

# Formalization of Nested Multisets, Hereditary Multisets, and Syntactic Ordinals

Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel

February 23, 2021

## 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

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>More about Multisets</b>	<b>3</b>
2.1	Basic Setup	3
2.2	Lemmas about Intersection, Union and Pointwise Inclusion	3
2.3	Lemmas about Filter and Image	4
2.4	Lemmas about Sum	6
2.5	Lemmas about Remove	6
2.6	Lemmas about Replicate	8
2.7	Multiset and Set Conversions	9
2.8	Duplicate Removal	9
2.9	Repeat Operation	11
2.10	Cartesian Product	11
2.11	Transfer Rules	15
2.12	Even More about Multisets	16
2.12.1	Multisets and Functions	16
2.12.2	Multisets and Lists	16
2.12.3	More on Multisets and Functions	17
2.12.4	More on Multiset Order	18
<b>3</b>	<b>Signed (Finite) Multisets</b>	<b>19</b>
3.1	Definition of Signed Multisets	19
3.2	Basic Operations on Signed Multisets	19
3.2.1	Conversion to Set and Membership	20
3.2.2	Union	22
3.2.3	Difference	22
3.2.4	Equality of Signed Multisets	23
3.3	Conversions from and to Multisets	23
3.3.1	Pointwise Ordering Induced by <i>zcount</i>	25
3.3.2	Subset is an Order	26
3.4	Replicate and Repeat Operations	26
3.4.1	Filter (with Comprehension Syntax)	27
3.5	Uncategorized	28
3.6	Image	28
3.7	Multiset Order	28

<b>4</b>	<b>Nested Multisets</b>	<b>32</b>
4.1	Type Definition . . . . .	32
4.2	Dershowitz and Manna’s Nested Multiset Order . . . . .	33
<b>5</b>	<b>Hereditar(il)y (Finite) Multisets</b>	<b>38</b>
5.1	Type Definition . . . . .	39
5.2	Restriction of Dershowitz and Manna’s Nested Multiset Order . . . . .	39
5.3	Disjoint Union and Truncated Difference . . . . .	40
5.4	Infimum and Supremum . . . . .	42
5.5	Inequalities . . . . .	42
<b>6</b>	<b>Signed Hereditar(il)y (Finite) Multisets</b>	<b>43</b>
6.1	Type Definition . . . . .	43
6.2	Multiset Order . . . . .	44
6.3	Embedding and Projections of Syntactic Ordinals . . . . .	44
6.4	Disjoint Union and Difference . . . . .	44
6.5	Infimum and Supremum . . . . .	45
<b>7</b>	<b>Syntactic Ordinals in Cantor Normal Form</b>	<b>46</b>
7.1	Natural (Hessenberg) Product . . . . .	46
7.2	Inequalities . . . . .	47
7.3	Embedding of Natural Numbers . . . . .	50
7.4	Embedding of Extended Natural Numbers . . . . .	51
7.5	Head Omega . . . . .	51
7.6	More Inequalities and Some Equalities . . . . .	53
7.7	Conversions to Natural Numbers . . . . .	58
7.8	An Example . . . . .	58
<b>8</b>	<b>Signed Syntactic Ordinals in Cantor Normal Form</b>	<b>59</b>
8.1	Natural (Hessenberg) Product . . . . .	59
8.2	Embedding of Natural Numbers . . . . .	61
8.3	Embedding of Extended Natural Numbers . . . . .	61
8.4	Inequalities and Some (Dis)equalities . . . . .	61
8.5	An Example . . . . .	64
<b>9</b>	<b>Bridge between Huffman’s Ordinal Library and the Syntactic Ordinals</b>	<b>65</b>
9.1	Missing Lemmas about Huffman’s Ordinals . . . . .	65
9.2	Embedding of Syntactic Ordinals into Huffman’s Ordinals . . . . .	65
<b>10</b>	<b>Termination of McCarthy’s 91 Function</b>	<b>69</b>
<b>11</b>	<b>Termination of the Hydra Battle</b>	<b>72</b>
<b>12</b>	<b>Termination of the Goodstein Sequence</b>	<b>73</b>
12.1	Lemmas about Division . . . . .	73
12.2	Hereditary and Nonhereditary Base- $n$ Systems . . . . .	73
12.3	Encoding of Natural Numbers into Ordinals . . . . .	74
12.4	Decoding of Natural Numbers from Ordinals . . . . .	78
12.5	The Goodstein Sequence and Goodstein’s Theorem . . . . .	81
<b>13</b>	<b>Towards Decidability of Behavioral Equivalence for Unary PCF</b>	<b>82</b>
13.1	Preliminaries . . . . .	82
13.2	Types . . . . .	82
13.3	Terms . . . . .	83
13.4	Substitution . . . . .	86
13.5	Typing . . . . .	87
13.6	Definition 10 and Lemma 11 from Schmidt-Schauß’s paper . . . . .	88

# 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*:  $A \subseteq\# B \implies A \subseteq\# \text{add\_mset } x \ B$   
**by** (*metis add\_mset\_diff\_bothsides diff\_subset\_eq\_self multiset\_inter\_def subset\_mset.inf.absorb2*)

**lemma** *subset\_add\_mset\_notin\_subset\_mset*:  $\langle A \subseteq\# \text{add\_mset } b \ B \implies b \notin\# A \implies A \subseteq\# B \rangle$   
**by** (*simp add: subset\_mset.le\_iff\_sup*)

**lemma** *subset\_msetE*:  $\llbracket A \subseteq\# B; \llbracket A \subseteq\# B; \neg B \subseteq\# A \rrbracket \implies R \rrbracket \implies R$   
**by** (*simp add: subset\_mset.less\_le\_not\_le*)

**lemma** *Diff\_triv\_mset*:  $M \cap\# N = \{\#\} \implies M - N = M$   
**by** (*metis diff\_intersect\_left\_idem diff\_zero*)

**lemma** *diff\_intersect\_sym\_diff*:  $(A - B) \cap\# (B - A) = \{\#\}$   
**unfolding** *multiset\_inter\_def*

```

proof -
  have  $A - (B - (B - A)) = A - B$ 
    by (metis diff_intersect_right_idem multiset_inter_def)
  then show  $A - B - (A - B - (B - A)) = \{\#\}$ 
    by (metis diff_add diff_cancel diff_subset_eq_self subset_mset.diff_add)
qed

```

```

declare subset_msetE [elim!]

```

```

lemma subseq_mset_subseq_mset:  $\text{subseq } xs \ ys \implies \text{mset } xs \subseteq\# \text{mset } ys$ 
proof (induct xs arbitrary: ys)
  case (Cons x xs)
    note Outer_Cons = this
    then show ?case
      proof (induct ys)
        case (Cons y ys)
          have  $\text{subseq } xs \ ys$ 
            by (metis Cons.prem(2) subseq_Cons' subseq_Cons2_iff)
          then show ?case
            using Cons by (metis mset.simps(2) mset_subset_eq_add_mset_cancel subseq_Cons2_iff
              subset_mset_imp_subset_add_mset)
        qed simp
      qed simp

```

## 2.3 Lemmas about Filter and Image

```

lemma count_image_mset_ge_count:  $\text{count } (\text{image\_mset } f \ A) \ (f \ b) \geq \text{count } A \ b$ 
  by (induction A) auto

```

```

lemma count_image_mset_inj:
  assumes  $\langle \text{inj } f \rangle$ 
  shows  $\langle \text{count } (\text{image\_mset } f \ M) \ (f \ x) = \text{count } M \ x \rangle$ 
  by (induct M) (use assms in <auto simp: inj_on_def>)

```

```

lemma count_image_mset_le_count_inj_on:
   $\text{inj\_on } f \ (\text{set\_mset } M) \implies \text{count } (\text{image\_mset } f \ M) \ y \leq \text{count } M \ (\text{inv\_into } (\text{set\_mset } M) \ f \ y)$ 

```

```

proof (induct M)
  case (add x M)
    note  $ih = \text{this}(1)$  and  $\text{inj\_xM} = \text{this}(2)$ 

    have  $\text{inj\_M}: \text{inj\_on } f \ (\text{set\_mset } M)$ 
      using  $\text{inj\_xM}$  by simp

    show ?case
      proof (cases x ∈# M)
        case  $x \ \text{in\_M}: \text{True}$ 
          show ?thesis
          proof (cases y = f x)
            case  $y \ \text{eq\_fx}: \text{True}$ 
              show ?thesis
              using  $x \ \text{in\_M}$   $ih[OF \ \text{inj\_M}]$  unfolding  $y \ \text{eq\_fx}$  by (simp add: inj_M insert_absorb)
            next
              case  $y \ \text{ne\_fx}: \text{False}$ 
                show ?thesis
                using  $x \ \text{in\_M}$   $ih[OF \ \text{inj\_M}]$   $y \ \text{ne\_fx}$  insert_absorb by fastforce
              qed
            next
              case  $x \ \text{ni\_M}: \text{False}$ 
                show ?thesis
                proof (cases y = f x)
                  case  $y \ \text{eq\_fx}: \text{True}$ 
                    have  $f \ x \notin\# \text{image\_mset } f \ M$ 
                      using  $x \ \text{ni\_M}$   $\text{inj\_xM}$  by force
                    thus ?thesis

```

**unfolding**  $y\_eq\_fx$   
**by** (*metis* (*no\_types*) *inj\_xM count\_add\_mset count\_greater\_eq\_Suc\_zero\_iff count\_inI image\_mset\_add\_mset inv\_into\_f\_f union\_single\_eq\_member*)  
**next**  
**case**  $y\_ne\_fx$ : *False*  
**show** *?thesis*  
**proof** (*rule ccontr*)  
**assume** *neg\_conj*:  $\neg$  *count (image\_mset f (add\_mset x M)) y*  
 $\leq$  *count (add\_mset x M) (inv\_into (set\_mset (add\_mset x M)) f y)*  
  
**have** *cnt\_y*: *count (add\_mset (f x) (image\_mset f M)) y = count (image\_mset f M) y*  
**using**  $y\_ne\_fx$  **by** *simp*  
  
**have**  $inv\_into (set\_mset M) f y \in\# add\_mset x M \implies$   
 $inv\_into (set\_mset (add\_mset x M)) f (f (inv\_into (set\_mset M) f y)) =$   
 $inv\_into (set\_mset M) f y$   
**by** (*meson inj\_xM inv\_into\_f\_f*)  
**hence**  $0 < count (image_mset f (add_mset x M)) y \implies$   
 $count M (inv\_into (set\_mset M) f y) = 0 \vee x = inv\_into (set\_mset M) f y$   
**using** *neg\_conj cnt\_y ih[OF inj\_M]*  
**by** (*metis* (*no\_types*) *count\_add\_mset count\_greater\_zero\_iff count\_inI f\_inv\_into\_f image\_mset\_add\_mset set\_image\_mset*)  
**thus** *False*  
**using** *neg\_conj cnt\_y x\_ni\_M ih[OF inj\_M]*  
**by** (*metis* (*no\_types*) *count\_greater\_zero\_iff count\_inI eq\_iff image\_mset\_add\_mset less\_imp\_le*)  
**qed**  
**qed**  
**qed**  
**qed** *simp*

**lemma** *mset\_filter\_compl*:  $mset (filter p xs) + mset (filter (Not \circ p) xs) = mset xs$   
**by** (*induction xs*) (*auto simp: ac\_simps*)

Near duplicate of *filter\_eq\_replicate\_mset*:  $\{\#y \in\# ?D. y = ?x\# \} = replicate\_mset (count ?D ?x) ?x$ .

**lemma** *filter\_mset\_eq*:  $filter\_mset ((=) L) A = replicate\_mset (count A L) L$   
**by** (*auto simp: multiset\_eq\_iff*)

**lemma** *filter\_mset\_cong[fundef\_cong]*:  
**assumes**  $M = M' \wedge a. a \in\# M \implies P a = Q a$   
**shows**  $filter\_mset P M = filter\_mset Q M$   
**proof** –  
**have**  $M - filter\_mset Q M = filter\_mset (\lambda a. \neg Q a) M$   
**by** (*metis multiset\_partition add\_diff\_cancel\_left'*)  
**then show** *?thesis*  
**by** (*auto simp: filter\_mset\_eq\_conv assms*)  
**qed**

**lemma** *image\_mset\_filter\_swap*:  $image\_mset f \{\# x \in\# M. P (f x)\# \} = \{\# x \in\# image\_mset f M. P x\# \}$   
**by** (*induction M*) *auto*

**lemma** *image\_mset\_cong2*:  
 $(\wedge x. x \in\# M \implies f x = g x) \implies M = N \implies image\_mset f M = image\_mset g N$   
**by** (*hypsubst, rule image\_mset\_cong*)

**lemma** *filter\_mset\_empty\_conv*:  $\langle filter\_mset P M = \{\#\# \} \rangle = (\forall L \in\# M. \neg P L)$   
**by** (*induction M*) *auto*

**lemma** *multiset\_filter\_mono2*:  $\langle filter\_mset P A \subseteq\# filter\_mset Q A \longleftrightarrow (\forall a \in\# A. P a \longrightarrow Q a) \rangle$   
**by** (*induction A*) (*auto intro: subset\_mset.order.trans*)

**lemma** *image\_filter\_cong*:  
**assumes**  $\langle \wedge C. C \in\# M \implies P C \implies f C = g C \rangle$

**shows**  $\langle \{ \#f C. C \in \# \{ \#C \in \# M. P C \# \} \# \} = \{ \#g C \mid C \in \# M. P C \# \} \rangle$   
**using** *assms* **by** (*induction M*) *auto*

**lemma** *image\_mset\_filter\_swap2*:  $\langle \{ \#C \in \# \{ \#P x. x \in \# D \# \}. Q C \# \} = \{ \#P x. x \in \# \{ \#C \mid C \in \# D. Q (P C) \# \} \# \} \rangle$   
**by** (*simp add: image\_mset\_filter\_swap*)

**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**  $\{ \#x \in \# \text{mset\_set } X. P x \# \} = \{ \# \}$

**proof** –

**have** *empty\_empty*:  $\bigwedge Y. \text{set\_mset } Y = \{ \} \implies Y = \{ \# \}$

**by** *auto*

**from** *assms* **have** *set\_mset*  $\{ \#x \in \# \text{mset\_set } X. P x \# \} = \{ \}$

**by** *auto*

**then show** *?thesis*

**by** (*rule empty\_empty*)

**qed**

## 2.4 Lemmas about Sum

**lemma** *sum\_image\_mset\_sum\_map*[*simp*]:  $\text{sum\_mset } (\text{image\_mset } f (\text{mset } xs)) = \text{sum\_list } (\text{map } f xs)$   
**by** (*metis mset\_map sum\_mset\_sum\_list*)

**lemma** *sum\_image\_mset\_mono*:

**fixes**  $f :: 'a \Rightarrow 'b::\text{canonically\_ordered\_monoid\_add}$

**assumes**  $sub: A \subseteq \# B$

**shows**  $(\sum m \in \# A. f m) \leq (\sum m \in \# B. f m)$

**by** (*metis image\_mset\_union le\_iff\_add subset\_mset.add\_diff\_inverse sum\_mset.union*)

**lemma** *sum\_image\_mset\_mono\_mem*:

$n \in \# M \implies f n \leq (\sum m \in \# M. f m)$  **for**  $f :: 'a \Rightarrow 'b::\text{canonically\_ordered\_monoid\_add}$

**using** *le\_iff\_add multi\_member\_split* **by** *fastforce*

**lemma** *count\_sum\_mset\_if\_1\_0*:  $\langle \text{count } M a = (\sum x \in \# M. \text{if } x = a \text{ then } 1 \text{ else } 0) \rangle$

**by** (*induction M*) *auto*

**lemma** *sum\_mset\_dvd*:

**fixes**  $k :: 'a::\text{comm\_semiring\_1\_cancel}$

**assumes**  $\forall m \in \# M. k \text{ dvd } f m$

**shows**  $k \text{ dvd } (\sum m \in \# M. f m)$

**using** *assms* **by** (*induct M*) *auto*

**lemma** *sum\_mset\_distrib\_div\_if\_dvd*:

**fixes**  $k :: 'a::\text{unique\_euclidean\_semiring}$

**assumes**  $\forall m \in \# M. k \text{ dvd } f m$

**shows**  $(\sum m \in \# M. f m) \text{ div } k = (\sum m \in \# M. f m \text{ div } k)$

**using** *assms* **by** (*induct M*) (*auto simp: div\_plus\_div\_distrib\_dvd\_left*)

## 2.5 Lemmas about Remove

**lemma** *set\_mset\_minus\_replicate\_mset*[*simp*]:

$n \geq \text{count } A a \implies \text{set\_mset } (A - \text{replicate\_mset } n a) = \text{set\_mset } A - \{ a \}$

$n < \text{count } A a \implies \text{set\_mset } (A - \text{replicate\_mset } n a) = \text{set\_mset } A$

**unfolding** *set\_mset\_def* **by** (*auto split: if\_split simp: not\_in\_iff*)

**abbreviation** *removeAll\_mset* ::  $'a \Rightarrow 'a \text{ multiset} \Rightarrow 'a \text{ multiset}$  **where**

*removeAll\_mset C M*  $\equiv M - \text{replicate\_mset } (\text{count } M C) C$

**lemma** *mset\_removeAll*[*simp, code*]:  $\text{removeAll\_mset } C (\text{mset } L) = \text{mset } (\text{removeAll } C L)$

by (induction L) (auto simp: ac\_simps multiset\_eq\_iff split: if\_split\_asm)

**lemma** *removeAll\_mset\_filter\_mset*:  $\text{removeAll\_mset } C M = \text{filter\_mset } ((\neq) C) M$   
 by (induction M) (auto simp: ac\_simps multiset\_eq\_iff)

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

**lemma** *removeAll\_subseteq\_remove1\_mset*:  $\text{removeAll\_mset } x M \subseteq\# \text{remove1\_mset } x M$   
 by (auto simp: subseteq\_mset\_def)

**lemma** *in\_remove1\_mset\_neq*:  
**assumes** *ab*:  $a \neq b$   
**shows**  $a \in\# \text{remove1\_mset } b C \longleftrightarrow a \in\# C$   
**by** (metis assms diff\_single\_trivial in\_diffD insert\_DiffM insert\_noteq\_member)

**lemma** *size\_mset\_removeAll\_mset\_le\_iff*:  $\text{size } (\text{removeAll\_mset } x M) < \text{size } M \longleftrightarrow x \in\# M$   
 by (auto intro: count\_inI mset\_subset\_size simp: subset\_mset\_def multiset\_eq\_iff)

**lemma** *size\_remove1\_mset>If*:  $\langle \text{size } (\text{remove1\_mset } x M) = \text{size } M - (\text{if } x \in\# M \text{ then } 1 \text{ else } 0) \rangle$   
 by (auto simp: size\_Diff\_subset\_Int)

**lemma** *size\_mset\_remove1\_mset\_le\_iff*:  $\text{size } (\text{remove1\_mset } x M) < \text{size } M \longleftrightarrow x \in\# M$   
**using** *less\_irrefl*  
**by** (fastforce intro!: mset\_subset\_size elim: in\_countE simp: subset\_mset\_def multiset\_eq\_iff)

**lemma** *remove\_1\_mset\_id\_iff\_notin*:  $\text{remove1\_mset } a M = M \longleftrightarrow a \notin\# M$   
**by** (meson diff\_single\_trivial multi\_drop\_mem\_not\_eq)

**lemma** *id\_remove\_1\_mset\_iff\_notin*:  $M = \text{remove1\_mset } a M \longleftrightarrow a \notin\# M$   
**using** *remove\_1\_mset\_id\_iff\_notin* **by** metis

**lemma** *remove1\_mset\_eqE*:  
 $\text{remove1\_mset } L x1 = M \implies$   
 $(L \in\# x1 \implies x1 = M + \{\#L\# \} \implies P) \implies$   
 $(L \notin\# x1 \implies x1 = M \implies P) \implies$   
 $P$   
**by** (cases L  $\in\#$  x1) auto

**lemma** *image\_filter\_ne\_mset[simp]*:  
 $\text{image\_mset } f \{\#x \in\# M. f x \neq y\# \} = \text{removeAll\_mset } y (\text{image\_mset } f M)$   
**by** (induction M) simp\_all

**lemma** *image\_mset\_remove1\_mset\_if*:  
 $\text{image\_mset } f (\text{remove1\_mset } a M) =$   
 $(\text{if } a \in\# M \text{ then } \text{remove1\_mset } (f a) (\text{image\_mset } f M) \text{ else } \text{image\_mset } f M)$   
**by** (auto simp: image\_mset\_Diff)

**lemma** *filter\_mset\_neq*:  $\{\#x \in\# M. x \neq y\# \} = \text{removeAll\_mset } y M$   
**by** (metis add\_diff\_cancel\_left' filter\_eq\_replicate\_mset multiset\_partition)

**lemma** *filter\_mset\_neq\_cond*:  $\{\#x \in\# M. P x \wedge x \neq y\# \} = \text{removeAll\_mset } y \{\#x \in\# M. P x\# \}$   
**by** (metis filter\_filter\_mset filter\_mset\_neq)

