Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

August 16, 2018

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 *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*]

2.2 Lemmas about Intersection, Union and Pointwise Inclusion

lemma *subset_add_mset_notin_subset_mset*: ‹$A \subseteq\# \mathit{add\_mset}\ b\ B \Longrightarrow b \notin\# A \Longrightarrow A \subseteq\# B$›
by (*simp add*: *subset_mset.le_iff_sup*)

lemma *subset_msetE*: $[\![A \subset\# B; [\![A \subseteq\# B; \neg\ B \subseteq\# A]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*simp add*: *subset_mset.less_le_not_le*)

lemma *Diff_triv_mset*: $M \cap\# N = \{\#\} \Longrightarrow M - N = M$
by (*metis diff_intersect_left_idem diff_zero*)

lemma *diff_intersect_sym_diff*: $(A - B) \cap\# (B - A) = \{\#\}$
unfolding *multiset_inter_def*
proof −
have $A - (B - (B - A)) = A - B$
by (*metis diff_intersect_right_idem multiset_inter_def*)
then show $A - B - (A - B - (B - A)) = \{\#\}$
by (*metis diff_add diff_cancel diff_subset_eq_self subset_mset.diff_add*)
qed

declare *subset_msetE* [*elim!*]

2.3 Lemmas about Filter and Image

lemma *count_image_mset_ge_count*: $\mathit{count}\ (\mathit{image\_mset}\ f\ A)\ (f\ b) \geq \mathit{count}\ A\ b$

```
  by (induction A) auto

lemma count_image_mset_inj:
  assumes ‹inj f›
  shows ‹count (image_mset f M) (f x) = count M x›
  by (induct M) (use assms in ‹auto simp: inj_on_def›)

lemma count_image_mset_le_count_inj_on:
  inj_on f (set_mset M) ⟹ count (image_mset f M) y ≤ count M (inv_into (set_mset M) f y)
proof (induct M)
  case (add x M)
  note ih = this(1) and inj_xM = this(2)

  have inj_M: inj_on f (set_mset M)
    using inj_xM by simp

  show ?case
  proof (cases x ∈# M)
    case x_in_M: True
    show ?thesis
    proof (cases y = f x)
      case y_eq_fx: True
      show ?thesis
        using x_in_M ih[OF inj_M] unfolding y_eq_fx by (simp add: inj_M insert_absorb)
    next
      case y_ne_fx: False
      show ?thesis
        using x_in_M ih[OF inj_M] y_ne_fx insert_absorb by fastforce
    qed
  next
    case x_ni_M: False
    show ?thesis
    proof (cases y = f x)
      case y_eq_fx: True
      have f x ∉# image_mset f M
        using x_ni_M inj_xM by force
      thus ?thesis
        unfolding y_eq_fx
        by (metis (no_types) inj_xM count_add_mset count_greater_eq_Suc_zero_iff count_inI
          image_mset_add_mset inv_into_f_f union_single_eq_member)
    next
      case y_ne_fx: False
      show ?thesis
      proof (rule ccontr)
        assume neg_conj: ¬ count (image_mset f (add_mset x M)) y
          ≤ count (add_mset x M) (inv_into (set_mset (add_mset x M)) f y)

        have cnt_y: count (add_mset (f x) (image_mset f M)) y = count (image_mset f M) y
          using y_ne_fx by simp

        have inv_into (set_mset M) f y ∈# add_mset x M ⟹
          inv_into (set_mset (add_mset x M)) f (f (inv_into (set_mset M) f y)) =
          inv_into (set_mset M) f y
          by (meson inj_xM inv_into_f_f)
        hence 0 < count (image_mset f (add_mset x M)) y ⟹
          count M (inv_into (set_mset M) f y) = 0 ∨ x = inv_into (set_mset M) f y
          using neg_conj cnt_y ih[OF inj_M]
          by (metis (no_types) count_add_mset count_greater_zero_iff count_inI f_inv_into_f
            image_mset_add_mset set_image_mset)
        thus False
          using neg_conj cnt_y x_ni_M ih[OF inj_M]
          by (metis (no_types) count_greater_zero_iff count_inI eq_iff image_mset_add_mset
            less_imp_le)
```

qed
qed
qed
qed *simp*

lemma *mset_filter_compl*: *mset* (*filter p xs*) + *mset* (*filter* (*Not* ∘ *p*) *xs*) = *mset xs*
by (*induction xs*) (*auto simp*: *mset_filter ac_simps*)

Near duplicate of *filter_eq_replicate_mset*: {#*y* ∈# *?D*. *y* = *?x*#} = *replicate_mset* (*count ?D ?x*) *?x*.

lemma *filter_mset_eq*: *filter_mset* ((=) *L*) *A* = *replicate_mset* (*count A L*) *L*
by (*auto simp*: *multiset_eq_iff*)

lemma *filter_mset_cong*[*fundef_cong*]:
assumes *M* = *M'* ⋀*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 assms*)
qed

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*:
(⋀*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*)

declare *image_mset_cong2* [*cong*]

2.4 Lemmas about Sum

lemma *sum_image_mset_sum_map*[*simp*]: *sum_mset* (*image_mset f* (*mset xs*)) = *sum_list* (*map f xs*)
by (*metis mset_map sum_mset_sum_list*)

lemma *sum_image_mset_mono*:
fixes *f* :: *'a* ⇒ *'b*::*canonically_ordered_monoid_add*
assumes *sub*: *A* ⊆# *B*
shows (∑ *m* ∈# *A*. *f m*) ≤ (∑ *m* ∈# *B*. *f m*)
by (*metis image_mset_union le_iff_add sub subset_mset.add_diff_inverse sum_mset.union*)

lemma *sum_image_mset_mono_mem*:
n ∈# *M* ⟹ *f n* ≤ (∑ *m* ∈# *M*. *f m*) **for** *f* :: *'a* ⇒ *'b*::*canonically_ordered_monoid_add*
using *le_iff_add multi_member_split* **by** *fastforce*

lemma *count_sum_mset_if_1_0*: ‹*count M a* = (∑ *x*∈#*M*. *if x* = *a then 1 else 0*)›
by (*induction M*) *auto*

lemma *sum_ mset_ dvd*:
fixes $k :: 'a::comm\_semiring\_1\_cancel$
assumes $\forall m \in\# M.\ k\ dvd\ f\ m$
shows $k\ dvd\ (\sum m \in\# M.\ f\ m)$
using *assms* **by** (*induct M*) *auto*

lemma *sum_ mset_ distrib_ div_ if_ dvd*:
fixes $k :: 'a::unique\_euclidean\_semiring$
assumes $\forall m \in\# M.\ k\ dvd\ f\ m$
shows $(\sum m \in\# M.\ f\ m)\ div\ k = (\sum m \in\# M.\ f\ m\ div\ k)$
using *assms* **by** (*induct M*) (*auto simp*: *div_ plus_ div_ distrib_ dvd_ left*)

2.5 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* ((≠) *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 *removeAll_ subseteq_ remove1_ mset*: *removeAll_ mset x M* $\subseteq\#$ *remove1_ mset x M*
by (*auto simp*: *subseteq_ mset_ def*)

lemma *in_ remove1_ mset_ neq*:
assumes *ab*: $a \neq b$
shows $a \in\#$ *remove1_ mset b C* $\longleftrightarrow a \in\# C$
by (*metis assms diff_ single_ trivial in_ diffD insert_ DiffM insert_ noteq_ member*)

lemma *size_ mset_ removeAll_ mset_ le_ iff*: *size* (*removeAll_ mset x M*) < *size M* $\longleftrightarrow x \in\# M$
by (*auto intro*: *count_ inI mset_ subset_ size simp*: *subset_ mset_ def multiset_ eq_ iff*)

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*)

lemma *size_ mset_ remove1_ mset_ le_ iff*: *size* (*remove1_ mset x M*) < *size M* $\longleftrightarrow x \in\# M$
using *less_ irrefl*
by (*fastforce intro*!: *mset_ subset_ size elim*: *in_ countE simp*: *subset_ mset_ def multiset_ eq_ iff*)

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* $\in\#$ *x1* $\Longrightarrow$ *x1* = *M* + {#*L*#} $\Longrightarrow$ *P*) $\Longrightarrow$
(*L* $\notin\#$ *x1* $\Longrightarrow$ *x1* = *M* $\Longrightarrow$ *P*) $\Longrightarrow$
P
by (*cases L* $\in\#$ *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 filter_filter_mset filter_mset_neq*)

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)

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*

lemma *add_mset_less_imp_less_remove1_mset*:
assumes *xM_lt_N*: *add_mset x M* < *N*
shows *M* < *remove1_mset x N*
proof −
have *M* < *N*
using *assms le_multiset_right_total mset_le_trans* **by** *blast*
then show *?thesis*
by (*metis add_less_cancel_right add_mset_add_single diff_single_trivial insert_DiffM2 xM_lt_N*)
qed

2.6 Lemmas about Replicate

lemma *replicate_mset_minus_replicate_mset_same*[*simp*]:
replicate_mset m x − *replicate_mset n x* = *replicate_mset* (*m* − *n*) *x*
by (*induct m arbitrary*: *n*, *simp*, *metis left_diff_repeat_mset_distrib' repeat_mset_replicate_mset*)

lemma *replicate_mset_subset_iff_lt*[*simp*]: *replicate_mset m x* ⊂# *replicate_mset n x* ⟷ *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* ⊆# *replicate_mset n x* ⟷ *m* ≤ *n*
by (*induct n m rule*: *diff_induct*) *auto*

lemma *replicate_mset_lt_iff_lt*[*simp*]: *replicate_mset m x* < *replicate_mset n x* ⟷ *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* ≤ *replicate_mset n x* ⟷ *m* ≤ *n*
by (*induct n m rule*: *diff_induct*) *auto*

lemma *replicate_mset_eq_iff*[*simp*]:
replicate_mset m x = *replicate_mset n y* ⟷ *m* = *n* ∧ (*m* ≠ *0* ⟶ *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*} ⟷ (∃ *n* > *0*. *U* = *replicate_mset n a*)
by (*rule*, *metis count_greater_zero_iff count_replicate_mset insertI1 multi_count_eq singletonD*
zero_less_iff_neq_zero, *force*)

lemma *ex_replicate_mset_if_all_elems_eq*:
assumes ∀ *x* ∈# *M*. *x* = *y*
shows ∃ *n*. *M* = *replicate_mset n y*
using *assms* **by** (*metis count_replicate_mset mem_Collect_eq multiset_eqI neq0_conv set_mset_def*)

2.7 Multiset and Set Conversions

lemma *count_mset_set_if*: *count* (*mset_set A*) *a* = (*if a* ∈ *A* ∧ *finite A then 1 else 0*)
by *auto*

lemma *mset_set_set_mset_empty_mempty*[*iff*]: *mset_set* (*set_mset D*) = {#} ⟷ *D* = {#}
by (*simp add*: *mset_set_empty_iff*)

lemma *count_mset_set_le_one*: *count* (*mset_set A*) *x* ≤ *1*
by (*simp add*: *count_mset_set_if*)

lemma *mset_set_set_mset_subseteq*[*simp*]: *mset_set* (*set_mset A*) ⊆# *A*
by (*simp add*: *mset_set_set_mset_msubset*)

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 *sorted_sorted_list_of_multiset*[*simp*]:
sorted (*sorted_list_of_multiset* (*M* :: *'a*::*linorder multiset*))
by (*metis mset_sorted_list_of_multiset sorted_list_of_multiset_mset sorted_sort*)

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

lemma *sorted_list_of_multiset_eq_Nil*[*simp*]: *sorted_list_of_multiset M* = [] ⟷ *M* = {#}
by (*metis mset_sorted_list_of_multiset sorted_list_of_multiset_empty*)

2.8 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*)

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* $\Longrightarrow$ *size C* = *card* (*set_mset C*)
by (*induction C*) *auto*

lemma *distinct_mset_add*:
distinct_mset (*L* + *L'*) $\longleftrightarrow$ *distinct_mset L* $\wedge$ *distinct_mset L'* $\wedge$ *L* $\cap\#$ *L'* = {#}
by (*induction L arbitrary*: *L'*) *auto*

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

lemma *distinct_finite_set_mset_subseteq_iff* [*iff*]:
assumes *distinct_mset M finite N*
shows *set_mset M* $\subseteq$ *N* $\longleftrightarrow$ *M* $\subseteq\#$ *mset_set N*
by (*metis assms distinct_mset_set_mset_ident finite_set_mset msubset_mset_set_iff*)

lemma *distinct_mem_diff_mset*:
assumes *dist*: *distinct_mset M* **and** *mem*: *x* $\in$ *set_mset* (*M* − *N*)
shows *x* $\notin$ *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 *distinct_mset M distinct_mset N set_mset M* = *set_mset N*
shows *M* = *N*
using *assms distinct_mset_set_mset_ident* **by** *fastforce*

lemma *distinct_mset_union_mset*[*simp*]:
‹*distinct_mset* (*D* $\cup\#$ *C*) $\longleftrightarrow$ *distinct_mset D* $\wedge$ *distinct_mset C*›
unfolding *distinct_mset_count_less_1* **by** *force*

lemma *distinct_mset_inter_mset*:
distinct_mset C $\Longrightarrow$ *distinct_mset* (*C* $\cap\#$ *D*)
distinct_mset D $\Longrightarrow$ *distinct_mset* (*C* $\cap\#$ *D*)
by (*simp_all add*: *multiset_inter_def*,
metis distinct_mset_minus multiset_inter_commute multiset_inter_def)

lemma *distinct_mset_remove1_All*: *distinct_mset C* $\Longrightarrow$ *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*#} $\longleftrightarrow$ *a* $\neq$ *b*
by *auto*

lemma *distinct_mset_filter*: *distinct_mset M* $\Longrightarrow$ *distinct_mset* {# *L* $\in\#$ *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*) $\Longrightarrow$ *distinct_mset* (*image_mset f M*) $\longleftrightarrow$ *distinct_mset M*›
by (*induction M*) (*auto simp*: *inj_on_def*)

2.9 Repeat Operation

lemma *repeat_mset_compower*: *repeat_mset n A* = (((+) *A*) ^^ *n*) {#}
by (*induction n*) *auto*

lemma *repeat_mset_prod*: *repeat_mset* (*m* * *n*) *A* = (((+) (*repeat_mset n A*)) ^^ *m*) {#}
by (*induction m*) (*auto simp*: *repeat_mset_distrib*)

2.10 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** ×# *80*) **where**
Times_mset A B ≡ *Sigma_mset A* (λ_. *B*)

hide-const (**open**) *Times_mset*

Contrary to the set version $A \times B$, 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* ×# *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*
by (*unfold Sigma_mset_def*) *auto*

Elimination of (*a*, *b*) ∈# *A* ×# *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*#} ×# *B* = *image_mset* (*Pair a*) *B*
unfolding *Sigma_mset_def* **by** *auto*

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

lemma *Times_mset_single_single*[*simp*]: {#*a*#} ×# {#*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* ×# {#} = {#}
by (*auto simp*: *multiset_eq_iff count_Sigma_mset*)

lemma *Sigma_mset_mono*:
assumes *A* ⊆# *C* **and** ⋀*x*. *x* ∈# *A* ⟹ *B x* ⊆# *D x*
shows *Sigma_mset A B* ⊆# *Sigma_mset C D*
proof −
have *count A a* ∗ *count* (*B a*) *b* ≤ *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*) ∈# *Sigma_mset A B*) = (*a* ∈# *A* ∧ *b* ∈# *B a*)
by *blast*

lemma *mem_Times_mset_iff*: *x* ∈# *A* ×# *B* ⟷ *fst x* ∈# *A* ∧ *snd x* ∈# *B*
by (*induct x*) *simp*

lemma *Sigma_mset_empty_iff*: (*SIGMAMSET i*∈#*I*. *X i*) = {#} ⟷ (∀*i*∈#*I*. *X i* = {#})
by (*auto simp*: *Sigma_mset_def*)

lemma *Times_mset_subset_mset_cancel1*: *x* ∈# *A* ⟹ (*A* ×# *B* ⊆# *A* ×# *C*) = (*B* ⊆# *C*)
by (*auto simp*: *subseteq_mset_def count_Sigma_mset*)

lemma *Times_mset_subset_mset_cancel2*: *x* ∈# *C* ⟹ (*A* ×# *C* ⊆# *B* ×# *C*) = (*A* ⊆# *B*)
by (*auto simp*: *subseteq_mset_def count_Sigma_mset*)

lemma *Times_mset_eq_cancel2*: *x* ∈# *C* ⟹ (*A* ×# *C* = *B* ×# *C*) = (*A* = *B*)
by (*auto simp*: *multiset_eq_iff count_Sigma_mset dest*!: *in_countE*)

lemma *split_paired_Ball_mset_Sigma_mset*[*simp*]:
(∀*z*∈#*Sigma_mset A B*. *P z*) ⟷ (∀*x*∈#*A*. ∀*y*∈#*B x*. *P* (*x*, *y*))
by *blast*

lemma *split_paired_Bex_mset_Sigma_mset*[*simp*]:
(∃*z*∈#*Sigma_mset A B*. *P z*) ⟷ (∃*x*∈#*A*. ∃*y*∈#*B x*. *P* (*x*, *y*))
by *blast*

lemma *sum_mset_if_eq_constant*:
(∑*x*∈#*M*. *if a* = *x then* (*f x*) *else* 0) = (((+) (*f a*)) ^^ (*count M a*)) 0
by (*induction M*) (*auto simp*: *ac_simps*)

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

lemma *untion_image_mset_Pair_distribute*:
⋃#{#*image_mset* (*Pair x*) (*C x*). *x* ∈# *J* − *I*#} =
⋃# {#*image_mset* (*Pair x*) (*C x*). *x* ∈# *J*#} − ⋃#{#*image_mset* (*Pair x*) (*C x*). *x* ∈# *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* $i \in\# I$. 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*: $(A \cup\# B) \times\# C = A \times\# C \cup\# B \times\# C$
by (*fact Sigma_mset_Un_distrib1*)

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

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

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

lemma *Times_insert_left*: $A \times\#$ *add_mset* x B = $A \times\# B +$ *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 $A \times\# B = A \times\# B +$ *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 $(A \times\# B)$ = (*if* $B = \{\#\}$ *then* $\{\#\}$ *else repeat_mset* (*size* B) A)
by (*induct* B) (*auto simp*: *Times_mset_single_right ac_simps Times_insert_left*)

