *diff_induct*) (*auto intro*: *subset_mset.gr_zeroI gr_zeroI*)

lemma *replicate_mset_le_iff_le*[*simp*]: *replicate_mset m x* $\leq$ *replicate_mset n x* $\longleftrightarrow$ $m \leq n$
by (*induct n m rule*: *diff_induct*) *auto*

lemma *replicate_mset_eq_iff*[*simp*]:
replicate_mset m x = *replicate_mset n y* $\longleftrightarrow$ $m = n \wedge (m \neq 0 \longrightarrow x = y)$
by (*cases m*; *cases n*; *simp*)
(*metis in_replicate_mset insert_noteq_member size_replicate_mset union_single_eq_diff*)

lemma *replicate_mset_plus*: *replicate_mset* $(a + b)$ *C* = *replicate_mset a C* + *replicate_mset b C*
by (*induct a*) (*auto simp*: *ac_simps*)

lemma *mset_replicate_replicate_mset*: *mset* (*replicate n L*) = *replicate_mset n L*
by (*induction n*) *auto*

lemma *set_mset_single_iff_replicate_mset*:
set_mset U = {*a*} $\longleftrightarrow$ $(\exists\, n > 0.\ U$ = *replicate_mset n a*) (**is** *?S* $\longleftrightarrow$ *?R*)
proof
assume *?S*
show *?R*
proof (*rule ccontr*)
assume $\neg$ *?R*
have $\forall\, n.\ U \neq$ *replicate_mset n a*
using ‹*?S*› ‹$\neg$ *?R*› **by** (*metis gr_zeroI insert_not_empty set_mset_replicate_mset_subset*)
then obtain *b* **where** $b \in\# U$ **and** $b \neq a$
by (*metis count_replicate_mset mem_Collect_eq multiset_eqI neq0_conv set_mset_def*)
then show *False*
using ‹*?S*› **by** *auto*
qed
qed *auto*

lemma *ex_replicate_mset_if_all_elems_eq*:
assumes $\forall\, x \in\# M.\ x = y$
shows $\exists\, n.\ M$ = *replicate_mset n y*
using *assms* **by** (*metis count_replicate_mset mem_Collect_eq multiset_eqI neq0_conv set_mset_def*)

2.9 Multiset and Set Conversions

lemma *count_mset_set_if*: *count* (*mset_set A*) *a* = (*if a* $\in$ *A* $\wedge$ *finite A then 1 else 0*)
by *auto*

lemma *mset_set_set_mset_empty_mempty*[*iff*]: *mset_set* (*set_mset D*) = {#} $\longleftrightarrow$ *D* = {#}
by (*auto dest*: *arg_cong*[*of* _ _ *set_mset*])

lemma *count_mset_set_le_one*: *count* (*mset_set A*) *x* $\leq$ *1*
by (*simp add*: *count_mset_set_if*)

lemma *mset_set_subseteq_mset_set*[*iff*]:
assumes *finite A finite B*
shows *mset_set A* $\subseteq\#$ *mset_set B* $\longleftrightarrow$ $A \subseteq B$
by (*metis assms contra_subsetD count_mset_set*(*1*,*3*) *count_mset_set_le_one finite_set_mset_mset_set*
less_eq_nat.simps(*1*) *mset_subset_eqI set_mset_mono*)

lemma *mset_set_set_mset_subseteq*[*simp*]: *mset_set* (*set_mset A*) $\subseteq\#$ *A*

by (*simp add*: *subseteq_mset_def count_mset_set_if*)

lemma *mset_sorted_list_of_set*[*simp*]: *mset* (*sorted_list_of_set A*) = *mset_set A*
by (*metis mset_sorted_list_of_multiset sorted_list_of_mset_set*)

lemma *mset_take_subseteq*: *mset* (*take n xs*) ⊆# *mset xs*
apply (*induct xs arbitrary*: *n*)
apply *simp*
by (*case_tac n*) *simp_all*

2.10 Duplicate Removal

definition *remdups_mset* :: *'v multiset* ⇒ *'v multiset* **where**
remdups_mset S = *mset_set* (*set_mset S*)

lemma *set_mset_remdups_mset*[*simp*]: ‹*set_mset* (*remdups_mset A*) = *set_mset A*›
unfolding *remdups_mset_def* **by** *auto*

lemma *count_remdups_mset_eq_1*: *a* ∈# *remdups_mset A* ⟷ *count* (*remdups_mset A*) *a* = *1*
unfolding *remdups_mset_def* **by** (*auto simp*: *count_eq_zero_iff intro*: *count_inI*)

lemma *remdups_mset_empty*[*simp*]: *remdups_mset* {#} = {#}
unfolding *remdups_mset_def* **by** *auto*

lemma *remdups_mset_singleton*[*simp*]: *remdups_mset* {#*a*#} = {#*a*#}
unfolding *remdups_mset_def* **by** *auto*

lemma *remdups_mset_eq_empty*[*iff*]: *remdups_mset D* = {#} ⟷ *D* = {#}
unfolding *remdups_mset_def* **by** *blast*

lemma *remdups_mset_singleton_sum*[*simp*]:
remdups_mset (*add_mset a A*) = (*if a* ∈# *A then remdups_mset A else add_mset a* (*remdups_mset A*))
unfolding *remdups_mset_def* **by** (*simp_all add*: *insert_absorb*)

lemma *mset_remdups_remdups_mset*[*simp*]: *mset* (*remdups D*) = *remdups_mset* (*mset D*)
by (*induction D*) (*auto simp add*: *ac_simps*)

declare *mset_remdups_remdups_mset*[*symmetric*, *code*]

definition *distinct_mset* :: *'a multiset* ⇒ *bool* **where**
distinct_mset S ⟷ (∀ *a*. *a* ∈# *S* ⟶ *count S a* = *1*)

Another characterisation of *distinct_mset*:

lemma *distinct_mset_count_less_1*: *distinct_mset S* ⟷ (∀ *a*. *count S a* ≤ *1*)
using *eq_iff nat_le_linear* **unfolding** *distinct_mset_def* **by** *fastforce*

lemma *distinct_mset_empty*[*simp*]: *distinct_mset* {#}
unfolding *distinct_mset_def* **by** *auto*

lemma *distinct_mset_singleton*: *distinct_mset* {#*a*#}
unfolding *distinct_mset_def* **by** *auto*

lemma *distinct_mset_union*:
assumes *dist*: *distinct_mset* (*A* + *B*)
shows *distinct_mset A*
unfolding *distinct_mset_count_less_1*
proof (*rule allI*)
fix *a*
have ‹*count A a* ≤ *count* (*A* + *B*) *a*› **by** *auto*
moreover have ‹*count* (*A* + *B*) *a* ≤ *1*›
using *dist* **unfolding** *distinct_mset_count_less_1* **by** *auto*
ultimately show ‹*count A a* ≤ *1*›
by *simp*
qed

lemma *distinct_mset_minus*[*simp*]:
distinct_mset A ⟹ *distinct_mset* (*A* − *B*)
by (*metis diff_subset_eq_self mset_subset_eq_exists_conv distinct_mset_union*)

lemma *count_remdups_mset_If*: ‹*count* (*remdups_mset A*) *a* = (*if a* ∈# *A then 1 else 0*)›
unfolding *remdups_mset_def* **by** *auto*

lemma *distinct_mset_rempdups_union_mset*:
assumes *distinct_mset A* **and** *distinct_mset B*
shows *A* ∪# *B* = *remdups_mset* (*A* + *B*)
using *assms nat_le_linear* **unfolding** *remdups_mset_def*
by (*force simp add*: *multiset_eq_iff max_def count_mset_set_if distinct_mset_def not_in_iff*)

lemma *distinct_mset_add_mset*[*simp*]: *distinct_mset* (*add_mset a L*) ⟷ *a* ∉# *L* ∧ *distinct_mset L*
unfolding *distinct_mset_def*
apply (*rule iffI*)
apply (*auto split*: *if_split_asm*; *fail*)[]
by (*auto simp*: *not_in_iff*; *fail*)

lemma *distinct_mset_size_eq_card*: *distinct_mset C* ⟹ *size C* = *card* (*set_mset C*)
by (*induction C*) *auto*

lemma *distinct_mset_add*:
distinct_mset (*L* + *L'*) ⟷ *distinct_mset L* ∧ *distinct_mset L'* ∧ *L* ∩# *L'* = {#} (**is** *?A* ⟷ *?B*)
by (*induction L arbitrary*: *L'*) *auto*

lemma *distinct_mset_set_mset_ident*[*simp*]: *distinct_mset M* ⟹ *mset_set* (*set_mset M*) = *M*
by (*induction M*) *auto*

lemma *distinct_finite_set_mset_subseteq_iff*[*iff*]:
assumes *dist*: *distinct_mset M* **and** *fin*: *finite N*
shows *set_mset M* ⊆ *N* ⟷ *M* ⊆# *mset_set N*
proof
assume *set_mset M* ⊆ *N*
then show *M* ⊆# *mset_set N*
by (*metis dist distinct_mset_set_mset_ident fin finite_subset mset_set_subseteq_mset_set*)
next
assume *M* ⊆# *mset_set N*
then show *set_mset M* ⊆ *N*
by (*metis contra_subsetD empty_iff finite_set_mset_mset_set infinite_set_mset_mset_set*
set_mset_mono subsetI)
qed

lemma *distinct_mem_diff_mset*:
assumes *dist*: *distinct_mset M* **and** *mem*: *x* ∈ *set_mset* (*M* − *N*)
shows *x* ∉ *set_mset N*
proof −
have *count M x* = *1*
using *dist mem* **by** (*meson distinct_mset_def in_diffD*)
then show *?thesis*
using *mem* **by** (*metis count_greater_eq_one_iff in_diff_count not_less*)
qed

lemma *distinct_set_mset_eq*:
assumes
dist_m: *distinct_mset M* **and**
dist_n: *distinct_mset N* **and**
set_eq: *set_mset M* = *set_mset N*
shows *M* = *N*
proof −
have *mset_set* (*set_mset M*) = *mset_set* (*set_mset N*)
using *set_eq* **by** *simp*

thus *?thesis*
using *dist_ m dist_ n* **by** *auto*
qed

lemma *distinct_ mset_ union_ mset*[*simp*]:
⟨*distinct_ mset* (*D* ∪# *C*) ⟷ *distinct_ mset D* ∧ *distinct_ mset C*⟩
unfolding *distinct_ mset_ count_ less_ 1* **by** *force*

lemma *distinct_ mset_ inter_ mset*:
assumes
distinct_ mset D **and**
distinct_ mset C
shows *distinct_ mset* (*D* ∩# *C*)
using *assms* **unfolding** *distinct_ mset_ count_ less_ 1*
by (*meson dual_ order.trans subset_ mset.inf_ le2 subseteq_ mset_ def*)

lemma *distinct_ mset_ remove1_ All*: *distinct_ mset C* ⟹ *remove1_ mset L C* = *removeAll_ mset L C*
by (*auto simp*: *multiset_ eq_ iff distinct_ mset_ count_ less_ 1*)

lemma *distinct_ mset_ size_ 2*: *distinct_ mset* {#*a*, *b*#} ⟷ *a* ≠ *b*
by *auto*

lemma *distinct_ mset_ filter*: *distinct_ mset M* ⟹ *distinct_ mset* {# *L* ∈# *M*. *P L*#}
by (*simp add*: *distinct_ mset_ def*)

lemma *distinct_ mset_ mset_ distinct*[*simp*]: ⟨*distinct_ mset* (*mset xs*) = *distinct xs*⟩
by (*induction xs*) *auto*

lemma *distinct_ image_ mset_ inj*:
⟨*inj_ on f* (*set_ mset M*) ⟹ *distinct_ mset* (*image_ mset f M*) ⟷ *distinct_ mset M*⟩
by (*induction M*) (*auto simp*: *inj_ on_ def*)

2.11 Repeat Operation

lemma *repeat_ mset_ compower*: *repeat_ mset n A* = ((*op* + *A*) ^^ *n*) {#}
by (*induction n*) *auto*

lemma *repeat_ mset_ prod*: *repeat_ mset* (*m* ∗ *n*) *A* = ((*op* + (*repeat_ mset n A*)) ^^ *m*) {#}
by (*induction m*) (*auto simp*: *repeat_ mset_ distrib*)

2.12 Cartesian Product

Definition of the cartesian products over multisets. The construction mimics of the cartesian product on sets and use the same theorem names (adding only the suffix *_ mset* to Sigma and Times). See file `~~/src/HOL/Product_Type.thy`

definition *Sigma_ mset* :: *'a multiset* ⇒ (*'a* ⇒ *'b multiset*) ⇒ (*'a* × *'b*) *multiset* **where**
Sigma_ mset A B ≡ ⋃# {#{#(*a*, *b*). *b* ∈# *B a*#}. *a* ∈# *A* #}

abbreviation *Times_ mset* :: *'a multiset* ⇒ *'b multiset* ⇒ (*'a* × *'b*) *multiset* (**infixr** ×*mset 80*) **where**
Times_ mset A B ≡ *Sigma_ mset A* (λ_. *B*)

hide-const (**open**) *Times_ mset*

Contrary to the set version *A* × *B*, we use the non-ASCII symbol ∈#.

syntax
_ Sigma_ mset :: [*pttrn*, *'a multiset*, *'b multiset*] => (*'a* ∗ *'b*) *multiset*
((*3SIGMAMSET* _ ∈#_./ _) [*0*, *0*, *10*] *10*)
translations
SIGMAMSET x∈#*A*. *B* == *CONST Sigma_ mset A* (λ*x*. *B*)

Link between the multiset and the set cartesian product:

lemma *Times_ mset_ Times*: *set_ mset* (*A* ×*mset B*) = *set_ mset A* × *set_ mset B*

unfolding *Sigma_mset_def* **by** *auto*

lemma *Sigma_msetI* [*intro*!]: ⟦*a* ∈# *A*; *b* ∈# *B a*⟧ ⟹ (*a*, *b*) ∈# *Sigma_mset A B*
by (*unfold Sigma_mset_def*) *auto*

lemma *Sigma_msetE* [*elim*!]:
⟦ *c*∈# *Sigma_mset A B*;
⋀*x y*.⟦ *x*∈#*A*; *y*∈#*B*(*x*); *c*=(*x*,*y*) ⟧ ⟹ *P*
⟧ ⟹ *P*
— The general elimination rule.
by (*unfold Sigma_mset_def*) *auto*

Elimination of (*a*, *b*) ∈# *A* ×*mset B* – introduces no eigenvariables.

lemma *Sigma_msetD1*: (*a*, *b*) ∈# *Sigma_mset A B* ⟹ *a* ∈# *A*
by *blast*

lemma *Sigma_msetD2*: (*a*, *b*) ∈# *Sigma_mset A B* ⟹ *b* ∈# *B a*
by *blast*

lemma *Sigma_msetE2*:
⟦ (*a*, *b*) ∈# *Sigma_mset A B*;
⟦ *a*∈#*A*; *b*∈#*B*(*a*) ⟧ ⟹ *P*
⟧ ⟹ *P*
by *blast*

lemma *Sigma_mset_cong*:
⟦*A* = *B*; ⋀*x*. *x* ∈# *B* ⟹ *C x* = *D x*⟧
⟹ (*SIGMAMSET x*∈#*A*. *C x*) = (*SIGMAMSET x*∈# *B*. *D x*)
by (*metis* (*mono_tags*, *lifting*) *Sigma_mset_def image_mset_cong*)

lemma *count_sum_mset*: *count* (⋃#*M*) *b* = (∑ *P*∈#*M*. *count P b*)
by (*induction M*) *auto*

lemma *Sigma_mset_plus_distrib1*[*simp*]: *Sigma_mset* (*A* + *B*) *C* = *Sigma_mset A C* + *Sigma_mset B C*
unfolding *Sigma_mset_def* **by** *auto*

lemma *Sigma_mset_plus_distrib2*[*simp*]:
Sigma_mset A (λ*i*. *B i* + *C i*) = *Sigma_mset A B* + *Sigma_mset A C*
unfolding *Sigma_mset_def* **by** (*induction A*) (*auto simp*: *multiset_eq_iff*)

lemma *Times_mset_single_left*: {#*a*#} ×*mset B* = *image_mset* (*Pair a*) *B*
unfolding *Sigma_mset_def* **by** *auto*

lemma *Times_mset_single_right*: *A* ×*mset* {#*b*#} = *image_mset* (λ*a*. *Pair a b*) *A*
unfolding *Sigma_mset_def* **by** (*induction A*) *auto*

lemma *Times_mset_single_single*[*simp*]: {#*a*#} ×*mset* {#*b*#} = {#(*a*, *b*)#}
unfolding *Sigma_mset_def* **by** *simp*

lemma *count_image_mset_Pair*:
count (*image_mset* (*Pair a*) *B*) (*x*, *b*) = (*if x* = *a then count B b else* 0)
by (*induction B*) *auto*

lemma *count_Sigma_mset*: *count* (*Sigma_mset A B*) (*a*, *b*) = *count A a* ∗ *count* (*B a*) *b*
by (*induction A*) (*auto simp*: *Sigma_mset_def count_image_mset_Pair*)

lemma *Sigma_mset_empty1* [*simp*]: *Sigma_mset* {#} *B* = {#}
unfolding *Sigma_mset_def* **by** *auto*

lemma *Sigma_mset_empty2* [*simp*]: *A* ×*mset* {#} = {#}
by (*auto simp*: *multiset_eq_iff count_Sigma_mset*)

lemma *Sigma_mset_mono*:

assumes $A \subseteq\# C$ **and** $\bigwedge x.\ x \in\# A \Longrightarrow B\ x \subseteq\# D\ x$
shows *Sigma_mset A B* $\subseteq\#$ *Sigma_mset C D*
proof −
have *count A a* $*$ *count* (*B a*) *b* $\leq$ *count C a* $*$ *count* (*D a*) *b* **for** *a b*
using *assms* **unfolding** *subseteq_mset_def* **by** (*metis count_inI eq_iff mult_eq_0_iff mult_le_mono*)
then show *?thesis*
by (*auto simp*: *subseteq_mset_def count_Sigma_mset*)
qed

lemma *mem_Sigma_mset_iff*[*iff*]: $((a,b) \in\#$ *Sigma_mset A B*$) = (a \in\# A \wedge b \in\# B\ a)$
by *blast*

lemma *mem_Times_mset_iff*: $x \in\# A \times$*mset B* $\longleftrightarrow$ *fst x* $\in\#$ *A* $\wedge$ *snd x* $\in\#$ *B*
by (*induct x*) *simp*

lemma *Sigma_mset_empty_iff*: (*SIGMAMSET* $i \in\# I.\ X\ i) = \{\#\} \longleftrightarrow (\forall\, i \in\# I.\ X\ i = \{\#\})$
by (*auto simp*: *Sigma_mset_def*)

lemma *Times_mset_subset_mset_cancel1*: $x \in\# A \Longrightarrow (A \times$*mset B* $\subseteq\#$ *A* $\times$*mset C*$) = (B \subseteq\# C)$
by (*auto simp*: *subseteq_mset_def count_Sigma_mset*)

lemma *Times_mset_subset_mset_cancel2*: $x \in\# C \Longrightarrow (A \times$*mset C* $\subseteq\#$ *B* $\times$*mset C*$) = (A \subseteq\# B)$
by (*auto simp*: *subseteq_mset_def count_Sigma_mset*)

lemma *Times_mset_eq_cancel2*: $x \in\# C \Longrightarrow (A \times$*mset C* $= B \times$*mset C*$) = (A = B)$
by (*auto simp*: *multiset_eq_iff count_Sigma_mset dest*!: *in_countE*)

lemma *split_paired_Ball_mset_Sigma_mset* [*simp*, *no_atp*]:
$(\forall\, z \in\#$ *Sigma_mset A B*. *P z*$) \longleftrightarrow (\forall\, x \in\# A.\ \forall\, y \in\# B\ x.\ P\ (x,\ y))$
by *blast*

lemma *split_paired_Bex_mset_Sigma_mset* [*simp*, *no_atp*]:
$(\exists\, z \in\#$ *Sigma_mset A B*. *P z*$) \longleftrightarrow (\exists\, x \in\# A.\ \exists\, y \in\# B\ x.\ P\ (x,\ y))$
by *blast*

lemma *sum_mset_if_eq_constant*:
$(\sum x \in\# M.$ *if a* $=$ *x then* (*f x*) *else 0*$) =$ ((*op* $+$ (*f a*)) ^^ (*count M a*)) *0*
by (*induction M*) (*auto simp*: *ac_simps*)

lemma *iterate_op_plus*: ((*op* $+$ *k*) ^^ *m*) *0* $=$ *k* $*$ *m*
by (*induction m*) *auto*

lemma *untion_image_mset_Pair_distribute*:
$\bigcup\#\{\#$*image_mset* (*Pair x*) (*C x*). *x* $\in\#$ *J* $-$ *I*$\#\} = \bigcup\#\{\#$*image_mset* (*Pair x*) (*C x*). *x* $\in\#$ *J*$\#\} -$
$\bigcup\#\{\#$*image_mset* (*Pair x*) (*C x*). *x* $\in\#$ *I*$\#\}$
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant iterate_op_plus diff_mult_distrib2*)

lemma *Sigma_mset_Un_distrib1*: *Sigma_mset* (*I* $\cup\#$ *J*) *C* $=$ *Sigma_mset I C* $\cup\#$ *Sigma_mset J C*
by (*auto simp*: *Sigma_mset_def sup_subset_mset_def untion_image_mset_Pair_distribute*)

lemma *Sigma_mset_Un_distrib2*: (*SIGMAMSET* $i \in\# I.$ *A i* $\cup\#$ *B i*$) =$ *Sigma_mset I A* $\cup\#$ *Sigma_mset I B*
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def diff_mult_distrib2 iterate_op_plus max_def not_in_iff*)

lemma *Sigma_mset_Int_distrib1*: *Sigma_mset* (*I* $\cap\#$ *J*) *C* $=$ *Sigma_mset I C* $\cap\#$ *Sigma_mset J C*
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def iterate_op_plus min_def not_in_iff*)

lemma *Sigma_mset_Int_distrib2*: (*SIGMAMSET* $i \in\# I.$ *A i* $\cap\#$ *B i*$) =$ *Sigma_mset I A* $\cap\#$ *Sigma_mset I B*
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def iterate_op_plus min_def not_in_iff*)

lemma *Sigma_mset_Diff_distrib1*: *Sigma_mset* $(I - J)$ C = *Sigma_mset* I C − *Sigma_mset* J C
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def iterate_op_plus min_def not_in_iff diff_mult_distrib2*)

