Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

October 10, 2017

Abstract

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

Contents

1 Introduction

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

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

2 More about Multisets

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

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

2.1 Basic Setup

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

mset_subset_eqD[*dest*, *intro?*]

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

sum_mset_sum_list[*simp*]

2.2 Lemmas about Intersection, Union and Pointwise Inclusion

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

2.3 Lemmas about Filter and Image

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

lemma *count_image_mset_inj*:
assumes ‹*inj f*›
shows ‹$\mathit{count}\ (\mathit{image\_mset}\ f\ M)\ (f\ x) = \mathit{count}\ M\ x$›
⟨*proof*⟩

lemma *count_image_mset_le_count_inj_on*:

inj_ on f (set_ mset M) $\Longrightarrow$ *count (image_ mset f M) y* $\leq$ *count M (inv_ into (set_ mset M) f y)*
⟨*proof*⟩

lemma *mset_ filter_ compl*: *mset (filter p xs) + mset (filter (Not* $\circ$ *p) xs) = mset xs*
⟨*proof*⟩

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

lemma *filter_ mset_ eq*: *filter_ mset (op = L) A = replicate_ mset (count A L) L*
⟨*proof*⟩

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

lemma *filter_ mset_ cong*:
assumes [*simp*]: *M = M'* **and** [*simp*]: $\bigwedge$*a. a* $\in\#$ *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* $\in\#$ *M. P (f x)*#} = {# *x* $\in\#$ *image_ mset f M. P x*#}
⟨*proof*⟩

lemma *image_ mset_ cong2*[*cong*]:
($\bigwedge$*x. x* $\in\#$ *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* = {#}) = ($\forall$ *L*$\in\#$*M. ¬ P L*)›
⟨*proof*⟩

lemma *multiset_ filter_ mono2*: ‹*filter_ mset P A* $\subseteq\#$ *filter_ mset Q A* $\longleftrightarrow$ ($\forall$ *a*$\in\#$*A. P a* $\longrightarrow$ *Q a*)›
⟨*proof*⟩

lemma *image_ filter_ cong*:
assumes ‹$\bigwedge$*C. C* $\in\#$ *M* $\Longrightarrow$ *P C* $\Longrightarrow$ *f C = g C*›
shows ‹{#*f C. C* $\in\#$ {#*C* $\in\#$ *M. P C*#}#} = {#*g C*| *C*$\in\#$ *M. P C*#}›
⟨*proof*⟩

lemma *image_ mset_ filter_ swap2*: ‹{#*C* $\in\#$ {#*P x. x* $\in\#$ *D*#}. *Q C* #} = {#*P x. x* $\in\#$ {#*C*| *C* $\in\#$ *D. Q* (*P C*)#}#}›
⟨*proof*⟩

2.4 Lemmas about Sum

lemma *sum_ image_ mset_ mono*:
fixes *f* :: *'a* $\Rightarrow$ *'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* $\Rightarrow$ *'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*::*semiring_ div*
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*]:
$n \geq count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A - \{a\}$
$n < count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A$
⟨*proof*⟩

abbreviation *removeAll_mset* :: $'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
$removeAll\_mset\ C\ M \equiv M - replicate\_mset\ (count\ M\ C)\ C$

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

lemma *removeAll_mset_filter_mset*: $removeAll\_mset\ C\ M = filter\_mset\ (op \neq\ C)\ M$
⟨*proof*⟩

abbreviation *remove1_mset* :: $'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
$remove1\_mset\ C\ M \equiv M - \{\#C\#\}$

lemma *removeAll_subseteq_remove1_mset*: $removeAll\_mset\ x\ M \subseteq\# remove1\_mset\ x\ M$
⟨*proof*⟩

lemma *in_remove1_mset_neq*:
assumes *ab*: $a \neq b$
shows $a \in\# remove1\_mset\ b\ C \longleftrightarrow a \in\# C$
⟨*proof*⟩

lemma *size_mset_removeAll_mset_le_iff*: $size\ (removeAll\_mset\ x\ M) < size\ M \longleftrightarrow x \in\# M$
⟨*proof*⟩

lemma *size_remove1_mset_If*: ‹$size\ (remove1\_mset\ x\ M) = size\ M - (if\ x \in\# M\ then\ 1\ else\ 0)$›
⟨*proof*⟩

lemma *size_mset_remove1_mset_le_iff*: $size\ (remove1\_mset\ x\ M) < size\ M \longleftrightarrow x \in\# M$
⟨*proof*⟩

lemma *single_remove1_mset_eq*:
$add\_mset\ a\ (remove1\_mset\ a\ M) = M \longleftrightarrow a \in\# M$
⟨*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*: $\{\#x \in\# M.\ P\ x \wedge x \neq y\#\} = removeAll\_mset\ y\ \{\#\ x{\in}\#M.\ P\ x\#\}$
⟨*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*:
$A - remove1\_mset\ b\ B = (if\ b \in\# B \wedge b \in\# A \wedge count\ A\ b \geq count\ B\ b\ then\ \{\#b\#\} + (A - B)\ else\ A - B)$
⟨*proof*⟩

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

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

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

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

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*]: $replicate\_mset\ m\ x \subset\# replicate\_mset\ n\ x \longleftrightarrow m < n$
⟨*proof*⟩

lemma *replicate_mset_subseteq_iff_le*[*simp*]: $replicate\_mset\ m\ x \subseteq\# replicate\_mset\ n\ x \longleftrightarrow m \leq n$
⟨*proof*⟩

lemma *replicate_mset_lt_iff_lt*[*simp*]: $replicate\_mset\ m\ x < replicate\_mset\ n\ x \longleftrightarrow m < n$
⟨*proof*⟩

lemma *replicate_mset_le_iff_le*[*simp*]: $replicate\_mset\ m\ x \leq replicate\_mset\ n\ x \longleftrightarrow m \leq n$
⟨*proof*⟩

lemma *replicate_mset_eq_iff* [*simp*]:
$replicate\_mset\ m\ x = replicate\_mset\ n\ y \longleftrightarrow m = n \wedge (m \neq 0 \longrightarrow x = y)$
⟨*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]*

