Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

May 26, 2024

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 possibly negative multiplicities), as well as signed hereditary multisets and signed ordinals (e.g., $\omega^2 - 2\omega + 1$).

We refer to the following conference paper for details:

> Jasmin Christian Blanchette, Mathias Fleury, Dmitriy Traytel:
> Nested Multisets, Hereditary Multisets, and Syntactic Ordinals in Isabelle/HOL.
> FSCD 2017: 11:1-11:18
> https://hal.inria.fr/hal-01599176/document

2 More about Multisets

theory *Multiset_More*
  imports
    HOL−Library.Multiset_Order
    HOL−Library.Sublist
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_mset_imp_subset_add_mset*: $A \subseteq\# B \Longrightarrow A \subseteq\# \mathit{add\_mset}\ x\ B$
  ⟨*proof*⟩

lemma *subset_add_mset_notin_subset_mset*: ‹$A \subseteq\# \mathit{add\_mset}\ b\ B \Longrightarrow b \notin\# A \Longrightarrow A \subseteq\# B$›
  ⟨*proof*⟩

lemma *subset_msetE* [*elim!*]: $[\![A \subset\# B; [\![A \subseteq\# B; \neg\, B \subseteq\# A]\!] \Longrightarrow R]\!] \Longrightarrow R$
  ⟨*proof*⟩

lemma *Diff_triv_mset*: $M \cap\# N = \{\#\} \Longrightarrow M - N = M$
  ⟨*proof*⟩

lemma *diff_intersect_sym_diff*: $(A - B) \cap\# (B - A) = \{\#\}$
  ⟨*proof*⟩

lemma *subseq_mset_subseteq_mset*: *subseq xs ys* $\Longrightarrow$ *mset xs* $\subseteq$# *mset ys*
⟨*proof*⟩

lemma *finite_mset_set_inter*:
‹*finite A* $\Longrightarrow$ *finite B* $\Longrightarrow$ *mset_set* (*A* ∩ *B*) = *mset_set A* ∩# *mset_set B*›
⟨*proof*⟩

2.3 Lemmas about Filter and Image

lemma *count_image_mset_ge_count*: *count* (*image_mset f A*) (*f b*) ≥ *count A b*
⟨*proof*⟩

lemma *count_image_mset_inj*:
assumes ‹*inj f*›
shows ‹*count* (*image_mset f M*) (*f x*) = *count M x*›
⟨*proof*⟩

lemma *count_image_mset_le_count_inj_on*:
inj_on f (*set_mset M*) $\Longrightarrow$ *count* (*image_mset f M*) *y* ≤ *count M* (*inv_into* (*set_mset M*) *f y*)
⟨*proof*⟩

lemma *mset_filter_compl*: *mset* (*filter p xs*) + *mset* (*filter* (*Not* ∘ *p*) *xs*) = *mset xs*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *filter_mset_cong*[*fundef_cong*]:
assumes *M* = *M'* ⋀*a*. *a* ∈# *M* $\Longrightarrow$ *P a* = *Q a*
shows *filter_mset P M* = *filter_mset Q M*
⟨*proof*⟩

lemma *image_mset_filter_swap*: *image_mset f* {# *x* ∈# *M*. *P* (*f x*)#} = {# *x* ∈# *image_mset f M*. *P x*#}
⟨*proof*⟩

lemma *image_mset_cong2*:
(⋀*x*. *x* ∈# *M* $\Longrightarrow$ *f x* = *g x*) $\Longrightarrow$ *M* = *N* $\Longrightarrow$ *image_mset f M* = *image_mset g N*
⟨*proof*⟩

lemma *filter_mset_empty_conv*: ‹(*filter_mset P M* = {#}) = (∀ *L*∈#*M*. ¬ *P L*)›
⟨*proof*⟩

lemma *multiset_filter_mono2*: ‹*filter_mset P A* ⊆# *filter_mset Q A* ⟷ (∀ *a*∈#*A*. *P a* ⟶ *Q a*)›
⟨*proof*⟩

lemma *image_filter_cong*:
assumes ‹⋀*C*. *C* ∈# *M* $\Longrightarrow$ *P C* $\Longrightarrow$ *f C* = *g C*›
shows ‹{#*f C*. *C* ∈# {#*C* ∈# *M*. *P C*#}#} = {#*g C* | *C*∈# *M*. *P C*#}›
⟨*proof*⟩

lemma *image_mset_filter_swap2*: ‹{#*C* ∈# {#*P x*. *x* ∈# *D*#}. *Q C* #} = {#*P x*. *x* ∈# {#*C*| *C* ∈# *D*. *Q* (*P C*)#}#}›
⟨*proof*⟩

declare *image_mset_cong2* [*cong*]

lemma *filter_mset_empty_if_finite_and_filter_set_empty*:
assumes
{*x* ∈ *X*. *P x*} = {} **and**
finite X
shows {#*x* ∈# *mset_set X*. *P x*#} = {#}
⟨*proof*⟩

2.4 Lemmas about Sum

lemma *sum_image_mset_sum_map*[*simp*]: *sum_mset* (*image_mset f* (*mset xs*)) = *sum_list* (*map f xs*)
⟨*proof*⟩

lemma *sum_image_mset_mono*:
fixes f :: *'a* ⇒ *'b*::*canonically_ordered_monoid_add*
assumes *sub*: $A \subseteq\# B$
shows $(\sum m \in\# A.\ f\ m) \leq (\sum m \in\# B.\ f\ m)$
⟨*proof*⟩

lemma *sum_image_mset_mono_mem*:
$n \in\# M \Longrightarrow f\ n \leq (\sum m \in\# M.\ f\ m)$ **for** f :: *'a* ⇒ *'b*::*canonically_ordered_monoid_add*
⟨*proof*⟩

lemma *count_sum_mset_if_1_0*: ‹*count M a* = ($\sum x \in\# M.$ *if x = a then 1 else 0*)›
⟨*proof*⟩

lemma *sum_mset_dvd*:
fixes k :: *'a*::*comm_semiring_1_cancel*
assumes $\forall m \in\# M.$ *k dvd f m*
shows *k dvd* $(\sum m \in\# M.\ f\ m)$
⟨*proof*⟩

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*)
⟨*proof*⟩

2.5 Lemmas about Remove

lemma *set_mset_minus_replicate_mset*[*simp*]:
$n \geq$ *count A a* ⟹ *set_mset* (*A* − *replicate_mset n a*) = *set_mset A* − {*a*}
$n <$ *count A a* ⟹ *set_mset* (*A* − *replicate_mset n a*) = *set_mset A*
⟨*proof*⟩

abbreviation *removeAll_mset* :: *'a* ⇒ *'a multiset* ⇒ *'a multiset* **where**
removeAll_mset C M ≡ *M* − *replicate_mset* (*count M C*) *C*

lemma *mset_removeAll*[*simp*, *code*]: *removeAll_mset C* (*mset L*) = *mset* (*removeAll C L*)
⟨*proof*⟩

lemma *removeAll_mset_filter_mset*: *removeAll_mset C M* = *filter_mset* ((≠) *C*) *M*
⟨*proof*⟩

abbreviation *remove1_mset* :: *'a* ⇒ *'a multiset* ⇒ *'a multiset* **where**
remove1_mset C M ≡ *M* − {#*C*#}

lemma *removeAll_subseteq_remove1_mset*: *removeAll_mset x M* ⊆# *remove1_mset x M*
⟨*proof*⟩

lemma *in_remove1_mset_neq*:
assumes *ab*: $a \neq b$
shows *a* ∈# *remove1_mset b C* ⟷ *a* ∈# *C*
⟨*proof*⟩

lemma *size_mset_removeAll_mset_le_iff*: *size* (*removeAll_mset x M*) < *size M* ⟷ *x* ∈# *M*
⟨*proof*⟩

lemma *size_remove1_mset_If*: ‹*size* (*remove1_mset x M*) = *size M* − (*if x* ∈# *M then 1 else 0*)›
⟨*proof*⟩

lemma *size_mset_remove1_mset_le_iff*: *size* (*remove1_mset x M*) < *size M* ⟷ *x* ∈# *M*

⟨*proof*⟩

lemma *remove_1_mset_id_iff_notin*: $remove1\_mset\ a\ M = M \longleftrightarrow a \notin\# M$
⟨*proof*⟩

lemma *id_remove_1_mset_iff_notin*: $M = remove1\_mset\ a\ M \longleftrightarrow a \notin\# M$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *image_filter_ne_mset*[*simp*]:
$image\_mset\ f\ \{\#x \in\# M.\ f\ x \neq y\#\} = removeAll\_mset\ y\ (image\_mset\ f\ M)$
⟨*proof*⟩

lemma *image_mset_remove1_mset_if*:
$image\_mset\ f\ (remove1\_mset\ a\ M) =$
$(if\ a \in\# M\ then\ remove1\_mset\ (f\ a)\ (image\_mset\ f\ M)\ else\ image\_mset\ f\ M)$
⟨*proof*⟩

lemma *filter_mset_neq*: $\{\#x \in\# M.\ x \neq y\#\} = removeAll\_mset\ y\ M$
⟨*proof*⟩

lemma *filter_mset_neq_cond*: $\{\#x \in\# M.\ P\ x \land x \neq y\#\} = removeAll\_mset\ y\ \{\#\ x{\in}\#M.\ P\ x\#\}$
⟨*proof*⟩

lemma *remove1_mset_add_mset_If*:
$remove1\_mset\ L\ (add\_mset\ L'\ C) = (if\ L = L'\ then\ C\ else\ remove1\_mset\ L\ C + \{\#L'\#\})$
⟨*proof*⟩

lemma *minus_remove1_mset_if*:
$A - remove1\_mset\ b\ B = (if\ b \in\# B \land b \in\# A \land count\ A\ b \geq count\ B\ b\ then\ \{\#b\#\} + (A - B)\ else\ A - B)$
⟨*proof*⟩

lemma *add_mset_eq_add_mset_ne*:
$a \neq b \Longrightarrow add\_mset\ a\ A = add\_mset\ b\ B \longleftrightarrow a \in\# B \land b \in\# A \land A = add\_mset\ b\ (B - \{\#a\#\})$
⟨*proof*⟩

lemma *add_mset_eq_add_mset*: ‹$add\_mset\ a\ M = add\_mset\ b\ M' \longleftrightarrow$
$(a = b \land M = M') \lor (a \neq b \land b \in\# M \land add\_mset\ a\ (M - \{\#b\#\}) = M')$›
⟨*proof*⟩

lemma *add_mset_remove_trivial_iff*: ‹$N = add\_mset\ a\ (N - \{\#b\#\}) \longleftrightarrow a \in\# N \land a = b$›
⟨*proof*⟩

lemma *trivial_add_mset_remove_iff*: ‹$add\_mset\ a\ (N - \{\#b\#\}) = N \longleftrightarrow a \in\# N \land a = b$›
⟨*proof*⟩

lemma *remove1_single_empty_iff*[*simp*]: ‹$remove1\_mset\ L\ \{\#L'\#\} = \{\#\} \longleftrightarrow L = L'$›
⟨*proof*⟩

lemma *add_mset_less_imp_less_remove1_mset*:
assumes *xM_lt_N*: $add\_mset\ x\ M < N$
shows $M < remove1\_mset\ x\ N$
⟨*proof*⟩

lemma *remove_diff_multiset*[*simp*]: ‹$x13 \notin\# A \Longrightarrow A - add\_mset\ x13\ B = A - B$›
⟨*proof*⟩

lemma *removeAll_notin*: ‹$a \notin\# A \Longrightarrow$ *removeAll_mset a A* = A›
⟨*proof*⟩

lemma *mset_drop_upto*: ‹*mset* (*drop a N*) = {#*N*!*i*. *i* ∈# *mset_set* {*a*..<*length N*}#}›
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *replicate_mset_subset_iff_lt*[*simp*]: *replicate_mset m x* ⊂# *replicate_mset n x* ⟷ *m* < *n*
⟨*proof*⟩

lemma *replicate_mset_subseteq_iff_le*[*simp*]: *replicate_mset m x* ⊆# *replicate_mset n x* ⟷ *m* ≤ *n*
⟨*proof*⟩

lemma *replicate_mset_lt_iff_lt*[*simp*]: *replicate_mset m x* < *replicate_mset n x* ⟷ *m* < *n*
⟨*proof*⟩

lemma *replicate_mset_le_iff_le*[*simp*]: *replicate_mset m x* ≤ *replicate_mset n x* ⟷ *m* ≤ *n*
⟨*proof*⟩

lemma *replicate_mset_eq_iff*[*simp*]:
replicate_mset m x = *replicate_mset n y* ⟷ *m* = *n* ∧ (*m* ≠ *0* ⟶ *x* = *y*)
⟨*proof*⟩

lemma *replicate_mset_plus*: *replicate_mset* (*a* + *b*) *C* = *replicate_mset a C* + *replicate_mset b C*
⟨*proof*⟩

lemma *mset_replicate_replicate_mset*: *mset* (*replicate n L*) = *replicate_mset n L*
⟨*proof*⟩

lemma *set_mset_single_iff_replicate_mset*: *set_mset U* = {*a*} ⟷ (∃ *n* > *0*. *U* = *replicate_mset n a*)
⟨*proof*⟩

lemma *ex_replicate_mset_if_all_elems_eq*:
assumes ∀ *x* ∈# *M*. *x* = *y*
shows ∃ *n*. *M* = *replicate_mset n y*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mset_set_set_mset_empty_mempty*[*iff*]: *mset_set* (*set_mset D*) = {#} ⟷ *D* = {#}
⟨*proof*⟩

lemma *count_mset_set_le_one*: *count* (*mset_set A*) *x* ≤ *1*
⟨*proof*⟩

lemma *mset_set_set_mset_subseteq*[*simp*]: *mset_set* (*set_mset A*) ⊆# *A*
⟨*proof*⟩

lemma *mset_sorted_list_of_set*[*simp*]: *mset* (*sorted_list_of_set A*) = *mset_set A*
⟨*proof*⟩

lemma *sorted_sorted_list_of_multiset*[*simp*]:
sorted (*sorted_list_of_multiset* (*M* :: ′*a*::*linorder multiset*))
⟨*proof*⟩

lemma *mset_take_subseteq*: *mset* (*take n xs*) ⊆# *mset xs*
⟨*proof*⟩

lemma *sorted_list_of_multiset_eq_Nil*[*simp*]: *sorted_list_of_multiset M* = [] ⟷ *M* = {#}
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *count_remdups_mset_eq_1*: *a* ∈# *remdups_mset A* ⟷ *count* (*remdups_mset A*) *a* = *1*
⟨*proof*⟩

lemma *remdups_mset_empty*[*simp*]: *remdups_mset* {#} = {#}
⟨*proof*⟩

lemma *remdups_mset_singleton*[*simp*]: *remdups_mset* {#*a*#} = {#*a*#}
⟨*proof*⟩

lemma *remdups_mset_eq_empty*[*iff*]: *remdups_mset D* = {#} ⟷ *D* = {#}
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *mset_remdups_remdups_mset*[*simp*]: *mset* (*remdups D*) = *remdups_mset* (*mset D*)
⟨*proof*⟩

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

lemma *count_remdups_mset_If*: ‹*count* (*remdups_mset A*) *a* = (*if a* ∈# *A then 1 else 0*)›
⟨*proof*⟩

lemma *notin_add_mset_remdups_mset*:
‹*a* ∉# *A* ⟹ *add_mset a* (*remdups_mset A*) = *remdups_mset* (*add_mset a A*)›
⟨*proof*⟩

2.9 Repeat Operation

lemma *repeat_mset_compower*: *repeat_mset n A* = (((+) *A*) ^^ *n*) {#}
⟨*proof*⟩

lemma *repeat_mset_prod*: *repeat_mset* (*m* ∗ *n*) *A* = (((+) (*repeat_mset n A*)) ^^ *m*) {#}
⟨*proof*⟩

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 ≡ $\sum_{\#}$ {#{#(*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 $\in\#$.

