Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

September 18, 2020

Abstract

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

Contents

1 Introduction

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

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

We refer to the following conference paper for details:

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

2 More about Multisets

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

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

2.1 Basic Setup

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

mset_subset_eqD[*dest*, *intro?*]

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

sum_mset_sum_list[*simp*]

2.2 Lemmas about Intersection, Union and Pointwise Inclusion

lemma *subset_mset_imp_subset_add_mset*: $A \subseteq\# B \Longrightarrow A \subseteq\# add\_mset\ x\ B$
⟨*proof*⟩

lemma *subset_add_mset_notin_subset_mset*: ‹$A \subseteq\# add\_mset\ b\ B \Longrightarrow b \notin\# A \Longrightarrow A \subseteq\# B$›
⟨*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*!]

lemma *subseq_ mset_ subseteq_ mset*: *subseq xs ys* $\Longrightarrow$ *mset xs* $\subseteq\#$ *mset ys*
⟨*proof*⟩

2.3 Lemmas about Filter and Image

lemma *count_ image_ mset_ ge_ count*: *count* (*image_ mset f A*) (*f b*) $\geq$ *count A b*
⟨*proof*⟩

lemma *count_ image_ mset_ inj*:
assumes ‹*inj f*›
shows ‹*count* (*image_ mset f M*) (*f x*) = *count M x*›
⟨*proof*⟩

lemma *count_ image_ mset_ le_ count_ inj_ on*:
inj_ on f (*set_ mset M*) $\Longrightarrow$ *count* (*image_ mset f M*) *y* $\leq$ *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* = {#}) = ($\forall$ *L*$\in\#$*M*. $\neg$ *P L*)›
⟨*proof*⟩

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

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

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

declare *image_ mset_ cong2* [*cong*]

lemma *filter_ mset_ empty_ if_ finite_ and_ filter_ set_ empty*:
assumes
{*x* $\in$ *X*. *P x*} = {} **and**
finite X
shows {#*x* $\in\#$ *mset_ set X*. *P x*#} = {#}
⟨*proof*⟩

2.4 Lemmas about Sum

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

lemma *sum_image_mset_mono*:
fixes *f* :: *'a* ⇒ *'b*::*canonically_ordered_monoid_add*
assumes *sub*: *A* ⊆# *B*
shows (∑ *m* ∈# *A*. *f m*) ≤ (∑ *m* ∈# *B*. *f m*)
⟨*proof*⟩

lemma *sum_image_mset_mono_mem*:
n ∈# *M* ⟹ *f n* ≤ (∑ *m* ∈# *M*. *f m*) **for** *f* :: *'a* ⇒ *'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* ∈# *M*. *k dvd f m*
shows *k dvd* (∑ *m* ∈# *M*. *f m*)
⟨*proof*⟩

lemma *sum_mset_distrib_div_if_dvd*:
fixes *k* :: *'a*::*unique_euclidean_semiring*
assumes ∀ *m* ∈# *M*. *k dvd f m*
shows (∑ *m* ∈# *M*. *f m*) *div k* = (∑ *m* ∈# *M*. *f m div k*)
⟨*proof*⟩

2.5 Lemmas about Remove

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

lemma *remove_1_mset_id_iff_notin*: *remove1_mset a M* = *M* ⟷ *a* ∉# *M*
⟨*proof*⟩

lemma *id_remove_1_mset_iff_notin*: *M* = *remove1_mset a M* ⟷ *a* ∉# *M*
⟨*proof*⟩

lemma *remove1_mset_eqE*:
remove1_mset L x1 = *M* ⟹
(*L* ∈# *x1* ⟹ *x1* = *M* + {#*L*#} ⟹ *P*) ⟹
(*L* ∉# *x1* ⟹ *x1* = *M* ⟹ *P*) ⟹
P
⟨*proof*⟩

lemma *image_filter_ne_mset*[*simp*]:
image_mset f {#*x* ∈# *M*. *f x* ≠ *y*#} = *removeAll_mset y* (*image_mset f M*)
⟨*proof*⟩

lemma *image_mset_remove1_mset_if*:
image_mset f (*remove1_mset a M*) =
(*if a* ∈# *M then remove1_mset* (*f a*) (*image_mset f M*) *else image_mset f M*)
⟨*proof*⟩

lemma *filter_mset_neq*: {#*x* ∈# *M*. *x* ≠ *y*#} = *removeAll_mset y M*
⟨*proof*⟩

lemma *filter_mset_neq_cond*: {#*x* ∈# *M*. *P x* ∧ *x* ≠ *y*#} = *removeAll_mset y* {# *x*∈#*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* ∈# *B* ∧ *b* ∈# *A* ∧ *count A b* ≥ *count B b then* {#*b*#} + (*A* − *B*) *else A* − *B*)
⟨*proof*⟩

lemma *add_mset_eq_add_mset_ne*:
a ≠ *b* ⟹ *add_mset a A* = *add_mset b B* ⟷ *a* ∈# *B* ∧ *b* ∈# *A* ∧ *A* = *add_mset b* (*B* − {#*a*#})
⟨*proof*⟩

lemma *add_mset_eq_add_mset*: ‹*add_mset a M* = *add_mset b M*′ ⟷
(*a* = *b* ∧ *M* = *M*′) ∨ (*a* ≠ *b* ∧ *b* ∈# *M* ∧ *add_mset a* (*M* − {#*b*#}) = *M*′)›
⟨*proof*⟩

lemma *add_mset_remove_trivial_iff*: ‹*N* = *add_mset a* (*N* − {#*b*#}) ⟷ *a* ∈# *N* ∧ *a* = *b*›
⟨*proof*⟩

lemma *trivial_add_mset_remove_iff*: ‹*add_mset a* (*N* − {#*b*#}) = *N* ⟷ *a* ∈# *N* ∧ *a* = *b*›
⟨*proof*⟩

lemma *remove1_single_empty_iff*[*simp*]: ‹*remove1_mset L* {#*L*′#} = {#} ⟷ *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 ⊂# replicate_mset n x ⟷ m < n*
⟨*proof*⟩