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

lemma *distinct_mset_count_less_1*: *distinct_mset S* ⟷ (∀ *a*. *count S a* ≤ *1*)
⟨*proof*⟩

lemma *distinct_mset_empty*[*simp*]: *distinct_mset* {#}
⟨*proof*⟩

lemma *distinct_mset_singleton*: *distinct_mset* {#*a*#}
⟨*proof*⟩

lemma *distinct_mset_union*:
assumes *dist*: *distinct_mset* (*A* + *B*)
shows *distinct_mset A*
⟨*proof*⟩

lemma *distinct_mset_minus*[*simp*]: *distinct_mset A* ⟹ *distinct_mset* (*A* − *B*)
⟨*proof*⟩

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

lemma *distinct_mset_rempdups_union_mset*:
assumes *distinct_mset A* **and** *distinct_mset B*
shows *A* ∪# *B* = *remdups_mset* (*A* + *B*)
⟨*proof*⟩

lemma *distinct_mset_add_mset*[*simp*]: *distinct_mset* (*add_mset a L*) ⟷ *a* ∉# *L* ∧ *distinct_mset L*
⟨*proof*⟩

lemma *distinct_mset_size_eq_card*: *distinct_mset C* ⟹ *size C* = *card* (*set_mset C*)
⟨*proof*⟩

lemma *distinct_mset_add*:
distinct_mset (*L* + *L'*) ⟷ *distinct_mset L* ∧ *distinct_mset L'* ∧ *L* ∩# *L'* = {#}
⟨*proof*⟩

lemma *distinct_mset_set_mset_ident*[*simp*]: *distinct_mset M* ⟹ *mset_set* (*set_mset M*) = *M*
⟨*proof*⟩

lemma *distinct_finite_set_mset_subseteq_iff*[*iff*]:
assumes *distinct_mset M finite N*
shows *set_mset M* ⊆ *N* ⟷ *M* ⊆# *mset_set N*
⟨*proof*⟩

lemma *distinct_mem_diff_mset*:
assumes *dist*: *distinct_mset M* **and** *mem*: *x* ∈ *set_mset* (*M* − *N*)
shows *x* ∉ *set_mset N*
⟨*proof*⟩

lemma *distinct_set_mset_eq*:
assumes *distinct_mset M distinct_mset N set_mset M* = *set_mset N*
shows *M* = *N*
⟨*proof*⟩

lemma *distinct_mset_union_mset*[*simp*]:
‹*distinct_mset* (*D* ∪# *C*) ⟷ *distinct_mset D* ∧ *distinct_mset C*›
⟨*proof*⟩

lemma *distinct_mset_inter_mset*:
distinct_mset C ⟹ *distinct_mset* (*C* ∩# *D*)

distinct_mset D ⟹ distinct_mset (C ∩# D)
⟨*proof*⟩

lemma *distinct_mset_remove1_All*: *distinct_mset C ⟹ remove1_mset L C = removeAll_mset L C*
⟨*proof*⟩

lemma *distinct_mset_size_2*: *distinct_mset {#a, b#} ⟷ a ≠ b*
⟨*proof*⟩

lemma *distinct_mset_filter*: *distinct_mset M ⟹ distinct_mset {# L ∈# M. P L#}*
⟨*proof*⟩

lemma *distinct_mset_mset_distinct*[*simp*]: ‹*distinct_mset (mset xs) = distinct xs*›
⟨*proof*⟩

lemma *distinct_image_mset_inj*:
‹*inj_on f (set_mset M) ⟹ distinct_mset (image_mset f M) ⟷ distinct_mset M*›
⟨*proof*⟩

2.9 Repeat Operation

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

lemma *repeat_mset_prod*: *repeat_mset (m * n) A = ((op + (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 ≡ ⋃# {#{#(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 (λx. 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 $(a, b) \in\# A \times\# B$ – 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* (⋃# *M*) *b* = (∑ *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* ⊆# *C* **and** ⋀*x*. *x*∈#*A* ⟹ *B x* ⊆# *D x*
shows *Sigma_mset A B* ⊆# *Sigma_mset C D*
⟨*proof*⟩

lemma *mem_Sigma_mset_iff*[*iff*]: ((*a*,*b*) ∈# *Sigma_mset A B*) = (*a* ∈# *A* ∧ *b* ∈# *B a*)
⟨*proof*⟩

lemma *mem_Times_mset_iff*: *x* ∈# *A* ×# *B* ⟷ *fst x* ∈# *A* ∧ *snd x* ∈# *B*
⟨*proof*⟩

lemma *Sigma_mset_empty_iff*: (*SIGMAMSET i*∈#*I*. *X i*) = {#} ⟷ (∀ *i*∈#*I*. *X i* = {#})
⟨*proof*⟩

lemma *Times_mset_subset_mset_cancel1*: *x* ∈# *A* ⟹ (*A* ×# *B* ⊆# *A* ×# *C*) = (*B* ⊆# *C*)
⟨*proof*⟩

lemma *Times_mset_subset_mset_cancel2*: *x* ∈# *C* ⟹ (*A* ×# *C* ⊆# *B* ×# *C*) = (*A* ⊆# *B*)
⟨*proof*⟩

lemma *Times_mset_eq_cancel2*: *x* ∈# *C* ⟹ (*A* ×# *C* = *B* ×# *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*:
$(\sum x{\in}\#M.\ if\ a = x\ then\ (f\ x)\ else\ 0) = ((op\ +\ (f\ a))\ \hat{}\hat{}\ (count\ M\ a))\ 0$
⟨*proof*⟩

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

lemma *untion_image_mset_Pair_distribute*:
$\bigcup\#\{\#image\_mset\ (Pair\ x)\ (C\ x).\ x\ \in\#\ J\ -\ I\#\} =$
$\bigcup\#\ \{\#image\_mset\ (Pair\ x)\ (C\ x).\ x\ \in\#\ J\#\} - \bigcup\#\{\#image\_mset\ (Pair\ x)\ (C\ x).\ x\ \in\#\ I\#\}$
⟨*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*: $Sigma\_mset\ (\bigcup\#X)\ B = (\bigcup\#\ (image\_mset\ (\lambda A.\ Sigma\_mset\ A\ B)\ X))$
⟨*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*: $A\ \times\#\ add\_mset\ x\ B = A\ \times\#\ B\ +\ image\_mset\ (\lambda a.\ Pair\ a\ x)\ A$
⟨*proof*⟩