lemma *Sigma_mset_Diff_distrib2*: (*SIGMAMSET* $i∈\#I$. A i − B i) = *Sigma_mset* I A − *Sigma_mset* I B
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def iterate_op_plus min_def not_in_iff diff_mult_distrib*)

lemma *Sigma_mset_Union*: *Sigma_mset* $(\bigcup\#X)$ B = $(\bigcup\#$ (*image_mset* $(\lambda A.$ *Sigma_mset* A $B)$ $X))$
by (*auto simp*: *multiset_eq_iff count_sum_mset count_image_mset_Pair sum_mset_if_eq_constant Sigma_mset_def iterate_op_plus min_def not_in_iff sum_mset_distrib_left*)

lemma *Times_mset_Un_distrib1*: $(A \cup\# B) \times mset\ C = A \times mset\ C \cup\# B \times mset\ C$
by (*fact Sigma_mset_Un_distrib1*)

lemma *Times_mset_Int_distrib1*: $(A \cap\# B) \times mset\ C = A \times mset\ C \cap\# B \times mset\ C$
by (*fact Sigma_mset_Int_distrib1*)

lemma *Times_mset_Diff_distrib1*: $(A - B) \times mset\ C = A \times mset\ C - B \times mset\ C$
by (*fact Sigma_mset_Diff_distrib1*)

lemma *Times_mset_empty*[*simp*]: $A \times mset\ B = \{\#\} \longleftrightarrow A = \{\#\} \vee B = \{\#\}$
by (*auto simp*: *Sigma_mset_empty_iff*)

lemma *Times_insert_left*: $A \times mset$ *add_mset* x B = $A \times mset\ B$ + *image_mset* $(\lambda a.$ *Pair* a $x)$ A
unfolding *add_mset_add_single*[*of* x B] *Sigma_mset_plus_distrib2*
by (*simp add*: *Times_mset_single_right*)

lemma *Times_insert_right*: *add_mset* a $A \times mset\ B$ = $A \times mset\ B$ + *image_mset* (*Pair* a) B
unfolding *add_mset_add_single*[*of* a A] *Sigma_mset_plus_distrib1*
by (*simp add*: *Times_mset_single_left*)

lemma *fst_image_mset_times_mset* [*simp*]:
image_mset fst $(A \times mset\ B)$ = (*if* $B = \{\#\}$ *then* $\{\#\}$ *else repeat_mset* (*size* B) A)
by (*induction* B) (*auto simp*: *Times_mset_single_right ac_simps Times_insert_left*)

lemma *snd_image_mset_times_mset* [*simp*]:
image_mset snd $(A \times mset\ B)$ = (*if* $A = \{\#\}$ *then* $\{\#\}$ *else repeat_mset* (*size* A) B)
by (*induction* B) (*auto simp add*: *Times_mset_single_right image_mset_const Times_insert_left*)

lemma *product_swap_mset*:
image_mset prod.swap $(A \times mset\ B)$ = $B \times mset\ A$
by (*induction* A) (*auto simp add*: *Times_mset_single_left Times_mset_single_right Times_insert_right Times_insert_left*)

context
begin

qualified definition *product_mset* :: *'a multiset* ⇒ *'b multiset* ⇒ (*'a* × *'b*) *multiset* **where**
[*code_abbrev*]: *product_mset* A B = $A \times mset\ B$

lemma *member_product_mset*: $x \in\#$ *Multiset_More.product_mset* A $B \longleftrightarrow x \in\# A \times mset\ B$
by (*simp add*: *Multiset_More.product_mset_def*)

end

lemma *count_Sigma_mset_abs_def*: *count* (*Sigma_mset* A B) = $(\lambda(a, b) \Rightarrow$ *count* A a ∗ *count* $(B\ a)$ $b)$
by (*auto simp*: *fun_eq_iff count_Sigma_mset*)

lemma *Times_mset_image_mset1*: *image_mset* f $A \times mset\ B$ = *image_mset* $(\lambda(a, b).\ (f\ a, b))$ $(A \times mset\ B)$
by (*induct* B) (*auto simp*: *Times_insert_left*)

lemma *Times_mset_image_mset2*: $A \times mset$ *image_mset* f B = *image_mset* $(\lambda(a, b).\ (a, f\ b))$ $(A \times mset\ B)$

by (*induct A*) (*auto simp*: *Times_ insert_ right*)

lemma *sum_ le_ singleton*: $A \subseteq \{x\} \Longrightarrow sum\ f\ A = (if\ x \in A\ then\ f\ x\ else\ 0)$
by (*auto simp*: *subset_ singleton_ iff elim*: *finite_ subset*)

lemma *Times_ mset_ assoc*:
$(A \times mset\ B) \times mset\ C = image\_ mset\ (\lambda(a,\ b,\ c).\ ((a,\ b),\ c))\ (A \times mset\ B \times mset\ C)$
by (*auto simp*: *multiset_ eq_ iff count_ Sigma_ mset count_ image_ mset vimage_ def Times_ mset_ Times*
Int_ commute count_ eq_ zero_ iff
intro!: *trans*[*OF _ sym*[*OF sum_ le_ singleton*[*of _ (_, _, _)*]]]
cong: *sum.cong if_ cong*)

2.13 Transfer Rules

lemma *plus_ multiset_ transfer*[*transfer_ rule*]:
(*rel_ fun* (*rel_ mset R*) (*rel_ fun* (*rel_ mset R*) (*rel_ mset R*))) (*op +*) (*op +*)
by (*unfold rel_ fun_ def rel_ mset_ def*)
(*force dest*: *list_ all2_ appendI intro*: *exI*[*of _ _ @ _*] *conjI*[*rotated*])

lemma *minus_ multiset_ transfer*[*transfer_ rule*]:
assumes [*transfer_ rule*]: *bi_ unique R*
shows (*rel_ fun* (*rel_ mset R*) (*rel_ fun* (*rel_ mset R*) (*rel_ mset R*))) (*op −*) (*op −*)
proof (*unfold rel_ fun_ def rel_ mset_ def*, *safe*)
fix *xs ys xs′ ys′*
assume [*transfer_ rule*]: *list_ all2 R xs ys list_ all2 R xs′ ys′*
have *list_ all2 R* (*fold remove1 xs′ xs*) (*fold remove1 ys′ ys*)
by *transfer_ prover*
moreover
have *mset* (*fold remove1 xs′ xs*) = *mset xs* − *mset xs′*
by (*induct xs′ arbitrary*: *xs*) *auto*
moreover
have *mset* (*fold remove1 ys′ ys*) = *mset ys* − *mset ys′*
by (*induct ys′ arbitrary*: *ys*) *auto*
ultimately show $\exists xs''\ ys''.$
$mset\ xs'' = mset\ xs - mset\ xs' \wedge mset\ ys'' = mset\ ys - mset\ ys' \wedge list\_ all2\ R\ xs''\ ys''$
by *blast*
qed

declare *rel_ mset_ Zero*[*transfer_ rule*]

lemma *count_ transfer*[*transfer_ rule*]:
assumes *bi_ unique R*
shows (*rel_ fun* (*rel_ mset R*) (*rel_ fun R op* =)) *count count*
unfolding *rel_ fun_ def rel_ mset_ def* **proof** *safe*
fix *x y xs ys*
assume *list_ all2 R xs ys R x y*
then show *count* (*mset xs*) *x* = *count* (*mset ys*) *y*
proof (*induct xs ys rule*: *list.rel_ induct*)
case (*Cons x′ xs y′ ys*)
then show *?case* **using** *assms* **unfolding** *bi_ unique_ alt_ def2*
by (*auto simp*: *rel_ fun_ def*)
qed *simp*
qed

lemma *subseteq_ multiset_ transfer*[*transfer_ rule*]:
assumes [*transfer_ rule*]: *bi_ unique R right_ total R*
shows (*rel_ fun* (*rel_ mset R*) (*rel_ fun* (*rel_ mset R*) (*op* =)))
(*λM N*. *filter_ mset* (*Domainp R*) *M* ⊆# *filter_ mset* (*Domainp R*) *N*) (*op* ⊆#)
proof −
have *count_ filter_ mset_ less*:
$(\forall a.\ count\ (filter\_ mset\ (Domainp\ R)\ M)\ a \leq count\ (filter\_ mset\ (Domainp\ R)\ N)\ a) \longleftrightarrow$
$(\forall a \in \{x\ .\ Domainp\ R\ x\}.\ count\ M\ a \leq count\ N\ a)$ **for** *M* **and** *N* **by** *auto*
show *?thesis* **unfolding** *subseteq_ mset_ def count_ filter_ mset_ less*
by *transfer_ prover*

qed

lemma *sum_mset_transfer*[*transfer_rule*]: *R 0 0* $\Longrightarrow$ *rel_fun R* (*rel_fun R R*) *op +* *op +* $\Longrightarrow$
(*rel_fun* (*rel_mset R*) *R*) *sum_mset sum_mset*
using *sum_list_transfer*[*of R*] **unfolding** *rel_fun_def rel_mset_def* **by** *auto*

lemma *Sigma_mset_transfer*[*transfer_rule*]:
(*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_fun R* (*rel_mset S*)) (*rel_mset* (*rel_prod R S*))))
Sigma_mset Sigma_mset
by (*unfold Sigma_mset_def*) *transfer_prover*

end

3 Signed (Finite) Multisets

theory *Signed_Multiset*
imports *Multiset_More*
abbrevs
!z = $_z$
begin

3.1 Definition of Signed Multisets

definition *equiv_zmset* :: *'a multiset* × *'a multiset* ⇒ *'a multiset* × *'a multiset* ⇒ *bool* **where**
equiv_zmset = (λ(*Mp*, *Mn*) (*Np*, *Nn*). *Mp* + *Nn* = *Np* + *Mn*)

quotient-type *'a zmultiset* = *'a multiset* × *'a multiset* / *equiv_zmset*
by (*rule equivpI*, *simp_all add*: *equiv_zmset_def reflp_def symp_def transp_def*)
(*metis multi_union_self_other_eq union_lcomm*)

3.2 Basic Operations on Signed Multisets

instantiation *zmultiset* :: (*type*) *cancel_comm_monoid_add*
begin

lift-definition *zero_zmultiset* :: *'a zmultiset* **is** ({#}, {#}) .

abbreviation *empty_zmset* :: *'a zmultiset* ($\{\#\}_z$) **where**
empty_zmset ≡ *0*

lift-definition *minus_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λ(*Mp*, *Mn*) (*Np*, *Nn*). (*Mp* + *Nn*, *Mn* + *Np*)
by (*auto simp*: *equiv_zmset_def union_commute union_lcomm*)

lift-definition *plus_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λ(*Mp*, *Mn*) (*Np*, *Nn*). (*Mp* + *Np*, *Mn* + *Nn*)
by (*auto simp*: *equiv_zmset_def union_commute union_lcomm*)

instance
by (*intro_classes*; *transfer*) (*auto simp*: *equiv_zmset_def*)

end

instantiation *zmultiset* :: (*type*) *group_add*
begin

lift-definition *uminus_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* **is** λ(*Mp*, *Mn*). (*Mn*, *Mp*)
by (*auto simp*: *equiv_zmset_def add.commute*)

instance
by (*intro_classes*; *transfer*) (*auto simp*: *equiv_zmset_def*)

end

lift-definition *zcount* :: *'a zmultiset* ⇒ *'a* ⇒ *int* **is**
λ(Mp, Mn) x. int (count Mp x) − int (count Mn x)
by (*auto simp del*: *of_nat_add simp*: *equiv_zmset_def fun_eq_iff multiset_eq_iff diff_eq_eq diff_add_eq eq_diff_eq of_nat_add*[*symmetric*])

lemma *zcount_inject*: *zcount M = zcount N ⟷ M = N*
by *transfer* (*auto simp del*: *of_nat_add simp*: *equiv_zmset_def fun_eq_iff multiset_eq_iff diff_eq_eq diff_add_eq eq_diff_eq of_nat_add*[*symmetric*])

lemma *zmultiset_eq_iff*: *M = N ⟷* (*∀ a. zcount M a = zcount N a*)
by (*simp only*: *zcount_inject*[*symmetric*] *fun_eq_iff*)

lemma *zmultiset_eqI*: (*⋀x. zcount A x = zcount B x*) *⟹ A = B*
using *zmultiset_eq_iff* **by** *auto*

lemma *zcount_uminus*[*simp*]: *zcount (− A) x = − zcount A x*
by *transfer auto*

lift-definition *add_zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λx (Mp, Mn). (add_mset x Mp, Mn)
by (*auto simp*: *equiv_zmset_def*)

syntax
zmultiset :: *args* ⇒ *'a zmultiset* ({#()#}$_z$)
translations
{#*x*, *xs*#}$_z$ == *CONST add_zmset x* {#*xs*#}$_z$
{#*x*#}$_z$ == *CONST add_zmset x* {#}$_z$

lemma *zcount_empty*[*simp*]: *zcount* {#}$_z$ *a = 0*
by *transfer auto*

lemma *zcount_add_zmset*[*simp*]:
zcount (add_zmset b A) a = (if b = a then zcount A a + 1 else zcount A a)
by *transfer auto*

lemma *zcount_single*: *zcount* {#*b*#}$_z$ *a = (if b = a then 1 else 0)*
by *simp*

lemma *add_add_same_iff_zmset*[*simp*]: *add_zmset a A = add_zmset a B ⟷ A = B*
by (*auto simp*: *zmultiset_eq_iff*)

lemma *add_zmset_commute*: *add_zmset x (add_zmset y M) = add_zmset y (add_zmset x M)*
by (*auto simp*: *zmultiset_eq_iff*)

lemma
singleton_ne_empty_zmset[*simp*]: {#*x*#}$_z$ ≠ {#}$_z$ **and**
empty_ne_singleton_zmset[*simp*]: {#}$_z$ ≠ {#*x*#}$_z$
by (*auto dest*!: *arg_cong2*[*of _ _ x _ zcount*])

lemma
singleton_ne_uminus_singleton_zmset[*simp*]: {#*x*#}$_z$ ≠ − {#*y*#}$_z$ **and**
uminus_singleton_ne_singleton_zmset[*simp*]: − {#*x*#}$_z$ ≠ {#*y*#}$_z$
by (*auto dest*!: *arg_cong2*[*of _ _ x x zcount*] *split*: *if_splits*)

3.2.1 Conversion to Set and Membership

definition *set_zmset* :: *'a zmultiset* ⇒ *'a set* **where**
set_zmset M = {x. zcount M x ≠ 0}

abbreviation *elem_zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *bool* **where**
elem_zmset a M ≡ a ∈ set_zmset M

notation

elem_zmset (*op* $\in\#_z$) **and**
elem_zmset ((_ / $\in\#_z$ _) [*51*, *51*] *50*)

notation (*ASCII*)
elem_zmset (*op* :#*hy*) **and**
elem_zmset ((_ / :#*hy* _) [*51*, *51*] *50*)

abbreviation *not_elem_zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *bool* **where**
not_elem_zmset a M ≡ *a* ∉ *set_zmset M*

notation
not_elem_zmset (*op* $\notin\#_z$) **and**
not_elem_zmset ((_ / $\notin\#_z$ _) [*51*, *51*] *50*)

notation (*ASCII*)
not_elem_zmset (*op* ~:#*hy*) **and**
not_elem_zmset ((_ / ~:#*hy* _) [*51*, *51*] *50*)

context
begin

qualified abbreviation *Ball* :: *'a zmultiset* ⇒ (*'a* ⇒ *bool*) ⇒ *bool* **where**
Ball M ≡ *Set.Ball* (*set_zmset M*)

qualified abbreviation *Bex* :: *'a zmultiset* ⇒ (*'a* ⇒ *bool*) ⇒ *bool* **where**
Bex M ≡ *Set.Bex* (*set_zmset M*)

end

syntax
_ *MBall* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* ((*3*∀ _ $\in\#_z$ _./ _) [*0*, *0*, *10*] *10*)
_ *MBex* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* ((*3*∃ _ $\in\#_z$ _./ _) [*0*, *0*, *10*] *10*)

syntax (*ASCII*)
_ *MBall* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* ((*3*∀ _ :$\#_z$ _./ _) [*0*, *0*, *10*] *10*)
_ *MBex* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* ((*3*∃ _ :$\#_z$ _./ _) [*0*, *0*, *10*] *10*)

translations
∀ *x*$\in\#_z$*A*. *P* ⇌ *CONST Signed_Multiset.Ball A* (λ*x*. *P*)
∃ *x*$\in\#_z$*A*. *P* ⇌ *CONST Signed_Multiset.Bex A* (λ*x*. *P*)

lemma *zcount_eq_zero_iff*: *zcount M x* = *0* ⟷ *x* $\notin\#_z$ *M*
by (*auto simp add*: *set_zmset_def*)

lemma *not_in_iff_zmset*: *x* $\notin\#_z$ *M* ⟷ *zcount M x* = *0*
by (*auto simp add*: *zcount_eq_zero_iff*)

lemma *zcount_ne_zero_iff*[*simp*]: *zcount M x* ≠ *0* ⟷ *x* $\in\#_z$ *M*
by (*auto simp add*: *set_zmset_def*)

lemma *zcount_inI*:
assumes *zcount M x* = *0* ⟹ *False*
shows *x* $\in\#_z$ *M*
proof (*rule ccontr*)
assume *x* $\notin\#_z$ *M*
with *assms* **show** *False* **by** (*simp add*: *not_in_iff_zmset*)
qed

lemma *set_zmset_empty*[*simp*]: *set_zmset* $\{\#\}_z$ = {}
by (*simp add*: *set_zmset_def*)

lemma *set_zmset_single*: *set_zmset* $\{\#b\#\}_z$ = {*b*}
by (*simp add*: *set_zmset_def*)

lemma *set_ zmset_ eq_ empty_ iff*[*simp*]: *set_ zmset M* = {} ⟷ *M* = {#}$_z$
by (*auto simp add*: *zmultiset_ eq_ iff zcount_ eq_ zero_ iff*)

lemma *finite_ count_ ne*: *finite* {*x*. *count M x* ≠ *count N x*}
proof −
have {*x*. *count M x* ≠ *count N x*} ⊆ *set_ mset M* ∪ *set_ mset N*
by (*auto simp*: *not_ in_ iff*)
moreover have *finite* (*set_ mset M* ∪ *set_ mset N*)
by (*rule finite_ UnI*[*OF finite_ set_ mset finite_ set_ mset*])
ultimately show *?thesis*
by (*rule finite_ subset*)
qed

lemma *finite_ set_ zmset*[*iff*]: *finite* (*set_ zmset M*)
unfolding *set_ zmset_ def* **by** *transfer* (*auto intro*: *finite_ count_ ne*)

lemma *zmultiset_ nonemptyE*[*elim*]:
assumes *A* ≠ {#}$_z$
obtains *x* **where** *x* ∈#$_z$ *A*
proof −
have ∃ *x*. *x* ∈#$_z$ *A*
by (*rule ccontr*) (*insert assms*, *auto*)
with *that* **show** *?thesis*
by *blast*
qed

3.2.2 Union

lemma *zcount_ union*[*simp*]: *zcount* (*M* + *N*) *a* = *zcount M a* + *zcount N a*
by *transfer auto*

lemma *union_ add_ left_ zmset*[*simp*]: *add_ zmset a A* + *B* = *add_ zmset a* (*A* + *B*)
by (*auto simp*: *zmultiset_ eq_ iff*)

lemma *union_ zmset_ add_ zmset_ right*[*simp*]: *A* + *add_ zmset a B* = *add_ zmset a* (*A* + *B*)
by (*auto simp*: *zmultiset_ eq_ iff*)

lemma *add_ zmset_ add_ single*: ‹*add_ zmset a A* = *A* + {#*a*#}$_z$›
by (*subst union_ zmset_ add_ zmset_ right*, *subst add.comm_ neutral*) (*rule refl*)

3.2.3 Difference

lemma *zcount_ diff*[*simp*]: *zcount* (*M* − *N*) *a* = *zcount M a* − *zcount N a*
by *transfer auto*

lemma *add_ zmset_ diff_ bothsides*: ‹*add_ zmset a M* − *add_ zmset a A* = *M* − *A*›
by (*auto simp*: *zmultiset_ eq_ iff*)

lemma *in_ diff_ zcount*: *a* ∈#$_z$ *M* − *N* ⟷ *zcount N a* ≠ *zcount M a*
by (*fastforce simp*: *set_ zmset_ def*)

lemma *diff_ add_ zmset*:
fixes *M N Q* :: *'a zmultiset*
shows *M* − (*N* + *Q*) = *M* − *N* − *Q*
by (*rule sym*) (*fact diff_ diff_ add*)

lemma *insert_ Diff_ zmset*[*simp*]: *add_ zmset x* (*M* − {#*x*#}$_z$) = *M*
by (*clarsimp simp*: *zmultiset_ eq_ iff*)

lemma *diff_ union_ swap_ zmset*: *add_ zmset b* (*M* − {#*a*#}$_z$) = *add_ zmset b M* − {#*a*#}$_z$
by (*auto simp add*: *zmultiset_ eq_ iff*)

lemma *diff_ add_ zmset_ swap*[*simp*]: *add_ zmset b M* − *A* = *add_ zmset b* (*M* − *A*)

by (*auto simp add*: *zmultiset_eq_iff*)

lemma *diff_diff_add_zmset*[*simp*]: $(M :: 'a\ zmultiset) - N - P = M - (N + P)$
by (*rule diff_diff_add*)

lemma *zmset_add*[*elim?*]:
obtains B **where** $A = add\_zmset\ a\ B$
proof −
have $A = add\_zmset\ a\ (A - \{\#a\#\}_z)$
by *simp*
with *that* **show** *thesis* **.**
qed

3.2.4 Equality of Signed Multisets

lemma *single_eq_single_zmset*[*simp*]: $\{\#a\#\}_z = \{\#b\#\}_z \longleftrightarrow a = b$
by (*auto simp add*: *zmultiset_eq_iff*)

lemma *multi_self_add_other_not_self_zmset*[*simp*]: $M = add\_zmset\ x\ M \longleftrightarrow False$
by (*auto simp add*: *zmultiset_eq_iff*)

lemma *add_zmset_remove_trivial*: ‹$add\_zmset\ x\ M - \{\#x\#\}_z = M$›
by *simp*

lemma *diff_single_eq_union_zmset*: $M - \{\#x\#\}_z = N \longleftrightarrow M = add\_zmset\ x\ N$
by *auto*

lemma *union_single_eq_diff_zmset*: $add\_zmset\ x\ M = N \Longrightarrow M = N - \{\#x\#\}_z$
unfolding *add_zmset_add_single*[*of* _ *M*] **by** (*fact add_implies_diff*)

lemma *add_zmset_eq_conv_diff*:
$add\_zmset\ a\ M = add\_zmset\ b\ N \longleftrightarrow$
$M = N \wedge a = b \vee M = add\_zmset\ b\ (N - \{\#a\#\}_z) \wedge N = add\_zmset\ a\ (M - \{\#b\#\}_z)$
by (*simp add*: *zmultiset_eq_iff*) *fastforce*

lemma *add_zmset_eq_conv_ex*:
$(add\_zmset\ a\ M = add\_zmset\ b\ N) =$
$(M = N \wedge a = b \vee (\exists K.\ M = add\_zmset\ b\ K \wedge N = add\_zmset\ a\ K))$
by (*auto simp add*: *add_zmset_eq_conv_diff*)

lemma *multi_member_split*: $\exists A.\ M = add\_zmset\ x\ A$
by (*rule exI*[**where** $x = M - \{\#x\#\}_z$]) *simp*

