Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

August 16, 2018

Abstract

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

Contents

1 Introduction **3**

2 More about Multisets **3**
2.1 Basic Setup . . . 3
2.2 Lemmas about Intersection, Union and Pointwise Inclusion . . . 3
2.3 Lemmas about Filter and Image . . . 3
2.4 Lemmas about Sum . . . 4
2.5 Lemmas about Remove . . . 5
2.6 Lemmas about Replicate . . . 6
2.7 Multiset and Set Conversions . . . 7
2.8 Duplicate Removal . . . 7
2.9 Repeat Operation . . . 9
2.10 Cartesian Product . . . 9
2.11 Transfer Rules . . . 12
2.12 Even More about Multisets . . . 13
2.12.1 Multisets and functions . . . 13
2.12.2 Multisets and lists . . . 13
2.12.3 More on multisets and functions . . . 13

3 Signed (Finite) Multisets **14**
3.1 Definition of Signed Multisets . . . 14
3.2 Basic Operations on Signed Multisets . . . 14
3.2.1 Conversion to Set and Membership . . . 16
3.2.2 Union . . . 17
3.2.3 Difference . . . 17
3.2.4 Equality of Signed Multisets . . . 18
3.3 Conversions from and to Multisets . . . 18
3.3.1 Pointwise Ordering Induced by *zcount* . . . 19
3.3.2 Subset is an Order . . . 21
3.4 Replicate and Repeat Operations . . . 21
3.4.1 Filter (with Comprehension Syntax) . . . 22
3.5 Uncategorized . . . 22
3.6 Image . . . 23
3.7 Multiset Order . . . 23

4 Nested Multisets **25**
4.1 Type Definition . . . 25
4.2 Dershowitz and Manna's Nested Multiset Order . . . 25

5 Hereditar(il)y (Finite) Multisets **27**
5.1 Type Definition . . . 27
5.2 Restriction of Dershowitz and Manna's Nested Multiset Order . . . 28
5.3 Disjoint Union and Truncated Difference . . . 28
5.4 Infimum and Supremum . . . 30
5.5 Inequalities . . . 30

6 Signed Hereditar(il)y (Finite) Multisets **31**
6.1 Type Definition . . . 31
6.2 Multiset Order . . . 31
6.3 Embedding and Projections of Syntactic Ordinals . . . 31
6.4 Disjoint Union and Difference . . . 32
6.5 Infimum and Supremum . . . 33

7 Syntactic Ordinals in Cantor Normal Form **33**
7.1 Natural (Hessenberg) Product . . . 33
7.2 Inequalities . . . 34
7.3 Embedding of Natural Numbers . . . 36
7.4 Embedding of Extended Natural Numbers . . . 36
7.5 Head Omega . . . 37
7.6 More Inequalities and Some Equalities . . . 37
7.7 Conversions to Natural Numbers . . . 40
7.8 An Example . . . 40

8 Signed Syntactic Ordinals in Cantor Normal Form **40**
8.1 Natural (Hessenberg) Product . . . 40
8.2 Embedding of Natural Numbers . . . 41
8.3 Embedding of Extended Natural Numbers . . . 41
8.4 Inequalities and Some (Dis)equalities . . . 42
8.5 An Example . . . 44

9 Bridge between Huffman's Ordinal Library and the Syntactic Ordinals **45**
9.1 Missing Lemmas about Huffman's Ordinals . . . 45
9.2 Embedding of Syntactic Ordinals into Huffman's Ordinals . . . 45

10 Termination of McCarthy's 91 Function **46**

11 Termination of the Hydra Battle **46**

12 Termination of the Goodstein Sequence **47**
12.1 Lemmas about Division . . . 47
12.2 Hereditary and Nonhereditary Base-n Systems . . . 47
12.3 Encoding of Natural Numbers into Ordinals . . . 48
12.4 Decoding of Natural Numbers from Ordinals . . . 48
12.5 The Goodstein Sequence and Goodstein's Theorem . . . 50

13 Towards Decidability of Behavioral Equivalence for Unary PCF **50**
13.1 Preliminaries . . . 50
13.2 Types . . . 50
13.3 Terms . . . 51
13.4 Substitution . . . 53
13.5 Typing . . . 54
13.6 Definition 10 and Lemma 11 from Schmidt-Schauß's paper . . . 55

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

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

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

declare *subset_msetE* [*elim*!]

2.3 Lemmas about Filter and Image

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

lemma *filter_mset_cong*[*fundef_cong*]:
assumes *M = M'* $\bigwedge$*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*:
($\bigwedge$*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* = {#}) = (∀ *L*∈#*M*. ¬ *P L*)›
⟨*proof*⟩

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

lemma *image_filter_cong*:
assumes ‹$\bigwedge$*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*∈# *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*⟩

declare *image_mset_cong2* [*cong*]

2.4 Lemmas about Sum

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

lemma *sum_image_mset_mono*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::*canonically_ordered_monoid_add*
assumes *sub*: *A* ⊆# *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* = (∑*x*∈#*M*. *if x = a then 1 else 0*)›
⟨*proof*⟩

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

lemma *sum_mset_distrib_div_if_dvd*:
fixes $k :: 'a::unique\_euclidean\_semiring$
assumes $\forall m \in\# M.\ k\ dvd\ f\ m$
shows $(\sum m \in\# M.\ f\ m)\ div\ k = (\sum m \in\# M.\ f\ m\ div\ k)$
⟨*proof*⟩