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*
  ⟨*proof*⟩

lemma *Sigma_msetI* [*intro*!]: ⟦*a* ∈# *A*; *b* ∈# *B a*⟧ ⟹ (*a*, *b*) ∈# *Sigma_mset A B*
  ⟨*proof*⟩

lemma *Sigma_msetE*[*elim*!]: ⟦*c* ∈# *Sigma_mset A B*; ⋀*x y*. ⟦*x* ∈# *A*; *y* ∈# *B x*; *c* = (*x*, *y*)⟧ ⟹ *P*⟧ ⟹ *P*
  ⟨*proof*⟩

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

lemma *Sigma_msetD1*: (*a*, *b*) ∈# *Sigma_mset A B* ⟹ *a* ∈# *A*
  ⟨*proof*⟩

lemma *Sigma_msetD2*: (*a*, *b*) ∈# *Sigma_mset A B* ⟹ *b* ∈# *B a*
  ⟨*proof*⟩

lemma *Sigma_msetE2*: ⟦(*a*, *b*) ∈# *Sigma_mset A B*; ⟦*a* ∈# *A*; *b* ∈# *B a*⟧ ⟹ *P*⟧ ⟹ *P*
  ⟨*proof*⟩

lemma *Sigma_mset_cong*:
  ⟦*A* = *B*; ⋀*x*. *x* ∈# *B* ⟹ *C x* = *D x*⟧ ⟹ (*SIGMAMSET x* ∈# *A*. *C x*) = (*SIGMAMSET x* ∈# *B*. *D x*)
  ⟨*proof*⟩

lemma *count_sum_mset*: *count* ($\sum_{\#}$ *M*) *b* = ($\sum$ *P* ∈# *M*. *count P b*)
  ⟨*proof*⟩

lemma *Sigma_mset_plus_distrib1*[*simp*]: *Sigma_mset* (*A* + *B*) *C* = *Sigma_mset A C* + *Sigma_mset B C*
  ⟨*proof*⟩

lemma *Sigma_mset_plus_distrib2*[*simp*]:
  Sigma_mset A (λ*i*. *B i* + *C i*) = *Sigma_mset A B* + *Sigma_mset A C*
  ⟨*proof*⟩

lemma *Times_mset_single_left*: {#*a*#} ×# *B* = *image_mset* (*Pair a*) *B*
  ⟨*proof*⟩

lemma *Times_mset_single_right*: *A* ×# {#*b*#} = *image_mset* (λ*a*. *Pair a b*) *A*
  ⟨*proof*⟩

lemma *Times_mset_single_single*[*simp*]: {#*a*#} ×# {#*b*#} = {#(*a*, *b*)#}
  ⟨*proof*⟩

lemma *count_image_mset_Pair*:
  count (*image_mset* (*Pair a*) *B*) (*x*, *b*) = (*if x* = *a then count B b else 0*)
  ⟨*proof*⟩

lemma *count_Sigma_mset*: *count* (*Sigma_mset A B*) (*a*, *b*) = *count A a* ∗ *count* (*B a*) *b*
  ⟨*proof*⟩

lemma *Sigma_mset_empty1*[*simp*]: *Sigma_mset* {#} *B* = {#}
  ⟨*proof*⟩

lemma *Sigma_mset_empty2*[*simp*]: *A* ×# {#} = {#}
  ⟨*proof*⟩

lemma *Sigma_mset_mono*:
assumes $A \subseteq\# C$ **and** $\bigwedge x.\ x \in\# A \Longrightarrow B\ x \subseteq\# D\ x$
shows *Sigma_mset* $A\ B \subseteq\#$ *Sigma_mset* $C\ D$
⟨*proof*⟩

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

lemma *mem_Times_mset_iff*: $x \in\# A \times\# B \longleftrightarrow$ *fst* $x \in\# A \wedge$ *snd* $x \in\# B$
⟨*proof*⟩

lemma *Sigma_mset_empty_iff*: (*SIGMAMSET* $i{\in}\#I.\ X\ i) = \{\#\} \longleftrightarrow (\forall\, i{\in}\#I.\ X\ i = \{\#\})$
⟨*proof*⟩

lemma *Times_mset_subset_mset_cancel1*: $x \in\# A \Longrightarrow (A \times\# B \subseteq\# A \times\# C) = (B \subseteq\# C)$
⟨*proof*⟩

lemma *Times_mset_subset_mset_cancel2*: $x \in\# C \Longrightarrow (A \times\# C \subseteq\# B \times\# C) = (A \subseteq\# B)$
⟨*proof*⟩

lemma *Times_mset_eq_cancel2*: $x \in\# C \Longrightarrow (A \times\# C = B \times\# C) = (A = B)$
⟨*proof*⟩

lemma *split_paired_Ball_mset_Sigma_mset*[*simp*]:
$(\forall\, z{\in}\#$*Sigma_mset* $A\ B.\ P\ z) \longleftrightarrow (\forall\, x{\in}\#A.\ \forall\, y{\in}\#B\ x.\ P\ (x,\ y))$
⟨*proof*⟩

lemma *split_paired_Bex_mset_Sigma_mset*[*simp*]:
$(\exists\, z{\in}\#$*Sigma_mset* $A\ B.\ P\ z) \longleftrightarrow (\exists\, x{\in}\#A.\ \exists\, y{\in}\#B\ x.\ P\ (x,\ y))$
⟨*proof*⟩

lemma *sum_mset_if_eq_constant*:
$(\sum x{\in}\#M.$ *if* $a = x$ *then* $(f\ x)$ *else* $0) = (((+)\ (f\ a))\ \widehat{}\widehat{}\ (count\ M\ a))\ 0$
⟨*proof*⟩

lemma *iterate_op_plus*: $(((+)\ k)\ \widehat{}\widehat{}\ m)\ 0 = k * m$
⟨*proof*⟩

lemma *untion_image_mset_Pair_distribute*:
$\sum_{\#}\{\#$*image_mset* (*Pair* x) ($C\ x$). $x \in\# J - I\#\} =$
$\sum_{\#}\{\#$*image_mset* (*Pair* x) ($C\ x$). $x \in\# J\#\} - \sum_{\#}\{\#$*image_mset* (*Pair* x) ($C\ x$). $x \in\# I\#\}$
⟨*proof*⟩

lemma *Sigma_mset_Un_distrib1*: *Sigma_mset* $(I \cup\# J)\ C =$ *Sigma_mset* $I\ C \cup\#$ *Sigma_mset* $J\ C$
⟨*proof*⟩

lemma *Sigma_mset_Un_distrib2*: (*SIGMAMSET* $i{\in}\#I.\ A\ i \cup\# B\ i) =$ *Sigma_mset* $I\ A \cup\#$ *Sigma_mset* $I\ B$
⟨*proof*⟩

lemma *Sigma_mset_Int_distrib1*: *Sigma_mset* $(I \cap\# J)\ C =$ *Sigma_mset* $I\ C \cap\#$ *Sigma_mset* $J\ C$
⟨*proof*⟩

lemma *Sigma_mset_Int_distrib2*: (*SIGMAMSET* $i{\in}\#I.\ A\ i \cap\# B\ i) =$ *Sigma_mset* $I\ A \cap\#$ *Sigma_mset* $I\ B$
⟨*proof*⟩

lemma *Sigma_mset_Diff_distrib1*: *Sigma_mset* $(I - J)\ C =$ *Sigma_mset* $I\ C -$ *Sigma_mset* $J\ C$
⟨*proof*⟩

lemma *Sigma_mset_Diff_distrib2*: (*SIGMAMSET* $i{\in}\#I.\ A\ i - B\ i) =$ *Sigma_mset* $I\ A -$ *Sigma_mset* $I\ B$
⟨*proof*⟩

lemma *Sigma_mset_Union*: *Sigma_mset* $(\sum_{\#}X)\ B = (\sum_{\#}$ (*image_mset* ($\lambda A.$ *Sigma_mset* $A\ B$) X))
⟨*proof*⟩

lemma *Times_mset_Un_distrib1*: $(A \cup\# B) \times\# C = A \times\# C \cup\# B \times\# C$
⟨*proof*⟩

lemma *Times_mset_Int_distrib1*: $(A \cap\# B) \times\# C = A \times\# C \cap\# B \times\# C$
⟨*proof*⟩

lemma *Times_mset_Diff_distrib1*: $(A - B) \times\# C = A \times\# C - B \times\# C$
⟨*proof*⟩

lemma *Times_mset_empty*[*simp*]: $A \times\# B = \{\#\} \longleftrightarrow A = \{\#\} \vee B = \{\#\}$
⟨*proof*⟩

lemma *Times_insert_left*: $A \times\# \mathit{add\_mset}\ x\ B = A \times\# B + \mathit{image\_mset}\ (\lambda a.\ \mathit{Pair}\ a\ x)\ A$
⟨*proof*⟩

lemma *Times_insert_right*: $\mathit{add\_mset}\ a\ A \times\# B = A \times\# B + \mathit{image\_mset}\ (\mathit{Pair}\ a)\ B$
⟨*proof*⟩

lemma *fst_image_mset_times_mset* [*simp*]:
$\mathit{image\_mset}\ \mathit{fst}\ (A \times\# B) = (\mathit{if}\ B = \{\#\}\ \mathit{then}\ \{\#\}\ \mathit{else}\ \mathit{repeat\_mset}\ (\mathit{size}\ B)\ A)$
⟨*proof*⟩

lemma *snd_image_mset_times_mset* [*simp*]:
$\mathit{image\_mset}\ \mathit{snd}\ (A \times\# B) = (\mathit{if}\ A = \{\#\}\ \mathit{then}\ \{\#\}\ \mathit{else}\ \mathit{repeat\_mset}\ (\mathit{size}\ A)\ B)$
⟨*proof*⟩

lemma *product_swap_mset*: $\mathit{image\_mset}\ \mathit{prod.swap}\ (A \times\# B) = B \times\# A$
⟨*proof*⟩

context
begin

qualified definition $\mathit{product\_mset} :: {'a}\ \mathit{multiset} \Rightarrow {'b}\ \mathit{multiset} \Rightarrow ({'a} \times {'b})\ \mathit{multiset}$ **where**
[*code_abbrev*]: $\mathit{product\_mset}\ A\ B = A \times\# B$

lemma *member_product_mset*: $x \in\# \mathit{product\_mset}\ A\ B \longleftrightarrow x \in\# A \times\# B$
⟨*proof*⟩

end

lemma *count_Sigma_mset_abs_def*: $\mathit{count}\ (\mathit{Sigma\_mset}\ A\ B) = (\lambda(a, b) \Rightarrow \mathit{count}\ A\ a * \mathit{count}\ (B\ a)\ b)$
⟨*proof*⟩

lemma *Times_mset_image_mset1*: $\mathit{image\_mset}\ f\ A \times\# B = \mathit{image\_mset}\ (\lambda(a, b).\ (f\ a, b))\ (A \times\# B)$
⟨*proof*⟩

lemma *Times_mset_image_mset2*: $A \times\# \mathit{image\_mset}\ f\ B = \mathit{image\_mset}\ (\lambda(a, b).\ (a, f\ b))\ (A \times\# B)$
⟨*proof*⟩

lemma *sum_le_singleton*: $A \subseteq \{x\} \Longrightarrow \mathit{sum}\ f\ A = (\mathit{if}\ x \in A\ \mathit{then}\ f\ x\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *Times_mset_assoc*: $(A \times\# B) \times\# C = \mathit{image\_mset}\ (\lambda(a, b, c).\ ((a, b), c))\ (A \times\# B \times\# C)$
⟨*proof*⟩

2.11 Transfer Rules

lemma *plus_multiset_transfer*[*transfer_rule*]:
$(\mathit{rel\_fun}\ (\mathit{rel\_mset}\ R)\ (\mathit{rel\_fun}\ (\mathit{rel\_mset}\ R)\ (\mathit{rel\_mset}\ R)))\ (+)\ (+)$
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

2.12.2 Multisets and Lists

lemma *length_sorted_list_of_multiset*[*simp*]: *length* (*sorted_list_of_multiset A*) = *size A*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mset_list_of_mset*[*simp*]: *mset* (*list_of_mset m*) = *m*
⟨*proof*⟩

lemma *length_list_of_mset*[*simp*]: *length* (*list_of_mset A*) = *size A*
⟨*proof*⟩

lemma *range_mset_map*:
assumes *set_mset Ds* ⊆ *range f*
shows *Ds* ∈ *range* (λ*Cl*. *mset* (*map f Cl*))
⟨*proof*⟩

lemma *list_of_mset_empty*[*iff*]: *list_of_mset m* = [] ⟷ *m* = {#}
⟨*proof*⟩

lemma *in_mset_conv_nth*: (*x* ∈# *mset xs*) = (∃ *i*<*length xs*. *xs* ! *i* = *x*)
⟨*proof*⟩

lemma *in_mset_sum_list*:
assumes *L* ∈# *LL*

assumes *LL* ∈ *set Ci*
shows *L* ∈# *sum_list Ci*
⟨*proof*⟩

lemma *in_mset_sum_list2*:
assumes *L* ∈# *sum_list Ci*
obtains *LL* **where**
LL ∈ *set Ci*
L ∈# *LL*
⟨*proof*⟩

lemma *in_mset_sum_list_iff*: *a* ∈# *sum_list* $\mathcal{A}$ ⟷ (∃ *A* ∈ *set* $\mathcal{A}$. *a* ∈# *A*)
⟨*proof*⟩

lemma *subseteq_list_Union_mset*:
assumes *length Ci* = *n*
assumes *length CAi* = *n*
assumes ∀ *i*<*n*. *Ci* ! *i* ⊆# *CAi* ! *i*
shows $\sum_{\#}$ (*mset Ci*) ⊆# $\sum_{\#}$ (*mset CAi*)
⟨*proof*⟩

lemma *same_mset_distinct_iff*:
‹*mset M* = *mset M′* ⟹ *distinct M* ⟷ *distinct M′*›
⟨*proof*⟩