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

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

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

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

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

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

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

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

2.7 Multiset and Set Conversions

lemma *count_mset_set_if*: *count (mset_set A) a = (if a ∈ A ∧ finite A then 1 else 0)*
⟨*proof*⟩

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

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

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

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

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

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

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

2.8 Duplicate Removal

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

lemma *set_mset_remdups_mset*[*simp*]: ‹*set_mset* (*remdups_mset A*) = *set_mset A*›
⟨*proof*⟩

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

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

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

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

lemma *remdups_mset_singleton_sum*[*simp*]:
remdups_mset (*add_mset a A*) = (*if a* ∈# *A then remdups_mset A else add_mset a* (*remdups_mset A*))
⟨*proof*⟩

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

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

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* $\Longrightarrow$ *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* $\subseteq$ *N* $\longleftrightarrow$ *M* $\subseteq\#$ *mset_set N*
⟨*proof*⟩

lemma *distinct_mem_diff_mset*:
assumes *dist*: *distinct_mset M* **and** *mem*: *x* $\in$ *set_mset* (*M* − *N*)
shows *x* $\notin$ *set_mset N*
⟨*proof*⟩

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* $\cup\#$ *C*) $\longleftrightarrow$ *distinct_mset D* $\wedge$ *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* $\neq$ *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 $A \times B$, we use the non-ASCII symbol $\in\#$.

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

Link between the multiset and the set cartesian product:

lemma *Times_mset_Times*: $set\_mset\ (A \times\#\ B) = set\_mset\ A \times set\_mset\ B$
⟨*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 $(a, b) \in\#\ A \times\#\ B$ – introduces no eigenvariables.

lemma *Sigma_msetD1*: $(a, b) \in\#\ Sigma\_mset\ A\ B \Longrightarrow a \in\#\ A$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Sigma_mset_Union*: *Sigma_mset* $(\bigcup\#X)\ B = (\bigcup\#\ (image\_mset\ (\lambda A.\ Sigma\_mset\ A\ B)\ X))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

context
begin

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

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

end

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

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

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

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

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

2.11 Transfer Rules

lemma *plus_multiset_transfer*[*transfer_rule*]:
$(rel\_fun\ (rel\_mset\ R)\ (rel\_fun\ (rel\_mset\ R)\ (rel\_mset\ R)))\ (+)\ (+)$
⟨*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

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

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

lemma *list_of_mset_exi*: ∃ *l*. *m* = *mset l*
⟨*proof*⟩

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

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

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

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

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

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

assumes $LL \in set\ Ci$
shows $L \in\# sum\_list\ Ci$
⟨*proof*⟩

lemma *in_mset_sum_list2*:
assumes $L \in\# sum\_list\ Ci$
obtains LL **where**
$LL \in set\ Ci$
$L \in\# LL$
⟨*proof*⟩

lemma *in_mset_sum_list_iff*: $a \in\# sum\_list\ \mathcal{A} \longleftrightarrow (\exists A \in set\ \mathcal{A}.\ a \in\# A)$
⟨*proof*⟩

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

2.12.3 More on Multisets and Functions

lemma *subseteq_mset_size_eql*: $X \subseteq\# Y \Longrightarrow size\ Y = size\ X \Longrightarrow X = Y$
⟨*proof*⟩

lemma *image_mset_of_subset_list*:
assumes $image\_mset\ \eta\ C' = mset\ lC$
shows $\exists qC'.\ map\ \eta\ qC' = lC \wedge mset\ qC' = C'$
⟨*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*⟩

2.12.4 More on Multiset Order

lemma *less_multiset_doubletons*:
assumes

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

end

3 Signed (Finite) Multisets

theory *Signed_ Multiset*
imports *Multiset_ More*
abbrevs
$!z = {}_z$
begin

3.1 Definition of Signed Multisets

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

quotient-type *'a zmultiset* = *'a multiset* × *'a multiset* / *equiv_ zmset*
⟨*proof*⟩

3.2 Basic Operations on Signed Multisets

instantiation *zmultiset* :: (*type*) *cancel_ comm_ monoid_ add*
begin

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

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

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

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

instance
⟨*proof*⟩

end

instantiation *zmultiset* :: (*type*) *group_ add*
begin

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

instance
⟨*proof*⟩

end

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

lemma *zcount_ inject*: *zcount M* = *zcount N* ⟷ *M* = *N*

⟨*proof*⟩

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

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

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

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

syntax
_zmultiset :: *args* ⇒ $'a\ zmultiset$ $(\{\#(\_)\#\}_z)$
translations
$\{\#x,\ xs\#\}_z == CONST\ add\_zmset\ x\ \{\#xs\#\}_z$
$\{\#x\#\}_z == CONST\ add\_zmset\ x\ \{\#\}_z$