2.5 Lemmas about Remove

lemma *set_mset_minus_replicate_mset*[*simp*]:
$n \geq count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A - \{a\}$
$n < count\ A\ a \Longrightarrow set\_mset\ (A - replicate\_mset\ n\ a) = set\_mset\ A$
⟨*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*: $removeAll\_mset\ C\ M = filter\_mset\ ((\neq)\ C)\ M$
⟨*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 *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 $\Longrightarrow$ *distinct_ mset* (*C* $\cap\#$ *D*)
distinct_ mset D $\Longrightarrow$ *distinct_ mset* (*C* $\cap\#$ *D*)
⟨*proof*⟩

lemma *distinct_ mset_ remove1_ All*: *distinct_ mset C* $\Longrightarrow$ *remove1_ mset L C* = *removeAll_ mset L C*
⟨*proof*⟩

lemma *distinct_ mset_ size_ 2*: *distinct_ mset* {#*a*, *b*#} $\longleftrightarrow$ *a* ≠ *b*
⟨*proof*⟩

lemma *distinct_ mset_ filter*: *distinct_ mset M* $\Longrightarrow$ *distinct_ mset* {# *L* $\in\#$ *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*) $\Longrightarrow$ *distinct_ mset* (*image_ mset f M*) $\longleftrightarrow$ *distinct_ mset M*›
⟨*proof*⟩

2.9 Repeat Operation

lemma *repeat_ mset_ compower*: *repeat_ mset n A* = (((+) *A*) ^^ *n*) {#}
⟨*proof*⟩

lemma *repeat_ mset_ prod*: *repeat_ mset* (*m* ∗ *n*) *A* = (((+) (*repeat_ mset n A*)) ^^ *m*) {#}
⟨*proof*⟩

2.10 Cartesian Product

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

definition *Sigma_ mset* :: ′*a multiset* ⇒ (′*a* ⇒ ′*b multiset*) ⇒ (′*a* × ′*b*) *multiset* **where**
Sigma_ mset A B ≡ $\bigcup\#$ {#{#(*a*, *b*). *b* $\in\#$ *B a*#}. *a* $\in\#$ *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* × *B*, we use the non-ASCII symbol $\in\#$.

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

Link between the multiset and the set cartesian product:

lemma *Times_ mset_ Times*: *set_ mset* (*A* ×# *B*) = *set_ mset A* × *set_ mset B*
⟨*proof*⟩

lemma *Sigma_ msetI* [*intro*!]: ⟦*a* $\in\#$ *A*; *b* $\in\#$ *B a*⟧ $\Longrightarrow$ (*a*, *b*) $\in\#$ *Sigma_ mset A B*
⟨*proof*⟩

lemma *Sigma_ msetE*[*elim*!]: ⟦*c* $\in\#$ *Sigma_ mset A B*; $\bigwedge x\ y$. ⟦*x* $\in\#$ *A*; *y* $\in\#$ *B x*; *c* = (*x*, *y*)⟧ $\Longrightarrow$ *P*⟧ $\Longrightarrow$ *P*
⟨*proof*⟩

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

lemma *Sigma_ msetD1*: (*a*, *b*) $\in\#$ *Sigma_ mset A B* $\Longrightarrow$ *a* $\in\#$ *A*
⟨*proof*⟩

lemma *Sigma_msetD2*: $(a, b) \in\# \mathit{Sigma\_mset}\ A\ B \Longrightarrow b \in\# B\ a$
⟨*proof*⟩

lemma *Sigma_msetE2*: $[\![(a, b) \in\# \mathit{Sigma\_mset}\ A\ B; [\![a \in\# A; b \in\# B\ a]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *Sigma_mset_cong*:
$[\![A = B; \bigwedge x.\ x \in\# B \Longrightarrow C\ x = D\ x]\!] \Longrightarrow (\mathit{SIGMAMSET}\ x \in\# A.\ C\ x) = (\mathit{SIGMAMSET}\ x \in\# B.\ D\ x)$
⟨*proof*⟩

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

lemma *Sigma_mset_plus_distrib1*[*simp*]: $\mathit{Sigma\_mset}\ (A + B)\ C = \mathit{Sigma\_mset}\ A\ C + \mathit{Sigma\_mset}\ B\ C$
⟨*proof*⟩

lemma *Sigma_mset_plus_distrib2*[*simp*]:
$\mathit{Sigma\_mset}\ A\ (\lambda i.\ B\ i + C\ i) = \mathit{Sigma\_mset}\ A\ B + \mathit{Sigma\_mset}\ A\ C$
⟨*proof*⟩

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

lemma *Times_mset_single_right*: $A \times\# \{\#b\#\} = \mathit{image\_mset}\ (\lambda a.\ \mathit{Pair}\ a\ b)\ A$
⟨*proof*⟩

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

lemma *count_image_mset_Pair*:
$\mathit{count}\ (\mathit{image\_mset}\ (\mathit{Pair}\ a)\ B)\ (x, b) = (\mathit{if}\ x = a\ \mathit{then}\ \mathit{count}\ B\ b\ \mathit{else}\ 0)$
⟨*proof*⟩