2.12.3 More on Multisets and Functions

lemma *subseteq_mset_size_eql*: *X* ⊆# *Y* ⟹ *size Y* = *size X* ⟹ *X* = *Y*
⟨*proof*⟩

lemma *image_mset_of_subset_list*:
assumes *image_mset* η *C′* = *mset lC*
shows ∃ *qC′*. *map* η *qC′* = *lC* ∧ *mset qC′* = *C′*
⟨*proof*⟩

lemma *image_mset_of_subset*:
assumes *A* ⊆# *image_mset* η *C′*
shows ∃ *A′*. *image_mset* η *A′* = *A* ∧ *A′* ⊆# *C′*
⟨*proof*⟩

lemma *all_the_same*: ∀ *x* ∈# *X*. *x* = *y* ⟹ *card* (*set_mset X*) ≤ *Suc 0*
⟨*proof*⟩

lemma *Melem_subseteq_Union_mset*[*simp*]:
assumes *x* ∈# *T*
shows *x* ⊆# $\sum_{\#}$ *T*
⟨*proof*⟩

lemma *Melem_subset_eq_sum_list*[*simp*]:
assumes *x* ∈# *mset T*
shows *x* ⊆# *sum_list T*
⟨*proof*⟩

lemma *less_subset_eq_Union_mset*[*simp*]:
assumes *i* < *length CAi*
shows *CAi* ! *i* ⊆# $\sum_{\#}$(*mset CAi*)
⟨*proof*⟩

lemma *less_subset_eq_sum_list*[*simp*]:
assumes *i* < *length CAi*
shows *CAi* ! *i* ⊆# *sum_list CAi*
⟨*proof*⟩

2.12.4 More on Multiset Order

lemma *less_multiset_doubletons*:
assumes
$y < t \vee y < s$
$x < t \vee x < s$
shows
$\{\#y,\ x\#\} < \{\#t,\ s\#\}$
⟨*proof*⟩

end

3 Signed (Finite) Multisets

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

unbundle *multiset.lifting*

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*
⟨*proof*⟩

3.2 Basic Operations on Signed Multisets

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

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

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*)
⟨*proof*⟩

lift-definition *plus_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λ(*Mp*, *Mn*) (*Np*, *Nn*). (*Mp* + *Np*, *Mn* + *Nn*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

lift-definition *uminus_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* **is** λ(*Mp*, *Mn*). (*Mn*, *Mp*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

lift-definition *zcount* :: *'a zmultiset* ⇒ *'a* ⇒ *int* **is**
$\lambda(Mp, Mn)\ x.\ int\ (count\ Mp\ x) - int\ (count\ Mn\ x)$
⟨*proof*⟩

lemma *zcount_inject*: $zcount\ M = zcount\ N \longleftrightarrow M = N$
⟨*proof*⟩

lemma *zmultiset_eq_iff*: $M = N \longleftrightarrow (\forall a.\ zcount\ M\ a = zcount\ N\ a)$
⟨*proof*⟩

lemma *zmultiset_eqI*: $(\bigwedge x.\ zcount\ A\ x = zcount\ B\ x) \Longrightarrow A = B$
⟨*proof*⟩

lemma *zcount_uminus*[*simp*]: $zcount\ (-\ A)\ x = -\ zcount\ A\ x$
⟨*proof*⟩

lift-definition *add_zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
$\lambda x\ (Mp, Mn).\ (add\_mset\ x\ Mp, Mn)$
⟨*proof*⟩

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*]: $zcount\ \{\#\}_z\ a = 0$
⟨*proof*⟩

lemma *zcount_add_zmset*[*simp*]:
$zcount\ (add\_zmset\ b\ A)\ a = (if\ b = a\ then\ zcount\ A\ a + 1\ else\ zcount\ A\ a)$
⟨*proof*⟩

lemma *zcount_single*: $zcount\ \{\#b\#\}_z\ a = (if\ b = a\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *add_add_same_iff_zmset*[*simp*]: $add\_zmset\ a\ A = add\_zmset\ a\ B \longleftrightarrow A = B$
⟨*proof*⟩

lemma *add_zmset_commute*: $add\_zmset\ x\ (add\_zmset\ y\ M) = add\_zmset\ y\ (add\_zmset\ x\ M)$
⟨*proof*⟩

lemma
singleton_ne_empty_zmset[*simp*]: $\{\#x\#\}_z \neq \{\#\}_z$ **and**
empty_ne_singleton_zmset[*simp*]: $\{\#\}_z \neq \{\#x\#\}_z$
⟨*proof*⟩

lemma
singleton_ne_uminus_singleton_zmset[*simp*]: $\{\#x\#\}_z \neq -\ \{\#y\#\}_z$ **and**
uminus_singleton_ne_singleton_zmset[*simp*]: $-\ \{\#x\#\}_z \neq \{\#y\#\}_z$
⟨*proof*⟩

3.2.1 Conversion to Set and Membership

definition *set_zmset* :: *'a zmultiset* ⇒ *'a set* **where**
$set\_zmset\ M = \{x.\ zcount\ M\ x \neq 0\}$

abbreviation *elem_zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *bool* **where**
$elem\_zmset\ a\ M \equiv a \in set\_zmset\ M$

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

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
ZMBall :: *pttrn* ⇒ 'a set ⇒ bool ⇒ bool ((3∀∈#$_z$_./ _) [*0*, *0*, *10*] *10*)
ZMBex :: *pttrn* ⇒ 'a set ⇒ bool ⇒ bool ((3∃∈#$_z$_./ _) [*0*, *0*, *10*] *10*)

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

translations
∀ x∈#$_z$A. P ⇌ CONST Signed_Multiset.Ball A (λx. P)
∃ x∈#$_z$A. P ⇌ CONST Signed_Multiset.Bex A (λx. P)

lemma *zcount_eq_zero_iff*: *zcount M x* = *0* ⟷ *x* ∉#$_z$ *M*
⟨*proof*⟩

lemma *not_in_iff_zmset*: *x* ∉#$_z$ *M* ⟷ *zcount M x* = *0*
⟨*proof*⟩

lemma *zcount_ne_zero_iff*[*simp*]: *zcount M x* ≠ *0* ⟷ *x* ∈#$_z$ *M*
⟨*proof*⟩

lemma *zcount_inI*:
assumes *zcount M x* = *0* ⟹ *False*
shows *x* ∈#$_z$ *M*
⟨*proof*⟩

lemma *set_zmset_empty*[*simp*]: *set_zmset* {#}$_z$ = {}
⟨*proof*⟩

lemma *set_zmset_single*: *set_zmset* {#b#}$_z$ = {b}
⟨*proof*⟩

lemma *set_zmset_eq_empty_iff*[*simp*]: *set_zmset M* = {} ⟷ *M* = {#}$_z$
⟨*proof*⟩

lemma *finite_count_ne*: *finite* {x. *count M x* ≠ *count N x*}
⟨*proof*⟩

lemma *finite_set_zmset*[*iff*]: *finite* (*set_zmset* M)
⟨*proof*⟩

lemma *zmultiset_nonemptyE*[*elim*]:
assumes $A \neq \{\#\}_z$
obtains x **where** $x \in\#_z A$
⟨*proof*⟩

3.2.2 Union

lemma *zcount_union*[*simp*]: *zcount* $(M + N)$ a = *zcount* M a + *zcount* N a
⟨*proof*⟩

lemma *union_add_left_zmset*[*simp*]: *add_zmset* a $A + B$ = *add_zmset* a $(A + B)$
⟨*proof*⟩

lemma *union_zmset_add_zmset_right*[*simp*]: A + *add_zmset* a B = *add_zmset* a $(A + B)$
⟨*proof*⟩

lemma *add_zmset_add_single*: ‹*add_zmset* a $A = A + \{\#a\#\}_z$›
⟨*proof*⟩

3.2.3 Difference

lemma *zcount_diff*[*simp*]: *zcount* $(M - N)$ a = *zcount* M a − *zcount* N a
⟨*proof*⟩

lemma *add_zmset_diff_bothsides*: ‹*add_zmset* a M − *add_zmset* a $A = M - A$›
⟨*proof*⟩

lemma *in_diff_zcount*: $a \in\#_z M - N \longleftrightarrow$ *zcount* N $a \neq$ *zcount* M a
⟨*proof*⟩

lemma *diff_add_zmset*:
fixes M N Q :: *'a zmultiset*
shows $M - (N + Q) = M - N - Q$
⟨*proof*⟩

lemma *insert_Diff_zmset*[*simp*]: *add_zmset* x $(M - \{\#x\#\}_z) = M$
⟨*proof*⟩

lemma *diff_union_swap_zmset*: *add_zmset* b $(M - \{\#a\#\}_z)$ = *add_zmset* b $M - \{\#a\#\}_z$
⟨*proof*⟩

lemma *diff_add_zmset_swap*[*simp*]: *add_zmset* b $M - A$ = *add_zmset* b $(M - A)$
⟨*proof*⟩

lemma *diff_diff_add_zmset*[*simp*]: $(M$ :: *'a zmultiset*$) - N - P = M - (N + P)$
⟨*proof*⟩

lemma *zmset_add*[*elim?*]:
obtains B **where** A = *add_zmset* a B
⟨*proof*⟩

3.2.4 Equality of Signed Multisets

lemma *single_eq_single_zmset*[*simp*]: $\{\#a\#\}_z = \{\#b\#\}_z \longleftrightarrow a = b$
⟨*proof*⟩

lemma *multi_self_add_other_not_self_zmset*[*simp*]: M = *add_zmset* x $M \longleftrightarrow$ *False*
⟨*proof*⟩

lemma *add_zmset_remove_trivial*: ‹*add_zmset* x $M - \{\#x\#\}_z = M$›

⟨*proof*⟩

lemma *diff_single_eq_union_zmset*: $M - \{\#x\#\}_z = N \longleftrightarrow M = add\_zmset\ x\ N$
⟨*proof*⟩

lemma *union_single_eq_diff_zmset*: $add\_zmset\ x\ M = N \Longrightarrow M = N - \{\#x\#\}_z$
⟨*proof*⟩

lemma *add_zmset_eq_conv_diff*:
$add\_zmset\ a\ M = add\_zmset\ b\ N \longleftrightarrow$
$M = N \wedge a = b \vee M = add\_zmset\ b\ (N - \{\#a\#\}_z) \wedge N = add\_zmset\ a\ (M - \{\#b\#\}_z)$
⟨*proof*⟩

lemma *add_zmset_eq_conv_ex*:
$(add\_zmset\ a\ M = add\_zmset\ b\ N) =$
$(M = N \wedge a = b \vee (\exists K.\ M = add\_zmset\ b\ K \wedge N = add\_zmset\ a\ K))$
⟨*proof*⟩

lemma *multi_member_split*: $\exists A.\ M = add\_zmset\ x\ A$
⟨*proof*⟩

3.3 Conversions from and to Multisets

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

lemma *zmset_of_inject*[*simp*]: $zmset\_of\ M = zmset\_of\ N \longleftrightarrow M = N$
⟨*proof*⟩

lemma *zmset_of_empty*[*simp*]: $zmset\_of\ \{\#\} = \{\#\}_z$
⟨*proof*⟩

lemma *zmset_of_add_mset*[*simp*]: $zmset\_of\ (add\_mset\ x\ M) = add\_zmset\ x\ (zmset\_of\ M)$
⟨*proof*⟩

lemma *zcount_of_mset*[*simp*]: $zcount\ (zmset\_of\ M)\ x = int\ (count\ M\ x)$
⟨*proof*⟩

lemma *zmset_of_plus*: $zmset\_of\ (M + N) = zmset\_of\ M + zmset\_of\ N$
⟨*proof*⟩

lift-definition *mset_pos* :: $'a\ zmultiset \Rightarrow 'a\ multiset$ **is** $\lambda(Mp,\ Mn).\ count\ (Mp - Mn)$
⟨*proof*⟩

lift-definition *mset_neg* :: $'a\ zmultiset \Rightarrow 'a\ multiset$ **is** $\lambda(Mp,\ Mn).\ count\ (Mn - Mp)$
⟨*proof*⟩

lemma
zmset_of_inverse[*simp*]: $mset\_pos\ (zmset\_of\ M) = M$ **and**
minus_zmset_of_inverse[*simp*]: $mset\_neg\ (-\ zmset\_of\ M) = M$
⟨*proof*⟩

lemma *neg_zmset_pos*[*simp*]: $mset\_neg\ (zmset\_of\ M) = \{\#\}$
⟨*proof*⟩

lemma
count_mset_pos[*simp*]: $count\ (mset\_pos\ M)\ x = nat\ (zcount\ M\ x)$ **and**
count_mset_neg[*simp*]: $count\ (mset\_neg\ M)\ x = nat\ (-\ zcount\ M\ x)$
⟨*proof*⟩

lemma
mset_pos_empty[*simp*]: $mset\_pos\ \{\#\}_z = \{\#\}$ **and**
mset_neg_empty[*simp*]: $mset\_neg\ \{\#\}_z = \{\#\}$
⟨*proof*⟩

lemma
mset_pos_singleton[*simp*]: *mset_pos* $\{\#x\#\}_z$ = $\{\#x\#\}$ **and**
mset_neg_singleton[*simp*]: *mset_neg* $\{\#x\#\}_z$ = $\{\#\}$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *mset_pos_uminus*[*simp*]: *mset_pos* (− A) = *mset_neg* A
⟨*proof*⟩

lemma *mset_neg_uminus*[*simp*]: *mset_neg* (− A) = *mset_pos* A
⟨*proof*⟩

lemma *mset_pos_plus*[*simp*]:
mset_pos (A + B) = (*mset_pos* A − *mset_neg* B) + (*mset_pos* B − *mset_neg* A)
⟨*proof*⟩

lemma *mset_neg_plus*[*simp*]:
mset_neg (A + B) = (*mset_neg* A − *mset_pos* B) + (*mset_neg* B − *mset_pos* A)
⟨*proof*⟩

lemma *mset_pos_diff*[*simp*]:
mset_pos (A − B) = (*mset_pos* A − *mset_pos* B) + (*mset_neg* B − *mset_neg* A)
⟨*proof*⟩

lemma *mset_neg_diff*[*simp*]:
mset_neg (A − B) = (*mset_neg* A − *mset_neg* B) + (*mset_pos* B − *mset_pos* A)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *decompose_zmset_of2*:
obtains A B C **where**
M = *zmset_of* A + C **and**
N = *zmset_of* B + C
⟨*proof*⟩

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**
$A \subset\#_z B \longleftrightarrow A \subseteq\#_z B \land A \neq B$