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

lemma *product_swap_mset*: *image_mset prod.swap* $(A \times\# B) = B \times\# A$
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 ×# B*

lemma *member_product_mset*: *x ∈# product_mset A B ⟷ x ∈# A ×# B*
by (*simp add*: *Multiset_More.product_mset_def*)

end

lemma *count_Sigma_mset_abs_def*: *count* (*Sigma_mset A B*) = (λ(*a*, *b*) ⇒ *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 ×# B = image_mset* (λ(*a*, *b*). (*f a*, *b*)) (*A ×# B*)
by (*induct B*) (*auto simp*: *Times_insert_left*)

lemma *Times_mset_image_mset2*: *A ×# image_mset f B = image_mset* (λ(*a*, *b*). (*a*, *f b*)) (*A ×# B*)
by (*induct A*) (*auto simp*: *Times_insert_right*)

lemma *sum_le_singleton*: *A* ⊆ {*x*} ⟹ *sum f A* = (*if x* ∈ *A then f x else* 0)
by (*auto simp*: *subset_singleton_iff elim*: *finite_subset*)

lemma *Times_mset_assoc*: (*A ×# B*) *×# C = image_mset* (λ(*a*, *b*, *c*). ((*a*, *b*), *c*)) (*A ×# B ×# 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*)

2.11 Transfer Rules

lemma *plus_multiset_transfer*[*transfer_rule*]:
(*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_mset R*) (*rel_mset R*))) (+) (+)
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*))) (−) (−)
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 ∃*xs″ ys″*.
mset xs″ = *mset xs* − *mset xs′* ∧ *mset ys″* = *mset ys* − *mset ys′* ∧ *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* (=))) *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*) (=)))
    (λ*M N*. *filter_mset* (*Domainp R*) *M* ⊆# *filter_mset* (*Domainp R*) *N*) (⊆#)
proof −
  have *count_filter_mset_less*:
    (∀ *a*. *count* (*filter_mset* (*Domainp R*) *M*) *a* ≤ *count* (*filter_mset* (*Domainp R*) *N*) *a*) ⟷
     (∀ *a* ∈ {*x*. *Domainp R x*}. *count M a* ≤ *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 ⟹ *rel_fun R* (*rel_fun R R*) (+) (+) ⟹ (*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*

2.12 Even More about Multisets

2.12.1 Multisets and functions

lemma *range_image_mset*:
  assumes *set_mset Ds* ⊆ *range f*
  shows *Ds* ∈ *range* (*image_mset f*)
proof −
  have ∀ *D*. *D* ∈# *Ds* ⟶ (∃ *C*. *f C* = *D*)
    using *assms* **by** *blast*
  then obtain *f_i* **where**
    f_p: ∀ *D*. *D* ∈# *Ds* ⟶ (*f* (*f_i D*) = *D*)
    by *metis*
  define *Cs* **where**
    Cs ≡ *image_mset f_i Ds*
  from *f_p Cs_def* **have** *image_mset f Cs* = *Ds*
    by *auto*
  then show *?thesis*
    by *blast*
qed

2.12.2 Multisets and lists

definition *list_of_mset* :: *'a multiset* ⇒ *'a list* **where**
  list_of_mset m = (*SOME l*. *m* = *mset l*)

lemma *list_of_mset_exi*: ∃ *l*. *m* = *mset l*
  using *ex_mset* **by** *metis*

lemma [*simp*]: *mset* (*list_of_mset m*) = *m*
  by (*metis* (*mono_tags*, *lifting*) *ex_mset list_of_mset_def someI_ex*)

lemma *range_mset_map*:
  assumes *set_mset Ds* ⊆ *range f*
  shows *Ds* ∈ *range* (λ*Cl*. *mset* (*map f Cl*))
proof −
  have *Ds* ∈ *range* (*image_mset f*)
    by (*simp add*: *assms range_image_mset*)
  then obtain *Cs* **where** *Cs_p*: *image_mset f Cs* = *Ds*
    by *auto*
  define *Cl* **where** *Cl* = *list_of_mset Cs*
  then have *mset Cl* = *Cs*
    by *auto*

```
  then have image_mset f (mset Cl) = Ds
    using Cs_p by auto
  then have mset (map f Cl) = Ds
    by auto
  then show ?thesis
    by auto
qed

lemma list_of_mset_empty[iff]: list_of_mset m = [] ⟷ m = {#}
  by (metis (mono_tags, lifting) ex_mset list_of_mset_def mset_zero_iff_right someI_ex)

lemma in_mset_conv_nth: (x ∈# mset xs) = (∃i<length xs. xs ! i = x)
  by (auto simp: in_set_conv_nth)

lemma in_mset_sum_list:
  assumes L ∈# LL
  assumes LL ∈ set Ci
  shows L ∈# sum_list Ci
  using assms by (induction Ci) auto

lemma in_mset_sum_list2:
  assumes L ∈# sum_list Ci
  obtains LL where
    LL ∈ set Ci
    L ∈# LL
  using assms by (induction Ci) auto

lemma subseteq_list_Union_mset:
  assumes length Ci = n
  assumes length CAi = n
  assumes ∀i<n.  Ci ! i ⊆# CAi ! i
  shows ⋃#mset Ci ⊆# ⋃#mset CAi
  using assms proof (induction n arbitrary: Ci CAi)
  case 0
  then show ?case by auto
next
  case (Suc n)
  from Suc have ∀i<n. tl Ci ! i ⊆# tl CAi ! i
    by (simp add: nth_tl)
  hence ⋃#mset (tl Ci) ⊆# ⋃#mset (tl CAi) using Suc by auto
  moreover
  have hd Ci ⊆# hd CAi using Suc
    by (metis hd_conv_nth length_greater_0_conv zero_less_Suc)
  ultimately
  show ⋃#mset Ci ⊆# ⋃#mset CAi
    using Suc by (cases Ci; cases CAi) (auto intro: subset_mset.add_mono)
qed
```

2.12.3 More on multisets and functions

```
lemma subseteq_mset_size_eql: X ⊆# Y ⟹ size Y = size X ⟹ X = Y
  using mset_subset_size subset_mset_def by fastforce

lemma image_mset_of_subset_list:
  assumes image_mset η C' = mset lC
  shows ∃qC'. map η qC' = lC ∧ mset qC' = C'
  using assms apply (induction lC arbitrary: C')
  subgoal by simp
  subgoal by (fastforce dest!: msed_map_invR intro: exI[of _ ‹_ # _›])
  done

lemma image_mset_of_subset:
  assumes A ⊆# image_mset η C'
  shows ∃A'. image_mset η A' = A ∧ A' ⊆# C'
```

```
proof −
  define C where C = image_mset η C'

  define lA where lA = list_of_mset A
  define lD where lD = list_of_mset (C−A)
  define lC where lC = lA @ lD

  have mset lC = C
    using C_def assms unfolding lD_def lC_def lA_def by auto
  then have ∃ qC'. map η qC' = lC ∧ mset qC' = C'
    using assms image_mset_of_subset_list unfolding C_def by metis
  then obtain qC' where qC'_p: map η qC' = lC ∧ mset qC' = C'
    by auto
  let ?lA' = take (length lA) qC'
  have m: map η ?lA' = lA
    using qC'_p lC_def
    by (metis append_eq_conv_conj take_map)
  let ?A' = mset ?lA'

  have image_mset η ?A' = A
    using m using lA_def
    by (metis (full_types) ex_mset list_of_mset_def mset_map someI_ex)
  moreover have ?A' ⊆# C'
    using qC'_p unfolding lA_def
    using mset_take_subseteq by blast
  ultimately show ?thesis by blast
qed

lemma all_the_same: ∀ x ∈# X. x = y ⟹ card (set_mset X) ≤ Suc 0
  by (metis card.empty card.insert card_mono finite.intros(1) finite_insert le_SucI singletonI subsetI)

lemma Melem_subseteq_Union_mset[simp]:
  assumes x ∈# T
  shows x ⊆# ⋃#T
  using assms sum_mset.remove by force

lemma Melem_subset_eq_sum_list[simp]:
  assumes x ∈# mset T
  shows x ⊆# sum_list T
  using assms by (metis mset_subset_eq_add_left sum_mset.remove sum_mset_sum_list)

lemma less_subset_eq_Union_mset[simp]:
  assumes i < length CAi
  shows CAi ! i ⊆# ⋃#mset CAi
proof −
  from assms have CAi ! i ∈# mset CAi
    by auto
  then show ?thesis
    by auto
qed

lemma less_subset_eq_sum_list[simp]:
  assumes i < length CAi
  shows CAi ! i ⊆# sum_list CAi
proof −
  from assms have CAi ! i ∈# mset CAi
    by auto
  then show ?thesis
    by auto
qed

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**
λ*x* (*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 (*'*(∈#$_z$*'*)) **and**
elem_zmset ((_ / ∈#$_z$ _) [*51*, *51*] *50*)

notation (*ASCII*)
elem_zmset (*'*(:#*z'*)) **and**
elem_zmset ((_ / :#*z* _) [*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 (*'*(∉#$_z$*'*)) **and**
not_elem_zmset ((_ / ∉#$_z$ _) [*51*, *51*] *50*)

notation (*ASCII*)
  not_ elem_ zmset ('(~:#z')) **and**
  not_ elem_ zmset ((_/ ~:#z _) [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∀ _ ∈#z _./ _) [0, 0, 10] 10)
  _ *MBex* :: *pttrn* ⇒ 'a *set* ⇒ *bool* ⇒ *bool* ((3∃ _ ∈#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*∈#z*A*. *P* ⇌ *CONST Signed_ Multiset.Ball A* (λ*x*. *P*)
  ∃*x*∈#z*A*. *P* ⇌ *CONST Signed_ Multiset.Bex A* (λ*x*. *P*)

lemma *zcount_ eq_ zero_ iff*: *zcount M x* = *0* ⟷ *x* ∉#z *M*
  by (*auto simp add*: *set_ zmset_ def*)

lemma *not_ in_ iff_ zmset*: *x* ∉#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* ∈#z *M*
  by (*auto simp add*: *set_ zmset_ def*)

lemma *zcount_ inI*:
  assumes *zcount M x* = *0* ⟹ *False*
  shows *x* ∈#z *M*
proof (*rule ccontr*)
  assume *x* ∉#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 $A \neq \{\#\}_z$
obtains *x* **where** $x \in\#_z A$
proof −
have $\exists x.\ x \in\#_z A$
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*: $a \in\#_z M - N \longleftrightarrow$ *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* ⇒ *'a zmultiset* **is** $\lambda f.\ (Abs\_multiset\ f, \{\#\})$.

lemma *zmset_ of_ inject*[*simp*]: *zmset_ of M* = *zmset_ of N* ⟷ *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* ⇒ *'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** $\lambda(Mp, Mn).\ count\ (Mn - Mp)$
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 ⇒ 'a zmultiset ⇒ 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 ⇒ 'a zmultiset ⇒ 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 ⇒ 'a zmultiset ⇒ bool* (**infix** $\supseteq\#_z$ *50*)
where
$supseteq\_zmset\ A\ B \equiv B \subseteq\#_z A$

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

notation (*input*)
subseteq_zmset (**infix** $\subseteq\#_z$ *50*) **and**
supseteq_zmset (**infix** $\supseteq\#_z$ *50*)

notation (*ASCII*)
subseteq_zmset (**infix** $\subseteq\#_z$ *50*) **and**
subset_zmset (**infix** $\subset\#_z$ *50*) **and**
supseteq_zmset (**infix** $\supseteq\#_z$ *50*) **and**
supset_zmset (**infix** $>\#_z$ *50*)

interpretation *subset_zmset*: *ordered_ab_semigroup_add_imp_le* (+) (−) ($\subseteq\#_z$) ($\subset\#_z$)
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 (+) *0* (−) ($\subseteq\#_z$) ($\subset\#_z$)
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* $(\subseteq\#_z)$ $(\subset\#_z)$
by *unfold_locales*

3.4 Replicate and Repeat Operations

definition *replicate_zmset* :: $nat \Rightarrow 'a \Rightarrow 'a\ zmultiset$ **where**
$replicate\_zmset\ n\ x = (add\_zmset\ x\ \hat{}\hat{}\ n)\ \{\#\}_z$

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* :: $nat \Rightarrow 'a\ zmultiset \Rightarrow 'a\ zmultiset$ **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'*:
$\langle i \geq j \Longrightarrow repeat\_zmset\ (i - j)\ u = repeat\_zmset\ i\ u - repeat\_zmset\ j\ u \rangle$
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*{# :#*z* _./ _#}))
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* ⊆#$_z$ *B*
shows *filter_zmset f A* ⊆#$_z$ *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*]: $\{\#y \in\#_z\ M.\ True\#\} = M$ **and**
filter_zmset_False[*simp*]: $\{\#y \in\#_z\ M.\ False\#\} = \{\#\}_z$
by (*auto simp*: *zmultiset_eq_iff*)

3.5 Uncategorized

lemma *multi_drop_mem_not_eq_zmset*: $B - \{\#c\#\}_z \neq B$
by (*simp add*: *diff_single_eq_union_zmset*)

lemma *zmultiset_partition*: $M = \{\#x \in\#_z\ M.\ P\ x\ \#\} + \{\#x \in\#_z\ M.\ \neg\ P\ x\#\}$
by (*subst zmultiset_eq_iff*) *auto*

3.6 Image

definition *image_zmset* :: $('a \Rightarrow 'b) \Rightarrow 'a\ zmultiset \Rightarrow 'b\ zmultiset$ **where**
image_zmset f M =
zmset_of (*fold_mset* (*add_mset* $\circ$ *f*) {#} (*mset_pos M*)) −
zmset_of (*fold_mset* (*add_mset* $\circ$ *f*) {#} (*mset_neg M*))

3.7 Multiset Order

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

lift-definition *less_zmultiset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ **is**
$\lambda(Mp, Mn)\ (Np, Nn).\ Mp + Nn < Mn + Np$
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 $A1 + D2 < B2 + C1 \longleftrightarrow A1 + A2 + D2 < A2 + B2 + C1$
by *simp*
also have $\ldots \longleftrightarrow B1 + B2 + D2 < A2 + B2 + C1$
unfolding *ab* **by** (*rule refl*)
also have $\ldots \longleftrightarrow B1 + D2 < A2 + C1$
by *simp*
also have $\ldots \longleftrightarrow B1 + D1 + D2 < A2 + C1 + D1$
by *simp*
also have $\ldots \longleftrightarrow B1 + C1 + C2 < A2 + C1 + D1$
using *cd* **by** (*simp add*: *add.assoc*)
also have $\ldots \longleftrightarrow B1 + C2 < A2 + D1$
by *simp*
finally show $A1 + D2 < B2 + C1 \longleftrightarrow B1 + C2 < A2 + D1$
by *assumption*
qed

definition *less_eq_zmultiset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ **where**
less_eq_zmultiset M' M $\longleftrightarrow M' < M \vee M' = M$

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*: $A1 + A2 \neq B1 + B2$

{
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*: $\neg\ x < y \longleftrightarrow x \geq y$ **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* $\longleftrightarrow$ *M < N*
by (*clarsimp simp*: *zmset_of_def*, *transfer*, *simp*)+

lemma *zmset_of_le*: *zmset_of M* $\leq$ *zmset_of N* $\longleftrightarrow$ *M* $\leq$ *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*)

lemma *uminus_add_conv_diff_mset*[*cancelation_simproc_pre*]: ‹$-a + b = b - a$› **for** *a* :: ‹*'a zmultiset*›
by (*simp add*: *add.commute*)

lemma *uminus_add_add_uminus*[*cancelation_simproc_pre*]: ‹$b - a + c = b + c - a$› **for** *a* :: ‹*'a zmultiset*›
by (*simp add*: *uminus_add_conv_diff_mset zmset_subset_eq_zmultiset_union_diff_commute*)

lemma *add_zmset_eq_add_NO_MATCH*[*cancelation_simproc_pre*]:
‹*NO_MATCH* $\{\#\}_z$ *H* $\Longrightarrow$ *add_zmset a H* = $\{\#a\#\}_z$ + *H*›
by *auto*

lemma *repeat_zmset_iterate_add*: ‹*repeat_zmset n M* = *iterate_add n M*›
unfolding *iterate_add_def* **by** (*induction n*) *auto*

declare *repeat_zmset_iterate_add*[*cancelation_simproc_pre*]

declare *repeat_zmset_iterate_add*[*symmetric*, *cancelation_simproc_post*]

simproc-setup *zmseteq_cancel_numerals*
((*l*::*'a zmultiset*) + *m* = *n* | (*l*::*'a zmultiset*) = *m* + *n* |
add_zmset a m = *n* | *m* = *add_zmset a n* |
replicate_zmset p a = *n* | *m* = *replicate_zmset p a* |
repeat_zmset p m = *n* | *m* = *repeat_zmset p m*) =
‹*fn phi* => *Cancel_Simprocs.eq_cancel*›

lemma *zmset_subseteq_add_iff1*:
‹$j \leq i \Longrightarrow$ (*repeat_zmset i u* + *m* $\subseteq\#_z$ *repeat_zmset j u* + *n*) = (*repeat_zmset* $(i - j)$ *u* + *m* $\subseteq\#_z$ *n*)›
by (*simp add*: *add.commute add_diff_eq left_diff_repeat_zmset_distrib' subset_eq_diff_conv_zmset*)

lemma *zmset_subseteq_add_iff2*:
‹$i \leq j \Longrightarrow$ (*repeat_zmset i u* + *m* $\subseteq\#_z$ *repeat_zmset j u* + *n*) = (*m* $\subseteq\#_z$ *repeat_zmset* $(j - i)$ *u* + *n*)›
proof −
assume $i \leq j$
then have $\bigwedge z$. *repeat_zmset j* (*z*::*'a zmultiset*) − *repeat_zmset i z* = *repeat_zmset* $(j - i)$ *z*
by (*simp add*: *left_diff_repeat_zmset_distrib'*)
then show *?thesis*
by (*metis add.commute diff_diff_eq2 subset_eq_diff_conv_zmset*)
qed

lemma *zmset_subset_add_iff1*:
‹$j \leq i \Longrightarrow$ (*repeat_zmset i u* + *m* $\subset\#_z$ *repeat_zmset j u* + *n*) = (*repeat_zmset* $(i - j)$ *u* + *m* $\subset\#_z$ *n*)›
by (*simp add*: *subset_zmset.less_le_not_le zmset_subseteq_add_iff1 zmset_subseteq_add_iff2*)

lemma *zmset_subset_add_iff2*:

⟨$i \leq j \Longrightarrow (repeat\_zmset\ i\ u + m \subset\#_z repeat\_zmset\ j\ u + n) = (m \subset\#_z repeat\_zmset\ (j - i)\ u + n)$⟩
by (*simp add*: *subset_zmset.less_le_not_le zmset_subseteq_add_iff1 zmset_subseteq_add_iff2*)

ML-file ⟨*zmultiset_simprocs.ML*⟩

simproc-setup *zmsetsubset_cancel*
((l::'*a zmultiset*) + $m \subset\#_z n$ | (l::'*a zmultiset*) $\subset\#_z m + n$ |
add_zmset a m $\subset\#_z n$ | $m \subset\#_z$ *add_zmset a n* |
replicate_zmset p a $\subset\#_z n$ | $m \subset\#_z$ *replicate_zmset p a* |
repeat_zmset p m $\subset\#_z n$ | $m \subset\#_z$ *repeat_zmset p m*) =
⟨*fn phi => ZMultiset_Simprocs.subset_cancel_zmsets*⟩

simproc-setup *zmsetsubseteq_cancel*
((l::'*a zmultiset*) + $m \subseteq\#_z n$ | (l::'*a zmultiset*) $\subseteq\#_z m + n$ |
add_zmset a m $\subseteq\#_z n$ | $m \subseteq\#_z$ *add_zmset a n* |
replicate_zmset p a $\subseteq\#_z n$ | $m \subseteq\#_z$ *replicate_zmset p a* |
repeat_zmset p m $\subseteq\#_z n$ | $m \subseteq\#_z$ *repeat_zmset p m*) =
⟨*fn phi => ZMultiset_Simprocs.subseteq_cancel_zmsets*⟩

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

simproc-setup *zmsetless_cancel*
((l::'*a*::*preorder zmultiset*) + $m < n$ | (l::'*a zmultiset*) $< m + n$ |
add_zmset a m $< n$ | $m <$ *add_zmset a n* |
replicate_zmset p a $< n$ | $m <$ *replicate_zmset p a* |
repeat_zmset p m $< n$ | $m <$ *repeat_zmset p m*) =
⟨*fn phi => Cancel_Simprocs.less_cancel*⟩

simproc-setup *zmsetless_eq_cancel*
((l::'*a*::*preorder zmultiset*) + $m \leq n$ | (l::'*a zmultiset*) $\leq m + n$ |
add_zmset a m $\leq n$ | $m \leq$ *add_zmset a n* |
replicate_zmset p a $\leq n$ | $m \leq$ *replicate_zmset p a* |
repeat_zmset p m $\leq n$ | $m \leq$ *repeat_zmset p m*) =
⟨*fn phi => Cancel_Simprocs.less_eq_cancel*⟩

simproc-setup *zmsetdiff_cancel*
(n + (l::'*a zmultiset*) | (l::'*a zmultiset*) − m |
add_zmset a m − n | m − *add_zmset a n* |
replicate_zmset p r − n | m − *replicate_zmset p r* |
repeat_zmset p m − n | m − *repeat_zmset p m*) =
⟨*fn phi => Cancel_Simprocs.diff_cancel*⟩

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 $M \leq N$
obtains A B C **where**
$M = zmset\_of\ A + C$ **and** $N = zmset\_of\ B + C$ **and** $A \leq B$
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*: $M \subseteq\#_z N \Longrightarrow M \leq N$
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*: $M \subset\#_z N \Longrightarrow M < N$
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 $\exists y.\ zcount\ M\ y < zcount\ N\ y$
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

instance *zmultiset* :: (*preorder*) *no_top*
proof
fix M :: ‹'*a zmultiset*›
obtain *a* :: '*a* **where** *True* **by** *fast*
let *?M* = ‹*zmset_of* (*mset_pos M*) + *zmset_of* (*mset_neg M*)›
have ‹$M <$ *add_zmset a ?M* + *?M*›
by (*subst mset_pos_neg_partition*)
(*auto simp*: *subset_zmset_def subseteq_zmset_def zmultiset_eq_iff*
intro!: *subset_imp_less_zmset*)
then show ‹$\exists N.\ M < N$›
by *blast*
qed

end

4 Nested Multisets

theory *Nested_Multiset*
imports *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* :: '*a nmultiset* ⇒ *bool* **where**
$(\bigwedge X.\ X \in\#\ M \Longrightarrow no\_elem\ X) \Longrightarrow no\_elem\ (MSet\ M)$

inductive-set *sub_nmset* :: ('*a nmultiset* × '*a nmultiset*) *set* **where**
$X \in\#\ M \Longrightarrow (X,\ MSet\ M) \in sub\_nmset$

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

primrec *depth_nmset* :: '*a nmultiset* ⇒ *nat* (|_|) **where**