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

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

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

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

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

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

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

3.2.1 Conversion to Set and Membership

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

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

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

notation (*ASCII*)
$elem\_zmset\ ('(:\#z'))$ **and**
$elem\_zmset\ ((\_/\ :\#z\ \_)\ [51,\ 51]\ 50)$

abbreviation *not_elem_zmset* :: $'a \Rightarrow 'a\ zmultiset \Rightarrow bool$ **where**

not_elem_zmset a M ≡ *a* ∉ *set_zmset M*

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

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

context
begin

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

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

end

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

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

translations
∀ *x*∈#$_z$*A*. *P* ⇌ *CONST Signed_Multiset.Ball A* (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 $A \neq \{\#\}_z$
obtains x **where** $x \in\#_z A$
⟨*proof*⟩

3.2.2 Union

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

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

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

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

3.2.3 Difference

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

lemma *add_zmset_diff_bothsides*: ‹$add\_zmset\ a\ M - add\_zmset\ a\ A = M - A$›
⟨*proof*⟩

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

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

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

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

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

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

lemma *zmset_add*[*elim?*]:
obtains B **where** $A = add\_zmset\ a\ B$
⟨*proof*⟩

3.2.4 Equality of Signed Multisets

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

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

lemma *add_zmset_remove_trivial*: ‹$add\_zmset\ x\ M - \{\#x\#\}_z = M$›
⟨*proof*⟩

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

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

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

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

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

3.3 Conversions from and to Multisets

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma
mset_pos_neg_partition: $M = zmset\_of\ (mset\_pos\ M) - zmset\_of\ (mset\_neg\ M)$ **and**
mset_pos_as_neg: $zmset\_of\ (mset\_pos\ M) = zmset\_of\ (mset\_neg\ M) + M$ **and**
mset_neg_as_pos: $zmset\_of\ (mset\_neg\ M) = zmset\_of\ (mset\_pos\ M) - M$
⟨*proof*⟩

lemma *mset_pos_uminus*[*simp*]: $mset\_pos\ (-\ A) = mset\_neg\ A$
⟨*proof*⟩

lemma *mset_neg_uminus*[*simp*]: $mset\_neg\ (-\ A) = mset\_pos\ A$
⟨*proof*⟩

lemma *mset_pos_plus*[*simp*]:
$mset\_pos\ (A + B) = (mset\_pos\ A - mset\_neg\ B) + (mset\_pos\ B - mset\_neg\ A)$
⟨*proof*⟩

lemma *mset_neg_plus*[*simp*]:
$mset\_neg\ (A + B) = (mset\_neg\ A - mset\_pos\ B) + (mset\_neg\ B - mset\_pos\ A)$
⟨*proof*⟩

lemma *mset_pos_diff*[*simp*]:
$mset\_pos\ (A - B) = (mset\_pos\ A - mset\_pos\ B) + (mset\_neg\ B - mset\_neg\ A)$
⟨*proof*⟩

lemma *mset_neg_diff*[*simp*]:
$mset\_neg\ (A - B) = (mset\_neg\ A - mset\_neg\ B) + (mset\_pos\ B - mset\_pos\ A)$
⟨*proof*⟩

lemma *mset_pos_neg_dual*:
$mset\_pos\ a + mset\_pos\ b + (mset\_neg\ a - mset\_pos\ b) + (mset\_neg\ b - mset\_pos\ a) =$
$mset\_neg\ a + mset\_neg\ b + (mset\_pos\ a - mset\_neg\ b) + (mset\_pos\ b - mset\_neg\ a)$
⟨*proof*⟩

lemma *decompose_zmset_of2*:
obtains $A\ B\ C$ **where**
$M = zmset\_of\ A + C$ **and**
$N = zmset\_of\ B + C$
⟨*proof*⟩

3.3.1 Pointwise Ordering Induced by *zcount*

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

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

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

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

notation (*input*)
subseteq_zmset (**infix** $\subseteq\#_z$ 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*: $\langle add\_zmset\ a\ A \subseteq\#_z add\_zmset\ a\ B \longleftrightarrow A \subseteq\#_z B\rangle$
⟨*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*)
_ ZMCollect :: *pttrn* ⇒ *'a zmultiset* ⇒ *bool* ⇒ *'a zmultiset* ((*1*{#_ :#z _./ _#}))
syntax

ZMCollect :: pttrn ⇒ 'a zmultiset ⇒ bool ⇒ 'a zmultiset ((1{# ∈#z _./ _#}))
translations
{#x ∈#z M. P#} == CONST filter_zmset (λx. 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 ∈#z M. Q x ∧ P x#}
⟨*proof*⟩

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

3.5 Uncategorized

lemma *multi_drop_mem_not_eq_zmset*: *B* − {#c#}z ≠ *B*
⟨*proof*⟩

lemma *zmultiset_partition*: *M* = {#x ∈#z M. P x #} + {#x ∈#z M. ¬ P x#}
⟨*proof*⟩

3.6 Image

definition *image_zmset* :: ('a ⇒ 'b) ⇒ 'a zmultiset ⇒ 'b zmultiset **where**
image_zmset f M =
zmset_of (*fold_mset* (*add_mset* ∘ *f*) {#} (*mset_pos M*)) −
zmset_of (*fold_mset* (*add_mset* ∘ *f*) {#} (*mset_neg M*))