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

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

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* (+) (−) ($\subseteq\#_z$) ($\subset\#_z$)
⟨*proof*⟩

interpretation *subset_zmset*:
ordered_ab_semigroup_monoid_add_imp_le (+) *0* (−) ($\subseteq\#_z$) ($\subset\#_z$)
⟨*proof*⟩

lemma *zmset_subset_eqI*: ($\bigwedge a.$ *zcount A a* $\leq$ *zcount B a*) $\Longrightarrow$ $A \subseteq\#_z B$
⟨*proof*⟩

lemma *zmset_subset_eq_zcount*: $A \subseteq\#_z B \Longrightarrow$ *zcount A a* $\leq$ *zcount B a*
⟨*proof*⟩

lemma *zmset_subset_eq_add_zmset_cancel*: ‹*add_zmset a A* $\subseteq\#_z$ *add_zmset a B* $\longleftrightarrow$ $A \subseteq\#_z B$›
⟨*proof*⟩

lemma *zmset_subset_eq_zmultiset_union_diff_commute*:
$A - B + C = A + C - B$ **for** *A B C* :: *'a zmultiset*
⟨*proof*⟩

lemma *zmset_subset_eq_insertD*: *add_zmset x A* $\subseteq\#_z B \Longrightarrow A \subset\#_z B$
⟨*proof*⟩

lemma *zmset_subset_insertD*: *add_zmset x A* $\subset\#_z B \Longrightarrow A \subset\#_z B$
⟨*proof*⟩

lemma *subset_eq_diff_conv_zmset*: $A - C \subseteq\#_z B \longleftrightarrow A \subseteq\#_z B + C$
⟨*proof*⟩

lemma *multi_psub_of_add_self_zmset*[*simp*]: $A \subset\#_z$ *add_zmset x A*
⟨*proof*⟩

lemma *multi_psub_self_zmset*: $A \subset\#_z A$ = *False*
⟨*proof*⟩

lemma *zmset_subset_add_zmset*[*simp*]: *add_zmset x N* $\subset\#_z$ *add_zmset x M* $\longleftrightarrow N \subset\#_z M$
⟨*proof*⟩

lemma *zmset_of_subseteq_iff*[*simp*]: *zmset_of M* $\subseteq\#_z$ *zmset_of N* $\longleftrightarrow M \subseteq\# N$
⟨*proof*⟩

lemma *zmset_of_subset_iff*[*simp*]: *zmset_of M* $\subset\#_z$ *zmset_of N* $\longleftrightarrow M \subset\# N$
⟨*proof*⟩

lemma
mset_pos_supset: $A \subseteq\#_z$ *zmset_of* (*mset_pos A*) **and**
mset_neg_supset: $- A \subseteq\#_z$ *zmset_of* (*mset_neg A*)
⟨*proof*⟩

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** $A \subset\# B$
⟨*proof*⟩

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** $A \subseteq\# B$
  ⟨*proof*⟩

3.3.2 Subset is an Order

interpretation *subset_zmset*: *order* $(\subseteq\#_z)$ $(\subset\#_z)$
  ⟨*proof*⟩

3.4 Replicate and Repeat Operations

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

lemma *replicate_zmset_0*[*simp*]: $replicate\_zmset\ 0\ x = \{\#\}_z$
  ⟨*proof*⟩

lemma *replicate_zmset_Suc*[*simp*]: $replicate\_zmset\ (Suc\ n)\ x = add\_zmset\ x\ (replicate\_zmset\ n\ x)$
  ⟨*proof*⟩

lemma *count_replicate_zmset*[*simp*]:
  $zcount\ (replicate\_zmset\ n\ x)\ y = (if\ y = x\ then\ of\_nat\ n\ else\ 0)$
  ⟨*proof*⟩

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

lemma *count_repeat_zmset*[*simp*]: $zcount\ (repeat\_zmset\ i\ A)\ a = of\_nat\ i * zcount\ A\ a$
  ⟨*proof*⟩

lemma *repeat_zmset_right*[*simp*]: $repeat\_zmset\ a\ (repeat\_zmset\ b\ A) = repeat\_zmset\ (a * b)\ A$
  ⟨*proof*⟩

lemma *left_diff_repeat_zmset_distrib'*:
  ‹$i \geq j \Longrightarrow repeat\_zmset\ (i - j)\ u = repeat\_zmset\ i\ u - repeat\_zmset\ j\ u$›
  ⟨*proof*⟩

lemma *left_add_mult_distrib_zmset*:
  $repeat\_zmset\ i\ u + (repeat\_zmset\ j\ u + k) = repeat\_zmset\ (i{+}j)\ u + k$
  ⟨*proof*⟩

lemma *repeat_zmset_distrib*: $repeat\_zmset\ (m + n)\ A = repeat\_zmset\ m\ A + repeat\_zmset\ n\ A$
  ⟨*proof*⟩

lemma *repeat_zmset_distrib2*[*simp*]:
  $repeat\_zmset\ n\ (A + B) = repeat\_zmset\ n\ A + repeat\_zmset\ n\ B$
  ⟨*proof*⟩

lemma *repeat_zmset_replicate_zmset*[*simp*]: $repeat\_zmset\ n\ \{\#a\#\}_z = replicate\_zmset\ n\ a$
  ⟨*proof*⟩

lemma *repeat_zmset_distrib_add_zmset*[*simp*]:
  $repeat\_zmset\ n\ (add\_zmset\ a\ A) = replicate\_zmset\ n\ a + repeat\_zmset\ n\ A$
  ⟨*proof*⟩

lemma *repeat_zmset_empty*[*simp*]: $repeat\_zmset\ n\ \{\#\}_z = \{\#\}_z$
  ⟨*proof*⟩

3.4.1 Filter (with Comprehension Syntax)

lift-definition *filter_zmset* :: (*'a* ⇒ *bool*) ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**

$\lambda P\ (Mp,\ Mn).\ (filter\_mset\ P\ Mp,\ filter\_mset\ P\ Mn)$
⟨*proof*⟩

syntax (*ASCII*)
$\_ZMCollect :: pttrn \Rightarrow 'a\ zmultiset \Rightarrow bool \Rightarrow 'a\ zmultiset\ ((1\{\#\_ :\#z\ \_./\ \_\#\}))$
syntax
$\_ZMCollect :: pttrn \Rightarrow 'a\ zmultiset \Rightarrow bool \Rightarrow 'a\ zmultiset\ ((1\{\#\_ \in\#_z\ \_./\ \_\#\}))$
translations
$\{\#x \in\#_z\ M.\ P\#\} == CONST\ filter\_zmset\ (\lambda x.\ P)\ M$

lemma *count_filter_zmset*[*simp*]:
$zcount\ (filter\_zmset\ P\ M)\ a = (if\ P\ a\ then\ zcount\ M\ a\ else\ 0)$
⟨*proof*⟩

lemma *filter_empty_zmset*[*simp*]: $filter\_zmset\ P\ \{\#\}_z = \{\#\}_z$
⟨*proof*⟩

lemma *filter_single_zmset*: $filter\_zmset\ P\ \{\#x\#\}_z = (if\ P\ x\ then\ \{\#x\#\}_z\ else\ \{\#\}_z)$
⟨*proof*⟩

lemma *filter_union_zmset*[*simp*]: $filter\_zmset\ P\ (M + N) = filter\_zmset\ P\ M + filter\_zmset\ P\ N$
⟨*proof*⟩

lemma *filter_diff_zmset*[*simp*]: $filter\_zmset\ P\ (M - N) = filter\_zmset\ P\ M - filter\_zmset\ P\ N$
⟨*proof*⟩

lemma *filter_add_zmset*[*simp*]:
$filter\_zmset\ P\ (add\_zmset\ x\ A) =$
$(if\ P\ x\ then\ add\_zmset\ x\ (filter\_zmset\ P\ A)\ else\ filter\_zmset\ P\ A)$
⟨*proof*⟩

lemma *zmultiset_filter_mono*:
assumes $A \subseteq\#_z B$
shows $filter\_zmset\ f\ A \subseteq\#_z filter\_zmset\ f\ B$
⟨*proof*⟩

lemma *filter_filter_zmset*: $filter\_zmset\ P\ (filter\_zmset\ Q\ M) = \{\#x \in\#_z\ M.\ Q\ x \wedge P\ x\#\}$
⟨*proof*⟩

lemma
filter_zmset_True[*simp*]: $\{\#y \in\#_z\ M.\ True\#\} = M$ **and**
filter_zmset_False[*simp*]: $\{\#y \in\#_z\ M.\ False\#\} = \{\#\}_z$
⟨*proof*⟩

3.5 Uncategorized

lemma *multi_drop_mem_not_eq_zmset*: $B - \{\#c\#\}_z \neq B$
⟨*proof*⟩

lemma *zmultiset_partition*: $M = \{\#x \in\#_z\ M.\ P\ x\ \#\} + \{\#x \in\#_z\ M.\ \neg\ P\ x\#\}$
⟨*proof*⟩

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* ⇒ *'a zmultiset* ⇒ *bool* **is**
$\lambda(Mp, Mn)$ (Np, Nn). $Mp + Nn < Mn + Np$
⟨*proof*⟩

definition *less_eq_zmultiset* :: *'a zmultiset* ⇒ *'a zmultiset* ⇒ *bool* **where**
less_eq_zmultiset M' $M \longleftrightarrow M' < M \lor M' = M$

instance
⟨*proof*⟩

end

instance *zmultiset* :: (*preorder*) *ordered_cancel_comm_monoid_add*
⟨*proof*⟩

instance *zmultiset* :: (*preorder*) *ordered_ab_group_add*
⟨*proof*⟩

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*
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *zmset_of_less*: *zmset_of* M < *zmset_of* $N \longleftrightarrow M < N$
⟨*proof*⟩

lemma *zmset_of_le*: *zmset_of* $M \leq$ *zmset_of* $N \longleftrightarrow M \leq N$
⟨*proof*⟩

instance *zmultiset* :: (*preorder*) *ordered_ab_semigroup_add*
⟨*proof*⟩

lemma *uminus_add_conv_diff_mset*[*cancelation_simproc_pre*]: ‹$-a + b = b - a$› **for** a :: ‹*'a zmultiset*›
⟨*proof*⟩

lemma *uminus_add_add_uminus*[*cancelation_simproc_pre*]: ‹$b - a + c = b + c - a$› **for** a :: ‹*'a zmultiset*›
⟨*proof*⟩

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

lemma *repeat_zmset_iterate_add*: ‹*repeat_zmset* n M = *iterate_add* n M›
⟨*proof*⟩

declare *repeat_zmset_iterate_add*[*cancelation_simproc_pre*]

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

⟨*ML*⟩

lemma *zmset_subseteq_add_iff1*:
$\langle j \leq i \Longrightarrow (repeat\_zmset\ i\ u + m \subseteq\#_z\ repeat\_zmset\ j\ u + n) = (repeat\_zmset\ (i - j)\ u + m \subseteq\#_z\ n)\rangle$
⟨*proof*⟩

lemma *zmset_subseteq_add_iff2*:
$\langle i \leq j \Longrightarrow (repeat\_zmset\ i\ u + m \subseteq\#_z\ repeat\_zmset\ j\ u + n) = (m \subseteq\#_z\ repeat\_zmset\ (j - i)\ u + n)\rangle$
⟨*proof*⟩

lemma *zmset_subset_add_iff1*:
$\langle j \leq i \Longrightarrow (repeat\_zmset\ i\ u + m \subset\#_z\ repeat\_zmset\ j\ u + n) = (repeat\_zmset\ (i - j)\ u + m \subset\#_z\ n)\rangle$
⟨*proof*⟩

lemma *zmset_subset_add_iff2*:
$\langle i \leq j \Longrightarrow (repeat\_zmset\ i\ u + m \subset\#_z\ repeat\_zmset\ j\ u + n) = (m \subset\#_z\ repeat\_zmset\ (j - i)\ u + n)\rangle$
⟨*proof*⟩

⟨*ML*⟩

instance *zmultiset* :: (*preorder*) *ordered_ab_semigroup_add_imp_le*
⟨*proof*⟩

⟨*ML*⟩

instance *zmultiset* :: (*linorder*) *linordered_cancel_ab_semigroup_add*
⟨*proof*⟩

lemma *less_mset_zmsetE*:
assumes $M < N$
obtains $A\ B\ C$ **where**
$M = zmset\_of\ A + C$ **and** $N = zmset\_of\ B + C$ **and** $A < B$
⟨*proof*⟩

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** $A \leq B$
⟨*proof*⟩

lemma *subset_eq_imp_le_zmset*: $M \subseteq\#_z\ N \Longrightarrow M \leq N$
⟨*proof*⟩

lemma *subset_imp_less_zmset*: $M \subset\#_z\ N \Longrightarrow M < N$
⟨*proof*⟩

lemma *lt_imp_ex_zcount_lt*:
assumes *m_lt_n*: $M < N$
shows $\exists y.\ zcount\ M\ y < zcount\ N\ y$
⟨*proof*⟩

instance *zmultiset* :: (*preorder*) *no_top*
⟨*proof*⟩

lifting-update *multiset.lifting*
lifting-forget *multiset.lifting*

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*⟩

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*: $x \in\#\ M \Longrightarrow |x| < |MSet\ M|$
⟨*proof*⟩

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**
$less\_multiset\_ext_{DM}\ R\ M\ N \longleftrightarrow$
$(\exists X\ Y.\ X \neq \{\#\} \wedge X \subseteq\#\ N \wedge M = (N - X) + Y \wedge (\forall k.\ k \in\#\ Y \longrightarrow (\exists a.\ a \in\#\ X \wedge R\ k\ a)))$

lemma $less\_multiset\_ext_{DM}\_imp\_mult$:
assumes
N_A: $set\_mset\ N \subseteq A$ **and** *M_A*: $set\_mset\ M \subseteq A$ **and** *less*: $less\_multiset\_ext_{DM}\ R\ M\ N$
shows $(M,\ N) \in mult\ \{(x,\ y).\ x \in A \wedge y \in A \wedge R\ x\ y\}$
⟨*proof*⟩

lemma $mult\_imp\_less\_multiset\_ext_{DM}$:
assumes
N_A: $set\_mset\ N \subseteq A$ **and** *M_A*: $set\_mset\ M \subseteq A$ **and**
trans: $\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ R\ x\ y \longrightarrow R\ y\ z \longrightarrow R\ x\ z$ **and**
in_mult: $(M,\ N) \in mult\ \{(x,\ y).\ x \in A \wedge y \in A \wedge R\ x\ y\}$
shows $less\_multiset\_ext_{DM}\ R\ M\ N$
⟨*proof*⟩

lemma $less\_multiset\_ext_{DM}\_iff\_mult$:
assumes
N_A: $set\_mset\ N \subseteq A$ **and** *M_A*: $set\_mset\ M \subseteq A$ **and**
trans: $\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ R\ x\ y \longrightarrow R\ y\ z \longrightarrow R\ x\ z$
shows $less\_multiset\_ext_{DM}\ R\ M\ N \longleftrightarrow (M,\ N) \in mult\ \{(x,\ y).\ x \in A \wedge y \in A \wedge R\ x\ y\}$
⟨*proof*⟩

instantiation *nmultiset* :: (*preorder*) *preorder*
begin

lemma $less\_multiset\_ext_{DM}\_cong$[*fundef_cong*]:
$(\bigwedge X\ Y\ k\ a.\ X \neq \{\#\} \Longrightarrow X \subseteq\#\ N \Longrightarrow M = (N - X) + Y \Longrightarrow k \in\#\ Y \Longrightarrow R\ k\ a = S\ k\ a) \Longrightarrow$
$less\_multiset\_ext_{DM}\ R\ M\ N = less\_multiset\_ext_{DM}\ S\ M\ N$
⟨*proof*⟩

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*
⟨*proof*⟩
termination
⟨*proof*⟩