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

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

lemma *snd_image_mset_times_mset* [*simp*]:
image_mset snd (*A* ×# *B*) = (*if A* = {#} *then* {#} *else repeat_mset* (*size A*) *B*)
⟨*proof*⟩

lemma *product_swap_mset*: *image_mset prod.swap* (*A* ×# *B*) = *B* ×# *A*
⟨*proof*⟩

context
begin

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

lemma *member_product_mset*: *x* ∈# *Multiset_More.product_mset A B* ⟷ *x* ∈# *A* ×# *B*
⟨*proof*⟩

end

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

lemma *Times_mset_image_mset1*: *image_mset f A* ×# *B* = *image_mset* (λ(*a*, *b*). (*f a*, *b*)) (*A* ×# *B*)
⟨*proof*⟩

lemma *Times_mset_image_mset2*: *A* ×# *image_mset f B* = *image_mset* (λ(*a*, *b*). (*a*, *f b*)) (*A* ×# *B*)
⟨*proof*⟩

lemma *sum_le_singleton*: *A* ⊆ {*x*} ⟹ *sum f A* = (*if x* ∈ *A then f x else* 0)
⟨*proof*⟩

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

2.11 Transfer Rules

lemma *plus_multiset_transfer*[*transfer_rule*]:
(*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_mset R*) (*rel_mset R*))) (*op* +) (*op* +)
⟨*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*))) (*op* −) (*op* −)
⟨*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 op* =)) *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*) (*op* =)))
(λ*M N*. *filter_mset* (*Domainp R*) *M* ⊆# *filter_mset* (*Domainp R*) *N*) (*op* ⊆#)
⟨*proof*⟩

lemma *sum_mset_transfer*[*transfer_rule*]:
R 0 0 ⟹ *rel_fun R* (*rel_fun R R*) *op* + *op* + ⟹ (*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

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 [*simp*]: *mset* (*list_ of_ mset m*) = *m*
⟨*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 *subseteq_ list_ Union_ mset*:
assumes *length Ci* = *n*
assumes *length CAi* = *n*
assumes ∀ *i*<*n*. *Ci* ! *i* ⊆# *CAi* ! *i*
shows ⋃#*mset Ci* ⊆# ⋃#*mset CAi*
⟨*proof*⟩

2.12.3 More on multisets and functions

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*: $\forall x \in\# X.\ x = y \Longrightarrow card\ (set\_mset\ X) \leq Suc\ 0$
⟨*proof*⟩

lemma *Melem_subseteq_Union_mset*[*simp*]:
assumes $x \in\# T$
shows $x \subseteq\# \bigcup\# T$
⟨*proof*⟩

lemma *Melem_subset_eq_sum_list* [*simp*]:
assumes $x \in\#$ *mset* T
shows $x \subseteq\#$ *sum_list* T
⟨*proof*⟩

lemma *less_subset_eq_Union_mset*[*simp*]:
assumes $i <$ *length* CAi
shows $CAi\ !\ i \subseteq\# \bigcup\#$*mset* CAi
⟨*proof*⟩

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

end

3 Signed (Finite) Multisets

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

3.1 Definition of Signed Multisets

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

quotient-type *'a zmultiset* = *'a multiset* × *'a multiset* / *equiv_zmset*
⟨*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**
λ(*Mp*, *Mn*) *x*. *int* (*count Mp x*) − *int* (*count Mn x*)
⟨*proof*⟩

lemma *zcount_ inject*: *zcount M* = *zcount N* ⟷ *M* = *N*
⟨*proof*⟩

lemma *zmultiset_ eq_ iff*: *M* = *N* ⟷ (∀ *a*. *zcount M a* = *zcount N a*)
⟨*proof*⟩

lemma *zmultiset_ eqI*: (⋀*x*. *zcount A x* = *zcount B x*) ⟹ *A* = *B*
⟨*proof*⟩

lemma *zcount_ uminus*[*simp*]: *zcount* (− *A*) *x* = − *zcount A x*
⟨*proof*⟩

lift-definition *add_ zmset* :: *'a* ⇒ *'a zmultiset* ⇒ *'a zmultiset* **is**
λ*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* ⟷ *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*#}$_z$ ≠ {#}$_z$ **and**
empty_ ne_ singleton_ zmset[*simp*]: {#}$_z$ ≠ {#*x*#}$_z$
⟨*proof*⟩

lemma
singleton_ ne_ uminus_ singleton_ zmset[*simp*]: {#*x*#}$_z$ ≠ − {#*y*#}$_z$ **and**
uminus_ singleton_ ne_ singleton_ zmset[*simp*]: − {#*x*#}$_z$ ≠ {#*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* ≠ *0*}

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

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

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

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

context
begin

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

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

end

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

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

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

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

lemma *finite_count_ne*: *finite* $\{x.\ count\ M\ x \neq count\ N\ x\}$
⟨*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** ⊆#$_z$ *50*) **where**
A ⊆#$_z$ *B* ⟷ (∀ *a*. *zcount A a* ≤ *zcount B a*)

definition *subset_zmset* :: *'a zmultiset* ⇒ *'a zmultiset* ⇒ *bool* (**infix** ⊂#$_z$ *50*) **where**
A ⊂#$_z$ *B* ⟷ *A* ⊆#$_z$ *B* ∧ *A* ≠ *B*