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

lemma *Sigma_mset_empty1*[*simp*]: $\mathit{Sigma\_mset}\ \{\#\}\ B = \{\#\}$
⟨*proof*⟩

lemma *Sigma_mset_empty2*[*simp*]: $A \times\# \{\#\} = \{\#\}$
⟨*proof*⟩

lemma *Sigma_mset_mono*:
assumes $A \subseteq\# C$ **and** $\bigwedge x.\ x \in\# A \Longrightarrow B\ x \subseteq\# D\ x$
shows $\mathit{Sigma\_mset}\ A\ B \subseteq\# \mathit{Sigma\_mset}\ C\ D$
⟨*proof*⟩

lemma *mem_Sigma_mset_iff*[*iff*]: $((a,b) \in\# \mathit{Sigma\_mset}\ A\ B) = (a \in\# A \wedge b \in\# B\ a)$
⟨*proof*⟩

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

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

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

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

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

⟨*proof*⟩

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

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

lemma *sum_mset_if_eq_constant*:
(∑ *x*∈#*M*. *if a* = *x then* (*f x*) *else* 0) = (((+) (*f a*)) ^^ (*count M a*)) 0
⟨*proof*⟩

lemma *iterate_op_plus*: (((+) *k*) ^^ *m*) 0 = *k* ∗ *m*
⟨*proof*⟩

lemma *untion_image_mset_Pair_distribute*:
⋃#{#*image_mset* (*Pair x*) (*C x*). *x* ∈# *J* − *I*#} =
⋃# {#*image_mset* (*Pair x*) (*C x*). *x* ∈# *J*#} − ⋃#{#*image_mset* (*Pair x*) (*C x*). *x* ∈# *I*#}
⟨*proof*⟩

lemma *Sigma_mset_Un_distrib1*: *Sigma_mset* (*I* ∪# *J*) *C* = *Sigma_mset I C* ∪# *Sigma_mset J C*
⟨*proof*⟩

lemma *Sigma_mset_Un_distrib2*: (*SIGMAMSET i*∈#*I*. *A i* ∪# *B i*) = *Sigma_mset I A* ∪# *Sigma_mset I B*
⟨*proof*⟩

lemma *Sigma_mset_Int_distrib1*: *Sigma_mset* (*I* ∩# *J*) *C* = *Sigma_mset I C* ∩# *Sigma_mset J C*
⟨*proof*⟩

lemma *Sigma_mset_Int_distrib2*: (*SIGMAMSET i*∈#*I*. *A i* ∩# *B i*) = *Sigma_mset I A* ∩# *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*∈#*I*. *A i* − *B i*) = *Sigma_mset I A* − *Sigma_mset I B*
⟨*proof*⟩

lemma *Sigma_mset_Union*: *Sigma_mset* (⋃#*X*) *B* = (⋃# (*image_mset* (λ*A*. *Sigma_mset A B*) *X*))
⟨*proof*⟩

lemma *Times_mset_Un_distrib1*: (*A* ∪# *B*) ×# *C* = *A* ×# *C* ∪# *B* ×# *C*
⟨*proof*⟩

lemma *Times_mset_Int_distrib1*: (*A* ∩# *B*) ×# *C* = *A* ×# *C* ∩# *B* ×# *C*
⟨*proof*⟩

lemma *Times_mset_Diff_distrib1*: (*A* − *B*) ×# *C* = *A* ×# *C* − *B* ×# *C*
⟨*proof*⟩

lemma *Times_mset_empty*[*simp*]: *A* ×# *B* = {#} ⟷ *A* = {#} ∨ *B* = {#}
⟨*proof*⟩

lemma *Times_insert_left*: *A* ×# *add_mset x B* = *A* ×# *B* + *image_mset* (λ*a*. *Pair a x*) *A*
⟨*proof*⟩

lemma *Times_insert_right*: *add_mset a A* ×# *B* = *A* ×# *B* + *image_mset* (*Pair a*) *B*
⟨*proof*⟩

lemma *fst_image_mset_times_mset* [*simp*]:
image_mset fst (*A* ×# *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* ∈# *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*))) (+) (+)
⟨*proof*⟩

lemma *minus_multiset_transfer*[*transfer_rule*]:
assumes [*transfer_rule*]: *bi_unique R*
shows (*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_mset R*) (*rel_mset R*))) (−) (−)
⟨*proof*⟩

declare *rel_mset_Zero*[*transfer_rule*]

lemma *count_transfer*[*transfer_rule*]:
assumes *bi_unique R*
shows (*rel_fun* (*rel_mset R*) (*rel_fun R* (=))) *count count*
⟨*proof*⟩

lemma *subseteq_multiset_transfer*[*transfer_rule*]:
assumes [*transfer_rule*]: *bi_unique R right_total R*
shows (*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_mset R*) (=)))
(λ*M N*. *filter_mset* (*Domainp R*) *M* ⊆# *filter_mset* (*Domainp R*) *N*) (⊆#)
⟨*proof*⟩

lemma *sum_mset_transfer*[*transfer_rule*]:
R 0 0 ⟹ *rel_fun R* (*rel_fun R R*) (+) (+) ⟹ (*rel_fun* (*rel_mset R*) *R*) *sum_mset sum_mset*
⟨*proof*⟩

lemma *Sigma_mset_transfer*[*transfer_rule*]:

(*rel_fun* (*rel_mset R*) (*rel_fun* (*rel_fun R* (*rel_mset S*)) (*rel_mset* (*rel_prod R S*))))
Sigma_mset Sigma_mset
⟨*proof*⟩

2.12 Even More about Multisets

2.12.1 Multisets and functions

lemma *range_image_mset*:
assumes *set_mset Ds* ⊆ *range f*
shows *Ds* ∈ *range* (*image_mset f*)
⟨*proof*⟩

2.12.2 Multisets and lists

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

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

lemma *image_mset_of_subset*:
assumes $A \subseteq\# image\_mset\ \eta\ C'$
shows $\exists A'.\ image\_mset\ \eta\ A' = A \wedge A' \subseteq\# C'$
⟨*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* :: $'a\ multiset \times 'a\ multiset \Rightarrow 'a\ multiset \times 'a\ multiset \Rightarrow bool$ **where**
$equiv\_zmset = (\lambda(Mp,\ Mn)\ (Np,\ Nn).\ Mp + Nn = Np + Mn)$

quotient-type $'a\ zmultiset = 'a\ multiset \times 'a\ multiset\ /\ equiv\_zmset$
⟨*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**
$empty\_zmset \equiv 0$

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

lift-definition *plus_zmultiset* :: $'a\ zmultiset \Rightarrow 'a\ zmultiset \Rightarrow 'a\ zmultiset$ **is**
$\lambda(Mp,\ Mn)\ (Np,\ Nn).\ (Mp + Np,\ Mn + Nn)$
⟨*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*]: $\{\#x\#\}_z \neq -\{\#y\#\}_z$ **and**
uminus_singleton_ne_singleton_zmset[*simp*]: $-\{\#x\#\}_z \neq \{\#y\#\}_z$
⟨*proof*⟩

3.2.1 Conversion to Set and Membership

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

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

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

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

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

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

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

context
begin

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

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

end

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

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

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

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

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

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

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

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

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

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

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

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

lemma *zmultiset_nonemptyE*[*elim*]:
assumes *A ≠ {#}z*
obtains *x* **where** *x ∈#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 ∈#z M − N ⟷ zcount N a ≠ 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**

$A \subseteq\#_z\ B \longleftrightarrow (\forall a.\ zcount\ A\ a \leq zcount\ B\ a)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *subset_mset_zmsetE*:
assumes *M* $\subset\#_z$ *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* $\subset\#$ *B*
⟨*proof*⟩

lemma *subseteq_mset_zmsetE*:
assumes *M* $\subseteq\#_z$ *N*
obtains *A B C* **where**
M = *zmset_of A* + *C* **and** *N* = *zmset_of B* + *C* **and** *A* $\subseteq\#$ *B*
⟨*proof*⟩

3.3.2 Subset is an Order

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

3.4 Replicate and Repeat Operations

definition *replicate_zmset* :: *nat* ⇒ *'a* ⇒ *'a zmultiset* **where**
replicate_zmset n x = (*add_zmset x* ^^ *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* ≥ *j* ⟹ *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*: $M = \{\#x \in\#_z\ M.\ P\ x\ \#\} + \{\#x \in\#_z\ M.\ \neg\ P\ x\#\}$
⟨*proof*⟩

3.6 Image

definition *image_zmset* :: $('a \Rightarrow 'b) \Rightarrow 'a\ zmultiset \Rightarrow 'b\ zmultiset$ **where**
image_zmset f M =
zmset_of (*fold_mset* (*add_mset* ∘ *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*]: $\langle -a + b = b - a\rangle$ **for** $a :: \langle 'a\ zmultiset\rangle$
⟨*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: $set\_mset\ N \subseteq A$ **and** M_A: $set\_mset\ M \subseteq A$ **and**
$trans$: $\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ R\ x\ y \longrightarrow R\ y\ z \longrightarrow R\ x\ z$
shows $less\_multiset\_ext_{DM}\ R\ M\ N \longleftrightarrow (M,\ N) \in mult\ \{(x,\ y).\ x \in A \wedge y \in A \wedge R\ x\ y\}$
⟨*proof*⟩

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

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

function $less\_nmultiset$ :: $'a\ nmultiset \Rightarrow 'a\ nmultiset \Rightarrow bool$ **where**
$less\_nmultiset\ (Elem\ a)\ (Elem\ b) \longleftrightarrow a < b$
| $less\_nmultiset\ (Elem\ a)\ (MSet\ M) \longleftrightarrow True$
| $less\_nmultiset\ (MSet\ M)\ (Elem\ a) \longleftrightarrow False$
| $less\_nmultiset\ (MSet\ M)\ (MSet\ N) \longleftrightarrow less\_multiset\_ext_{DM}\ less\_nmultiset\ M\ N$
⟨*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 $trans\_less\_nmultiset$: $X < Y \Longrightarrow Y < Z \Longrightarrow X < Z$ **for** $X\ Y\ Z$ :: $'a\ nmultiset$
⟨*proof*⟩

lemma $irrefl\_less\_nmultiset$:
fixes X :: $'a\ nmultiset$
shows $X < X \Longrightarrow False$
⟨*proof*⟩

lemma $antisym\_less\_nmultiset$:
fixes $X\ Y$ :: $'a\ nmultiset$
shows $X < Y \Longrightarrow Y < X \Longrightarrow False$
⟨*proof*⟩