**lemma** *remove1\_mset\_add\_mset>If*:  
 $\text{remove1\_mset } L (\text{add\_mset } L' C) = (\text{if } L = L' \text{ then } C \text{ else } \text{remove1\_mset } L C + \{\#L'\# \})$   
**by** (auto simp: multiset\_eq\_iff)

**lemma** *minus\_remove1\_mset\_if*:  
 $A - \text{remove1\_mset } b B = (\text{if } b \in\# B \wedge b \in\# A \wedge \text{count } A b \geq \text{count } B b \text{ then } \{\#b\# \} + (A - B) \text{ else } A - B)$   
**by** (auto simp: multiset\_eq\_iff count\_greater\_zero\_iff[symmetric]  
 simp del: count\_greater\_zero\_iff)

**lemma** *add\_mset\_eq\_add\_mset\_ne*:

$a \neq b \implies \text{add\_mset } a \ A = \text{add\_mset } b \ B \iff a \in\# B \wedge b \in\# A \wedge A = \text{add\_mset } b \ (B - \{\#a\#})$   
**by** (*metis* (*no\_types*, *lifting*) *diff\_single\_eq\_union* *diff\_union\_swap* *multi\_self\_add\_other\_not\_self* *remove\_1\_mset\_id\_iff\_notin* *union\_single\_eq\_diff*)

**lemma** *add\_mset\_eq\_add\_mset*:  $\langle \text{add\_mset } a \ M = \text{add\_mset } b \ M' \iff$

$(a = b \wedge M = M') \vee (a \neq b \wedge b \in\# M \wedge \text{add\_mset } a \ (M - \{\#b\#}) = M' \rangle$

**by** (*metis* *add\_mset\_eq\_add\_mset\_ne* *add\_mset\_remove\_trivial* *union\_single\_eq\_member*)

**lemma** *add\_mset\_remove\_trivial\_iff*:  $\langle N = \text{add\_mset } a \ (N - \{\#b\#}) \iff a \in\# N \wedge a = b \rangle$

**by** (*metis* *add\_left\_cancel* *add\_mset\_remove\_trivial* *insert\_DiffM2* *single\_eq\_single* *size\_mset\_remove1\_mset\_le\_iff* *union\_single\_eq\_member*)

**lemma** *trivial\_add\_mset\_remove\_iff*:  $\langle \text{add\_mset } a \ (N - \{\#b\#}) = N \iff a \in\# N \wedge a = b \rangle$

**by** (*subst* *eq\_commute*) (*fact* *add\_mset\_remove\_trivial\_iff*)

**lemma** *remove1\_single\_empty\_iff[simp]*:  $\langle \text{remove1\_mset } L \ \{\#L'\#\} = \{\#\} \iff L = L' \rangle$

**using** *add\_mset\_remove\_trivial\_iff* **by** *fastforce*

**lemma** *add\_mset\_less\_imp\_less\_remove1\_mset*:

**assumes** *xM\_lt\_N*:  $\text{add\_mset } x \ M < N$

**shows**  $M < \text{remove1\_mset } x \ N$

**proof** –

**have**  $M < N$

**using** *assms* *le\_multiset\_right\_total* *mset\_le\_trans* **by** *blast*

**then show** *?thesis*

**by** (*metis* *add\_less\_cancel\_right* *add\_mset\_add\_single* *diff\_single\_trivial* *insert\_DiffM2* *xM\_lt\_N*)

**qed**

## 2.6 Lemmas about Replicate

**lemma** *replicate\_mset\_minus\_replicate\_mset\_same[simp]*:

$\text{replicate\_mset } m \ x - \text{replicate\_mset } n \ x = \text{replicate\_mset } (m - n) \ x$

**by** (*induct* *m* *arbitrary*: *n*, *simp*, *metis* *left\_diff\_repeat\_mset\_distrib'* *repeat\_mset\_replicate\_mset*)

**lemma** *replicate\_mset\_subset\_iff\_lt[simp]*:  $\text{replicate\_mset } m \ x \subset\# \text{replicate\_mset } n \ x \iff m < n$

**by** (*induct* *n* *m* *rule*: *diff\_induct*) (*auto* *intro*: *subset\_mset.gr\_zeroI*)

**lemma** *replicate\_mset\_subseteq\_iff\_le[simp]*:  $\text{replicate\_mset } m \ x \subseteq\# \text{replicate\_mset } n \ x \iff m \leq n$

**by** (*induct* *n* *m* *rule*: *diff\_induct*) *auto*

**lemma** *replicate\_mset\_lt\_iff\_lt[simp]*:  $\text{replicate\_mset } m \ x < \text{replicate\_mset } n \ x \iff m < n$

**by** (*induct* *n* *m* *rule*: *diff\_induct*) (*auto* *intro*: *subset\_mset.gr\_zeroI* *gr\_zeroI*)

**lemma** *replicate\_mset\_le\_iff\_le[simp]*:  $\text{replicate\_mset } m \ x \leq \text{replicate\_mset } n \ x \iff m \leq n$

**by** (*induct* *n* *m* *rule*: *diff\_induct*) *auto*

**lemma** *replicate\_mset\_eq\_iff[simp]*:

$\text{replicate\_mset } m \ x = \text{replicate\_mset } n \ y \iff m = n \wedge (m \neq 0 \implies x = y)$

**by** (*cases* *m*; *cases* *n*; *simp*)

(*metis* *in\_replicate\_mset* *insert\_noteq\_member* *size\_replicate\_mset* *union\_single\_eq\_diff*)

**lemma** *replicate\_mset\_plus*:  $\text{replicate\_mset } (a + b) \ C = \text{replicate\_mset } a \ C + \text{replicate\_mset } b \ C$

**by** (*induct* *a*) (*auto* *simp*: *ac\_simps*)

**lemma** *mset\_replicate\_replicate\_mset*:  $\text{mset } (\text{replicate } n \ L) = \text{replicate\_mset } n \ L$

**by** (*induction* *n*) *auto*

**lemma** *set\_mset\_single\_iff\_replicate\_mset*:  $\text{set\_mset } U = \{a\} \iff (\exists n > 0. U = \text{replicate\_mset } n \ a)$

**by** (*rule*, *metis* *count\_greater\_zero\_iff* *count\_replicate\_mset* *insertI1* *multi\_count\_eq\_singletonD* *zero\_less\_iff\_neq\_zero*, *force*)

**lemma** *ex\_replicate\_mset\_if\_all\_elems\_eq*:



**assumes**  $\forall x \in \# M. x = y$   
**shows**  $\exists n. M = \text{replicate\_mset } n \ y$   
**using** *assms* **by** (*metis count\_replicate\_mset mem\_Collect\_eq multiset\_eqI neq0\_conv set\_mset\_def*)

## 2.7 Multiset and Set Conversions

**lemma** *count\_mset\_set\_if*:  $\text{count } (\text{mset\_set } A) \ a = (\text{if } a \in A \wedge \text{finite } A \text{ then } 1 \text{ else } 0)$   
**by** *auto*

**lemma** *mset\_set\_set\_mset\_empty\_mempty*[*iff*]:  $\text{mset\_set } (\text{set\_mset } D) = \{\#\} \longleftrightarrow D = \{\#\}$   
**by** (*simp add: mset\_set\_empty\_iff*)

**lemma** *count\_mset\_set\_le\_one*:  $\text{count } (\text{mset\_set } A) \ x \leq 1$   
**by** (*simp add: count\_mset\_set\_if*)

**lemma** *mset\_set\_set\_mset\_subseteq*[*simp*]:  $\text{mset\_set } (\text{set\_mset } A) \subseteq \# A$   
**by** (*simp add: mset\_set\_set\_mset\_msubset*)

**lemma** *mset\_sorted\_list\_of\_set*[*simp*]:  $\text{mset } (\text{sorted\_list\_of\_set } A) = \text{mset\_set } A$   
**by** (*metis mset\_sorted\_list\_of\_multiset sorted\_list\_of\_mset\_set*)

**lemma** *sorted\_sorted\_list\_of\_multiset*[*simp*]:  
 $\text{sorted } (\text{sorted\_list\_of\_multiset } (M :: 'a::\text{linorder multiset}))$   
**by** (*metis mset\_sorted\_list\_of\_multiset sorted\_list\_of\_multiset\_mset sorted\_sort*)

**lemma** *mset\_take\_subseteq*:  $\text{mset } (\text{take } n \ xs) \subseteq \# \text{mset } xs$   
**apply** (*induct xs arbitrary: n*)  
**apply** *simp*  
**by** (*case\_tac n*) *simp\_all*

**lemma** *sorted\_list\_of\_multiset\_eq\_Nil*[*simp*]:  $\text{sorted\_list\_of\_multiset } M = [] \longleftrightarrow M = \{\#\}$   
**by** (*metis mset\_sorted\_list\_of\_multiset sorted\_list\_of\_multiset\_empty*)

## 2.8 Duplicate Removal

**definition** *remdups\_mset* ::  $'v \text{ multiset} \Rightarrow 'v \text{ multiset}$  **where**  
 $\text{remdups\_mset } S = \text{mset\_set } (\text{set\_mset } S)$

**lemma** *set\_mset\_remdups\_mset*[*simp*]:  $\text{set\_mset } (\text{remdups\_mset } A) = \text{set\_mset } A$   
**unfolding** *remdups\_mset\_def* **by** *auto*

**lemma** *count\_remdups\_mset\_eq\_1*:  $a \in \# \text{remdups\_mset } A \longleftrightarrow \text{count } (\text{remdups\_mset } A) \ a = 1$   
**unfolding** *remdups\_mset\_def* **by** (*auto simp: count\_eq\_zero\_iff intro: count\_inI*)

**lemma** *remdups\_mset\_empty*[*simp*]:  $\text{remdups\_mset } \{\#\} = \{\#\}$   
**unfolding** *remdups\_mset\_def* **by** *auto*

**lemma** *remdups\_mset\_singleton*[*simp*]:  $\text{remdups\_mset } \{\#a\} = \{\#a\}$   
**unfolding** *remdups\_mset\_def* **by** *auto*

**lemma** *remdups\_mset\_eq\_empty*[*iff*]:  $\text{remdups\_mset } D = \{\#\} \longleftrightarrow D = \{\#\}$   
**unfolding** *remdups\_mset\_def* **by** *blast*

**lemma** *remdups\_mset\_singleton\_sum*[*simp*]:  
 $\text{remdups\_mset } (\text{add\_mset } a \ A) = (\text{if } a \in \# A \text{ then } \text{remdups\_mset } A \text{ else } \text{add\_mset } a \ (\text{remdups\_mset } A))$   
**unfolding** *remdups\_mset\_def* **by** (*simp\_all add: insert\_absorb*)

**lemma** *mset\_remdups\_remdups\_mset*[*simp*]:  $\text{mset } (\text{remdups } D) = \text{remdups\_mset } (\text{mset } D)$   
**by** (*induction D*) (*auto simp add: ac\_simps*)

**declare** *mset\_remdups\_remdups\_mset*[*symmetric, code*]

**definition** *distinct\_mset* ::  $'a \text{ multiset} \Rightarrow \text{bool}$  **where**  
 $\text{distinct\_mset } S \longleftrightarrow (\forall a. a \in \# S \longrightarrow \text{count } S \ a = 1)$

**lemma** *distinct\_mset\_count\_less\_1*:  $\text{distinct\_mset } S \longleftrightarrow (\forall a. \text{count } S \ a \leq 1)$   
**using** *eq\_iff\_nat\_le\_linear* **unfolding** *distinct\_mset\_def* **by** *fastforce*

**lemma** *distinct\_mset\_empty[simp]*:  $\text{distinct\_mset } \{\#\}$   
**unfolding** *distinct\_mset\_def* **by** *auto*

**lemma** *distinct\_mset\_singleton*:  $\text{distinct\_mset } \{\#a\}$   
**unfolding** *distinct\_mset\_def* **by** *auto*

**lemma** *distinct\_mset\_union*:  
**assumes** *dist*:  $\text{distinct\_mset } (A + B)$   
**shows**  $\text{distinct\_mset } A$   
**unfolding** *distinct\_mset\_count\_less\_1*  
**proof** (*rule allI*)  
**fix** *a*  
**have**  $\langle \text{count } A \ a \leq \text{count } (A + B) \ a \rangle$  **by** *auto*  
**moreover have**  $\langle \text{count } (A + B) \ a \leq 1 \rangle$   
**using** *dist* **unfolding** *distinct\_mset\_count\_less\_1* **by** *auto*  
**ultimately show**  $\langle \text{count } A \ a \leq 1 \rangle$   
**by** *simp*  
**qed**

**lemma** *distinct\_mset\_minus[simp]*:  $\text{distinct\_mset } A \Longrightarrow \text{distinct\_mset } (A - B)$   
**by** (*metis diff\_subset\_eq\_self mset\_subset\_eq\_exists\_conv distinct\_mset\_union*)

**lemma** *count\_remdups\_mset>If*:  $\langle \text{count } (\text{remdups\_mset } A) \ a = (\text{if } a \in\# \ A \ \text{then } 1 \ \text{else } 0) \rangle$   
**unfolding** *remdups\_mset\_def* **by** *auto*

**lemma** *distinct\_mset\_remdups\_union\_mset*:  
**assumes**  $\text{distinct\_mset } A$  **and**  $\text{distinct\_mset } B$   
**shows**  $A \cup\# \ B = \text{remdups\_mset } (A + B)$   
**using** *assms nat\_le\_linear* **unfolding** *remdups\_mset\_def*  
**by** (*force simp add: multiset\_eq\_iff\_max\_def count\_mset\_set\_if distinct\_mset\_def not\_in\_iff*)

**lemma** *distinct\_mset\_add\_mset[simp]*:  $\text{distinct\_mset } (\text{add\_mset } a \ L) \longleftrightarrow a \notin\# \ L \wedge \text{distinct\_mset } L$   
**unfolding** *distinct\_mset\_def*  
**apply** (*rule iffI*)  
**apply** (*auto split: if\_split\_asm; fail*)[]  
**by** (*auto simp: not\_in\_iff; fail*)

**lemma** *distinct\_mset\_size\_eq\_card*:  $\text{distinct\_mset } C \Longrightarrow \text{size } C = \text{card } (\text{set\_mset } C)$   
**by** (*induction C*) *auto*

**lemma** *distinct\_mset\_add*:  
 $\text{distinct\_mset } (L + L') \longleftrightarrow \text{distinct\_mset } L \wedge \text{distinct\_mset } L' \wedge L \cap\# \ L' = \{\#\}$   
**by** (*induction L arbitrary: L'*) *auto*

**lemma** *distinct\_mset\_set\_mset\_ident[simp]*:  $\text{distinct\_mset } M \Longrightarrow \text{mset\_set } (\text{set\_mset } M) = M$   
**by** (*induction M*) *auto*

**lemma** *distinct\_finite\_set\_mset\_subseteq\_iff[iff]*:  
**assumes**  $\text{distinct\_mset } M$  *finite* *N*  
**shows**  $\text{set\_mset } M \subseteq N \longleftrightarrow M \subseteq\# \ \text{mset\_set } N$   
**by** (*metis assms distinct\_mset\_set\_mset\_ident finite\_set\_mset msubset\_mset\_set\_iff*)

**lemma** *distinct\_mem\_diff\_mset*:  
**assumes** *dist*:  $\text{distinct\_mset } M$  **and** *mem*:  $x \in \text{set\_mset } (M - N)$   
**shows**  $x \notin \text{set\_mset } N$   
**proof** –  
**have**  $\text{count } M \ x = 1$   
**using** *dist mem* **by** (*meson distinct\_mset\_def in\_diffD*)  
**then show** *?thesis*

using mem by (metis count\_greater\_eq\_one\_iff\_in\_diff\_count not\_less)  
qed

**lemma** *distinct\_set\_mset\_eq*:  
assumes *distinct\_mset M distinct\_mset N set\_mset M = set\_mset N*  
shows  $M = N$   
using *assms distinct\_mset\_set\_mset\_ident* by *fastforce*

**lemma** *distinct\_mset\_union\_mset*[*simp*]:  
 $\langle \text{distinct\_mset } (D \cup\# C) \longleftrightarrow \text{distinct\_mset } D \wedge \text{distinct\_mset } C \rangle$   
unfolding *distinct\_mset\_count\_less\_1* by *force*

**lemma** *distinct\_mset\_inter\_mset*:  
 $\text{distinct\_mset } C \implies \text{distinct\_mset } (C \cap\# D)$   
 $\text{distinct\_mset } D \implies \text{distinct\_mset } (C \cap\# D)$   
by (*simp\_all add: multiset\_inter\_def*,  
*metis distinct\_mset\_minus multiset\_inter\_commute multiset\_inter\_def*)

**lemma** *distinct\_mset\_remove1\_All*:  $\text{distinct\_mset } C \implies \text{remove1\_mset } L C = \text{removeAll\_mset } L C$   
by (*auto simp: multiset\_eq\_iff distinct\_mset\_count\_less\_1*)

**lemma** *distinct\_mset\_size\_2*:  $\text{distinct\_mset } \{\#a, b\# \} \longleftrightarrow a \neq b$   
by *auto*

**lemma** *distinct\_mset\_filter*:  $\text{distinct\_mset } M \implies \text{distinct\_mset } \{\# L \in\# M. P L\# \}$   
by (*simp add: distinct\_mset\_def*)

**lemma** *distinct\_mset\_mset\_distinct*[*simp*]:  $\langle \text{distinct\_mset } (\text{mset } xs) = \text{distinct } xs \rangle$   
by (*induction xs*) *auto*

**lemma** *distinct\_image\_mset\_inj*:  
 $\langle \text{inj\_on } f (\text{set\_mset } M) \implies \text{distinct\_mset } (\text{image\_mset } f M) \longleftrightarrow \text{distinct\_mset } M \rangle$   
by (*induction M*) (*auto simp: inj\_on\_def*)

## 2.9 Repeat Operation

**lemma** *repeat\_mset\_compower*:  $\text{repeat\_mset } n A = (((+) A) \sim n) \{\#\}$   
by (*induction n*) *auto*

**lemma** *repeat\_mset\_prod*:  $\text{repeat\_mset } (m * n) A = (((+) (\text{repeat\_mset } n A)) \sim m) \{\#\}$   
by (*induction m*) (*auto simp: repeat\_mset\_distrib*)

## 2.10 Cartesian Product

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

**definition** *Sigma\_mset* ::  $'a \text{ multiset} \Rightarrow ('a \Rightarrow 'b \text{ multiset}) \Rightarrow ('a \times 'b) \text{ multiset}$  **where**  
 $\text{Sigma\_mset } A B \equiv \sum\# \{\#\{ \#(a, b). b \in\# B a\#\}. a \in\# A \#\}$

**abbreviation** *Times\_mset* ::  $'a \text{ multiset} \Rightarrow 'b \text{ multiset} \Rightarrow ('a \times 'b) \text{ multiset}$  (**infix**  $\times\#$  80) **where**  
 $\text{Times\_mset } A B \equiv \text{Sigma\_mset } A (\lambda_. B)$

**hide-const** (**open**) *Times\_mset*

Contrary to the set version  $A \times B$ , we use the non-ASCII symbol  $\in\#$ .