3.7 Multiset Order

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

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

definition *less_eq_zmultiset* :: 'a zmultiset ⇒ 'a zmultiset ⇒ *bool* **where**
less_eq_zmultiset M' M ⟷ *M'* < *M* ∨ *M'* = *M*

instance
⟨*proof*⟩

end

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

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

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

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

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

lemma *not_lt_iff_ge_zmset*: ¬ *x* < *y* ⟷ *x* ≥ *y* **for** *x y* :: *'a zmultiset*
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *zmset_of_less*: *zmset_of M* < *zmset_of N* ⟷ *M* < *N*
⟨*proof*⟩

lemma *zmset_of_le*: *zmset_of M* ≤ *zmset_of N* ⟷ *M* ≤ *N*
⟨*proof*⟩

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

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

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

lemma *add_zmset_eq_add_NO_MATCH*[*cancelation_simproc_pre*]:
‹*NO_MATCH* {#}$_z$ *H* ⟹ *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* ≤ *i* ⟹ (*repeat_zmset i u* + *m* ⊆#$_z$ *repeat_zmset j u* + *n*) = (*repeat_zmset* (*i* − *j*) *u* + *m* ⊆#$_z$ *n*)›
⟨*proof*⟩

lemma *zmset_subseteq_add_iff2*:
‹*i* ≤ *j* ⟹ (*repeat_zmset i u* + *m* ⊆#$_z$ *repeat_zmset j u* + *n*) = (*m* ⊆#$_z$ *repeat_zmset* (*j* − *i*) *u* + *n*)›

⟨*proof*⟩

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

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

⟨*ML*⟩

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

⟨*ML*⟩

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

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

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

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

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

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

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

end

4 Nested Multisets

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

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

4.1 Type Definition

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

inductive *no_elem* :: *'a nmultiset* ⇒ *bool* **where**
(⋀*X*. *X* ∈# *M* ⟹ *no_elem X*) ⟹ *no_elem* (*MSet M*)

inductive-set *sub_nmset* :: (*'a nmultiset* × *'a nmultiset*) *set* **where**
X ∈# *M* ⟹ (*X*, *MSet M*) ∈ *sub_nmset*

lemma *wf_sub_nmset*[*simp*]: *wf sub_nmset*
⟨*proof*⟩

primrec *depth_nmset* :: *'a nmultiset* ⇒ *nat* (|_|) **where**
|*Elem a*| = *0*
| |*MSet M*| = (*let X* = *set_mset* (*image_mset depth_nmset M*) *in if X* = {} *then 0 else Suc* (*Max X*))

lemma *depth_nmset_MSet*: *x* ∈# *M* ⟹ |*x*| < |*MSet M*|
⟨*proof*⟩

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

4.2 Dershowitz and Manna's Nested Multiset Order

The Dershowitz–Manna extension:

definition *less_multiset_ext*$_{DM}$ :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a multiset* ⇒ *'a multiset* ⇒ *bool* **where**
less_multiset_ext$_{DM}$ *R M N* ⟷
(∃ *X Y*. *X* ≠ {#} ∧ *X* ⊆# *N* ∧ *M* = (*N* − *X*) + *Y* ∧ (∀ *k*. *k* ∈# *Y* ⟶ (∃ *a*. *a* ∈# *X* ∧ *R k a*)))

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

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

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

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

lemma *less_multiset_ext*$_{DM}$*_cong*[*fundef_cong*]:
(⋀*X Y k a*. *X* ≠ {#} ⟹ *X* ⊆# *N* ⟹ *M* = (*N* − *X*) + *Y* ⟹ *k* ∈# *Y* ⟹ *R k a* = *S k a*) ⟹
less_multiset_ext$_{DM}$ *R M N* = *less_multiset_ext*$_{DM}$ *S M N*
⟨*proof*⟩

function *less_nmultiset* :: *'a nmultiset* ⇒ *'a nmultiset* ⇒ *bool* **where**
less_nmultiset (*Elem a*) (*Elem b*) ⟷ *a* < *b*
| *less_nmultiset* (*Elem a*) (*MSet M*) ⟷ *True*
| *less_nmultiset* (*MSet M*) (*Elem a*) ⟷ *False*
| *less_nmultiset* (*MSet M*) (*MSet N*) ⟷ *less_multiset_ext*$_{DM}$ *less_nmultiset M N*
⟨*proof*⟩
termination
⟨*proof*⟩

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

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

lemma *trans_less_nmultiset*: $X < Y \Longrightarrow Y < Z \Longrightarrow X < Z$ **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: $n \leq m + n$ **for** n :: *hmultiset*
⟨*proof*⟩

lemma *le_ zero_ eq_ hmset*[*simp*]: $M \leq 0 \longleftrightarrow M = 0$ **for** M :: *hmultiset*
⟨*proof*⟩

lemma *not_ less_ zero_ hmset*[*simp*]: $\neg\ M < 0$ **for** M :: *hmultiset*
⟨*proof*⟩

lemma *not_ gr_ zero_ hmset*[*simp*]: $\neg\ 0 < M \longleftrightarrow M = 0$ **for** M :: *hmultiset*
⟨*proof*⟩

lemma *zero_ less_ iff_ neq_ zero_ hmset*: $0 < M \longleftrightarrow M \neq 0$ **for** M :: *hmultiset*
⟨*proof*⟩

lemma *zero_ less_ HMSet_ iff*[*simp*]: $0 < HMSet\ M \longleftrightarrow M \neq \{\#\}$
⟨*proof*⟩

lemma *gr_ zeroI_ hmset*: $(M = 0 \Longrightarrow False) \Longrightarrow 0 < M$ **for** M :: *hmultiset*
⟨*proof*⟩

lemma *gr_ implies_ not_ zero_ hmset*: $M < N \Longrightarrow N \neq 0$ **for** $M\ N$ :: *hmultiset*
⟨*proof*⟩

lemma *add_ eq_ 0_ iff_ both_ eq_ 0_ hmset*[*simp*]: $M + N = 0 \longleftrightarrow M = 0 \wedge N = 0$ **for** $M\ N$ :: *hmultiset*
⟨*proof*⟩