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*⟩

lemma *irrefl_less_nmultiset*:
fixes *X* :: *'a nmultiset*
shows *X* < *X* ⟹ *False*
⟨*proof*⟩

lemma *antisym_less_nmultiset*:
fixes *X Y* :: *'a nmultiset*
shows *X* < *Y* ⟹ *Y* < *X* ⟹ *False*
⟨*proof*⟩

definition *less_eq_nmultiset* :: *'a nmultiset* ⇒ *'a nmultiset* ⇒ *bool* **where**
less_eq_nmultiset X Y = (*X* < *Y* ∨ *X* = *Y*)

instance
⟨*proof*⟩

lemma $less\_multiset\_ext_{DM}\_less$: $less\_multiset\_ext_{DM}$ (<) = (<)
⟨*proof*⟩

end

instantiation *nmultiset* :: (*order*) *order*
begin

instance
⟨*proof*⟩

end

instantiation *nmultiset* :: (*linorder*) *linorder*
begin

lemma *total_less_nmultiset*:
fixes *X Y* :: *'a nmultiset*
shows ¬ *X* < *Y* ⟹ *Y* ≠ *X* ⟹ *Y* < *X*
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *less_depth_nmset_imp_less_nmultiset*: |*X*| < |*Y*| ⟹ *X* < *Y*
⟨*proof*⟩

lemma *less_nmultiset_imp_le_depth_nmset*: $X < Y \Longrightarrow |X| \leq |Y|$
⟨*proof*⟩

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*⟩

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

lemma *depth_nmset_eq_0*[*simp*]: $|X| = 0 \longleftrightarrow (X = MSet\ \{\#\} \vee (\exists x.\ X = Elem\ x))$
⟨*proof*⟩

lemma *depth_nmset_eq_Suc*[*simp*]: $|X| = Suc\ n \longleftrightarrow$
$(\exists N.\ X = MSet\ N \wedge (\exists Y \in\#\ N.\ |Y| = n) \wedge (\forall Y \in\#\ N.\ |Y| \leq n))$
⟨*proof*⟩

lemma *wf_less_nmultiset_depth*:
wf $\{(X :: 'a\ nmultiset, Y).\ |X| = i \wedge |Y| = i \wedge X < Y\}$
⟨*proof*⟩

lemma *wf_less_nmultiset*: *wf* $\{(X :: 'a\ nmultiset, Y :: 'a\ nmultiset).\ X < Y\}$ (**is** *wf ?R*)
⟨*proof*⟩

instance ⟨*proof*⟩

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* $\longleftrightarrow$ *A* = *B*
⟨*proof*⟩

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*⟩

setup-lifting *type_definition_hmultiset*

lemma *HMSet_alt*: *HMSet* = *Abs_hmultiset o MSet o image_mset Rep_hmultiset*
⟨*proof*⟩

lemma *HMSet_transfer*[*transfer_rule*]: *rel_fun* (*rel_mset pcr_hmultiset*) *pcr_hmultiset MSet HMSet*
⟨*proof*⟩

5.2 Restriction of Dershowitz and Manna's Nested Multiset Order

instantiation *hmultiset* :: *linorder*
begin

lift-definition *less_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** (<) ⟨*proof*⟩
lift-definition *less_eq_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** (≤) ⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *less_HMSet_iff_less_multiset_ext*$_{DM}$: *HMSet M* < *HMSet N* ⟷ *less_multiset_ext*$_{DM}$ (<) *M N*
⟨*proof*⟩

lemma *hmsetmset_less*[*simp*]: *hmsetmset M* < *hmsetmset N* ⟷ *M* < *N*
⟨*proof*⟩

lemma *hmsetmset_le*[*simp*]: *hmsetmset M* ≤ *hmsetmset N* ⟷ *M* ≤ *N*
⟨*proof*⟩

lemma *wf_less_hmultiset*: *wf* {(*X* :: *hmultiset*, *Y* :: *hmultiset*). *X* < *Y*}
⟨*proof*⟩

instance *hmultiset* :: *wellorder*
⟨*proof*⟩

lemma *HMSet_less*[*simp*]: *HMSet M* < *HMSet N* ⟷ *M* < *N*
⟨*proof*⟩

lemma *HMSet_le*[*simp*]: *HMSet M* ≤ *HMSet N* ⟷ *M* ≤ *N*
⟨*proof*⟩

lemma *mem_imp_less_HMSet*: *k* ∈# *L* ⟹ *k* < *HMSet L*
⟨*proof*⟩

lemma *mem_hmsetmset_imp_less*: *M* ∈# *hmsetmset N* ⟹ *M* < *N*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *hmsetmset_0*[*simp*]: *hmsetmset 0* = {#}
⟨*proof*⟩

lemma
HMSet_eq_0_iff[*simp*]: *HMSet m* = *0* ⟷ *m* = {#} **and**
zero_eq_HMSet[*simp*]: *0* = *HMSet m* ⟷ *m* = {#}
⟨*proof*⟩

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
⟨*proof*⟩

end

lemma *HMSet_plus*: *HMSet* (*A* + *B*) = *HMSet A* + *HMSet B*
⟨*proof*⟩

lemma *HMSet_diff*: *HMSet* (*A* − *B*) = *HMSet A* − *HMSet B*
⟨*proof*⟩

lemma *hmsetmset_plus*: *hmsetmset* (*M* + *N*) = *hmsetmset M* + *hmsetmset N*
⟨*proof*⟩

lemma *hmsetmset_diff*: *hmsetmset* (*M* − *N*) = *hmsetmset M* − *hmsetmset N*
⟨*proof*⟩

lemma *diff_diff_add_hmset*[*simp*]: *a* − *b* − *c* = *a* − (*b* + *c*) **for** *a b c* :: *hmultiset*
⟨*proof*⟩

instance *hmultiset* :: *comm_monoid_diff*
⟨*proof*⟩

⟨*ML*⟩

instance *hmultiset* :: *ordered_cancel_comm_monoid_add*
⟨*proof*⟩

instance *hmultiset* :: *ordered_ab_semigroup_add_imp_le*
⟨*proof*⟩

instantiation *hmultiset* :: *order_bot*
begin

definition *bot_hmultiset* :: *hmultiset* **where**
bot_hmultiset = *0*

instance
⟨*proof*⟩

end

instance *hmultiset* :: *no_top*
⟨*proof*⟩

lemma *le_minus_plus_same_hmset*: *m* ≤ *m* − *n* + *n* **for** *m n* :: *hmultiset*
⟨*proof*⟩

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
⟨*proof*⟩

end

5.5 Inequalities

lemma *zero_le_hmset*[*simp*]: *0* ≤ *M* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma
le_add1_hmset: *n* ≤ *n* + *m* **and**
le_add2_hmset: *n* ≤ *m* + *n* **for** *n* :: *hmultiset*
⟨*proof*⟩

lemma *le_zero_eq_hmset*[*simp*]: *M* ≤ *0* ⟷ *M* = *0* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma *not_less_zero_hmset*[*simp*]: ¬ *M* < *0* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma *not_gr_zero_hmset*[*simp*]: ¬ *0* < *M* ⟷ *M* = *0* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma *zero_less_iff_neq_zero_hmset*: *0* < *M* ⟷ *M* ≠ *0* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma *zero_less_HMSet_iff*[*simp*]: *0* < *HMSet M* ⟷ *M* ≠ {#}
⟨*proof*⟩

lemma *gr_zeroI_hmset*: (*M* = *0* ⟹ *False*) ⟹ *0* < *M* **for** *M* :: *hmultiset*
⟨*proof*⟩

lemma *gr_implies_not_zero_hmset*: *M* < *N* ⟹ *N* ≠ *0* **for** *M N* :: *hmultiset*
⟨*proof*⟩

lemma *add_eq_0_iff_both_eq_0_hmset*[*simp*]: *M* + *N* = *0* ⟷ *M* = *0* ∧ *N* = *0* **for** *M N* :: *hmultiset*
⟨*proof*⟩

lemma *trans_less_add1_hmset*: *i* < *j* ⟹ *i* < *j* + *m* **for** *i j m* :: *hmultiset*
⟨*proof*⟩

lemma *trans_less_add2_hmset*: *i* < *j* ⟹ *i* < *m* + *j* **for** *i j m* :: *hmultiset*
⟨*proof*⟩

lemma *trans_le_add1_hmset*: *i* ≤ *j* ⟹ *i* ≤ *j* + *m* **for** *i j m* :: *hmultiset*
⟨*proof*⟩

lemma *trans_le_add2_hmset*: *i* ≤ *j* ⟹ *i* ≤ *m* + *j* **for** *i j m* :: *hmultiset*
⟨*proof*⟩

lemma *diff_le_self_hmset*: *m* − *n* ≤ *m* **for** *m n* :: *hmultiset*
⟨*proof*⟩

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*

⟨*proof*⟩

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** (<) ⟨*proof*⟩
lift-definition *less_eq_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *bool* **is** (≤) ⟨*proof*⟩

instance
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *zhmset_of_less*: *zhmset_of M* < *zhmset_of N* ⟷ *M* < *N*
⟨*proof*⟩

lemma *zhmset_of_le*: *zhmset_of M* ≤ *zhmset_of N* ⟷ *M* ≤ *N*
⟨*proof*⟩

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** $\{\#\}_z$ ⟨*proof*⟩

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

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

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*
⟨*proof*⟩

lemma *hmsetmset_0*: *hmsetmset 0* = {#}
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *zhmset_of_0*: *zhmset_of 0* = *0*
⟨*proof*⟩

lemma *hmset_pos_plus*:
hmset_pos (*A* + *B*) = (*hmset_pos A* − *hmset_neg B*) + (*hmset_pos B* − *hmset_neg A*)
⟨*proof*⟩

lemma *hmset_neg_plus*:
hmset_neg (*A* + *B*) = (*hmset_neg A* − *hmset_pos B*) + (*hmset_neg B* − *hmset_pos A*)
⟨*proof*⟩

lemma *zhmset_pos_neg_partition*: *M* = *zhmset_of* (*hmset_pos M*) − *zhmset_of* (*hmset_neg M*)
⟨*proof*⟩

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

lemma *zhmset_neg_as_pos*: *zhmset_of* (*hmset_neg M*) = *zhmset_of* (*hmset_pos M*) − *M*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *zhmset_of_sum_list*: *zhmset_of* (*sum_list Ms*) = *sum_list* (*map zhmset_of Ms*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

instantiation *zhmultiset* :: *ab_group_add*
begin

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

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

instance
⟨*proof*⟩

end

6.5 Infimum and Supremum

instance *zhmultiset* :: *ordered_cancel_comm_monoid_add*
⟨*proof*⟩

instance *zhmultiset* :: *ordered_ab_group_add*
⟨*proof*⟩

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
⟨*proof*⟩

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*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *empty_times_left_hmset*[*simp*]: *HMSet* {#} ∗ *M* = *0*
⟨*proof*⟩

lemma *empty_times_right_hmset*[*simp*]: *M* ∗ *HMSet* {#} = *0*
⟨*proof*⟩

lemma *singleton_times_left_hmset*[*simp*]: $\omega\hat{}M * N = HMSet\ (image\_mset\ ((+)\ M)\ (hmsetmset\ N))$
⟨*proof*⟩

lemma *singleton_times_right_hmset*[*simp*]: $N * \omega\hat{}M = HMSet\ (image\_mset\ ((+)\ M)\ (hmsetmset\ N))$
⟨*proof*⟩

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′*
⟨*proof*⟩

lemma *plus_hmultiset_transfer*[*transfer_rule*]:
(*rel_fun pcr_hmultiset* (*rel_fun pcr_hmultiset pcr_hmultiset*)) *plus_nmultiset* (+)
⟨*proof*⟩

lemma *Times_mset_monoL*:
assumes *less*: $M < N$ **and** *Z_nemp*: $Z \neq \{\#\}$
shows $M \times\# Z < N \times\# Z$
⟨*proof*⟩

lemma *times_hmultiset_monoL*:
$a < b \Longrightarrow 0 < c \Longrightarrow a * c < b * c$ **for** *a b c* :: *hmultiset*
⟨*proof*⟩

instance *hmultiset* :: *linordered_semiring_strict*
⟨*proof*⟩

lemma *mult_le_mono1_hmset*: $i \le j \Longrightarrow i * k \le j * k$ **for** *i j k* :: *hmultiset*
⟨*proof*⟩

lemma *mult_le_mono2_hmset*: $i \le j \Longrightarrow k * i \le k * j$ **for** *i j k* :: *hmultiset*
⟨*proof*⟩

lemma *mult_le_mono_hmset*: $i \le j \Longrightarrow k \le l \Longrightarrow i * k \le j * l$ **for** *i j k l* :: *hmultiset*
⟨*proof*⟩

lemma *less_iff_add1_le_hmset*: $m < n \longleftrightarrow m + 1 \le n$ **for** *m n* :: *hmultiset*
⟨*proof*⟩

lemma *zero_less_iff_1_le_hmset*: $0 < n \longleftrightarrow 1 \le n$ **for** *n* :: *hmultiset*
⟨*proof*⟩

lemma *less_add_1_iff_le_hmset*: $m < n + 1 \longleftrightarrow m \le n$ **for** *m n* :: *hmultiset*
⟨*proof*⟩

instance *hmultiset* :: *ordered_cancel_comm_semiring*
⟨*proof*⟩

instance *hmultiset* :: *zero_less_one*
⟨*proof*⟩

instance *hmultiset* :: *linordered_semiring_1_strict*
⟨*proof*⟩

instance *hmultiset* :: *bounded_lattice_bot*
⟨*proof*⟩

instance *hmultiset* :: *linordered_nonzero_semiring*

⟨proof⟩

instance *hmultiset* :: *semiring_no_zero_divisors*
⟨proof⟩

lemma *lt_1_iff_eq_0_hmset*: $M < 1 \longleftrightarrow M = 0$ **for** *M* :: *hmultiset*
⟨proof⟩

lemma *zero_less_mult_iff_hmset*[*simp*]: $0 < m * n \longleftrightarrow 0 < m \wedge 0 < n$ **for** *m n* :: *hmultiset*
⟨proof⟩

lemma *one_le_mult_iff_hmset*[*simp*]: $1 \leq m * n \longleftrightarrow 1 \leq m \wedge 1 \leq n$ **for** *m n* :: *hmultiset*
⟨proof⟩

lemma *mult_less_cancel2_hmset*[*simp*]: $m * k < n * k \longleftrightarrow 0 < k \wedge m < n$ **for** *k m n* :: *hmultiset*
⟨proof⟩

lemma *mult_less_cancel1_hmset*[*simp*]: $k * m < k * n \longleftrightarrow 0 < k \wedge m < n$ **for** *k m n* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel1_hmset*[*simp*]: $k * m \leq k * n \longleftrightarrow (0 < k \longrightarrow m \leq n)$ **for** *k m n* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel2_hmset*[*simp*]: $m * k \leq n * k \longleftrightarrow (0 < k \longrightarrow m \leq n)$ **for** *k m n* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel_left1_hmset*: $y > 0 \Longrightarrow x \leq x * y$ **for** *x y* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel_left2_hmset*: $y \leq 1 \Longrightarrow x * y \leq x$ **for** *x y* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel_right1_hmset*: $y > 0 \Longrightarrow x \leq y * x$ **for** *x y* :: *hmultiset*
⟨proof⟩

lemma *mult_le_cancel_right2_hmset*: $y \leq 1 \Longrightarrow y * x \leq x$ **for** *x y* :: *hmultiset*
⟨proof⟩

lemma *le_square_hmset*: $m \leq m * m$ **for** *m* :: *hmultiset*
⟨proof⟩

lemma *le_cube_hmset*: $m \leq m * (m * m)$ **for** *m* :: *hmultiset*
⟨proof⟩

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*
⟨proof⟩

lemma *gt_0_lt_mult_gt_1_hmset*:
fixes *m n* :: *hmultiset*
assumes $m > 0$ **and** $n > 1$
shows $m < m * n$
⟨proof⟩

instance *hmultiset* :: *linordered_comm_semiring_strict*
⟨proof⟩

7.3 Embedding of Natural Numbers

lemma *of_nat_hmset*: *of_nat n* = *HMSet* (*replicate_mset n 0*)
⟨proof⟩

lemma *of_nat_inject_hmset*[*simp*]: (*of_nat m* :: *hmultiset*) = *of_nat n* ⟷ *m* = *n*