**syntax**

$\_Sigma\_mset :: [\text{ptrn}, 'a \text{ multiset}, 'b \text{ multiset}] \Rightarrow ('a * 'b) \text{ multiset}$   
 $((3\SIGMAMSET \_ \in\# \_ / \_) [0, 0, 10] 10)$

**translations**

$\text{SIGMAMSET } x \in\# A. B == \text{CONST Sigma\_mset } A (\lambda x. B)$

Link between the multiset and the set cartesian product:

**lemma** *Times\_mset\_Times*:  $set\_mset (A \times\# B) = set\_mset A \times set\_mset B$   
**unfolding** *Sigma\_mset\_def* **by** *auto*

**lemma** *Sigma\_msetI* [*intro!*]:  $\llbracket a \in\# A; b \in\# B \ a \rrbracket \implies (a, b) \in\# Sigma\_mset A B$   
**by** (*unfold Sigma\_mset\_def*) *auto*

**lemma** *Sigma\_msetE*[*elim!*]:  $\llbracket c \in\# Sigma\_mset A B; \bigwedge x y. \llbracket x \in\# A; y \in\# B \ x; c = (x, y) \rrbracket \implies P \rrbracket \implies P$   
**by** (*unfold Sigma\_mset\_def*) *auto*

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

**lemma** *Sigma\_msetD1*:  $(a, b) \in\# Sigma\_mset A B \implies a \in\# A$   
**by** *blast*

**lemma** *Sigma\_msetD2*:  $(a, b) \in\# Sigma\_mset A B \implies b \in\# B \ a$   
**by** *blast*

**lemma** *Sigma\_msetE2*:  $\llbracket (a, b) \in\# Sigma\_mset A B; \llbracket a \in\# A; b \in\# B \ a \rrbracket \implies P \rrbracket \implies P$   
**by** *blast*

**lemma** *Sigma\_mset\_cong*:  
 $\llbracket A = B; \bigwedge x. x \in\# B \implies C \ x = D \ x \rrbracket \implies (SIGMAMSET \ x \in\# A. C \ x) = (SIGMAMSET \ x \in\# B. D \ x)$   
**by** (*metis (mono\_tags, lifting) Sigma\_mset\_def image\_mset\_cong*)

**lemma** *count\_sum\_mset*:  $count (\sum\# M) \ b = (\sum P \in\# M. count \ P \ b)$   
**by** (*induction M*) *auto*

**lemma** *Sigma\_mset\_plus\_distrib1*[*simp*]:  $Sigma\_mset (A + B) \ C = Sigma\_mset A \ C + Sigma\_mset B \ C$   
**unfolding** *Sigma\_mset\_def* **by** *auto*

**lemma** *Sigma\_mset\_plus\_distrib2*[*simp*]:  
 $Sigma\_mset A (\lambda i. B \ i + C \ i) = Sigma\_mset A \ B + Sigma\_mset A \ C$   
**unfolding** *Sigma\_mset\_def* **by** (*induction A*) (*auto simp: multiset\_eq\_iff*)

**lemma** *Times\_mset\_single\_left*:  $\{\#a\#\} \times\# B = image\_mset (Pair \ a) \ B$   
**unfolding** *Sigma\_mset\_def* **by** *auto*

**lemma** *Times\_mset\_single\_right*:  $A \times\# \{\#b\#\} = image\_mset (\lambda a. Pair \ a \ b) \ A$   
**unfolding** *Sigma\_mset\_def* **by** (*induction A*) *auto*

**lemma** *Times\_mset\_single\_single*[*simp*]:  $\{\#a\#\} \times\# \{\#b\#\} = \{\#(a, b)\#\}$   
**unfolding** *Sigma\_mset\_def* **by** *simp*

**lemma** *count\_image\_mset\_Pair*:  
 $count (image\_mset (Pair \ a) \ B) (x, b) = (if \ x = a \ then \ count \ B \ b \ else \ 0)$   
**by** (*induction B*) *auto*

**lemma** *count\_Sigma\_mset*:  $count (Sigma\_mset A \ B) (a, b) = count \ A \ a * count (B \ a) \ b$   
**by** (*induction A*) (*auto simp: Sigma\_mset\_def count\_image\_mset\_Pair*)

**lemma** *Sigma\_mset\_empty1*[*simp*]:  $Sigma\_mset \ \{\#\} \ B = \{\#\}$   
**unfolding** *Sigma\_mset\_def* **by** *auto*

**lemma** *Sigma\_mset\_empty2*[*simp*]:  $A \times\# \{\#\} = \{\#\}$   
**by** (*auto simp: multiset\_eq\_iff count\_Sigma\_mset*)

**lemma** *Sigma\_mset\_mono*:  
**assumes**  $A \subseteq\# C$  **and**  $\bigwedge x. x \in\# A \implies B \ x \subseteq\# D \ x$   
**shows**  $Sigma\_mset A \ B \subseteq\# Sigma\_mset C \ D$

**proof** –

**have**  $count \ A \ a * count (B \ a) \ b \leq count \ C \ a * count (D \ a) \ b$  **for**  $a \ b$   
**using** *assms* **unfolding** *subteq\_mset\_def* **by** (*metis count\_inI eq\_iff mult\_eq\_0\_iff mult\_le\_mono*)  
**then show** *?thesis*  
**by** (*auto simp: subteq\_mset\_def count\_Sigma\_mset*)

qed

**lemma** *mem\_Sigma\_mset\_iff*[iff]:  $((a,b) \in\# \text{Sigma\_mset } A \ B) = (a \in\# A \wedge b \in\# B \ a)$   
by *blast*

**lemma** *mem\_Times\_mset\_iff*:  $x \in\# A \times\# B \longleftrightarrow \text{fst } x \in\# A \wedge \text{snd } x \in\# B$   
by (*induct x*) *simp*

**lemma** *Sigma\_mset\_empty\_iff*:  $(\text{SIGMAMSET } i \in\# I. X \ i) = \{\#\} \longleftrightarrow (\forall i \in\# I. X \ i = \{\#\})$   
by (*auto simp: Sigma\_mset\_def*)

**lemma** *Times\_mset\_subset\_mset\_cancel1*:  $x \in\# A \implies (A \times\# B \subseteq\# A \times\# C) = (B \subseteq\# C)$   
by (*auto simp: subteq\_mset\_def count\_Sigma\_mset*)

**lemma** *Times\_mset\_subset\_mset\_cancel2*:  $x \in\# C \implies (A \times\# C \subseteq\# B \times\# C) = (A \subseteq\# B)$   
by (*auto simp: subteq\_mset\_def count\_Sigma\_mset*)

**lemma** *Times\_mset\_eq\_cancel2*:  $x \in\# C \implies (A \times\# C = B \times\# C) = (A = B)$   
by (*auto simp: multiset\_eq\_iff count\_Sigma\_mset dest!: in\_countE*)

**lemma** *split\_paired\_Ball\_mset\_Sigma\_mset*[*simp*]:  
 $(\forall z \in\# \text{Sigma\_mset } A \ B. P \ z) \longleftrightarrow (\forall x \in\# A. \forall y \in\# B \ x. P \ (x, y))$   
by *blast*

**lemma** *split\_paired\_Bex\_mset\_Sigma\_mset*[*simp*]:  
 $(\exists z \in\# \text{Sigma\_mset } A \ B. P \ z) \longleftrightarrow (\exists x \in\# A. \exists y \in\# B \ x. P \ (x, y))$   
by *blast*

**lemma** *sum\_mset\_if\_eq\_constant*:  
 $(\sum x \in\# M. \text{if } a = x \text{ then } (f \ x) \text{ else } 0) = (((+) (f \ a)) \ \frown \ (\text{count } M \ a)) \ 0$   
by (*induction M*) (*auto simp: ac\_simps*)

**lemma** *iterate\_op\_plus*:  $((+(+) \ k) \ \frown \ m) \ 0 = k * m$   
by (*induction m*) *auto*

**lemma** *untion\_image\_mset\_Pair\_distribute*:  
 $\sum\# \{\#\text{image\_mset } (\text{Pair } x) \ (C \ x). \ x \in\# J - I\#\} =$   
 $\sum\# \{\#\text{image\_mset } (\text{Pair } x) \ (C \ x). \ x \in\# J\#\} - \sum\# \{\#\text{image\_mset } (\text{Pair } x) \ (C \ x). \ x \in\# I\#\}$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*iterate\_op\_plus diff\_mult\_distrib2*)

**lemma** *Sigma\_mset\_Un\_distrib1*:  $\text{Sigma\_mset } (I \cup\# J) \ C = \text{Sigma\_mset } I \ C \cup\# \text{Sigma\_mset } J \ C$   
by (*auto simp: Sigma\_mset\_def sup\_subset\_mset\_def untion\_image\_mset\_Pair\_distribute*)

**lemma** *Sigma\_mset\_Un\_distrib2*:  $(\text{SIGMAMSET } i \in\# I. A \ i \cup\# B \ i) = \text{Sigma\_mset } I \ A \cup\# \text{Sigma\_mset } I \ B$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*Sigma\_mset\_def diff\_mult\_distrib2 iterate\_op\_plus max\_def not\_in\_iff*)

**lemma** *Sigma\_mset\_Int\_distrib1*:  $\text{Sigma\_mset } (I \cap\# J) \ C = \text{Sigma\_mset } I \ C \cap\# \text{Sigma\_mset } J \ C$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*Sigma\_mset\_def iterate\_op\_plus min\_def not\_in\_iff*)

**lemma** *Sigma\_mset\_Int\_distrib2*:  $(\text{SIGMAMSET } i \in\# I. A \ i \cap\# B \ i) = \text{Sigma\_mset } I \ A \cap\# \text{Sigma\_mset } I \ B$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*Sigma\_mset\_def iterate\_op\_plus min\_def not\_in\_iff*)

**lemma** *Sigma\_mset\_Diff\_distrib1*:  $\text{Sigma\_mset } (I - J) \ C = \text{Sigma\_mset } I \ C - \text{Sigma\_mset } J \ C$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*Sigma\_mset\_def iterate\_op\_plus min\_def not\_in\_iff diff\_mult\_distrib2*)

**lemma** *Sigma\_mset\_Diff\_distrib2*:  $(\text{SIGMAMSET } i \in\# I. A \ i - B \ i) = \text{Sigma\_mset } I \ A - \text{Sigma\_mset } I \ B$   
by (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant*  
*Sigma\_mset\_def iterate\_op\_plus min\_def not\_in\_iff diff\_mult\_distrib*)

**lemma** *Sigma\_mset\_Union*:  $\text{Sigma\_mset } (\sum \# X) B = (\sum \# (\text{image\_mset } (\lambda A. \text{Sigma\_mset } A B) X))$   
**by** (*auto simp: multiset\_eq\_iff count\_sum\_mset count\_image\_mset\_Pair sum\_mset\_if\_eq\_constant Sigma\_mset\_def iterate\_op\_plus min\_def not\_in\_iff sum\_mset\_distrib\_left*)

**lemma** *Times\_mset\_Un\_distrib1*:  $(A \cup \# B) \times \# C = A \times \# C \cup \# B \times \# C$   
**by** (*fact Sigma\_mset\_Un\_distrib1*)

**lemma** *Times\_mset\_Int\_distrib1*:  $(A \cap \# B) \times \# C = A \times \# C \cap \# B \times \# C$   
**by** (*fact Sigma\_mset\_Int\_distrib1*)

**lemma** *Times\_mset\_Diff\_distrib1*:  $(A - B) \times \# C = A \times \# C - B \times \# C$   
**by** (*fact Sigma\_mset\_Diff\_distrib1*)

**lemma** *Times\_mset\_empty[simp]*:  $A \times \# B = \{\#\} \longleftrightarrow A = \{\#\} \vee B = \{\#\}$   
**by** (*auto simp: Sigma\_mset\_empty\_iff*)

**lemma** *Times\_insert\_left*:  $A \times \# \text{add\_mset } x B = A \times \# B + \text{image\_mset } (\lambda a. \text{Pair } a x) A$   
**unfolding** *add\_mset\_add\_single[of x B] Sigma\_mset\_plus\_distrib2*  
**by** (*simp add: Times\_mset\_single\_right*)

**lemma** *Times\_insert\_right*:  $\text{add\_mset } a A \times \# B = A \times \# B + \text{image\_mset } (\text{Pair } a) B$   
**unfolding** *add\_mset\_add\_single[of a A] Sigma\_mset\_plus\_distrib1*  
**by** (*simp add: Times\_mset\_single\_left*)

**lemma** *fst\_image\_mset\_times\_mset [simp]*:  
 $\text{image\_mset } \text{fst } (A \times \# B) = (\text{if } B = \{\#\} \text{ then } \{\#\} \text{ else } \text{repeat\_mset } (\text{size } B) A)$   
**by** (*induct B*) (*auto simp: Times\_mset\_single\_right ac\_simps Times\_insert\_left*)

**lemma** *snd\_image\_mset\_times\_mset [simp]*:  
 $\text{image\_mset } \text{snd } (A \times \# B) = (\text{if } A = \{\#\} \text{ then } \{\#\} \text{ else } \text{repeat\_mset } (\text{size } A) B)$   
**by** (*induct B*) (*auto simp add: Times\_mset\_single\_right Times\_insert\_left image\_mset\_const\_eq*)

**lemma** *product\_swap\_mset*:  $\text{image\_mset } \text{prod.swap } (A \times \# B) = B \times \# A$   
**by** (*induction A*) (*auto simp add: Times\_mset\_single\_left Times\_mset\_single\_right Times\_insert\_right Times\_insert\_left*)

**context**  
**begin**

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

**lemma** *member\_product\_mset*:  $x \in \# \text{product\_mset } A B \longleftrightarrow x \in \# A \times \# B$   
**by** (*simp add: Multiset\_More.product\_mset\_def*)

**end**

**lemma** *count\_Sigma\_mset\_abs\_def*:  $\text{count } (\text{Sigma\_mset } A B) = (\lambda(a, b) \Rightarrow \text{count } A a * \text{count } (B a) b)$   
**by** (*auto simp: fun\_eq\_iff count\_Sigma\_mset*)

**lemma** *Times\_mset\_image\_mset1*:  $\text{image\_mset } f A \times \# B = \text{image\_mset } (\lambda(a, b). (f a, b)) (A \times \# B)$   
**by** (*induct B*) (*auto simp: Times\_insert\_left*)

**lemma** *Times\_mset\_image\_mset2*:  $A \times \# \text{image\_mset } f B = \text{image\_mset } (\lambda(a, b). (a, f b)) (A \times \# B)$   
**by** (*induct A*) (*auto simp: Times\_insert\_right*)

**lemma** *sum\_le\_singleton*:  $A \subseteq \{x\} \Longrightarrow \text{sum } f A = (\text{if } x \in A \text{ then } f x \text{ else } 0)$   
**by** (*auto simp: subset\_singleton\_iff elim: finite\_subset*)

**lemma** *Times\_mset\_assoc*:  $(A \times \# B) \times \# C = \text{image\_mset } (\lambda(a, b, c). ((a, b), c)) (A \times \# B \times \# C)$   
**by** (*auto simp: multiset\_eq\_iff count\_Sigma\_mset count\_image\_mset vimage\_def Times\_mset\_Times\_Int\_commute count\_eq\_zero\_iff intro!: trans[OF \_ sym[OF sum\_le\_singleton[of \_ (\_, \_, \_)]]*)

cong: sum.cong if\_cong)

## 2.11 Transfer Rules

**lemma** *plus\_multiset\_transfer*[transfer\_rule]:  
 (rel\_fun (rel\_mset R) (rel\_fun (rel\_mset R) (rel\_mset R))) (+) (+)  
 by (unfold rel\_fun\_def rel\_mset\_def)  
 (force dest: list\_all2\_appendI intro: exI[of \_ \_ @ \_] conjI[rotated])

**lemma** *minus\_multiset\_transfer*[transfer\_rule]:  
 assumes [transfer\_rule]: bi\_unique R  
 shows (rel\_fun (rel\_mset R) (rel\_fun (rel\_mset R) (rel\_mset R))) (-) (-)  
**proof** (unfold rel\_fun\_def rel\_mset\_def, safe)  
 fix xs ys xs' ys'  
 assume [transfer\_rule]: list\_all2 R xs ys list\_all2 R xs' ys'  
 have list\_all2 R (fold remove1 xs' xs) (fold remove1 ys' ys)  
 by transfer\_prover  
 moreover have mset (fold remove1 xs' xs) = mset xs - mset xs'  
 by (induct xs' arbitrary: xs) auto  
 moreover have mset (fold remove1 ys' ys) = mset ys - mset ys'  
 by (induct ys' arbitrary: ys) auto  
 ultimately show  $\exists xs'' ys''$ .  
 mset xs'' = mset xs - mset xs'  $\wedge$  mset ys'' = mset ys - mset ys'  $\wedge$  list\_all2 R xs'' ys''  
 by blast  
**qed**

**declare** rel\_mset\_Zero[transfer\_rule]

**lemma** *count\_transfer*[transfer\_rule]:  
 assumes bi\_unique R  
 shows (rel\_fun (rel\_mset R) (rel\_fun R (=))) count count  
**unfolding** rel\_fun\_def rel\_mset\_def **proof** safe  
 fix x y xs ys  
 assume list\_all2 R xs ys R x y  
 then show count (mset xs) x = count (mset ys) y  
**proof** (induct xs ys rule: list.rel\_induct)  
 case (Cons x' xs y' ys)  
 then show ?case  
 using assms **unfolding** bi\_unique\_alt\_def2 **by** (auto simp: rel\_fun\_def)  
**qed** simp  
**qed**

**lemma** *subseq\_multiset\_transfer*[transfer\_rule]:  
 assumes [transfer\_rule]: bi\_unique R right\_total R  
 shows (rel\_fun (rel\_mset R) (rel\_fun (rel\_mset R) (=)))  
 ( $\lambda M N$ . filter\_mset (Domainp R) M  $\subseteq$ # filter\_mset (Domainp R) N) ( $\subseteq$ #)  
**proof** -  
 have count\_filter\_mset\_less:  
 ( $\forall a$ . count (filter\_mset (Domainp R) M) a  $\leq$  count (filter\_mset (Domainp R) N) a)  $\longleftrightarrow$   
 ( $\forall a \in \{x. \text{Domainp } R \ x\}$ . count M a  $\leq$  count N a) **for** M **and** N **by** auto  
 show ?thesis **unfolding** subseq\_mset\_def count\_filter\_mset\_less  
 by transfer\_prover  
**qed**

**lemma** *sum\_mset\_transfer*[transfer\_rule]:  
 R 0 0  $\implies$  rel\_fun R (rel\_fun R R) (+) (+)  $\implies$  (rel\_fun (rel\_mset R) R) sum\_mset sum\_mset  
 using sum\_list\_transfer[of R] **unfolding** rel\_fun\_def rel\_mset\_def **by** auto

**lemma** *Sigma\_mset\_transfer*[transfer\_rule]:  
 (rel\_fun (rel\_mset R) (rel\_fun (rel\_fun R (rel\_mset S)) (rel\_mset (rel\_prod R S))))  
 Sigma\_mset Sigma\_mset  
**by** (unfold Sigma\_mset\_def) transfer\_prover

## 2.12 Even More about Multisets

### 2.12.1 Multisets and Functions

**lemma** *range\_image\_mset*:  
 **assumes**  $set\_mset\ Ds \subseteq range\ f$   
 **shows**  $Ds \in range\ (image\_mset\ f)$   
**proof** –  
 **have**  $\forall D. D \in \# Ds \longrightarrow (\exists C. f\ C = D)$   
 **using** *assms* **by** *blast*  
 **then obtain**  $f\_i$  **where**  
  $f\_p: \forall D. D \in \# Ds \longrightarrow (f\ (f\_i\ D) = D)$   
 **by** *metis*  
 **define**  $Cs$  **where**  
  $Cs \equiv image\_mset\ f\_i\ Ds$   
 **from**  $f\_p$   $Cs\_def$  **have**  $image\_mset\ f\ Cs = Ds$   
 **by** *auto*  
 **then show** *?thesis*  
 **by** *blast*  
**qed**

### 2.12.2 Multisets and Lists

**lemma** *length\_sorted\_list\_of\_multiset[simp]*:  $length\ (sorted\_list\_of\_multiset\ A) = size\ A$   
 **by** (*metis mset\_sorted\_list\_of\_multiset size\_mset*)

**definition** *list\_of\_mset* :: 'a multiset  $\Rightarrow$  'a list **where**  
  $list\_of\_mset\ m = (SOME\ l. m = mset\ l)$

**lemma** *list\_of\_mset\_exi*:  $\exists l. m = mset\ l$   
 **using** *ex\_mset* **by** *metis*

**lemma** *mset\_list\_of\_mset[simp]*:  $mset\ (list\_of\_mset\ m) = m$   
 **by** (*metis (mono\_tags, lifting) ex\_mset list\_of\_mset\_def someI\_ex*)

**lemma** *length\_list\_of\_mset[simp]*:  $length\ (list\_of\_mset\ A) = size\ A$   
 **unfolding** *list\_of\_mset\_def* **by** (*metis (mono\_tags) ex\_mset size\_mset someI\_ex*)

**lemma** *range\_mset\_map*:  
 **assumes**  $set\_mset\ Ds \subseteq range\ f$   
 **shows**  $Ds \in range\ (\lambda Cl. mset\ (map\ f\ Cl))$   
**proof** –  
 **have**  $Ds \in range\ (image\_mset\ f)$   
 **by** (*simp add: assms range\_image\_mset*)  
 **then obtain**  $Cs$  **where**  $Cs\_p: image\_mset\ f\ Cs = Ds$   
 **by** *auto*  
 **define**  $Cl$  **where**  $Cl = list\_of\_mset\ Cs$   
 **then have**  $mset\ Cl = Cs$   
 **by** *auto*  
 **then have**  $image\_mset\ f\ (mset\ Cl) = Ds$   
 **using**  $Cs\_p$  **by** *auto*  
 **then have**  $mset\ (map\ f\ Cl) = Ds$   
 **by** *auto*  
 **then show** *?thesis*  
 **by** *auto*  
**qed**

**lemma** *list\_of\_mset\_empty[iff]*:  $list\_of\_mset\ m = [] \longleftrightarrow m = \{\#\}$   
 **by** (*metis (mono\_tags, lifting) ex\_mset list\_of\_mset\_def mset\_zero\_iff\_right someI\_ex*)

**lemma** *in\_mset\_conv\_nth*:  $(x \in \# mset\ xs) = (\exists i < length\ xs. xs\ !\ i = x)$   
 **by** (*auto simp: in\_set\_conv\_nth*)

**lemma** *in\_mset\_sum\_list*:  
 **assumes**  $L \in \# LL$



**assumes**  $LL \in \text{set } Ci$   
**shows**  $L \in\# \text{sum\_list } Ci$   
**using** *assms* **by** (*induction*  $Ci$ ) *auto*