lemma *trans_ less_ add1_ hmset*: $i < j \Longrightarrow i < j + m$ **for** $i\ j\ m$ :: *hmultiset*
⟨*proof*⟩

lemma *trans_ less_ add2_ hmset*: $i < j \Longrightarrow i < m + j$ **for** $i\ j\ m$ :: *hmultiset*
⟨*proof*⟩

lemma *trans_ le_ add1_ hmset*: $i \leq j \Longrightarrow i \leq j + m$ **for** $i\ j\ m$ :: *hmultiset*
⟨*proof*⟩

lemma *trans_ le_ add2_ hmset*: $i \leq j \Longrightarrow i \leq m + j$ **for** $i\ j\ m$ :: *hmultiset*
⟨*proof*⟩

lemma *diff_ le_ self_ hmset*: $m - n \leq m$ **for** $m\ n$ :: *hmultiset*
⟨*proof*⟩

end

6 Signed Hereditar(il)y (Finite) Multisets

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

6.1 Type Definition

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

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

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

setup-lifting *type_ definition_ zhmultiset*

6.2 Multiset Order

instantiation *zhmultiset* :: *linorder*
begin

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

instance
⟨*proof*⟩

end

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

6.3 Embedding and Projections of Syntactic Ordinals

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

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

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

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

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

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

6.4 Disjoint Union and Difference

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

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

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

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

lemmas *ZHMSet_ plus* = *plus_ zhmultiset.abs_ eq*[*symmetric*]
lemmas *ZHMSet_ diff* = *minus_ zhmultiset.abs_ eq*[*symmetric*]
lemmas *zhmsetmset_ plus* = *plus_ zhmultiset.rep_ eq*
lemmas *zhmsetmset_ diff* = *minus_ zhmultiset.rep_ eq*

lemma *zhmset_ of_ plus*: *zhmset_ of* (*A* + *B*) = *zhmset_ of A* + *zhmset_ of B*
⟨*proof*⟩

lemma *hmsetmset_ 0*[*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 ≤ N*
obtains *A B C* **where** *M = zhmset_of A + C* **and** *N = zhmset_of B + C* **and** *A ≤ B*
⟨*proof*⟩

instantiation *zhmultiset* :: *ab_group_add*
begin

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

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

instance
⟨*proof*⟩

end

6.5 Infimum and Supremum

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

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

instantiation *zhmultiset* :: *distrib_lattice*
begin

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

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

instance
⟨*proof*⟩

end

end

7 Syntactic Ordinals in Cantor Normal Form

theory *Syntactic_Ordinal*
imports *Hereditary_Multiset HOL−Library.Product_Order HOL−Library.Extended_Nat*
begin

7.1 Natural (Hessenberg) Product

instantiation *hmultiset* :: *comm_semiring_1*
begin

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

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

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

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

lemma *hmsetmset_times*:
hmsetmset (*m* ∗ *n*) = *image_mset* (*case_prod* (+)) (*hmsetmset m* ×# *hmsetmset n*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

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

lemma *singleton_times_left_hmset*[*simp*]: ω^*M* ∗ *N* = *HMSet* (*image_mset* ((+) *M*) (*hmsetmset N*))
⟨*proof*⟩

lemma *singleton_times_right_hmset*[*simp*]: *N* ∗ ω^*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* $n \longleftrightarrow m = n$
⟨*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 $n \leq m$
shows $k + of\_nat\ m - of\_nat\ n = k + of\_nat\ (m - n)$
⟨*proof*⟩

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

lemma *of_nat_ne_ω*[*simp*]: $of\_nat\ n \neq \omega$
⟨*proof*⟩

lemma *of_nat_less_hmset*[*simp*]: $(of\_nat\ M :: hmultiset) < of\_nat\ N \longleftrightarrow M < N$
⟨*proof*⟩

lemma *of_nat_le_hmset*[*simp*]: $(of\_nat\ M :: hmultiset) \leq of\_nat\ N \longleftrightarrow M \leq N$
⟨*proof*⟩

lemma *of_nat_times_ω_exp*: $of\_nat\ n * \omega\hat{}m = HMSet\ (replicate\_mset\ n\ m)$
⟨*proof*⟩

lemma *ω_exp_times_of_nat*: $\omega\hat{}m * of\_nat\ n = HMSet\ (replicate\_mset\ n\ m)$
⟨*proof*⟩

7.4 Embedding of Extended Natural Numbers

primrec *hmset_of_enat* :: $enat \Rightarrow hmultiset$ **where**
$hmset\_of\_enat\ (enat\ n) = of\_nat\ n$
| $hmset\_of\_enat\ \infty = \omega$

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*]: $hmset\_of\_enat\ n \leq \omega$
⟨*proof*⟩

lemma *hmset_of_enat_eq_ω_iff*[*simp*]: $hmset\_of\_enat\ n = \omega \longleftrightarrow n = \infty$
⟨*proof*⟩

7.5 Head Omega

definition *head_ω* :: $hmultiset \Rightarrow hmultiset$ **where**
$head\_\omega\ M = (if\ M = 0\ then\ 0\ else\ \omega\hat{}(Max\ (set\_mset\ (hmsetmset\ M))))$

lemma *head_ω_subseteq*: $hmsetmset\ (head\_\omega\ M) \subseteq\# hmsetmset\ M$
⟨*proof*⟩

lemma *head_ω_eq_0_iff*[*simp*]: $head\_\omega\ m = 0 \longleftrightarrow m = 0$
⟨*proof*⟩

lemma *head_ω_0*[*simp*]: $head\_\omega\ 0 = 0$
⟨*proof*⟩

lemma *head_ω_1*[*simp*]: $head\_\omega\ 1 = 1$
⟨*proof*⟩

lemma *head_ω_ of_ nat*[*simp*]: *head_ω* (*of_ nat n*) = (*if n* = *0 then 0 else 1*)
⟨*proof*⟩