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

instance
⟨*proof*⟩

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

end

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

instance
⟨*proof*⟩

end

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

lemma *total_less_nmultiset*:
  fixes $X\ Y :: 'a\ nmultiset$
  shows $\neg\ X < Y \Longrightarrow Y \neq X \Longrightarrow Y < X$
⟨*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* ⇒ *unit nmultiset* **where**
  Rep_hmultiset (*HMSet M*) = *MSet* (*image_mset Rep_hmultiset M*)

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

lemma *type_definition_hmultiset*: *type_definition Rep_hmultiset Abs_hmultiset* {*X*. *no_elem X*}
⟨*proof*⟩

setup-lifting *type_definition_hmultiset*

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

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

5.2 Restriction of Dershowitz and Manna's Nested Multiset Order

instantiation *hmultiset* :: *linorder*
begin

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

instance
⟨*proof*⟩

end

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

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

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

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

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

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

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

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

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

5.3 Disjoint Union and Truncated Difference

instantiation *hmultiset* :: *cancel_comm_monoid_add*
begin

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

lemma *hmsetmset_empty_iff*[*simp*]: *hmsetmset n* = {#} ⟷ *n* = *0*
⟨*proof*⟩

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

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

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

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

instance
⟨*proof*⟩

end

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

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

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

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

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

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

⟨*ML*⟩

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

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

instantiation *hmultiset* :: *order_bot*
begin

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

instance
⟨*proof*⟩

end

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

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

5.4 Infimum and Supremum

instantiation *hmultiset* :: *distrib_lattice*
begin

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

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

instance
⟨*proof*⟩

end

5.5 Inequalities

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end

6 Signed Hereditar(il)y (Finite) Multisets

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

6.1 Type Definition

typedef *zhmultiset = UNIV :: hmultiset zmultiset set*
  morphisms *zhmsetmset ZHMSet*
  ⟨*proof*⟩

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

declare
  zhmsetmset_ inverse [simp]
  zhmsetmset_ inject [simp]

setup-lifting *type_ definition_ zhmultiset*

6.2 Multiset Order

instantiation *zhmultiset :: linorder*
begin

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

instance
  ⟨*proof*⟩

end

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

6.3 Embedding and Projections of Syntactic Ordinals

abbreviation *zhmset_ of :: hmultiset ⇒ zhmultiset* **where**
  zhmset_ of M ≡ ZHMSet (zmset_ of (hmsetmset M))

lemma *zhmset_ of_ inject[simp]*: *zhmset_ of M = zhmset_ of N ⟷ M = N*
  ⟨*proof*⟩

lemma *zhmset_ of_ less*: *zhmset_ of M < zhmset_ of N ⟷ M < N*
  ⟨*proof*⟩

lemma *zhmset_ of_ le*: *zhmset_ of M ≤ zhmset_ of N ⟷ M ≤ N*
  ⟨*proof*⟩

abbreviation *hmset_ pos :: zhmultiset ⇒ hmultiset* **where**
  hmset_ pos M ≡ HMSet (mset_ pos (zhmsetmset M))

abbreviation *hmset_ neg :: zhmultiset ⇒ hmultiset* **where**
  hmset_ neg M ≡ HMSet (mset_ neg (zhmsetmset M))

6.4 Disjoint Union and Difference

instantiation *zhmultiset* :: *cancel_ comm_ monoid_ add*
begin

lift-definition *zero_ zhmultiset* :: *zhmultiset* **is** $\{\#\}_z$ ⟨*proof*⟩

lift-definition *plus_ zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* ⇒ *zhmultiset* **is**
$\lambda A\ B.\ A + B$ ⟨*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** $\lambda A.\ -\ A$ ⟨*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^* ≡ $\lambda m.$ *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* (+)) (*hmsetmset A* ×# *hmsetmset B*))

lemma *hmsetmset_times*:
hmsetmset $(m * n) =$ *image_mset* (*case_prod* (+)) (*hmsetmset* $m \times\#$ *hmsetmset* n)
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

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

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

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

7.2 Inequalities

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

lemma *plus_nmultiset_mono*:
assumes *less*: $(X, Y) < (X', Y')$ **and** *no_elem*: *no_elem* X *no_elem* Y *no_elem* X' *no_elem* Y'
shows *plus_nmultiset* X $Y <$ *plus_nmultiset* X' Y'
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma
less_imp_minus_plus_hmset: $m < n \Longrightarrow k < k - m + n$ **and**
le_imp_minus_plus_hmset: $m \leq n \Longrightarrow k \leq k - m + n$ **for** $k\ m\ n$ :: *hmultiset*
⟨*proof*⟩

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

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

7.3 Embedding of Natural Numbers

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

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

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

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

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

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

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

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

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

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

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

7.4 Embedding of Extended Natural Numbers

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

lemma *hmset_of_enat_0*[*simp*]: *hmset_of_enat 0* = *0*
⟨*proof*⟩

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

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

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

⟨*proof*⟩

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

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

7.5 Head Omega

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

lemma *head_ω_subseteq*: *hmsetmset* (*head_ω M*) ⊆# *hmsetmset M*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