3.3 Conversions from and to Multisets

lift-definition *zmset_of* :: $'a\ multiset \Rightarrow 'a\ zmultiset$ **is** $\lambda f.\ (Abs\_multiset\ f, \{\#\})$ **.**

lemma *zmset_of_inject*[*simp*]: $zmset\_of\ M = zmset\_of\ N \longleftrightarrow M = N$
by (*simp add*: *zmset_of_def*, *transfer*, *auto simp*: *equiv_zmset_def*)

lemma *zmset_of_empty*[*simp*]: $zmset\_of\ \{\#\} = \{\#\}_z$
by (*simp add*: *zmset_of_def zero_zmultiset_def*)

lemma *zmset_of_add_mset*[*simp*]: $zmset\_of\ (add\_mset\ x\ M) = add\_zmset\ x\ (zmset\_of\ M)$
by *transfer* (*auto simp*: *equiv_zmset_def add_mset_def cong*: *if_cong*)

lemma *zcount_of_mset*[*simp*]: $zcount\ (zmset\_of\ M)\ x = int\ (count\ M\ x)$
by (*induct M*) *auto*

lemma *zmset_of_plus*: $zmset\_of\ (M + N) = zmset\_of\ M + zmset\_of\ N$
by (*transfer*, *auto simp*: *equiv_zmset_def eq_onp_same_args plus_multiset.abs_eq*)+

lift-definition *mset_pos* :: $'a\ zmultiset \Rightarrow 'a\ multiset$ **is** $\lambda(Mp, Mn).\ count\ (Mp - Mn)$
by (*clarsimp simp*: *equiv_zmset_def intro*!: *arg_cong*[*of* _ _ *count*])

(*metis add.commute add_diff_cancel_right*)

lift-definition *mset_neg* :: *'a zmultiset* ⇒ *'a multiset* **is** λ(*Mp*, *Mn*). *count* (*Mn* − *Mp*)
by (*clarsimp simp*: *equiv_zmset_def intro*!: *arg_cong*[*of* _ _ *count*])
(*metis add.commute add_diff_cancel_right*)

lemma
zmset_of_inverse[*simp*]: *mset_pos* (*zmset_of M*) = *M* **and**
minus_zmset_of_inverse[*simp*]: *mset_neg* (− *zmset_of M*) = *M*
by (*transfer*, *simp*)+

lemma *neg_zmset_pos*[*simp*]: *mset_neg* (*zmset_of M*) = {#}
by (*rule zmset_of_inject*[*THEN iffD1*], *simp*, *transfer*, *auto simp*: *equiv_zmset_def*)+

lemma
count_mset_pos[*simp*]: *count* (*mset_pos M*) *x* = *nat* (*zcount M x*) **and**
count_mset_neg[*simp*]: *count* (*mset_neg M*) *x* = *nat* (− *zcount M x*)
by (*transfer*; *auto*)+

lemma
mset_pos_empty[*simp*]: *mset_pos* {#}$_z$ = {#} **and**
mset_neg_empty[*simp*]: *mset_neg* {#}$_z$ = {#}
by (*rule multiset_eqI*, *simp*)+

lemma
mset_pos_singleton[*simp*]: *mset_pos* {#*x*#}$_z$ = {#*x*#} **and**
mset_neg_singleton[*simp*]: *mset_neg* {#*x*#}$_z$ = {#}
by (*rule multiset_eqI*, *simp*)+

lemma
mset_pos_neg_partition: *M* = *zmset_of* (*mset_pos M*) − *zmset_of* (*mset_neg M*) **and**
mset_pos_as_neg: *zmset_of* (*mset_pos M*) = *zmset_of* (*mset_neg M*) + *M* **and**
mset_neg_as_pos: *zmset_of* (*mset_neg M*) = *zmset_of* (*mset_pos M*) − *M*
by (*rule zmultiset_eqI*, *simp*)+

lemma *mset_pos_uminus*[*simp*]: *mset_pos* (− *A*) = *mset_neg A*
by (*rule multiset_eqI*) *simp*

lemma *mset_neg_uminus*[*simp*]: *mset_neg* (− *A*) = *mset_pos A*
by (*rule multiset_eqI*) *simp*

lemma *mset_pos_plus*[*simp*]:
mset_pos (*A* + *B*) = (*mset_pos A* − *mset_neg B*) + (*mset_pos B* − *mset_neg A*)
by (*rule multiset_eqI*) *simp*

lemma *mset_neg_plus*[*simp*]:
mset_neg (*A* + *B*) = (*mset_neg A* − *mset_pos B*) + (*mset_neg B* − *mset_pos A*)
by (*rule multiset_eqI*) *simp*

lemma *mset_pos_diff*[*simp*]:
mset_pos (*A* − *B*) = (*mset_pos A* − *mset_pos B*) + (*mset_neg B* − *mset_neg A*)
by (*rule mset_pos_plus*[*of A* − *B*, *simplified*])

lemma *mset_neg_diff*[*simp*]:
mset_neg (*A* − *B*) = (*mset_neg A* − *mset_neg B*) + (*mset_pos B* − *mset_pos A*)
by (*rule mset_neg_plus*[*of A* − *B*, *simplified*])

lemma *mset_pos_neg_dual*:
mset_pos a + *mset_pos b* + (*mset_neg a* − *mset_pos b*) + (*mset_neg b* − *mset_pos a*) =
mset_neg a + *mset_neg b* + (*mset_pos a* − *mset_neg b*) + (*mset_pos b* − *mset_neg a*)
using [[*linarith_split_limit* = *20*]] **by** (*rule multiset_eqI*) *simp*

lemma *decompose_zmset_of2*:

obtains $A\ B\ C$ **where**
$M = zmset\_of\ A + C$ **and**
$N = zmset\_of\ B + C$
proof
let $?A = zmset\_of\ (mset\_pos\ M + mset\_neg\ N)$
let $?B = zmset\_of\ (mset\_pos\ N + mset\_neg\ M)$
let $?C = -\ (zmset\_of\ (mset\_neg\ M) + zmset\_of\ (mset\_neg\ N))$

show $M = ?A + ?C$
by (*simp add*: *zmset_of_plus mset_pos_neg_partition*)
show $N = ?B + ?C$
by (*simp add*: *zmset_of_plus diff_add_zmset mset_pos_neg_partition*)
qed

3.3.1 Pointwise Ordering Induced by *zcount*

definition *subseteq_zmset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ (**infix** $\subseteq\#_z$ *50*) **where**
$A \subseteq\#_z B \longleftrightarrow (\forall a.\ zcount\ A\ a \leq zcount\ B\ a)$

definition *subset_zmset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ (**infix** $\subset\#_z$ *50*) **where**
$A \subset\#_z B \longleftrightarrow A \subseteq\#_z B \wedge A \neq B$

abbreviation (*input*)
supseteq_zmset :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ (**infix** $\supseteq\#_z$ *50*)
where
$supseteq\_zmset\ A\ B \equiv B \subseteq\#_z A$

abbreviation (*input*)
supset_zmset :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ (**infix** $\supset\#_z$ *50*)
where
$supset\_zmset\ A\ B \equiv B \subset\#_z A$

notation (*input*)
subseteq_zmset (**infix** $\leq\#_z$ *50*) **and**
supseteq_zmset (**infix** $\geq\#_z$ *50*)

notation (*ASCII*)
subseteq_zmset (**infix** $<=\#_z$ *50*) **and**
subset_zmset (**infix** $<\#_z$ *50*) **and**
supseteq_zmset (**infix** $>=\#_z$ *50*) **and**
supset_zmset (**infix** $>\#_z$ *50*)

interpretation *subset_zmset*: *ordered_ab_semigroup_add_imp_le op + op − op* $\subseteq\#_z$ *op* $\subset\#_z$
by *unfold_locales* (*auto simp add*: *subset_zmset_def subseteq_zmset_def zmultiset_eq_iff intro*: *order_trans antisym*)

interpretation *subset_zmset*:
ordered_ab_semigroup_monoid_add_imp_le op + 0 op − op $\leq\#_z$ *op* $<\#_z$
by *unfold_locales*

lemma *zmset_subset_eqI*: $(\bigwedge a.\ zcount\ A\ a \leq zcount\ B\ a) \Longrightarrow A \subseteq\#_z B$
by (*simp add*: *subseteq_zmset_def*)

lemma *zmset_subset_eq_zcount*: $A \subseteq\#_z B \Longrightarrow zcount\ A\ a \leq zcount\ B\ a$
by (*simp add*: *subseteq_zmset_def*)

lemma *zmset_subset_eq_add_zmset_cancel*: $\langle add\_zmset\ a\ A \subseteq\#_z add\_zmset\ a\ B \longleftrightarrow A \subseteq\#_z B\rangle$
unfolding *add_zmset_add_single*[*of* _ *A*] *add_zmset_add_single*[*of* _ *B*]
by (*rule subset_zmset.add_le_cancel_right*)

lemma *zmset_subset_eq_zmultiset_union_diff_commute*:
$A - B + C = A + C - B$ **for** $A\ B\ C :: 'a\ zmultiset$
by (*simp add*: *add.commute add_diff_eq*)

lemma *zmset_subset_eq_insertD*: *add_zmset x A* $\subseteq\#_z$ *B* $\Longrightarrow$ *A* $\subset\#_z$ *B*
unfolding *subset_zmset_def subseteq_zmset_def*
by (*metis* (*no_types*) *add.commute add_le_same_cancel2 zcount_add_zmset dual_order.trans le_cases le_numeral_extra*(*2*))

lemma *zmset_subset_insertD*: *add_zmset x A* $\subset\#_z$ *B* $\Longrightarrow$ *A* $\subset\#_z$ *B*
by (*rule zmset_subset_eq_insertD*) (*rule subset_zmset.less_imp_le*)

lemma *subset_eq_diff_conv_zmset*: *A* − *C* $\subseteq\#_z$ *B* $\longleftrightarrow$ *A* $\subseteq\#_z$ *B* + *C*
by (*simp add*: *subseteq_zmset_def ordered_ab_group_add_class.diff_le_eq*)

lemma *multi_psub_of_add_self_zmset*[*simp*]: *A* $\subset\#_z$ *add_zmset x A*
by (*auto simp*: *subset_zmset_def subseteq_zmset_def*)

lemma *multi_psub_self_zmset*: *A* $\subset\#_z$ *A* = *False*
by *simp*

lemma *zmset_subset_add_zmset*[*simp*]: *add_zmset x N* $\subset\#_z$ *add_zmset x M* $\longleftrightarrow$ *N* $\subset\#_z$ *M*
unfolding *add_zmset_add_single*[*of _ N*] *add_zmset_add_single*[*of _ M*]
by (*fact subset_zmset.add_less_cancel_right*)

lemma *zmset_of_subseteq_iff*[*simp*]: *zmset_of M* $\subseteq\#_z$ *zmset_of N* $\longleftrightarrow$ *M* $\subseteq\#$ *N*
by (*simp add*: *subseteq_zmset_def subseteq_mset_def*)

lemma *zmset_of_subset_iff*[*simp*]: *zmset_of M* $\subset\#_z$ *zmset_of N* $\longleftrightarrow$ *M* $\subset\#$ *N*
by (*simp add*: *subset_zmset_def subset_mset_def*)

lemma
mset_pos_supset: *A* $\subseteq\#_z$ *zmset_of* (*mset_pos A*) **and**
mset_neg_supset: − *A* $\subseteq\#_z$ *zmset_of* (*mset_neg A*)
by (*auto intro*: *zmset_subset_eqI*)

lemma *subset_mset_zmsetE*:
assumes *M* $\subset\#_z$ *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* $\subset\#$ *B*
by (*metis assms decompose_zmset_of2 subset_zmset.add_less_cancel_right zmset_of_subset_iff*)

lemma *subseteq_mset_zmsetE*:
assumes *M* $\subseteq\#_z$ *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* $\subseteq\#$ *B*
by (*metis assms add.commute add.right_neutral subset_mset.order_refl subset_mset_def subset_mset_zmsetE subset_zmset_def zmset_of_empty*)

3.3.2 Subset is an Order

interpretation *subset_zmset*: *order op* $\subseteq\#_z$ *op* $\subset\#_z$
by *unfold_locales*

3.4 Replicate and Repeat Operations

definition *replicate_zmset* :: *nat* ⇒ *'a* ⇒ *'a zmultiset* **where**
replicate_zmset n x = (*add_zmset x* ^^ *n*) $\{\#\}_z$

lemma *replicate_zmset_0*[*simp*]: *replicate_zmset 0 x* = $\{\#\}_z$
unfolding *replicate_zmset_def* **by** *simp*

lemma *replicate_zmset_Suc*[*simp*]: *replicate_zmset* (*Suc n*) *x* = *add_zmset x* (*replicate_zmset n x*)
unfolding *replicate_zmset_def* **by** (*induct n*) (*auto intro*: *add.commute*)

lemma *count_replicate_zmset*[*simp*]:
zcount (*replicate_zmset n x*) *y* = (*if y* = *x then of_nat n else 0*)
unfolding *replicate_zmset_def* **by** (*induct n*) *auto*

fun *repeat_zmset* :: *nat* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **where**
repeat_zmset 0 _ = {#}$_z$ |
repeat_zmset (*Suc n*) *A* = *A* + *repeat_zmset n A*

lemma *count_repeat_zmset*[*simp*]: *zcount* (*repeat_zmset i A*) *a* = *of_nat i* ∗ *zcount A a*
by (*induct i*) (*auto simp*: *semiring_normalization_rules*(*3*))

lemma *repeat_zmset_right*[*simp*]: *repeat_zmset a* (*repeat_zmset b A*) = *repeat_zmset* (*a* ∗ *b*) *A*
by (*auto simp*: *zmultiset_eq_iff left_diff_distrib'*)

lemma *left_diff_repeat_zmset_distrib'*:
⟨*i* ≥ *j* ⟹ *repeat_zmset* (*i* − *j*) *u* = *repeat_zmset i u* − *repeat_zmset j u*⟩
by (*auto simp*: *zmultiset_eq_iff int_distrib*(*3*) *of_nat_diff*)

lemma *left_add_mult_distrib_zmset*:
repeat_zmset i u + (*repeat_zmset j u* + *k*) = *repeat_zmset* (*i*+*j*) *u* + *k*
by (*auto simp*: *zmultiset_eq_iff add_mult_distrib int_distrib*(*1*))

lemma *repeat_zmset_distrib*: *repeat_zmset* (*m* + *n*) *A* = *repeat_zmset m A* + *repeat_zmset n A*
by (*auto simp*: *zmultiset_eq_iff Nat.add_mult_distrib int_distrib*(*1*))

lemma *repeat_zmset_distrib2*[*simp*]:
repeat_zmset n (*A* + *B*) = *repeat_zmset n A* + *repeat_zmset n B*
by (*auto simp*: *zmultiset_eq_iff add_mult_distrib2 int_distrib*(*2*))

lemma *repeat_zmset_replicate_zmset*[*simp*]: *repeat_zmset n* {#*a*#}$_z$ = *replicate_zmset n a*
by (*auto simp*: *zmultiset_eq_iff*)

lemma *repeat_zmset_distrib_add_zmset*[*simp*]:
repeat_zmset n (*add_zmset a A*) = *replicate_zmset n a* + *repeat_zmset n A*
by (*auto simp*: *zmultiset_eq_iff int_distrib*(*2*))

lemma *repeat_zmset_empty*[*simp*]: *repeat_zmset n* {#}$_z$ = {#}$_z$
by (*induct n*) *simp_all*

3.4.1 Filter (with Comprehension Syntax)

lift-definition *filter_zmset* :: (*'a* ⇒ *bool*) ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λ*P* (*Mp*, *Mn*). (*filter_mset P Mp*, *filter_mset P Mn*)
by (*auto simp del*: *filter_union_mset simp*: *equiv_zmset_def filter_union_mset*[*symmetric*])

syntax (*ASCII*)
_MCollect :: *pttrn* ⇒ *'a zmultiset* ⇒ *bool* ⇒ *'a zmultiset* ((*1*{#*_* :#*hy _*./ *_*#}))
syntax
_MCollect :: *pttrn* ⇒ *'a zmultiset* ⇒ *bool* ⇒ *'a zmultiset* ((*1*{#*_* ∈#$_z$ *_*./ *_*#}))
translations
{#*x* ∈#$_z$ *M*. *P*#} == *CONST filter_zmset* (λ*x*. *P*) *M*

lemma *count_filter_zmset*[*simp*]:
zcount (*filter_zmset P M*) *a* = (*if P a then zcount M a else 0*)
by *transfer auto*

lemma *filter_empty_zmset*[*simp*]: *filter_zmset P* {#}$_z$ = {#}$_z$
by (*rule zmultiset_eqI*) *simp*

lemma *filter_single_zmset*: *filter_zmset P* {#*x*#}$_z$ = (*if P x then* {#*x*#}$_z$ *else* {#}$_z$)
by (*rule zmultiset_eqI*) *simp*

lemma *filter_union_zmset*[*simp*]: *filter_zmset P* (*M* + *N*) = *filter_zmset P M* + *filter_zmset P N*
by (*rule zmultiset_eqI*) *simp*

lemma *filter_diff_zmset*[*simp*]: *filter_zmset P* (*M* − *N*) = *filter_zmset P M* − *filter_zmset P N*
by (*rule zmultiset_eqI*) *simp*

lemma *filter_ add_ zmset*[*simp*]:
filter_ zmset P (*add_ zmset x A*) =
(*if P x then add_ zmset x* (*filter_ zmset P A*) *else filter_ zmset P A*)
by (*auto simp*: *zmultiset_ eq_ iff*)

lemma *zmultiset_ filter_ mono*:
assumes $A \subseteq\#_z B$
shows *filter_ zmset f A* $\subseteq\#_z$ *filter_ zmset f B*
using *assms* **by** (*simp add*: *subseteq_ zmset_ def*)

lemma *filter_ filter_ zmset*: *filter_ zmset P* (*filter_ zmset Q M*) = {#*x* ∈# *M*. *Q x* ∧ *P x*#}
by (*auto simp*: *zmultiset_ eq_ iff*)

lemma
filter_ zmset_ True[*simp*]: {#*y* ∈#$_z$ *M*. *True*#} = *M* **and**
filter_ zmset_ False[*simp*]: {#*y* ∈#$_z$ *M*. *False*#} = {#}$_z$
by (*auto simp*: *zmultiset_ eq_ iff*)

3.5 Uncategorized

lemma *multi_ drop_ mem_ not_ eq_ zmset*: *B* − {#*c*#}$_z$ ≠ *B*
by (*simp add*: *diff_ single_ eq_ union_ zmset*)

lemma *zmultiset_ partition*: *M* = {#*x* ∈#$_z$ *M*. *P x* #} + {#*x* ∈#$_z$ *M*. ¬ *P x*#}
by (*subst zmultiset_ eq_ iff*) *auto*

3.6 Image

definition *image_ zmset* :: ('*a* ⇒ '*b*) ⇒ '*a zmultiset* ⇒ '*b zmultiset* **where**
image_ zmset f M =
zmset_ of (*fold_ mset* (*add_ mset* ∘ *f*) {#} (*mset_ pos M*)) −
zmset_ of (*fold_ mset* (*add_ mset* ∘ *f*) {#} (*mset_ neg M*))