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

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

lemma *le_ imp_ head_ω_ le*:
assumes *m_ le_ n*: *m* ≤ *n*
shows *head_ω m* ≤ *head_ω n*
⟨*proof*⟩

lemma *head_ω_ lt_ imp_ lt*: *head_ω m* < *head_ω n* ⟹ *m* < *n*
⟨*proof*⟩

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

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

7.6 More Inequalities and Some Equalities

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

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

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

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

lemma *of_ nat_ le_ω*[*simp*]: *of_ nat n* ≤ *ω*
⟨*proof*⟩

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

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

lemma *not_ω_ lt_ of_ nat*[*simp*]: ¬ *ω* < *of_ nat n*
⟨*proof*⟩

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

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

lemma *not_ω_ le_ of_ nat*[*simp*]: ¬ *ω* ≤ *of_ nat n*
⟨*proof*⟩

lemma *not_ω_ le_ numeral*[*simp*]: ¬ *ω* ≤ *numeral n*
⟨*proof*⟩

lemma *zero_ ne_ω*[*simp*]: *0* ≠ *ω*
⟨*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**
$\omega\_ne\_of\_nat$[*simp*]: $\omega \neq of\_nat\ m$ **and**
$\omega\_ne\_numeral$[*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* :: $hmultiset \Rightarrow nat$ **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* :: $hmultiset \Rightarrow nat$ **where**

sum_ coefs M = size (hmsetmset M)

lemma *sum_ coefs_ distrib_ plus*[*simp*]: *sum_ coefs (M + N) = sum_ coefs M + sum_ coefs N*
⟨*proof*⟩

lemma *sum_ coefs_ gt_ 0*: *sum_ coefs M > 0 ⟷ M > 0*
⟨*proof*⟩

7.8 An Example

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

lemma *ludwig_ waldmann_ less*:
fixes *α1 α2 β1 β2 γ δ* :: *hmultiset*
assumes
αβ2γ_ lt_ αβ1γ: *α2 + β2 * γ < α1 + β1 * γ* **and**
β2_ le_ β1: *β2 ≤ β1* **and**
γ_ lt_ δ: *γ < δ*
shows *α2 + β2 * δ < α1 + β1 * δ*
⟨*proof*⟩

end

8 Signed Syntactic Ordinals in Cantor Normal Form

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

8.1 Natural (Hessenberg) Product

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

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

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

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

lemma *ω_z_ as_ ω*: *ω_z = zhmset_ of ω*
⟨*proof*⟩

lift-definition *times_ zhmultiset* :: *zhmultiset ⇒ zhmultiset ⇒ zhmultiset* **is**
λM N.
*zmset_ of (hmsetmset (HMSet (mset_ pos M) * HMSet (mset_ pos N)))*
*− zmset_ of (hmsetmset (HMSet (mset_ pos M) * HMSet (mset_ neg N)))*
*+ zmset_ of (hmsetmset (HMSet (mset_ neg M) * HMSet (mset_ neg N)))*
*− zmset_ of (hmsetmset (HMSet (mset_ neg M) * HMSet (mset_ pos N)))* ⟨*proof*⟩

lemmas *zhmsetmset_ times = times_ zhmultiset.rep_ eq*

instance
⟨*proof*⟩

end

lemma *zhmset_ of_ 1*: *zhmset_ of 1 = 1*
⟨*proof*⟩

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

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

8.2 Embedding of Natural Numbers

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

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

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

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

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

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

8.3 Embedding of Extended Natural Numbers

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

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

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

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

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

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

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

8.4 Inequalities and Some (Dis)equalities

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

instantiation *zhmultiset* :: *linordered_idom*
begin

definition *sgn_zhmultiset* :: *zhmultiset* ⇒ *zhmultiset* **where**

sgn_zhmultiset M = (if M = 0 then 0 else if M > 0 then 1 else −1)

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

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

instance
⟨*proof*⟩

end

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *not_ω_z_lt_1*[*simp*]: $\neg\ \omega_z < 1$
⟨*proof*⟩

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

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

lemma *not_ω_z_le_1*[*simp*]: $\neg\ \omega_z \leq 1$
⟨*proof*⟩

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

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

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

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

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

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

lemma
zhmset_of_enat_inject[*simp*]: *zhmset_of_enat* m = *zhmset_of_enat* $n \longleftrightarrow m = n$ **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* ⟹ *ordinal_ of_ hmset k* < *ordinal_ of_ hmset l*
⟨*proof*⟩

lemma *ordinal_ of_ hmset_ less*[*simp*]: *ordinal_ of_ hmset k* < *ordinal_ of_ hmset l* ⟷ *k* < *l*
⟨*proof*⟩

end

10 Termination of McCarthy's 91 Function

theory *McCarthy_ 91*
imports *HOL−Library.Multiset_ Order*
begin

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

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

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

definition τ :: *nat* ⇒ *int* ⇒ *int multiset* **where**

$\tau\ n\ z = mset\ (map\ (\lambda i.\ (f\ \hat{}\hat{}\ nat\ i)\ z)\ [0..int\ n - 1])$