**lemma** *in\_mset\_sum\_list2*:  
**assumes**  $L \in\# \text{sum\_list } Ci$   
**obtains**  $LL$  **where**  
 $LL \in \text{set } Ci$   
 $L \in\# LL$   
**using** *assms* **by** (*induction*  $Ci$ ) *auto*

**lemma** *in\_mset\_sum\_list\_iff*:  $a \in\# \text{sum\_list } \mathcal{A} \iff (\exists A \in \text{set } \mathcal{A}. a \in\# A)$   
**by** (*metis* *in\_mset\_sum\_list* *in\_mset\_sum\_list2*)

**lemma** *subteq\_list\_Union\_mset*:  
**assumes**  $\text{length } Ci = n$   
**assumes**  $\text{length } CAi = n$   
**assumes**  $\forall i < n. Ci ! i \subseteq\# CAi ! i$   
**shows**  $\sum\# (\text{mset } Ci) \subseteq\# \sum\# (\text{mset } CAi)$   
**using** *assms* **proof** (*induction*  $n$  *arbitrary*:  $Ci$   $CAi$ )  
**case** 0  
**then show** ?*case* **by** *auto*  
**next**  
**case** (*Suc*  $n$ )  
**from** *Suc* **have**  $\forall i < n. \text{tl } Ci ! i \subseteq\# \text{tl } CAi ! i$   
**by** (*simp* *add*: *nth\_tl*)  
**hence**  $\sum\# (\text{mset } (\text{tl } Ci)) \subseteq\# \sum\# (\text{mset } (\text{tl } CAi))$  **using** *Suc* **by** *auto*  
**moreover**  
**have**  $\text{hd } Ci \subseteq\# \text{hd } CAi$  **using** *Suc*  
**by** (*metis* *hd\_conv\_nth* *length\_greater\_0\_conv* *zero\_less\_Suc*)  
**ultimately**  
**show**  $\sum\# (\text{mset } Ci) \subseteq\# \sum\# (\text{mset } CAi)$   
**using** *Suc* **by** (*cases*  $Ci$ ; *cases*  $CAi$ ) (*auto* *intro*: *subset\_mset.add\_mono*)  
**qed**

### 2.12.3 More on Multisets and Functions

**lemma** *subteq\_mset\_size\_eq*:  $X \subseteq\# Y \implies \text{size } Y = \text{size } X \implies X = Y$   
**using** *mset\_subset\_size* *subset\_mset\_def* **by** *fastforce*

**lemma** *image\_mset\_of\_subset\_list*:  
**assumes**  $\text{image\_mset } \eta \ C' = \text{mset } lC$   
**shows**  $\exists qC'. \text{map } \eta \ qC' = lC \wedge \text{mset } qC' = C'$   
**using** *assms* **apply** (*induction*  $lC$  *arbitrary*:  $C'$ )  
**subgoal** **by** *simp*  
**subgoal** **by** (*fastforce* *dest*!: *msed\_map\_invR* *intro*: *exI*[*of* \_  $\langle$  \_  $\#$  \_])  
**done**

**lemma** *image\_mset\_of\_subset*:  
**assumes**  $A \subseteq\# \text{image\_mset } \eta \ C'$   
**shows**  $\exists A'. \text{image\_mset } \eta \ A' = A \wedge A' \subseteq\# C'$   
**proof** –  
**define**  $C$  **where**  $C = \text{image\_mset } \eta \ C'$

**define**  $lA$  **where**  $lA = \text{list\_of\_mset } A$   
**define**  $lD$  **where**  $lD = \text{list\_of\_mset } (C - A)$   
**define**  $lC$  **where**  $lC = lA @ lD$

**have**  $\text{mset } lC = C$   
**using**  $C\_def$  *assms* **unfolding**  $lD\_def$   $lC\_def$   $lA\_def$  **by** *auto*  
**then** **have**  $\exists qC'. \text{map } \eta \ qC' = lC \wedge \text{mset } qC' = C'$   
**using** *assms* *image\_mset\_of\_subset\_list* **unfolding**  $C\_def$  **by** *metis*  
**then** **obtain**  $qC'$  **where**  $qC'\_p: \text{map } \eta \ qC' = lC \wedge \text{mset } qC' = C'$

```

  by auto
let ?lA' = take (length lA) qC'
have m: map η ?lA' = lA
  using qC'_p lC_def
  by (metis append_eq_conv_conj take_map)
let ?A' = mset ?lA'

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

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

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

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

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

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

```

#### 2.12.4 More on Multiset Order

```

lemma less_multiset_doubletons:
  assumes
    y < t ∨ y < s
    x < t ∨ x < s
  shows
    {#y, x#} < {#t, s#}
  unfolding less_multisetDM
proof (intro exI)
  let ?X = {#t, s#}
  let ?Y = {#y, x#}
  show ?X ≠ {#} ∧ ?X ⊆# {#t, s#} ∧ {#y, x#} = {#t, s#} - ?X + ?Y
    ∧ (∀ k. k ∈# ?Y → (∃ a. a ∈# ?X ∧ k < a))
  using add_eq_conv_diff assms by auto

```

qed

end

### 3 Signed (Finite) Multisets

**theory** *Signed\_Multiset*

**imports** *Multiset\_More*

**abbrevs**

!z = z

**begin**

#### 3.1 Definition of Signed Multisets

**definition** *equiv\_zmset* :: 'a multiset  $\times$  'a multiset  $\Rightarrow$  'a multiset  $\times$  'a multiset  $\Rightarrow$  bool **where**  
*equiv\_zmset* =  $(\lambda(Mp, Mn) (Np, Nn). Mp + Nn = Np + Mn)$

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

**by** (rule *equivI*, *simp\_all* add: *equiv\_zmset\_def reflp\_def symp\_def transp\_def*)  
(metis *multi\_union\_self\_other\_eq union\_lcomm*)

#### 3.2 Basic Operations on Signed Multisets

**instantiation** *zmultiset* :: (type) *cancel\_comm\_monoid\_add*  
**begin**

**lift-definition** *zero\_zmultiset* :: 'a *zmultiset* **is** ({#}, {#}) .

**abbreviation** *empty\_zmset* :: 'a *zmultiset* ({#}\_z) **where**  
*empty\_zmset*  $\equiv$  0

**lift-definition** *minus\_zmultiset* :: 'a *zmultiset*  $\Rightarrow$  'a *zmultiset*  $\Rightarrow$  'a *zmultiset* **is**  
 $\lambda(Mp, Mn) (Np, Nn). (Mp + Nn, Mn + Np)$   
**by** (auto *simp*: *equiv\_zmset\_def union\_commute union\_lcomm*)

**lift-definition** *plus\_zmultiset* :: 'a *zmultiset*  $\Rightarrow$  'a *zmultiset*  $\Rightarrow$  'a *zmultiset* **is**  
 $\lambda(Mp, Mn) (Np, Nn). (Mp + Np, Mn + Nn)$   
**by** (auto *simp*: *equiv\_zmset\_def union\_commute union\_lcomm*)

**instance**

**by** (*intro\_classes*; *transfer*) (auto *simp*: *equiv\_zmset\_def*)

end

**instantiation** *zmultiset* :: (type) *group\_add*  
**begin**

**lift-definition** *uminus\_zmultiset* :: 'a *zmultiset*  $\Rightarrow$  'a *zmultiset* **is**  $\lambda(Mp, Mn). (Mn, Mp)$   
**by** (auto *simp*: *equiv\_zmset\_def add commute*)

**instance**

**by** (*intro\_classes*; *transfer*) (auto *simp*: *equiv\_zmset\_def*)

end

**lift-definition** *zcount* :: 'a *zmultiset*  $\Rightarrow$  'a  $\Rightarrow$  int **is**

$\lambda(Mp, Mn) x. \text{int } (\text{count } Mp \ x) - \text{int } (\text{count } Mn \ x)$

**by** (auto *simp* del: *of\_nat\_add simp: equiv\_zmset\_def fun\_eq\_iff multiset\_eq\_iff diff\_eq\_eq*  
*diff\_add\_eq\_eq diff\_eq\_eq of\_nat\_add[symmetric]*)

**lemma** *zcount\_inject*: *zcount* M = *zcount* N  $\longleftrightarrow$  M = N

**by** *transfer* (auto *simp* del: *of\_nat\_add simp: equiv\_zmset\_def fun\_eq\_iff multiset\_eq\_iff*  
*diff\_eq\_eq diff\_add\_eq\_eq diff\_eq\_eq of\_nat\_add[symmetric]*)

**lemma** *zmultiset\_eq\_iff*:  $M = N \iff (\forall a. \text{zcount } M \ a = \text{zcount } N \ a)$   
**by** (*simp only*: *zcount\_inject[symmetric] fun\_eq\_iff*)

**lemma** *zmultiset\_eqI*:  $(\bigwedge x. \text{zcount } A \ x = \text{zcount } B \ x) \implies A = B$   
**using** *zmultiset\_eq\_iff* **by** *auto*

**lemma** *zcount\_uminus[simp]*:  $\text{zcount } (- A) \ x = - \text{zcount } A \ x$   
**by** *transfer auto*

**lift-definition** *add\_zmset* ::  $'a \Rightarrow 'a \text{ zmultiset} \Rightarrow 'a \text{ zmultiset}$  **is**  
 $\lambda x \ (Mp, Mn). \ (\text{add\_mset } x \ Mp, Mn)$   
**by** (*auto simp: equiv\_zmset\_def*)

**syntax**

*\_zmultiset* ::  $\text{args} \Rightarrow 'a \text{ zmultiset} \ (\{\#(\_) \# \}_z)$

**translations**

$\{\#x, xs\# \}_z == \text{CONST } \text{add\_zmset } x \ \{\#xs\# \}_z$

$\{\#x\# \}_z == \text{CONST } \text{add\_zmset } x \ \{\# \}_z$

**lemma** *zcount\_empty[simp]*:  $\text{zcount } \{\# \}_z \ a = 0$   
**by** *transfer auto*

**lemma** *zcount\_add\_zmset[simp]*:  
 $\text{zcount } (\text{add\_zmset } b \ A) \ a = (\text{if } b = a \ \text{then } \text{zcount } A \ a + 1 \ \text{else } \text{zcount } A \ a)$   
**by** *transfer auto*

**lemma** *zcount\_single*:  $\text{zcount } \{\#b\# \}_z \ a = (\text{if } b = a \ \text{then } 1 \ \text{else } 0)$   
**by** *simp*

**lemma** *add\_add\_same\_iff\_zmset[simp]*:  $\text{add\_zmset } a \ A = \text{add\_zmset } a \ B \iff A = B$   
**by** (*auto simp: zmultiset\_eq\_iff*)

**lemma** *add\_zmset\_commute*:  $\text{add\_zmset } x \ (\text{add\_zmset } y \ M) = \text{add\_zmset } y \ (\text{add\_zmset } x \ M)$   
**by** (*auto simp: zmultiset\_eq\_iff*)

**lemma**

*singleton\_ne\_empty\_zmset[simp]*:  $\{\#x\# \}_z \neq \{\# \}_z$  **and**

*empty\_ne\_singleton\_zmset[simp]*:  $\{\# \}_z \neq \{\#x\# \}_z$

**by** (*auto dest!: arg\_cong2[of \_ \_ x \_ zcount]*)

**lemma**

*singleton\_ne\_uminus\_singleton\_zmset[simp]*:  $\{\#x\# \}_z \neq - \{\#y\# \}_z$  **and**

*uminus\_singleton\_ne\_singleton\_zmset[simp]*:  $- \{\#x\# \}_z \neq \{\#y\# \}_z$

**by** (*auto dest!: arg\_cong2[of \_ \_ x x zcount] split: if\_splits*)

### 3.2.1 Conversion to Set and Membership

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

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

**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* ::  $'a \Rightarrow 'a \text{ zmultiset} \Rightarrow \text{bool}$  **where**  
 $\text{not\_elem\_zmset } a \ M \equiv a \notin \text{set\_zmset } M$

**notation**

$not\_elem\_zmset$  ( $(\notin\#_z)$ ) **and**  
 $not\_elem\_zmset$  ( $(\_/\notin\#_z \_)$  [51, 51] 50)

**notation (ASCII)**

$not\_elem\_zmset$  ( $(\sim\#_z)$ ) **and**  
 $not\_elem\_zmset$  ( $(\_/\sim\#_z \_)$  [51, 51] 50)

**context****begin**

**qualified abbreviation**  $Ball :: 'a\ zmset \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$  **where**  
 $Ball\ M \equiv Set.Ball\ (set\_zmset\ M)$

**qualified abbreviation**  $Bex :: 'a\ zmset \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$  **where**  
 $Bex\ M \equiv Set.Bex\ (set\_zmset\ M)$

**end****syntax**

$\_ZMBall :: ptrn \Rightarrow 'a\ set \Rightarrow bool \Rightarrow bool$  ( $(\exists\forall\_ \in\#_z \_/\ \_)$  [0, 0, 10] 10)  
 $\_ZMBex :: ptrn \Rightarrow 'a\ set \Rightarrow bool \Rightarrow bool$  ( $(\exists\exists\_ \in\#_z \_/\ \_)$  [0, 0, 10] 10)

**syntax (ASCII)**

$\_ZMBall :: ptrn \Rightarrow 'a\ set \Rightarrow bool \Rightarrow bool$  ( $(\exists\forall\_ \#_z \_/\ \_)$  [0, 0, 10] 10)  
 $\_ZMBex :: ptrn \Rightarrow 'a\ set \Rightarrow bool \Rightarrow bool$  ( $(\exists\exists\_ \#_z \_/\ \_)$  [0, 0, 10] 10)

**translations**

$\forall x \in\#_z A. P \Rightarrow CONST\ Signed\_Multiset.Ball\ A\ (\lambda x. P)$   
 $\exists x \in\#_z A. P \Rightarrow CONST\ Signed\_Multiset.Bex\ A\ (\lambda x. P)$

**lemma**  $zcount\_eq\_zero\_iff: zcount\ M\ x = 0 \longleftrightarrow x \notin\#_z\ M$   
**by** (*auto simp add: set\\_zmset\\_def*)

**lemma**  $not\_in\_iff\_zmset: x \notin\#_z\ M \longleftrightarrow zcount\ M\ x = 0$   
**by** (*auto simp add: zcount\\_eq\\_zero\\_iff*)

**lemma**  $zcount\_ne\_zero\_iff[simp]: zcount\ M\ x \neq 0 \longleftrightarrow x \in\#_z\ M$   
**by** (*auto simp add: set\\_zmset\\_def*)

**lemma**  $zcount\_inI:$

**assumes**  $zcount\ M\ x = 0 \implies False$

**shows**  $x \in\#_z\ M$

**proof** (*rule ccontr*)

**assume**  $x \notin\#_z\ M$

**with assms show**  $False$  **by** (*simp add: not\\_in\\_iff\\_zmset*)

**qed**

**lemma**  $set\_zmset\_empty[simp]: set\_zmset\ \{\#\}_z = \{\}$   
**by** (*simp add: set\\_zmset\\_def*)

**lemma**  $set\_zmset\_single: set\_zmset\ \{\#b\#}_z = \{b\}$   
**by** (*simp add: set\\_zmset\\_def*)

**lemma**  $set\_zmset\_eq\_empty\_iff[simp]: set\_zmset\ M = \{\} \longleftrightarrow M = \{\#\}_z$   
**by** (*auto simp add: zmset\\_eq\\_iff zcount\\_eq\\_zero\\_iff*)

**lemma**  $finite\_count\_ne: finite\ \{x. count\ M\ x \neq count\ N\ x\}$

**proof** –

**have**  $\{x. count\ M\ x \neq count\ N\ x\} \subseteq set\_mset\ M \cup set\_mset\ N$

**by** (*auto simp: not\\_in\\_iff*)

**moreover have**  $finite\ (set\_mset\ M \cup set\_mset\ N)$

by (rule finite\_UnI[OF finite\_set\_mset finite\_set\_mset])  
 ultimately show ?thesis  
 by (rule finite\_subset)  
 qed

**lemma** finite\_set\_zmset[iff]: finite (set\_zmset M)  
 unfolding set\_zmset\_def by transfer (auto intro: finite\_count\_ne)

**lemma** zmset\_nonemptyE[elim]:  
 assumes  $A \neq \{\#\}_z$   
 obtains  $x$  where  $x \in \#_z A$   
**proof** –  
 have  $\exists x. x \in \#_z A$   
 by (rule ccontr) (insert assms, auto)  
 with that show ?thesis  
 by blast  
 qed

### 3.2.2 Union

**lemma** zcount\_union[simp]: zcount (M + N) a = zcount M a + zcount N a  
 by transfer auto

**lemma** union\_add\_left\_zmset[simp]: add\_zmset a A + B = add\_zmset a (A + B)  
 by (auto simp: zmset\_eq\_iff)

**lemma** union\_zmset\_add\_zmset\_right[simp]: A + add\_zmset a B = add\_zmset a (A + B)  
 by (auto simp: zmset\_eq\_iff)

**lemma** add\_zmset\_add\_single:  $\langle \text{add\_zmset } a \ A = A + \{\#a\# \}_z \rangle$   
 by (subst union\_zmset\_add\_zmset\_right, subst add.comm\_neutral) (rule refl)

### 3.2.3 Difference

**lemma** zcount\_diff[simp]: zcount (M - N) a = zcount M a - zcount N a  
 by transfer auto

**lemma** add\_zmset\_diff\_bothersides:  $\langle \text{add\_zmset } a \ M - \text{add\_zmset } a \ A = M - A \rangle$   
 by (auto simp: zmset\_eq\_iff)

**lemma** in\_diff\_zcount:  $a \in \#_z M - N \iff zcount \ N \ a \neq zcount \ M \ a$   
 by (fastforce simp: set\_zmset\_def)

**lemma** diff\_add\_zmset:  
 fixes M N Q :: 'a zmset  
 shows  $M - (N + Q) = M - N - Q$   
 by (rule sym) (fact diff\_diff\_add)

**lemma** insert\_Diff\_zmset[simp]:  $\text{add\_zmset } x \ (M - \{\#x\# \}_z) = M$   
 by (clarsimp simp: zmset\_eq\_iff)

**lemma** diff\_union\_swap\_zmset:  $\text{add\_zmset } b \ (M - \{\#a\# \}_z) = \text{add\_zmset } b \ M - \{\#a\# \}_z$   
 by (auto simp add: zmset\_eq\_iff)

**lemma** diff\_add\_zmset\_swap[simp]:  $\text{add\_zmset } b \ M - A = \text{add\_zmset } b \ (M - A)$   
 by (auto simp add: zmset\_eq\_iff)

**lemma** diff\_diff\_add\_zmset[simp]:  $(M :: 'a zmset) - N - P = M - (N + P)$   
 by (rule diff\_diff\_add)

**lemma** zmset\_add[elim?]:  
 obtains B where  $A = \text{add\_zmset } a \ B$   
**proof** –  
 have  $A = \text{add\_zmset } a \ (A - \{\#a\# \}_z)$

by *simp*  
with *that show thesis .*  
qed

### 3.2.4 Equality of Signed Multisets

**lemma** *single\_eq\_single\_zmset*[*simp*]:  $\{\#a\# \}_z = \{\#b\# \}_z \longleftrightarrow a = b$   
by (*auto simp add: zmset\_eq\_iff*)

**lemma** *multi\_self\_add\_other\_not\_self\_zmset*[*simp*]:  $M = \text{add\_zmset } x \ M \longleftrightarrow \text{False}$   
by (*auto simp add: zmset\_eq\_iff*)

**lemma** *add\_zmset\_remove\_trivial*:  $\langle \text{add\_zmset } x \ M - \{\#x\# \}_z = M \rangle$   
by *simp*

**lemma** *diff\_single\_eq\_union\_zmset*:  $M - \{\#x\# \}_z = N \longleftrightarrow M = \text{add\_zmset } x \ N$   
by *auto*

**lemma** *union\_single\_eq\_diff\_zmset*:  $\text{add\_zmset } x \ M = N \implies M = N - \{\#x\# \}_z$   
**unfolding** *add\_zmset\_add\_single*[*of \_ M*] by (*fact add\_implies\_diff*)

**lemma** *add\_zmset\_eq\_conv\_diff*:  
 $\text{add\_zmset } a \ M = \text{add\_zmset } b \ N \longleftrightarrow$   
 $M = N \wedge a = b \vee M = \text{add\_zmset } b \ (N - \{\#a\# \}_z) \wedge N = \text{add\_zmset } a \ (M - \{\#b\# \}_z)$   
by (*simp add: zmset\_eq\_iff*) *fastforce*

**lemma** *add\_zmset\_eq\_conv\_ex*:  
 $(\text{add\_zmset } a \ M = \text{add\_zmset } b \ N) =$   
 $(M = N \wedge a = b \vee (\exists K. M = \text{add\_zmset } b \ K \wedge N = \text{add\_zmset } a \ K))$   
by (*auto simp add: add\_zmset\_eq\_conv\_diff*)

**lemma** *multi\_member\_split*:  $\exists A. M = \text{add\_zmset } x \ A$   
by (*rule exI*[*where*  $x = M - \{\#x\# \}_z$ ]) *simp*

## 3.3 Conversions from and to Multisets

**lift-definition** *zmset\_of* ::  $'a \ \text{multiset} \Rightarrow 'a \ \text{zmset}$  **is**  $\lambda f. (\text{Abs\_multiset } f, \{\#\})$  .