7.6 More Inequalities and Some Equalities

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

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

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

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

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

lemma *numeral_le_ω*[*simp*]: $numeral\ n \leq \omega$
⟨*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 $\alpha 1\ \alpha 2\ \beta 1\ \beta 2\ \gamma\ \delta$ :: *hmultiset*
assumes
$\alpha\beta 2\gamma$_ *lt*_ $\alpha\beta 1\gamma$: $\alpha 2 + \beta 2 * \gamma < \alpha 1 + \beta 1 * \gamma$ **and**
$\beta 2$_ *le*_ $\beta 1$: $\beta 2 \leq \beta 1$ **and**
γ_ *lt*_ δ: $\gamma < \delta$
shows $\alpha 2 + \beta 2 * \delta < \alpha 1 + \beta 1 * \delta$
⟨*proof*⟩

end

8 Signed Syntactic Ordinals in Cantor Normal Form

theory *Signed_ Syntactic_ Ordinal*
imports *Signed_ Hereditary_ Multiset Syntactic_ Ordinal*
begin

8.1 Natural (Hessenberg) Product

instantiation *zhmultiset* :: *comm_ ring_ 1*
begin

abbreviation ω_z_ *exp* :: *hmultiset* ⇒ *zhmultiset* (ω_z ^) **where**
ω_z ^ ≡ λ*m*. *ZHMSet* $\{\#m\#\}_z$

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

abbreviation ω_z :: *zhmultiset* **where**
$\omega_z \equiv \omega_z\,\hat{}\,1$

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

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

lemmas *zhmsetmset_times* = *times_zhmultiset.rep_eq*

instance
⟨*proof*⟩

end

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

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

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

8.2 Embedding of Natural Numbers

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

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

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

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

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

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

8.3 Embedding of Extended Natural Numbers

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

lemma *zhmset_of_enat_0*[*simp*]: *zhmset_of_enat* 0 = 0
⟨*proof*⟩

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

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

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

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

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

8.4 Inequalities and Some (Dis)equalities

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

instantiation *zhmultiset* :: *linordered_idom*
begin

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

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

lemma *gt_0_times_gt_0_imp*:
fixes *a b* :: *zhmultiset*
assumes *a_gt0*: *a* > *0* **and** *b_gt0*: *b* > *0*
shows *a* ∗ *b* > *0*
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

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

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

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

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

lemma *gt_0_lt_mult_gt_1_zhmset*:
fixes *m n* :: *zhmultiset*
assumes *m* > *0* **and** *n* > *1*

shows $m < m * n$
⟨*proof*⟩

lemma *zero_less_iff_1_le_zhmset*: $0 < n \longleftrightarrow 1 \leq n$ **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*]: ¬ $\omega_z \leq$ *of_nat n*
⟨*proof*⟩

lemma *not_ω_z_le_numeral*[*simp*]: ¬ $\omega_z \leq$ *numeral n*
⟨*proof*⟩

lemma *zero_ne_ω_z*[*simp*]: *0* ≠ ω_z
⟨*proof*⟩

lemma *one_ne_ω_z*[*simp*]: *1* ≠ ω_z
⟨*proof*⟩

lemma *numeral_ne_ω_z*[*simp*]: *numeral n* ≠ ω_z
⟨*proof*⟩

lemma
ω_z*_ne_0*[*simp*]: $\omega_z \neq$ *0* **and**
ω_z*_ne_1*[*simp*]: $\omega_z \neq$ *1* **and**
ω_z*_ne_of_nat*[*simp*]: $\omega_z \neq$ *of_nat m* **and**
ω_z*_ne_numeral*[*simp*]: $\omega_z \neq$ *numeral n*
⟨*proof*⟩

lemma
zhmset_of_enat_inject[*simp*]: *zhmset_of_enat m* = *zhmset_of_enat n* ⟷ *m* = *n* **and**
zhmset_of_enat_lt_iff_lt[*simp*]: *zhmset_of_enat m* < *zhmset_of_enat n* ⟷ *m* < *n* **and**
zhmset_of_enat_le_iff_le[*simp*]: *zhmset_of_enat m* ≤ *zhmset_of_enat n* ⟷ *m* ≤ *n*
⟨*proof*⟩

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

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

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

8.5 An Example

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

lemma
fixes *α1 α2 β1 β2 γ δ* :: *hmultiset*
assumes
αβ2γ_lt_αβ1γ: *α2 + β2 * γ < α1 + β1 * γ* **and**
β2_le_β1: *β2 ≤ β1* **and**
γ_lt_δ: *γ < δ*
shows *α2 + β2 * δ < α1 + β1 * δ*
⟨*proof*⟩

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*: $k < l \Longrightarrow$ *ordinal_of_hmset* $k <$ *ordinal_of_hmset* l
⟨*proof*⟩

lemma *ordinal_of_hmset_less*[*simp*]: *ordinal_of_hmset* $k <$ *ordinal_of_hmset* $l \longleftrightarrow k < l$
⟨*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 (Suc m) y x = Cons y (f m y x)

lemma encode_f: encode (f n y x) = of_nat n * ω ^ (encode y) + encode x
  ⟨proof⟩

function d :: nat ⇒ lisp ⇒ lisp where
  d n x =
   (if car x = Nil then cdr x
    else if car (car x) = Nil then f n (cdr (car x)) (cdr x)
    else Cons (d n (car x)) (cdr x))
  ⟨proof⟩
termination
  ⟨proof⟩

declare d.simps[simp del]

function h :: nat ⇒ lisp ⇒ lisp where
  h n x = (if x = Nil then Nil else h (n + 1) (d n x))
  ⟨proof⟩
termination
⟨proof⟩

declare h.simps[simp del]

end
```

12 Termination of the Goodstein Sequence

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

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

12.1 Lemmas about Division