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

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

notation (*input*)
  subseteq_zmset (**infix** ⊆#$_z$ *50*) **and**
  supseteq_zmset (**infix** ⊇#$_z$ *50*)

notation (*ASCII*)
  subseteq_zmset (**infix** ⊆#$_z$ *50*) **and**
  subset_zmset (**infix** ⊂#$_z$ *50*) **and**
  supseteq_zmset (**infix** ⊇#$_z$ *50*) **and**
  supset_zmset (**infix** >#$_z$ *50*)

interpretation *subset_zmset*: *ordered_ab_semigroup_add_imp_le op + op − op* ⊆#$_z$ *op* ⊂#$_z$
  ⟨*proof*⟩

interpretation *subset_zmset*:
  ordered_ab_semigroup_monoid_add_imp_le op + 0 op − op ⊆#$_z$ *op* ⊂#$_z$
  ⟨*proof*⟩

lemma *zmset_subset_eqI*: (⋀*a*. *zcount A a* ≤ *zcount B a*) ⟹ *A* ⊆#$_z$ *B*
  ⟨*proof*⟩

lemma *zmset_subset_eq_zcount*: *A* ⊆#$_z$ *B* ⟹ *zcount A a* ≤ *zcount B a*
  ⟨*proof*⟩

lemma *zmset_subset_eq_add_zmset_cancel*: ‹*add_zmset a A* ⊆#$_z$ *add_zmset a B* ⟷ *A* ⊆#$_z$ *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* ⊆#$_z$ *B* ⟹ *A* ⊂#$_z$ *B*
  ⟨*proof*⟩

lemma *zmset_subset_insertD*: *add_zmset x A* ⊂#$_z$ *B* ⟹ *A* ⊂#$_z$ *B*
  ⟨*proof*⟩

lemma *subset_eq_diff_conv_zmset*: *A − C* ⊆#$_z$ *B* ⟷ *A* ⊆#$_z$ *B + C*
  ⟨*proof*⟩

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

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

lemma *zmset_subset_add_zmset*[*simp*]: *add_zmset x N* ⊂#$_z$ *add_zmset x M* ⟷ *N* ⊂#$_z$ *M*
  ⟨*proof*⟩

lemma *zmset_of_subseteq_iff*[*simp*]: *zmset_of M* ⊆#$_z$ *zmset_of N* ⟷ *M* ⊆# *N*
  ⟨*proof*⟩

lemma *zmset_of_subset_iff*[*simp*]: *zmset_of M* ⊂#$_z$ *zmset_of N* ⟷ *M* ⊂# *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* *op* $\subseteq\#_z$ *op* $\subset\#_z$
⟨*proof*⟩

3.4 Replicate and Repeat Operations

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

lemma *replicate_zmset_0*[*simp*]: *replicate_zmset* 0 $x = \{\#\}_z$
⟨*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**
λ*P* (*Mp*, *Mn*). (*filter_mset P Mp*, *filter_mset P Mn*)
⟨*proof*⟩

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

lemma *count_filter_zmset*[*simp*]:
zcount (*filter_zmset P M*) *a* = (*if P a then zcount M a else 0*)
⟨*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* ⊆#*z* *B*
shows *filter_zmset f A* ⊆#*z* *filter_zmset f B*
⟨*proof*⟩

lemma *filter_filter_zmset*: *filter_zmset P* (*filter_zmset Q M*) = {#*x* ∈# *M*. *Q x* ∧ *P x*#}
⟨*proof*⟩

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

3.5 Uncategorized

lemma *multi_drop_mem_not_eq_zmset*: *B* − $\{\#c\#\}_z$ ≠ *B*
⟨*proof*⟩

lemma *zmultiset_partition*: *M* = {#*x* ∈#*z* *M*. *P x* #} + {#*x* ∈#*z* *M*. ¬ *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* ∘ *f*) {#} (*mset_pos M*)) −
zmset_of (*fold_mset* (*add_mset* ∘ *f*) {#} (*mset_neg M*))

3.7 Multiset Order

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

lift-definition *less_zmultiset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow bool$ **is**
$\lambda(Mp, Mn)\ (Np, Nn).\ Mp + Nn < Mn + Np$
⟨*proof*⟩

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

instance
⟨*proof*⟩

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* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow 'a\ zmultiset$ **where**
inf_zmultiset A B = (*if A* < *B then A else B*)

definition *sup_zmultiset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow 'a\ zmultiset$ **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* ⟷ *M* < *N*
⟨*proof*⟩

lemma *zmset_of_le*: *zmset_of M* ≤ *zmset_of N* ⟷ *M* ≤ *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*:
⟨$j \leq i \Longrightarrow$ (*repeat_zmset i u* $+ m \subseteq\#_z$ *repeat_zmset j u* $+ n$) $=$ (*repeat_zmset* $(i - j)$ $u + m \subseteq\#_z n$)⟩
⟨*proof*⟩

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

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

lemma *zmset_subset_add_iff2*:
⟨$i \leq j \Longrightarrow$ (*repeat_zmset i u* $+ m \subset\#_z$ *repeat_zmset j u* $+ n$) $=$ ($m \subset\#_z$ *repeat_zmset* $(j - i)$ $u + n$)⟩
⟨*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*⟩

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**
  (⋀*X*. *X* ∈# *M* ⟹ *no_elem X*) ⟹ *no_elem* (*MSet M*)

inductive-set *sub_nmset* :: (*'a nmultiset* × *'a nmultiset*) *set* **where**
  X ∈# *M* ⟹ (*X*, *MSet M*) ∈ *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* ∈# *M* ⟹ |*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* ⟷
   (∃*X Y*. *X* ≠ {#} ∧ *X* ⊆# *N* ∧ *M* = (*N* − *X*) + *Y* ∧ (∀*k*. *k* ∈# *Y* ⟶ (∃*a*. *a* ∈# *X* ∧ *R k a*)))

lemma *less_multiset_ext*$_{DM}$*_imp_mult*:
  assumes
    N_A: *set_mset N* ⊆ *A* **and** *M_A*: *set_mset M* ⊆ *A* **and** *less*: *less_multiset_ext*$_{DM}$ *R M N*
  shows (*M*, *N*) ∈ *mult* {(*x*, *y*). *x* ∈ *A* ∧ *y* ∈ *A* ∧ *R x y*}
⟨*proof*⟩