**lemma** *zmset\_of\_inject*[*simp*]:  $\text{zmset\_of } M = \text{zmset\_of } N \longleftrightarrow M = N$   
by (*simp add: zmset\_of\_def, transfer, auto simp: equiv\_zmset\_def*)

**lemma** *zmset\_of\_empty*[*simp*]:  $\text{zmset\_of } \{\#\} = \{\#\}_z$   
by (*simp add: zmset\_of\_def zero\_zmultiset\_def*)

**lemma** *zmset\_of\_add\_mset*[*simp*]:  $\text{zmset\_of } (\text{add\_mset } x \ M) = \text{add\_zmset } x \ (\text{zmset\_of } M)$   
by *transfer* (*auto simp: equiv\_zmset\_def add\_mset\_def cong: if\_cong*)

**lemma** *zcount\_of\_mset*[*simp*]:  $\text{zcount } (\text{zmset\_of } M) \ x = \text{int } (\text{count } M \ x)$   
by (*induct M*) *auto*

**lemma** *zmset\_of\_plus*:  $\text{zmset\_of } (M + N) = \text{zmset\_of } M + \text{zmset\_of } N$   
by (*transfer, auto simp: equiv\_zmset\_def eq\_onp\_same\_args plus\_multiset.abs\_eq*) $+$

**lift-definition** *mset\_pos* ::  $'a \ \text{zmset} \Rightarrow 'a \ \text{multiset}$  **is**  $\lambda (Mp, Mn). \text{count } (Mp - Mn)$   
by (*clarsimp simp: equiv\_zmset\_def intro!: arg\_cong*[*of \_ \_ count*])  
(*metis add.commute add\_diff\_cancel\_right*)

**lift-definition** *mset\_neg* ::  $'a \ \text{zmset} \Rightarrow 'a \ \text{multiset}$  **is**  $\lambda (Mp, Mn). \text{count } (Mn - Mp)$   
by (*clarsimp simp: equiv\_zmset\_def intro!: arg\_cong*[*of \_ \_ count*])  
(*metis add.commute add\_diff\_cancel\_right*)

**lemma**  
*zmset\_of\_inverse*[*simp*]:  $\text{mset\_pos } (\text{zmset\_of } M) = M$  **and**  
*minus\_zmset\_of\_inverse*[*simp*]:  $\text{mset\_neg } (- \text{zmset\_of } M) = M$

by (transfer, simp)+

**lemma** *neg\_zmset\_pos*[simp]:  $mset\_neg (zmset\_of M) = \{\#\}$   
by (rule *zmset\_of\_inject*[THEN *iffD1*], simp, transfer, auto simp: *equiv\_zmset\_def*)+

**lemma**  
*count\_mset\_pos*[simp]:  $count (mset\_pos M) x = nat (zcount M x)$  **and**  
*count\_mset\_neg*[simp]:  $count (mset\_neg M) x = nat (- zcount M x)$   
by (transfer; auto)+

**lemma**  
*mset\_pos\_empty*[simp]:  $mset\_pos \{\#\}_z = \{\#\}$  **and**  
*mset\_neg\_empty*[simp]:  $mset\_neg \{\#\}_z = \{\#\}$   
by (rule *multiset\_eqI*, simp)+

**lemma**  
*mset\_pos\_singleton*[simp]:  $mset\_pos \{\#x\}_z = \{\#x\}$  **and**  
*mset\_neg\_singleton*[simp]:  $mset\_neg \{\#x\}_z = \{\#\}$   
by (rule *multiset\_eqI*, simp)+

**lemma**  
*mset\_pos\_neg\_partition*:  $M = zmset\_of (mset\_pos M) - zmset\_of (mset\_neg M)$  **and**  
*mset\_pos\_as\_neg*:  $zmset\_of (mset\_pos M) = zmset\_of (mset\_neg M) + M$  **and**  
*mset\_neg\_as\_pos*:  $zmset\_of (mset\_neg M) = zmset\_of (mset\_pos M) - M$   
by (rule *zmultiset\_eqI*, simp)+

**lemma** *mset\_pos\_uminus*[simp]:  $mset\_pos (- A) = mset\_neg A$   
by (rule *multiset\_eqI*) simp

**lemma** *mset\_neg\_uminus*[simp]:  $mset\_neg (- A) = mset\_pos A$   
by (rule *multiset\_eqI*) simp

**lemma** *mset\_pos\_plus*[simp]:  
 $mset\_pos (A + B) = (mset\_pos A - mset\_neg B) + (mset\_pos B - mset\_neg A)$   
by (rule *multiset\_eqI*) simp

**lemma** *mset\_neg\_plus*[simp]:  
 $mset\_neg (A + B) = (mset\_neg A - mset\_pos B) + (mset\_neg B - mset\_pos A)$   
by (rule *multiset\_eqI*) simp

**lemma** *mset\_pos\_diff*[simp]:  
 $mset\_pos (A - B) = (mset\_pos A - mset\_pos B) + (mset\_neg B - mset\_neg A)$   
by (rule *mset\_pos\_plus*[of  $A - B$ , *simplified*])

**lemma** *mset\_neg\_diff*[simp]:  
 $mset\_neg (A - B) = (mset\_neg A - mset\_neg B) + (mset\_pos B - mset\_pos A)$   
by (rule *mset\_neg\_plus*[of  $A - B$ , *simplified*])

**lemma** *mset\_pos\_neg\_dual*:  
 $mset\_pos a + mset\_pos b + (mset\_neg a - mset\_pos b) + (mset\_neg b - mset\_pos a) =$   
 $mset\_neg a + mset\_neg b + (mset\_pos a - mset\_neg b) + (mset\_pos b - mset\_neg a)$   
using [[*linarith\_split\_limit* = 20]] by (rule *multiset\_eqI*) simp

**lemma** *decompose\_zmset\_of2*:  
**obtains**  $A B C$  **where**  
 $M = zmset\_of A + C$  **and**  
 $N = zmset\_of B + C$

**proof**  
**let**  $?A = zmset\_of (mset\_pos M + mset\_neg N)$   
**let**  $?B = zmset\_of (mset\_pos N + mset\_neg M)$   
**let**  $?C = - (zmset\_of (mset\_neg M) + zmset\_of (mset\_neg N))$

**show**  $M = ?A + ?C$



by (simp add: zmsset\_of\_plus mset\_pos\_neg\_partition)  
 show  $N = ?B + ?C$   
 by (simp add: zmsset\_of\_plus diff\_add\_zmsset mset\_pos\_neg\_partition)  
 qed

### 3.3.1 Pointwise Ordering Induced by $zcount$

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

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

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

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

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

**notation** (ASCII)  
 $subseteq\_zmsset$  (**infix**  $\subseteq\#_z$  50) **and**  
 $subset\_zmsset$  (**infix**  $\subset\#_z$  50) **and**  
 $supseteq\_zmsset$  (**infix**  $\supseteq\#_z$  50) **and**  
 $supset\_zmsset$  (**infix**  $\supset\#_z$  50)

**interpretation**  $subset\_zmsset$ :  $ordered\_ab\_semigroup\_add\_imp\_le$  (+) (-) ( $\subseteq\#_z$ ) ( $\subset\#_z$ )  
**by**  $unfold\_locales$  (auto simp add: subset\_zmsset\_def subseteq\_zmsset\_def zmultiset\_eq\_iff  
 intro: order\_trans antisym)

**interpretation**  $subset\_zmsset$ :  
 $ordered\_ab\_semigroup\_monoid\_add\_imp\_le$  (+) 0 (-) ( $\subseteq\#_z$ ) ( $\subset\#_z$ )  
**by**  $unfold\_locales$

**lemma**  $zmsset\_subset\_eqI$ :  $(\bigwedge a. zcount\ A\ a \leq zcount\ B\ a) \Longrightarrow A \subseteq\#_z B$   
**by** (simp add: subseteq\_zmsset\_def)

**lemma**  $zmsset\_subset\_eq\_zcount$ :  $A \subseteq\#_z B \Longrightarrow zcount\ A\ a \leq zcount\ B\ a$   
**by** (simp add: subseteq\_zmsset\_def)

**lemma**  $zmsset\_subset\_eq\_add\_zmsset\_cancel$ :  $\langle add\_zmsset\ a\ A \subseteq\#_z\ add\_zmsset\ a\ B \longleftrightarrow A \subseteq\#_z B \rangle$   
**unfolding**  $add\_zmsset\_add\_single$ [of  $\_ A$ ]  $add\_zmsset\_add\_single$ [of  $\_ B$ ]  
**by** (rule subset\_zmsset.add\_le\_cancel\_right)

**lemma**  $zmsset\_subset\_eq\_zmultiset\_union\_diff\_commute$ :  
 $A - B + C = A + C - B$  **for**  $A\ B\ C :: 'a\ zmultiset$   
**by** (simp add: add commute add\_diff\_eq)

**lemma**  $zmsset\_subset\_eq\_insertD$ :  $add\_zmsset\ x\ A \subseteq\#_z B \Longrightarrow A \subset\#_z B$   
**unfolding**  $subset\_zmsset\_def$   $subseqeq\_zmsset\_def$   
**by** (metis (no\_types) add commute add\_le\_same\_cancel2 zcount\_add\_zmsset dual\_order.trans le\_cases  
 le\_numeral\_extra(2))

**lemma**  $zmsset\_subset\_insertD$ :  $add\_zmsset\ x\ A \subset\#_z B \Longrightarrow A \subset\#_z B$   
**by** (rule zmsset\_subset\_eq\_insertD) (rule subset\_zmsset.less\_imp\_le)

**lemma**  $subset\_eq\_diff\_conv\_zmsset$ :  $A - C \subseteq\#_z B \longleftrightarrow A \subseteq\#_z B + C$

by (simp add: subseteq\_zmset\_def ordered\_ab\_group\_add\_class.diff\_le\_eq)

**lemma** *multi\_psub\_of\_add\_self\_zmset*[simp]:  $A \subset\#_z \text{add\_zmset } x \ A$   
 by (auto simp: subset\_zmset\_def subseteq\_zmset\_def)

**lemma** *multi\_psub\_self\_zmset*:  $A \subset\#_z A = \text{False}$   
 by simp

**lemma** *zmset\_subset\_add\_zmset*[simp]:  $\text{add\_zmset } x \ N \subset\#_z \text{add\_zmset } x \ M \iff N \subset\#_z M$   
**unfolding** *add\_zmset\_add\_single*[of \_ N] *add\_zmset\_add\_single*[of \_ M]  
 by (fact subset\_zmset.add\_less\_cancel\_right)

**lemma** *zmset\_of\_subseteq\_iff*[simp]:  $\text{zmset\_of } M \subseteq\#_z \text{zmset\_of } N \iff M \subseteq\# \ N$   
 by (simp add: subseteq\_zmset\_def subseteq\_mset\_def)

**lemma** *zmset\_of\_subset\_iff*[simp]:  $\text{zmset\_of } M \subset\#_z \text{zmset\_of } N \iff M \subset\# \ N$   
 by (simp add: subset\_zmset\_def subset\_mset\_def)

**lemma**  
*mset\_pos\_supset*:  $A \subseteq\#_z \text{zmset\_of } (\text{mset\_pos } A)$  **and**  
*mset\_neg\_supset*:  $- A \subseteq\#_z \text{zmset\_of } (\text{mset\_neg } A)$   
 by (auto intro: zmset\_subset\_eqI)

**lemma** *subset\_mset\_zmsetE*:  
**assumes**  $M \subset\#_z N$   
**obtains**  $A \ B \ C$  **where**  
 $M = \text{zmset\_of } A + C$  **and**  $N = \text{zmset\_of } B + C$  **and**  $A \subset\# \ B$   
 by (metis assms decompose\_zmset\_of2 subset\_zmset.add\_less\_cancel\_right zmset\_of\_subset\_iff)

**lemma** *subseteq\_mset\_zmsetE*:  
**assumes**  $M \subseteq\#_z N$   
**obtains**  $A \ B \ C$  **where**  
 $M = \text{zmset\_of } A + C$  **and**  $N = \text{zmset\_of } B + C$  **and**  $A \subseteq\# \ B$   
 by (metis assms add.commute add.right\_neutral subset\_mset.order\_refl subset\_mset\_def subset\_mset\_zmsetE subset\_zmset\_def zmset\_of\_empty)

### 3.3.2 Subset is an Order

**interpretation** *subset\_zmset*: *order* ( $\subseteq\#_z$ ) ( $\subset\#_z$ )  
 by *unfold\_locales*

## 3.4 Replicate and Repeat Operations

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

**lemma** *replicate\_zmset\_0*[simp]:  $\text{replicate\_zmset } 0 \ x = \{\#\}_z$   
**unfolding** *replicate\_zmset\_def* **by** *simp*

**lemma** *replicate\_zmset\_Suc*[simp]:  $\text{replicate\_zmset } (\text{Suc } n) \ x = \text{add\_zmset } x \ (\text{replicate\_zmset } n \ x)$   
**unfolding** *replicate\_zmset\_def* **by** (induct n) (auto intro: add.commute)

**lemma** *count\_replicate\_zmset*[simp]:  
 $\text{zcount } (\text{replicate\_zmset } n \ x) \ y = (\text{if } y = x \ \text{then } \text{of\_nat } n \ \text{else } 0)$   
**unfolding** *replicate\_zmset\_def* **by** (induct n) *auto*

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

**lemma** *count\_repeat\_zmset*[simp]:  $\text{zcount } (\text{repeat\_zmset } i \ A) \ a = \text{of\_nat } i * \text{zcount } A \ a$   
**by** (induct i) (auto simp: semiring\_normalization\_rules(3))

**lemma** *repeat\_zmset\_right*[simp]:  $\text{repeat\_zmset } a \ (\text{repeat\_zmset } b \ A) = \text{repeat\_zmset } (a * b) \ A$

by (auto simp: zmultiset\_eq\_iff left\_diff\_distrib')

**lemma** left\_diff\_repeat\_zmset\_distrib':

$\langle i \geq j \implies \text{repeat\_zmset } (i - j) \ u = \text{repeat\_zmset } i \ u - \text{repeat\_zmset } j \ u \rangle$

by (auto simp: zmultiset\_eq\_iff int\_distrib(3) of\_nat\_diff)

**lemma** left\_add\_mult\_distrib\_zmset:

$\text{repeat\_zmset } i \ u + (\text{repeat\_zmset } j \ u + k) = \text{repeat\_zmset } (i+j) \ u + k$

by (auto simp: zmultiset\_eq\_iff add\_mult\_distrib int\_distrib(1))

**lemma** repeat\_zmset\_distrib:  $\text{repeat\_zmset } (m + n) \ A = \text{repeat\_zmset } m \ A + \text{repeat\_zmset } n \ A$

by (auto simp: zmultiset\_eq\_iff Nat.add\_mult\_distrib int\_distrib(1))

**lemma** repeat\_zmset\_distrib2[simp]:

$\text{repeat\_zmset } n \ (A + B) = \text{repeat\_zmset } n \ A + \text{repeat\_zmset } n \ B$

by (auto simp: zmultiset\_eq\_iff add\_mult\_distrib2 int\_distrib(2))

**lemma** repeat\_zmset\_replicate\_zmset[simp]:  $\text{repeat\_zmset } n \ \{\#a\#_z = \text{replicate\_zmset } n \ a$

by (auto simp: zmultiset\_eq\_iff)

**lemma** repeat\_zmset\_distrib\_add\_zmset[simp]:

$\text{repeat\_zmset } n \ (\text{add\_zmset } a \ A) = \text{replicate\_zmset } n \ a + \text{repeat\_zmset } n \ A$

by (auto simp: zmultiset\_eq\_iff int\_distrib(2))

**lemma** repeat\_zmset\_empty[simp]:  $\text{repeat\_zmset } n \ \{\#\}_z = \{\#\}_z$

by (induct n) simp\_all

### 3.4.1 Filter (with Comprehension Syntax)

**lift-definition** filter\_zmset :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a zmultiset  $\Rightarrow$  'a zmultiset is

$\lambda P \ (Mp, Mn). \ (\text{filter\_mset } P \ Mp, \text{filter\_mset } P \ Mn)$

by (auto simp del: filter\_union\_mset simp: equiv\_zmset\_def filter\_union\_mset[symmetric])

**syntax** (ASCII)

$\_ZMCollect \ :: \ \text{pttrn} \ \Rightarrow \ 'a \ \text{zmultiset} \ \Rightarrow \ \text{bool} \ \Rightarrow \ 'a \ \text{zmultiset} \ ((1\{\#\_ : \#z \_ / \_ \#\}))$

**syntax**

$\_ZMCollect \ :: \ \text{pttrn} \ \Rightarrow \ 'a \ \text{zmultiset} \ \Rightarrow \ \text{bool} \ \Rightarrow \ 'a \ \text{zmultiset} \ ((1\{\#\_ \in \#z \_ / \_ \#\}))$

**translations**

$\{\#x \in \#z \ M. \ P\#\} \ == \ \text{CONST } \text{filter\_zmset} \ (\lambda x. \ P) \ M$

**lemma** count\_filter\_zmset[simp]:

$\text{zcount} \ (\text{filter\_zmset } P \ M) \ a = (\text{if } P \ a \ \text{then } \text{zcount } M \ a \ \text{else } 0)$

by transfer auto

**lemma** filter\_empty\_zmset[simp]:  $\text{filter\_zmset } P \ \{\#\}_z = \{\#\}_z$

by (rule zmultiset\_eqI) simp

**lemma** filter\_single\_zmset:  $\text{filter\_zmset } P \ \{\#x\#_z = (\text{if } P \ x \ \text{then } \{\#x\#_z \ \text{else } \{\#\}_z)$

by (rule zmultiset\_eqI) simp

**lemma** filter\_union\_zmset[simp]:  $\text{filter\_zmset } P \ (M + N) = \text{filter\_zmset } P \ M + \text{filter\_zmset } P \ N$

by (rule zmultiset\_eqI) simp

**lemma** filter\_diff\_zmset[simp]:  $\text{filter\_zmset } P \ (M - N) = \text{filter\_zmset } P \ M - \text{filter\_zmset } P \ N$

by (rule zmultiset\_eqI) simp

**lemma** filter\_add\_zmset[simp]:

$\text{filter\_zmset } P \ (\text{add\_zmset } x \ A) =$

$(\text{if } P \ x \ \text{then } \text{add\_zmset } x \ (\text{filter\_zmset } P \ A) \ \text{else } \text{filter\_zmset } P \ A)$

by (auto simp: zmultiset\_eq\_iff)

**lemma** zmultiset\_filter\_mono:

assumes  $A \subseteq_{\#z} B$

shows  $\text{filter\_zmset } f \ A \subseteq_{\#z} \text{filter\_zmset } f \ B$

using *assms* by (*simp add: subseteq\_zmset\_def*)

**lemma** *filter\_filter\_zmset*:  $\text{filter\_zmset } P (\text{filter\_zmset } Q M) = \{\#x \in \#_z M. Q x \wedge P x\# \}$   
 by (*auto simp: zm��et\_eq\_iff*)

**lemma**

*filter\_zm��et\_True*[*simp*]:  $\{\#y \in \#_z M. \text{True}\# \} = M$  **and**  
*filter\_zm��et\_False*[*simp*]:  $\{\#y \in \#_z M. \text{False}\# \} = \{\#\}_z$   
 by (*auto simp: zm��et\_eq\_iff*)

### 3.5 Uncategorized

**lemma** *multi\_drop\_mem\_not\_eq\_zm��et*:  $B - \{\#c\# \}_z \neq B$   
 by (*simp add: diff\_single\_eq\_union\_zm��et*)

**lemma** *zm��et\_partition*:  $M = \{\#x \in \#_z M. P x \#\} + \{\#x \in \#_z M. \neg P x\#\}$   
 by (*subst zm��et\_eq\_iff*) *auto*

### 3.6 Image

**definition** *image\_zm��et* :: ( $'a \Rightarrow 'b$ )  $\Rightarrow 'a$  *zm��et*  $\Rightarrow 'b$  *zm��et* **where**  
*image\_zm��et* *f* *M* =  
 $\text{zm��et\_of } (\text{fold\_m��et } (\text{add\_m��et } \circ f) \{\#\} (\text{m��et\_pos } M)) -$   
 $\text{zm��et\_of } (\text{fold\_m��et } (\text{add\_m��et } \circ f) \{\#\} (\text{m��et\_neg } M))$

### 3.7 Multiset Order

**instantiation** *zm��et* :: (*preorder*) *order*  
**begin**

**lift-definition** *less\_zm��et* ::  $'a$  *zm��et*  $\Rightarrow 'a$  *zm��et*  $\Rightarrow \text{bool}$  **is**  
 $\lambda(Mp, Mn) (Np, Nn). Mp + Nn < Mn + Np$