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

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

end

11 Termination of the Hydra Battle

theory *Hydra_Battle*
imports *Syntactic_Ordinal*
begin

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

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

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

primrec *encode* :: *lisp* ⇒ *hmultiset* **where**
encode Nil = *0*
| *encode* (*Cons l r*) = $\omega\ \hat{}$(*encode l*) + *encode r*

primrec $f :: nat \Rightarrow lisp \Rightarrow lisp \Rightarrow lisp$ **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* ∗ $\omega\ \hat{}$(*encode y*) + *encode x*
⟨*proof*⟩

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

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

end

12 Termination of the Goodstein Sequence

theory *Goodstein_Sequence*
imports *Multiset_More Syntactic_Ordinal*
begin

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

12.1 Lemmas about Division

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

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

12.2 Hereditary and Nonhereditary Base-n Systems

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

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

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

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

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

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

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

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

12.3 Encoding of Natural Numbers into Ordinals

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

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

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

lemma *encode_Suc*:
*encode e (Suc n) = of_nat (Suc n mod base) * ω^(encode 0 e) + encode (e + 1) (Suc n div base)*
⟨*proof*⟩

lemma *encode_0_iff*: *encode e n = 0 ⟷ n = 0*
⟨*proof*⟩

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

lemma *encode_exp_0*: *encode e n = encode 0 (base ^ e * n)*
⟨*proof*⟩

lemma *mem_hmsetmset_encodeD*: *M ∈# hmsetmset (encode e n) ⟹ ∃ e' ≥ e. M = encode 0 e'*
⟨*proof*⟩

lemma *less_imp_encode_less*: *n < p ⟹ encode e n < encode e p*
⟨*proof*⟩

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

lemma *aligned$_e$_encode*: *aligned$_e$ e (encode e M)*
⟨*proof*⟩

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

12.4 Decoding of Natural Numbers from Ordinals

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

lemma *decode_unfold*: *decode e M = (∑ m ∈# hmsetmset M. base ^ decode 0 m) div base ^ e*
⟨*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*: $inj\_on\ (decode\ e)\ \{M.\ well\_base_h\ M \wedge aligned_d\ e\ M\}$
  ⟨*proof*⟩

lemma *decode_0_iff*: $well\_base_h\ M \Longrightarrow aligned_d\ e\ M \Longrightarrow decode\ e\ M = 0 \longleftrightarrow M = 0$
  ⟨*proof*⟩

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

lemma *encode_decode_exp_0*: $well\_base_h\ M \Longrightarrow encode\ 0\ (decode\ 0\ M) = M$
  ⟨*proof*⟩

end

lemma *well_base_h_mono_base*:
  assumes
    $well_h$: $well\_base_h\ base\ M$ **and**
    two: $2 \leq base$ **and**
    bases: $base \leq base'$
  shows $well\_base_h\ base'\ M$
  ⟨*proof*⟩

12.5 The Goodstein Sequence and Goodstein's Theorem

context
  fixes *start* :: *nat*
begin

primrec *goodstein* :: $nat \Rightarrow nat$ **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*: $\exists\, i.\ goodstein\ i = 0$
⟨*proof*⟩

end

end

13 Towards Decidability of Behavioral Equivalence for Unary PCF

theory *Unary_PCF*
  imports
    HOL−Library.FSet

HOL−Library.Countable_Set_Type
HOL−Library.Nat_Bijection
Hereditary_Multiset
List−Index.List_Index
begin

13.1 Preliminaries

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

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

13.2 Types

datatype $type = \mathcal{B}\ (\mathcal{B}) \mid Fun\ type\ type\ (\textbf{infixr}\ \rightarrow\ 65)$

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

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

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

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

lemma *dest_fun_mk_fun*[*simp*]: $dest\_fun\ (Ts \rightarrow\!\!\rightarrow T) = Ts\ @\ dest\_fun\ T$
⟨*proof*⟩

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

lemma δ*_mk_fun*: $\delta\ (Ts \rightarrow\!\!\rightarrow T) = HMSet\ (hmsetmset\ (\delta\ T) + mset\ (map\ \delta\ Ts))$
⟨*proof*⟩

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

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*: $i \neq j \Longrightarrow xT \neq yU \Longrightarrow$ *fresh* $yU\ t \Longrightarrow$
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* $t \Longrightarrow$ *open_ Var* $i\ xT$ (*close_ Var* $i\ xT\ t$) = t
⟨*proof*⟩

end

lemma *close_ Var_ inj*: *lc* $t \Longrightarrow$ *lc* $u \Longrightarrow$ *close_ Var* $i\ xT\ t$ = *close_ Var* $i\ xT\ u \Longrightarrow t = u$
⟨*proof*⟩

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

lemma *Apps_ snoc*: $f \cdot (xs\ @\ [x]) = f \cdot xs \cdot x$
⟨*proof*⟩

lemma *Apps_ append*: $f \cdot (xs\ @\ ys) = f \cdot\ xs \cdot\ ys$
⟨*proof*⟩

lemma *Apps_ inj*[*simp*]: $f \cdot ts = g \cdot ts \longleftrightarrow f = g$
⟨*proof*⟩