lemma *mult_imp_less_multiset_ext*$_{DM}$:
  assumes
    N_A: *set_mset N* ⊆ *A* **and** *M_A*: *set_mset M* ⊆ *A* **and**
    trans: ∀*x* ∈ *A*. ∀*y* ∈ *A*. ∀*z* ∈ *A*. *R x y* ⟶ *R y z* ⟶ *R x z* **and**
    in_mult: (*M*, *N*) ∈ *mult* {(*x*, *y*). *x* ∈ *A* ∧ *y* ∈ *A* ∧ *R x y*}
  shows *less_multiset_ext*$_{DM}$ *R M N*
  ⟨*proof*⟩

lemma *less_multiset_ext*$_{DM}$*_iff_mult*:
  assumes
    N_A: *set_mset N* ⊆ *A* **and** *M_A*: *set_mset M* ⊆ *A* **and**
    trans: ∀*x* ∈ *A*. ∀*y* ∈ *A*. ∀*z* ∈ *A*. *R x y* ⟶ *R y z* ⟶ *R x z*
  shows *less_multiset_ext*$_{DM}$ *R M N* ⟷ (*M*, *N*) ∈ *mult* {(*x*, *y*). *x* ∈ *A* ∧ *y* ∈ *A* ∧ *R x y*}

⟨*proof*⟩

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

lemma *less_multiset_ext$_{DM}$_cong*[*fundef_cong*]:
(⋀*X Y k a*. *X* ≠ {#} ⟹ *X* ⊆# *N* ⟹ *M* = (*N* − *X*) + *Y* ⟹ *k* ∈# *Y* ⟹ *R k a* = *S k a*) ⟹
less_multiset_ext$_{DM}$ R M N = *less_multiset_ext$_{DM}$ S M N*
⟨*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$_{DM}$ op* <= (*op* <)
⟨*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*: $|X| < |Y| \Longrightarrow X < Y$
⟨*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* $\Rightarrow$ *unit nmultiset* **where**
Rep_hmultiset (*HMSet M*) = *MSet* (*image_mset Rep_hmultiset M*)

primrec (*nonexhaustive*) *Abs_hmultiset* :: *unit nmultiset* $\Rightarrow$ *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* ∘ *MSet* ∘ *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** *op* < ⟨*proof*⟩
lift-definition *less_eq_hmultiset* :: *hmultiset* ⇒ *hmultiset* ⇒ *bool* **is** *op* ≤ ⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *less_HMSet_iff_less_multiset_ext*$_{DM}$: *HMSet M* < *HMSet N* ⟷ *less_multiset_ext*$_{DM}$ (*op* <) *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** *op* < ⟨*proof*⟩
lift-definition *less_ eq_ zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *bool* **is** *op* ≤ ⟨*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**
$\lambda A\ B.\ A + B$ ⟨*proof*⟩

lift-definition *minus_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **is**
$\lambda A\ B.\ A - B$ ⟨*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*[*simp*]: *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*: $M \leq N$
obtains $A\ B\ C$ **where** M = *zhmset_of* $A + C$ **and** N = *zhmset_of* $B + C$ **and** $A \leq B$
⟨*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^ ≡ λ*m*. *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* (*op* +)) (*hmsetmset A* ×# *hmsetmset B*))

lemma *hmsetmset_times*:
hmsetmset (*m* ∗ *n*) = *image_mset* (*case_prod* (*op* +)) (*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*]: ω^*M* * *N* = *HMSet* (*image_mset* ((*op* +) *M*) (*hmsetmset N*))
⟨*proof*⟩

lemma *singleton_times_right_hmset*[*simp*]: *N* * ω^*M* = *HMSet* (*image_mset* ((*op* +) *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 op* +
⟨*proof*⟩

lemma *Times_mset_monoL*:
assumes *less*: *M* < *N* **and** *Z_nemp*: *Z* ≠ {#}
shows *M* ×# *Z* < *N* ×# *Z*
⟨*proof*⟩

lemma *times_hmultiset_monoL*:
a < *b* ⟹ *0* < *c* ⟹ *a* * *c* < *b* * *c* **for** *a b c* :: *hmultiset*
⟨*proof*⟩

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

lemma *mult_le_mono1_hmset*: *i* ≤ *j* ⟹ *i* * *k* ≤ *j* * *k* **for** *i j k* :: *hmultiset*
⟨*proof*⟩

lemma *mult_le_mono2_hmset*: *i* ≤ *j* ⟹ *k* * *i* ≤ *k* * *j* **for** *i j k* :: *hmultiset*
⟨*proof*⟩

lemma *mult_le_mono_hmset*: *i* ≤ *j* ⟹ *k* ≤ *l* ⟹ *i* * *k* ≤ *j* * *l* **for** *i j k l* :: *hmultiset*
⟨*proof*⟩

lemma *less_iff_add1_le_hmset*: *m* < *n* ⟷ *m* + *1* ≤ *n* **for** *m n* :: *hmultiset*
⟨*proof*⟩

lemma *zero_less_iff_1_le_hmset*: *0* < *n* ⟷ *1* ≤ *n* **for** *n* :: *hmultiset*
⟨*proof*⟩

lemma *less_add_1_iff_le_hmset*: *m* < *n* + *1* ⟷ *m* ≤ *n* **for** *m n* :: *hmultiset*
⟨*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 *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*]: $\neg\ \omega < 1$
⟨*proof*⟩

lemma *not_ω_lt_of_nat*[*simp*]: $\neg\ \omega < of\_nat\ n$
⟨*proof*⟩

lemma *not_ω_lt_numeral*[*simp*]: $\neg\ \omega < numeral\ n$
⟨*proof*⟩

lemma *not_ω_le_1*[*simp*]: $\neg\ \omega \leq 1$
⟨*proof*⟩

lemma *not_ω_le_of_nat*[*simp*]: $\neg\ \omega \leq of\_nat\ n$
⟨*proof*⟩

lemma *not_ω_le_numeral*[*simp*]: $\neg\ \omega \leq numeral\ n$
⟨*proof*⟩

lemma *zero_ne_ω*[*simp*]: $0 \neq \omega$
⟨*proof*⟩

lemma *one_ne_ω*[*simp*]: $1 \neq \omega$
⟨*proof*⟩

lemma *numeral_ne_ω*[*simp*]: $numeral\ n \neq \omega$
⟨*proof*⟩

lemma
ω_ne_0[*simp*]: $\omega \neq 0$ **and**
ω_ne_1[*simp*]: $\omega \neq 1$ **and**
ω_ne_of_nat[*simp*]: $\omega \neq of\_nat\ m$ **and**
ω_ne_numeral[*simp*]: $\omega \neq numeral\ n$
⟨*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*]: $hmset\_of\_enat\ m \leq hmset\_of\_enat\ n \longleftrightarrow m \leq n$
⟨*proof*⟩