3.7 Multiset Order

instantiation *zmultiset* :: (*preorder*) *order*
begin

lift-definition *less_ zmultiset* :: '*a zmultiset* ⇒ '*a zmultiset* ⇒ *bool* **is**
λ(*Mp*, *Mn*) (*Np*, *Nn*). *Mp* + *Nn* < *Mn* + *Np*
proof (*clarsimp simp*: *equiv_ zmset_ def*)
fix *A1 B2 B1 A2 C1 D2 D1 C2* :: '*a multiset*
assume
ab: *A1* + *A2* = *B1* + *B2* **and**
cd: *C1* + *C2* = *D1* + *D2*

have *A1* + *D2* < *B2* + *C1* ⟷ *A1* + *A2* + *D2* < *A2* + *B2* + *C1*
by *simp*
also have ... ⟷ *B1* + *B2* + *D2* < *A2* + *B2* + *C1*
unfolding *ab* **by** (*rule refl*)
also have ... ⟷ *B1* + *D2* < *A2* + *C1*
by *simp*
also have ... ⟷ *B1* + *D1* + *D2* < *A2* + *C1* + *D1*
by *simp*
also have ... ⟷ *B1* + *C1* + *C2* < *A2* + *C1* + *D1*
using *cd* **by** (*simp add*: *add.assoc*)
also have ... ⟷ *B1* + *C2* < *A2* + *D1*
by *simp*
finally show *A1* + *D2* < *B2* + *C1* ⟷ *B1* + *C2* < *A2* + *D1*
by *assumption*
qed

definition *less_ eq_ zmultiset* :: '*a zmultiset* ⇒ '*a zmultiset* ⇒ *bool* **where**

```
  less_eq_zmultiset M' M ⟷ M' < M ∨ M' = M

instance
proof ((intro_classes; unfold less_eq_zmultiset_def; transfer),
    auto simp: equiv_zmset_def union_commute)
  fix A1 B1 D C B2 A2 :: 'a multiset
  assume ab: A1 + A2 ≠ B1 + B2

  {
    assume ab1: A1 + C < B1 + D

    {
      assume ab2: D + A2 < C + B2
      show A1 + A2 < B1 + B2
      proof −
        have f1: ⋀m. D + A2 + m < C + B2 + m
          using ab2 add_less_cancel_right by blast
        have ⋀m. C + (A1 + m) < D + (B1 + m)
          by (simp add: ab1 add.commute)
        then have D + (A2 + A1) < D + (B1 + B2)
          using f1 by (metis add.assoc add.commute mset_le_trans)
        then show ?thesis
          by (simp add: add.commute)
      qed
    }
    {
      assume ab2: D + A2 = C + B2
      show A1 + A2 < B1 + B2
      proof −
        have ⋀m. C + A1 + m < D + B1 + m
          by (simp add: ab1 add.commute)
        then have D + (A2 + A1) < D + (B1 + B2)
          by (metis (no_types) ab2 add.assoc add.commute)
        then show ?thesis
          by (simp add: add.commute)
      qed
    }
  }

  {
    assume ab1: A1 + C = B1 + D

    {
      assume ab2: D + A2 < C + B2
      show A1 + A2 < B1 + B2
      proof −
        have A1 + (D + A2) < B1 + (D + B2)
          by (metis (no_types) ab1 ab2 add.assoc add_less_cancel_left)
        then show ?thesis
          by simp
      qed
    }
    {
      assume ab2: D + A2 = C + B2
      have False
        by (metis (no_types) ab ab1 ab2 add.assoc add.commute add_diff_cancel_right')
      thus A1 + A2 < B1 + B2
        by sat
    }
  }
qed

end
```

instance *zmultiset* :: (*preorder*) *ordered_cancel_comm_monoid_add*
by (*intro_classes*, *unfold less_eq_zmultiset_def*, *transfer*, *auto simp*: *equiv_zmset_def*)

instance *zmultiset* :: (*preorder*) *ordered_ab_group_add*
by (*intro_classes*; *transfer*; *auto simp*: *equiv_zmset_def*)

instantiation *zmultiset* :: (*linorder*) *distrib_lattice*
begin

definition *inf_zmultiset* :: 'a *zmultiset* ⇒ 'a *zmultiset* ⇒ 'a *zmultiset* **where**
inf_zmultiset A B = (*if A* < *B then A else B*)

definition *sup_zmultiset* :: 'a *zmultiset* ⇒ 'a *zmultiset* ⇒ 'a *zmultiset* **where**
sup_zmultiset A B = (*if B* > *A then B else A*)

lemma *not_lt_iff_ge_zmset*: ¬ *x* < *y* ⟷ *x* ≥ *y* **for** *x y* :: 'a *zmultiset*
by (*unfold less_eq_zmultiset_def*, *transfer*, *auto simp*: *equiv_zmset_def algebra_simps*)

instance
by *intro_classes* (*auto simp*: *less_eq_zmultiset_def inf_zmultiset_def sup_zmultiset_def*
dest!: *not_lt_iff_ge_zmset*[*THEN iffD1*])

end

lemma *zmset_of_less*: *zmset_of M* < *zmset_of N* ⟷ *M* < *N*
by (*clarsimp simp*: *zmset_of_def*, *transfer*, *simp*)+

lemma *zmset_of_le*: *zmset_of M* ≤ *zmset_of N* ⟷ *M* ≤ *N*
by (*simp_all add*: *less_eq_zmultiset_def zmset_of_def*; *transfer*; *auto simp*: *equiv_zmset_def*)

instance *zmultiset* :: (*preorder*) *ordered_ab_semigroup_add*
by (*intro_classes*, *unfold less_eq_zmultiset_def*, *transfer*, *auto simp*: *equiv_zmset_def*)

instance *zmultiset* :: (*preorder*) *ordered_ab_semigroup_add_imp_le*
by (*intro_classes*; *unfold less_eq_zmultiset_def*; *transfer*; *auto simp*: *equiv_zmset_def*)

instance *zmultiset* :: (*linorder*) *linordered_cancel_ab_semigroup_add*
by (*intro_classes*, *unfold less_eq_zmultiset_def*, *transfer*, *auto simp*: *equiv_zmset_def add.commute*)

lemma *less_mset_zmsetE*:
assumes *M* < *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* < *B*
by (*metis add_less_imp_less_right assms decompose_zmset_of2 zmset_of_less*)

lemma *less_eq_mset_zmsetE*:
assumes *M* ≤ *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* ≤ *B*
by (*metis add.commute add.right_neutral assms le_neq_trans less_imp_le less_mset_zmsetE order_refl*
zmset_of_empty)

lemma *subset_eq_imp_le_zmset*: *M* ⊆#$_z$ *N* ⟹ *M* ≤ *N*
by (*metis* (*no_types*) *add_mono_thms_linordered_semiring*(*3*) *subset_eq_imp_le_multiset*
subseteq_mset_zmsetE zmset_of_le)

lemma *subset_imp_less_zmset*: *M* ⊂#$_z$ *N* ⟹ *M* < *N*
by (*metis le_neq_trans subset_eq_imp_le_zmset subset_zmset_def*)

lemma *lt_imp_ex_zcount_lt*:
assumes *m_lt_n*: *M* < *N*
shows ∃ *y*. *zcount M y* < *zcount N y*

proof (*rule ccontr*, *clarsimp*)
  assume $\forall y.\ \neg\ zcount\ M\ y < zcount\ N\ y$
  hence $\forall y.\ zcount\ M\ y \geq zcount\ N\ y$
    by (*simp add*: *leI*)
  hence $M \supseteq\#_z\ N$
    by (*simp add*: *zmset_subset_eqI*)
  hence $M \geq N$
    by (*simp add*: *subset_eq_imp_le_zmset*)
  thus *False*
    using *m_lt_n* **by** *simp*
qed

end

4 Nested Multisets

theory *Nested_Multiset*
imports ~~/*src/HOL/Library/Multiset_Order*
begin

declare *multiset.map_comp* [*simp*]
declare *multiset.map_cong* [*cong*]

4.1 Type Definition

datatype *'a nmultiset* =
  Elem 'a
| *MSet 'a nmultiset multiset*

inductive *no_elem* **where**
  $(\bigwedge X.\ X \in\#\ M \Longrightarrow no\_elem\ X) \Longrightarrow no\_elem\ (MSet\ M)$

inductive-set *nmultiset_sub* **where**
  $X \in\#\ M \Longrightarrow (X,\ MSet\ M) \in nmultiset\_sub$

lemma *wf_nmultiset_sub*[*simp*]: *wf nmultiset_sub*
proof (*rule wfUNIVI*)
  fix $P :: 'a\ nmultiset \Rightarrow bool$ **and** $M :: 'a\ nmultiset$
  assume *IH*: $\forall M.\ (\forall N.\ (N,\ M) \in nmultiset\_sub \longrightarrow P\ N) \longrightarrow P\ M$
  show *P M*
    by (*induct M*; *rule IH*[*rule_format*]) (*auto simp*: *nmultiset_sub.simps*)
qed

primrec *depth_nmultiset* (|_|) **where**
  $|Elem\ a| = 0$
| $|MSet\ M| =$
    (*let X = set_mset* (*image_mset depth_nmultiset M*)
    in if X = {} then 0 else Suc (*Max X*))

lemma *depth_nmultiset_MSet*: $x \in\#\ M \Longrightarrow |x| < |MSet\ M|$
  by (*auto simp*: *less_Suc_eq_le*)

declare *depth_nmultiset.simps*(*2*)[*simp del*]

4.2 Dershowitz and Manna's Nested Multiset Order

The Dershowitz–Manna extension:

definition *less_multiset_ext*$_{DM}$ **where**
  less_multiset_ext$_{DM}$ *R M N* $\longleftrightarrow$
   $(\exists X\ Y.\ X \neq \{\#\} \wedge X \leq\#\ N \wedge M = (N - X) + Y \wedge (\forall k.\ k \in\#\ Y \longrightarrow (\exists a.\ a \in\#\ X \wedge R\ k\ a)))$

lemma *less_multiset_ext*$_{DM}$*_imp_mult*:

```
  assumes
    N_A: set_mset N ⊆ A and M_A: set_mset M ⊆ A and less: less_multiset_ext_DM R M N
  shows (M, N) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
proof −
  from less obtain X Y where
    X ≠ {#} and X ≤# N and M = N − X + Y and ∀k. k ∈# Y ⟶ (∃a. a ∈# X ∧ R k a)
    unfolding less_multiset_ext_DM_def by blast
  with N_A M_A have (N − X + Y, N − X + X) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
    by (intro one_step_implies_mult, blast,
      metis (mono_tags, lifting) case_prodI mem_Collect_eq mset_subset_eqD mset_subset_eq_add_right
        subsetCE)
  with ‹M = N − X + Y› ‹X ≤# N› show (M, N) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
    by (simp add: subset_mset.diff_add)
qed

lemma mult_imp_less_multiset_ext_DM:
  assumes
    N_A: set_mset N ⊆ A and M_A: set_mset M ⊆ A and
    trans: ∀x ∈ A. ∀y ∈ A. ∀z ∈ A. R x y ⟶ R y z ⟶ R x z and
    in_mult: (M, N) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
  shows less_multiset_ext_DM R M N
  using in_mult N_A M_A unfolding mult_def less_multiset_ext_DM_def
proof induct
  case (base N)
  then obtain y M0 X where N = add_mset y M0 and M = M0 + X and ∀a. a ∈# X ⟶ R a y
    unfolding mult1_def by auto
  thus ?case
    by (auto intro: exI[of _ {#y#}])
next
  case (step N N')
  note N_N'_in_mult1 = this(2) and ih = this(3) and N'_A = this(4) and M_A = this(5)

  have N_A: set_mset N ⊆ A
    using N_N'_in_mult1 N'_A unfolding mult1_def by auto

  obtain Y X where y_nemp: Y ≠ {#} and y_sub_N: Y ⊆# N and M_eq: M = N − Y + X and
    ex_y: ∀x. x ∈# X ⟶ (∃y. y ∈# Y ∧ R x y)
    using ih[OF N_A M_A] by blast

  obtain z M0 Ya where N'_eq: N' = M0 + {#z#} and N_eq: N = M0 + Ya and
    z_gt: ∀y. y ∈# Ya ⟶ y ∈ A ∧ z ∈ A ∧ R y z
    using N_N'_in_mult1[unfolded mult1_def] by auto

  let ?Za = Y − Ya + {#z#}
  let ?Xa = X + Ya + (Y − Ya) − Y

  have xa_sub_x_ya: set_mset ?Xa ⊆ set_mset (X + Ya)
    by (metis diff_subset_eq_self in_diffD subsetI subset_mset.diff_diff_right)

  have x_A: set_mset X ⊆ A
    using M_A M_eq by auto
  have ya_A: set_mset Ya ⊆ A
    by (simp add: subsetI z_gt)

  have ex_y': ∃y. y ∈# Y − Ya + {#z#} ∧ R x y if x_in: x ∈# X + Ya for x
  proof (cases x ∈# X)
    case True
    then obtain y where y_in: y ∈# Y and y_gt_x: R x y
      using ex_y by blast
    show ?thesis
    proof (cases y ∈# Ya)
      case False
      hence y ∈# Y − Ya + {#z#}
```

```
      using y_in count_greater_zero_iff in_diff_count by fastforce
    thus ?thesis
      using y_gt_x by blast
  next
    case True
    hence y ∈ A and z ∈ A and R y z
      using z_gt by blast+
    hence R x z
      using trans y_gt_x x_A ya_A x_in by (meson subsetCE union_iff)
    thus ?thesis
      by auto
  qed
next
  case False
  hence x ∈# Ya
    using x_in by auto
  hence x ∈ A and z ∈ A and R x z
    using z_gt by blast+
  thus ?thesis
    by auto
qed

show ?case
proof (rule exI[of _ ?Za], rule exI[of _ ?Xa], intro conjI)
  show Y − Ya + {#z#} ⊆# N'
    using mset_subset_eq_mono_add subset_eq_diff_conv y_sub_N N_eq N'_eq
    by (simp add: subset_eq_diff_conv)
next
  show M = N' − (Y − Ya + {#z#}) + (X + Ya + (Y − Ya) − Y)
    unfolding M_eq N_eq N'_eq by (auto simp: multiset_eq_iff)
next
  show ∀x. x ∈# X + Ya + (Y − Ya) − Y ⟶ (∃y. y ∈# Y − Ya + {#z#} ∧ R x y)
    using ex_y' xa_sub_x_ya by blast
qed auto
qed

lemma less_multiset_ext_DM_iff_mult:
  assumes
    N_A: set_mset N ⊆ A and M_A: set_mset M ⊆ A and
    trans: ∀x ∈ A. ∀y ∈ A. ∀z ∈ A. R x y ⟶ R y z ⟶ R x z
  shows less_multiset_ext_DM R M N ⟷ (M, N) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
  using mult_imp_less_multiset_ext_DM[OF assms] less_multiset_ext_DM_imp_mult[OF N_A M_A] by blast

instantiation nmultiset :: (preorder) preorder
begin

lemma less_multiset_ext_DM_cong[fundef_cong]:
  (⋀X Y k a. X ≠ {#} ⟹ X ≤# N ⟹ M = (N − X) + Y ⟹ k ∈# Y ⟹ R k a = S k a) ⟹
  less_multiset_ext_DM R M N = less_multiset_ext_DM S M N
  unfolding less_multiset_ext_DM_def by metis

function (sequential) less_nmultiset :: 'a nmultiset ⇒ 'a nmultiset ⇒ bool where
  less_nmultiset (Elem a) (Elem b) = (a < b)
| less_nmultiset (Elem a) (MSet M) = True
| less_nmultiset (MSet M) (Elem a) = False
| less_nmultiset (MSet M) (MSet N) = less_multiset_ext_DM less_nmultiset M N
  by pat_completeness auto
termination
  by (relation nmultiset_sub <*lex*> nmultiset_sub, fastforce,
    metis nmultiset_sub.simps in_lex_prod mset_subset_eqD mset_subset_eq_add_right)

lemmas less_nmultiset_induct =
  less_nmultiset.induct[case_names Elem_Elem Elem_MSet MSet_Elem MSet_MSet]
```

lemmas *less_nmultiset_cases* =
less_nmultiset.cases[*case_names Elem_Elem Elem_MSet MSet_Elem MSet_MSet*]

lemma *trans_less_nmultiset*:
fixes $X\ Y\ Z :: 'a\ nmultiset$
shows $X < Y \Longrightarrow Y < Z \Longrightarrow X < Z$
proof (*induct Max* $\{|X|, |Y|, |Z|\}$ *arbitrary*: $X\ Y\ Z$
rule: *less_induct*)
case *less*
from *less*(*2*,*3*) **show** *?case*
proof (*cases* X; *cases* Y; *cases* Z)
fix $M\ N\ N' :: 'a\ nmultiset\ multiset$
define A **where** $A = set\_mset\ M \cup set\_mset\ N \cup set\_mset\ N'$
assume *XYZ*: $X = MSet\ M\ Y = MSet\ N\ Z = MSet\ N'$
then have *trans*: $\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ x < y \longrightarrow y < z \longrightarrow x < z$
by (*auto elim*!: *less*(*1*)[*rotated* -1] *dest*!: *depth_nmultiset_MSet simp add*: *A_def*)
have $set\_mset\ M \subseteq A\ set\_mset\ N \subseteq A\ set\_mset\ N' \subseteq A$
unfolding *A_def* **by** *auto*
with *less*(*2*,*3*) *XYZ* **show** $X < Z$
by (*auto simp*: *less_multiset_ext*$_{DM}$*_iff_mult*[*OF _ _ trans*] *mult_def*)
qed (*auto elim*: *less_trans*)
qed

lemma *irrefl_less_nmultiset*:
fixes $X :: 'a\ nmultiset$
shows $X < X \Longrightarrow False$
proof (*induct X*)
case (*MSet M*)
from *MSet*(*2*) **show** *?case*
unfolding *less_nmultiset.simps less_multiset_ext*$_{DM}$*_def*
proof *safe*
fix $X\ Y :: 'a\ nmultiset\ multiset$
define XY **where** $XY = \{(x, y).\ x \in\# X \wedge y \in\# Y \wedge y < x\}$
then have *fin*: *finite XY* **and** *trans*: *trans XY*
by (*auto simp*: *trans_def intro*: *trans_less_nmultiset*
finite_subset[*OF _ finite_cartesian_product*])
assume $X \neq \{\#\}\ X \subseteq\# M\ M = M - X + Y$
then have $X = Y$
by (*auto simp*: *mset_subset_eq_exists_conv*)
with *MSet*(*1*) ‹$X \subseteq\# M$› **have** *irrefl XY*
unfolding *XY_def* **by** (*force dest*: *mset_subset_eqD simp*: *irrefl_def*)
with *trans* **have** *acyclic XY*
by (*simp add*: *acyclic_irrefl*)
moreover
assume $\forall k.\ k \in\# Y \longrightarrow (\exists a.\ a \in\# X \wedge k < a)$
with ‹$X = Y$› ‹$X \neq \{\#\}$› **have** $\neg$ *acyclic XY*
by (*intro notI*, *elim finite_acyclic_wf*[*OF fin*, *elim_format*])
(*auto dest*!: *spec*[*of _ set_mset Y*] *simp*: *wf_eq_minimal XY_def*)
ultimately show *False* **by** *blast*
qed
qed *simp*

lemma *antisym_less_nmultiset*:
fixes $X\ Y :: 'a\ nmultiset$
shows $X < Y \Longrightarrow Y < X \Longrightarrow False$
using *trans_less_nmultiset irrefl_less_nmultiset* **by** *blast*

definition *less_eq_nmultiset* :: $'a\ nmultiset \Rightarrow 'a\ nmultiset \Rightarrow bool$ **where**
less_eq_nmultiset X Y = $(X < Y \vee X = Y)$

instance
proof (*intro_classes*, *goal_cases less_def refl trans*)

```
  case (less_def x y)
  then show ?case
    unfolding less_eq_nmultiset_def by (metis irrefl_less_nmultiset antisym_less_nmultiset)
next
  case (refl x)
  then show ?case
    unfolding less_eq_nmultiset_def by blast
next
  case (trans x y z)
  then show ?case
    unfolding less_eq_nmultiset_def by (metis trans_less_nmultiset)
qed

lemma less_multiset_ext_DM_less: less_multiset_ext_DM op <= (op <)
  unfolding fun_eq_iff less_multiset_ext_DM_def less_multiset_DM by blast

end

instantiation nmultiset :: (order) order
begin

instance
proof (intro_classes, goal_cases antisym)
  case (antisym x y)
  then show ?case
    unfolding less_eq_nmultiset_def by (metis trans_less_nmultiset irrefl_less_nmultiset)
qed

end

instantiation nmultiset :: (linorder) linorder
begin

lemma total_less_nmultiset:
  fixes X Y :: 'a nmultiset
  shows ¬ X < Y ⟹ Y ≠ X ⟹ Y < X
proof (induct X Y rule: less_nmultiset_induct)
  case (MSet_MSet M N)
  then show ?case
    unfolding nmultiset.inject less_nmultiset.simps less_multiset_ext_DM_less less_multiset_HO
    by (metis add_diff_cancel_left' count_inI diff_add_zero in_diff_count less_imp_not_less
      mset_subset_eq_multiset_union_diff_commute subset_mset.order.refl)
qed auto

instance
proof (intro_classes, goal_cases total)
  case (total x y)
  then show ?case
    unfolding less_eq_nmultiset_def by (metis total_less_nmultiset)
qed

end

lemma less_depth_nmultiset_imp_less_nmultiset:
  |X| < |Y| ⟹ X < Y
proof (induct X Y rule: less_nmultiset_induct)
  case (MSet_MSet M N)
  then show ?case
  proof (cases M = {#})
    case False
    with MSet_MSet show ?thesis
      by (auto 0 4 simp: depth_nmultiset.simps(2) less_multiset_ext_DM_def not_le Max_gr_iff
        intro: exI[of _ N] split: if_splits)
```

```
  qed (auto simp: depth_nmultiset.simps(2) less_multiset_ext_DM_less split: if_splits)
qed simp_all

lemma less_nmultiset_imp_le_depth_nmultiset:
  X < Y ⟹ |X| ≤ |Y|
proof (induct X Y rule: less_nmultiset_induct)
  case (MSet_MSet M N)
  then have M < N by (simp add: less_multiset_ext_DM_less)
  then show ?case
  proof (cases M = {#} N = {#} rule: bool.exhaust[case_product bool.exhaust])
    case [simp]: False_False
    show ?thesis
    unfolding depth_nmultiset.simps(2) Let_def False_False Suc_le_mono set_image_mset image_is_empty
      set_mset_eq_empty_iff if_False
    proof (intro iffD2[OF Max_le_iff] ballI iffD2[OF Max_ge_iff]; (elim imageE)?; simp)
      fix X
      assume [simp]: X ∈# M
      with MSet_MSet(1)[of N M X, simplified] ‹M < N› show ∃ Y∈#N. |X| ≤ |Y|
        by (meson ex_gt_imp_less_multiset less_asym' less_depth_nmultiset_imp_less_nmultiset
          not_le_imp_less)
    qed
  qed (auto simp: depth_nmultiset.simps(2))
qed simp_all

lemma eq_mlex_I:
  fixes f :: 'a ⇒ nat and R :: 'a ⇒ 'a ⇒ bool
  assumes ⋀X Y. f X < f Y ⟹ R X Y and antisymP R
  shows {(X, Y). R X Y} = f <*mlex*> {(X, Y). f X = f Y ∧ R X Y}
proof safe
  fix X Y
  assume R X Y
  show (X, Y) ∈ f <*mlex*> {(X, Y). f X = f Y ∧ R X Y}
  proof (cases f X f Y rule: linorder_cases)
    case less
    with ‹R X Y› show ?thesis
      by (elim mlex_less)
  next
    case equal
    with ‹R X Y› show ?thesis
      by (intro mlex_leq) auto
  next
    case greater
    from ‹R X Y› assms(1)[OF greater] ‹antisymP R› greater show ?thesis
      unfolding antisym_def by auto
  qed
next
  fix X Y
  assume (X, Y) ∈ f <*mlex*> {(X, Y). f X = f Y ∧ R X Y}
  then show R X Y
    unfolding mlex_prod_def by (auto simp: assms(1))
qed

instantiation nmultiset :: (wellorder) wellorder
begin

lemma depth_nmultiset_eq_0[simp]: |X| = 0 ⟷ (X = MSet {#} ∨ (∃ x. X = Elem x))
  by (cases X; simp add: depth_nmultiset.simps(2))

lemma depth_nmultiset_eq_Suc[simp]: |X| = Suc n ⟷
  (∃ N. X = MSet N ∧ (∃ Y ∈# N. |Y| = n) ∧ (∀ Y ∈# N. |Y| ≤ n))
  by (cases X; auto simp add: depth_nmultiset.simps(2) intro!: Max_eqI)
    (metis (no_types, lifting) Max_in finite_imageI finite_set_mset imageE image_is_empty
      set_mset_eq_empty_iff)
```

```
lemma wf_less_nmultiset_depth:
  wf {(X :: 'a nmultiset, Y). |X| = i ∧ |Y| = i ∧ X < Y}
proof (induct i rule: less_induct)
  case (less i)
  define A :: 'a nmultiset set where A = {X. |X| < i}
  from less have wf ((depth_nmultiset :: 'a nmultiset ⇒ nat) <*mlex*>
      (⋃j < i. {(X, Y). |X| = j ∧ |Y| = j ∧ X < Y}))
    by (intro wf_UN wf_mlex) auto
  then have *: wf (mult {(X :: 'a nmultiset, Y). X ∈ A ∧ Y ∈ A ∧ X < Y})
    by (intro wf_mult, elim wf_subset) (force simp: A_def mlex_prod_def not_less_iff_gr_or_eq
      dest!: less_depth_nmultiset_imp_less_nmultiset)
  show ?case
  proof (cases i)
    case 0
    then show ?thesis
      by (auto simp: inj_on_def intro!: wf_subset[OF
        wf_Un[OF wf_map_prod_image[OF wf, of Elem] wf_UN[of Elem ' UNIV λx. {(x, MSet {#})}]]])
  next
    case (Suc n)
    then show ?thesis
      by (intro wf_subset[OF wf_map_prod_image[OF *, of MSet]])
        (auto 0 4 simp: map_prod_def image_iff inj_on_def A_def
          dest!: less_multiset_ext_DM_imp_mult[of _ A, rotated -1] split: prod.splits)
  qed
qed

lemma wf_less_nmultiset: wf {(X :: 'a nmultiset, Y :: 'a nmultiset). X < Y} (is wf ?R)
proof -
  have ?R = depth_nmultiset <*mlex*> {(X, Y). |X| = |Y| ∧ X < Y}
    by (rule eq_mlex_I) (auto simp: antisym_def less_depth_nmultiset_imp_less_nmultiset)
  also have {(X, Y). |X| = |Y| ∧ X < Y} = (⋃i. {(X, Y). |X| = i ∧ |Y| = i ∧ X < Y})
    by auto
  finally show ?thesis
    by (fastforce intro: wf_mlex wf_Union wf_less_nmultiset_depth)
qed

instance using wf_less_nmultiset unfolding wf_def mem_Collect_eq prod.case by intro_classes metis

end

end
```