```
  |Elem a| = 0
| |MSet M| = (let X = set_mset (image_mset depth_nmset M) in if X = {} then 0 else Suc (Max X))

lemma depth_nmset_MSet: x ∈# M ⟹ |x| < |MSet M|
  by (auto simp: less_Suc_eq_le)

declare depth_nmset.simps(2)[simp del]
```

4.2 Dershowitz and Manna's Nested Multiset Order

The Dershowitz–Manna extension:

```
definition less_multiset_ext_DM :: ('a ⇒ 'a ⇒ bool) ⇒ 'a multiset ⇒ 'a multiset ⇒ bool where
  less_multiset_ext_DM R M N ⟷
   (∃X Y. X ≠ {#} ∧ X ⊆# N ∧ M = (N − X) + Y ∧ (∀k. k ∈# Y ⟶ (∃a. a ∈# X ∧ 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 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 sub_nmset <*lex*> sub_nmset, fastforce,
    metis sub_nmset.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: X < Y ⟹ Y < Z ⟹ X < Z for X Y Z :: 'a nmultiset
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 ∪ set_mset N ∪ set_mset N'
    assume XYZ: X = MSet M Y = MSet N Z = MSet N'
    then have trans: ∀x ∈ A. ∀y ∈ A. ∀z ∈ A. x < y ⟶ y < z ⟶ x < z
      by (auto elim!: less(1)[rotated −1] dest!: depth_nmset_MSet simp add: A_def)
    have set_mset M ⊆ A set_mset N ⊆ A set_mset N' ⊆ 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 ⟹ 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 ∈# X ∧ y ∈# Y ∧ 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 ≠ {#} X ⊆# M M = M − X + Y
    then have X = Y
      by (auto simp: mset_subset_eq_exists_conv)
    with MSet(1) ‹X ⊆# 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 ∀k. k ∈# Y ⟶ (∃a. a ∈# X ∧ k < a)
    with ‹X = Y› ‹X ≠ {#}› have ¬ 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 ⟹ Y < X ⟹ False
  using trans_less_nmultiset irrefl_less_nmultiset by blast

definition less_eq_nmultiset :: 'a nmultiset ⇒ 'a nmultiset ⇒ bool where
  less_eq_nmultiset X Y = (X < Y ∨ 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 (<) = (<)
  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_nmset_imp_less_nmultiset*: $|X| < |Y| \Longrightarrow X < Y$
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_nmset.simps*(2) *less_multiset_ext*$_{DM}$*_def not_le Max_gr_iff*
        intro: *exI*[*of _ N*] *split*: *if_splits*)
  qed (*auto simp*: *depth_nmset.simps*(2) *less_multiset_ext*$_{DM}$*_less split*: *if_splits*)
qed *simp_all*

lemma *less_nmultiset_imp_le_depth_nmset*: $X < Y \Longrightarrow |X| \leq |Y|$
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_nmset.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*. $|X| \leq |Y|$
        by (*meson ex_gt_imp_less_multiset less_asym' less_depth_nmset_imp_less_nmultiset*
          not_le_imp_less)
    qed
  qed (*auto simp*: *depth_nmset.simps*(2))
qed *simp_all*

lemma *eq_mlex_I*:
  fixes $f :: 'a \Rightarrow nat$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes $\bigwedge X\ Y.\ f\ X < f\ Y \Longrightarrow R\ X\ Y$ **and** *antisymp R*
  shows $\{(X,\ Y).\ R\ X\ Y\} = f\ <*mlex*>\ \{(X,\ Y).\ f\ X = f\ Y \wedge R\ X\ Y\}$
proof *safe*
  fix *X Y*
  assume *R X Y*
  show $(X,\ Y) \in f\ <*mlex*>\ \{(X,\ Y).\ f\ X = f\ Y \wedge R\ X\ Y\}$
  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 *antisymp_def* **by** *auto*
  qed
next
  fix *X Y*
  assume $(X,\ Y) \in f\ <*mlex*>\ \{(X,\ Y).\ f\ X = f\ Y \wedge R\ X\ Y\}$
  then show *R X Y*
    unfolding *mlex_prod_def* **by** (*auto simp*: *assms*(*1*))
qed

instantiation *nmultiset* :: (*wellorder*) *wellorder*
begin

lemma *depth_nmset_eq_0*[*simp*]: $|X| = 0 \longleftrightarrow (X = MSet\ \{\#\} \vee (\exists x.\ X = Elem\ x))$
by (*cases X*; *simp add*: *depth_nmset.simps*(2))

lemma *depth_nmset_eq_Suc*[*simp*]: $|X| = Suc\ n \longleftrightarrow$
$(\exists N.\ X = MSet\ N \wedge (\exists Y \in\# N.\ |Y| = n) \wedge (\forall Y \in\# N.\ |Y| \leq n))$
by (*cases X*; *auto simp add*: *depth_nmset.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 $\{(X :: 'a\ nmultiset,\ Y).\ |X| = i \wedge |Y| = i \wedge X < Y\}$
proof (*induct i rule*: *less_induct*)
case (*less i*)
define $A :: 'a\ nmultiset\ set$ **where** $A = \{X.\ |X| < i\}$
from *less* **have** *wf* $((depth\_nmset :: 'a\ nmultiset \Rightarrow nat)\ <*mlex*>$
$(\bigcup j < i.\ \{(X,\ Y).\ |X| = j \wedge |Y| = j \wedge X < Y\}))$
by (*intro wf_UN wf_mlex*) *auto*
then have $*$: *wf* $(mult\ \{(X :: 'a\ nmultiset,\ Y).\ X \in A \wedge Y \in A \wedge X < Y\})$
by (*intro wf_mult*, *elim wf_subset*) (*force simp*: *A_def mlex_prod_def not_less_iff_gr_or_eq*
dest!: *less_depth_nmset_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* $\lambda x.\ \{(x,\ MSet\ \{\#\})\}$]]])
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_nmset* $<*mlex*>$ $\{(X,\ Y).\ |X| = |Y| \wedge X < Y\}$
by (*rule eq_mlex_I*) (*auto simp*: *antisymp_def less_depth_nmset_imp_less_nmultiset*)
also have $\{(X,\ Y).\ |X| = |Y| \wedge X < Y\} = (\bigcup i.\ \{(X,\ Y).\ |X| = i \wedge |Y| = i \wedge X < Y\})$
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** (<) .
lift-definition *less_ eq_ hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** (≤) .

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

end

lemma *less_ HMSet_ iff_ less_ multiset_ ext*$_{DM}$: *HMSet M* < *HMSet N* ⟷ *less_ multiset_ ext*$_{DM}$ (<) *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
  ∃X' Y'. X' ≠ {#} ∧ X' ⊆# N ∧ N − X + Y = N − X' + Y' ∧
   (∀k. k ∈# Y' ⟶ (∃a. a ∈# X' ∧ 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 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])

lemma mem_imp_less_HMSet: k ∈# L ⟹ k < HMSet L
 by (induct k arbitrary: L) (auto intro: ex_gt_imp_less_multiset)

lemma mem_hmsetmset_imp_less: M ∈# hmsetmset N ⟹ M < N
 using mem_imp_less_HMSet by force
```

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 hmsetmset_0[simp]: hmsetmset 0 = {#}
 by 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*)

simproc-setup *hmseteq_ cancel*
((*l*::*hmultiset*) + *m* = *n* | (*l*::*hmultiset*) = *m* + *n*) =
‹*fn phi* => *Cancel_ Simprocs.eq_ cancel*›

simproc-setup *hmsetdiff_ cancel*
(((*l*::*hmultiset*) + *m*) − *n* | (*l*::*hmultiset*) − (*m* + *n*)) =
‹*fn phi* => *Cancel_ Simprocs.diff_ cancel*›

simproc-setup *hmsetless_ cancel*
((*l*::*hmultiset*) + *m* < *n* | (*l*::*hmultiset*) < *m* + *n*) =
‹*fn phi* => *Cancel_ Simprocs.less_ cancel*›

simproc-setup *hmsetless_ eq_ cancel*
((*l*::*hmultiset*) + *m* ≤ *n* | (*l*::*hmultiset*) ≤ *m* + *n*) =
‹*fn phi* => *Cancel_ Simprocs.less_ eq_ cancel*›

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*)

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 {#0#}
    by (simp add: zero_hmultiset_def)
  thus ?case
    by (rule exI)
qed


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_mset)
  }
  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
```

5.5 Inequalities

```
lemma zero_le_hmset[simp]: 0 ≤ M for M :: hmultiset
  by (simp add: order_le_less) (metis hmsetmset_less le_multiset_empty_left hmsetmset_empty_iff)

lemma
  le_add1_hmset: n ≤ n + m and
  le_add2_hmset: n ≤ m + n for n :: hmultiset
  by simp+

lemma le_zero_eq_hmset[simp]: M ≤ 0 ⟷ M = 0 for M :: hmultiset
  by (simp add: dual_order.antisym)

lemma not_less_zero_hmset[simp]: ¬ 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 *zero_less_HMSet_iff*[*simp*]: $0 < HMSet\ M \longleftrightarrow M \neq \{\#\}$
by (*simp only*: *zero_less_iff_neq_zero_hmset HMSet_eq_0_iff*)

lemma *gr_zeroI_hmset*: $(M = 0 \Longrightarrow False) \Longrightarrow 0 < M$ **for** M :: *hmultiset*
using *not_gr_zero_hmset* **by** *blast*

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 *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 (*simp add*: *add.commute trans_less_add1_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 *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)

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** (<) .
lift-definition *less_eq_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *bool* **is** (≤) .

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*: *mult.assoc 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 HOL−Library.Product_Order HOL−Library.Extended_Nat*
begin

7.1 Natural (Hessenberg) Product

instantiation *hmultiset :: comm_semiring_1*
begin

abbreviation *ω_exp :: hmultiset ⇒ hmultiset* (*ω^*) **where**
ω^ ≡ λm. HMSet {#m#}

definition *one_hmultiset :: hmultiset* **where**
1 = ω^0

abbreviation *ω :: hmultiset* **where**
ω ≡ ω^1

definition *times_hmultiset :: hmultiset ⇒ hmultiset ⇒ hmultiset* **where**
*A * B = HMSet (image_mset (case_prod (+)) (hmsetmset A ×# hmsetmset B))*

lemma *hmsetmset_times*:
*hmsetmset (m * n) = image_mset (case_prod (+)) (hmsetmset m ×# 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]: ω^M * N = HMSet (image_mset ((+) M) (hmsetmset N))
  by (simp add: times_hmultiset_def Times_mset_single_left)

lemma singleton_times_right_hmset[simp]: N * ω^M = HMSet (image_mset ((+) 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 (+)
  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 ×# Z < N ×# 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 ×# Z
  let ?Y = Y ×# Z

  show ?thesis
    unfolding less_multiset_DM
  proof (intro exI conjI)
    show M ×# Z = N ×# 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:
  a < b ⟹ 0 < c ⟹ a * c < b * c for a b c :: hmultiset
  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_strict_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 mult_le_mono1_hmset: i ≤ j ⟹ i * k ≤ j * k for i j k :: hmultiset
  by (simp add: mult_right_mono)

lemma mult_le_mono2_hmset: i ≤ j ⟹ k * i ≤ k * j for i j k :: hmultiset
  by (simp add: mult_left_mono)

lemma mult_le_mono_hmset: i ≤ j ⟹ k ≤ l ⟹ i * k ≤ j * l for i j k l :: hmultiset
  by (simp add: mult_mono)

lemma less_iff_add1_le_hmset: m < n ⟷ m + 1 ≤ 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 ⟷ 1 ≤ n for n :: hmultiset
  by (rule less_iff_add1_le_hmset[of 0, simplified])

lemma less_add_1_iff_le_hmset: m < n + 1 ⟷ m ≤ 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 :: zero_less_one
  by intro_classes (simp add: zero_less_iff_neq_zero_hmset)

instance hmultiset :: linordered_semiring_1_strict
  by intro_classes

instance hmultiset :: bounded_lattice_bot
  by intro_classes

instance hmultiset :: linordered_nonzero_semiring
  by intro_classes simp

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 ⟷ M = 0 for M :: hmultiset
  by (simp add: less_iff_add1_le_hmset)

lemma zero_less_mult_iff_hmset[simp]: 0 < m * n ⟷ 0 < m ∧ 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 ≤ m * n ⟷ 1 ≤ m ∧ 1 ≤ 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
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 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*

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 less_multiset_ext*$_{DM}$*_less*)

lemma *of_nat_ne_ω*[*simp*]: *of_nat n* ≠ *ω*
by (*simp add*: *neq_iff*)

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

lemma *of_nat_times_ω_exp*: *of_nat n* ∗ *ω*^*m* = *HMSet* (*replicate_mset n m*)
by (*induct n*) (*simp_all add*: *hmsetmset_plus one_hmultiset_def*)

lemma *ω_exp_times_of_nat*: *ω*^*m* ∗ *of_nat n* = *HMSet* (*replicate_mset n m*)
using *of_nat_times_ω_exp* **by** *simp*

7.4 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*

7.5 Head Omega

definition *head_ω* :: *hmultiset* ⇒ *hmultiset* **where**
head_ω M = (*if M* = *0 then 0 else ω*^(*Max* (*set_mset* (*hmsetmset M*))))

lemma *head_ω_subseteq*: *hmsetmset* (*head_ω M*) ⊆# *hmsetmset M*
unfolding *head_ω_def* **by** *simp*

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*
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* **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*,
      metis Max_ in dual_ order.antisym finite_ set_ mset le_ in_ le_ max le_ less 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_ mset*, *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_ mset M* < *Max_ 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
```