⟨*proof*⟩

lemma *of_nat_minus_hmset*: *of_nat* (*m* − *n*) = (*of_nat m* :: *hmultiset*) − *of_nat n*
⟨*proof*⟩

lemma *plus_of_nat_plus_of_nat_hmset*:
k + *of_nat m* + *of_nat n* = *k* + *of_nat* (*m* + *n*) **for** *k* :: *hmultiset*
⟨*proof*⟩

lemma *plus_of_nat_minus_of_nat_hmset*:
fixes *k* :: *hmultiset*
assumes *n* ≤ *m*
shows *k* + *of_nat m* − *of_nat n* = *k* + *of_nat* (*m* − *n*)
⟨*proof*⟩

lemma *of_nat_lt_ω*[*simp*]: *of_nat n* < *ω*
⟨*proof*⟩

lemma *of_nat_ne_ω*[*simp*]: *of_nat n* ≠ *ω*
⟨*proof*⟩

lemma *of_nat_less_hmset*[*simp*]: (*of_nat M* :: *hmultiset*) < *of_nat N* ⟷ *M* < *N*
⟨*proof*⟩

lemma *of_nat_le_hmset*[*simp*]: (*of_nat M* :: *hmultiset*) ≤ *of_nat N* ⟷ *M* ≤ *N*
⟨*proof*⟩

lemma *of_nat_times_ω_exp*: *of_nat n* ∗ *ω*^*m* = *HMSet* (*replicate_mset n m*)
⟨*proof*⟩

lemma *ω_exp_times_of_nat*: *ω*^*m* ∗ *of_nat n* = *HMSet* (*replicate_mset n m*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *hmset_of_enat_1*[*simp*]: *hmset_of_enat 1* = *1*
⟨*proof*⟩

lemma *hmset_of_enat_of_nat*[*simp*]: *hmset_of_enat* (*of_nat n*) = *of_nat n*
⟨*proof*⟩

lemma *hmset_of_enat_numeral*[*simp*]: *hmset_of_enat* (*numeral n*) = *numeral n*
⟨*proof*⟩

lemma *hmset_of_enat_le_ω*[*simp*]: *hmset_of_enat n* ≤ *ω*
⟨*proof*⟩

lemma *hmset_of_enat_eq_ω_iff*[*simp*]: *hmset_of_enat n* = *ω* ⟷ *n* = ∞
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *head_ω_eq_0_iff*[*simp*]: *head_ω m = 0* ⟷ *m = 0*
⟨*proof*⟩

lemma *head_ω_0*[*simp*]: *head_ω 0 = 0*
⟨*proof*⟩

lemma *head_ω_1*[*simp*]: *head_ω 1 = 1*
⟨*proof*⟩

lemma *head_ω_of_nat*[*simp*]: *head_ω* (*of_nat n*) = (*if n = 0 then 0 else 1*)
⟨*proof*⟩

lemma *head_ω_numeral*[*simp*]: *head_ω* (*numeral n*) = *1*
⟨*proof*⟩

lemma *head_ω_ω*[*simp*]: *head_ω ω = ω*
⟨*proof*⟩

lemma *le_imp_head_ω_le*:
assumes *m_le_n*: *m* ≤ *n*
shows *head_ω m* ≤ *head_ω n*
⟨*proof*⟩

lemma *head_ω_lt_imp_lt*: *head_ω m* < *head_ω n* ⟹ *m* < *n*
⟨*proof*⟩

lemma *head_ω_plus*[*simp*]: *head_ω* (*m* + *n*) = *sup* (*head_ω m*) (*head_ω n*)
⟨*proof*⟩

lemma *head_ω_times*[*simp*]: *head_ω* (*m* ∗ *n*) = *head_ω m* ∗ *head_ω n*
⟨*proof*⟩

7.6 More Inequalities and Some Equalities

lemma *zero_lt_ω*[*simp*]: *0* < ω
⟨*proof*⟩

lemma *one_lt_ω*[*simp*]: *1* < ω
⟨*proof*⟩

lemma *numeral_lt_ω*[*simp*]: *numeral n* < ω
⟨*proof*⟩

lemma *one_le_ω*[*simp*]: *1* ≤ ω
⟨*proof*⟩

lemma *of_nat_le_ω*[*simp*]: *of_nat n* ≤ ω
⟨*proof*⟩

lemma *numeral_le_ω*[*simp*]: *numeral n* ≤ ω
⟨*proof*⟩

lemma *not_ω_lt_1*[*simp*]: ¬ ω < *1*
⟨*proof*⟩

lemma *not_ω_lt_of_nat*[*simp*]: ¬ ω < *of_nat n*
⟨*proof*⟩

lemma *not_ω_lt_numeral*[*simp*]: ¬ ω < *numeral n*
⟨*proof*⟩

lemma *not_ω_le_1*[*simp*]: ¬ ω ≤ *1*
⟨*proof*⟩

lemma $not\_\omega\_le\_of\_nat[simp]$: $\neg\ \omega \le of\_nat\ n$
⟨proof⟩

lemma $not\_\omega\_le\_numeral[simp]$: $\neg\ \omega \le numeral\ n$
⟨proof⟩

lemma $zero\_ne\_\omega[simp]$: $0 \ne \omega$
⟨proof⟩

lemma $one\_ne\_\omega[simp]$: $1 \ne \omega$
⟨proof⟩

lemma $numeral\_ne\_\omega[simp]$: $numeral\ n \ne \omega$
⟨proof⟩

lemma
$\omega\_ne\_0[simp]$: $\omega \ne 0$ **and**
$\omega\_ne\_1[simp]$: $\omega \ne 1$ **and**
$\omega\_ne\_of\_nat[simp]$: $\omega \ne of\_nat\ m$ **and**
$\omega\_ne\_numeral[simp]$: $\omega \ne numeral\ n$
⟨proof⟩

lemma
$hmset\_of\_enat\_inject[simp]$: $hmset\_of\_enat\ m = hmset\_of\_enat\ n \longleftrightarrow m = n$ **and**
$hmset\_of\_enat\_less[simp]$: $hmset\_of\_enat\ m < hmset\_of\_enat\ n \longleftrightarrow m < n$ **and**
$hmset\_of\_enat\_le[simp]$: $hmset\_of\_enat\ m \le hmset\_of\_enat\ n \longleftrightarrow m \le n$
⟨proof⟩

lemma $lt\_\omega\_imp\_ex\_of\_nat$:
assumes M_lt_ω: $M < \omega$
shows $\exists\, n.\ M = of\_nat\ n$
⟨proof⟩

lemma $le\_\omega\_imp\_ex\_hmset\_of\_enat$:
assumes M_le_ω: $M \le \omega$
shows $\exists\, n.\ M = hmset\_of\_enat\ n$
⟨proof⟩

lemma $lt\_\omega\_lt\_\omega\_imp\_times\_lt\_\omega$: $M < \omega \Longrightarrow N < \omega \Longrightarrow M * N < \omega$
⟨proof⟩

lemma $times\_\omega\_minus\_of\_nat[simp]$: $m * \omega - of\_nat\ n = m * \omega$
⟨proof⟩

lemma $times\_\omega\_minus\_numeral[simp]$: $m * \omega - numeral\ n = m * \omega$
⟨proof⟩

lemma $\omega\_minus\_of\_nat[simp]$: $\omega - of\_nat\ n = \omega$
⟨proof⟩

lemma $\omega\_minus\_1[simp]$: $\omega - 1 = \omega$
⟨proof⟩

lemma $\omega\_minus\_numeral[simp]$: $\omega - numeral\ n = \omega$
⟨proof⟩

lemma $hmset\_of\_enat\_minus\_enat[simp]$: $hmset\_of\_enat\ (m - enat\ n) = hmset\_of\_enat\ m - of\_nat\ n$
⟨proof⟩

lemma $of\_nat\_lt\_hmset\_of\_enat\_iff$: $of\_nat\ m < hmset\_of\_enat\ n \longleftrightarrow enat\ m < n$
⟨proof⟩

lemma *of_nat_le_hmset_of_enat_iff*: $of\_nat\ m \leq hmset\_of\_enat\ n \longleftrightarrow enat\ m \leq n$
⟨*proof*⟩

lemma *hmset_of_enat_lt_iff_ne_infinity*: $hmset\_of\_enat\ x < \omega \longleftrightarrow x \neq \infty$
⟨*proof*⟩

lemma *minus_diff_sym_hmset*: $m - (m - n) = n - (n - m)$ **for** $m\ n :: hmultiset$
⟨*proof*⟩

lemma *diff_plus_sym_hmset*: $(c - b) + b = (b - c) + c$ **for** $b\ c :: hmultiset$
⟨*proof*⟩

lemma *times_diff_plus_sym_hmset*: $a * (c - b) + a * b = a * (b - c) + a * c$ **for** $a\ b\ c :: hmultiset$
⟨*proof*⟩

lemma *times_of_nat_minus_left*:
$(of\_nat\ m - of\_nat\ n) * l = of\_nat\ m * l - of\_nat\ n * l$ **for** $l :: hmultiset$
⟨*proof*⟩

lemma *times_of_nat_minus_right*:
$l * (of\_nat\ m - of\_nat\ n) = l * of\_nat\ m - l * of\_nat\ n$ **for** $l :: hmultiset$
⟨*proof*⟩

lemma *lt_ω_imp_times_minus_left*: $m < \omega \Longrightarrow n < \omega \Longrightarrow (m - n) * l = m * l - n * l$
⟨*proof*⟩

lemma *lt_ω_imp_times_minus_right*: $m < \omega \Longrightarrow n < \omega \Longrightarrow l * (m - n) = l * m - l * n$
⟨*proof*⟩

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*⟩

lemma *hmset_pair_decompose_less*:
assumes *m1_lt_m2*: $m1 < m2$
shows $\exists k\ n1\ n2.\ m1 = k + n1 \wedge m2 = k + n2 \wedge head\_\omega\ n1 < head\_\omega\ n2$
⟨*proof*⟩

lemma *hmset_pair_decompose_less_eq*:
assumes $m1 \leq m2$
shows $\exists k\ n1\ n2.\ m1 = k + n1 \wedge m2 = k + n2 \wedge (head\_\omega\ n1 < head\_\omega\ n2 \vee n1 = 0 \wedge n2 = 0)$
⟨*proof*⟩

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*⟩

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 $Ap\ An\ Bp\ Bn\ Cp\ Cn\ Dp\ Dn :: hmultiset$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *offset_hmset_numeral[simp]: offset_hmset (numeral n) = numeral n*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sum_coefs_gt_0: sum_coefs M > 0 ⟷ M > 0*
⟨*proof*⟩

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 $\alpha 1\ \alpha 2\ \beta 1\ \beta 2\ \gamma\ \delta$ :: *hmultiset*
assumes
$\alpha\beta 2\gamma$*_lt_*$\alpha\beta 1\gamma$: $\alpha 2 + \beta 2 * \gamma < \alpha 1 + \beta 1 * \gamma$ **and**
$\beta 2$*_le_*$\beta 1$: $\beta 2 \leq \beta 1$ **and**
γ*_lt_*δ: $\gamma < \delta$
shows $\alpha 2 + \beta 2 * \delta < \alpha 1 + \beta 1 * \delta$
⟨*proof*⟩

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#}$_z$ ⟨*proof*⟩

abbreviation ω_z :: *zhmultiset* **where**
ω_z ≡ ω_z^1

lemma ω_z*_as_*ω: ω_z = *zhmset_of* ω
⟨*proof*⟩

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))) ⟨*proof*⟩

lemmas *zhmsetmset_times = times_zhmultiset.rep_eq*

instance
⟨*proof*⟩

end

lemma *zhmset_of_1*: *zhmset_of 1 = 1*
⟨*proof*⟩

lemma *zhmset_of_times*: *zhmset_of (A * B) = zhmset_of A * zhmset_of B*
⟨*proof*⟩

lemma *zhmset_of_prod_list*:
zhmset_of (prod_list Ms) = prod_list (map zhmset_of Ms)
⟨*proof*⟩

8.2 Embedding of Natural Numbers

lemma *of_nat_zhmset*: *of_nat n = zhmset_of (of_nat n)*
⟨*proof*⟩

lemma *of_nat_inject_zhmset*[*simp*]: (*of_nat m* :: *zhmultiset*) = *of_nat n* ⟷ *m = n*
⟨*proof*⟩

lemma *plus_of_nat_plus_of_nat_zhmset*:
k + of_nat m + of_nat n = k + of_nat (m + n) **for** *k* :: *zhmultiset*
⟨*proof*⟩

lemma *plus_of_nat_minus_of_nat_zhmset*:
fixes *k* :: *zhmultiset*
assumes $n \leq m$
shows *k + of_nat m − of_nat n = k + of_nat (m − n)*
⟨*proof*⟩

lemma *of_nat_lt_*ω_z[*simp*]: *of_nat n* < ω_z
⟨*proof*⟩

lemma *of_nat_ne_*ω_z[*simp*]: *of_nat n* ≠ ω_z
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *zhmset_of_enat_1*[*simp*]: *zhmset_of_enat 1 = 1*
⟨*proof*⟩

lemma *zhmset_of_enat_of_nat*[*simp*]: *zhmset_of_enat (of_nat n) = of_nat n*
⟨*proof*⟩

lemma *zhmset_of_enat_numeral*[*simp*]: *zhmset_of_enat (numeral n) = numeral n*
⟨*proof*⟩

lemma *zhmset_of_enat_le_*ω_z[*simp*]: *zhmset_of_enat n* ≤ ω_z
⟨*proof*⟩

lemma *zhmset_of_enat_eq_*ω_z*_iff*[*simp*]: *zhmset_of_enat n* = ω_z ⟷ *n* = ∞
⟨*proof*⟩