lemma *lt_ω_imp_ex_of_nat*:
assumes *M_lt_ω*: $M < \omega$
shows $\exists n.\ M = of\_nat\ n$
⟨*proof*⟩

lemma *le_ω_imp_ex_hmset_of_enat*:
assumes *M_le_ω*: $M \leq \omega$
shows $\exists n.\ M = hmset\_of\_enat\ n$
⟨*proof*⟩

lemma *lt_ω_lt_ω_imp_times_lt_ω*: $M < \omega \Longrightarrow N < \omega \Longrightarrow M * N < \omega$
⟨*proof*⟩

lemma *times_ω_minus_of_nat*[*simp*]: $m * \omega - of\_nat\ n = m * \omega$
⟨*proof*⟩

lemma *times_ω_minus_numeral*[*simp*]: $m * \omega - numeral\ n = m * \omega$
⟨*proof*⟩

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

lemma *ω_minus_1*[*simp*]: $\omega - 1 = \omega$

⟨*proof*⟩

lemma *ω_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 *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

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_exp* :: *hmultiset* ⇒ *zhmultiset* (*ω^*) **where**
*ω^* ≡ λ*m*. *ZHMSet* {#*m*#}$_z$

lift-definition *one_zhmultiset* :: *zhmultiset* **is** {#*0*#}$_z$ ⟨*proof*⟩

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

lemma $\omega_z\_as\_\omega$: $\omega_z = zhmset\_of\ \omega$
⟨*proof*⟩

lift-definition $times\_zhmultiset :: zhmultiset \Rightarrow zhmultiset \Rightarrow zhmultiset$ **is**
$\lambda M\ N.$
$zmset\_of\ (hmsetmset\ (HMSet\ (mset\_pos\ M) * HMSet\ (mset\_pos\ N)))$
$- zmset\_of\ (hmsetmset\ (HMSet\ (mset\_pos\ M) * HMSet\ (mset\_neg\ N)))$
$+ zmset\_of\ (hmsetmset\ (HMSet\ (mset\_neg\ M) * HMSet\ (mset\_neg\ N)))$
$- zmset\_of\ (hmsetmset\ (HMSet\ (mset\_neg\ M) * HMSet\ (mset\_pos\ N)))$ ⟨*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 $of\_nat\_inject\_zhmset[simp]$: $(of\_nat\ m :: zhmultiset) = of\_nat\ n \longleftrightarrow m = n$
⟨*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 $of\_nat\_lt\_\omega_z[simp]$: $of\_nat\ n < \omega_z$
⟨*proof*⟩

lemma $of\_nat\_ne\_\omega_z[simp]$: $of\_nat\ n \neq \omega_z$
⟨*proof*⟩

8.3 Embedding of Extended Natural Numbers

primrec $zhmset\_of\_enat :: enat \Rightarrow zhmultiset$ **where**
$zhmset\_of\_enat\ (enat\ n) = of\_nat\ n$
| $zhmset\_of\_enat\ \infty = \omega_z$

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*: $m < n + 1 \longleftrightarrow m \leq n$ **for** $m\ n$ :: *zhmultiset*
⟨*proof*⟩

lemma *nonneg_le_mult_right_mono_zhmset*:
fixes $x\ y\ z$ :: *zhmultiset*
assumes x: $0 \leq x$ **and** y: $0 < y$ **and** z: $x \leq z$
shows $x \leq y * z$
⟨*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
$\omega_z\_ne\_0$[*simp*]: $\omega_z \neq 0$ **and**
$\omega_z\_ne\_1$[*simp*]: $\omega_z \neq 1$ **and**
$\omega_z\_ne\_of\_nat$[*simp*]: $\omega_z \neq of\_nat\ m$ **and**
$\omega_z\_ne\_numeral$[*simp*]: $\omega_z \neq numeral\ n$
⟨*proof*⟩

lemma
zhmset_of_enat_inject[*simp*]: $zhmset\_of\_enat\ m = zhmset\_of\_enat\ n \longleftrightarrow m = n$ **and**
zhmset_of_enat_lt_iff_lt[*simp*]: $zhmset\_of\_enat\ m < zhmset\_of\_enat\ n \longleftrightarrow m < n$ **and**
zhmset_of_enat_le_iff_le[*simp*]: $zhmset\_of\_enat\ m \leq zhmset\_of\_enat\ n \longleftrightarrow m \leq n$
⟨*proof*⟩

lemma *of_nat_lt_zhmset_of_enat_iff*: $of\_nat\ m < zhmset\_of\_enat\ n \longleftrightarrow enat\ m < n$
⟨*proof*⟩

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

lemma *zhmset_of_enat_lt_iff_ne_infinity*: $zhmset\_of\_enat\ x < \omega_z \longleftrightarrow x \neq \infty$
⟨*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 $\alpha 1\ \alpha 2\ \beta 1\ \beta 2\ \gamma\ \delta$ :: *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

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*: $f \hat{\ }\hat{\ } n = (\mathit{if}\ n = 0\ \mathit{then}\ \mathit{id}\ \mathit{else}\ f \circ f \hat{\ }\hat{\ } (n - 1))$
⟨*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 $f :: \mathit{int} \Rightarrow \mathit{int}$ **where**
$f\ x = (\mathit{if}\ x > 100\ \mathit{then}\ x - 10\ \mathit{else}\ 91)$