5 Hereditar(il)y (Finite) Multisets

```
theory Hereditary_Multiset
imports Multiset_More Nested_Multiset
begin
```

5.1 Type Definition

```
datatype hmultiset =
  HMSet (hmsetmset: hmultiset multiset)

lemma hmsetmset_inject[simp]: hmsetmset A = hmsetmset B ⟷ A = B
  by (blast intro: hmultiset.expand)

primrec Rep_hmultiset :: hmultiset ⇒ unit nmultiset where
  Rep_hmultiset (HMSet M) = MSet (image_mset Rep_hmultiset M)

primrec (nonexhaustive) Abs_hmultiset :: unit nmultiset ⇒ hmultiset where
  Abs_hmultiset (MSet M) = HMSet (image_mset Abs_hmultiset M)
```

lemma *type_definition_hmultiset*:
  type_definition Rep_hmultiset Abs_hmultiset {*X*. *no_elem X*}
proof (*unfold_locales*, *unfold mem_Collect_eq*)
  fix *X*
  show *no_elem* (*Rep_hmultiset X*)
    by (*induct X*) (*auto intro*!: *no_elem.intros*)
  show *Abs_hmultiset* (*Rep_hmultiset X*) = *X*
    by (*induct X*) *auto*
next
  fix *Y* :: *unit nmultiset*
  assume *no_elem Y*
  thus *Rep_hmultiset* (*Abs_hmultiset Y*) = *Y*
    by (*induct Y rule*: *no_elem.induct*) *auto*
qed

setup-lifting *type_definition_hmultiset*

lemma *HMSet_alt*: *HMSet* = *Abs_hmultiset o MSet o image_mset Rep_hmultiset*
  by (*auto simp*: *type_definition.Rep_inverse*[*OF type_definition_hmultiset*])

lemma *HMSet_transfer*[*transfer_rule*]: *rel_fun* (*rel_mset pcr_hmultiset*) *pcr_hmultiset MSet HMSet*
  unfolding *HMSet_alt* **by** (*force simp*: *rel_fun_def multiset.in_rel nmultiset.rel_eq*
    pcr_hmultiset_def cr_hmultiset_def
    type_definition.Rep_inverse[*OF type_definition_hmultiset*] *intro*!: *multiset.map_cong*)

5.2 Restriction of Dershowitz and Manna's Nested Multiset Order

instantiation *hmultiset* :: *linorder*
begin

lift-definition *less_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** *op* < .
lift-definition *less_eq_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** *op* ≤ .

instance
  by (*intro_classes*; *transfer*) *auto*

end

lemma *less_HMSet_iff_less_multiset_ext*$_{DM}$: *HMSet M* < *HMSet N* ⟷ *less_multiset_ext*$_{DM}$ (*op* <) *M N*
  unfolding *less_multiset_ext*$_{DM}$*_def*
proof (*transfer*, *unfold less_nmultiset.simps less_multiset_ext*$_{DM}$*_def*, *safe*)
  fix *M N* :: *unit nmultiset multiset* **and** *X Y*
  assume ∗: *pred_mset no_elem* (*N* − *X* + *Y*) *pred_mset no_elem N X* ≠ {#}
    X ⊆# *N* ∀*k*. *k* ∈# *Y* ⟶ (∃*a*. *a* ∈# *X* ∧ *k* < *a*)
  then have *X* ∈ *Collect* (*pred_mset no_elem*)
    unfolding *multiset.pred_set mem_Collect_eq* **by** (*metis rev_subsetD set_mset_mono*)
  from ∗(*1*) **have** *Y* ∈ *Collect* (*pred_mset no_elem*)
    unfolding *multiset.pred_set mem_Collect_eq* **by** (*metis add_diff_cancel_left*′ *in_diffD*)
  show
    ∃*X*′∈*Collect* (*pred_mset no_elem*). ∃*Y*′∈*Collect* (*pred_mset no_elem*).
      X′ ≠ {#} ∧ *filter_mset no_elem X*′ ⊆# *filter_mset no_elem N* ∧ *N* − *X* + *Y* = *N* − *X*′ + *Y*′ ∧
      (∀*k*∈*Collect no_elem*. *k* ∈# *Y*′ ⟶ (∃*a*∈*Collect no_elem*. *a* ∈# *X*′ ∧ *k* < *a*))
    by (*rule bexI*[*OF* _ ‹*X* ∈ *Collect* (*pred_mset no_elem*)›],
      rule bexI[*OF* _ ‹*Y* ∈ *Collect* (*pred_mset no_elem*)›])
      (*insert* ∗; *force simp*: *set_mset_diff multiset.pred_set multiset_filter_mono*)
next
  fix *M N* :: *unit nmultiset multiset* **and** *X Y*
  assume ∗:
    pred_mset no_elem (*N* − *X* + *Y*) *pred_mset no_elem N pred_mset no_elem X pred_mset no_elem Y*
    X ≠ {#} *filter_mset no_elem X* ⊆# *filter_mset no_elem N*
    ∀*k*∈*Collect no_elem*. *k* ∈# *Y* ⟶ (∃*a*∈*Collect no_elem*. *a* ∈# *X* ∧ *k* < *a*)
  then have [*simp*]: *filter_mset no_elem X* = *X filter_mset no_elem N* = *N*
    unfolding *filter_mset_eq_conv* **by** (*auto simp*: *multiset.pred_set*)
  show

$\exists X'\ Y'.\ X' \neq \{\#\} \wedge X' \subseteq\# N \wedge\ N - X + Y = N - X' + Y' \wedge$
$(\forall k.\ k \in\# Y' \longrightarrow (\exists a.\ a \in\# X' \wedge k < a))$
by (*rule exI*[*of* _ *X*], *rule exI*[*of* _ *Y*]) (*insert* ∗; *auto simp*: *multiset.pred_set*)
qed

lemma *hmsetmset_less*[*simp*]: *hmsetmset M* < *hmsetmset N* ⟷ *M* < *N*
by (*cases M*, *cases N*, *simp add*: *less_multiset_ext*$_{DM}$ *less_HMSet_iff_less_multiset_ext*$_{DM}$)

lemma *hmsetmset_le*[*simp*]: *hmsetmset M* ≤ *hmsetmset N* ⟷ *M* ≤ *N*
unfolding *le_less hmsetmset_less* **by** (*metis hmultiset.collapse*)

lemma *wf_less_hmultiset*: *wf* {(*X* :: *hmultiset*, *Y* :: *hmultiset*). *X* < *Y*}
unfolding *wf_eq_minimal* **by** *transfer* (*insert wf_less_nmultiset*[*unfolded wf_eq_minimal*], *fast*)

instance *hmultiset* :: *wellorder*
using *wf_less_hmultiset* **unfolding** *wf_def mem_Collect_eq prod.case* **by** *intro_classes metis*

lemma *HMSet_less*[*simp*]: *HMSet M* < *HMSet N* ⟷ *M* < *N*
by (*simp add*: *less_HMSet_iff_less_multiset_ext*$_{DM}$ *less_multiset_ext*$_{DM}$*_less*)

lemma *HMSet_le*[*simp*]: *HMSet M* ≤ *HMSet N* ⟷ *M* ≤ *N*
by (*simp add*: *hmsetmset_le*[*symmetric*])

inductive-set *hmultiset_sub* **where**
X ∈# *M* ⟹ (*X*, *HMSet M*) ∈ *hmultiset_sub*

lemma *wf_hmultiset_sub*[*simp*]: *wf hmultiset_sub*
proof (*rule wfUNIVI*)
fix *P* **and** *M* :: *hmultiset*
assume *ih*: ∀ *M*. (∀ *N*. (*N*, *M*) ∈ *hmultiset_sub* ⟶ *P N*) ⟶ *P M*
show *P M*
by (*induct M*; *rule ih*[*rule_format*]) (*auto simp*: *hmultiset_sub.simps*)
qed

5.3 Disjoint Union and Truncated Difference

instantiation *hmultiset* :: *cancel_comm_monoid_add*
begin

definition *zero_hmultiset* :: *hmultiset* **where**
0 = *HMSet* {#}

lemma *hmsetmset_empty_iff*[*simp*]: *hmsetmset n* = {#} ⟷ *n* = *0*
unfolding *zero_hmultiset_def* **by** (*cases n*) *simp*

lemma
HMSet_eq_0_iff[*simp*]: *HMSet m* = *0* ⟷ *m* = {#} **and**
zero_eq_HMSet[*simp*]: *0* = *HMSet m* ⟷ *m* = {#}
by (*cases m*) (*auto simp*: *zero_hmultiset_def*)

definition *plus_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *hmultiset* **where**
A + *B* = *HMSet* (*hmsetmset A* + *hmsetmset B*)

definition *minus_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *hmultiset* **where**
A − *B* = *HMSet* (*hmsetmset A* − *hmsetmset B*)

instance
by *intro_classes* (*auto simp*: *zero_hmultiset_def plus_hmultiset_def minus_hmultiset_def*)

end

lemma *HMSet_plus*: *HMSet* (*A* + *B*) = *HMSet A* + *HMSet B*
by (*simp add*: *plus_hmultiset_def*)

```
lemma HMSet_diff: HMSet (A - B) = HMSet A - HMSet B
  by (simp add: minus_hmultiset_def)

lemma hmsetmset_plus: hmsetmset (M + N) = hmsetmset M + hmsetmset N
  by (simp add: plus_hmultiset_def)

lemma hmsetmset_diff: hmsetmset (M - N) = hmsetmset M - hmsetmset N
  by (simp add: minus_hmultiset_def)

lemma diff_diff_add_hmset[simp]: a - b - c = a - (b + c) for a b c :: hmultiset
  by (fact diff_diff_add)

instance hmultiset :: comm_monoid_diff
  by intro_classes (auto simp: zero_hmultiset_def minus_hmultiset_def)

instantiation hmultiset :: order_bot
begin

definition bot_hmultiset :: hmultiset where
  bot_hmultiset = 0

instance
proof (intro_classes, unfold bot_hmultiset_def zero_hmultiset_def, transfer, goal_cases bot_least)
  case (bot_least x)
  thus ?case
    by (induct x rule: no_elem.induct) (auto simp: less_eq_nmultiset_def less_multiset_ext_DM_less)
qed

end

instance hmultiset :: no_top
proof (intro_classes, goal_cases gt_ex)
  case (gt_ex a)
  have a < a + HMSet {#HMSet {#}#}
    by (simp add: plus_hmultiset_def hmsetmset_less[symmetric] less_multiset_ext_DM_def)
  thus ?case
    by (rule exI)
qed

instance hmultiset :: ordered_cancel_comm_monoid_add
  by intro_classes (simp del: hmsetmset_less add: plus_hmultiset_def order_le_less
    hmsetmset_less[symmetric] less_multiset_ext_DM_less)

instance hmultiset :: ordered_ab_semigroup_add_imp_le
  by intro_classes (simp add: plus_hmultiset_def order_le_less less_multiset_ext_DM_less)

lemma le_minus_plus_same_hmset: m ≤ m - n + n for m n :: hmultiset
proof (cases m n rule: hmultiset.exhaust[case_product hmultiset.exhaust])
  case (HMSet_HMSet m0 n0)
  note m = this(1) and n = this(2)

  {
    assume n0 ⊆# m0
    hence m0 = m0 - n0 + n0
      by simp
  }
  moreover
  {
    assume ¬ n0 ⊆# m0
    hence m0 ⊂# m0 - n0 + n0
      by (metis mset_subset_eq_add_right subset_eq_diff_conv subset_mset.dual_order.refl
        subset_mset_def)
    hence m0 < m0 - n0 + n0
```

by (*rule subset_ imp_ less_ multiset*)
}
ultimately show *?thesis*
by (*simp* (*no_ asm*) *add*: *m n order_ le_ less plus_ hmultiset_ def minus_ hmultiset_ def*) *blast*
qed

5.4 Infimum and Supremum

instantiation *hmultiset* :: *distrib_ lattice*
begin

definition *inf_ hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *hmultiset* **where**
inf_ hmultiset A B = (*if A* < *B then A else B*)

definition *sup_ hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *hmultiset* **where**
sup_ hmultiset A B = (*if B* > *A then B else A*)

instance
by *intro_ classes* (*auto simp*: *inf_ hmultiset_ def sup_ hmultiset_ def*)

end

end

6 Signed Hereditar(il)y (Finite) Multisets

theory *Signed_ Hereditary_ Multiset*
imports *Signed_ Multiset Hereditary_ Multiset*
begin

6.1 Type Definition

typedef *zhmultiset* = *UNIV* :: *hmultiset zmultiset set*
morphisms *zhmsetmset ZHMSet*
by *simp*

lemmas *ZHMSet_ inverse*[*simp*] = *ZHMSet_ inverse*[*OF UNIV_ I*]
lemmas *ZHMSet_ inject*[*simp*] = *ZHMSet_ inject*[*OF UNIV_ I UNIV_ I*]

declare
zhmsetmset_ inverse [*simp*]
zhmsetmset_ inject [*simp*]

setup-lifting *type_ definition_ zhmultiset*

6.2 Multiset Order

instantiation *zhmultiset* :: *linorder*
begin

lift-definition *less_ zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *bool* **is** *op* < .
lift-definition *less_ eq_ zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *bool* **is** *op* ≤ .

instance
by (*intro_ classes*; *transfer*) *auto*

end

lemmas *ZHMSet_ less*[*simp*] = *less_ zhmultiset.abs_ eq*
lemmas *ZHMSet_ le*[*simp*] = *less_ eq_ zhmultiset.abs_ eq*
lemmas *zhmsetmset_ less*[*simp*] = *less_ zhmultiset.rep_ eq*[*symmetric*]
lemmas *zhmsetmset_ le*[*simp*] = *less_ eq_ zhmultiset.rep_ eq*[*symmetric*]

6.3 Embedding and Projections of Syntactic Ordinals

abbreviation *zhmset_of* :: *hmultiset* ⇒ *zhmultiset* **where**
zhmset_of M ≡ *ZHMSet* (*zmset_of* (*hmsetmset M*))

lemma *zhmset_of_inject*[*simp*]: *zhmset_of M* = *zhmset_of N* ⟷ *M* = *N*
by *simp*

lemma *zhmset_of_less*: *zhmset_of M* < *zhmset_of N* ⟷ *M* < *N*
by (*simp add*: *zmset_of_less*)

lemma *zhmset_of_le*: *zhmset_of M* ≤ *zhmset_of N* ⟷ *M* ≤ *N*
by (*simp add*: *zmset_of_le*)

abbreviation *hmset_pos* :: *zhmultiset* ⇒ *hmultiset* **where**
hmset_pos M ≡ *HMSet* (*mset_pos* (*zhmsetmset M*))

abbreviation *hmset_neg* :: *zhmultiset* ⇒ *hmultiset* **where**
hmset_neg M ≡ *HMSet* (*mset_neg* (*zhmsetmset M*))

6.4 Disjoint Union and Difference

instantiation *zhmultiset* :: *cancel_comm_monoid_add*
begin

lift-definition *zero_zhmultiset* :: *zhmultiset* **is** $\{\#\}_z$ **.**

lift-definition *plus_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **is**
λ*A B*. *A* + *B* **.**

lift-definition *minus_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **is**
λ*A B*. *A* − *B* **.**

lemmas *ZHMSet_plus* = *plus_zhmultiset.abs_eq*[*symmetric*]
lemmas *ZHMSet_diff* = *minus_zhmultiset.abs_eq*[*symmetric*]
lemmas *zhmsetmset_plus* = *plus_zhmultiset.rep_eq*
lemmas *zhmsetmset_diff* = *minus_zhmultiset.rep_eq*

lemma *zhmset_of_plus*: *zhmset_of* (*A* + *B*) = *zhmset_of A* + *zhmset_of B*
by (*simp add*: *hmsetmset_plus ZHMSet_plus zmset_of_plus*)

lemma *hmsetmset_0*[*simp*]: *hmsetmset 0* = {#}
by (*rule hmultiset.inject*[*THEN iffD1*]) (*simp add*: *zero_hmultiset_def*)

instance
by (*intro_classes*; *transfer*) (*auto intro*: *linordered_field_class.sign_simps*(*1*) *add.commute*)

end

lemma *zhmset_of_0*: *zhmset_of 0* = *0*
by (*simp add*: *zero_zhmultiset_def*)

lemma *hmset_pos_plus*:
hmset_pos (*A* + *B*) = (*hmset_pos A* − *hmset_neg B*) + (*hmset_pos B* − *hmset_neg A*)
by (*simp add*: *HMSet_diff HMSet_plus zhmsetmset_plus*)

lemma *hmset_neg_plus*:
hmset_neg (*A* + *B*) = (*hmset_neg A* − *hmset_pos B*) + (*hmset_neg B* − *hmset_pos A*)
by (*simp add*: *HMSet_diff HMSet_plus zhmsetmset_plus*)

lemma *zhmset_pos_neg_partition*: *M* = *zhmset_of* (*hmset_pos M*) − *zhmset_of* (*hmset_neg M*)
by (*cases M*, *simp add*: *ZHMSet_diff*[*symmetric*], *rule mset_pos_neg_partition*)

lemma *zhmset_pos_as_neg*: *zhmset_of* (*hmset_pos M*) = *zhmset_of* (*hmset_neg M*) + *M*

using *mset_pos_as_neg zhmsetmset_plus zhmsetmset_inject* **by** *fastforce*

lemma *zhmset_neg_as_pos*: *zhmset_of* (*hmset_neg M*) = *zhmset_of* (*hmset_pos M*) − *M*
using *zhmsetmset_diff mset_neg_as_pos zhmsetmset_inject* **by** *fastforce*

lemma *hmset_pos_neg_dual*:
hmset_pos a + *hmset_pos b* + (*hmset_neg a* − *hmset_pos b*) + (*hmset_neg b* − *hmset_pos a*) =
hmset_neg a + *hmset_neg b* + (*hmset_pos a* − *hmset_neg b*) + (*hmset_pos b* − *hmset_neg a*)
by (*simp add*: *HMSet_plus*[*symmetric*] *HMSet_diff*[*symmetric*]) (*rule mset_pos_neg_dual*)

lemma *zhmset_of_sum_list*: *zhmset_of* (*sum_list Ms*) = *sum_list* (*map zhmset_of Ms*)
by (*induct Ms*) (*auto simp*: *zero_zhmultiset_def zhmset_of_plus*)

lemma *less_hmset_zhmsetE*:
assumes *m_lt_n*: *M* < *N*
obtains *A B C* **where** *M* = *zhmset_of A* + *C* **and** *N* = *zhmset_of B* + *C* **and** *A* < *B*
by (*rule less_mset_zmsetE*[*OF m_lt_n*[*folded zhmsetmset_less*]])
(*metis hmsetmset_less hmultiset.sel ZHMSet_plus zhmsetmset_inverse*)

lemma *less_eq_hmset_zhmsetE*:
assumes *m_le_n*: *M* ≤ *N*
obtains *A B C* **where** *M* = *zhmset_of A* + *C* **and** *N* = *zhmset_of B* + *C* **and** *A* ≤ *B*
by (*rule less_eq_mset_zmsetE*[*OF m_le_n*[*folded zhmsetmset_le*]])
(*metis hmsetmset_le hmultiset.sel ZHMSet_plus zhmsetmset_inverse*)

instantiation *zhmultiset* :: *ab_group_add*
begin

lift-definition *uminus_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* **is** λ*A*. − *A* **.**

lemmas *ZHMSet_uminus* = *uminus_zhmultiset.abs_eq*[*symmetric*]
lemmas *zhmsetmset_uminus* = *uminus_zhmultiset.rep_eq*

instance
by (*intro_classes*; *transfer*; *simp*)

end

6.5 Infimum and Supremum

instance *zhmultiset* :: *ordered_cancel_comm_monoid_add*
by (*intro_classes*; *transfer*) (*auto simp*: *add_left_mono*)

instance *zhmultiset* :: *ordered_ab_group_add*
by (*intro_classes*; *transfer*; *simp*)

instantiation *zhmultiset* :: *distrib_lattice*
begin

definition *inf_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **where**
inf_zhmultiset A B = (*if A* < *B then A else B*)

definition *sup_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **where**
sup_zhmultiset A B = (*if B* > *A then B else A*)

instance
by *intro_classes* (*auto simp*: *inf_zhmultiset_def sup_zhmultiset_def*)

end

end

7 Syntactic Ordinals in Cantor Normal Form

theory *Syntactic_ Ordinal*
imports *Hereditary_ Multiset ~~/src/HOL/Library/Product_ Order ~~/src/HOL/Library/Extended_ Nat*
begin