8.4 Inequalities and Some (Dis)equalities

instance *zhmultiset* :: *zero_less_one*
⟨*proof*⟩

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*⟩

instance
⟨*proof*⟩

end

lemma *le_zhmset_of_pos*: *M* ≤ *zhmset_of* (*hmset_pos M*)
⟨*proof*⟩

lemma *minus_zhmset_of_pos_le*: − *zhmset_of* (*hmset_neg M*) ≤ *M*
⟨*proof*⟩

lemma *zhmset_of_nonneg*[*simp*]: *zhmset_of M* ≥ *0*
⟨*proof*⟩

lemma
fixes *n* :: *zhmultiset*
assumes *0* ≤ *m*
shows
le_add1_hmset: *n* ≤ *n* + *m* **and**
le_add2_hmset: *n* ≤ *m* + *n*
⟨*proof*⟩

lemma *less_iff_add1_le_zhmset*: *m* < *n* ⟷ *m* + *1* ≤ *n* **for** *m n* :: *zhmultiset*
⟨*proof*⟩

lemma *gt_0_lt_mult_gt_1_zhmset*:
fixes *m n* :: *zhmultiset*
assumes *m* > *0* **and** *n* > *1*
shows *m* < *m* ∗ *n*
⟨*proof*⟩

lemma *zero_less_iff_1_le_zhmset*: *0* < *n* ⟷ *1* ≤ *n* **for** *n* :: *zhmultiset*
⟨*proof*⟩

lemma *less_add_1_iff_le_hmset*: *m* < *n* + *1* ⟷ *m* ≤ *n* **for** *m n* :: *zhmultiset*
⟨*proof*⟩

lemma *nonneg_le_mult_right_mono_zhmset*:
fixes *x y z* :: *zhmultiset*
assumes *x*: *0* ≤ *x* **and** *y*: *0* < *y* **and** *z*: *x* ≤ *z*
shows *x* ≤ *y* ∗ *z*
⟨*proof*⟩

instance *hmultiset* :: *ordered_cancel_comm_semiring*

⟨*proof*⟩

instance *hmultiset* :: *linordered_semiring_1_strict*
⟨*proof*⟩

instance *hmultiset* :: *bounded_lattice_bot*
⟨*proof*⟩

instance *hmultiset* :: *zero_less_one*
⟨*proof*⟩

instance *hmultiset* :: *linordered_nonzero_semiring*
⟨*proof*⟩

instance *hmultiset* :: *semiring_no_zero_divisors*
⟨*proof*⟩

lemma *zero_lt_*ω_z[*simp*]: $0 < \omega_z$
⟨*proof*⟩

lemma *one_lt_*ω[*simp*]: $1 < \omega_z$
⟨*proof*⟩

lemma *numeral_lt_*ω_z[*simp*]: *numeral n* $< \omega_z$
⟨*proof*⟩

lemma *one_le_*ω_z[*simp*]: $1 \leq \omega_z$
⟨*proof*⟩

lemma *of_nat_le_*ω_z[*simp*]: *of_nat n* $\leq \omega_z$
⟨*proof*⟩

lemma *numeral_le_*ω_z[*simp*]: *numeral n* $\leq \omega_z$
⟨*proof*⟩

lemma *not_*ω_z*_lt_1*[*simp*]: $\neg\ \omega_z < 1$
⟨*proof*⟩

lemma *not_*ω_z*_lt_of_nat*[*simp*]: $\neg\ \omega_z <$ *of_nat n*
⟨*proof*⟩

lemma *not_*ω_z*_lt_numeral*[*simp*]: $\neg\ \omega_z <$ *numeral n*
⟨*proof*⟩

lemma *not_*ω_z*_le_1*[*simp*]: $\neg\ \omega_z \leq 1$
⟨*proof*⟩

lemma *not_*ω_z*_le_of_nat*[*simp*]: $\neg\ \omega_z \leq$ *of_nat n*
⟨*proof*⟩

lemma *not_*ω_z*_le_numeral*[*simp*]: $\neg\ \omega_z \leq$ *numeral n*
⟨*proof*⟩

lemma *zero_ne_*ω_z[*simp*]: $0 \neq \omega_z$
⟨*proof*⟩

lemma *one_ne_*ω_z[*simp*]: $1 \neq \omega_z$
⟨*proof*⟩

lemma *numeral_ne_*ω_z[*simp*]: *numeral n* $\neq \omega_z$
⟨*proof*⟩

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
⟨proof⟩

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
⟨proof⟩

lemma of_nat_lt_zhmset_of_enat_iff: of_nat m < zhmset_of_enat n ⟷ enat m < n
⟨proof⟩

lemma of_nat_le_zhmset_of_enat_iff: of_nat m ≤ zhmset_of_enat n ⟷ enat m ≤ n
⟨proof⟩

lemma zhmset_of_enat_lt_iff_ne_infinity: zhmset_of_enat x < ω_z ⟷ x ≠ ∞
⟨proof⟩

8.5 An Example

A new proof of $[\![?\alpha 2.0 + ?\beta 2.0 * ?\gamma < ?\alpha 1.0 + ?\beta 1.0 * ?\gamma; ?\beta 2.0 \leq ?\beta 1.0; ?\gamma < ?\delta]\!] \Longrightarrow ?\alpha 2.0 + ?\beta 2.0 * ?\delta < ?\alpha 1.0 + ?\beta 1.0 * ?\delta$:

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⟩

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
⟨proof⟩

end

lemma insort_bot[simp]: insort bot xs = bot # xs **for** xs :: 'a::{order_bot,linorder} list
⟨proof⟩

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*
⟨*proof*⟩

lemma *from_cnf_0_iff*[*simp*]: *from_cnf ks* = *0* ⟷ *ks* = []
⟨*proof*⟩

lemma *from_cnf_append*[*simp*]: *from_cnf* (*ks* @ *ls*) = *from_cnf ks* + *from_cnf ls*
⟨*proof*⟩

lemma *subseq_from_cnf_less_eq*: *Sublist.subseq ks ls* ⟹ *from_cnf ks* ≤ *from_cnf ls*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *ordinal_of_hmset_suc*[*simp*]: *ordinal_of_hmset* (*k* + *1*) = *ordinal_of_hmset k* + *1*
⟨*proof*⟩

lemma *ordinal_of_hmset_1*[*simp*]: *ordinal_of_hmset 1* = *1*
⟨*proof*⟩

lemma *ordinal_of_hmset_ω*[*simp*]: *ordinal_of_hmset* ω_h = ω
⟨*proof*⟩

lemma *ordinal_of_hmset_singleton*[*simp*]: *ordinal_of_hmset* (ω^*k*) = ω ∗∗ *ordinal_of_hmset k*
⟨*proof*⟩

lemma *ordinal_of_hmset_iff*[*simp*]: *ordinal_of_hmset k* = *0* ⟷ *k* = *0*
⟨*proof*⟩

lemma *less_imp_ordinal_of_hmset_less*: *k* < *l* ⟹ *ordinal_of_hmset k* < *ordinal_of_hmset l*
⟨*proof*⟩

lemma *ordinal_of_hmset_less*[*simp*]: *ordinal_of_hmset k* < *ordinal_of_hmset l* ⟷ *k* < *l*
⟨*proof*⟩

end

10 Termination of McCarthy's 91 Function

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

lemma *funpow_rec*: *f* ^^ *n* = (*if n* = *0 then id else f* ∘ *f* ^^ (*n* − *1*))
⟨*proof*⟩

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 :: *int* ⇒ *int* **where**
  f x = (if x > 100 then x − 10 else 91)

definition τ :: *nat* ⇒ *int* ⇒ *int multiset* **where**
  τ n z = mset (map (λi. (f ^^ nat i) z) [0..int n − 1])

function g :: *nat* ⇒ *int* ⇒ *int* **where**
  g n z = (if n = 0 then z else if z > 100 then g (n − 1) (z − 10) else g (n + 1) (z + 11))
  ⟨*proof*⟩
termination
⟨*proof*⟩

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*
  ⟨*proof*⟩

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))
  ⟨*proof*⟩
termination
  ⟨*proof*⟩

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))
  ⟨*proof*⟩
termination

⟨*proof*⟩

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*: $m\ div\ n * n \leq m$ **for** $m\ n :: nat$
⟨*proof*⟩

lemma *power_div_same_base*:
$b\ \hat{}\ y \neq 0 \Longrightarrow x \geq y \Longrightarrow b\ \hat{}\ x\ div\ b\ \hat{}\ y = b\ \hat{}\ (x - y)$ **for** $b :: 'a::semidom\_divide$
⟨*proof*⟩

12.2 Hereditary and Nonhereditary Base-n Systems

context
fixes *base* :: *nat*
assumes *base_ge_2*: $base \geq 2$
begin

inductive *well_base* :: *'a multiset* ⇒ *bool* **where**
$(\forall n.\ count\ M\ n < base) \Longrightarrow well\_base\ M$

lemma *well_base_filter*: $well\_base\ M \Longrightarrow well\_base\ \{\#m \in\#\ M.\ p\ m\#\}$
⟨*proof*⟩

lemma *well_base_image_inj*: $well\_base\ M \Longrightarrow inj\_on\ f\ (set\_mset\ M) \Longrightarrow well\_base\ (image\_mset\ f\ M)$
⟨*proof*⟩

lemma *well_base_bound*:
assumes
well_base M **and**
$\forall m \in\#\ M.\ m < n$
shows $(\sum m \in\#\ M.\ base\ \hat{}\ m) < base\ \hat{}\ n$
⟨*proof*⟩

inductive $well\_base_h$ :: *hmultiset* ⇒ *bool* **where**
$(\forall N \in\#\ hmsetmset\ M.\ well\_base_h\ N) \Longrightarrow well\_base\ (hmsetmset\ M) \Longrightarrow well\_base_h\ M$

lemma $well\_base_h\_mono\_hmset$: $well\_base_h\ M \Longrightarrow hmsetmset\ N \subseteq\#\ hmsetmset\ M \Longrightarrow well\_base_h\ N$
⟨*proof*⟩

lemma $well\_base_h\_imp\_well\_base$: $well\_base_h\ M \Longrightarrow well\_base\ (hmsetmset\ M)$
⟨*proof*⟩

12.3 Encoding of Natural Numbers into Ordinals

function *encode* :: *nat* ⇒ *nat* ⇒ *hmultiset* **where**
$encode\ e\ n =$
$(if\ n = 0\ then\ 0\ else\ of\_nat\ (n\ mod\ base) * \omega\hat{}(encode\ 0\ e) + encode\ (e + 1)\ (n\ div\ base))$
⟨*proof*⟩

termination
⟨*proof*⟩

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

lemma *encode_0*[*simp*]: $encode\ e\ 0 = 0$
⟨*proof*⟩

lemma *encode_Suc*:
$encode\ e\ (Suc\ n) = of\_nat\ (Suc\ n\ mod\ base) * \omega\hat{}(encode\ 0\ e) + encode\ (e + 1)\ (Suc\ n\ div\ base)$
⟨*proof*⟩

lemma *encode_0_iff*: $encode\ e\ n = 0 \longleftrightarrow n = 0$
⟨*proof*⟩

lemma *encode_Suc_exp*: $encode\ (Suc\ e)\ n = encode\ e\ (base * n)$
⟨*proof*⟩

lemma *encode_exp_0*: $encode\ e\ n = encode\ 0\ (base\ \hat{}\ e * n)$
⟨*proof*⟩

lemma *mem_hmsetmset_encodeD*: $M \in\# hmsetmset\ (encode\ e\ n) \Longrightarrow \exists\, e' \geq e.\ M = encode\ 0\ e'$
⟨*proof*⟩

lemma *less_imp_encode_less*: $n < p \Longrightarrow encode\ e\ n < encode\ e\ p$
⟨*proof*⟩

inductive $aligned_e$ :: $nat \Rightarrow hmultiset \Rightarrow bool$ **where**
$(\forall\, m \in\# hmsetmset\ M.\ m \geq encode\ 0\ e) \Longrightarrow aligned_e\ e\ M$

lemma $aligned_e$*_encode*: $aligned_e\ e\ (encode\ e\ M)$
⟨*proof*⟩

lemma *well_base*$_h$*_encode*: $well\_base_h\ (encode\ e\ n)$
⟨*proof*⟩

12.4 Decoding of Natural Numbers from Ordinals

primrec *decode* :: $nat \Rightarrow hmultiset \Rightarrow nat$ **where**
$decode\ e\ (HMSet\ M) = (\sum m \in\# M.\ base\ \hat{}\ decode\ 0\ m)\ div\ base\ \hat{}\ e$

lemma *decode_unfold*: $decode\ e\ M = (\sum m \in\# hmsetmset\ M.\ base\ \hat{}\ decode\ 0\ m)\ div\ base\ \hat{}\ e$
⟨*proof*⟩

lemma *decode_0*[*simp*]: $decode\ e\ 0 = 0$
⟨*proof*⟩

inductive $aligned_d$ :: $nat \Rightarrow hmultiset \Rightarrow bool$ **where**
$(\forall\, m \in\# hmsetmset\ M.\ decode\ 0\ m \geq e) \Longrightarrow aligned_d\ e\ M$

lemma $aligned_d$*_0*[*simp*]: $aligned_d\ 0\ M$
⟨*proof*⟩

lemma $aligned_d$*_mono_exp_Suc*: $aligned_d\ (Suc\ e)\ M \Longrightarrow aligned_d\ e\ M$
⟨*proof*⟩

lemma $aligned_d$*_mono_hmset*:
assumes $aligned_d\ e\ M$ **and** $hmsetmset\ M' \subseteq\# hmsetmset\ M$
shows $aligned_d\ e\ M'$
⟨*proof*⟩

lemma *decode_exp_shift_Suc*:
assumes $align_d$: $aligned_d\ (Suc\ e)\ M$
shows $decode\ e\ M = base * decode\ (Suc\ e)\ M$

⟨*proof*⟩

lemma *decode_exp_shift*:
assumes $aligned_d$ *e M*
shows *decode 0 M* = *base* ^ *e* ∗ *decode e M*
⟨*proof*⟩

lemma *decode_plus*:
assumes $align_d$*_M*: $aligned_d$ *e M*
shows *decode e* (*M* + *N*) = *decode e M* + *decode e N*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *inj_decode*: *inj_on* (*decode e*) {*M*. *well_base*$_h$ *M* ∧ $aligned_d$ *e M*}
⟨*proof*⟩

lemma *decode_0_iff*: *well_base*$_h$ *M* ⟹ $aligned_d$ *e M* ⟹ *decode e M* = *0* ⟷ *M* = *0*
⟨*proof*⟩

lemma *decode_encode*: *decode e* (*encode e n*) = *n*
⟨*proof*⟩

lemma *encode_decode_exp_0*: *well_base*$_h$ *M* ⟹ *encode 0* (*decode 0 M*) = *M*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

theorem *goodsteins_theorem*: ∃ *i*. *goodstein i* = *0*
⟨*proof*⟩

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*: $UNIV = UNIV \times UNIV$
⟨*proof*⟩

lemma *infinite_cartesian_productI1*: $infinite\ A \Longrightarrow B \neq \{\} \Longrightarrow infinite\ (A \times B)$
⟨*proof*⟩

13.2 Types

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

definition *mk_fun* (**infixr** →→ *65*) **where**
Ts →→ *T* = *fold* (→) (*rev Ts*) *T*

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

definition *arity* **where**
arity T = *length* (*dest_fun T*)

lemma *mk_fun_dest_fun*[*simp*]: *dest_fun T* →→ $\mathcal{B}$ = *T*
⟨*proof*⟩

lemma *dest_fun_mk_fun*[*simp*]: *dest_fun* (*Ts* →→ *T*) = *Ts* @ *dest_fun T*
⟨*proof*⟩

primrec δ **where**
δ $\mathcal{B}$ = *HMSet* {#}
| δ (*T* → *U*) = *HMSet* (*add_mset* (δ *T*) (*hmsetmset* (δ *U*)))

lemma *δ_mk_fun*: δ (*Ts* →→ *T*) = *HMSet* (*hmsetmset* (δ *T*) + *mset* (*map* δ *Ts*))
⟨*proof*⟩

lemma *type_induct* [*case_names Fun*]:
assumes
($\bigwedge T.\ (\bigwedge T1\ T2.\ T = T1 \to T2 \Longrightarrow P\ T1) \Longrightarrow$
$(\bigwedge T1\ T2.\ T = T1 \to T2 \Longrightarrow P\ T2) \Longrightarrow P\ T$)
shows *P T*
⟨*proof*⟩