```
lemma div_mult_le: m div n * n ≤ 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**
$\forall m \in\# M.\ m < n$
shows $(\sum m \in\# M.\ base\ \hat{}\ m) < base\ \hat{}\ n$
⟨*proof*⟩

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

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

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

12.3 Encoding of Natural Numbers into Ordinals

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

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

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

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

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

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

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

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

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

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

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

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

12.4 Decoding of Natural Numbers from Ordinals

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

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

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

inductive $aligned_d$ :: *nat ⇒ hmultiset ⇒ bool* **where**
(∀ *m* ∈# *hmsetmset M*. *decode 0 m ≥ e*) ⟹ $aligned_d$ *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* (→) (*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*: $\delta\ (Ts \twoheadrightarrow T) = HMSet\ (hmsetmset\ (\delta\ T) + mset\ (map\ \delta\ Ts))$
⟨*proof*⟩

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

13.3 Terms

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

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

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

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

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

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

abbreviation *close0_Var* ≡ *close_Var 0*

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

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

lemma *ex_fresh*: ∃ *x*. (*x* :: *char list*, *T*) |∉| *A*
⟨*proof*⟩

inductive *lc* **where**
lc_Var[*simp*]: *lc* ⟨*xT*⟩

| *lc_B*[*simp*]: *lc* (*b* :: *bool*)
| *lc_Seq*: *lc e1* ⟹ *lc e2* ⟹ *lc* (*e1* ? *e2*)
| *lc_App*: *lc e1* ⟹ *lc e2* ⟹ *lc* (*e1* · *e2*)
| *lc_Abs*: (∀*x*. (*x*, *T*) |∉| *X* ⟶ *lc* (*open0_ Var* (*x*, *T*) *e*)) ⟹ *lc* (Λ⟨*T*⟩ *e*)

declare *lc.intros*[*intro*]

definition *body T t* ≡ (∃*X*. ∀*x*. (*x*, *T*) |∉| *X* ⟶ *lc* (*open0_ Var* (*x*, *T*) *t*))

lemma *lc_Abs_iff_body*: *lc* (Λ⟨*T*⟩ *t*) ⟷ *body T t*
⟨*proof*⟩

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

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

lemma *close_Var_open_Var*[*simp*]: *fresh xT t* ⟹ *close_ Var i xT* (*open_ Var i xT t*) = *t*
⟨*proof*⟩

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

context begin

private lemma *open_Var_open_Var_close_Var*: *i* ≠ *j* ⟹ *xT* ≠ *yU* ⟹ *fresh yU t* ⟹
open_ Var i yU (*open_ Var j zV* (*close_ Var j xT t*)) = *open_ Var j zV* (*close_ Var j xT* (*open_ Var i yU t*))
⟨*proof*⟩

lemma *open_Var_close_Var*[*simp*]: *lc t* ⟹ *open_ Var i xT* (*close_ Var i xT t*) = *t*
⟨*proof*⟩

end

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

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

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

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

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

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

$(f \cdot ts = \Lambda\langle T\rangle\ t) = (\Lambda\langle T\rangle\ t = f \wedge ts = [])$
⟨*proof*⟩

lemma *Apps_Var_eq*[*simp*]: $\langle xT\rangle \cdot ss = \langle yU\rangle \cdot ts \longleftrightarrow xT = yU \wedge ss = ts$
⟨*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*]:
$fold\ (\rightarrow)\ (rev\ Ts)\ T = \mathcal{B} \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
$\mathcal{B} = fold\ (\rightarrow)\ (rev\ Ts)\ T \longleftrightarrow Ts = [] \wedge T = \mathcal{B}$
⟨*proof*⟩

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

13.4 Substitution

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

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

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

context begin

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

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

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

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

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

end

context begin

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

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

end

lemma *subst_intro*: $fresh\ xT\ t \Longrightarrow lc\ u \Longrightarrow open0\ u\ t = subst\ xT\ u\ (open0\_Var\ xT\ t)$
⟨*proof*⟩

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

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

lemma *lc_open_Var*: $lc\ u \Longrightarrow lc\ (open\_Var\ i\ xT\ u)$
⟨*proof*⟩

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

13.5 Typing

inductive *welltyped* :: $expr \Rightarrow type \Rightarrow bool$ (**infix** ::: 60) **where**
welltyped_Var[*intro*!]: $\langle (x,\ T) \rangle ::: T$
| *welltyped_B*[*intro*!]: $(b :: bool) ::: \mathcal{B}$
| *welltyped_Seq*[*intro*!]: $e1 ::: \mathcal{B} \Longrightarrow e2 ::: \mathcal{B} \Longrightarrow e1\ ?\ e2 ::: \mathcal{B}$
| *welltyped_App*[*intro*]: $e1 ::: T \to U \Longrightarrow e2 ::: T \Longrightarrow e1 \cdot e2 ::: U$
| *welltyped_Abs*[*intro*]: $(\forall x.\ (x,\ T)\ |\notin|\ X \longrightarrow open0\_Var\ (x,\ T)\ e ::: U) \Longrightarrow \Lambda\langle T \rangle\ e ::: T \to U$