7.1 Natural (Hessenberg) Product

instantiation *hmultiset* :: *comm_ semiring_ 1*
begin

definition *one_ hmultiset* :: *hmultiset* **where**
1 = HMSet {#0#}

definition *times_ hmultiset* :: *hmultiset ⇒ hmultiset ⇒ hmultiset* **where**
*A * B = HMSet (image_ mset (case_ prod (op +)) (hmsetmset A ×mset hmsetmset B))*

lemma *hmsetmset_ times*:
*hmsetmset (m * n) = image_ mset (case_ prod (op +)) (hmsetmset m ×mset hmsetmset n)*
unfolding *times_ hmultiset_ def* **by** *simp*

instance
proof (*intro_ classes*, *goal_ cases assoc comm one distrib_ plus zeroL zeroR zero_ one*)
case (*assoc a b c*)
thus *?case*
by (*auto simp*: *times_ hmultiset_ def Times_ mset_ image_ mset1 Times_ mset_ image_ mset2*
Times_ mset_ assoc ac_ simps intro: *multiset.map_ cong*)
next
case (*comm a b*)
thus *?case*
unfolding *times_ hmultiset_ def*
by (*subst product_ swap_ mset*[*symmetric*]) (*auto simp*: *ac_ simps intro*: *multiset.map_ cong*)
next
case (*one a*)
thus *?case*
by (*auto simp*: *one_ hmultiset_ def times_ hmultiset_ def Times_ mset_ single_ left*)
next
case (*distrib_ plus a b c*)
thus *?case*
by (*auto simp*: *plus_ hmultiset_ def times_ hmultiset_ def*)
next
case (*zeroL a*)
thus *?case*
by (*auto simp*: *times_ hmultiset_ def*)
next
case (*zeroR a*)
thus *?case*
by (*auto simp*: *times_ hmultiset_ def*)
next
case *zero_ one*
thus *?case*
by (*auto simp*: *one_ hmultiset_ def*)
qed

end

lemma *empty_ times_ left_ hmset*[*simp*]: *HMSet {#} * M = 0*
by (*simp add*: *times_ hmultiset_ def*)

lemma *empty_ times_ right_ hmset*[*simp*]: *M * HMSet {#} = 0*
by (*metis mult_ zero_ right zero_ hmultiset_ def*)

lemma *singleton_ times_ left_ hmset*[*simp*]:
*HMSet {#M#} * N = HMSet (image_ mset ((op +) M) (hmsetmset N))*

by (*simp add*: *times_hmultiset_def Times_mset_single_left*)

lemma *singleton_times_right_hmset*[*simp*]:
N ∗ *HMSet* {#*M*#} = *HMSet* (*image_mset* ((*op* +) *M*) (*hmsetmset N*))
by (*metis mult.commute singleton_times_left_hmset*)

7.2 Inequalities

definition *plus_nmultiset* :: *unit nmultiset* ⇒ *unit nmultiset* ⇒ *unit nmultiset* **where**
plus_nmultiset X Y = *Rep_hmultiset* (*Abs_hmultiset X* + *Abs_hmultiset Y*)

lemma *plus_nmultiset_mono*:
assumes *less*: (*X*, *Y*) < (*X*′, *Y*′) **and** *no_elem*: *no_elem X no_elem Y no_elem X*′ *no_elem Y*′
shows *plus_nmultiset X Y* < *plus_nmultiset X*′ *Y*′
using *less*[*unfolded less_le_not_le*] *no_elem*
by (*auto simp*: *plus_nmultiset_def plus_hmultiset_def less_multiset_ext*$_{DM}$*_less less_eq_nmultiset_def*
union_less_mono type_definition.Abs_inverse[*OF type_definition_hmultiset*, *simplified*]
elim!: *no_elem.cases*)

lemma *plus_hmultiset_transfer*[*transfer_rule*]:
(*rel_fun pcr_hmultiset* (*rel_fun pcr_hmultiset pcr_hmultiset*)) *plus_nmultiset op* +
unfolding *rel_fun_def plus_nmultiset_def pcr_hmultiset_def nmultiset.rel_eq eq_OO cr_hmultiset_def*
by (*auto simp*: *type_definition.Rep_inverse*[*OF type_definition_hmultiset*])

lemma *Times_mset_monoL*:
assumes *less*: *M* < *N* **and** *Z_nemp*: *Z* ≠ {#}
shows *M* ×*mset Z* < *N* ×*mset Z*
proof −
obtain *Y X* **where**
Y_nemp: *Y* ≠ {#} **and** *Y_sub_N*: *Y* ⊆# *N* **and** *M_eq*: *M* = *N* − *Y* + *X* **and**
ex_Y: ∀*x*. *x* ∈# *X* ⟶ (∃*y*. *y* ∈# *Y* ∧ *x* < *y*)
using *less*[*unfolded less_multiset*$_{DM}$] **by** *blast*

let *?X* = *X* ×*mset Z*
let *?Y* = *Y* ×*mset Z*

show *?thesis*
unfolding *less_multiset*$_{DM}$
proof (*intro exI conjI*)
show *M* ×*mset Z* = *N* ×*mset Z* − *?Y* + *?X*
unfolding *M_eq* **by** (*auto simp*: *Sigma_mset_Diff_distrib1*)
next
obtain *y* **where** *y*: ∀*x*. *x* ∈# *X* ⟶ *y x* ∈# *Y* ∧ *x* < *y x*
using *ex_Y* **by** *moura*

show ∀*x*. *x* ∈# *?X* ⟶ (∃*y*. *y* ∈# *?Y* ∧ *x* < *y*)
proof (*intro allI impI*)
fix *x*
assume *x* ∈# *?X*
thus ∃*y*. *y* ∈# *?Y* ∧ *x* < *y*
using *y* **by** (*intro exI*[*of* _ (*y* (*fst x*), *snd x*)]) (*auto simp*: *less_le_not_le*)
qed
qed (*auto simp*: *Z_nemp Y_nemp Y_sub_N Sigma_mset_mono*)
qed

lemma *times_hmultiset_monoL*:
fixes *a b c* :: *hmultiset*
shows *a* < *b* ⟹ 0 < *c* ⟹ *a* ∗ *c* < *b* ∗ *c*
by (*cases a*, *cases b*, *cases c*, *hypsubst_thin*,
unfold times_hmultiset_def zero_hmultiset_def hmultiset.sel, *transfer*,
auto simp: *less_multiset_ext*$_{DM}$*_less multiset.pred_set intro*!: *image_mset_mono Times_mset_monoL*
elim!: *plus_nmultiset_mono*)

instance *hmultiset* :: *linordered_semiring_strict*

by *intro_classes* (*subst* (*1 2*) *mult.commute*, *rule times_hmultiset_monoL*)

lemma *zero_le_hmset*[*simp*]: $0 \leq M$ **for** M :: *hmultiset*
by (*simp add*: *order_le_less*) (*metis hmsetmset_less le_multiset_empty_left hmsetmset_empty_iff*)

lemma *mult_le_mono1_hmset*: $i \leq j \Longrightarrow i * k \leq j * k$ **for** $i\ j\ k$ :: *hmultiset*
by (*simp add*: *mult_right_mono*)

lemma *mult_le_mono2_hmset*: $i \leq j \Longrightarrow k * i \leq k * j$ **for** $i\ j\ k$ :: *hmultiset*
by (*simp add*: *mult_left_mono*)

lemma *mult_le_mono_hmset*: $i \leq j \Longrightarrow k \leq l \Longrightarrow i * k \leq j * l$ **for** $i\ j\ k\ l$ :: *hmultiset*
by (*simp add*: *mult_mono*)

lemma
le_add1_hmset: $n \leq n + m$ **and**
le_add2_hmset: $n \leq m + n$ **for** n :: *hmultiset*
by *simp+*

lemma *le_zero_eq_hmset*[*simp*]: $M \leq 0 \longleftrightarrow M = 0$ **for** M :: *hmultiset*
proof (*cases M*)
case (*HMSet x*)
thus *?thesis*
by (*cases x*) (*auto simp*: *zero_hmultiset_def order_le_less less_multiset_ext*$_{DM}$*_less*)
qed

lemma *not_less_zero_hmset*[*simp*]: $\neg\ M < 0$ **for** M :: *hmultiset*
using *not_le zero_le_hmset* **by** *blast*

lemma *not_gr_zero_hmset*[*simp*]: $\neg\ 0 < M \longleftrightarrow M = 0$ **for** M :: *hmultiset*
using *neqE not_less_zero_hmset* **by** *blast*

lemma *zero_less_iff_neq_zero_hmset*: $0 < M \longleftrightarrow M \neq 0$ **for** M :: *hmultiset*
using *not_gr_zero_hmset* **by** *blast*

lemma *gr_zeroI_hmset*: $(M = 0 \Longrightarrow \mathit{False}) \Longrightarrow 0 < M$ **for** M :: *hmultiset*
by (*rule zero_less_iff_neq_zero_hmset*[*THEN iffD2*]) *iprover*

lemma *gr_implies_not_zero_hmset*: $M < N \Longrightarrow N \neq 0$ **for** $M\ N$ :: *hmultiset*
by *auto*

lemma *add_eq_0_iff_both_eq_0_hmset*[*simp*]: $M + N = 0 \longleftrightarrow M = 0 \wedge N = 0$ **for** $M\ N$ :: *hmultiset*
by (*intro add_nonneg_eq_0_iff zero_le_hmset*)

lemma *zero_eq_add_iff_both_eq_0_hmset*[*simp*]: $0 = M + N \longleftrightarrow M = 0 \wedge N = 0$ **for** $M\ N$ :: *hmultiset*
using *add_eq_0_iff_both_eq_0_hmset*[*of M N*] **unfolding** *eq_commute*[*of 0*] **.**

lemma *trans_less_add1_hmset*: $i < j \Longrightarrow i < j + m$ **for** $i\ j\ m$ :: *hmultiset*
by (*metis add_increasing2 leD le_less not_gr_zero_hmset*)

lemma *trans_less_add2_hmset*: $i < j \Longrightarrow i < m + j$ **for** $i\ j\ m$ :: *hmultiset*
by (*metis add_increasing leD le_less not_gr_zero_hmset*)

lemma *trans_le_add1_hmset*: $i \leq j \Longrightarrow i \leq j + m$ **for** $i\ j\ m$ :: *hmultiset*
by (*simp add*: *add_increasing2*)

lemma *trans_le_add2_hmset*: $i \leq j \Longrightarrow i \leq m + j$ **for** $i\ j\ m$ :: *hmultiset*
by (*simp add*: *add_increasing*)

lemma *less_iff_add1_le_hmset*: $m < n \longleftrightarrow m + 1 \leq n$ **for** $m\ n$ :: *hmultiset*
proof (*cases m n rule*: *hmultiset.exhaust*[*case_product hmultiset.exhaust*])
case (*HMSet_HMSet m0 n0*)
note m = *this*(*1*) **and** n = *this*(*2*)

```
  show ?thesis
  proof (simp add: m n one_hmultiset_def plus_hmultiset_def order.order_iff_strict
      less_multiset_ext_DM_less, intro iffI)
    assume m0_lt_n0: m0 < n0
    note
      m0_ne_n0 = m0_lt_n0[unfolded less_multiset_HO, THEN conjunct1] and
      ex_n0_gt_m0 = m0_lt_n0[unfolded less_multiset_HO, THEN conjunct2, rule_format]

    {
      assume zero_m0_gt_n0: add_mset 0 m0 > n0
      note
        n0_ne_0m0 = zero_m0_gt_n0[unfolded less_multiset_HO, THEN conjunct1] and
        ex_0m0_gt_n0 = zero_m0_gt_n0[unfolded less_multiset_HO, THEN conjunct2, rule_format]

      {
        fix y
        assume m0y_lt_n0y: count m0 y < count n0 y

        have ∃ x > y. count n0 x < count m0 x
        proof (cases count (add_mset 0 m0) y < count n0 y)
          case True
          then obtain aa where
            aa_gt_y: aa > y and
            count_n0aa_lt_count_0m0aa: count n0 aa < count (add_mset 0 m0) aa
            using ex_0m0_gt_n0 by blast
          have aa ≠ 0
            by (rule gr_implies_not_zero_hmset[OF aa_gt_y])
          hence count (add_mset 0 m0) aa = count m0 aa
            by simp
          thus ?thesis
            using count_n0aa_lt_count_0m0aa aa_gt_y by auto
        next
          case not_0m0_y_lt_n0y: False
          hence y_eq_0: y = 0
            by (metis count_add_mset m0y_lt_n0y)
          have sm0y_eq_n0y: Suc (count m0 y) = count n0 y
            using m0y_lt_n0y not_0m0_y_lt_n0y count_add_mset[of 0 _ 0] unfolding y_eq_0 by simp

          obtain bb where count n0 bb < count (add_mset 0 m0) bb
            using lt_imp_ex_count_lt[OF zero_m0_gt_n0] by blast
          hence n0bb_lt_m0bb: count n0 bb < count m0 bb
            unfolding count_add_mset by (metis (full_types) less_irrefl_nat sm0y_eq_n0y y_eq_0)
          hence bb ≠ 0
            using sm0y_eq_n0y y_eq_0 by auto
          thus ?thesis
            unfolding y_eq_0 using n0bb_lt_m0bb not_gr_zero_hmset by blast
        qed
      }
      hence n0 < m0
        unfolding less_multiset_HO using m0_ne_n0 by blast
      hence False
        using m0_lt_n0 by simp
    }
    thus add_mset 0 m0 < n0 ∨ add_mset 0 m0 = n0
      using antisym_conv3 by blast
  next
    assume add_mset 0 m0 < n0 ∨ add_mset 0 m0 = n0
    thus m0 < n0
      using dual_order.strict_trans le_multiset_right_total by blast
  qed
qed
```

lemma *zero_less_iff_1_le_hmset*: $0 < n \longleftrightarrow 1 \leq n$ **for** n :: *hmultiset*
by (*rule less_iff_add1_le_hmset*[*of* 0, *simplified*])

lemma *less_add_1_iff_le_hmset*: $m < n + 1 \longleftrightarrow m \leq n$ **for** $m\ n$ :: *hmultiset*
by (*rule less_iff_add1_le_hmset*[*of* $m\ n + 1$, *simplified*])

instance *hmultiset* :: *ordered_cancel_comm_semiring*
by *intro_classes* (*simp add*: *mult_le_mono2_hmset*)

instance *hmultiset* :: *linordered_semiring_1_strict*
by *intro_classes*

instance *hmultiset* :: *bounded_lattice_bot*
by *intro_classes*

instance *hmultiset* :: *zero_less_one*
by *intro_classes* (*simp add*: *zero_less_iff_neq_zero_hmset*)

instance *hmultiset* :: *linordered_nonzero_semiring*
by *intro_classes*

instance *hmultiset* :: *semiring_no_zero_divisors*
by *intro_classes* (*use mult_pos_pos not_gr_zero_hmset* **in** *blast*)

lemma *lt_1_iff_eq_0_hmset*: $M < 1 \longleftrightarrow M = 0$ **for** M :: *hmultiset*
by (*simp add*: *less_iff_add1_le_hmset*)

lemma *zero_less_mult_iff_hmset*[*simp*]: $0 < m * n \longleftrightarrow 0 < m \wedge 0 < n$ **for** $m\ n$ :: *hmultiset*
using *mult_eq_0_iff not_gr_zero_hmset* **by** *blast*

lemma *one_le_mult_iff_hmset*[*simp*]: $1 \leq m * n \longleftrightarrow 1 \leq m \wedge 1 \leq n$ **for** $m\ n$ :: *hmultiset*
by (*metis lt_1_iff_eq_0_hmset mult_eq_0_iff not_le*)

lemma *mult_less_cancel2_hmset*[*simp*]: $m * k < n * k \longleftrightarrow 0 < k \wedge m < n$ **for** $k\ m\ n$ :: *hmultiset*
by (*metis gr_zeroI_hmset leD leI le_cases mult_right_mono mult_zero_right times_hmultiset_monoL*)

lemma *mult_less_cancel1_hmset*[*simp*]: $k * m < k * n \longleftrightarrow 0 < k \wedge m < n$ **for** $k\ m\ n$ :: *hmultiset*
by (*simp add*: *mult.commute*[*of k*])

lemma *mult_le_cancel1_hmset*[*simp*]: $k * m \leq k * n \longleftrightarrow (0 < k \longrightarrow m \leq n)$ **for** $k\ m\ n$ :: *hmultiset*
by (*simp add*: *linorder_not_less*[*symmetric*], *auto*)

lemma *mult_le_cancel2_hmset*[*simp*]: $m * k \leq n * k \longleftrightarrow (0 < k \longrightarrow m \leq n)$ **for** $k\ m\ n$ :: *hmultiset*
by (*simp add*: *linorder_not_less*[*symmetric*], *auto*)

lemma *mult_le_cancel_left1_hmset*: $y > 0 \Longrightarrow x \leq x * y$ **for** $x\ y$ :: *hmultiset*
by (*metis zero_less_iff_1_le_hmset mult.commute mult.left_neutral mult_le_cancel2_hmset*)

lemma *mult_le_cancel_left2_hmset*: $y \leq 1 \Longrightarrow x * y \leq x$ **for** $x\ y$ :: *hmultiset*
by (*metis mult.commute mult.left_neutral mult_le_cancel2_hmset*)

lemma *mult_le_cancel_right1_hmset*: $y > 0 \Longrightarrow x \leq y * x$ **for** $x\ y$ :: *hmultiset*
by (*subst mult.commute*) (*fact mult_le_cancel_left1_hmset*)

lemma *mult_le_cancel_right2_hmset*: $y \leq 1 \Longrightarrow y * x \leq x$ **for** $x\ y$ :: *hmultiset*
by (*subst mult.commute*) (*fact mult_le_cancel_left2_hmset*)

lemma *le_square_hmset*: $m \leq m * m$ **for** m :: *hmultiset*
using *mult_le_cancel_left1_hmset* **by** *force*

lemma *le_cube_hmset*: $m \leq m * (m * m)$ **for** m :: *hmultiset*
using *mult_le_cancel_left1_hmset* **by** *force*

lemma *diff_le_self_hmset*: $m - n \leq m$ **for** $m\ n$ :: *hmultiset*
by (*metis add.commute add.right_neutral diff_add_zero diff_diff_add_hmset le_minus_plus_same_hmset*)

lemma
less_imp_minus_plus_hmset: $m < n \Longrightarrow k < k - m + n$ **and**
le_imp_minus_plus_hmset: $m \leq n \Longrightarrow k \leq k - m + n$ **for** $k\ m\ n$ :: *hmultiset*
by (*meson add_less_cancel_left leD le_minus_plus_same_hmset less_le_trans not_le_imp_less*)+

lemma *gt_0_lt_mult_gt_1_hmset*:
fixes $m\ n$ :: *hmultiset*
assumes $m > 0$ **and** $n > 1$
shows $m < m * n$
using *assms* **by** (*metis mult.right_neutral mult_less_cancel1_hmset*)

instance *hmultiset* :: *linordered_comm_semiring_strict*
by *intro_classes simp*

7.3 Omega

definition ω :: *hmultiset* **where**
ω = *HMSet* {#*1*#}

7.4 Embedding of Natural Numbers

lemma *of_nat_hmset*: *of_nat n* = *HMSet* (*replicate_mset n 0*)
by (*induct n*) (*auto simp*: *zero_hmultiset_def one_hmultiset_def plus_hmultiset_def*)

lemma *of_nat_inject_hmset*[*simp*]: (*of_nat m* :: *hmultiset*) = *of_nat n* $\longleftrightarrow$ $m = n$
unfolding *of_nat_hmset* **by** *simp*

lemma *of_nat_minus_hmset*: *of_nat* $(m - n)$ = (*of_nat m* :: *hmultiset*) − *of_nat n*
unfolding *of_nat_hmset minus_hmultiset_def* **by** *simp*

lemma *plus_of_nat_plus_of_nat_hmset*:
k + *of_nat m* + *of_nat n* = k + *of_nat* $(m + n)$ **for** k :: *hmultiset*
by (*simp add*: *add.assoc*)

lemma *plus_of_nat_minus_of_nat_hmset*:
fixes k :: *hmultiset*
assumes $n \leq m$
shows k + *of_nat m* − *of_nat n* = k + *of_nat* $(m - n)$
using *assms* **by** (*metis add.left_commute add_diff_cancel_left' le_add_diff_inverse of_nat_add*)

lemma *of_nat_lt_ω*[*simp*]: *of_nat n* < ω
by (*auto simp*: *of_nat_hmset zero_less_iff_neq_zero_hmset ω_def less_multiset_ext$_{DM}$_less*)
(*metis One_nat_def count_replicate_mset count_single gr_implies_not0 lessI less_multiset$_{HO}$
not_gr_zero_hmset zero_neq_one*)

lemma *of_nat_ne_ω*[*simp*]: *of_nat n* ≠ ω
by (*metis of_nat_lt_ω mset_le_asym mset_lt_single_iff*)

lemma *of_nat_less_hmset*[*simp*]: (*of_nat M* :: *hmultiset*) < *of_nat N* $\longleftrightarrow$ $M < N$
unfolding *of_nat_hmset less_multiset_ext$_{DM}$_less* **by** *simp*

lemma *of_nat_le_hmset*[*simp*]: (*of_nat M* :: *hmultiset*) ≤ *of_nat N* $\longleftrightarrow$ $M \leq N$
unfolding *of_nat_hmset order_le_less less_multiset_ext$_{DM}$_less* **by** *simp*

7.5 Embedding of Extended Natural Numbers

primrec *hmset_of_enat* :: *enat* ⇒ *hmultiset* **where**
hmset_of_enat (*enat n*) = *of_nat n*
| *hmset_of_enat* ∞ = ω

lemma *hmset_of_enat_0*[*simp*]: *hmset_of_enat 0 = 0*
by (*simp add*: *zero_enat_def*)

lemma *hmset_of_enat_1*[*simp*]: *hmset_of_enat 1 = 1*
by (*simp add*: *one_enat_def del*: *One_nat_def*)

lemma *hmset_of_enat_of_nat*[*simp*]: *hmset_of_enat* (*of_nat n*) = *of_nat n*
using *of_nat_eq_enat* **by** *auto*

lemma *hmset_of_enat_numeral*[*simp*]: *hmset_of_enat* (*numeral n*) = *numeral n*
by (*simp add*: *numeral_eq_enat*)

lemma *hmset_of_enat_le_ω*[*simp*]: *hmset_of_enat n* ≤ *ω*
using *of_nat_lt_ω*[*THEN less_imp_le*] **by** (*cases n*) *auto*

lemma *hmset_of_enat_eq_ω_iff*[*simp*]: *hmset_of_enat n* = *ω* ⟷ *n* = ∞
by (*cases n*) *auto*