7.6 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 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: 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*]: $\omega - of\_nat\ n = \omega$
using *times_ω_minus_of_nat*[*of 1*] **by** (*metis mult.left_neutral*)

lemma *ω_minus_1*[*simp*]: $\omega - 1 = \omega$
using *ω_minus_of_nat*[*of 1*] **by** *simp*

lemma *ω_minus_numeral*[*simp*]: $\omega - numeral\ n = \omega$
using *times_ω_minus_numeral*[*of 1*] **by** (*metis mult.left_neutral*)

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*: $of\_nat\ m < hmset\_of\_enat\ n \longleftrightarrow enat\ m < n$
by (*metis hmset_of_enat.simps*(*1*) *hmset_of_enat_less*)

lemma *of_nat_le_hmset_of_enat_iff*: $of\_nat\ m \leq hmset\_of\_enat\ n \longleftrightarrow enat\ m \leq n$
by (*metis hmset_of_enat.simps*(*1*) *hmset_of_enat_le*)

lemma *hmset_of_enat_lt_iff_ne_infinity*: $hmset\_of\_enat\ x < \omega \longleftrightarrow x \neq \infty$
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*: $\bigwedge h\ ha :: hmultiset.\ h - (ha + h) = 0$
by (*simp add*: *add.commute*)
have *f2*: $\bigwedge h\ ha\ hb :: hmultiset.\ h + ha - (h - hb) = hb + ha - (hb - h)$
by (*metis* (*no_types*) *add_diff_cancel_right minus_diff_sym_hmset*)
have $\bigwedge h\ ha\ hb :: hmultiset.\ h + (ha + hb) - hb = h + ha$
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*: $m < \omega \Longrightarrow n < \omega \Longrightarrow (m - n) * l = m * l - n * l$
by (*metis lt_ω_imp_ex_of_nat times_of_nat_minus_left*)

lemma *lt_ω_imp_times_minus_right*: $m < \omega \Longrightarrow n < \omega \Longrightarrow l * (m - n) = l * m - l * n$
by (*metis lt_ω_imp_ex_of_nat times_of_nat_minus_right*)

lemma *hmset_pair_decompose*:
$\exists k\ n1\ n2.\ m1 = k + n1 \wedge m2 = k + n2 \wedge (head\_\omega\ n1 \neq head\_\omega\ n2 \vee n1 = 0 \wedge n2 = 0)$
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)
```

7.7 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])
```

7.8 An Example

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

```
lemma ludwig_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
    β1: β1 = β0 + β1a and
    β2: β2 = β0 + β2a 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 + β2a * γ < α1 + β1a * γ
    by (metis add_less_cancel_right semiring_normalization_rules(23))

  have α2 + β2 * δ = α2 + β0 * δ + β2a * δ
    unfolding β2 by (simp add: ab_semigroup_add_class.add_ac(1) distrib_right)
  also have ... = α2 + β0 * δ + β2a * η + β2a * δa
    unfolding δ by (simp add: distrib_left semiring_normalization_rules(25))
  also have ... ≤ α2 + β0 * δ + β2a * η + β2a * δa + β2a * γa
    by simp
  also have ... = α2 + β2a * γ + β0 * δ + β2a * δa
    unfolding γ distrib_left add.assoc[symmetric] by (simp add: semiring_normalization_rules(23))
  also have ... < α1 + β1a * γ + β0 * δ + β2a * δa
    using * by simp
  also have ... = α1 + β1a * η + β1a * γa + β0 * η + β0 * δa + β2a * δa
    unfolding γ δ distrib_left add.assoc[symmetric] by (rule refl)
  also have ... ≤ α1 + β1a * η + β0 * η + β0 * δa + β1a * δa
  proof −
    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
    thus ?thesis
      by simp
  qed
  finally show ?thesis
    unfolding β1 δ
    by (simp add: distrib_left distrib_right add.assoc[symmetric] semiring_normalization_rules(23))
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

abbreviation ω_z_exp :: hmultiset ⇒ zhmultiset (ω_z ^) where
  ω_z ^ ≡ λm. ZHMSet {#m#}_z
```

```
lift-definition one_zhmultiset :: zhmultiset is {#0#}z .

abbreviation ωz :: zhmultiset where
  ωz ≡ ωz ^1

lemma ωz_as_ω: ωz = zhmset_of ω
  by simp

lift-definition times_zhmultiset :: zhmultiset ⇒ zhmultiset ⇒ zhmultiset is
  λM 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 hmset_pos_plus hmset_neg_plus)
     (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.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 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*

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** (*simp add*: *of_nat_diff*)

lemma *of_nat_lt_ω_z*[*simp*]: *of_nat n* < *ω_z*
unfolding *ω_z_as_ω* **using** *of_nat_lt_ω of_nat_zhmset zhmset_of_less* **by** *presburger*

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*)

8.3 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*

8.4 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 (simp add: less_eq_zhmultiset.rep_eq mset_pos_supset subset_eq_imp_le_zmset)

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*: $0 < n \longleftrightarrow 1 \leq n$ **for** n :: *zhmultiset*
by (*rule less_iff_add1_le_zhmset*[*of* 0, *simplified*])

lemma *less_add_1_iff_le_hmset*: $m < n + 1 \longleftrightarrow m \leq n$ **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: $0 \leq x$ **and** y: $0 < y$ **and** z: $x \leq z$
shows $x \leq y * z$
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*]: ¬ $\omega_z \leq$ *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*)

8.5 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 α1* + *?z β1* * *?z γ* + *?z β2* * (*?z δ* − *?z γ*)
using *αβ2γ_lt_αβ1γ′* **by** (*simp add*: *algebra_simps*)
also have ... ≤ *?z α1* + *?z β1* * *?z γ* + *?z β1* * (*?z δ* − *?z γ*)
using *β2_le_β1′ γ_lt_δ′* **by** *simp*
finally show *?thesis*
by (*simp add*: *zmset_of_less zhmset_of_times*[*symmetric*] *zhmset_of_plus*[*symmetric*] *algebra_simps*)
qed

end

theory *Syntactic_Ordinal_Bridge*
imports *HOL−Library.Sublist Ordinal.OrdinalOmega Syntactic_Ordinal*
abbrevs
$!h = _h$
begin

9 Bridge between Huffman's Ordinal Library and the Syntactic Ordinals

9.1 Missing Lemmas about Huffman's Ordinals

instantiation *ordinal* :: *order_bot*
begin

definition *bot_ordinal* :: *ordinal* **where**
bot_ordinal = *0*

instance
by *intro_classes* (*simp add*: *bot_ordinal_def*)

end

lemma *insort_bot*[*simp*]: *insort bot xs* = *bot* # *xs* **for** *xs* :: *'a*::{*order_bot*,*linorder*} *list*
by (*simp add*: *insort_is_Cons*)

lemmas *insort_0_ordinal*[*simp*] = *insort_bot*[*of xs* :: *ordinal list* **for** *xs*, *unfolded bot_ordinal_def*]

lemma *from_cnf_less_ω_exp*:
assumes ∀ *k* ∈ *set ks*. *k* < *l*
shows *from_cnf ks* < *ω* ** *l*
using *assms* **by** (*induct ks*) (*auto simp*: *additive_principal.sum_less additive_principal_omega_exp*)

lemma *from_cnf_0_iff*[*simp*]: *from_cnf ks* = *0* ⟷ *ks* = []
by (*induct ks*) (*auto simp*: *ordinal_plus_not_0*)

lemma *from_cnf_append*[*simp*]: *from_cnf* (*ks* @ *ls*) = *from_cnf ks* + *from_cnf ls*
by (*induct ks*) (*auto simp*: *ordinal_plus_assoc*)

lemma *subseq_from_cnf_less_eq*: *Sublist.subseq ks ls* ⟹ *from_cnf ks* ≤ *from_cnf ls*
by (*induct rule*: *list_emb.induct*) (*auto intro*: *ordinal_le_plusL order_trans*)

9.2 Embedding of Syntactic Ordinals into Huffman's Ordinals

abbreviation ω_h :: *hmultiset* **where**
ω_h ≡ *Syntactic_Ordinal.ω*

abbreviation ω_h*_exp* :: *hmultiset* ⇒ *hmultiset* (ω_h ^) **where**
ω_h ^ ≡ *Syntactic_Ordinal.ω_exp*

primrec *ordinal_of_hmset* :: *hmultiset* ⇒ *ordinal* **where**
ordinal_of_hmset (*HMSet M*) =
from_cnf (*rev* (*sorted_list_of_multiset* (*image_mset ordinal_of_hmset M*)))

lemma *ordinal_of_hmset_0*[*simp*]: *ordinal_of_hmset 0* = *0*
unfolding *zero_hmultiset_def* **by** *simp*

lemma *ordinal_of_hmset_suc*[*simp*]: *ordinal_of_hmset* (*k* + *1*) = *ordinal_of_hmset k* + *1*
unfolding *plus_hmultiset_def one_hmultiset_def* **by** (*cases k*) *simp*

lemma *ordinal_of_hmset_1*[*simp*]: *ordinal_of_hmset 1* = *1*

```
  using ordinal_of_hmset_suc[of 0] by simp

lemma ordinal_of_hmset_ω[simp]: ordinal_of_hmset ω_h = ω
  by simp

lemma ordinal_of_hmset_singleton[simp]: ordinal_of_hmset (ω ^ k) = ω ** ordinal_of_hmset k
  by simp

lemma ordinal_of_hmset_iff[simp]: ordinal_of_hmset k = 0 ⟷ k = 0
  by (induct k) auto

lemma less_imp_ordinal_of_hmset_less: k < l ⟹ ordinal_of_hmset k < ordinal_of_hmset l
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(k, l). {#k, l#}
        λ(k, l). k < l ⟶ ordinal_of_hmset k < ordinal_of_hmset l (k, l),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix k l
  assume
    ih: ⋀ka la. {#ka, la#} < {#k, l#} ⟹ ka < la ⟹ ordinal_of_hmset ka < ordinal_of_hmset la and
    k_lt_l: k < l

  show ordinal_of_hmset k < ordinal_of_hmset l
  proof (cases k = 0)
    case True
    thus ?thesis
      using k_lt_l ordinal_neq_0 by fastforce
  next
    case k_nz: False

    have l_nz: l ≠ 0
      using k_lt_l by auto

    define K where K: K = hmsetmset k
    define L where L: L = hmsetmset l

    have k: k = HMSet K and l: l = HMSet L
      by (simp_all add: K L)

    have K_lt_L: K < L
      unfolding K L using k_lt_l by simp

    define x where x: x = Max_mset K
    define Ka where Ka: Ka = K - {#x#}

    have k_eq_xKa: k = HMSet (add_mset x Ka)
      using K x Ka k_nz by auto
    have x_max: ∀ a ∈# Ka. a ≤ x
      unfolding x Ka by (meson Max_ge finite_set_mset in_diffD)

    have ord_x_max: ∀ a ∈# Ka. ordinal_of_hmset a ≤ ordinal_of_hmset x
    proof
      fix a
      assume a_in: a ∈# Ka

      have a_le_x: a ≤ x
        by (simp add: x_max a_in)
      moreover
      {
        assume a_lt_x: a < x
        moreover have x_lt_k: x < k
          unfolding k_eq_xKa by (rule mem_imp_less_HMSet) simp
        ultimately have a_lt_k: a < k
```

```
    by simp

  have {#a, x#} < {#k#}
    using x_lt_k a_lt_k by simp
  also have ... < {#k, l#}
    unfolding k_eq_xKa using a_in
    by simp
  finally have ordinal_of_hmset a < ordinal_of_hmset x
    by (rule ih[OF _ a_lt_x])
 }
 ultimately show ordinal_of_hmset a ≤ ordinal_of_hmset x
  by force
qed

define y where y: y = Max_mset L
define La where La: La = L − {#y#}

have l_eq_yLa: l = HMSet (add_mset y La)
 using L y La l_nz by auto
have y_max: ∀ b ∈# La. b ≤ y
 unfolding y La by (meson Max_ge finite_set_mset in_diffD)

have ord_y_max: ∀ b ∈# La. ordinal_of_hmset b ≤ ordinal_of_hmset y
proof
 fix b
 assume b_in: b ∈# La

 have b_le_y: b ≤ y
  by (simp add: y_max b_in)
 moreover
 {
  assume b_lt_y: b < y
  moreover have y_lt_l: y < l
   unfolding l_eq_yLa by (rule mem_imp_less_HMSet) simp
  ultimately have b_lt_l: b < l
   by simp

  have {#b, y#} < {#l#}
   using y_lt_l b_lt_l by simp
  also have ... < {#k, l#}
   unfolding l_eq_yLa using b_in
   by simp
  finally have ordinal_of_hmset b < ordinal_of_hmset y
   by (rule ih[OF _ b_lt_y])
 }
 ultimately show ordinal_of_hmset b ≤ ordinal_of_hmset y
  by force
qed

{
 assume x_eq_y: x = y

 have ordinal_of_hmset (HMSet Ka) < ordinal_of_hmset (HMSet La)
 proof (rule ih)
  show {#HMSet Ka, HMSet La#} < {#k, l#}
   unfolding k l
   by (metis add_mset_add_single hmsetmset_less hmultiset.sel k k_eq_xKa l l_eq_yLa
     le_multiset_right_total mset_lt_single_iff union_less_mono)
 next
  have ω^x + HMSet Ka < ω^y + HMSet La
   using k_lt_l[unfolded k_eq_xKa l_eq_yLa]
   by (metis HMSet_plus add.commute add_mset_add_single)
  thus HMSet Ka < HMSet La
```

```
      using x_eq_y by simp
    qed
    hence ?thesis
      unfolding k_eq_xKa l_eq_yLa
      by (simp, subst (1 2) sorted_insort_is_snoc, simp_all add: ord_x_max ord_y_max,
        force simp: x_eq_y)
  }
  moreover
  {
    assume x_ne_y: x ≠ y

    have x_lt_y: x < y
      by (metis K L head_ω_def head_ω_lt_imp_lt hmsetmset_less hmultiset.sel k_lt_l k_nz l_nz
        less_imp_not_less mset_lt_single_iff neqE x x_ne_y y)

    have ord_y_smax_K: ordinal_of_hmset a < ordinal_of_hmset y if a_in_K: a ∈# K for a
    proof (rule ih)
      show {#a, y#} < {#k, l#}
        unfolding k_eq_xKa l_eq_yLa using a_in_K k k_eq_xKa
        by (metis add_mset_add_single mem_imp_less_HMSet mset_lt_single_iff union_less_mono
          union_single_eq_member)
    next
      show a < y
        by (metis Max_ge finite_set_mset less_le_trans not_less_iff_gr_or_eq that x x_lt_y)
    qed

    have ordinal_of_hmset k < ordinal_of_hmset (ω^y)
    proof (cases La)
      case empty
      show ?thesis
        unfolding k by (auto intro!: from_cnf_less_ω_exp simp: ord_y_smax_K)
    next
      case La: (add ya Lb)
      show ?thesis
      proof (rule ih)
        show {#k, ω^y#} < {#k, l#}
          unfolding l_eq_yLa La by simp
      next
        show k < ω^y
        proof −
          have ⋀m. x < Max_mset (add_mset y m)
            by (meson Max_ge finite_set_mset less_le_trans union_single_eq_member x_lt_y)
          then show ?thesis
            by (metis K x head_ω_def head_ω_lt_imp_lt hmsetmset_less hmultiset.sel k_nz
              mset_lt_single_iff x_lt_y)
        qed
      qed
    qed
    also have ... ≤ ordinal_of_hmset l
      unfolding l_eq_yLa
      by (auto simp del: from_cnf.simps intro!: subseq_from_cnf_less_eq
        simp: subseq_from_cnf_less_eq sorted_insort_is_snoc ord_y_max)
    ultimately have ?thesis
      by simp
  }
  ultimately show ?thesis
    by sat
 qed
qed

lemma ordinal_of_hmset_less[simp]: ordinal_of_hmset k < ordinal_of_hmset l ⟷ k < l
  using less_imp_not_less less_imp_ordinal_of_hmset_less neq_iff by blast
```

end

10 Termination of McCarthy's 91 Function

theory *McCarthy_91*
imports *HOL−Library.Multiset_Order*
begin

lemma *funpow_rec*: $f \hat{}\hat{} n = (\mathit{if}\ n = 0\ \mathit{then}\ \mathit{id}\ \mathit{else}\ f \circ f \hat{}\hat{} (n - 1))$
by (*induct n*) *auto*

The f function captures the semantics of McCarthy's 91 function. The g function is a tail-recursive implementation of the function, whose termination is established using the multiset order. The definitions follow Dershowitz and Manna.

definition $f :: \mathit{int} \Rightarrow \mathit{int}$ **where**
$f\ x = (\mathit{if}\ x > 100\ \mathit{then}\ x - 10\ \mathit{else}\ 91)$

definition $\tau :: \mathit{nat} \Rightarrow \mathit{int} \Rightarrow \mathit{int}\ \mathit{multiset}$ **where**
$\tau\ n\ z = \mathit{mset}\ (\mathit{map}\ (\lambda i.\ (f \hat{}\hat{} \mathit{nat}\ i)\ z)\ [0..\mathit{int}\ n - 1])$

function $g :: \mathit{nat} \Rightarrow \mathit{int} \Rightarrow \mathit{int}$ **where**
$g\ n\ z = (\mathit{if}\ n = 0\ \mathit{then}\ z\ \mathit{else}\ \mathit{if}\ z > 100\ \mathit{then}\ g\ (n - 1)\ (z - 10)\ \mathit{else}\ g\ (n + 1)\ (z + 11))$
by *pat_completeness auto*
termination
proof −
define $\mathit{lt} :: (\mathit{int} \times \mathit{int})\ \mathit{set}$ **where**
$\mathit{lt} = \{(a, b).\ b < a \wedge a \leq 111\}$

have *lt_trans*: *trans lt*
unfolding *trans_def lt_def* **by** *simp*
have *lt_irrefl*: *irrefl lt*
unfolding *irrefl_def lt_def* **by** *simp*

let *?LT* = *mult lt*
let *?T* = $\lambda(n, z).\ \tau\ n\ z$
let *?R* = *inv_image ?LT ?T*

show *?thesis*
proof (*relation ?R*)
show *wf ?R*
by (*auto simp*: *lt_def intro*!: *wf_inv_image*[*OF wf_mult*]
wf_subset[*OF wf_measure*[*of* $\lambda z.\ \mathit{nat}\ (111 - z)$]])
next
fix $n :: \mathit{nat}$ **and** $z :: \mathit{int}$
assume *n_ne_0*: $n \neq 0$

{
assume *z_gt_100*: $z > 100$

have $\mathit{map}\ (\lambda i.\ (f \hat{}\hat{} \mathit{nat}\ i)\ (z - 10))\ [0..\mathit{int}\ n - 2] =$
$\mathit{map}\ (\lambda i.\ (f \hat{}\hat{} \mathit{nat}\ i)\ z)\ [1..\mathit{int}\ n - 1]$
using *n_ne_0*
proof (*induct n rule*: *less_induct*)
case (*less n*)
note *ih* = *this*(1) **and** *n_ne_0* = *this*(2)
show *?case*
proof (*cases* $n = 1$)
case *True*
thus *?thesis*
by *simp*
next
case *False*