**proof** (*clarsimp simp: equiv\_zm��et\_def*)

**fix** *A1 B2 B1 A2 C1 D2 D1 C2* ::  $'a$  *multiset*

**assume**

*ab*:  $A1 + A2 = B1 + B2$  **and**

*cd*:  $C1 + C2 = D1 + D2$

**have**  $A1 + D2 < B2 + C1 \iff A1 + A2 + D2 < A2 + B2 + C1$

by *simp*

**also have**  $\dots \iff B1 + B2 + D2 < A2 + B2 + C1$

**unfolding** *ab* by (*rule refl*)

**also have**  $\dots \iff B1 + D2 < A2 + C1$

by *simp*

**also have**  $\dots \iff B1 + D1 + D2 < A2 + C1 + D1$

by *simp*

**also have**  $\dots \iff B1 + C1 + C2 < A2 + C1 + D1$

**using** *cd* by (*simp add: add.assoc*)

**also have**  $\dots \iff B1 + C2 < A2 + D1$

by *simp*

**finally show**  $A1 + D2 < B2 + C1 \iff B1 + C2 < A2 + D1$

by *assumption*

**qed**

**definition** *less\_eq\_zm��et* ::  $'a$  *zm��et*  $\Rightarrow 'a$  *zm��et*  $\Rightarrow \text{bool}$  **where**  
 $\text{less\_eq\_zm��et } M' M \iff M' < M \vee M' = M$

**instance**

**proof** (*(intro\_classes; unfold less\_eq\_zm��et\_def; transfer),*

*auto simp: equiv\_zm��et\_def union\_commute*)

**fix** *A1 B1 D C B2 A2* ::  $'a$  *multiset*

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

{

```

assume ab1: A1 + C < B1 + D

{
  assume ab2: D + A2 < C + B2
  show A1 + A2 < B1 + B2
  proof -
    have f1:  $\bigwedge m. D + A2 + m < C + B2 + m$ 
      using ab2 add_less_cancel_right by blast
    have  $\bigwedge m. C + (A1 + m) < D + (B1 + m)$ 
      by (simp add: ab1 add.commute)
    then have  $D + (A2 + A1) < D + (B1 + B2)$ 
      using f1 by (metis add.assoc add.commute mset_le_trans)
    then show ?thesis
      by (simp add: add.commute)
  qed
}
{
  assume ab2: D + A2 = C + B2
  show A1 + A2 < B1 + B2
  proof -
    have  $\bigwedge m. C + A1 + m < D + B1 + m$ 
      by (simp add: ab1 add.commute)
    then have  $D + (A2 + A1) < D + (B1 + B2)$ 
      by (metis (no_types) ab2 add.assoc add.commute)
    then show ?thesis
      by (simp add: add.commute)
  qed
}
}

{
  assume ab1: A1 + C = B1 + D

  {
    assume ab2: D + A2 < C + B2
    show A1 + A2 < B1 + B2
    proof -
      have  $A1 + (D + A2) < B1 + (D + B2)$ 
        by (metis (no_types) ab1 ab2 add.assoc add_less_cancel_left)
      then show ?thesis
        by simp
    qed
  }
  {
    assume ab2: D + A2 = C + B2
    have False
      by (metis (no_types) ab ab1 ab2 add.assoc add.commute add_diff_cancel_right')
    thus A1 + A2 < B1 + B2
      by sat
  }
}
qed

end

instance zmultiset :: (preorder) ordered_cancel_comm_monoid_add
  by (intro_classes, unfold less_eq_zmultiset_def, transfer, auto simp: equiv_zmset_def)

instance zmultiset :: (preorder) ordered_ab_group_add
  by (intro_classes; transfer; auto simp: equiv_zmset_def)

instantiation zmultiset :: (linorder) distrib_lattice
begin

```

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

**definition** *sup\_zmultiset* :: 'a zmultiset  $\Rightarrow$  'a zmultiset  $\Rightarrow$  'a zmultiset **where**  
*sup\_zmultiset* A B = (if B > A then B else A)

**lemma** *not\_lt\_iff\_ge\_zmset*:  $\neg x < y \longleftrightarrow x \geq y$  **for**  $x y :: 'a \text{ zmultiset}$   
**by** (unfold less\_eq\_zmultiset\_def, transfer, auto simp: equiv\_zmset\_def algebra\_simps)

**instance**

**by** *intro\_classes* (auto simp: less\_eq\_zmultiset\_def inf\_zmultiset\_def sup\_zmultiset\_def  
dest!: not\_lt\_iff\_ge\_zmset[THEN iffD1])

**end**

**lemma** *zmset\_of\_less*: *zmset\_of* M < *zmset\_of* N  $\longleftrightarrow$  M < N  
**by** (clarsimp simp: zmset\_of\_def, transfer, simp)+

**lemma** *zmset\_of\_le*: *zmset\_of* M  $\leq$  *zmset\_of* N  $\longleftrightarrow$  M  $\leq$  N  
**by** (simp\_all add: less\_eq\_zmultiset\_def zmset\_of\_def; transfer; auto simp: equiv\_zmset\_def)

**instance** *zmultiset* :: (preorder) ordered\_ab\_semigroup\_add  
**by** (*intro\_classes*, unfold less\_eq\_zmultiset\_def, transfer, auto simp: equiv\_zmset\_def)

**lemma** *uminus\_add\_conv\_diff\_mset*[*cancelation\_simproc\_pre*]:  $\langle -a + b = b - a \rangle$  **for**  $a :: 'a \text{ zmultiset}$   
**by** (simp add: add.commute)

**lemma** *uminus\_add\_add\_uminus*[*cancelation\_simproc\_pre*]:  $\langle b - a + c = b + c - a \rangle$  **for**  $a :: 'a \text{ zmultiset}$   
**by** (simp add: uminus\_add\_conv\_diff\_mset zmset\_subset\_eq\_zmultiset\_union\_diff\_commute)

**lemma** *add\_zmset\_eq\_add\_NO\_MATCH*[*cancelation\_simproc\_pre*]:  
 $\langle \text{NO\_MATCH } \{\#\}_z H \implies \text{add\_zmset } a H = \{\#a\# \}_z + H \rangle$   
**by** *auto*

**lemma** *repeat\_zmset\_iterate\_add*:  $\langle \text{repeat\_zmset } n M = \text{iterate\_add } n M \rangle$   
**unfolding** *iterate\_add\_def* **by** (*induction* n) *auto*

**declare** *repeat\_zmset\_iterate\_add*[*cancelation\_simproc\_pre*]

**declare** *repeat\_zmset\_iterate\_add*[*symmetric*, *cancelation\_simproc\_post*]

**simproc-setup** *zmseteq\_cancel\_numerals*

$((l :: 'a \text{ zmultiset}) + m = n \mid (l :: 'a \text{ zmultiset}) = m + n \mid$   
 $\text{add\_zmset } a m = n \mid m = \text{add\_zmset } a n \mid$   
 $\text{replicate\_zmset } p a = n \mid m = \text{replicate\_zmset } p a \mid$   
 $\text{repeat\_zmset } p m = n \mid m = \text{repeat\_zmset } p m) =$   
 $\langle \text{fn } \phi \implies \text{Cancel\_Simprocs.eq\_cancel} \rangle$

**lemma** *zmset\_subseteq\_add\_iff1*:

$\langle j \leq i \implies (\text{repeat\_zmset } i u + m \subseteq\#_z \text{repeat\_zmset } j u + n) = (\text{repeat\_zmset } (i - j) u + m \subseteq\#_z n) \rangle$   
**by** (simp add: add.commute add\_diff\_eq left\_diff\_repeat\_zmset\_distrib' subset\_eq\_diff\_conv\_zmset)

**lemma** *zmset\_subseteq\_add\_iff2*:

$\langle i \leq j \implies (\text{repeat\_zmset } i u + m \subseteq\#_z \text{repeat\_zmset } j u + n) = (m \subseteq\#_z \text{repeat\_zmset } (j - i) u + n) \rangle$

**proof** –

**assume**  $i \leq j$

**then have**  $\bigwedge z. \text{repeat\_zmset } j (z :: 'a \text{ zmultiset}) - \text{repeat\_zmset } i z = \text{repeat\_zmset } (j - i) z$

**by** (simp add: left\_diff\_repeat\_zmset\_distrib')

**then show** *?thesis*

**by** (*metis* add.commute diff\_diff\_eq2 subset\_eq\_diff\_conv\_zmset)

**qed**

**lemma** *zmset\_subset\_add\_iff1*:

$\langle j \leq i \implies (\text{repeat\_zmset } i \ u + m \subset\#_z \text{ repeat\_zmset } j \ u + n) = (\text{repeat\_zmset } (i - j) \ u + m \subset\#_z n) \rangle$   
**by** (*simp add: subset\_zmset.less\_le\_not\_le zmset\_subseteq\_add\_iff1 zmset\_subseteq\_add\_iff2*)

**lemma** *zmset\_subset\_add\_iff2*:

$\langle i \leq j \implies (\text{repeat\_zmset } i \ u + m \subset\#_z \text{ repeat\_zmset } j \ u + n) = (m \subset\#_z \text{ repeat\_zmset } (j - i) \ u + n) \rangle$   
**by** (*simp add: subset\_zmset.less\_le\_not\_le zmset\_subseteq\_add\_iff1 zmset\_subseteq\_add\_iff2*)

**ML-file**  $\langle \text{zmultiset\_simprocs.ML} \rangle$

**simproc-setup** *zmsetssubset\_cancel*

$((l::'a \ \text{zmultiset}) + m \subset\#_z n \mid (l::'a \ \text{zmultiset}) \subset\#_z m + n \mid$   
 $\text{add\_zmset } a \ m \subset\#_z n \mid m \subset\#_z \text{ add\_zmset } a \ n \mid$   
 $\text{replicate\_zmset } p \ a \subset\#_z n \mid m \subset\#_z \text{ replicate\_zmset } p \ a \mid$   
 $\text{repeat\_zmset } p \ m \subset\#_z n \mid m \subset\#_z \text{ repeat\_zmset } p \ m) =$   
 $\langle \text{fn } \phi \Rightarrow \text{ZMultiset\_Simprocs.subset\_cancel\_zmsets} \rangle$

**simproc-setup** *zmsetsubseteq\_cancel*

$((l::'a \ \text{zmultiset}) + m \subseteq\#_z n \mid (l::'a \ \text{zmultiset}) \subseteq\#_z m + n \mid$   
 $\text{add\_zmset } a \ m \subseteq\#_z n \mid m \subseteq\#_z \text{ add\_zmset } a \ n \mid$   
 $\text{replicate\_zmset } p \ a \subseteq\#_z n \mid m \subseteq\#_z \text{ replicate\_zmset } p \ a \mid$   
 $\text{repeat\_zmset } p \ m \subseteq\#_z n \mid m \subseteq\#_z \text{ repeat\_zmset } p \ m) =$   
 $\langle \text{fn } \phi \Rightarrow \text{ZMultiset\_Simprocs.subseteq\_cancel\_zmsets} \rangle$

**instance** *zmultiset* :: (*preorder*) *ordered\_ab\_semigroup\_add\_imp\_le*

**by** (*intro\_classes; unfold less\_eq\_zmultiset\_def; transfer; auto*)

**simproc-setup** *zmsetless\_cancel*

$((l::'a::\text{preorder } \text{zmultiset}) + m < n \mid (l::'a \ \text{zmultiset}) < m + n \mid$   
 $\text{add\_zmset } a \ m < n \mid m < \text{ add\_zmset } a \ n \mid$   
 $\text{replicate\_zmset } p \ a < n \mid m < \text{ replicate\_zmset } p \ a \mid$   
 $\text{repeat\_zmset } p \ m < n \mid m < \text{ repeat\_zmset } p \ m) =$   
 $\langle \text{fn } \phi \Rightarrow \text{Cancel\_Simprocs.less\_cancel} \rangle$

**simproc-setup** *zmsetless\_eq\_cancel*

$((l::'a::\text{preorder } \text{zmultiset}) + m \leq n \mid (l::'a \ \text{zmultiset}) \leq m + n \mid$   
 $\text{add\_zmset } a \ m \leq n \mid m \leq \text{ add\_zmset } a \ n \mid$   
 $\text{replicate\_zmset } p \ a \leq n \mid m \leq \text{ replicate\_zmset } p \ a \mid$   
 $\text{repeat\_zmset } p \ m \leq n \mid m \leq \text{ repeat\_zmset } p \ m) =$   
 $\langle \text{fn } \phi \Rightarrow \text{Cancel\_Simprocs.less\_eq\_cancel} \rangle$

**simproc-setup** *zmsetdiff\_cancel*

$(n + (l::'a \ \text{zmultiset}) \mid (l::'a \ \text{zmultiset}) - m \mid$   
 $\text{add\_zmset } a \ m - n \mid m - \text{ add\_zmset } a \ n \mid$   
 $\text{replicate\_zmset } p \ r - n \mid m - \text{ replicate\_zmset } p \ r \mid$   
 $\text{repeat\_zmset } p \ m - n \mid m - \text{ repeat\_zmset } p \ m) =$   
 $\langle \text{fn } \phi \Rightarrow \text{Cancel\_Simprocs.diff\_cancel} \rangle$

**instance** *zmultiset* :: (*linorder*) *linordered\_cancel\_ab\_semigroup\_add*

**by** (*intro\_classes, unfold less\_eq\_zmultiset\_def, transfer, auto simp: equiv\_zmset\_def add commute*)

**lemma** *less\_mset\_zmsetE*:

**assumes**  $M < N$

**obtains**  $A \ B \ C$  **where**

$M = \text{zmset\_of } A + C$  **and**  $N = \text{zmset\_of } B + C$  **and**  $A < B$

**by** (*metis add\_less\_imp\_less\_right assms decompose\_zmset\_of2 zmset\_of\_less*)

**lemma** *less\_eq\_mset\_zmsetE*:

**assumes**  $M \leq N$

**obtains**  $A \ B \ C$  **where**

$M = \text{zmset\_of } A + C$  **and**  $N = \text{zmset\_of } B + C$  **and**  $A \leq B$

**by** (*metis add commute add.right\_neutral assms le\_neq\_trans less\_imp\_le less\_mset\_zmsetE order\_refl zmset\_of\_empty*)

**lemma** *subset\_eq\_imp\_le\_zmset*:  $M \subseteq\#_z N \implies M \leq N$   
**by** (*metis* (*no\_types*) *add\_mono\_thms\_linordered\_semiring*(3) *subset\_eq\_imp\_le\_multiset* *subseteq\_mset\_zmsetE* *zmset\_of\_le*)

**lemma** *subset\_imp\_less\_zmset*:  $M \subset\#_z N \implies M < N$   
**by** (*metis* *le\_neq\_trans* *subset\_eq\_imp\_le\_zmset* *subset\_zmset\_def*)

**lemma** *lt\_imp\_ex\_zcount\_lt*:  
**assumes** *m\_lt\_n*:  $M < N$   
**shows**  $\exists y. \text{zcount } M \ y < \text{zcount } N \ y$   
**proof** (*rule* *ccontr*, *clarsimp*)  
**assume**  $\forall y. \neg \text{zcount } M \ y < \text{zcount } N \ y$   
**hence**  $\forall y. \text{zcount } M \ y \geq \text{zcount } N \ y$   
**by** (*simp* *add*: *leI*)  
**hence**  $M \supseteq\#_z N$   
**by** (*simp* *add*: *zmset\_subset\_eqI*)  
**hence**  $M \geq N$   
**by** (*simp* *add*: *subset\_eq\_imp\_le\_zmset*)  
**thus** *False*  
**using** *m\_lt\_n* **by** *simp*  
**qed**

**instance** *zmultiset* :: (*preorder*) *no\_top*

**proof**

**fix** *M* ::  $\langle 'a \ \text{zmultiset} \rangle$   
**obtain** *a* ::  $'a$  **where** *True* **by** *fast*  
**let**  $?M = \langle \text{zmset\_of } (\text{mset\_pos } M) + \text{zmset\_of } (\text{mset\_neg } M) \rangle$   
**have**  $\langle M < \text{add\_zmset } a \ ?M + ?M \rangle$   
**by** (*subst* *mset\_pos\_neg\_partition*)  
*(auto simp: subset\_zmset\_def subseteq\_zmset\_def zmultiset\_eq\_iff*  
*intro!: subset\_imp\_less\_zmset)*  
**then show**  $\langle \exists N. M < N \rangle$   
**by** *blast*

**qed**

**end**

## 4 Nested Multisets

**theory** *Nested\_Multiset*

**imports** *HOL-Library.Multiset\_Order*

**begin**

**declare** *multiset.map\_comp* [*simp*]

**declare** *multiset.map\_cong* [*cong*]

### 4.1 Type Definition

**datatype**  $'a \ \text{nmultiset} =$   
*Elem*  $'a$   
 $| \ \text{MSet } 'a \ \text{nmultiset} \ \text{multiset}$

**inductive** *no\_elem* ::  $'a \ \text{nmultiset} \Rightarrow \text{bool}$  **where**  
 $(\bigwedge X. X \in\# M \implies \text{no\_elem } X) \implies \text{no\_elem } (\text{MSet } M)$

**inductive-set** *sub\_nmset* ::  $( 'a \ \text{nmultiset} \times 'a \ \text{nmultiset} ) \ \text{set}$  **where**  
 $X \in\# M \implies (X, \text{MSet } M) \in \text{sub\_nmset}$

**lemma** *wf\_sub\_nmset*[*simp*]: *wf* *sub\_nmset*

**proof** (*rule* *wfUNIVI*)

**fix** *P* ::  $'a \ \text{nmultiset} \Rightarrow \text{bool}$  **and** *M* ::  $'a \ \text{nmultiset}$

**assume** *IH*:  $\forall M. (\forall N. (N, M) \in \text{sub\_nmset} \longrightarrow P \ N) \longrightarrow P \ M$



**show**  $P M$   
**by** (*induct*  $M$ ; *rule*  $IH[\text{rule\_format}]$ ) (*auto simp: sub\_nmset.simps*)  
**qed**

**primrec**  $\text{depth\_nmset} :: 'a \text{ nmultiset} \Rightarrow \text{nat} (\lfloor \_ \rfloor)$  **where**  
 $|\text{Elem } a| = 0$   
 $|\text{MSet } M| = (\text{let } X = \text{set\_mset } (\text{image\_mset } \text{depth\_nmset } M) \text{ in if } X = \{\} \text{ then } 0 \text{ else } \text{Suc } (\text{Max } X))$

**lemma**  $\text{depth\_nmset\_MSet}: x \in\# M \Longrightarrow |x| < |\text{MSet } M|$   
**by** (*auto simp: less\_Suc\_eq\_le*)

**declare**  $\text{depth\_nmset.simps}(2)[\text{simp del}]$

## 4.2 Dershowitz and Manna's Nested Multiset Order

The Dershowitz–Manna extension:

**definition**  $\text{less\_multiset\_ext}_{DM} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ multiset} \Rightarrow 'a \text{ multiset} \Rightarrow \text{bool}$  **where**  
 $\text{less\_multiset\_ext}_{DM} R M N \longleftrightarrow$   
 $(\exists X Y. X \neq \{\#\} \wedge X \subseteq\# N \wedge M = (N - X) + Y \wedge (\forall k. k \in\# Y \longrightarrow (\exists a. a \in\# X \wedge R k a)))$

**lemma**  $\text{less\_multiset\_ext}_{DM}\text{\_imp\_mult}$ :

**assumes**

$N\_A: \text{set\_mset } N \subseteq A$  **and**  $M\_A: \text{set\_mset } M \subseteq A$  **and**  $\text{less}: \text{less\_multiset\_ext}_{DM} R M N$

**shows**  $(M, N) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge R x y\}$

**proof** –

**from**  $\text{less}$  **obtain**  $X Y$  **where**

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

**unfolding**  $\text{less\_multiset\_ext}_{DM}\text{\_def}$  **by**  $\text{blast}$

**with**  $N\_A M\_A$  **have**  $(N - X + Y, N - X + X) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge R x y\}$

**by** (*intro one\_step\_implies\_mult, blast,*

*metis (mono\_tags, lifting) case\_prodI mem\_Collect\_eq mset\_subset\_eqD mset\_subset\_eq\_add\_right subsetCE*)

**with**  $(M = N - X + Y) (X \subseteq\# N)$  **show**  $(M, N) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge R x y\}$

**by** (*simp add: subset\_mset.diff\_add*)

**qed**

**lemma**  $\text{mult\_imp\_less\_multiset\_ext}_{DM}$ :

**assumes**

$N\_A: \text{set\_mset } N \subseteq A$  **and**  $M\_A: \text{set\_mset } M \subseteq A$  **and**

$\text{trans}: \forall x \in A. \forall y \in A. \forall z \in A. R x y \longrightarrow R y z \longrightarrow R x z$  **and**

$\text{in\_mult}: (M, N) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge R x y\}$