7.6 Head Omega

definition *head_ω* :: *hmultiset* ⇒ *hmultiset* **where**
head_ω M = (*if M* = *0 then 0 else HMSet* {#*Max* (*set_mset* (*hmsetmset M*))#})

lemma *head_ω_eq_0_iff*[*simp*]: *head_ω m* = *0* ⟷ *m* = *0*
unfolding *head_ω_def zero_hmultiset_def* **by** *simp*

lemma *head_ω_0*[*simp*]: *head_ω 0* = *0*
unfolding *head_ω_def* **by** *simp*

lemma *head_ω_1*[*simp*]: *head_ω 1* = *1*
unfolding *head_ω_def one_hmultiset_def* **by** *simp*

lemma *head_ω_of_nat*[*simp*]: *head_ω* (*of_nat n*) = (*if n* = *0 then 0 else 1*)
unfolding *head_ω_def one_hmultiset_def of_nat_hmset* **by** *simp*

lemma *head_ω_numeral*[*simp*]: *head_ω* (*numeral n*) = *1*
by (*metis head_ω_of_nat of_nat_numeral zero_neq_numeral*)

lemma *head_ω_ω*[*simp*]: *head_ω ω* = *ω*
unfolding *head_ω_def ω_def* **by** *simp*

lemma *le_imp_head_ω_le*:
assumes *m_le_n*: *m* ≤ *n*
shows *head_ω m* ≤ *head_ω n*
proof −
have *le_in_le_max*: ⋀*a M N*. *M* ≤ *N* ⟹ *a* ∈# *M* ⟹ *a* ≤ *Max* (*set_mset N*)
by (*metis* (*no_types*) *Max_ge finite_set_mset le_less less_eq_multiset*$_{HO}$ *linorder_not_less*
mem_Collect_eq neq0_conv order_trans set_mset_def)
show *?thesis*
using *m_le_n* **unfolding** *head_ω_def*
by (*cases m*, *cases n*,
auto simp del: *hmsetmset_le simp*: *head_ω_def hmsetmset_le*[*symmetric*] *zero_hmultiset_def*,
meson hmsetmset_le le_in_le_max[*OF _ Max_in*[*OF finite_set_mset*]] *set_mset_eq_empty_iff*)
qed

lemma *head_ω_lt_imp_lt*: *head_ω m* < *head_ω n* ⟹ *m* < *n*
unfolding *head_ω_def hmsetmset_less*[*symmetric*]
by (*rule all_lt_Max_imp_lt_multiset*, *auto simp*: *zero_hmultiset_def split*: *if_splits*)

lemma *head_ω_plus*[*simp*]: *head_ω* (*m* + *n*) = *sup* (*head_ω m*) (*head_ω n*)
proof (*cases m n rule*: *hmultiset.exhaust*[*case_product hmultiset.exhaust*])
case *m_n*: (*HMSet_ HMSet M N*)
show *?thesis*
proof (*cases Max* (*set_mset M*) < *Max* (*set_mset N*))

```
    case True
    thus ?thesis
      unfolding m_n head_ω_def sup_hmultiset_def zero_hmultiset_def plus_hmultiset_def
      by (simp add: Max.union max_def dual_order.strict_implies_order)
  next
    case False
    thus ?thesis
      unfolding m_n head_ω_def sup_hmultiset_def zero_hmultiset_def plus_hmultiset_def
      by simp (metis False Max.union finite_set_mset leI max_def set_mset_eq_empty_iff sup.commute)
  qed
qed

lemma head_ω_times[simp]: head_ω (m * n) = head_ω m * head_ω n
proof (cases m = 0 ∨ n = 0)
  case False
  hence m_nz: m ≠ 0 and n_nz: n ≠ 0
    by simp+

  define δ where δ = hmsetmset m
  define ε where ε = hmsetmset n

  have δ_nemp: δ ≠ {#}
    unfolding δ_def using m_nz by simp
  have ε_nemp: ε ≠ {#}
    unfolding ε_def using n_nz by simp

  let ?D = set_mset δ
  let ?E = set_mset ε
  let ?DE = {z. ∃x ∈ ?D. ∃y ∈ ?E. z = x + y}

  have max_D_in: Max ?D ∈ ?D
    using δ_nemp by simp
  have max_E_in: Max ?E ∈ ?E
    using ε_nemp by simp

  have Max ?DE = Max ?D + Max ?E
  proof (rule order_antisym, goal_cases le ge)
    case le
    have ⋀x y. x ∈ ?D ⟹ y ∈ ?E ⟹ x + y ≤ Max ?D + Max ?E
      by (simp add: add_mono)
    hence mem_imp_le: ⋀z. z ∈ ?DE ⟹ z ≤ Max ?D + Max ?E
      by auto
    show ?case
      by (intro mem_imp_le Max_in, simp, use δ_nemp ε_nemp in fast)
  next
    case ge
    have {z. ∃x ∈ {Max ?D}. ∃y ∈ {Max ?E}. z = x + y} ⊆ {z. ∃x ∈# δ. ∃y ∈# ε. z = x + y}
      using max_D_in max_E_in by fast
    thus ?case
      by simp
  qed
  thus ?thesis
    unfolding δ_def ε_def by (auto simp: head_ω_def image_def times_hmultiset_def)
qed auto
```

7.7 More Inequalities and Some Equalities

```
lemma zero_lt_ω[simp]: 0 < ω
  by (metis of_nat_lt_ω of_nat_0)

lemma one_lt_ω[simp]: 1 < ω
  by (metis enat_defs(2) hmset_of_enat.simps(1) hmset_of_enat_1 of_nat_lt_ω)

lemma numeral_lt_ω[simp]: numeral n < ω
```

using *hmset_of_enat_numeral*[*symmetric*] *hmset_of_enat.simps*(*1*) *of_nat_lt_ω numeral_eq_enat*
by *presburger*

lemma *one_le_ω*[*simp*]: *1* ≤ *ω*
by (*simp add*: *less_imp_le*)

lemma *of_nat_le_ω*[*simp*]: *of_nat n* ≤ *ω*
by (*simp add*: *le_less*)

lemma *numeral_le_ω*[*simp*]: *numeral n* ≤ *ω*
by (*simp add*: *less_imp_le*)

lemma *not_ω_lt_1*[*simp*]: ¬ *ω* < *1*
by (*simp add*: *not_less*)

lemma *not_ω_lt_of_nat*[*simp*]: ¬ *ω* < *of_nat n*
by (*simp add*: *not_less*)

lemma *not_ω_lt_numeral*[*simp*]: ¬ *ω* < *numeral n*
by (*simp add*: *not_less*)

lemma *not_ω_le_1*[*simp*]: ¬ *ω* ≤ *1*
by (*simp add*: *not_le*)

lemma *not_ω_le_of_nat*[*simp*]: ¬ *ω* ≤ *of_nat n*
by (*simp add*: *not_le*)

lemma *not_ω_le_numeral*[*simp*]: ¬ *ω* ≤ *numeral n*
by (*simp add*: *not_le*)

lemma *zero_ne_ω*[*simp*]: *0* ≠ *ω*
by (*metis not_ω_le_1 zero_le_hmset*)

lemma *one_ne_ω*[*simp*]: *1* ≠ *ω*
using *not_ω_le_1* **by** *force*

lemma *numeral_ne_ω*[*simp*]: *numeral n* ≠ *ω*
by (*metis not_ω_le_numeral numeral_le_ω*)

lemma
ω_ne_0[*simp*]: *ω* ≠ *0* **and**
ω_ne_1[*simp*]: *ω* ≠ *1* **and**
ω_ne_of_nat[*simp*]: *ω* ≠ *of_nat m* **and**
ω_ne_numeral[*simp*]: *ω* ≠ *numeral n*
using *zero_ne_ω one_ne_ω of_nat_ne_ω numeral_ne_ω* **by** *metis*+

lemma
hmset_of_enat_inject[*simp*]: *hmset_of_enat m* = *hmset_of_enat n* ⟷ *m* = *n* **and**
hmset_of_enat_less[*simp*]: *hmset_of_enat m* < *hmset_of_enat n* ⟷ *m* < *n* **and**
hmset_of_enat_le[*simp*]: *hmset_of_enat m* ≤ *hmset_of_enat n* ⟷ *m* ≤ *n*
by (*cases m*; *cases n*; *simp*)+

lemma *lt_ω_imp_ex_of_nat*:
assumes *M_lt_ω*: *M* < *ω*
shows ∃ *n*. *M* = *of_nat n*
proof −
have *M_lt_single_1*: *hmsetmset M* < {#*1*#}
by (*rule M_lt_ω*[*unfolded ω_def hmsetmset_less*[*symmetric*] *less_multiset_ext*$_{DM}$*_less hmultiset.sel*])

have *N* = *0* **if** *N* ∈# *hmsetmset M* **for** *N*
proof −
have *0* < *count* (*hmsetmset M*) *N*

```
    using that by auto
  hence N < 1
    by (metis (no_types) M_lt_single_1 count_single gr_implies_not0 less_eq_multiset_HO less_one
      neq_iff not_le)
  thus ?thesis
    by (simp add: lt_1_iff_eq_0_hmset)
 qed
 then obtain n where hmmM: M = HMSet (replicate_mset n 0)
  using ex_replicate_mset_if_all_elems_eq by (metis hmultiset.collapse)
 show ?thesis
  unfolding hmmM of_nat_hmset by blast
qed

lemma le_ω_imp_ex_hmset_of_enat:
 assumes M_le_ω: M ≤ ω
 shows ∃n. M = hmset_of_enat n
proof (cases M = ω)
 case True
 thus ?thesis
  by (metis hmset_of_enat.simps(2))
next
 case False
 thus ?thesis
  using M_le_ω lt_ω_imp_ex_of_nat by (metis hmset_of_enat.simps(1) le_less)
qed

lemma lt_ω_lt_ω_imp_times_lt_ω: M < ω ⟹ N < ω ⟹ M * N < ω
 by (metis lt_ω_imp_ex_of_nat of_nat_lt_ω of_nat_mult)

lemma times_ω_minus_of_nat[simp]: m * ω - of_nat n = m * ω
 by (auto intro!: Diff_triv_mset simp: ω_def times_hmultiset_def minus_hmultiset_def
   Times_mset_single_right of_nat_hmset disjunct_not_in image_def)

lemma times_ω_minus_numeral[simp]: m * ω - numeral n = m * ω
 by (metis of_nat_numeral times_ω_minus_of_nat)

lemma ω_minus_of_nat[simp]: ω - of_nat n = ω
 by (rule times_ω_minus_of_nat[of 1, simplified])

lemma ω_minus_1[simp]: ω - 1 = ω
 by (simp add: ω_minus_of_nat[of 1, simplified])

lemma ω_minus_numeral[simp]: ω - numeral n = ω
 by (rule times_ω_minus_numeral[of 1, simplified])

lemma hmset_of_enat_minus_enat[simp]: hmset_of_enat (m - enat n) = hmset_of_enat m - of_nat n
 by (cases m) (auto simp: of_nat_minus_hmset)

lemma of_nat_lt_hmset_of_enat_iff: of_nat m < hmset_of_enat n ⟷ enat m < n
 by (metis hmset_of_enat.simps(1) hmset_of_enat_less)

lemma of_nat_le_hmset_of_enat_iff: of_nat m ≤ hmset_of_enat n ⟷ enat m ≤ n
 by (metis hmset_of_enat.simps(1) hmset_of_enat_le)

lemma hmset_of_enat_lt_iff_ne_infinity: hmset_of_enat x < ω ⟷ x ≠ ∞
 by (cases x; simp)

lemma minus_diff_sym_hmset: m - (m - n) = n - (n - m) for m n :: hmultiset
 unfolding minus_hmultiset_def by simp (metis multiset_inter_def subset_mset.inf_aci(1))

lemma diff_plus_sym_hmset: (c - b) + b = (b - c) + c for b c :: hmultiset
proof -
 have f1: ⋀h ha :: hmultiset. h - (ha + h) = 0
```

```
    by (simp add: add.commute)
  have f2: ⋀h ha hb :: hmultiset. h + ha − (h − hb) = hb + ha − (hb − h)
    by (metis (no_types) add_diff_cancel_right minus_diff_sym_hmset)
  have ⋀h ha hb :: hmultiset. h + (ha + hb) − hb = h + ha
    by (metis (no_types) add.assoc add_diff_cancel_right')
  then show ?thesis
    using f2 f1 by (metis (no_types) add.commute add.right_neutral diff_diff_add_hmset)
qed

lemma times_diff_plus_sym_hmset: a * (c − b) + a * b = a * (b − c) + a * c for a b c :: hmultiset
  by (metis distrib_left diff_plus_sym_hmset)

lemma times_of_nat_minus_left:
  (of_nat m − of_nat n) * l = of_nat m * l − of_nat n * l for l :: hmultiset
  by (induct n m rule: diff_induct) (auto simp: ring_distribs)

lemma times_of_nat_minus_right:
  l * (of_nat m − of_nat n) = l * of_nat m − l * of_nat n for l :: hmultiset
  by (metis times_of_nat_minus_left mult.commute)

lemma lt_ω_imp_times_minus_left: m < ω ⟹ n < ω ⟹ (m − n) * l = m * l − n * l
  by (metis lt_ω_imp_ex_of_nat times_of_nat_minus_left)

lemma lt_ω_imp_times_minus_right: m < ω ⟹ n < ω ⟹ l * (m − n) = l * m − l * n
  by (metis lt_ω_imp_ex_of_nat times_of_nat_minus_right)

lemma hmset_pair_decompose:
  ∃k n1 n2. m1 = k + n1 ∧ m2 = k + n2 ∧ (head_ω n1 ≠ head_ω n2 ∨ n1 = 0 ∧ n2 = 0)
proof −
  define n1 where n1: n1 = m1 − m2
  define n2 where n2: n2 = m2 − m1
  define k where k1: k = m1 − n1

  have k2: k = m2 − n2
    using k1 unfolding n1 n2 by (simp add: minus_diff_sym_hmset)

  have m1 = k + n1
    unfolding k1
    by (metis (no_types) n1 add_diff_cancel_left add.commute add_diff_cancel_right' diff_add_zero
      diff_diff_add minus_diff_sym_hmset)
  moreover have m2 = k + n2
    unfolding k2
    by (metis n2 add.commute add_diff_cancel_left add_diff_cancel_left' add_diff_cancel_right'
      diff_add_zero diff_diff_add diff_zero k2 minus_diff_sym_hmset)
  moreover have hd_n: head_ω n1 ≠ head_ω n2 if n1_or_n2_nz: n1 ≠ 0 ∨ n2 ≠ 0
  proof (cases n1 = 0 n2 = 0 rule: bool.exhaust[case_product bool.exhaust])
    case False_False
    note n1_nz = this(1)[simplified] and n2_nz = this(2)[simplified]

    define δ1 where δ1 = hmsetmset n1
    define δ2 where δ2 = hmsetmset n2

    have δ1_inter_δ2: δ1 ∩# δ2 = {#}
      unfolding δ1_def δ2_def n1 n2 minus_hmultiset_def by (simp add: diff_intersect_sym_diff)

    have δ1_ne: δ1 ≠ {#}
      unfolding δ1_def using n1_nz by simp
    have δ2_ne: δ2 ≠ {#}
      unfolding δ2_def using n2_nz by simp

    have max_δ1: Max (set_mset δ1) ∈# δ1
      using δ1_ne by simp
    have max_δ2: Max (set_mset δ2) ∈# δ2
```

```
    using δ2_ne by simp
  have max_δ1_ne_δ2: Max (set_mset δ1) ≠ Max (set_mset δ2)
    using δ1_inter_δ2 disjunct_not_in max_δ1 max_δ2 by force

  show ?thesis
    using n1_nz n2_nz
    by (cases n1 rule: hmultiset.exhaust_sel, cases n2 rule: hmultiset.exhaust_sel,
      auto simp: head_ω_def zero_hmultiset_def max_δ1_ne_δ2[unfolded δ1_def δ2_def])
 qed (use n1_or_n2_nz in ⟨auto simp: head_ω_def⟩)
 ultimately show ?thesis
  by blast
qed

lemma hmset_pair_decompose_less:
 assumes m1_lt_m2: m1 < m2
 shows ∃k n1 n2. m1 = k + n1 ∧ m2 = k + n2 ∧ head_ω n1 < head_ω n2
proof −
 obtain k n1 n2 where
  m1: m1 = k + n1 and
  m2: m2 = k + n2 and
  hds: head_ω n1 ≠ head_ω n2 ∨ n1 = 0 ∧ n2 = 0
  using hmset_pair_decompose[of m1 m2] by blast

 {
  assume n1 = 0 and n2 = 0
  hence m1 = m2
   unfolding m1 m2 by simp
  hence False
   using m1_lt_m2 by simp
 }
 moreover
 {
  assume head_ω n1 > head_ω n2
  hence n1 > n2
   by (rule head_ω_lt_imp_lt)
  hence m1 > m2
   unfolding m1 m2 by simp
  hence False
   using m1_lt_m2 by simp
 }
 ultimately show ?thesis
  using m1 m2 hds by (blast elim: neqE)
qed

lemma hmset_pair_decompose_less_eq:
 assumes m1 ≤ m2
 shows ∃k n1 n2. m1 = k + n1 ∧ m2 = k + n2 ∧ (head_ω n1 < head_ω n2 ∨ n1 = 0 ∧ n2 = 0)
 using assms
 by (metis add_cancel_right_right hmset_pair_decompose_less order.not_eq_order_implies_strict)

lemma mono_cross_mult_less_hmset:
 fixes Aa A Ba B :: hmultiset
 assumes A_lt: A < Aa and B_lt: B < Ba
 shows A * Ba + B * Aa < A * B + Aa * Ba
proof −
 obtain j m1 m2 where A: A = j + m1 and Aa: Aa = j + m2 and hd_m: head_ω m1 < head_ω m2
  by (metis hmset_pair_decompose_less[OF A_lt])
 obtain k n1 n2 where B: B = k + n1 and Ba: Ba = k + n2 and hd_n: head_ω n1 < head_ω n2
  by (metis hmset_pair_decompose_less[OF B_lt])

 have hd_lt: head_ω (m1 * n2 + m2 * n1) < head_ω (m1 * n1 + m2 * n2)
 proof simp
  have ⋀h ha :: hmultiset. 0 < h ∨ ¬ ha < h
```

```
    by force
  hence ¬ head_ω m2 * head_ω n2 ≤ sup (head_ω m1 * head_ω n2) (head_ω m2 * head_ω n1)
    using hd_m hd_n sup_hmultiset_def by auto
  thus sup (head_ω m1 * head_ω n2) (head_ω m2 * head_ω n1)
    < sup (head_ω m1 * head_ω n1) (head_ω m2 * head_ω n2)
    by (meson leI sup.bounded_iff)
 qed
 show ?thesis
  unfolding A Aa B Ba ring_distribs by (simp add: algebra_simps head_ω_lt_imp_lt[OF hd_lt])
qed

lemma triple_cross_mult_hmset:
 An * (Bn * Cn + Bp * Cp − (Bn * Cp + Cn * Bp))
 + (Cn * (An * Bp + Bn * Ap − (An * Bn + Ap * Bp))
   + (Ap * (Bn * Cp + Cn * Bp − (Bn * Cn + Bp * Cp))
     + Cp * (An * Bn + Ap * Bp − (An * Bp + Bn * Ap)))) =
 An * (Bn * Cp + Cn * Bp − (Bn * Cn + Bp * Cp))
 + (Cn * (An * Bn + Ap * Bp − (An * Bp + Bn * Ap))
   + (Ap * (Bn * Cn + Bp * Cp − (Bn * Cp + Cn * Bp))
     + Cp * (An * Bp + Bn * Ap − (An * Bn + Ap * Bp))))
 for Ap An Bp Bn Cp Cn Dp Dn :: hmultiset
 apply (simp add: algebra_simps)
 apply (unfold add.assoc[symmetric])

 apply (rule add_right_cancel[THEN iffD1, of _ Cp * (An * Bp + Ap * Bn)])
 apply (unfold add.assoc)
 apply (subst times_diff_plus_sym_hmset)
 apply (unfold add.assoc[symmetric])
 apply (subst (12) add.commute)
 apply (subst (11) add.commute)
 apply (unfold add.assoc[symmetric])

 apply (rule add_right_cancel[THEN iffD1, of _ Cn * (An * Bn + Ap * Bp)])
 apply (unfold add.assoc)
 apply (subst times_diff_plus_sym_hmset)
 apply (unfold add.assoc[symmetric])
 apply (subst (14) add.commute)
 apply (subst (13) add.commute)
 apply (unfold add.assoc[symmetric])

 apply (rule add_right_cancel[THEN iffD1, of _ Ap * (Bn * Cn + Bp * Cp)])
 apply (unfold add.assoc)
 apply (subst times_diff_plus_sym_hmset)
 apply (unfold add.assoc[symmetric])
 apply (subst (16) add.commute)
 apply (subst (15) add.commute)
 apply (unfold add.assoc[symmetric])

 apply (rule add_right_cancel[THEN iffD1, of _ An * (Bn * Cp + Bp * Cn)])
 apply (unfold add.assoc)
 apply (subst times_diff_plus_sym_hmset)
 apply (unfold add.assoc[symmetric])
 apply (subst (18) add.commute)
 apply (subst (17) add.commute)
 apply (unfold add.assoc[symmetric])

 by (simp add: algebra_simps)
```

7.8 Conversions to Natural Numbers

```
definition offset_hmset :: hmultiset ⇒ nat where
 offset_hmset M = count (hmsetmset M) 0

lemma offset_hmset_of_nat[simp]: offset_hmset (of_nat n) = n
```

unfolding *offset_hmset_def of_nat_hmset* **by** *simp*

lemma *offset_hmset_numeral*[*simp*]: *offset_hmset* (*numeral n*) = *numeral n*
unfolding *offset_hmset_def* **by** (*metis offset_hmset_def offset_hmset_of_nat of_nat_numeral*)

definition *sum_coefs* :: *hmultiset* ⇒ *nat* **where**
sum_coefs M = *size* (*hmsetmset M*)

lemma *sum_coefs_distrib_plus*[*simp*]: *sum_coefs* (*M* + *N*) = *sum_coefs M* + *sum_coefs N*
unfolding *plus_hmultiset_def sum_coefs_def* **by** *simp*

lemma *sum_coefs_gt_0*: *sum_coefs M* > *0* ⟷ *M* > *0*
by (*auto simp*: *sum_coefs_def zero_hmultiset_def hmsetmset_less*[*symmetric*] *less_multiset_ext*$_{DM}$*_less nonempty_has_size*[*symmetric*])