```
    hence n_ge_2: n ≥ 2
      using n_ne_0 by simp

    have
      split_l: [0..int n − 2] = [0..int (n − 1) − 2] @ [int n − 2] and
      split_r: [1..int n − 1] = [1..int (n − 1) − 1] @ [int n − 1]
      using n_ge_2 by (induct n) (auto simp: upto_rec2)
    have f_repeat: (f ^^ (n − 2)) (z − 10) = (f ^^ (n − 1)) z
      using z_gt_100 n_ge_2 by (induct n, simp) (rename_tac m; case_tac m; simp add: f_def)+

    show ?thesis
      using n_ge_2 by (auto intro!: ih simp: split_l split_r f_repeat nat_diff_distrib')
  qed
 qed
 hence image_mset_eq: {#(f ^^ nat i) (z − 10). i ∈# mset [0..int n − 2]#} =
  {#(f ^^ nat i) z. i ∈# mset [1..int n − 1]#}
  by (fold mset_map) (intro arg_cong[of _ _ mset])

 have mset_eq_add_0_mset: mset [0..int n − 1] = add_mset 0 (mset [1..int n − 1])
  using n_ne_0 by (induct n) (auto simp: upto.simps)

 have nm1m1: int (n − 1) − 1 = int n − 2
  using n_ne_0 by simp

 show ((n − 1, z − 10), (n, z)) ∈ ?R
  by (auto simp: image_mset_eq mset_eq_add_0_mset nm1m1 τ_def simp del: One_nat_def
    intro: subset_implies_mult image_mset_subset_mono)
}
{
 assume z_le_100: ¬ z > 100

 have map_eq: map (λx. (f ^^ nat x) (z + 11)) [2..int n] =
  map (λi. (f ^^ nat i) z) [1..int n − 1]
  using n_ne_0
 proof (induct n rule: less_induct)
  case (less n)
  note ih = this(1) and n_ne_0 = this(2)
  show ?case
  proof (cases n = 1)
   case True
   thus ?thesis
    by simp
  next
   case False
   hence n_ge_2: n ≥ 2
    using n_ne_0 by simp

   have
    split_l: [2..int n] = [2..int (n − 1)] @ [int n] and
    split_r: [1..int n − 1] = [1..int (n − 1) − 1] @ [int n − 1]
    using n_ge_2 by (induct n) (auto simp: upto_rec2)
   from z_le_100 have f_f_z_11: f (f (z + 11)) = f z
    by (simp add: f_def)
   moreover define m where m = n − 2
   with n_ge_2 have n = m + 2
    by simp
   ultimately have f_repeat: (f ^^ n) (z + 11) = (f ^^ (n − 1)) z
    by (simp add: funpow_Suc_right del: funpow.simps)
   with n_ge_2 show ?thesis
    by (auto intro: ih [of nat (int n − 1)]
      simp: less.hyps split_l split_r nat_add_distrib nat_diff_distrib)
  qed
 qed
```

```
    have [0..int n] = [0..1] @ [2..int n]
      using n_ne_0 by (simp add: upto_rec1)
    hence {#(f ^^ nat x) (z + 11). x ∈# mset [0..int n]#} =
      {#(f ^^ nat x) (z + 11). x ∈# mset [0..1]#}
      + {#(f ^^ nat x) (z + 11). x ∈# mset [2..int n]#}
      by auto
    hence factor_out_first_two: {#(f ^^ nat x) (z + 11). x ∈# mset [0..int n]#} =
      {#z + 11, f (z + 11)#} + {#(f ^^ nat x) (z + 11). x ∈# mset [2..int n]#}
      by (auto simp: upto_rec1)

    let ?etc1 = {#(f ^^ nat i) (z + 11). i ∈# mset [2..int n]#}
    let ?etc2 = {#(f ^^ nat i) z. i ∈# mset [1..int n - 1]#}

    show ((n + 1, z + 11), (n, z)) ∈ ?R
    proof (cases z ≥ 90)
      case z_ge_90: True

      have {#z + 11, f (z + 11)#} + ?etc1 = {#z + 11, z + 1#} + ?etc2
        using z_ge_90
        by (auto intro!: arg_cong2[of _ _ _ _ add_mset] simp: map_eq f_def mset_map[symmetric]
          simp del: mset_map)
      hence image_mset_eq: {#(f ^^ nat x) (z + 11). x ∈# mset [0..int n]#} =
        {#z + 11, z + 1#} + ?etc2
        using factor_out_first_two by presburger

      have ({#z + 11, z + 1#}, {#z#}) ∈ mult1 lt
        using z_le_100 z_ge_90 by (auto intro!: mult1I simp: lt_def)
      hence ({#z + 11, z + 1#}, {#z#}) ∈ mult lt
        unfolding mult_def by simp
      hence ({#z + 11, z + 1#} + ?etc2, {#z#} + ?etc2) ∈ mult lt
        by (rule mult_cancel[THEN iffD2, OF lt_trans lt_irrefl])
      thus ?thesis
        using n_ne_0 by (auto simp: image_mset_eq τ_def upto_rec1[of 0 int n - 1])
    next
      case z_lt_90: False

      have {#z + 11, f (z + 11)#} + ?etc1 = {#z + 11, 91#} + ?etc2
        using z_lt_90
        by (auto intro!: arg_cong2[of _ _ _ _ add_mset] simp: map_eq f_def mset_map[symmetric]
          simp del: mset_map)
      hence image_mset_eq: {#(f ^^ nat x) (z + 11). x ∈# mset [0..int n]#} =
        {#z + 11, 91#} + ?etc2
        using factor_out_first_two by presburger

      have ({#z + 11, 91#}, {#z#}) ∈ mult1 lt
        using z_le_100 z_lt_90 by (auto intro!: mult1I simp: lt_def)
      hence ({#z + 11, 91#}, {#z#}) ∈ mult lt
        unfolding mult_def by simp
      hence ({#z + 11, 91#} + ?etc2, {#z#} + ?etc2) ∈ mult lt
        by (rule mult_cancel[THEN iffD2, OF lt_trans lt_irrefl])
      thus ?thesis
        using n_ne_0 by (auto simp: image_mset_eq τ_def upto_rec1[of 0 int n - 1])
    qed
  }
 qed
qed

declare g.simps [simp del]

end
```

11 Termination of the Hydra Battle

theory *Hydra_Battle*
imports *Syntactic_Ordinal*
begin

hide-const (**open**) *Nil Cons*

The h function and its auxiliaries f and d represent the hydra battle. The *encode* function converts a hydra (represented as a Lisp-like tree) to a syntactic ordinal. The definitions follow Dershowitz and Moser.

datatype *lisp* =
  Nil
| *Cons* (*car*: *lisp*) (*cdr*: *lisp*)
where
  car Nil = *Nil*
| *cdr Nil* = *Nil*

primrec *encode* :: *lisp* ⇒ *hmultiset* **where**
  encode Nil = *0*
| *encode* (*Cons l r*) = *ω*^(*encode l*) + *encode r*

primrec *f* :: *nat* ⇒ *lisp* ⇒ *lisp* ⇒ *lisp* **where**
  f 0 y x = *x*
| *f* (*Suc m*) *y x* = *Cons y* (*f m y x*)

lemma *encode_f*: *encode* (*f n y x*) = *of_nat n* ∗ *ω*^(*encode y*) + *encode x*
  unfolding *of_nat_times_ω_exp* **by** (*induct n*) (*auto simp*: *HMSet_plus*[*symmetric*])

function *d* :: *nat* ⇒ *lisp* ⇒ *lisp* **where**
  d n x =
   (*if car x* = *Nil then cdr x*
    else if car (*car x*) = *Nil then f n* (*cdr* (*car x*)) (*cdr x*)
    else Cons (*d n* (*car x*)) (*cdr x*))
  by *pat_completeness auto*
termination
  by (*relation measure* (λ(_, *x*). *size x*), *rule wf_measure*, *rename_tac n x*, *case_tac x*, *auto*)

declare *d.simps*[*simp del*]

function *h* :: *nat* ⇒ *lisp* ⇒ *lisp* **where**
  h n x = (*if x* = *Nil then Nil else h* (*n* + *1*) (*d n x*))
  by *pat_completeness auto*
termination
proof −
  let *?R* = *inv_image* {(*m*, *n*). *m* < *n*} (λ(*n*, *x*). *encode x*)

  show *?thesis*
  proof (*relation ?R*)
    show *wf ?R*
      by (*rule wf_inv_image*) (*rule wf*)
  next
    fix *n x*
    assume *x_cons*: *x* ≠ *Nil*
    thus ((*n* + *1*, *d n x*), *n*, *x*) ∈ *?R*
      unfolding *inv_image_def mem_Collect_eq prod.case*
    proof (*induct x*)
      case (*Cons l r*)
      note *ihl* = *this*(*1*)
      show *?case*
      proof (*subst d.simps*, *simp*, *intro conjI impI*)
        assume *l_cons*: *l* ≠ *Nil*
        {
          assume *car l* = *Nil*
          show *encode* (*f n* (*cdr l*) *r*) < *ω*^(*encode l*) + *encode r*

```
        using l_cons by (cases l) (auto simp: encode_f[unfolded of_nat_times_ω_exp])
      }
      {
        show encode (d n l) < encode l
          by (rule ihl[OF l_cons])
      }
    qed
  qed simp
 qed
qed

declare h.simps[simp del]

end
```

12 Termination of the Goodstein Sequence

```
theory Goodstein_Sequence
imports Multiset_More Syntactic_Ordinal
begin
```

The *goodstein* function returns the successive values of the Goodstein sequence. It is defined in terms of *encode* and *decode* functions, which convert between natural numbers and ordinals. The development culminates with a proof of Goodstein's theorem.

12.1 Lemmas about Division

```
lemma div_mult_le: m div n * n ≤ m for m n :: nat
  by (fact div_times_less_eq_dividend)

lemma power_div_same_base:
  b ^ y ≠ 0 ⟹ x ≥ y ⟹ b ^ x div b ^ y = b ^ (x - y) for b :: 'a::semidom_divide
  by (metis add_diff_inverse leD nonzero_mult_div_cancel_left power_add)
```

12.2 Hereditary and Nonhereditary Base-n Systems

```
context
  fixes base :: nat
  assumes base_ge_2: base ≥ 2
begin

inductive well_base :: 'a multiset ⇒ bool where
  (∀n. count M n < base) ⟹ well_base M

lemma well_base_filter: well_base M ⟹ well_base {#m ∈# M. p m#}
  by (auto simp: well_base.simps)

lemma well_base_image_inj: well_base M ⟹ inj_on f (set_mset M) ⟹ well_base (image_mset f M)
  unfolding well_base.simps by (metis count_image_mset_le_count_inj_on le_less_trans)

lemma well_base_bound:
  assumes
    well_base M and
    ∀m ∈# M. m < n
  shows (∑m ∈# M. base ^ m) < base ^ n
  using assms
proof (induct n arbitrary: M)
  case (Suc n)
  note ih = this(1) and well_M = this(2) and in_M_lt_Sn = this(3)

  let ?Meq = {#m ∈# M. m = n#}
  let ?Mne = {#m ∈# M. m ≠ n#}
  let ?K = {#base ^ m. m ∈# M#}
```

```
  have M: M = ?Meq + ?Mne
    by (simp add: multiset_partition)

  have well_Mne: well_base ?Mne
    by (rule well_base_filter[OF well_M])

  have in_Mne_lt_n: ∀ m ∈# ?Mne. m < n
    using in_M_lt_Sn by auto

  have sum_mset (image_mset ((^) base) ?Meq) ≤ (base − 1) * base ^ n
    unfolding filter_eq_replicate_mset using base_ge_2
    by simp (metis Suc_pred diff_self_eq_0 le_SucE less_imp_le less_le_trans less_numeral_extra(3)
      pos2 well_M well_base.cases zero_less_diff)
  moreover have base * base ^ n = base ^ n + (base − Suc 0) * base ^ n
    using base_ge_2 mult_eq_if by auto
  ultimately show ?case
    using ih[OF well_Mne in_Mne_lt_n] by (subst M) simp
qed simp

inductive well_base_h :: hmultiset ⇒ bool where
  (∀ N ∈# hmsetmset M. well_base_h N) ⟹ well_base (hmsetmset M) ⟹ well_base_h M

lemma well_base_h_mono_hmset: well_base_h M ⟹ hmsetmset N ⊆# hmsetmset M ⟹ well_base_h N
  by (induct rule: well_base_h.induct, rule well_base_h.intros, blast)
    (meson leD leI order_trans subseteq_mset_def well_base.simps)

lemma well_base_h_imp_well_base: well_base_h M ⟹ well_base (hmsetmset M)
  by (erule well_base_h.cases) simp
```

12.3 Encoding of Natural Numbers into Ordinals

```
function encode :: nat ⇒ nat ⇒ hmultiset where
  encode e n =
   (if n = 0 then 0 else of_nat (n mod base) * ω ^(encode 0 e) + encode (e + 1) (n div base))
  by pat_completeness auto
termination
  using base_ge_2
proof (relation measure (λ(e, n). n * (base ^ e + 1)); simp)
  fix e n :: nat
  assume n_ge_0: n > 0

  have e + e ≤ 2 ^ e
    by (induct e; simp) (metis add_diff_cancel_left' add_leD1 diff_is_0_eq' double_not_eq_Suc_double
      le_antisym mult_2 not_less_eq_eq power_eq_0_iff zero_neq_numeral)
  also have ... ≤ base ^ e
    using base_ge_2 by (simp add: power_mono)
  also have ... ≤ n * base ^ e
    using n_ge_0 by (simp add: Suc_leI)
  also have ... < n + n * base ^ e
    using n_ge_0 by simp
  finally show e + e < n + n * base ^ e
    by assumption

  have n div base * (base * base ^ e) ≤ n * base ^ e
    using base_ge_2 by (auto intro: div_mult_le)
  moreover have n div base < n
    using n_ge_0 base_ge_2 by simp
  ultimately show n div base + n div base * (base * base ^ e) < n + n * base ^ e
    by linarith
qed

declare encode.simps[simp del]
```

```
lemma encode_0[simp]: encode e 0 = 0
  by (subst encode.simps) simp

lemma encode_Suc:
  encode e (Suc n) = of_nat (Suc n mod base) * ω^(encode 0 e) + encode (e + 1) (Suc n div base)
  by (subst encode.simps) simp

lemma encode_0_iff: encode e n = 0 ⟷ n = 0
proof (induct n arbitrary: e rule: less_induct)
  case (less n)
  note ih = this

  show ?case
  proof (cases n)
    case 0
    thus ?thesis
      by simp
  next
    case n: (Suc m)
    show ?thesis
    proof (cases n mod base = 0)
      case True
      hence n div base ≠ 0
        using div_eq_0_iff n by fastforce
      thus ?thesis
        using ih[of Suc m div base] n
        by (simp add: encode_Suc) (metis One_nat_def base_ge_2 div_eq_dividend_iff div_le_dividend
          leD lessI nat_neq_iff numeral_2_eq_2)
    next
      case False
      thus ?thesis
        using n plus_hmultiset_def by (simp add: encode_Suc[unfolded of_nat_times_ω_exp])
    qed
  qed
qed

lemma encode_Suc_exp: encode (Suc e) n = encode e (base * n)
  using base_ge_2
  by (subst (1 2) encode.simps, subst (4) encode.simps, simp add: zero_hmultiset_def[symmetric])

lemma encode_exp_0: encode e n = encode 0 (base ^ e * n)
  by (induct e arbitrary: n) (simp_all add: encode_Suc_exp mult.assoc mult.commute)

lemma mem_hmsetmset_encodeD: M ∈# hmsetmset (encode e n) ⟹ ∃ e' ≥ e. M = encode 0 e'
proof (induct e n rule: encode.induct)
  case (1 e n)
  note ih = this(1-2) and M_in = this(3)

  show ?case
  proof (cases n)
    case 0
    thus ?thesis
      using M_in by simp
  next
    case n: (Suc m)

    {
      assume M ∈# replicate_mset (n mod base) (encode 0 e)
      hence ?thesis
        by (meson in_replicate_mset order_refl)
    }
    moreover
    {
```

```
    assume M ∈# hmsetmset (encode (e + 1) (n div base))
    hence ?thesis
      using ih(2) le_add1 n order_trans by blast
  }
  ultimately show ?thesis
    using M_in[unfolded n encode_Suc[unfolded of_nat_times_ω_exp], folded n]
    unfolding hmsetmset_plus by auto
 qed
qed

lemma less_imp_encode_less: n < p ⟹ encode e n < encode e p
proof (induct e n arbitrary: p rule: encode.induct)
 case (1 e n)
 note ih = this(1−2) and n_lt_p = this(3)

 show ?case
 proof (cases n = 0)
  case True
  thus ?thesis
    using n_lt_p base_ge_2 encode_0_iff[of e p] le_less by fastforce
 next
  case n_nz: False

  let ?Ma = replicate_mset (n mod base) (encode 0 e)
  let ?Na = replicate_mset (p mod base) (encode 0 e)
  let ?Pa = replicate_mset (n mod base − p mod base) (encode 0 e)

  have HMSet ?Ma + encode (Suc e) (n div base) < HMSet ?Na + encode (Suc e) (p div base)
  proof (cases n mod base < p mod base)
   case mod_lt: True
   show ?thesis
    by (rule add_less_le_mono, simp add: mod_lt,
      metis ih(2)[of p div base, OF n_nz] Suc_eq_plus1 div_le_mono le_less n_lt_p)
  next
   case mod_ge: False
   hence div_lt: n div base < p div base
    by (metis add_le_cancel_left div_le_mono div_mult_mod_eq le_neq_implies_less less_imp_le
      n_lt_p nat_neq_iff)

   let ?M = hmsetmset (encode (Suc e) (n div base))
   let ?N = hmsetmset (encode (Suc e) (p div base))

   have ?M < ?N
    by (auto intro!: ih(2)[folded Suc_eq_plus1] n_nz div_lt)
   then obtain X Y where
    X_nemp: X ≠ {#} and
    X_sub: X ⊆# ?N and
    M: ?M = ?N − X + Y and
    ex_gt: ∀ y. y ∈# Y ⟶ (∃ x. x ∈# X ∧ x > y)
    using less_multiset_DM by metis