**shows**  $\text{less\_multiset\_ext}_{DM} R M N$

**using**  $\text{in\_mult } N\_A M\_A$  **unfolding**  $\text{mult\_def less\_multiset\_ext}_{DM}\text{\_def}$

**proof** *induct*

**case** (*base*  $N$ )

**then obtain**  $y M0 X$  **where**  $N = \text{add\_mset } y M0$  **and**  $M = M0 + X$  **and**  $\forall a. a \in\# X \longrightarrow R a y$

**unfolding**  $\text{mult1\_def}$  **by** *auto*

**thus** *?case*

**by** (*auto intro: exI[of \_ \{\#y\#}])*)

**next**

**case** (*step*  $N N'$ )

**note**  $N\_N'\text{\_in\_mult1} = \text{this}(2)$  **and**  $\text{ih} = \text{this}(3)$  **and**  $N'\_A = \text{this}(4)$  **and**  $M\_A = \text{this}(5)$

**have**  $N\_A: \text{set\_mset } N \subseteq A$

**using**  $N\_N'\text{\_in\_mult1 } N'\_A$  **unfolding**  $\text{mult1\_def}$  **by** *auto*

**obtain**  $Y X$  **where**  $y\_nemp: Y \neq \{\#\}$  **and**  $y\_sub\_N: Y \subseteq\# N$  **and**  $M\_eq: M = N - Y + X$  **and**

$\text{ex\_y}: \forall x. x \in\# X \longrightarrow (\exists y. y \in\# Y \wedge R x y)$

**using**  $\text{ih}[OF N\_A M\_A]$  **by**  $\text{blast}$

**obtain**  $z M0 Ya$  **where**  $N'\_eq: N' = M0 + \{\#z\#\}$  **and**  $N\_eq: N = M0 + Ya$  **and**

$\text{z\_gt}: \forall y. y \in\# Ya \longrightarrow y \in A \wedge z \in A \wedge R y z$

**using**  $N\_N'\text{\_in\_mult1}[\text{unfolded } \text{mult1\_def}]$  **by** *auto*

```

let ?Za = Y - Ya + {#z#}
let ?Xa = X + Ya + (Y - Ya) - Y

have xa_sub_x_ya: set_mset ?Xa ⊆ set_mset (X + Ya)
  by (metis diff_subset_eq_self in_diffD subsetI subset_mset.diff_diff_right)

have x_A: set_mset X ⊆ A
  using M_A M_eq by auto
have ya_A: set_mset Ya ⊆ A
  by (simp add: subsetI z_gt)

have ex_y': ∃ y. y ∈# Y - Ya + {#z#} ∧ R x y if x_in: x ∈# X + Ya for x
proof (cases x ∈# X)
  case True
  then obtain y where y_in: y ∈# Y and y_gt_x: R x y
    using ex_y by blast
  show ?thesis
  proof (cases y ∈# Ya)
    case False
    hence y ∈# Y - Ya + {#z#}
      using y_in count_greater_zero_iff_in_diff_count by fastforce
    thus ?thesis
      using y_gt_x by blast
  next
  case True
  hence y ∈ A and z ∈ A and R y z
    using z_gt by blast+
  hence R x z
    using trans y_gt_x x_A ya_A x_in by (meson subsetCE union_iff)
  thus ?thesis
    by auto
  qed
next
  case False
  hence x ∈# Ya
    using x_in by auto
  hence x ∈ A and z ∈ A and R x z
    using z_gt by blast+
  thus ?thesis
    by auto
  qed

show ?case
proof (rule exI[of _ ?Za], rule exI[of _ ?Xa], intro conjI)
  show Y - Ya + {#z#} ⊆# N'
    using mset_subset_eq_mono_add subset_eq_diff_conv y_sub_N N_eq N'_eq
    by (simp add: subset_eq_diff_conv)
  next
  show M = N' - (Y - Ya + {#z#}) + (X + Ya + (Y - Ya) - Y)
    unfolding M_eq N_eq N'_eq by (auto simp: multiset_eq_iff)
  next
  show ∀ x. x ∈# X + Ya + (Y - Ya) - Y ⟶ (∃ y. y ∈# Y - Ya + {#z#} ∧ R x y)
    using ex_y' xa_sub_x_ya by blast
  qed auto
qed

lemma less_multiset_ext_DM_iff_mult:
  assumes
    N_A: set_mset N ⊆ A and M_A: set_mset M ⊆ A and
    trans: ∀ x ∈ A. ∀ y ∈ A. ∀ z ∈ A. R x y ⟶ R y z ⟶ R x z
  shows less_multiset_ext_DM R M N ⟷ (M, N) ∈ mult {(x, y). x ∈ A ∧ y ∈ A ∧ R x y}
  using mult_imp_less_multiset_ext_DM[OF assms] less_multiset_ext_DM_imp_mult[OF N_A M_A] by blast

```

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

**lemma** *less\_multiset\_ext\_DM\_cong*[*fundef\_cong*]:

$(\bigwedge X Y k a. X \neq \{\#\} \implies X \subseteq\# N \implies M = (N - X) + Y \implies k \in\# Y \implies R k a = S k a) \implies$   
 $less\_multiset\_ext_{DM} R M N = less\_multiset\_ext_{DM} S M N$   
**unfolding** *less\_multiset\_ext\_DM\_def* **by** *metis*

**function** *less\_nmultiset* :: 'a *nmultiset*  $\Rightarrow$  'a *nmultiset*  $\Rightarrow$  bool **where**

*less\_nmultiset* (Elem a) (Elem b)  $\longleftrightarrow a < b$   
| *less\_nmultiset* (Elem a) (MSet M)  $\longleftrightarrow$  True  
| *less\_nmultiset* (MSet M) (Elem a)  $\longleftrightarrow$  False  
| *less\_nmultiset* (MSet M) (MSet N)  $\longleftrightarrow less\_multiset\_ext_{DM} less\_nmultiset M N$   
**by** *pat\_completeness auto*

**termination**

**by** (relation *sub\_nmset*  $\langle *lex* \rangle$  *sub\_nmset*, *fastforce*,  
*metis sub\_nmset.simps in\_lex\_prod mset\_subset\_eqD mset\_subset\_eq\_add\_right*)

**lemmas** *less\_nmultiset\_induct* =

*less\_nmultiset.induct*[*case\_names Elem Elem Elem\_MSet MSet\_Elem MSet\_MSet*]

**lemmas** *less\_nmultiset\_cases* =

*less\_nmultiset.cases*[*case\_names Elem Elem Elem\_MSet MSet\_Elem MSet\_MSet*]

**lemma** *trans\_less\_nmultiset*:  $X < Y \implies Y < Z \implies X < Z$  **for**  $X Y Z :: 'a$  *nmultiset*

**proof** (*induct* *Max*  $\{|X|, |Y|, |Z|\}$  *arbitrary*:  $X Y Z$

*rule*: *less\_induct*)

**case** *less*

**from** *less*(2,3) **show** ?*case*

**proof** (*cases*  $X$ ; *cases*  $Y$ ; *cases*  $Z$ )

**fix**  $M N N' :: 'a$  *nmultiset multiset*

**define**  $A$  **where**  $A = set\_mset M \cup set\_mset N \cup set\_mset N'$

**assume**  $XYZ: X = MSet M Y = MSet N Z = MSet N'$

**then have** *trans*:  $\forall x \in A. \forall y \in A. x < y \longrightarrow y < z \longrightarrow x < z$

**by** (*auto elim!*: *less*(1)[*rotated* -1] *dest!*: *depth\_nmset\_MSet simp add: A\_def*)

**have**  $set\_mset M \subseteq A \ set\_mset N \subseteq A \ set\_mset N' \subseteq A$

**unfolding**  $A\_def$  **by** *auto*

**with** *less*(2,3)  $XYZ$  **show**  $X < Z$

**by** (*auto simp*: *less\_multiset\_ext\_DM\_iff\_mult*[*OF*  $\_ \_ trans$ ] *mult\_def*)

**qed** (*auto elim*: *less\_trans*)

**qed**

**lemma** *irrefl\_less\_nmultiset*:

**fixes**  $X :: 'a$  *nmultiset*

**shows**  $X < X \implies False$

**proof** (*induct*  $X$ )

**case** (MSet  $M$ )

**from** *MSet*(2) **show** ?*case*

**unfolding** *less\_nmultiset.simps less\_multiset\_ext\_DM\_def*

**proof** *safe*

**fix**  $X Y :: 'a$  *nmultiset multiset*

**define**  $XY$  **where**  $XY = \{(x, y). x \in\# X \wedge y \in\# Y \wedge y < x\}$

**then have** *fin*: *finite*  $XY$  **and** *trans*: *trans*  $XY$

**by** (*auto simp*: *trans\_def intro*: *trans\_less\_nmultiset*

*finite\_subset*[*OF*  $\_ finite\_cartesian\_product$ ])

**assume**  $X \neq \{\#\} X \subseteq\# M M = M - X + Y$

**then have**  $X = Y$

**by** (*auto simp*: *mset\_subset\_eq\_exists\_conv*)

**with** *MSet*(1)  $\langle X \subseteq\# M \rangle$  **have** *irrefl*  $XY$

**unfolding**  $XY\_def$  **by** (*force dest*: *mset\_subset\_eqD simp*: *irrefl\_def*)

**with** *trans* **have** *acyclic*  $XY$

**by** (*simp add*: *acyclic\_irrefl*)

```

moreover
assume  $\forall k. k \in \# Y \longrightarrow (\exists a. a \in \# X \wedge k < a)$ 
with  $\langle X = Y \rangle \langle X \neq \{\#\} \rangle$  have  $\neg \text{acyclic } XY$ 
  by (intro notI, elim finite_acyclic_wf[OF fin, elim_format])
    (auto dest!: spec[of _ set_mset Y] simp: wf_eq_minimal XY_def)
ultimately show False by blast
qed
qed simp

```

```

lemma antisym_less_nmultipset:
  fixes  $X Y :: 'a \text{ nmultipset}$ 
  shows  $X < Y \implies Y < X \implies \text{False}$ 
  using trans_less_nmultipset irrefl_less_nmultipset by blast

```

```

definition less_eq_nmultipset ::  $'a \text{ nmultipset} \Rightarrow 'a \text{ nmultipset} \Rightarrow \text{bool}$  where
  less_eq_nmultipset  $X Y = (X < Y \vee X = Y)$ 

```

```

instance
proof (intro_classes, goal_cases less_def refl trans)
  case (less_def  $x y$ )
  then show ?case
    unfolding less_eq_nmultipset_def by (metis irrefl_less_nmultipset antisym_less_nmultipset)
next
  case (refl  $x$ )
  then show ?case
    unfolding less_eq_nmultipset_def by blast
next
  case (trans  $x y z$ )
  then show ?case
    unfolding less_eq_nmultipset_def by (metis trans_less_nmultipset)
qed

```

```

lemma less_multiset_ext_DM_less: less_multiset_ext_DM ( $<$ ) = ( $<$ )
  unfolding fun_eq_iff less_multiset_ext_DM_def less_multiset_DM by blast

```

**end**

```

instantiation nmultipset :: (order) order
begin

```

```

instance
proof (intro_classes, goal_cases antisym)
  case (antisym  $x y$ )
  then show ?case
    unfolding less_eq_nmultipset_def by (metis trans_less_nmultipset irrefl_less_nmultipset)
qed

```

**end**

```

instantiation nmultipset :: (linorder) linorder
begin

```

```

lemma total_less_nmultipset:
  fixes  $X Y :: 'a \text{ nmultipset}$ 
  shows  $\neg X < Y \implies Y \neq X \implies Y < X$ 
proof (induct X Y rule: less_nmultipset_induct)
  case (MSet_MSet  $M N$ )
  then show ?case
    unfolding nmultipset.inject less_nmultipset.simps less_multiset_ext_DM_less less_multiset_HO
    by (metis add_diff_cancel_left' count_inI diff_add_zero_in_diff_count less_imp_not_less
      mset_subset_eq_multiset_union_diff_commute subset_mset.order.refl)
qed auto

```

```

instance
proof (intro_classes, goal_cases total)
  case (total x y)
  then show ?case
    unfolding less_eq_nmultiset_def by (metis total_less_nmultiset)
qed

end

lemma less_depth_nmultiset_imp_less_nmultiset:  $|X| < |Y| \implies X < Y$ 
proof (induct X Y rule: less_nmultiset_induct)
  case (MSet_MSet M N)
  then show ?case
  proof (cases M = {#})
    case False
    with MSet_MSet show ?thesis
      by (auto 0 4 simp: depth_nmultiset.simps(2) less_multiset_ext_DM_def not_le Max_gr_iff
        intro: exI[of _ N] split: if_splits)
    qed (auto simp: depth_nmultiset.simps(2) less_multiset_ext_DM_less split: if_splits)
  qed simp_all

lemma less_nmultiset_imp_le_depth_nmultiset:  $X < Y \implies |X| \leq |Y|$ 
proof (induct X Y rule: less_nmultiset_induct)
  case (MSet_MSet M N)
  then have  $M < N$  by (simp add: less_multiset_ext_DM_less)
  then show ?case
  proof (cases M = {#} N = {#} rule: bool.exhaust[case_product bool.exhaust])
    case [simp]: False_False
    show ?thesis
    unfolding depth_nmultiset.simps(2) Let_def False_False Suc_le_mono set_image_mset image_is_empty
      set_mset_eq_empty_iff if_False
    proof (intro iffD2[OF Max_le_iff] ballI iffD2[OF Max_ge_iff]; (elim imageE)?; simp)
      fix X
      assume [simp]:  $X \in \# M$ 
      with MSet_MSet(1)[of N M X, simplified]  $\langle M < N \rangle$  show  $\exists Y \in \# N. |X| \leq |Y|$ 
      by (meson ex_gt_imp_less_multiset less_asym' less_depth_nmultiset_imp_less_nmultiset
        not_le_imp_less)
    qed
  qed (auto simp: depth_nmultiset.simps(2))
qed simp_all

lemma eq_mlex_I:
  fixes  $f :: 'a \Rightarrow \text{nat}$  and  $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
  assumes  $\bigwedge X Y. f X < f Y \implies R X Y$  and antisym R
  shows  $\{(X, Y). R X Y\} = f \langle *mlex* \rangle \{(X, Y). f X = f Y \wedge R X Y\}$ 
proof safe
  fix X Y
  assume R X Y
  show  $(X, Y) \in f \langle *mlex* \rangle \{(X, Y). f X = f Y \wedge R X Y\}$ 
  proof (cases f X f Y rule: linorder_cases)
    case less
    with  $\langle R X Y \rangle$  show ?thesis
      by (elim mlex_less)
    next
    case equal
    with  $\langle R X Y \rangle$  show ?thesis
      by (intro mlex_leq) auto
    next
    case greater
    from  $\langle R X Y \rangle$  assms(1)[OF greater]  $\langle \text{antisym } R \rangle$  greater show ?thesis
      unfolding antisym_def by auto
  qed
qed
next

```

```

fix X Y
assume (X, Y) ∈ f <*> { (X, Y). f X = f Y ∧ R X Y }
then show R X Y
  unfolding mlex_prod_def by (auto simp: assms(1))
qed

instantiation nmultiset :: (wellorder) wellorder
begin

lemma depth_nmset_eq_0[simp]: |X| = 0 ↔ (X = MSet {} ∨ (∃ x. X = Elem x))
  by (cases X; simp add: depth_nmset.simps(2))

lemma depth_nmset_eq_Suc[simp]: |X| = Suc n ↔
  (∃ N. X = MSet N ∧ (∃ Y ∈# N. |Y| = n) ∧ (∀ Y ∈# N. |Y| ≤ n))
  by (cases X; auto simp add: depth_nmset.simps(2) intro!: Max_eqI
    (metis (no_types, lifting) Max_in finite_imageI finite_set_mset imageE image_is_empty
      set_mset_eq_empty_iff))

lemma wf_less_nmultiset_depth:
  wf {(X :: 'a nmultiset, Y). |X| = i ∧ |Y| = i ∧ X < Y}
proof (induct i rule: less_induct)
  case (less i)
  define A :: 'a nmultiset set where A = {X. |X| < i}
  from less have wf ((depth_nmset :: 'a nmultiset ⇒ nat) <*>
    (∪ j < i. {(X, Y). |X| = j ∧ |Y| = j ∧ X < Y}))
  by (intro wf_UN wf_mlex) auto
  then have *: wf (mult {(X :: 'a nmultiset, Y). X ∈ A ∧ Y ∈ A ∧ X < Y})
  by (intro wf_mult, elim wf_subset) (force simp: A_def mlex_prod_def not_less_iff_gr_or_eq
    dest!: less_depth_nmset_imp_less_nmultiset)
  show ?case
  proof (cases i)
    case 0
    then show ?thesis
    by (auto simp: inj_on_def intro!: wf_subset[OF
      wf_Un[OF wf_map_prod_image[OF wf, of Elem] wf_UN[of Elem ' UNIV λx. {(x, MSet {})}]]])
  next
    case (Suc n)
    then show ?thesis
    by (intro wf_subset[OF wf_map_prod_image[OF *, of MSet]])
      (auto 0 4 simp: map_prod_def image_iff inj_on_def A_def
        dest!: less_multiset_ext_DM_imp_mult[of _ A, rotated -1] split: prod.splits)
  qed
qed

lemma wf_less_nmultiset: wf {(X :: 'a nmultiset, Y :: 'a nmultiset). X < Y} (is wf ?R)
proof -
  have ?R = depth_nmset <*> {(X, Y). |X| = |Y| ∧ X < Y}
  by (rule eq_mlex_I) (auto simp: antisymp_def less_depth_nmset_imp_less_nmultiset)
  also have {(X, Y). |X| = |Y| ∧ X < Y} = (∪ i. {(X, Y). |X| = i ∧ |Y| = i ∧ X < Y})
  by auto
  finally show ?thesis
  by (fastforce intro: wf_mlex wf_Union wf_less_nmultiset_depth)
qed

instance using wf_less_nmultiset unfolding wf_def mem_Collect_eq prod.case by intro_classes metis
end
end

```

## 5 Hereditar(il)y (Finite) Multisets

theory Hereditary\_Multiset

```
imports Multiset_More Nested_Multiset
begin
```

## 5.1 Type Definition

```
datatype hmultiset =
  HMSet (hmsetmset: hmultiset multiset)
```

```
lemma hmsetmset_inject[simp]: hmsetmset A = hmsetmset B  $\longleftrightarrow$  A = B
  by (blast intro: hmultiset.expand)
```

```
primrec Rep_hmultiset :: hmultiset  $\Rightarrow$  unit nmultiset where
  Rep_hmultiset (HMSet M) = MSet (image_mset Rep_hmultiset M)
```

```
primrec (nonexhaustive) Abs_hmultiset :: unit nmultiset  $\Rightarrow$  hmultiset where
  Abs_hmultiset (MSet M) = HMSet (image_mset Abs_hmultiset M)
```

```
lemma type_definition_hmultiset: type_definition Rep_hmultiset Abs_hmultiset {X. no_elem X}
```

```
proof (unfold_locales, unfold mem_Collect_eq)
```

```
  fix X
  show no_elem (Rep_hmultiset X)
    by (induct X) (auto intro!: no_elem.intros)
  show Abs_hmultiset (Rep_hmultiset X) = X
    by (induct X) auto
```

```
next
```

```
  fix Y :: unit nmultiset
  assume no_elem Y
  thus Rep_hmultiset (Abs_hmultiset Y) = Y
    by (induct Y rule: no_elem.induct) auto
```

```
qed
```

```
setup-lifting type_definition_hmultiset
```

```
lemma HMSet_alt: HMSet = Abs_hmultiset o MSet o image_mset Rep_hmultiset
  by (auto simp: type_definition.Rep_inverse[OF type_definition_hmultiset])
```

```
lemma HMSet_transfer[transfer_rule]: rel_fun (rel_mset pcr_hmultiset) pcr_hmultiset MSet HMSet
```

```
  unfolding HMSet_alt by (force simp: rel_fun_def multiset.in_rel nmultiset.rel_eq
    pcr_hmultiset_def cr_hmultiset_def
    type_definition.Rep_inverse[OF type_definition_hmultiset] intro!: multiset.map_cong)
```

## 5.2 Restriction of Dershowitz and Manna's Nested Multiset Order

```
instantiation hmultiset :: linorder
begin
```

```
lift-definition less_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  bool is (<).
```

```
lift-definition less_eq_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  bool is ( $\leq$ ).
```

```
instance
```

```
  by (intro_classes; transfer) auto
```

```
end
```

```
lemma less_HMSet_iff_less_multiset_ext_DM: HMSet M < HMSet N  $\longleftrightarrow$  less_multiset_ext_DM (<) M N
```

```
  unfolding less_multiset_ext_DM_def
```

```
proof (transfer, unfold less_nmultiset.simps less_multiset_ext_DM_def, safe)
```

```
  fix M N :: unit nmultiset multiset and X Y
```

```
  assume *: pred_mset no_elem (N - X + Y) pred_mset no_elem N X  $\neq$  {#}
```

```
  X  $\subseteq$  # N  $\forall$  k. k  $\in$  # Y  $\longrightarrow$  ( $\exists$  a. a  $\in$  # X  $\wedge$  k < a)
```

```
  then have X  $\in$  Collect (pred_mset no_elem)
```

```
  unfolding multiset.pred_set mem_Collect_eq by (metis rev_subsetD set_mset_mono)
```

```
  from *(1) have Y  $\in$  Collect (pred_mset no_elem)
```

```
  unfolding multiset.pred_set mem_Collect_eq by (metis add_diff_cancel_left' in_diffD)
```