inductive-cases *welltypedE*[*elim*!]:
$\langle x \rangle ::: T$
$(i :: idx) ::: T$
$(b :: bool) ::: T$
$e1\ ?\ e2 ::: T$
$e1 \cdot e2 ::: T$
$\Lambda\langle T \rangle\ e ::: U$

lemma *welltyped_unique*: $t ::: T \Longrightarrow t ::: U \Longrightarrow T = U$
⟨*proof*⟩

lemma *welltyped_lc*[*simp*]: $t ::: T \Longrightarrow lc\ t$
⟨*proof*⟩

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

lemma *rename_ welltyped*: $u ::: U \Longrightarrow subst\ (x,\ T)\ \langle(y,\ T)\rangle\ u ::: U$
⟨*proof*⟩

lemma *welltyped_ Abs_ fresh*:
assumes *fresh* $(x,\ T)\ u$ *open0_ Var* $(x,\ T)\ u ::: U$
shows $\Lambda\langle T\rangle\ u ::: T \to U$
⟨*proof*⟩

lemma *Apps_ alt*: $f \cdot ts ::: T \longleftrightarrow$
$(\exists\, Ts.\ f ::: fold\ (\to)\ (rev\ Ts)\ T \wedge list\_ all2\ (:::)\ ts\ Ts)$
⟨*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**
$\pi\ T\ i\ 0 = (if\ i < ar\ T\ then\ nth\_ drop\ i\ (dest\_ fun\ T) \to\!\!\to \mathcal{B}\ else\ \mathcal{B})$
| $\pi\ T\ i\ (Suc\ j) = (if\ i < ar\ T \wedge j < ar\ (T!-i)$
$then\ \pi\ (T!-i)\ j\ 0 \to$
$map\ (\pi\ (T!-i)\ j\ o\ Suc)\ [0\ ..<\ ar\ (T!-i!-j)] \to\!\!\to \pi\ T\ i\ 0\ else\ \mathcal{B})$

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* (Λ[_] _ [100, 100] 800) **where**
  Λ[xTs] b = fold (λxT t. Λ⟨snd xT⟩ close0_ Var xT t) (rev xTs) b

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

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

lemma *variant_inj*: *variant i base* = *variant j base* ⟹ *i* = *j*
  ⟨*proof*⟩

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

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

lemma *Abss_Nil*[*simp*]: Λ[[]] *b* = *b*
  ⟨*proof*⟩

lemma *Abss_Cons*[*simp*]: Λ[(*x*#*xs*)] *b* = Λ⟨*snd x*⟩ (*close0_ Var x* (Λ[*xs*] *b*))
  ⟨*proof*⟩

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

lemma *welltyped_Apps*: *list_all2* (:::) *ts Ts* $\Longrightarrow$ *f* ::: *Ts* $\rightarrow\!\!\rightarrow$ *U* $\Longrightarrow$ *f* $\cdot$ *ts* ::: *U*
⟨*proof*⟩

lemma *welltyped_open_Var_close_Var*[*intro*!]:
t ::: *T* $\Longrightarrow$ *open0_Var x T* (*close0_Var x T t*) ::: *T*
⟨*proof*⟩

lemma *welltyped_Var_iff*[*simp*]:
⟨(*x*, *T*)⟩ ::: *U* $\longleftrightarrow$ *T* = *U*
⟨*proof*⟩

lemma *welltyped_bool_iff*[*simp*]: (*b* :: *bool*) ::: *T* $\longleftrightarrow$ *T* = $\mathcal{B}$
⟨*proof*⟩

lemma *welltyped_constant0_iff*[*simp*]: *constant0 T* ::: *U* $\longleftrightarrow$ (*U* = *T*)
⟨*proof*⟩

lemma *welltyped_constant_iff*[*simp*]: *constant T* ::: *U* $\longleftrightarrow$ (*U* = $\mathcal{B}$ $\rightarrow$ *T*)
⟨*proof*⟩

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

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

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

lemma *welltyped_Apps_iff*[*simp*]: *f* $\cdot$ *ts* ::: *U* $\longleftrightarrow$ ($\exists$ *Ts*. *f* ::: *Ts* $\rightarrow\!\!\rightarrow$ *U* $\wedge$ *list_all2* (:::) *ts Ts*)
⟨*proof*⟩

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

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

lemma *dest_fun_π_0*: *i* < *ar T* $\Longrightarrow$ *dest_fun* (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* $\rightarrow$ π *T i j*
⟨*proof*⟩

lemma δ*_gt_0*[*simp*]: *T* $\neq$ $\mathcal{B}$ $\Longrightarrow$ *HMSet* {#} < δ *T*
⟨*proof*⟩

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

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

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

lemma *dest_fun_alt*: *dest_fun T* = *map* (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 $i < ar\ T\ j \leq ar\ (T\ !-\ i)$
  shows $\delta\ (\pi\ \ T\ i\ j) < \delta\ \ T$
⟨*proof*⟩

end

end