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

function $g :: \mathit{nat} \Rightarrow \mathit{int} \Rightarrow \mathit{int}$ **where**
$g\ n\ z = (\mathit{if}\ n = 0\ \mathit{then}\ z\ \mathit{else}\ \mathit{if}\ z > 100\ \mathit{then}\ g\ (n - 1)\ (z - 10)\ \mathit{else}\ g\ (n + 1)\ (z + 11))$
⟨*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*) = $\omega \hat{\ }$(*encode l*) + *encode r*

primrec $f :: \mathit{nat} \Rightarrow \mathit{lisp} \Rightarrow \mathit{lisp} \Rightarrow \mathit{lisp}$ **where**
$f\ 0\ y\ x = x$
| $f\ (\mathit{Suc}\ m)\ y\ x = \mathit{Cons}\ y\ (f\ m\ y\ x)$

lemma *encode_f*: *encode* ($f\ n\ y\ x$) = *of_nat n* ∗ $\omega \hat{\ }$(*encode y*) + *encode x*
⟨*proof*⟩

function $d :: \mathit{nat} \Rightarrow \mathit{lisp} \Rightarrow \mathit{lisp}$ **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* ≤ *m* **for** *m n* :: *nat*
⟨*proof*⟩

lemma *power_div_same_base*:
b ^ *y* ≠ *0* ⟹ *x* ≥ *y* ⟹ *b* ^ *x div b* ^ *y* = *b* ^ (*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* ≥ *2*
begin

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

lemma *well_base_filter*: *well_base M* ⟹ *well_base* {#*m* ∈# *M*. *p m*#}
⟨*proof*⟩

lemma *well_base_image_inj*: *well_base M* ⟹ *inj_on f* (*set_mset M*) ⟹ *well_base* (*image_mset f M*)
⟨*proof*⟩

lemma *well_base_bound*:
assumes
well_base M **and**
∀ *m* ∈# *M*. *m* < *n*
shows (∑ *m* ∈# *M*. *base* ^ *m*) < *base* ^ *n*
⟨*proof*⟩

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

lemma *well_base$_h$_mono_hmset*: *well_base$_h$ M* $\Longrightarrow$ *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* $\Rightarrow$ *nat* $\Rightarrow$ *hmultiset* **where**
encode e n =
(*if n* = *0 then 0 else of_nat* (*n mod base*) $*$ ω ^ (*encode 0 e*) + *encode* (*e* + *1*) (*n div base*))
⟨*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*) $*$ ω ^ (*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* ^ *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* ^ *decode 0 m*) *div base* ^ *e*

lemma *decode_unfold*: *decode e M* = ($\sum$ *m* $\in\#$ *hmsetmset M*. *base* ^ *decode 0 m*) *div base* ^ *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* ⟹ *aligned$_d$ e M*
⟨*proof*⟩

lemma *aligned$_d$_mono_hmset*:
assumes *aligned$_d$ e M* **and** *hmsetmset M′* ⊆# *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 × UNIV*
  ⟨*proof*⟩

lemma *infinite_cartesian_productI1*: *infinite A ⟹ B ≠ {} ⟹ infinite (A × B)*
  ⟨*proof*⟩

13.2 Types

datatype *type = ℬ (ℬ) | Fun type type* (**infixr** → *65*)

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

primrec *dest_fun* **where**
  dest_fun 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 →→ ℬ = T*
  ⟨*proof*⟩

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

primrec *δ* **where**
  δ 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
   (⋀*T*. (⋀*T1 T2*. *T = T1 → T2 ⟹ P T1*) ⟹

$(\bigwedge T1\ T2.\ T = T1 \to T2 \Longrightarrow P\ T2) \Longrightarrow P\ T)$
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*]:

$\Lambda\langle T\rangle\ r \cdot t \neq \langle xT\rangle \cdot ss$
$\Lambda\langle T\rangle\ r \cdot t \neq (i :: idx) \cdot ss$
$\Lambda\langle T\rangle\ r \cdot t \neq (b :: bool) \cdot ss$
$\Lambda\langle T\rangle\ r \cdot t \neq (e1\ ?\ e2) \cdot ss$
⟨*proof*⟩

lemma *App_Abs_eq_Apps_Abs*[*simp*]: $\Lambda\langle T\rangle\ r \cdot t = \Lambda\langle T'\rangle\ r' \cdot ss \longleftrightarrow T = T' \wedge r = r' \wedge ss = [t]$
⟨*proof*⟩

lemma *Apps_Var_neq_Apps_Abs*[*simp*, *symmetric*, *simp*]: $\langle xT\rangle \cdot ss \neq \Lambda\langle T\rangle\ r \cdot ts$
⟨*proof*⟩

lemma *Apps_Var_neq_Apps_beta*[*simp*, *THEN not_sym*, *simp*]:
$\langle xT\rangle \cdot ss \neq \Lambda\langle T\rangle\ r \cdot s \cdot ts$
⟨*proof*⟩

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

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

lemma *fold_eq_Fun_iff*[*simp*]:
$fold\ op \to (rev\ Ts)\ T = U \to V \longleftrightarrow$
$(Ts = [] \wedge T = U \to V \vee (\exists Us.\ Ts = U \# Us \wedge fold\ op \to (rev\ Us)\ T = V))$
⟨*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*]:
$subst\ xT\ u\ (f \cdot xs) = subst\ xT\ u\ f \cdot map\ (subst\ xT\ u)\ xs$

⟨*proof*⟩

end

context begin

private lemma *fresh_close_Var_id*: *fresh xT t* ⟹ *close_Var k xT t* = *t*
⟨*proof*⟩

lemma *subst_close_Var*:
xT ≠ *yU* ⟹ *fresh yU u* ⟹ *subst xT u* (*close_Var i yU t*) = *close_Var i yU* (*subst xT u t*)
⟨*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*: $f \cdot ts ::: T \longleftrightarrow$
$(\exists Ts.\ f ::: fold\ op \to (rev\ Ts)\ T \wedge list\_all2\ (op :::)\ ts\ Ts)$
⟨*proof*⟩