   {
    fix x
    assume x_in_X: x ∈# X
    hence x_in_N: x ∈# ?N
     using X_sub by blast
    then obtain e' where
     e'_gt: e' > e and
     x: x = encode 0 e'
     by (auto simp: Suc_le_eq dest: mem_hmsetmset_encodeD)

    have x > encode 0 e
     unfolding x using ih(1)[OF n_nz] e'_gt by (blast dest: Suc_lessD)
```

```
    }
    hence ex_gt_e: ∃x ∈# X. x > encode 0 e
      using X_nemp by auto

    have X_sub': X ⊆# ?Na + ?N
      using X_sub by (simp add: subset_mset.add_increasing)
    have mam_eq: ?Ma + ?M = ?Na + ?N − X + (Y + ?Pa)
    proof −
      from mod_ge have ?Ma = ?Na + ?Pa
        by (simp add: replicate_mset_plus [symmetric])
      moreover have ?Na + ?N − X = ?Na + (?N − X)
        by (meson X_sub multiset_diff_union_assoc)
      ultimately show ?thesis
        by (simp add: M)
    qed
    have max_X: ⋀k. k ∈# Y + ?Pa ⟹ ∃a. a ∈# X ∧ k < a
      using ex_gt mod_ge ex_gt_e by (metis in_replicate_mset union_iff)

    show ?thesis
      by (subst (4 8) hmultiset.collapse[symmetric],
        unfold HMSet_plus[symmetric] HMSet_less less_multiset_DM,
        rule exI[of _ X], rule exI[of _ Y + ?Pa],
        intro conjI impI allI X_nemp X_sub' mam_eq, elim max_X)
  qed
  thus ?thesis
    using n_nz n_lt_p by (subst (1 2) encode.simps[unfolded of_nat_times_ω_exp]) auto
 qed
qed

inductive aligned_e :: nat ⇒ hmultiset ⇒ bool where
 (∀m ∈# hmsetmset M. m ≥ encode 0 e) ⟹ aligned_e e M

lemma aligned_e_encode: aligned_e e (encode e M)
 by (subst encode_exp_0, rule aligned_e.intros,
   metis encode_exp_0 leD leI lessI less_imp_encode_less lift_Suc_mono_less_iff
     mem_hmsetmset_encodeD)

lemma well_base_h_encode: well_base_h (encode e n)
proof (induct e n rule: encode.induct)
 case (1 e n)
 note ih = this

 have well2: ∀M ∈# hmsetmset (encode (Suc e) (n div base)). well_base_h M
   using ih(2) well_base_h.cases by (metis Suc_eq_plus1 Zero_not_Suc count_empty div_0
     encode_0_iff hmsetmset_empty_iff in_countE)

 have cnt1: count (hmsetmset (encode (Suc e) (n div base))) (encode 0 e) = 0
   using aligned_e_encode[unfolded aligned_e.simps]
     less_imp_encode_less[of n Suc n for n, simplified]
   by (meson count_inI leD)

 show ?case
 proof (rule well_base_h.intros)
   show ∀M ∈# hmsetmset (encode e n). well_base_h M
     by (subst encode.simps[unfolded of_nat_times_ω_exp],
       simp add: zero_hmultiset_def hmsetmset_plus, use ih(1) well2 in blast)
 next
   show well_base (hmsetmset (encode e n))
     using cnt1 base_ge_2
     by (subst encode.simps[unfolded of_nat_times_ω_exp],
       simp add: well_base.simps zero_hmultiset_def hmsetmset_plus,
       metis ih(2) well_base_h.simps Suc_eq_plus1 less_numeral_extra(3) well_base.simps)
 qed
```

qed

12.4 Decoding of Natural Numbers from Ordinals

primrec *decode* :: *nat* ⇒ *hmultiset* ⇒ *nat* **where**
decode e (*HMSet M*) = (∑ *m* ∈# *M*. *base* ^ *decode 0 m*) *div base* ^ *e*

lemma *decode_unfold*: *decode e M* = (∑ *m* ∈# *hmsetmset M*. *base* ^ *decode 0 m*) *div base* ^ *e*
by (*cases M*) *simp*

lemma *decode_0*[*simp*]: *decode e 0* = *0*
unfolding *zero_hmultiset_def* **by** *simp*

inductive $aligned_d$ :: *nat* ⇒ *hmultiset* ⇒ *bool* **where**
(∀ *m* ∈# *hmsetmset M*. *decode 0 m* ≥ *e*) ⟹ $aligned_d$ *e M*

lemma $aligned_d$*_0*[*simp*]: $aligned_d$ *0 M*
by (*rule* $aligned_d$*.intros*) *simp*

lemma $aligned_d$*_mono_exp_Suc*: $aligned_d$ (*Suc e*) *M* ⟹ $aligned_d$ *e M*
by (*auto simp*: $aligned_d$*.simps*)

lemma $aligned_d$*_mono_hmset*:
assumes $aligned_d$ *e M* **and** *hmsetmset M'* ⊆# *hmsetmset M*
shows $aligned_d$ *e M'*
using *assms* **by** (*auto simp*: $aligned_d$*.simps*)

lemma *decode_exp_shift_Suc*:
assumes $align_d$: $aligned_d$ (*Suc e*) *M*
shows *decode e M* = *base* ∗ *decode* (*Suc e*) *M*
proof (*subst* (*1 2*) *decode_unfold*, *subst* (*1 2*) *sum_mset_distrib_div_if_dvd*)
note *align'* = $align_d$[*unfolded* $aligned_d$*.simps*, *simplified*, *unfolded Suc_le_eq*]

show ∀ *m* ∈# *hmsetmset M*. *base* ^ *Suc e dvd base* ^ *decode 0 m*
using *align' Suc_leI le_imp_power_dvd* **by** *blast*

show ∀ *m* ∈# *hmsetmset M*. *base* ^ *e dvd base* ^ *decode 0 m*
using *align'* **by** (*simp add*: *le_imp_power_dvd le_less*)

have *base_e_nz*: *base* ^ *e* ≠ *0*
using *base_ge_2* **by** *simp*

have *mult_base*:
base ^ *decode 0 m div base* ^ *e* = *base* ∗ (*base* ^ *decode 0 m div* (*base* ∗ *base* ^ *e*))
if *m_in*: *m* ∈# *hmsetmset M* **for** *m*
using *m_in align'*
by (*subst power_div_same_base*[*OF base_e_nz*], *force*,
metis Suc_diff_Suc Suc_leI mult_is_0 power_Suc power_div_same_base power_not_zero)

show (∑ *m*∈#*hmsetmset M*. *base* ^ *decode 0 m div base* ^ *e*) =
base ∗ (∑ *m*∈#*hmsetmset M*. *base* ^ *decode 0 m div base* ^ *Suc e*)
by (*auto simp*: *sum_mset_distrib_left intro*!: *arg_cong*[*of* _ _ *sum_mset*] *image_mset_cong*
elim!: *mult_base*)
qed

lemma *decode_exp_shift*:
assumes $aligned_d$ *e M*
shows *decode 0 M* = *base* ^ *e* ∗ *decode e M*
using *assms* **by** (*induct e*) (*auto simp*: *decode_exp_shift_Suc dest*: $aligned_d$*_mono_exp_Suc*)

lemma *decode_plus*:
assumes $align_d$*_M*: $aligned_d$ *e M*
shows *decode e* (*M* + *N*) = *decode e M* + *decode e N*
using $align_d$*_M*[*unfolded* $aligned_d$*.simps*, *simplified*]

```
  by (subst (1 2 3) decode_unfold) (auto simp: hmsetmset_plus
    intro!: le_imp_power_dvd div_plus_div_distrib_dvd_left[OF sum_mset_dvd])

lemma less_imp_decode_less:
  assumes
    well_base_h M and
    aligned_d e M and
    aligned_d e N and
    M < N
  shows decode e M < decode e N
  using assms
proof (induct M arbitrary: N e rule: less_induct)
  case (less M)
  note ih = this(1) and well_h_M = this(2) and align_d_M = this(3) and align_d_N = this(4) and
    M_lt_N = this(5)

  obtain K Ma Na where
    M: M = K + Ma and
    N: N = K + Na and
    hds: head_ω Ma < head_ω Na
    using hmset_pair_decompose_less[OF M_lt_N] by blast

  obtain H where
    H: head_ω Na = ω^H
    using hds head_ω_def by fastforce
  have H_in: H ∈# hmsetmset Na
    by (metis (no_types) H Max_in add_mset_eq_single add_mset_not_empty finite_set_mset head_ω_def
      hmsetmset_empty_iff hmultiset.simps(1) set_mset_eq_empty_iff zero_hmultiset_def)

  have well_h_Ma: well_base_h Ma
    by (rule well_base_h_mono_hmset[OF well_h_M]) (simp add: M hmsetmset_plus)
  have align_d_K: aligned_d e K
    using M align_d_M aligned_d_mono_hmset hmsetmset_plus by auto
  have align_d_Ma: aligned_d e Ma
    using M align_d_M aligned_d_mono_hmset hmsetmset_plus by auto
  have align_d_Na: aligned_d e Na
    using N align_d_N aligned_d_mono_hmset hmsetmset_plus by auto

  have inj_on (decode 0) (set_mset (hmsetmset Ma))
    unfolding inj_on_def
  proof clarify
    fix x y
    assume
      x_in: x ∈# hmsetmset Ma and
      y_in: y ∈# hmsetmset Ma and
      dec_eq: decode 0 x = decode 0 y

    {
      fix x y
      assume
        x_in: x ∈# hmsetmset Ma and
        y_in: y ∈# hmsetmset Ma and
        x_lt_y: x < y

      have x_lt_M: x < M
        unfolding M using mem_hmsetmset_imp_less[OF x_in] by (simp add: trans_less_add2_hmset)
      have well_h_x: well_base_h x
        using well_h_Ma well_base_h.simps x_in by blast

      have decode 0 x < decode 0 y
        by (rule ih[OF x_lt_M well_h_x aligned_d_0 aligned_d_0 x_lt_y])
    }
    thus x = y
```

```
    using x_in y_in dec_eq by (metis leI less_irrefl_nat order.not_eq_order_implies_strict)
  qed
  hence well_dec_Ma: well_base (image_mset (decode 0) (hmsetmset Ma))
    by (rule well_base_image_inj[OF well_base_h_imp_well_base[OF well_h_Ma]])

  have H_bound: ∀m ∈# hmsetmset Ma. decode 0 m < decode 0 H
  proof
    fix m
    assume m_in: m ∈# hmsetmset Ma

    have ∀m ∈# hmsetmset (head_ω Ma). m < H
      using hds[unfolded H] using head_ω_def by auto
    hence m_lt_H: m < H
      using m_in
      by (metis Max_less_iff empty_iff finite_set_mset head_ω_def hmultiset.sel insert_iff
        set_mset_add_mset_insert)

    have m_lt_M: m < M
      using mem_hmsetmset_imp_less[OF m_in] by (simp add: M trans_less_add2_hmset)

    have well_h_m: well_base_h m
      using m_in well_h_Ma well_base_h.cases by blast

    show decode 0 m < decode 0 H
      by (rule ih[OF m_lt_M well_h_m aligned_d_0 aligned_d_0 m_lt_H])
  qed

  have decode 0 Ma < base ^ decode 0 H
    using well_base_bound[OF well_dec_Ma, simplified, OF H_bound] by (subst decode_unfold) simp
  also have ... ≤ decode 0 Na
    by (subst (2) decode_unfold, simp, rule sum_image_mset_mono_mem[OF H_in])
  finally have decode e Ma < decode e Na
    using decode_exp_shift[OF align_d_Ma] decode_exp_shift[OF align_d_Na] by simp
  thus decode e M < decode e N
    unfolding M N by (simp add: decode_plus[OF align_d_K])
qed

lemma inj_decode: inj_on (decode e) {M. well_base_h M ∧ aligned_d e M}
  unfolding inj_on_def Ball_def mem_Collect_eq
  by (metis less_imp_decode_less less_irrefl_nat neqE)

lemma decode_0_iff: well_base_h M ⟹ aligned_d e M ⟹ decode e M = 0 ⟷ M = 0
  by (metis aligned_d_0 decode_0 decode_exp_shift encode_0 less_imp_decode_less mult_0_right neqE
    not_less_zero well_base_h_encode)

lemma decode_encode: decode e (encode e n) = n
proof (induct e n rule: encode.induct)
  case (1 e n)
  note ih = this

  show ?case
  proof (cases n = 0)
    case n_nz: False

    have align_d1: aligned_d e (of_nat (n mod base) * ω^(encode 0 e))
      unfolding of_nat_times_ω_exp using n_nz by (auto simp: ih(1) aligned_d.simps)
    have align_d2: aligned_d (Suc e) (encode (Suc e) (n div base))
      by (safe intro!: aligned_d.intros, subst ih(1)[OF n_nz, symmetric],
        auto dest: mem_hmsetmset_encodeD intro!: Suc_le_eq[THEN iffD2]
          less_imp_decode_less[OF well_base_h_encode aligned_d_0 aligned_d_0] less_imp_encode_less)

    show ?thesis
      using ih base_ge_2
```

```
    by (subst encode.simps[unfolded of_nat_times_ω_exp])
      (simp add: decode_plus[OF align_d 1[unfolded of_nat_times_ω_exp]]
        decode_exp_shift_Suc[OF align_d 2])
  qed simp
qed

lemma encode_decode_exp_0: well_base_h M ⟹ encode 0 (decode 0 M) = M
  by (auto intro: inj_onD[OF inj_decode] decode_encode well_base_h_encode)

end

lemma well_base_h_mono_base:
  assumes
    well_h: well_base_h base M and
    two: 2 ≤ base and
    bases: base ≤ base'
  shows well_base_h base' M
  using two well_h
  by (induct rule: well_base_h.induct)
    (meson two bases less_le_trans order_trans well_base_h.intros well_base.simps)
```

12.5 The Goodstein Sequence and Goodstein's Theorem

```
context
  fixes start :: nat
begin

primrec goodstein :: nat ⇒ nat where
  goodstein 0 = start
| goodstein (Suc i) = decode (i + 3) 0 (encode (i + 2) 0 (goodstein i)) − 1

lemma goodstein_step:
  assumes gi_gt_0: goodstein i > 0
  shows encode (i + 2) 0 (goodstein i) > encode (i + 3) 0 (goodstein (i + 1))
proof −
  let ?Ei = encode (i + 2) 0 (goodstein i)
  let ?reencode = encode (i + 3) 0
  let ?decoded_Ei = decode (i + 3) 0 ?Ei

  have two_le: 2 ≤ i + 3
    by simp

  have well_base_h (i + 2) ?Ei
    by (rule well_base_h_encode) simp
  hence well_h: well_base_h (i + 3) ?Ei
    by (rule well_base_h_mono_base) simp_all

  have decoded_Ei_gt_0: ?decoded_Ei > 0
    by (metis gi_gt_0 gr0I encode_0_iff le_add2 decode_0_iff[OF _ well_h aligned_d_0] two_le)

  have ?reencode (?decoded_Ei − 1) < ?reencode ?decoded_Ei
    by (rule less_imp_encode_less[OF two_le]) (use decoded_Ei_gt_0 in linarith)
  also have ... = ?Ei
    by (simp only: encode_decode_exp_0[OF two_le well_h])
  finally show ?thesis
    by simp
qed

theorem goodsteins_theorem: ∃ i. goodstein i = 0
proof −
  let ?G = λi. encode (i + 2) 0 (goodstein i)