7.9 An Example

The following proof is based on an informal proof by Uwe Waldmann, inspired by a similar argument by Michel Ludwig.

lemma *waldmann_less*:
fixes *α1 α2 β1 β2 γ δ* :: *hmultiset*
assumes
αβ2γ_lt_αβ1γ: *α2* + *β2* ∗ *γ* < *α1* + *β1* ∗ *γ* **and**
β2_le_β1: *β2* ≤ *β1* **and**
γ_lt_δ: *γ* < *δ*
shows *α2* + *β2* ∗ *δ* < *α1* + *β1* ∗ *δ*
proof −
obtain *β0 β2a β1a* **where**
β2: *β2* = *β0* + *β2a* **and**
β1: *β1* = *β0* + *β1a* **and**
hd_β2a_vs_β1a: *head_ω β2a* < *head_ω β1a* ∨ *β2a* = *0* ∧ *β1a* = *0*
using *hmset_pair_decompose_less_eq*[*OF β2_le_β1*] **by** *blast*

obtain *η γa δa* **where**
γ: *γ* = *η* + *γa* **and**
δ: *δ* = *η* + *δa* **and**
hd_γa_lt_δa: *head_ω γa* < *head_ω δa*
using *hmset_pair_decompose_less*[*OF γ_lt_δ*] **by** *blast*

have *α2* + *β0* ∗ *γ* + *β2a* ∗ *γ* = *α2* + *β2* ∗ *γ*
unfolding *β2* **by** (*simp add*: *add.commute add.left_commute distrib_left mult.commute*)
also have ... < *α1* + *β1* ∗ *γ*
by (*rule αβ2γ_lt_αβ1γ*)
also have ... = *α1* + *β0* ∗ *γ* + *β1a* ∗ *γ*
unfolding *β1* **by** (*simp add*: *add.commute add.left_commute distrib_left mult.commute*)
finally have *α2* + *β0* ∗ *γ* + *β2a* ∗ *γ* < *α1* + *β0* ∗ *γ* + *β1a* ∗ *γ*
by *assumption*
hence ∗: *α2* + *β2a* ∗ *γ* < *α1* + *β1a* ∗ *γ*
by (*metis add_less_cancel_right semiring_normalization_rules*(*23*))

have ∗∗: *β1a* ∗ *γa* + *β2a* ∗ *δa* ≤ *β1a* ∗ *δa*
proof (*cases β2a* = *0* ∧ *β1a* = *0*)
case *False*
hence *head_ω β2a* < *head_ω β1a*
using *hd_β2a_vs_β1a* **by** *blast*
hence *head_ω* (*β1a* ∗ *γa* + *β2a* ∗ *δa*) < *head_ω* (*β1a* ∗ *δa*)
using *hd_γa_lt_δa* **by** (*auto intro*: *gr_zeroI_hmset simp*: *sup_hmultiset_def*)
hence *β1a* ∗ *γa* + *β2a* ∗ *δa* < *β1a* ∗ *δa*
by (*rule head_ω_lt_imp_lt*)
thus *?thesis*
by *simp*
qed *simp*

have $\alpha 2 + \beta 2 * \delta = \alpha 2 + \beta 0 * \delta + \beta 2a * \delta$
unfolding $\beta 2$ **by** (*simp add*: *ab_semigroup_add_class.add_ac*(1) *distrib_right*)
also have $\ldots = \alpha 2 + \beta 0 * \delta + \beta 2a * \eta + \beta 2a * \delta a$
unfolding δ **by** (*simp add*: *distrib_left semiring_normalization_rules*(25))
also have $\ldots \leq \alpha 2 + \beta 0 * \delta + \beta 2a * \eta + \beta 2a * \delta a + \beta 2a * \gamma a$
by *simp*
also have $\ldots = \alpha 2 + \beta 2a * \gamma + \beta 0 * \delta + \beta 2a * \delta a$
unfolding γ *distrib_left add.assoc*[*symmetric*] **by** (*simp add*: *semiring_normalization_rules*(23))
also have $\ldots < \alpha 1 + \beta 1a * \gamma + \beta 0 * \delta + \beta 2a * \delta a$
using $*$ **by** *simp*
also have $\ldots = \alpha 1 + \beta 1a * \eta + \beta 1a * \gamma a + \beta 0 * \eta + \beta 0 * \delta a + \beta 2a * \delta a$
unfolding γ δ *distrib_left add.assoc*[*symmetric*] **by** (*rule refl*)
also have $\ldots \leq \alpha 1 + \beta 1a * \eta + \beta 0 * \eta + \beta 0 * \delta a + \beta 1a * \delta a$
proof −
have $\neg\ \alpha 1 + \beta 1a * \eta + \beta 0 * \eta + \beta 0 * \delta a + \beta 1a * \delta a$
$< \alpha 1 + \beta 1a * \eta + \beta 0 * \eta + \beta 0 * \delta a + (\beta 1a * \gamma a + \beta 2a * \delta a)$
using $**$ **by** *force*
thus *?thesis*
by (*simp add*: *semiring_normalization_rules*(23,25))
qed
also have $\ldots = \alpha 1 + \beta 1 * \delta$
unfolding $\beta 1$ δ
by (*simp add*: *distrib_left distrib_right add.assoc*[*symmetric*] *semiring_normalization_rules*(23))
finally show *?thesis*
by *assumption*
qed

end

8 Signed Syntactic Ordinals in Cantor Normal Form

theory *Signed_Syntactic_Ordinal*
imports *Signed_Hereditary_Multiset Syntactic_Ordinal*
begin

8.1 Natural (Hessenberg) Product

instantiation *zhmultiset* :: *comm_ring_1*
begin

lift-definition *one_zhmultiset* :: *zhmultiset* **is** $\{\#0\#\}_z$.

lift-definition *times_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **is**
$\lambda M\ N.$
zmset_of (*hmsetmset* (*HMSet* (*mset_pos M*) ∗ *HMSet* (*mset_pos N*)))
− *zmset_of* (*hmsetmset* (*HMSet* (*mset_pos M*) ∗ *HMSet* (*mset_neg N*)))
+ *zmset_of* (*hmsetmset* (*HMSet* (*mset_neg M*) ∗ *HMSet* (*mset_neg N*)))
− *zmset_of* (*hmsetmset* (*HMSet* (*mset_neg M*) ∗ *HMSet* (*mset_pos N*))) .

lemmas *zhmsetmset_times* = *times_zhmultiset.rep_eq*

instance
proof (*intro_classes*, *goal_cases mult_assoc mult_comm mult_1 distrib zero_neq_one*)
case (*mult_assoc a b c*)
show *?case*
by (*transfer*,
simp add: *algebra_simps zmset_of_plus*[*symmetric*] *hmsetmset_plus*[*symmetric*] *HMSet_diff*,
rule triple_cross_mult_hmset)
next
case (*mult_comm a b*)
show *?case*
by *transfer* (*auto simp*: *algebra_simps*)
next

```
  case (mult_ 1 a)
  show ?case
    by transfer (auto simp: algebra_simps mset_pos_neg_partition[symmetric])
next
  case (distrib a b c)

  show ?case
    by (simp add: times_zhmultiset_def ZHMSet_plus[symmetric] zmset_of_plus[symmetric]
        hmsetmset_plus[symmetric] algebra_simps,
      simp add: add.assoc[symmetric] hmset_pos_plus hmset_neg_plus,
      simp add: algebra_simps,
      simp add: mult.commute[of _ hmset_pos c] mult.commute[of _ hmset_neg c]
        add.commute[of hmset_neg c * M hmset_pos c * N for M N]
        add.left_commute[of hmset_neg c * M hmset_pos c * N for M N],
      simp add: add.assoc[symmetric] ring_distribs(1)[symmetric] hmset_pos_neg_dual)
next
  case zero_neq_one
  show ?case
    unfolding zero_zhmultiset_def one_zhmultiset_def by simp
qed

end

lemma zhmset_of_1: zhmset_of 1 = 1
  by (simp add: one_hmultiset_def one_zhmultiset_def)

lemma zhmset_of_times: zhmset_of (A * B) = zhmset_of A * zhmset_of B
  by transfer simp

lemma zhmset_of_prod_list:
  zhmset_of (prod_list Ms) = prod_list (map zhmset_of Ms)
  by (induct Ms) (auto simp: one_hmultiset_def one_zhmultiset_def zhmset_of_times)
```

8.2 Omega

```
definition ω_z :: zhmultiset where
  ω_z = ZHMSet {#1#}_z

lemma ω_z_as_ω: ω_z = zhmset_of ω
  unfolding ω_z_def ω_def by simp
```

8.3 Embedding of Natural Numbers

```
lemma of_nat_zhmset: of_nat n = zhmset_of (of_nat n)
  by (induct n) (auto simp: zero_zhmultiset_def zhmset_of_plus zhmset_of_1)

lemma of_nat_inject_zhmset[simp]: (of_nat m :: zhmultiset) = of_nat n ⟷ m = n
  unfolding of_nat_zhmset by simp

lemma plus_of_nat_plus_of_nat_zhmset:
  k + of_nat m + of_nat n = k + of_nat (m + n) for k :: zhmultiset
  by (simp add: add.assoc)

lemma plus_of_nat_minus_of_nat_zhmset:
  fixes k :: zhmultiset
  assumes n ≤ m
  shows k + of_nat m - of_nat n = k + of_nat (m - n)
  using assms by (metis add.left_commute add_diff_cancel_left' le_add_diff_inverse of_nat_add)

lemma of_nat_lt_ω_z[simp]: of_nat n < ω_z
  by (simp add: ω_z_as_ω of_nat_zhmset zmset_of_less)

lemma of_nat_ne_ω_z[simp]: of_nat n ≠ ω_z
  by (metis of_nat_lt_ω_z mset_le_asym mset_lt_single_iff)
```

8.4 Embedding of Extended Natural Numbers

primrec *zhmset_of_enat* :: *enat* ⇒ *zhmultiset* **where**
  zhmset_of_enat (*enat n*) = *of_nat n*
| *zhmset_of_enat* ∞ = ω_z

lemma *zhmset_of_enat_0*[*simp*]: *zhmset_of_enat 0* = *0*
  by (*simp add*: *zero_enat_def*)

lemma *zhmset_of_enat_1*[*simp*]: *zhmset_of_enat 1* = *1*
  by (*simp add*: *one_enat_def del*: *One_nat_def*)

lemma *zhmset_of_enat_of_nat*[*simp*]: *zhmset_of_enat* (*of_nat n*) = *of_nat n*
  using *of_nat_eq_enat* **by** *auto*

lemma *zhmset_of_enat_numeral*[*simp*]: *zhmset_of_enat* (*numeral n*) = *numeral n*
  by (*simp add*: *numeral_eq_enat*)

lemma *zhmset_of_enat_le_*ω_z[*simp*]: *zhmset_of_enat n* ≤ ω_z
  using *of_nat_lt_*ω_z[*THEN less_imp_le*] **by** (*cases n*) *auto*

lemma *zhmset_of_enat_eq_*ω_z*_iff*[*simp*]: *zhmset_of_enat n* = ω_z ⟷ *n* = ∞
  by (*cases n*) *auto*

8.5 Inequalities and Some (Dis)equalities

instance *zhmultiset* :: *zero_less_one*
  by (*intro_classes*, *transfer*, *transfer*, *auto*)

instantiation *zhmultiset* :: *linordered_idom*
begin

definition *sgn_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* **where**
  sgn_zhmultiset M = (*if M* = *0 then 0 else if M* > *0 then 1 else* −*1*)

definition *abs_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* **where**
  abs_zhmultiset M = (*if M* < *0 then* − *M else M*)

lemma *gt_0_times_gt_0_imp*:
  fixes *a b* :: *zhmultiset*
  assumes *a_gt0*: *a* > *0* **and** *b_gt0*: *b* > *0*
  shows *a* ∗ *b* > *0*
proof −
  show *?thesis*
    using *a_gt0 b_gt0*
    by (*subst* (*asm*) (*2 4*) *zhmset_pos_neg_partition*, *simp*, *transfer*,
        simp del: *HMSet_less add*: *algebra_simps zmset_of_plus*[*symmetric*] *hmsetmset_plus*[*symmetric*]
        zmset_of_less HMSet_less[*symmetric*])
      (*rule mono_cross_mult_less_hmset*)
qed

instance
proof *intro_classes*
  fix *a b c* :: *zhmultiset*

  assume
    a_lt_b: *a* < *b* **and**
    zero_lt_c: *0* < *c*

  have *c* ∗ *b* < *c* ∗ *b* + *c* ∗ (*b* − *a*)
    using *gt_0_times_gt_0_imp* **by** (*simp add*: *a_lt_b zero_lt_c*)
  hence *c* ∗ *a* + *c* ∗ (*b* − *a*) < *c* ∗ *b* + *c* ∗ (*b* − *a*)
    by (*simp add*: *right_diff_distrib'*)
  thus *c* ∗ *a* < *c* ∗ *b*

```
    by simp
qed (auto simp: sgn_zhmultiset_def abs_zhmultiset_def)

end

lemma le_zhmset_of_pos: M ≤ zhmset_of (hmset_pos M)
  by (metis add_diff_cancel_left diff_le_eq hmultiset.sel less_eq_multiset_plus_left ZHMSet_le
    mset_pos_as_neg zhmsetmset_inverse zmset_of_le zmset_of_plus)

lemma minus_zhmset_of_pos_le: − zhmset_of (hmset_neg M) ≤ M
  by (metis le_zhmset_of_pos minus_le_iff mset_pos_uminus zhmsetmset_uminus)

lemma zhmset_of_nonneg[simp]: zhmset_of M ≥ 0
  by (metis hmsetmset_0 zero_le_hmset zero_zhmultiset_def zhmset_of_le zmset_of_empty)

lemma
  fixes n :: zhmultiset
  assumes 0 ≤ m
  shows
    le_add1_hmset: n ≤ n + m and
    le_add2_hmset: n ≤ m + n
  using assms by simp+

lemma less_iff_add1_le_zhmset: m < n ⟷ m + 1 ≤ n for m n :: zhmultiset
proof
  assume m_lt_n: m < n
  show m + 1 ≤ n
  proof −
    obtain hh :: hmultiset and zz :: zhmultiset and hha :: hmultiset where
      f1: m = zhmset_of hh + zz ∧ n = zhmset_of hha + zz ∧ hh < hha
      using less_hmset_zhmsetE[OF m_lt_n] by metis
    hence zhmset_of (hh + 1) ≤ zhmset_of hha
      by (metis (no_types) less_iff_add1_le_hmset zhmset_of_le)
    thus ?thesis
      using f1 by (simp add: zhmset_of_1 zhmset_of_plus)
  qed
qed simp

lemma gt_0_lt_mult_gt_1_zhmset:
  fixes m n :: zhmultiset
  assumes m > 0 and n > 1
  shows m < m * n
  using assms by simp

lemma zero_less_iff_1_le_zhmset: 0 < n ⟷ 1 ≤ n for n :: zhmultiset
  by (rule less_iff_add1_le_zhmset[of 0, simplified])

lemma less_add_1_iff_le_hmset: m < n + 1 ⟷ m ≤ n for m n :: zhmultiset
  by (rule less_iff_add1_le_zhmset[of m n + 1, simplified])

lemma nonneg_le_mult_right_mono_zhmset:
  fixes x y z :: zhmultiset
  assumes x: 0 ≤ x and y: 0 < y and z: x ≤ z
  shows x ≤ y * z
  using x zero_less_iff_1_le_zhmset[THEN iffD1, OF y] z
  by (meson dual_order.trans leD mult_less_cancel_right2 not_le_imp_less)

instance hmultiset :: ordered_cancel_comm_semiring
  by intro_classes

instance hmultiset :: linordered_semiring_1_strict
  by intro_classes
```

instance *hmultiset* :: *bounded_lattice_bot*
by *intro_classes*

instance *hmultiset* :: *zero_less_one*
by *intro_classes*

instance *hmultiset* :: *linordered_nonzero_semiring*
by *intro_classes*

instance *hmultiset* :: *semiring_no_zero_divisors*
by *intro_classes*

lemma *zero_lt_ω_z*[*simp*]: $0 < \omega_z$
by (*metis of_nat_lt_ω_z of_nat_0*)

lemma *one_lt_ω*[*simp*]: $1 < \omega_z$
by (*metis enat_defs*(2) *zhmset_of_enat.simps*(1) *zhmset_of_enat_1 of_nat_lt_ω_z*)

lemma *numeral_lt_ω_z*[*simp*]: *numeral* $n < \omega_z$
using *zhmset_of_enat_numeral*[*symmetric*] *zhmset_of_enat.simps*(1) *of_nat_lt_ω_z numeral_eq_enat*
by *presburger*

lemma *one_le_ω_z*[*simp*]: $1 \leq \omega_z$
by (*simp add*: *less_imp_le*)

lemma *of_nat_le_ω_z*[*simp*]: *of_nat* $n \leq \omega_z$
by (*simp add*: *le_less*)

lemma *numeral_le_ω_z*[*simp*]: *numeral* $n \leq \omega_z$
by (*simp add*: *less_imp_le*)

lemma *not_ω_z_lt_1*[*simp*]: $\neg\ \omega_z < 1$
by (*simp add*: *not_less*)

lemma *not_ω_z_lt_of_nat*[*simp*]: $\neg\ \omega_z <$ *of_nat* n
by (*simp add*: *not_less*)

lemma *not_ω_z_lt_numeral*[*simp*]: $\neg\ \omega_z <$ *numeral* n
by (*simp add*: *not_less*)

lemma *not_ω_z_le_1*[*simp*]: $\neg\ \omega_z \leq 1$
by (*simp add*: *not_le*)

lemma *not_ω_z_le_of_nat*[*simp*]: $\neg\ \omega_z \leq$ *of_nat* n
by (*simp add*: *not_le*)

lemma *not_ω_z_le_numeral*[*simp*]: $\neg\ \omega_z \leq$ *numeral* n
by (*simp add*: *not_le*)

lemma *zero_ne_ω_z*[*simp*]: $0 \neq \omega_z$
using *zero_lt_ω_z* **by** *linarith*

lemma *one_ne_ω_z*[*simp*]: $1 \neq \omega_z$
using *not_ω_z_le_1* **by** *force*

lemma *numeral_ne_ω_z*[*simp*]: *numeral* $n \neq \omega_z$
by (*metis not_ω_z_le_numeral numeral_le_ω_z*)

lemma
ω_z_ne_0[*simp*]: $\omega_z \neq 0$ **and**
ω_z_ne_1[*simp*]: $\omega_z \neq 1$ **and**
ω_z_ne_of_nat[*simp*]: $\omega_z \neq$ *of_nat* m **and**
ω_z_ne_numeral[*simp*]: $\omega_z \neq$ *numeral* n

using *zero_ne_ω_z one_ne_ω_z of_nat_ne_ω_z numeral_ne_ω_z* **by** *metis*+

lemma
zhmset_of_enat_inject[*simp*]: *zhmset_of_enat m* = *zhmset_of_enat n* ⟷ *m* = *n* **and**
zhmset_of_enat_lt_iff_lt[*simp*]: *zhmset_of_enat m* < *zhmset_of_enat n* ⟷ *m* < *n* **and**
zhmset_of_enat_le_iff_le[*simp*]: *zhmset_of_enat m* ≤ *zhmset_of_enat n* ⟷ *m* ≤ *n*
by (*cases m*; *cases n*; *simp*)+

lemma *of_nat_lt_zhmset_of_enat_iff*: *of_nat m* < *zhmset_of_enat n* ⟷ *enat m* < *n*
by (*metis zhmset_of_enat.simps*(*1*) *zhmset_of_enat_lt_iff_lt*)

lemma *of_nat_le_zhmset_of_enat_iff*: *of_nat m* ≤ *zhmset_of_enat n* ⟷ *enat m* ≤ *n*
by (*metis zhmset_of_enat.simps*(*1*) *zhmset_of_enat_le_iff_le*)

lemma *zhmset_of_enat_lt_iff_ne_infinity*: *zhmset_of_enat x* < *ω_z* ⟷ *x* ≠ ∞
by (*cases x*; *simp*)

8.6 An Example

A new proof of ⟦*?α2.0* + *?β2.0* ∗ *?γ* < *?α1.0* + *?β1.0* ∗ *?γ*; *?β2.0* ≤ *?β1.0*; *?γ* < *?δ*⟧ ⟹ *?α2.0* + *?β2.0* ∗ *?δ* < *?α1.0* + *?β1.0* ∗ *?δ*:

lemma
fixes *α1 α2 β1 β2 γ δ* :: *hmultiset*
assumes
αβ2γ_lt_αβ1γ: *α2* + *β2* ∗ *γ* < *α1* + *β1* ∗ *γ* **and**
β2_le_β1: *β2* ≤ *β1* **and**
γ_lt_δ: *γ* < *δ*
shows *α2* + *β2* ∗ *δ* < *α1* + *β1* ∗ *δ*
proof −
let *?Z* = *zhmset_of*

note *αβ2γ_lt_αβ1γ′* = *αβ2γ_lt_αβ1γ*[*THEN zhmset_of_less*[*THEN iffD2*],
simplified zhmset_of_plus zhmset_of_times]
note *β2_le_β1′* = *β2_le_β1*[*THEN zhmset_of_le*[*THEN iffD2*]]
note *γ_lt_δ′* = *γ_lt_δ*[*THEN zhmset_of_less*[*THEN iffD2*]]

have *?Z α2* + *?Z β2* ∗ *?Z δ* = *?Z α2* + *?Z β2* ∗ *?Z γ* + *?Z β2* ∗ (*?Z δ* − *?Z γ*)
by (*simp add*: *algebra_simps*)
also have ... < *?Z α1* + *?Z β1* ∗ *?Z γ* + *?Z β2* ∗ (*?Z δ* − *?Z γ*)
using *αβ2γ_lt_αβ1γ′* **by** *simp*
also have ... ≤ *?Z α1* + *?Z β1* ∗ *?Z γ* + *?Z β1* ∗ (*?Z δ* − *?Z γ*)
using *β2_le_β1′ γ_lt_δ′* **by** *simp*
also have ... = *?Z α1* + *?Z β1* ∗ *?Z δ*
by (*simp add*: *algebra_simps*)
finally have *?Z α2* + *?Z β2* ∗ *?Z δ* < *?Z α1* + *?Z β1* ∗ *?Z δ*
by *assumption*
thus *?thesis*
by (*simp add*: *zmset_of_less zhmset_of_times*[*symmetric*]
zhmset_of_plus[*symmetric*])
qed

end