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

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

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

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

lemma *fv_constant0*[*simp*]: $fv\ (constant0\ T) = \{|(''bool'', \mathcal{B})|\}$
⟨*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*]: $constant\ T ::: \mathcal{B} \to T$
⟨*proof*⟩

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

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

abbreviation *ar* **where**
$ar\ T \equiv length\ (dest\_fun\ T)$

lemma *size_nth_arg*[*simp*]: $i < ar\ T \Longrightarrow size\ (T\ !-\ i) < size\ T$
⟨*proof*⟩

fun $\pi :: type \Rightarrow nat \Rightarrow nat \Rightarrow type$ **where**
$\pi\ T\ i\ 0 = (if\ i < ar\ T\ then\ nth\_drop\ i\ (dest\_fun\ T) \to\to \mathcal{B}\ else\ \mathcal{B})$
$|\ \pi\ T\ i\ (Suc\ j) = (if\ i < ar\ T \wedge j < ar\ (T!-i)$
$\quad then\ \pi\ (T!-i)\ j\ 0 \to$
$\quad\ map\ (\pi\ (T!-i)\ j\ o\ Suc)\ [0\ ..<\ ar\ (T!-i!-j)] \to\to \pi\ T\ i\ 0\ else\ \mathcal{B})$

theorem *π_induct*[*rotated −2*, *consumes 2*, *case_names 0 Suc*]:
assumes $\bigwedge T\ i.\ i < ar\ T \Longrightarrow P\ T\ i\ 0$
and $\bigwedge T\ i\ j.\ i < ar\ T \Longrightarrow j < ar\ (T\ !-\ i) \Longrightarrow P\ (T\ !-\ i)\ j\ 0 \Longrightarrow$
$(\forall x < ar\ (T\ !-\ i\ !-\ j).\ P\ (T\ !-\ i)\ j\ (x + 1)) \Longrightarrow P\ T\ i\ (j + 1)$
shows $i < ar\ T \Longrightarrow j \le ar\ (T\ !-\ i) \Longrightarrow P\ T\ i\ j$
⟨*proof*⟩

definition $\varepsilon :: type \Rightarrow nat \Rightarrow type$ **where**
$\varepsilon\ T\ i = \pi\ T\ i\ 0 \to map\ (\pi\ T\ i\ o\ Suc)\ [0\ ..<\ ar\ (T!-i)] \to\to T$

definition *Abss* ($\Lambda[\_]\ \_$ [*100*, *100*] *800*) **where**
$\Lambda[xTs]\ b = fold\ (\lambda xT\ t.\ \Lambda\langle snd\ xT\rangle\ close0\_\ Var\ xT\ t)\ (rev\ xTs)\ b$

definition *Seqs* (**infixr** *??* *75*) **where**
$ts\ ??\ t = fold\ (\lambda u\ t.\ u\ ?\ t)\ (rev\ ts)\ t$

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 (op :::) 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* = $\mathcal{B}$
⟨*proof*⟩

lemma *welltyped_ constant0_ iff* [*simp*]: *constant0 T* ::: *U* ⟷ (*U* = *T*)
⟨*proof*⟩

lemma *welltyped_ constant_ iff* [*simp*]: *constant T* ::: *U* ⟷ (*U* = $\mathcal{B}$ → *T*)
⟨*proof*⟩

lemma *welltyped_ Seq_ iff* [*simp*]: *e1 ? e2* ::: *T* ⟷ (*T* = $\mathcal{B}$ ∧ *e1* ::: $\mathcal{B}$ ∧ *e2* ::: $\mathcal{B}$)
⟨*proof*⟩

lemma *welltyped_ Seqs_ iff* [*simp*]: *es ??* *e* ::: *T* ⟷
((*es* ≠ []) ⟶ *T* = $\mathcal{B}$) ∧ (∀ *e* ∈ *set es*. *e* ::: $\mathcal{B}$) ∧ *e* ::: *T*)
⟨*proof*⟩

lemma *welltyped_ App_ iff* [*simp*]: *f* · *t* ::: *U* ⟷ (∃ *T*. *f* ::: *T* → *U* ∧ *t* ::: *T*)
⟨*proof*⟩

lemma *welltyped_ Apps_ iff* [*simp*]: *f* ⋆ *ts* ::: *U* ⟷ (∃ *Ts*. *f* ::: *Ts* →→ *U* ∧ *list_ all2* (*op* :::) *ts Ts*)
⟨*proof*⟩

lemma *eq_ mk_ fun_ iff* [*simp*]: *T* = *Ts* →→ $\mathcal{B}$ ⟷ *Ts* = *dest_ fun T*
⟨*proof*⟩

lemma *map_ nth_ eq_ drop_ take*[*simp*]: *j* ≤ *length xs* ⟹ *map* (*nth xs*) [*i* ..< *j*] = *drop i* (*take j xs*)
⟨*proof*⟩

lemma *dest_ fun_ π_ 0*: *i* < *ar T* ⟹ *dest_ fun* (*π T i 0*) = *nth_ drop i* (*dest_ fun T*)
⟨*proof*⟩

lemma *welltyped_ E*: *E T i* ::: *ε T i* **and** *welltyped_ P*: *P T i j* ::: *T* → *π T i j*
⟨*proof*⟩

lemma *δ_ gt_ 0*[*simp*]: *T* ≠ $\mathcal{B}$ ⟹ *HMSet* {#} < *δ T*
⟨*proof*⟩

lemma *mset_ nth_ drop_ less*: *i* < *length xs* ⟹ *mset* (*nth_ drop i xs*) < *mset xs*
⟨*proof*⟩

lemma *map_ nth_ drop*: *i* < *length xs* ⟹ *map f* (*nth_ drop i xs*) = *nth_ drop i* (*map f xs*)
⟨*proof*⟩

lemma *empty_ less_ mset*: {#} < *mset xs* ⟷ *xs* ≠ []
⟨*proof*⟩

lemma *dest_ fun_ alt*: *dest_ fun T* = *map* (λ*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* ≤ *ar* (*T* !− *i*)
shows *δ* (*π T i j*) < *δ T*
⟨*proof*⟩

end

end