  obtain i where
    ¬ ?G i > ?G (i + 1)
```

using *wf_iff_no_infinite_down_chain*[*THEN iffD1*, *OF wf*,
unfolded not_ex not_all mem_Collect_eq prod.case, *rule_format*, *of ?G*]
by *auto*
hence *goodstein i = 0*
using *goodstein_step* **by** (*metis add.assoc gr0I one_plus_numeral semiring_norm*(*3*))
thus *?thesis*
by *blast*
qed

end

end

13 Towards Decidability of Behavioral Equivalence for Unary PCF

theory *Unary_PCF*
imports
HOL−Library.FSet
HOL−Library.Countable_Set_Type
HOL−Library.Nat_Bijection
Hereditary_Multiset
List−Index.List_Index
begin

13.1 Preliminaries

lemma *prod_UNIV*: $UNIV = UNIV \times UNIV$
by *auto*

lemma *infinite_cartesian_productI1*: $infinite\ A \Longrightarrow B \neq \{\} \Longrightarrow infinite\ (A \times B)$
by (*auto dest*!: *finite_cartesian_productD1*)

13.2 Types

datatype *type* = $\mathcal{B}$ ($\mathcal{B}$) | *Fun type type* (**infixr** $\rightarrow$ *65*)

definition *mk_fun* (**infixr** $\rightarrow\!\!\!\!\rightarrow$ *65*) **where**
$Ts \rightarrow\!\!\!\!\rightarrow T = fold\ (\rightarrow)\ (rev\ Ts)\ T$

primrec *dest_fun* **where**
dest_fun $\mathcal{B} = []$
| $dest\_fun\ (T \rightarrow U) = T\ \#\ dest\_fun\ U$

definition *arity* **where**
$arity\ T = length\ (dest\_fun\ T)$

lemma *mk_fun_dest_fun*[*simp*]: $dest\_fun\ T \rightarrow\!\!\!\!\rightarrow \mathcal{B} = T$
by (*induct T*) (*auto simp*: *mk_fun_def*)

lemma *dest_fun_mk_fun*[*simp*]: $dest\_fun\ (Ts \rightarrow\!\!\!\!\rightarrow T) = Ts\ @\ dest\_fun\ T$
by (*induct Ts*) (*auto simp*: *mk_fun_def*)

primrec δ **where**
$\delta\ \mathcal{B} = HMSet\ \{\#\}$
| $\delta\ (T \rightarrow U) = HMSet\ (add\_mset\ (\delta\ T)\ (hmsetmset\ (\delta\ U)))$

lemma δ*_mk_fun*: $\delta\ (Ts \rightarrow\!\!\!\!\rightarrow T) = HMSet\ (hmsetmset\ (\delta\ T) + mset\ (map\ \delta\ Ts))$
by (*induct Ts*) (*auto simp*: *mk_fun_def*)

lemma *type_induct* [*case_names Fun*]:
assumes
$(\bigwedge T.\ (\bigwedge T1\ T2.\ T = T1 \rightarrow T2 \Longrightarrow P\ T1) \Longrightarrow$
$(\bigwedge T1\ T2.\ T = T1 \rightarrow T2 \Longrightarrow P\ T2) \Longrightarrow P\ T)$

shows *P T*
proof (*induct T*)
case $\mathcal{B}$
show *?case* **by** (*rule assms*) *simp_all*
next
case *Fun*
show *?case* **by** (*rule assms*) (*insert Fun*, *simp_all*)
qed

13.3 Terms

type-synonym *name* = *string*
type-synonym *idx* = *nat*
datatype *expr* =
Var name ∗ *type* (⟨_⟩) | *Bound idx* | *B bool*
| *Seq expr expr* (**infixr** *?* 75) | *App expr expr* (**infixl** · 75)
| *Abs type expr* (Λ⟨_⟩ _ [100, 100] 800)

declare [[*coercion_enabled*]]
declare [[*coercion B*]]
declare [[*coercion Bound*]]

notation (**output**) *B* (_)
notation (**output**) *Bound* (_)

primrec *open* :: *idx* ⇒ *expr* ⇒ *expr* ⇒ *expr* **where**
open i t (*j* :: *idx*) = (*if i* = *j then t else j*)
| *open i t* ⟨*yU*⟩ = ⟨*yU*⟩
| *open i t* (*b* :: *bool*) = *b*
| *open i t* (*e1 ? e2*) = *open i t e1 ? open i t e2*
| *open i t* (*e1* · *e2*) = *open i t e1* · *open i t e2*
| *open i t* (Λ⟨*U*⟩ *e*) = Λ⟨*U*⟩ (*open* (*i* + 1) *t e*)

abbreviation *open0* ≡ *open 0*
abbreviation *open_Var i xT* ≡ *open i* ⟨*xT*⟩
abbreviation *open0_Var xT* ≡ *open 0* ⟨*xT*⟩

primrec *close_Var* :: *idx* ⇒ *name* × *type* ⇒ *expr* ⇒ *expr* **where**
close_Var i xT (*j* :: *idx*) = *j*
| *close_Var i xT* ⟨*yU*⟩ = (*if xT* = *yU then i else* ⟨*yU*⟩)
| *close_Var i xT* (*b* :: *bool*) = *b*
| *close_Var i xT* (*e1 ? e2*) = *close_Var i xT e1 ? close_Var i xT e2*
| *close_Var i xT* (*e1* · *e2*) = *close_Var i xT e1* · *close_Var i xT e2*
| *close_Var i xT* (Λ⟨*U*⟩ *e*) = Λ⟨*U*⟩ (*close_Var* (*i* + 1) *xT e*)

abbreviation *close0_Var* ≡ *close_Var 0*

primrec *fv* :: *expr* ⇒ (*name* × *type*) *fset* **where**
fv (*j* :: *idx*) = {||}
| *fv* ⟨*yU*⟩ = {|*yU*|}
| *fv* (*b* :: *bool*) = {||}
| *fv* (*e1 ? e2*) = *fv e1* |∪| *fv e2*
| *fv* (*e1* · *e2*) = *fv e1* |∪| *fv e2*
| *fv* (Λ⟨*U*⟩ *e*) = *fv e*

abbreviation *fresh x e* ≡ *x* |∉| *fv e*

lemma *ex_fresh*: ∃ *x*. (*x* :: *char list*, *T*) |∉| *A*
proof (*rule ccontr*, *unfold not_ex not_not*)
assume ∀ *x*. (*x*, *T*) |∈| *A*
then have *infinite* {*x*. (*x*, *T*) |∈| *A*} (**is** *infinite ?P*)
by (*auto simp add*: *infinite_UNIV_listI*)
moreover
have *?P* ⊆ *fst* ‘ *fset A*

```
    by (force simp: fmember.rep_eq)
  from finite_surj[OF _ this] have finite ?P
    by simp
  ultimately show False by blast
qed

inductive lc where
  lc_Var[simp]: lc ⟨xT⟩
| lc_B[simp]: lc (b :: bool)
| lc_Seq: lc e1 ⟹ lc e2 ⟹ lc (e1 ; e2)
| lc_App: lc e1 ⟹ lc e2 ⟹ lc (e1 · e2)
| lc_Abs: (∀x. (x, T) |∉| X ⟶ lc (open0_Var (x, T) e)) ⟹ lc (Λ⟨T⟩ e)

declare lc.intros[intro]

definition body T t ≡ (∃X. ∀x. (x, T) |∉| X ⟶ lc (open0_Var (x, T) t))

lemma lc_Abs_iff_body: lc (Λ⟨T⟩ t) ⟷ body T t
  unfolding body_def by (subst lc.simps) simp

lemma fv_open_Var: fresh xT t ⟹ fv (open_Var i xT t) |⊆| finsert xT (fv t)
  by (induct t arbitrary: i) auto

lemma fv_close_Var[simp]: fv (close_Var i xT t) = fv t |−| {|xT|}
  by (induct t arbitrary: i) auto

lemma close_Var_open_Var[simp]: fresh xT t ⟹ close_Var i xT (open_Var i xT t) = t
  by (induct t arbitrary: i) auto

lemma open_Var_inj: fresh xT t ⟹ fresh xT u ⟹ open_Var i xT t = open_Var i xT u ⟹ t = u
  by (metis close_Var_open_Var)

context begin

private lemma open_Var_open_Var_close_Var: i ≠ j ⟹ xT ≠ yU ⟹ fresh yU t ⟹
  open_Var i yU (open_Var j zV (close_Var j xT t)) = open_Var j zV (close_Var j xT (open_Var i yU t))
  by (induct t arbitrary: i j) auto

lemma open_Var_close_Var[simp]: lc t ⟹ open_Var i xT (close_Var i xT t) = t
proof (induction t arbitrary: i rule: lc.induct)
  case (lc_Abs T X e i)
  obtain x where x: fresh (x, T) e (x, T) ≠ xT (x, T) |∉| X
    using ex_fresh[of _ fv e |∪| finsert xT X] by blast
  with lc_Abs.IH have lc (open0_Var (x, T) e)
    open_Var (i + 1) xT (close_Var (i + 1) xT (open0_Var (x, T) e)) = open0_Var (x, T) e
    by auto
  with x show ?case
    by (auto simp: open_Var_open_Var_close_Var
      dest: fset_mp[OF fv_open_Var, rotated]
      intro!: open_Var_inj[of (x, T) _ e 0])
qed auto

end

lemma close_Var_inj: lc t ⟹ lc u ⟹ close_Var i xT t = close_Var i xT u ⟹ t = u
  by (metis open_Var_close_Var)

primrec Apps (infixl · 75) where
  f · [] = f
| f · (x # xs) = f · x · xs

lemma Apps_snoc: f · (xs @ [x]) = f · xs · x
  by (induct xs arbitrary: f) auto
```

lemma *Apps_append*: $f \cdot (xs \mathbin{@} ys) = f \cdot xs \cdot ys$
by (*induct xs arbitrary*: *f*) *auto*

lemma *Apps_inj*[*simp*]: $f \cdot ts = g \cdot ts \longleftrightarrow f = g$
by (*induct ts arbitrary*: *f g*) *auto*

lemma *eq_Apps_conv*[*simp*]:
fixes $i :: idx$ **and** $b :: bool$ **and** $f :: expr$ **and** $ts :: expr\ list$
shows
$(\langle m \rangle = f \cdot ts) = (\langle m \rangle = f \wedge ts = [])$
$(f \cdot ts = \langle m \rangle) = (\langle m \rangle = f \wedge ts = [])$
$(i = f \cdot ts) = (i = f \wedge ts = [])$
$(f \cdot ts = i) = (i = f \wedge ts = [])$
$(b = f \cdot ts) = (b = f \wedge ts = [])$
$(f \cdot ts = b) = (b = f \wedge ts = [])$
$(e1\ ?\ e2 = f \cdot ts) = (e1\ ?\ e2 = f \wedge ts = [])$
$(f \cdot ts = e1\ ?\ e2) = (e1\ ?\ e2 = f \wedge ts = [])$
$(\Lambda\langle T \rangle\ t = f \cdot ts) = (\Lambda\langle T \rangle\ t = f \wedge ts = [])$
$(f \cdot ts = \Lambda\langle T \rangle\ t) = (\Lambda\langle T \rangle\ t = f \wedge ts = [])$
by (*induct ts arbitrary*: *f*) *auto*

lemma *Apps_Var_eq*[*simp*]: $\langle xT \rangle \cdot ss = \langle yU \rangle \cdot ts \longleftrightarrow xT = yU \wedge ss = ts$
proof (*induct ss arbitrary*: *ts rule*: *rev_induct*)
case *snoc*
then show *?case* **by** (*induct ts rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)
qed *auto*

lemma *Apps_Abs_neq_Apps*[*simp*, *symmetric*, *simp*]:
$\Lambda\langle T \rangle\ r \cdot t \neq \langle xT \rangle \cdot ss$
$\Lambda\langle T \rangle\ r \cdot t \neq (i :: idx) \cdot ss$
$\Lambda\langle T \rangle\ r \cdot t \neq (b :: bool) \cdot ss$
$\Lambda\langle T \rangle\ r \cdot t \neq (e1\ ?\ e2) \cdot ss$
by (*induct ss rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)

lemma *App_Abs_eq_Apps_Abs*[*simp*]: $\Lambda\langle T \rangle\ r \cdot t = \Lambda\langle T' \rangle\ r' \cdot ss \longleftrightarrow T = T' \wedge r = r' \wedge ss = [t]$
by (*induct ss rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)

lemma *Apps_Var_neq_Apps_Abs*[*simp*, *symmetric*, *simp*]: $\langle xT \rangle \cdot ss \neq \Lambda\langle T \rangle\ r \cdot ts$
proof (*induct ss arbitrary*: *ts rule*: *rev_induct*)
case (*snoc a ss*)
then show *?case* **by** (*induct ts rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)
qed *simp*

lemma *Apps_Var_neq_Apps_beta*[*simp*, *THEN not_sym*, *simp*]:
$\langle xT \rangle \cdot ss \neq \Lambda\langle T \rangle\ r \cdot s \cdot ts$
by (*metis Apps_Var_neq_Apps_Abs Apps_append Apps_snoc eq_Apps_conv*(*9*))

lemma [*simp*]:
$(\Lambda\langle T \rangle\ r \cdot ts = \Lambda\langle T' \rangle\ r' \cdot s' \cdot ts') = (T = T' \wedge r = r' \wedge ts = s' \# ts')$
proof (*induct ts arbitrary*: *ts' rule*: *rev_induct*)
case *Nil*
then show *?case* **by** (*induct ts' rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)
next
case *snoc*
then show *?case* **by** (*induct ts' rule*: *rev_induct*) (*auto simp*: *Apps_snoc*)
qed

lemma *fold_eq_Bool_iff*[*simp*]:
$fold\ (\rightarrow)\ (rev\ Ts)\ T = \mathcal{B} \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
$\mathcal{B} = fold\ (\rightarrow)\ (rev\ Ts)\ T \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
by (*induct Ts*) *auto*

lemma *fold_eq_Fun_iff*[*simp*]:
  fold (→) (*rev Ts*) *T* = *U* → *V* ⟷
   (*Ts* = [] ∧ *T* = *U* → *V* ∨ (∃ *Us*. *Ts* = *U* # *Us* ∧ *fold* (→) (*rev Us*) *T* = *V*))
  by (*induct Ts*) *auto*

13.4 Substitution

primrec *subst* **where**
  subst xT t ⟨*yU*⟩ = (*if xT* = *yU then t else* ⟨*yU*⟩)
| *subst xT t* (*i* :: *idx*) = *i*
| *subst xT t* (*b* :: *bool*) = *b*
| *subst xT t* (*e1* ? *e2*) = *subst xT t e1* ? *subst xT t e2*
| *subst xT t* (*e1* · *e2*) = *subst xT t e1* · *subst xT t e2*
| *subst xT t* (Λ⟨*T*⟩ *e*) = Λ⟨*T*⟩ (*subst xT t e*)

lemma *fv_subst*:
  fv (*subst xT t u*) = *fv u* |−| {|*xT*|} |∪| (*if xT* |∈| *fv u then fv t else* {||})
  by (*induct u*) *auto*

lemma *subst_fresh*: *fresh xT u* ⟹ *subst xT t u* = *u*
  by (*induct u*) *auto*

context begin

private lemma *open_open_id*: *i* ≠ *j* ⟹ *open i t* (*open j t'* *u*) = *open j t'* *u* ⟹ *open i t u* = *u*
  by (*induct u arbitrary*: *i j*) (*auto 6 0*)

lemma *lc_open_id*: *lc u* ⟹ *open k t u* = *u*
proof (*induct u arbitrary*: *k rule*: *lc.induct*)
  case (*lc_Abs T X e*)
  obtain *x* **where** *x*: *fresh* (*x*, *T*) *e* (*x*, *T*) |∉| *X*
    using *ex_fresh*[*of* _ *fv e* |∪| *X*] **by** *blast*
  with *lc_Abs* **show** *?case*
    by (*auto intro*: *open_open_id dest*: *spec*[*of* _ *x*] *spec*[*of* _ *Suc k*])
qed *auto*

lemma *subst_open*: *lc u* ⟹ *subst xT u* (*open i t v*) = *open i* (*subst xT u t*) (*subst xT u v*)
  by (*induction v arbitrary*: *i*) (*auto intro*: *lc_open_id*[*symmetric*])

lemma *subst_open_Var*:
  xT ≠ *yU* ⟹ *lc u* ⟹ *subst xT u* (*open_Var i yU v*) = *open_Var i yU* (*subst xT u v*)
  by (*auto simp*: *subst_open*)

lemma *subst_Apps*[*simp*]:
  subst xT u (*f* · *xs*) = *subst xT u f* · *map* (*subst xT u*) *xs*
  by (*induct xs arbitrary*: *f*) *auto*

end

context begin

private lemma *fresh_close_Var_id*: *fresh xT t* ⟹ *close_Var k xT t* = *t*
  by (*induct t arbitrary*: *k*) *auto*

lemma *subst_close_Var*:
  xT ≠ *yU* ⟹ *fresh yU u* ⟹ *subst xT u* (*close_Var i yU t*) = *close_Var i yU* (*subst xT u t*)
  by (*induct t arbitrary*: *i*) (*auto simp*: *fresh_close_Var_id*)

end

lemma *subst_intro*: *fresh xT t* ⟹ *lc u* ⟹ *open0 u t* = *subst xT u* (*open0_Var xT t*)
  by (*auto simp*: *subst_fresh subst_open*)

lemma *lc_subst*[*simp*]: *lc u* ⟹ *lc t* ⟹ *lc* (*subst xT t u*)

proof (*induct u rule*: *lc.induct*)
  case (*lc_ Abs T X e*)
  then show *?case*
    by (*auto simp*: *subst_ open_ Var intro*!: *lc.lc_ Abs*[*of _ fv e* |∪| *X* |∪| {|*xT*|}])
qed *auto*

lemma *body_ subst*[*simp*]: *body U u* ⟹ *lc t* ⟹ *body U* (*subst xT t u*)
proof (*subst* (*asm*) *body_ def*, *elim conjE exE*)
  fix *X*
  assume [*simp*]: *lc t* ∀*x*. (*x*, *U*) |∉| *X* ⟶ *lc* (*open0_ Var* (*x*, *U*) *u*)
  show *body U* (*subst xT t u*)
  proof (*unfold body_ def*, *intro exI*[*of _ finsert xT X*] *conjI allI impI*)
    fix *x*
    assume (*x*, *U*) |∉| *finsert xT X*
    then show *lc* (*open0_ Var* (*x*, *U*) (*subst xT t u*))
      by (*auto simp*: *subst_ open_ Var*[*symmetric*])
  qed
qed

lemma *lc_ open_ Var*: *lc u* ⟹ *lc* (*open_ Var i xT u*)
  by (*simp add*: *lc_ open_ id*)

lemma *lc_ open*[*simp*]: *body U u* ⟹ *lc t* ⟹ *lc* (*open0 t u*)
proof (*unfold body_ def*, *elim conjE exE*)
  fix *X*
  assume [*simp*]: *lc t* ∀*x*. (*x*, *U*) |∉| *X* ⟶ *lc* (*open0_ Var* (*x*, *U*) *u*)
  with *ex_ fresh*[*of _ fv u* |∪| *X*] **obtain** *x* **where** [*simp*]: *fresh* (*x*, *U*) *u* (*x*, *U*) |∉| *X* **by** *blast*
  show *?thesis* **by** (*subst subst_ intro*[*of* (*x*, *U*)]) *auto*
qed

13.5 Typing

inductive *welltyped* :: *expr* ⇒ *type* ⇒ *bool* (**infix** ::: *60*) **where**
  welltyped_ Var[*intro*!]: ⟨(*x*, *T*)⟩ ::: *T*
| *welltyped_ B*[*intro*!]: (*b* :: *bool*) ::: *B*
| *welltyped_ Seq*[*intro*!]: *e1* ::: *B* ⟹ *e2* ::: *B* ⟹ *e1* ? *e2* ::: *B*
| *welltyped_ App*[*intro*]: *e1* ::: *T* → *U* ⟹ *e2* ::: *T* ⟹ *e1* · *e2* ::: *U*
| *welltyped_ Abs*[*intro*]: (∀*x*. (*x*, *T*) |∉| *X* ⟶ *open0_ Var* (*x*, *T*) *e* ::: *U*) ⟹ Λ⟨*T*⟩ *e* ::: *T* → *U*

inductive-cases *welltypedE*[*elim*!]:
  ⟨*x*⟩ ::: *T*
  (*i* :: *idx*) ::: *T*
  (*b* :: *bool*) ::: *T*
  e1 ? *e2* ::: *T*
  e1 · *e2* ::: *T*
  Λ⟨*T*⟩ *e* ::: *U*

lemma *welltyped_ unique*: *t* ::: *T* ⟹ *t* ::: *U* ⟹ *T* = *U*
proof (*induction t T arbitrary*: *U rule*: *welltyped.induct*)
  case (*welltyped_ Abs T X t U T'*)
  from *welltyped_ Abs.prems* **show** *?case*
  proof (*elim welltypedE*)
    fix *Y U'*
    obtain *x* **where** [*simp*]: (*x*, *T*) |∉| *X* (*x*, *T*) |∉| *Y*
      using *ex_ fresh*[*of _ X* |∪| *Y*] **by** *blast*
    assume [*simp*]: *T'* = *T* → *U'* ∀*x*. (*x*, *T*) |∉| *Y* ⟶ *open0_ Var* (*x*, *T*) *t* ::: *U'*
    show *T* → *U* = *T'*
      by (*auto intro*: *conjunct2*[*OF welltyped_ Abs.IH*[*rule_ format*], *rule_ format*, *of x*])
  qed
qed *blast*+

lemma *welltyped_ lc*[*simp*]: *t* ::: *T* ⟹ *lc t*
  by (*induction t T rule*: *welltyped.induct*) *auto*

lemma *welltyped_subst*[*intro*]:
  $u ::: U \Longrightarrow t ::: snd\ xT \Longrightarrow subst\ xT\ t\ u ::: U$
proof (*induction u U rule*: *welltyped.induct*)
  case (*welltyped_Abs T' X u U*)
  then show *?case* **unfolding** *subst.simps*
    by (*intro welltyped.welltyped_Abs*[*of _ finsert xT X*]) (*auto simp*: *subst_open_Var*[*symmetric*])
qed *auto*

lemma *rename_welltyped*: $u ::: U \Longrightarrow subst\ (x,\ T)\ \langle (y,\ T)\rangle\ u ::: U$
  by (*rule welltyped_subst*) *auto*

lemma *welltyped_Abs_fresh*:
  assumes *fresh* $(x,\ T)\ u$ *open0_Var* $(x,\ T)\ u ::: U$
  shows $\Lambda\langle T\rangle\ u ::: T \to U$
proof (*intro welltyped_Abs*[*of _ fv u*] *allI impI*)
  fix *y*
  assume *fresh* $(y,\ T)\ u$
  with *assms*(*2*) **have** *subst* $(x,\ T)\ \langle (y,\ T)\rangle$ $(open0\_Var\ (x,\ T)\ u) ::: U$ (**is** *?t* ::: _)
    by (*auto intro*: *rename_welltyped*)
  also have *?t* = *open0_Var* $(y,\ T)\ u$
    by (*subst subst_intro*[*symmetric*]) (*auto simp*: *assms*(*1*))
  finally show $open0\_Var\ (y,\ T)\ u ::: U$.
qed

lemma *Apps_alt*: $f \cdot ts ::: T \longleftrightarrow$
  $(\exists Ts.\ f ::: fold\ (\to)\ (rev\ Ts)\ T \wedge list\_all2\ (:::)\ ts\ Ts)$
proof (*induct ts arbitrary*: *f*)
  case (*Cons t ts*)
  from *Cons*(*1*)[*of f · t*] **show** *?case*
    by (*force simp*: *list_all2_Cons1*)
qed *simp*

13.6 Definition 10 and Lemma 11 from Schmidt-Schauß's paper

abbreviation $closed\ t \equiv fv\ t = \{||\}$

primrec *constant0* **where**
  $constant0\ \mathcal{B} = Var\ (''bool'', \mathcal{B})$
| $constant0\ (T \to U) = \Lambda\langle T\rangle\ (constant0\ U)$

definition $constant\ T = \Lambda\langle\mathcal{B}\rangle\ (close0\_Var\ (''bool'', \mathcal{B})\ (constant0\ T))$

lemma *fv_constant0*[*simp*]: $fv\ (constant0\ T) = \{|(''bool'', \mathcal{B})|\}$
  by (*induct T*) *auto*

lemma *closed_constant*[*simp*]: *closed* (*constant T*)
  unfolding *constant_def* **by** *auto*

lemma *welltyped_constant0*[*simp*]: $constant0\ T ::: T$
  by (*induct T*) (*auto simp*: *lc_open_id*)

lemma *lc_constant0*[*simp*]: *lc* (*constant0 T*)
  using *welltyped_constant0 welltyped_lc* **by** *blast*

lemma *welltyped_constant*[*simp*]: $constant\ T ::: \mathcal{B} \to T$
  unfolding *constant_def* **by** (*auto intro*: *welltyped_Abs_fresh*[*of ''bool''*])

definition *nth_drop* **where**
  $nth\_drop\ i\ xs \equiv take\ i\ xs\ @\ drop\ (Suc\ i)\ xs$

definition *nth_arg* (**infixl** !− *100*) **where**
  $nth\_arg\ T\ i \equiv nth\ (dest\_fun\ T)\ i$

abbreviation *ar* **where**