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*⟩

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*
  ⟨*proof*⟩

lemma *fv_open_Var*: *fresh xT t* ⟹ *fv* (*open_Var i xT t*) |⊆| *finsert xT* (*fv t*)
  ⟨*proof*⟩

lemma *fv_close_Var*[*simp*]: *fv* (*close_Var i xT t*) = *fv t* |−| {|*xT*|}
  ⟨*proof*⟩

lemma *close_Var_open_Var*[*simp*]: *fresh xT t* ⟹ *close_Var i xT* (*open_Var i xT t*) = *t*
  ⟨*proof*⟩

lemma *open_Var_inj*: *fresh xT t* ⟹ *fresh xT u* ⟹ *open_Var i xT t* = *open_Var i xT u* ⟹ *t* = *u*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *open_Var_close_Var*[*simp*]: *lc t* ⟹ *open_Var i xT* (*close_Var i xT t*) = *t*
⟨*proof*⟩

end

lemma *close_Var_inj*: *lc t* ⟹ *lc u* ⟹ *close_Var i xT t* = *close_Var i xT u* ⟹ *t* = *u*
⟨*proof*⟩

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

lemma *Apps_snoc*: *f* • (*xs* @ [*x*]) = *f* • *xs* · *x*
⟨*proof*⟩

lemma *Apps_append*: *f* • (*xs* @ *ys*) = *f* • *xs* • *ys*
⟨*proof*⟩

lemma *Apps_inj*[*simp*]: *f* • *ts* = *g* • *ts* ⟷ *f* = *g*
⟨*proof*⟩

lemma *eq_Apps_conv*[*simp*]:
fixes *i* :: *idx* **and** *b* :: *bool* **and** *f* :: *expr* **and** *ts* :: *expr list*
shows
(⟨*m*⟩ = *f* • *ts*) = (⟨*m*⟩ = *f* ∧ *ts* = [])
(*f* • *ts* = ⟨*m*⟩) = (⟨*m*⟩ = *f* ∧ *ts* = [])
(*i* = *f* • *ts*) = (*i* = *f* ∧ *ts* = [])
(*f* • *ts* = *i*) = (*i* = *f* ∧ *ts* = [])
(*b* = *f* • *ts*) = (*b* = *f* ∧ *ts* = [])
(*f* • *ts* = *b*) = (*b* = *f* ∧ *ts* = [])
(*e1* ? *e2* = *f* • *ts*) = (*e1* ? *e2* = *f* ∧ *ts* = [])
(*f* • *ts* = *e1* ? *e2*) = (*e1* ? *e2* = *f* ∧ *ts* = [])
(Λ⟨*T*⟩ *t* = *f* • *ts*) = (Λ⟨*T*⟩ *t* = *f* ∧ *ts* = [])
(*f* • *ts* = Λ⟨*T*⟩ *t*) = (Λ⟨*T*⟩ *t* = *f* ∧ *ts* = [])
⟨*proof*⟩

lemma *Apps_Var_eq*[*simp*]: ⟨*xT*⟩ • *ss* = ⟨*yU*⟩ • *ts* ⟷ *xT* = *yU* ∧ *ss* = *ts*
⟨*proof*⟩

lemma *Apps_Abs_neq_Apps*[*simp*, *symmetric*, *simp*]:
Λ⟨*T*⟩ *r* · *t* ≠ ⟨*xT*⟩ • *ss*
Λ⟨*T*⟩ *r* · *t* ≠ (*i* :: *idx*) • *ss*
Λ⟨*T*⟩ *r* · *t* ≠ (*b* :: *bool*) • *ss*
Λ⟨*T*⟩ *r* · *t* ≠ (*e1* ? *e2*) • *ss*
⟨*proof*⟩

lemma *App_Abs_eq_Apps_Abs*[*simp*]: Λ⟨*T*⟩ *r* · *t* = Λ⟨*T'*⟩ *r'* • *ss* ⟷ *T* = *T'* ∧ *r* = *r'* ∧ *ss* = [*t*]
⟨*proof*⟩

lemma *Apps_Var_neq_Apps_Abs*[*simp*, *symmetric*, *simp*]: ⟨*xT*⟩ • *ss* ≠ Λ⟨*T*⟩ *r* • *ts*
⟨*proof*⟩

lemma *Apps_Var_neq_Apps_beta*[*simp*, *THEN not_sym*, *simp*]:
⟨*xT*⟩ • *ss* ≠ Λ⟨*T*⟩ *r* · *s* • *ts*

⟨*proof*⟩

lemma [*simp*]:
$(\Lambda\langle T\rangle\ r \bullet ts = \Lambda\langle T'\rangle\ r' \cdot s' \bullet ts') = (T = T' \wedge r = r' \wedge ts = s' \# ts')$
⟨*proof*⟩

lemma *fold_eq_Bool_iff* [*simp*]:
$fold\ (\rightarrow)\ (rev\ Ts)\ T = \mathcal{B} \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
$\mathcal{B} = fold\ (\rightarrow)\ (rev\ Ts)\ T \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
⟨*proof*⟩

lemma *fold_eq_Fun_iff* [*simp*]:
$fold\ (\rightarrow)\ (rev\ Ts)\ T = U \rightarrow V \longleftrightarrow$
$(Ts = [] \wedge T = U \rightarrow V \vee (\exists\ Us.\ Ts = U \# Us \wedge fold\ (\rightarrow)\ (rev\ Us)\ T = V))$
⟨*proof*⟩

13.4 Substitution

primrec *subst* **where**
$subst\ xT\ t\ \langle yU\rangle = (if\ xT = yU\ then\ t\ else\ \langle yU\rangle)$
$|\ subst\ xT\ t\ (i :: idx) = i$
$|\ subst\ xT\ t\ (b :: bool) = b$
$|\ subst\ xT\ t\ (e1\ ?\ e2) = subst\ xT\ t\ e1\ ?\ subst\ xT\ t\ e2$
$|\ subst\ xT\ t\ (e1 \cdot e2) = subst\ xT\ t\ e1 \cdot subst\ xT\ t\ e2$
$|\ subst\ xT\ t\ (\Lambda\langle T\rangle\ e) = \Lambda\langle T\rangle\ (subst\ xT\ t\ e)$

lemma *fv_subst*:
$fv\ (subst\ xT\ t\ u) = fv\ u\ |-|\ \{|xT|\}\ |\cup|\ (if\ xT\ |\in|\ fv\ u\ then\ fv\ t\ else\ \{||\})$
⟨*proof*⟩

lemma *subst_fresh*: $fresh\ xT\ u \Longrightarrow subst\ xT\ t\ u = u$
⟨*proof*⟩

context begin

private lemma *open_open_id*: $i \neq j \Longrightarrow open\ i\ t\ (open\ j\ t'\ u) = open\ j\ t'\ u \Longrightarrow open\ i\ t\ u = u$
⟨*proof*⟩

lemma *lc_open_id*: $lc\ u \Longrightarrow open\ k\ t\ u = u$
⟨*proof*⟩

lemma *subst_open*: $lc\ u \Longrightarrow subst\ xT\ u\ (open\ i\ t\ v) = open\ i\ (subst\ xT\ u\ t)\ (subst\ xT\ u\ v)$
⟨*proof*⟩

lemma *subst_open_Var*:
$xT \neq yU \Longrightarrow lc\ u \Longrightarrow subst\ xT\ u\ (open\_Var\ i\ yU\ v) = open\_Var\ i\ yU\ (subst\ xT\ u\ v)$
⟨*proof*⟩

lemma *subst_Apps* [*simp*]:
$subst\ xT\ u\ (f \bullet xs) = subst\ xT\ u\ f \bullet map\ (subst\ xT\ u)\ xs$
⟨*proof*⟩

end

context begin

private lemma *fresh_close_Var_id*: $fresh\ xT\ t \Longrightarrow close\_Var\ k\ xT\ t = t$
⟨*proof*⟩

lemma *subst_close_Var*:
$xT \neq yU \Longrightarrow fresh\ yU\ u \Longrightarrow subst\ xT\ u\ (close\_Var\ i\ yU\ t) = close\_Var\ i\ yU\ (subst\ xT\ u\ t)$
⟨*proof*⟩

end

lemma *subst_intro*: *fresh xT t* ⟹ *lc u* ⟹ *open0 u t* = *subst xT u* (*open0_Var xT t*)
⟨*proof*⟩

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

lemma *body_subst*[*simp*]: *body U u* ⟹ *lc t* ⟹ *body U* (*subst xT t u*)
⟨*proof*⟩

lemma *lc_open_Var*: *lc u* ⟹ *lc* (*open_Var i xT u*)
⟨*proof*⟩

lemma *lc_open*[*simp*]: *body U u* ⟹ *lc t* ⟹ *lc* (*open0 t u*)
⟨*proof*⟩

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*⟩

lemma *welltyped_lc*[*simp*]: *t* ::: *T* ⟹ *lc t*
⟨*proof*⟩

lemma *welltyped_subst*[*intro*]:
u ::: *U* ⟹ *t* ::: *snd xT* ⟹ *subst xT t u* ::: *U*
⟨*proof*⟩

lemma *rename_welltyped*: *u* ::: *U* ⟹ *subst* (*x*, *T*) ⟨(*y*, *T*)⟩ *u* ::: *U*
⟨*proof*⟩

lemma *welltyped_Abs_fresh*:
assumes *fresh* (*x*, *T*) *u* *open0_Var* (*x*, *T*) *u* ::: *U*
shows Λ⟨*T*⟩ *u* ::: *T* → *U*
⟨*proof*⟩

lemma *Apps_alt*: *f* • *ts* ::: *T* ⟷
(∃ *Ts*. *f* ::: *fold* (→) (*rev Ts*) *T* ∧ *list_all2* (:::) *ts Ts*)
⟨*proof*⟩

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

abbreviation *closed t* ≡ *fv t* = {||}

primrec *constant0* **where**
constant0 ℬ = *Var* (''*bool*'', ℬ)
| *constant0* (*T* → *U*) = Λ⟨*T*⟩ (*constant0 U*)

definition *constant T* = Λ⟨ℬ⟩ (*close0_Var* (''*bool*'', ℬ) (*constant0 T*))

lemma fv_constant0[simp]: fv (constant0 T) = {|(''bool'', ℬ)|}
⟨proof⟩

lemma closed_constant[simp]: closed (constant T)
⟨proof⟩

lemma welltyped_constant0[simp]: constant0 T ::: T
⟨proof⟩

lemma lc_constant0[simp]: lc (constant0 T)
⟨proof⟩

lemma welltyped_constant[simp]: constant T ::: ℬ → T
⟨proof⟩

definition nth_drop **where**
nth_drop i xs ≡ take i xs @ drop (Suc i) xs

definition nth_arg (**infixl** !− 100) **where**
nth_arg T i ≡ 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
⟨proof⟩

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
⟨proof⟩

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
⟨proof⟩

lemma variant_inj2:
CHR ''∗'' ∉ set b1 ⟹ CHR ''∗'' ∉ set b2 ⟹ variant i b1 = variant j b2 ⟹ b1 = b2
⟨proof⟩

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
  ⟨proof⟩

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

lemma welltyped_Abss: b ::: U ⟹ T = map snd xTs →→ U ⟹ Λ[xTs] b ::: T
  ⟨proof⟩

lemma welltyped_Apps: list_all2 (:::) ts Ts ⟹ f ::: Ts →→ U ⟹ f • ts ::: U
  ⟨proof⟩

lemma welltyped_open_Var_close_Var[intro!]:
  t ::: T ⟹ open0_Var xT (close0_Var xT t) ::: T
  ⟨proof⟩

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

lemma welltyped_bool_iff[simp]: (b :: bool) ::: T ⟷ T = 𝓑
  ⟨proof⟩

lemma welltyped_constant0_iff[simp]: constant0 T ::: U ⟷ (U = T)
  ⟨proof⟩

lemma welltyped_constant_iff[simp]: constant T ::: U ⟷ (U = 𝓑 → T)
  ⟨proof⟩
```

lemma *welltyped_Seq_iff*[*simp*]: $e1\ ?\ e2 ::: T \longleftrightarrow (T = \mathcal{B} \wedge e1 ::: \mathcal{B} \wedge e2 ::: \mathcal{B})$
⟨*proof*⟩

lemma *welltyped_Seqs_iff*[*simp*]: $es\ ??\ e ::: T \longleftrightarrow$
$((es \neq [] \longrightarrow T = \mathcal{B}) \wedge (\forall e \in set\ es.\ e ::: \mathcal{B}) \wedge e ::: T)$
⟨*proof*⟩

lemma *welltyped_App_iff*[*simp*]: $f \cdot t ::: U \longleftrightarrow (\exists T.\ f ::: T \rightarrow U \wedge t ::: T)$
⟨*proof*⟩

lemma *welltyped_Apps_iff*[*simp*]: $f \bullet ts ::: U \longleftrightarrow (\exists Ts.\ f ::: Ts \rightarrow\rightarrow U \wedge list\_all2\ (:::)\ ts\ Ts)$
⟨*proof*⟩

lemma *eq_mk_fun_iff*[*simp*]: $T = Ts \rightarrow\rightarrow \mathcal{B} \longleftrightarrow Ts = dest\_fun\ T$
⟨*proof*⟩

lemma *map_nth_eq_drop_take*[*simp*]: $j \leq length\ xs \Longrightarrow map\ (nth\ xs)\ [i\ ..< j] = drop\ i\ (take\ j\ xs)$
⟨*proof*⟩

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

lemma *welltyped_E*: $E\ T\ i ::: \varepsilon\ T\ i$ **and** *welltyped_P*: $P\ T\ i\ j ::: T \rightarrow \pi\ T\ i\ j$
⟨*proof*⟩

lemma *δ_gt_0*[*simp*]: $T \neq \mathcal{B} \Longrightarrow HMSet\ \{\#\} < \delta\ T$
⟨*proof*⟩

lemma *mset_nth_drop_less*: $i < length\ xs \Longrightarrow mset\ (nth\_drop\ i\ xs) < mset\ xs$
⟨*proof*⟩

lemma *map_nth_drop*: $i < length\ xs \Longrightarrow map\ f\ (nth\_drop\ i\ xs) = nth\_drop\ i\ (map\ f\ xs)$
⟨*proof*⟩

lemma *empty_less_mset*: $\{\#\} < mset\ xs \longleftrightarrow xs \neq []$
⟨*proof*⟩

lemma *dest_fun_alt*: $dest\_fun\ T = map\ (\lambda i.\ T\ !\!-\ i)\ [0..<ar\ T]$
⟨*proof*⟩

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 $\delta\ (\pi\ T\ i\ j) < \delta\ T$
⟨*proof*⟩

end

end