lemma *eq_ Apps_ conv*[*simp*]:
fixes i :: *idx* **and** b :: *bool* **and** f :: *expr* **and** ts :: *expr list*
shows
$(\langle m\rangle = f \cdot ts) = (\langle m\rangle = f \wedge ts = [])$
$(f \cdot ts = \langle m\rangle) = (\langle m\rangle = f \wedge ts = [])$
$(i = f \cdot ts) = (i = f \wedge ts = [])$
$(f \cdot ts = i) = (i = f \wedge ts = [])$
$(b = f \cdot ts) = (b = f \wedge ts = [])$
$(f \cdot ts = b) = (b = f \wedge ts = [])$
$(e1\ ?\ e2 = f \cdot ts) = (e1\ ?\ e2 = f \wedge ts = [])$
$(f \cdot ts = e1\ ?\ e2) = (e1\ ?\ e2 = f \wedge ts = [])$
$(\Lambda\langle T\rangle\ t = f \cdot ts) = (\Lambda\langle T\rangle\ t = f \wedge ts = [])$
$(f \cdot ts = \Lambda\langle T\rangle\ t) = (\Lambda\langle T\rangle\ t = f \wedge ts = [])$
⟨*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 (→) (*rev Ts*) *T* = $\mathcal{B}$ ⟷ *Ts* = [] ∧ *T* = $\mathcal{B}$
$\mathcal{B}$ = *fold* (→) (*rev Ts*) *T* ⟷ *Ts* = [] ∧ *T* = $\mathcal{B}$
⟨*proof*⟩

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

13.4 Substitution

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

lemma *fv_subst*:
fv (*subst xT t u*) = *fv u* |−| {|*xT*|} |∪| (*if xT* |∈| *fv u then fv t else* {||})
⟨*proof*⟩

lemma *subst_fresh*: *fresh xT u* ⟹ *subst xT t u* = *u*
⟨*proof*⟩

context begin

private lemma *open_open_id*: *i* ≠ *j* ⟹ *open i t* (*open j t*′ *u*) = *open j t*′ *u* ⟹ *open i t u* = *u*
⟨*proof*⟩

lemma *lc_open_id*: *lc u* ⟹ *open k t u* = *u*
⟨*proof*⟩

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

lemma *subst_open_Var*:
xT ≠ *yU* ⟹ *lc u* ⟹ *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* · *xs*) = *subst xT u f* · *map* (*subst xT u*) *xs*
⟨*proof*⟩

end

context begin

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

lemma *subst_close_Var*:
xT ≠ *yU* ⟹ *fresh yU u* ⟹ *subst xT u* (*close_Var i yU t*) = *close_Var i yU* (*subst xT u t*)
⟨*proof*⟩

end

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

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

⟨*proof*⟩

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

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

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

13.5 Typing

inductive *welltyped* :: *expr* ⇒ *type* ⇒ *bool* (**infix** ::: *60*) **where**
welltyped_Var[*intro*!]: ⟨(*x*, *T*)⟩ ::: *T*
| *welltyped_B*[*intro*!]: (*b* :: *bool*) ::: $\mathcal{B}$
| *welltyped_Seq*[*intro*!]: *e1* ::: $\mathcal{B}$ ⟹ *e2* ::: $\mathcal{B}$ ⟹ *e1 ? e2* ::: $\mathcal{B}$
| *welltyped_App*[*intro*]: *e1* ::: *T* → *U* ⟹ *e2* ::: *T* ⟹ *e1* · *e2* ::: *U*
| *welltyped_Abs*[*intro*]: (∀ *x*. (*x*, *T*) |∉| *X* ⟶ *open0_Var* (*x*, *T*) *e* ::: *U*) ⟹ Λ⟨*T*⟩ *e* ::: *T* → *U*

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

lemma *welltyped_unique*: *t* ::: *T* ⟹ *t* ::: *U* ⟹ *T* = *U*
⟨*proof*⟩

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

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

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

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

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

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

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

primrec *constant0* **where**
constant0 $\mathcal{B}$ = *Var* (''*bool*'', $\mathcal{B}$)
| *constant0* (*T* → *U*) = Λ⟨*T*⟩ (*constant0 U*)

definition *constant T* = Λ⟨$\mathcal{B}$⟩ (*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 ≡ *take i xs* @ *drop* (*Suc i*) *xs*

definition *nth_arg* (**infixl** !− *100*) **where**
nth_arg T i ≡ *nth* (*dest_fun T*) *i*

abbreviation *ar* **where**
ar T ≡ *length* (*dest_fun T*)

lemma *size_nth_arg*[*simp*]: *i* < *ar T* ⟹ *size* (*T* !− *i*) < *size T*
⟨*proof*⟩

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

theorem π_*induct*[*rotated* −*2*, *consumes 2*, *case_names 0 Suc*]:
assumes ⋀*T i*. *i* < *ar T* ⟹ *P T i 0*
and ⋀*T i j*. *i* < *ar T* ⟹ *j* < *ar* (*T* !− *i*) ⟹ *P* (*T* !− *i*) *j 0* ⟹
(∀ *x* < *ar* (*T* !− *i* !− *j*). *P* (*T* !− *i*) *j* (*x* + *1*)) ⟹ *P T i* (*j* + *1*)
shows *i* < *ar T* ⟹ *j* ≤ *ar* (*T* !− *i*) ⟹ *P T i j*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma welltyped_Seq_iff[simp]: e1 ? e2 ::: T ⟷ (T = ℬ ∧ e1 ::: ℬ ∧ e2 ::: ℬ)
  ⟨proof⟩

lemma welltyped_Seqs_iff[simp]: es ?? e ::: T ⟷
```

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

lemma *welltyped_App_iff*[*simp*]: $f \cdot t ::: U \longleftrightarrow (\exists T.\ f ::: T \to U \wedge t ::: T)$
⟨*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*: $i < ar\ T \Longrightarrow dest\_fun\ (\pi\ T\ i\ 0) = nth\_drop\ i\ (dest\_fun\ T)$
⟨*proof*⟩

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

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

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

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

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

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

context notes $\pi.simps$[*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