```
  ar T ≡ length (dest_fun T)

lemma size_nth_arg[simp]: i < ar T ⟹ size (T !− i) < size T
  by (induct T arbitrary: i) (force simp: nth_Cons' nth_arg_def gr0_conv_Suc)+

fun π :: type ⇒ nat ⇒ nat ⇒ type where
  π T i 0 = (if i < ar T then nth_drop i (dest_fun T) →→ B else B)
| π T i (Suc j) = (if i < ar T ∧ j < ar (T!−i)
    then π (T!−i) j 0 →
      map (π (T!−i) j o Suc) [0 ..< ar (T!−i!−j)] →→ π T i 0 else B)

theorem π_induct[rotated −2, consumes 2, case_names 0 Suc]:
  assumes ⋀T i. i < ar T ⟹ P T i 0
    and ⋀T i j. i < ar T ⟹ j < ar (T !− i) ⟹ P (T !− i) j 0 ⟹
      (∀x < ar (T !− i !− j). P (T !− i) j (x + 1)) ⟹ P T i (j + 1)
  shows i < ar T ⟹ j ≤ ar (T !− i) ⟹ P T i j
  by (induct T i j rule: π.induct) (auto intro: assms[simplified])

definition ε :: type ⇒ nat ⇒ type where
  ε T i = π T i 0 → map (π T i o Suc) [0 ..< ar (T!−i)] →→ T

definition Abss (Λ[_] _ [100, 100] 800) where
  Λ[xTs] b = fold (λxT t. Λ⟨snd xT⟩ close0_Var xT t) (rev xTs) b

definition Seqs (infixr ?? 75) where
  ts ?? t = fold (λu t. u ? t) (rev ts) t

definition variant k base = base @ replicate k CHR ''*''

lemma variant_inj: variant i base = variant j base ⟹ i = j
  unfolding variant_def by auto

lemma variant_inj2:
  CHR ''*'' ∉ set b1 ⟹ CHR ''*'' ∉ set b2 ⟹ variant i b1 = variant j b2 ⟹ b1 = b2
  unfolding variant_def
  by (auto simp: append_eq_append_conv2)
    (metis Nil_is_append_conv hd_append2 hd_in_set hd_rev last_ConsR
     last_snoc replicate_append_same rev_replicate)+

fun E :: type ⇒ nat ⇒ expr and P :: type ⇒ nat ⇒ nat ⇒ expr where
  E T i = (if i < ar T then (let
      Ti = T!−i;
      x = λk. (variant k ''x'', T!−k);
      xs = map x [0 ..< ar T];
      xx_var = ⟨nth xs i⟩;
      x_vars = map (λx. ⟨x⟩) (nth_drop i xs);
      yy = (''z'', π T i 0);
      yy_var = ⟨yy⟩;
      y = λj. (variant j ''y'', π T i (j + 1));
      ys = map y [0 ..< ar Ti];
      e = λj. ⟨y j⟩ • (P Ti j 0 · xx_var # map (λk. P Ti j (k + 1) · xx_var) [0 ..< ar (Ti!−j)]);
      guards = map (λi. xx_var •
          map (λj. constant (Ti!−j) · (if i = j then e i • x_vars else True)) [0 ..< ar Ti])
        [0 ..< ar Ti]
    in Λ[(yy # ys @ xs)] (guards ?? (yy_var • x_vars))) else constant (ε T i) · False)
| P T i 0 =
    (if i < ar T then (let
      f = (''f'', T);
      f_var = ⟨f⟩;
      x = λk. (variant k ''x'', T!−k);
      xs = nth_drop i (map x [0 ..< ar T]);
      x_vars = insert_nth i (constant (T!−i) · True) (map (λx. ⟨x⟩) xs)
    in Λ[(f # xs)] (f_var • x_vars)) else constant (T → π T i 0) · False)
```

```
| P T i (Suc j) = (if i < ar T ∧ j < ar (T!−i) then (let
      Ti = T!−i;
      Tij = Ti!−j;
      f = (''f'', T);
      f_var = ⟨f⟩;
      x = λk. (variant k ''x'', T!−k);
      xs = nth_drop i (map x [0 ..< ar T]);
      yy = (''z'', π Ti j 0);
      yy_var = ⟨yy⟩;
      y = λk. (variant k ''y'', π Ti j (k + 1));
      ys = map y [0 ..< ar Tij];
      y_vars = yy_var # map (λx. ⟨x⟩) ys;
      x_vars = insert_nth i (E Ti j · y_vars) (map (λx. ⟨x⟩) xs)
    in Λ[(f # yy # ys @ xs)] (f_var · x_vars)) else constant (T → π T i (j + 1)) · False)

lemma Abss_Nil[simp]: Λ[[]] b = b
  unfolding Abss_def by simp

lemma Abss_Cons[simp]: Λ[(x#xs)] b = Λ⟨snd x⟩ (close0_Var x (Λ[xs] b))
  unfolding Abss_def by simp

lemma welltyped_Abss: b ::: U ⟹ T = map snd xTs →→ U ⟹ Λ[xTs] b ::: T
  by (hypsubst_thin, induct xTs) (auto simp: mk_fun_def intro!: welltyped_Abs_fresh)

lemma welltyped_Apps: list_all2 (:::) ts Ts ⟹ f ::: Ts →→ U ⟹ f · ts ::: U
  by (induct ts Ts arbitrary: f rule: list.rel_induct) (auto simp: mk_fun_def)

lemma welltyped_open_Var_close_Var[intro!]:
  t ::: T ⟹ open0_Var xT (close0_Var xT t) ::: T
  by auto

lemma welltyped_Var_iff[simp]:
  ⟨(x, T)⟩ ::: U ⟷ T = U
  by auto

lemma welltyped_bool_iff[simp]: (b :: bool) ::: T ⟷ T = ℬ
  by auto

lemma welltyped_constant0_iff[simp]: constant0 T ::: U ⟷ (U = T)
  by (induct T arbitrary: U) (auto simp: ex_fresh lc_open_id)

lemma welltyped_constant_iff[simp]: constant T ::: U ⟷ (U = ℬ → T)
  unfolding constant_def
proof (intro iffI, elim welltypedE, hypsubst_thin, unfold type.inject simp_thms)
  fix X U
  assume ∀x. (x, ℬ) |∉| X ⟶ open0_Var (x, ℬ) (close0_Var (''bool'', ℬ) (constant0 T)) ::: U
  moreover obtain x where (x, ℬ) |∉| X using ex_fresh[of ℬ X] by blast
  ultimately have open0_Var (x, ℬ) (close0_Var (''bool'', ℬ) (constant0 T)) ::: U by simp
  then have open0_Var (''bool'', ℬ) (close0_Var (''bool'', ℬ) (constant0 T)) ::: U
    using rename_welltyped[of ‹open0_Var (x, ℬ) (close0_Var (''bool'', ℬ) (constant0 T))›
      U x ℬ ''bool''⁀]
    by (auto simp: subst_open subst_fresh)
  then show U = T by auto
qed (auto intro!: welltyped_Abs_fresh)

lemma welltyped_Seq_iff[simp]: e1 ? e2 ::: T ⟷ (T = ℬ ∧ e1 ::: ℬ ∧ e2 ::: ℬ)
  by auto

lemma welltyped_Seqs_iff[simp]: es ?? e ::: T ⟷
  ((es ≠ [] ⟶ T = ℬ) ∧ (∀e ∈ set es. e ::: ℬ) ∧ e ::: T)
  by (induct es arbitrary: e) (auto simp: Seqs_def)

lemma welltyped_App_iff[simp]: f · t ::: U ⟷ (∃T. f ::: T → U ∧ t ::: T)
```

by *auto*

lemma *welltyped_Apps_iff*[*simp*]: $f \cdot ts ::: U \longleftrightarrow (\exists Ts.\ f ::: Ts \rightarrow\!\!\!\!\rightarrow U \wedge list\_all2\ (:::)\ ts\ Ts)$
by (*induct ts arbitrary*: *f*) (*auto 0 3 simp*: *mk_fun_def list_all2_Cons1 intro*: *exI*[*of* _ _ # _])

lemma *eq_mk_fun_iff*[*simp*]: $T = Ts \rightarrow\!\!\!\!\rightarrow \mathcal{B} \longleftrightarrow Ts = dest\_fun\ T$
by *auto*

lemma *map_nth_eq_drop_take*[*simp*]: $j \leq length\ xs \Longrightarrow map\ (nth\ xs)\ [i\ ..<\ j] = drop\ i\ (take\ j\ xs)$
by (*induct j*) (*auto simp*: *take_Suc_conv_app_nth*)

lemma *dest_fun_π_0*: $i < ar\ T \Longrightarrow dest\_fun\ (\pi\ T\ i\ 0) = nth\_drop\ i\ (dest\_fun\ T)$
by *auto*

lemma *welltyped_E*: $E\ T\ i ::: \varepsilon\ T\ i$ **and** *welltyped_P*: $P\ T\ i\ j ::: T \rightarrow \pi\ T\ i\ j$
proof (*induct T i* **and** *T i j rule*: *E_P.induct*)
case (*1 T i*)
note *P.simps*[*simp del*] *π.simps*[*simp del*] *ε_def*[*simp*] *nth_drop_def*[*simp*] *nth_arg_def*[*simp*]
from *1*(*1*)[*OF* _ *refl refl refl refl refl refl refl refl refl*]
1(*2*)[*OF* _ *refl refl refl refl refl refl refl refl refl*]
show *?case*
by (*auto 0 4 simp*: *Let_def o_def take_map*[*symmetric*] *drop_map*[*symmetric*]
list_all2_conv_all_nth nth_append min_def dest_fun_π_0 π.simps[*of T i*]
intro!: *welltyped_Abs_fresh welltyped_Abss*[*of* _ $\mathcal{B}$])
next
case (*2 T i*)
show *?case*
by (*auto simp*: *Let_def take_map drop_map o_def list_all2_conv_all_nth nth_append nth_Cons'*
nth_drop_def nth_arg_def
intro!: *welltyped_constant welltyped_Abs_fresh welltyped_Abss*[*of* _ $\mathcal{B}$])
next
case (*3 T i j*)
note *E.simps*[*simp del*] *π.simps*[*simp del*] *Abss_Cons*[*simp del*] *ε_def*[*simp*]
nth_drop_def[*simp*] *nth_arg_def*[*simp*]
from *3*(*1*)[*OF* _ *refl refl refl refl refl refl refl refl refl refl refl*]
show *?case*
by (*auto 0 3 simp*: *Let_def o_def take_map*[*symmetric*] *drop_map*[*symmetric*]
list_all2_conv_all_nth nth_append nth_Cons' min_def π.simps[*of T i*]
intro!: *welltyped_Abs_fresh welltyped_Abss*[*of* _ $\mathcal{B}$])
qed

lemma *δ_gt_0*[*simp*]: $T \neq \mathcal{B} \Longrightarrow HMSet\ \{\#\} < \delta\ T$
by (*cases T*) *auto*

lemma *mset_nth_drop_less*: $i < length\ xs \Longrightarrow mset\ (nth\_drop\ i\ xs) < mset\ xs$
by (*induct xs arbitrary*: *i*) (*auto simp*: *take_Cons' nth_drop_def gr0_conv_Suc*)

lemma *map_nth_drop*: $i < length\ xs \Longrightarrow map\ f\ (nth\_drop\ i\ xs) = nth\_drop\ i\ (map\ f\ xs)$
by (*induct xs arbitrary*: *i*) (*auto simp*: *take_Cons' nth_drop_def gr0_conv_Suc*)

lemma *empty_less_mset*: $\{\#\} < mset\ xs \longleftrightarrow xs \neq []$
by *auto*

lemma *dest_fun_alt*: $dest\_fun\ T = map\ (\lambda i.\ T\ !-\ i)\ [0..<ar\ T]$
unfolding *list_eq_iff_nth_eq nth_arg_def* **by** *auto*

context notes *π.simps*[*simp del*] **notes** *One_nat_def*[*simp del*] **begin**

lemma *δ_π*:
assumes $i < ar\ T\ j \leq ar\ (T\ !-\ i)$
shows $\delta\ (\pi\ T\ i\ j) < \delta\ T$
using *assms* **proof** (*induct T i j rule*: *π_induct*)
fix *T i*

```
  assume i < ar T
  then show δ (π T i 0) < δ T
    by (subst (2) mk_fun_dest_fun[symmetric, of T], unfold δ_mk_fun)
      (auto simp: δ_mk_fun mset_map[symmetric] take_map[symmetric] drop_map[symmetric] π.simps
        mset_nth_drop_less map_nth_drop simp del: mset_map)
next
  fix T i j
  let ?Ti = T !- i
  assume [rule_format, simp]: i < ar T j < ar ?Ti δ (π ?Ti j 0) < δ ?Ti
    ∀k < ar (?Ti !- j). δ (π ?Ti j (k + 1)) < δ ?Ti
  define X and Y and M where
    [simp]: X = {#δ ?Ti#} and
    [simp]: Y = {#δ (π ?Ti j 0)#} + {#δ (π ?Ti j (k + 1)). k ∈# mset [0 ..< ar (?Ti !- j)]#} and
    [simp]: M ≡ {# δ z. z ∈# mset (nth_drop i (dest_fun T))#}
  have δ (π T i (j + 1)) = HMSet (Y + M)
    by (auto simp: One_nat_def π.simps δ_mk_fun)
  also have Y + M < X + M
    unfolding less_multiset_DM by (rule exI[of _ X], rule exI[of _ Y]) auto
  also have HMSet (X + M) = δ T
    unfolding M_def
    by (subst (2) mk_fun_dest_fun[symmetric, of T], subst (2) id_take_nth_drop[of i dest_fun T])
      (auto simp: δ_mk_fun nth_arg_def nth_drop_def)
  finally show δ (π T i (j + 1)) < δ T by simp
qed

end

end
```