**show**  
 $\exists X' \in \text{Collect } (\text{pred\_mset no\_elem}). \exists Y' \in \text{Collect } (\text{pred\_mset no\_elem}).$   
 $X' \neq \{\#\} \wedge \text{filter\_mset no\_elem } X' \subseteq \# \text{ filter\_mset no\_elem } N \wedge N - X + Y = N - X' + Y' \wedge$   
 $(\forall k \in \text{Collect no\_elem}. k \in \# Y' \longrightarrow (\exists a \in \text{Collect no\_elem}. a \in \# X' \wedge k < a))$   
**by** (*rule*  $\text{beI}[OF\_ \langle X \in \text{Collect } (\text{pred\_mset no\_elem}) \rangle]$ ,  
*rule*  $\text{beI}[OF\_ \langle Y \in \text{Collect } (\text{pred\_mset no\_elem}) \rangle]$ )  
*(insert \*; force simp: set\_mset\_diff multiset.pred\_set multiset\_filter\_mono)*

**next**  
**fix**  $M N :: \text{unit nmultiset multiset and } X Y$   
**assume** \*:  
 $\text{pred\_mset no\_elem } (N - X + Y) \text{ pred\_mset no\_elem } N \text{ pred\_mset no\_elem } X \text{ pred\_mset no\_elem } Y$   
 $X \neq \{\#\} \text{ filter\_mset no\_elem } X \subseteq \# \text{ filter\_mset no\_elem } N$   
 $\forall k \in \text{Collect no\_elem}. k \in \# Y \longrightarrow (\exists a \in \text{Collect no\_elem}. a \in \# X \wedge k < a)$   
**then have** [*simp*]:  $\text{filter\_mset no\_elem } X = X \text{ filter\_mset no\_elem } N = N$   
**unfolding**  $\text{filter\_mset\_eq\_conv}$  **by** (*auto simp: multiset.pred\_set*)  
**show**  
 $\exists X' Y'. X' \neq \{\#\} \wedge X' \subseteq \# N \wedge N - X + Y = N - X' + Y' \wedge$   
 $(\forall k. k \in \# Y' \longrightarrow (\exists a. a \in \# X' \wedge k < a))$   
**by** (*rule*  $\text{exI}[of\_ X]$ , *rule*  $\text{exI}[of\_ Y]$ ) (*insert \*; auto simp: multiset.pred\_set*)

**qed**

**lemma**  $\text{hmsetmset\_less}[simp]$ :  $\text{hmsetmset } M < \text{hmsetmset } N \longleftrightarrow M < N$   
**by** (*cases*  $M$ , *cases*  $N$ , *simp add: less\_multiset\_ext<sub>DM</sub>\_less less\_HMSet\_iff\_less\_multiset\_ext<sub>DM</sub>*)

**lemma**  $\text{hmsetmset\_le}[simp]$ :  $\text{hmsetmset } M \leq \text{hmsetmset } N \longleftrightarrow M \leq N$   
**unfolding**  $\text{le\_less hmsetmset\_less}$  **by** (*metis hmultiset.collapse*)

**lemma**  $\text{wf\_less\_hmultiset}$ :  $\text{wf } \{(X :: \text{hmultiset}, Y :: \text{hmultiset}). X < Y\}$   
**unfolding**  $\text{wf\_eq\_minimal}$  **by** *transfer (insert wf\_less\_nmultiset[unfolded wf\_eq\_minimal], fast)*

**instance**  $\text{hmultiset} :: \text{wellorder}$   
**using**  $\text{wf\_less\_hmultiset}$  **unfolding**  $\text{wf\_def mem\_Collect\_eq prod.case}$  **by** *intro\_classes metis*

**lemma**  $\text{HMSet\_less}[simp]$ :  $\text{HMSet } M < \text{HMSet } N \longleftrightarrow M < N$   
**by** (*simp add: less\_HMSet\_iff\_less\_multiset\_ext<sub>DM</sub>\_less less\_multiset\_ext<sub>DM</sub>\_less*)

**lemma**  $\text{HMSet\_le}[simp]$ :  $\text{HMSet } M \leq \text{HMSet } N \longleftrightarrow M \leq N$   
**by** (*simp add: hmsetmset\_le[symmetric]*)

**lemma**  $\text{mem\_imp\_less\_HMSet}$ :  $k \in \# L \implies k < \text{HMSet } L$   
**by** (*induct*  $k$  *arbitrary: L*) (*auto intro: ex\_gt\_imp\_less\_multiset*)

**lemma**  $\text{mem\_hmsetmset\_imp\_less}$ :  $M \in \# \text{hmsetmset } N \implies M < N$   
**using**  $\text{mem\_imp\_less\_HMSet}$  **by** *force*

### 5.3 Disjoint Union and Truncated Difference

**instantiation**  $\text{hmultiset} :: \text{cancel\_comm\_monoid\_add}$   
**begin**

**definition**  $\text{zero\_hmultiset} :: \text{hmultiset where}$   
 $0 = \text{HMSet } \{\#\}$

**lemma**  $\text{hmsetmset\_empty\_iff}[simp]$ :  $\text{hmsetmset } n = \{\#\} \longleftrightarrow n = 0$   
**unfolding**  $\text{zero\_hmultiset\_def}$  **by** (*cases*  $n$ ) *simp*

**lemma**  $\text{hmsetmset\_0}[simp]$ :  $\text{hmsetmset } 0 = \{\#\}$   
**by** *simp*

**lemma**  
 $\text{HMSet\_eq\_0\_iff}[simp]$ :  $\text{HMSet } m = 0 \longleftrightarrow m = \{\#\}$  **and**  
 $\text{zero\_eq\_HMSet}[simp]$ :  $0 = \text{HMSet } m \longleftrightarrow m = \{\#\}$   
**by** (*cases*  $m$ ) (*auto simp: zero\_hmultiset\_def*)



```

definition plus_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  hmultiset where
  A + B = HMSet (hmultiset A + hmultiset B)

definition minus_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  hmultiset where
  A - B = HMSet (hmultiset A - hmultiset B)

instance
  by intro_classes (auto simp: zero_hmultiset_def plus_hmultiset_def minus_hmultiset_def)

end

lemma HMSet_plus: HMSet (A + B) = HMSet A + HMSet B
  by (simp add: plus_hmultiset_def)

lemma HMSet_diff: HMSet (A - B) = HMSet A - HMSet B
  by (simp add: minus_hmultiset_def)

lemma hmultiset_plus: hmultiset (M + N) = hmultiset M + hmultiset N
  by (simp add: plus_hmultiset_def)

lemma hmultiset_diff: hmultiset (M - N) = hmultiset M - hmultiset N
  by (simp add: minus_hmultiset_def)

lemma diff_diff_add_hmultiset[simp]: a - b - c = a - (b + c) for a b c :: hmultiset
  by (fact diff_diff_add)

instance hmultiset :: comm_monoid_diff
  by intro_classes (auto simp: zero_hmultiset_def minus_hmultiset_def)

simproc-setup hmultiset_eq_cancel
  ((l::hmultiset) + m = n | (l::hmultiset) = m + n) =
  ⟨fn phi => Cancel_Simprocs.eq_cancel⟩

simproc-setup hmultiset_diff_cancel
  (((l::hmultiset) + m) - n | (l::hmultiset) - (m + n)) =
  ⟨fn phi => Cancel_Simprocs.diff_cancel⟩

simproc-setup hmultiset_less_cancel
  ((l::hmultiset) + m < n | (l::hmultiset) < m + n) =
  ⟨fn phi => Cancel_Simprocs.less_cancel⟩

simproc-setup hmultiset_less_eq_cancel
  ((l::hmultiset) + m ≤ n | (l::hmultiset) ≤ m + n) =
  ⟨fn phi => Cancel_Simprocs.less_eq_cancel⟩

instance hmultiset :: ordered_cancel_comm_monoid_add
  by intro_classes (simp del: hmultiset_less add: plus_hmultiset_def order_le_less
    hmultiset_less[symmetric] less_multiset_extDM_less)

instance hmultiset :: ordered_ab_semigroup_add_imp_le
  by intro_classes (simp add: plus_hmultiset_def order_le_less less_multiset_extDM_less)

instantiation hmultiset :: order_bot
begin

definition bot_hmultiset :: hmultiset where
  bot_hmultiset = 0

instance
proof (intro_classes, unfold bot_hmultiset_def zero_hmultiset_def, transfer, goal_cases bot_least)
  case (bot_least x)
  thus ?case
  by (induct x rule: no_elem.induct) (auto simp: less_eq_nmultiset_def less_multiset_extDM_less)

```

qed

end

```
instance hmultiset :: no_top
proof (intro_classes, goal_cases gt_ex)
  case (gt_ex a)
  have a < a + HMSet {#0#}
  by (simp add: zero_hmultiset_def)
  thus ?case
  by (rule exI)
qed
```

**lemma** *le\_minus\_plus\_same\_hmset*:  $m \leq m - n + n$  for  $m\ n :: hmultiset$

```
proof (cases m n rule: hmultiset.exhaust[case_product hmultiset.exhaust])
  case (HMSet_HMSet m0 n0)
  note m = this(1) and n = this(2)
```

```
{
  assume n0  $\subseteq\#$  m0
  hence m0 = m0 - n0 + n0
  by simp
}
moreover
{
  assume  $\neg$  n0  $\subseteq\#$  m0
  hence m0  $\subset\#$  m0 - n0 + n0
  by (metis mset_subset_eq_add_right subset_eq_diff_conv subset_mset.dual_order.refl
    subset_mset_def)
  hence m0 < m0 - n0 + n0
  by (rule subset_imp_less_mset)
}
ultimately show ?thesis
by (simp (no_asm) add: m n order_le_less plus_hmultiset_def minus_hmultiset_def) blast
qed
```

## 5.4 Infimum and Supremum

```
instantiation hmultiset :: distrib_lattice
begin
```

```
definition inf_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  hmultiset where
  inf_hmultiset A B = (if A < B then A else B)
```

```
definition sup_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  hmultiset where
  sup_hmultiset A B = (if B > A then B else A)
```

```
instance
  by intro_classes (auto simp: inf_hmultiset_def sup_hmultiset_def)
```

end

## 5.5 Inequalities

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

```
lemma
  le_add1_hmset:  $n \leq n + m$  and
  le_add2_hmset:  $n \leq m + n$  for  $n :: hmultiset$ 
  by simp+
```

```
lemma le_zero_eq_hmset[simp]:  $M \leq 0 \iff M = 0$  for  $M :: hmultiset$ 
```

```

by (simp add: dual_order.antisym)

lemma not_less_zero_hmset[simp]:  $\neg M < 0$  for  $M :: \text{hmultiset}$ 
  using not_le zero_le_hmset by blast

lemma not_gr_zero_hmset[simp]:  $\neg 0 < M \longleftrightarrow M = 0$  for  $M :: \text{hmultiset}$ 
  using neqE not_less_zero_hmset by blast

lemma zero_less_iff_neq_zero_hmset:  $0 < M \longleftrightarrow M \neq 0$  for  $M :: \text{hmultiset}$ 
  using not_gr_zero_hmset by blast

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

lemma gr_zeroI_hmset:  $(M = 0 \implies \text{False}) \implies 0 < M$  for  $M :: \text{hmultiset}$ 
  using not_gr_zero_hmset by blast

lemma gr_implies_not_zero_hmset:  $M < N \implies N \neq 0$  for  $M N :: \text{hmultiset}$ 
  by auto

lemma add_eq_0_iff_both_eq_0_hmset[simp]:  $M + N = 0 \longleftrightarrow M = 0 \wedge N = 0$  for  $M N :: \text{hmultiset}$ 
  by (intro add_nonneg_eq_0_iff zero_le_hmset)

lemma trans_less_add1_hmset:  $i < j \implies i < j + m$  for  $i j m :: \text{hmultiset}$ 
  by (metis add_increasing2 leD le_less not_gr_zero_hmset)

lemma trans_less_add2_hmset:  $i < j \implies i < m + j$  for  $i j m :: \text{hmultiset}$ 
  by (simp add: add_commute trans_less_add1_hmset)

lemma trans_le_add1_hmset:  $i \leq j \implies i \leq j + m$  for  $i j m :: \text{hmultiset}$ 
  by (simp add: add_increasing2)

lemma trans_le_add2_hmset:  $i \leq j \implies i \leq m + j$  for  $i j m :: \text{hmultiset}$ 
  by (simp add: add_increasing)

lemma diff_le_self_hmset:  $m - n \leq m$  for  $m n :: \text{hmultiset}$ 
  by (metis add_commute add.right_neutral diff_add_zero diff_diff_add_hmset
    le_minus_plus_same_hmset)

```

end

## 6 Signed Hereditar(il)y (Finite) Multisets

```

theory Signed_Hereditary_Multiset
imports Signed_Multiset Hereditary_Multiset
begin

```

### 6.1 Type Definition

```

typedef zhmultiset = UNIV :: hmultiset zmultiset set
  morphisms zhmssetmset ZHMSet
  by simp

lemmas ZHMSet_inverse[simp] = ZHMSet_inverse[OF UNIV_I]
lemmas ZHMSet_inject[simp] = ZHMSet_inject[OF UNIV_I UNIV_I]

declare
  zhmssetmset_inverse [simp]
  zhmssetmset_inject [simp]

setup-lifting type_definition_zhmultiset

```

## 6.2 Multiset Order

**instantiation** *zhmultiset* :: *linorder*  
**begin**

**lift-definition** *less\_zhmultiset* :: *zhmultiset*  $\Rightarrow$  *zhmultiset*  $\Rightarrow$  *bool* **is** ( $<$ ) .  
**lift-definition** *less\_eq\_zhmultiset* :: *zhmultiset*  $\Rightarrow$  *zhmultiset*  $\Rightarrow$  *bool* **is** ( $\leq$ ) .

**instance**  
**by** (*intro\_classes*; *transfer*) *auto*

**end**

**lemmas** *ZHMSet\_less[simp]* = *less\_zhmultiset.abs\_eq*  
**lemmas** *ZHMSet\_le[simp]* = *less\_eq\_zhmultiset.abs\_eq*  
**lemmas** *zhmsetmset\_less[simp]* = *less\_zhmultiset.rep\_eq[symmetric]*  
**lemmas** *zhmsetmset\_le[simp]* = *less\_eq\_zhmultiset.rep\_eq[symmetric]*

## 6.3 Embedding and Projections of Syntactic Ordinals

**abbreviation** *zhmset\_of* :: *hmultiset*  $\Rightarrow$  *zhmultiset* **where**  
*zhmset\_of* *M*  $\equiv$  *ZHMSet* (*zmset\_of* (*hmssetmset* *M*))

**lemma** *zhmset\_of\_inject[simp]*: *zhmset\_of* *M* = *zhmset\_of* *N*  $\longleftrightarrow$  *M* = *N*  
**by** *simp*

**lemma** *zhmset\_of\_less*: *zhmset\_of* *M* < *zhmset\_of* *N*  $\longleftrightarrow$  *M* < *N*  
**by** (*simp* *add*: *zmset\_of\_less*)

**lemma** *zhmset\_of\_le*: *zhmset\_of* *M*  $\leq$  *zhmset\_of* *N*  $\longleftrightarrow$  *M*  $\leq$  *N*  
**by** (*simp* *add*: *zmset\_of\_le*)

**abbreviation** *hmset\_pos* :: *zhmultiset*  $\Rightarrow$  *hmultiset* **where**  
*hmset\_pos* *M*  $\equiv$  *HMSet* (*mset\_pos* (*zhmsetmset* *M*))

**abbreviation** *hmset\_neg* :: *zhmultiset*  $\Rightarrow$  *hmultiset* **where**  
*hmset\_neg* *M*  $\equiv$  *HMSet* (*mset\_neg* (*zhmsetmset* *M*))

## 6.4 Disjoint Union and Difference

**instantiation** *zhmultiset* :: *cancel\_comm\_monoid\_add*  
**begin**

**lift-definition** *zero\_zhmultiset* :: *zhmultiset* **is**  $\{\#\}_z$  .

**lift-definition** *plus\_zhmultiset* :: *zhmultiset*  $\Rightarrow$  *zhmultiset*  $\Rightarrow$  *zhmultiset* **is**  
 $\lambda A B. A + B$  .

**lift-definition** *minus\_zhmultiset* :: *zhmultiset*  $\Rightarrow$  *zhmultiset*  $\Rightarrow$  *zhmultiset* **is**  
 $\lambda A B. A - B$  .

**lemmas** *ZHMSet\_plus* = *plus\_zhmultiset.abs\_eq[symmetric]*  
**lemmas** *ZHMSet\_diff* = *minus\_zhmultiset.abs\_eq[symmetric]*  
**lemmas** *zhmsetmset\_plus* = *plus\_zhmultiset.rep\_eq*  
**lemmas** *zhmsetmset\_diff* = *minus\_zhmultiset.rep\_eq*

**lemma** *zhmset\_of\_plus*: *zhmset\_of* (*A* + *B*) = *zhmset\_of* *A* + *zhmset\_of* *B*  
**by** (*simp* *add*: *hmssetmset\_plus* *ZHMSet\_plus* *zmset\_of\_plus*)

**lemma** *hmssetmset\_0[simp]*: *hmssetmset* 0 =  $\{\#\}$   
**by** (*rule* *hmultiset.inject[THEN iffD1]*) (*simp* *add*: *zero\_hmultiset\_def*)

**instance**  
**by** (*intro\_classes*; *transfer*) (*auto* *intro*: *mult.assoc* *add.commute*)

end

**lemma** *zhmset\_of\_0*:  $zhmset\_of\ 0 = 0$   
by (*simp add: zero\_zhmultiset\_def*)

**lemma** *hmset\_pos\_plus*:  
 $hmset\_pos\ (A + B) = (hmset\_pos\ A - hmset\_neg\ B) + (hmset\_pos\ B - hmset\_neg\ A)$   
by (*simp add: HMSet\_diff HMSet\_plus zhmsetmset\_plus*)

**lemma** *hmset\_neg\_plus*:  
 $hmset\_neg\ (A + B) = (hmset\_neg\ A - hmset\_pos\ B) + (hmset\_neg\ B - hmset\_pos\ A)$   
by (*simp add: HMSet\_diff HMSet\_plus zhmsetmset\_plus*)

**lemma** *zhmset\_pos\_neg\_partition*:  $M = zhmset\_of\ (hmset\_pos\ M) - zhmset\_of\ (hmset\_neg\ M)$   
by (*cases M, simp add: ZHMSet\_diff[symmetric], rule mset\_pos\_neg\_partition*)

**lemma** *zhmset\_pos\_as\_neg*:  $zhmset\_of\ (hmset\_pos\ M) = zhmset\_of\ (hmset\_neg\ M) + M$   
using *mset\_pos\_as\_neg zhmsetmset\_plus zhmsetmset\_inject* by *fastforce*

**lemma** *zhmset\_neg\_as\_pos*:  $zhmset\_of\ (hmset\_neg\ M) = zhmset\_of\ (hmset\_pos\ M) - M$   
using *zhmsetmset\_diff mset\_neg\_as\_pos zhmsetmset\_inject* by *fastforce*

**lemma** *hmset\_pos\_neg\_dual*:  
 $hmset\_pos\ a + hmset\_pos\ b + (hmset\_neg\ a - hmset\_pos\ b) + (hmset\_neg\ b - hmset\_pos\ a) =$   
 $hmset\_neg\ a + hmset\_neg\ b + (hmset\_pos\ a - hmset\_neg\ b) + (hmset\_pos\ b - hmset\_neg\ a)$   
by (*simp add: HMSet\_plus[symmetric] HMSet\_diff[symmetric]*) (*rule mset\_pos\_neg\_dual*)

**lemma** *zhmset\_of\_sum\_list*:  $zhmset\_of\ (sum\_list\ Ms) = sum\_list\ (map\ zhmset\_of\ Ms)$   
by (*induct Ms*) (*auto simp: zero\_zhmultiset\_def zhmset\_of\_plus*)

**lemma** *less\_hmset\_zhmsetE*:  
assumes *m\_lt\_n*:  $M < N$   
obtains *A B C* **where**  $M = zhmset\_of\ A + C$  **and**  $N = zhmset\_of\ B + C$  **and**  $A < B$   
by (*rule less\_mset\_zmsetE[OF m\_lt\_n[folded zhmsetmset\_less]]*)  
(*metis hmsetmset\_less hmultiset.sel ZHMSet\_plus zhmsetmset\_inverse*)

**lemma** *less\_eq\_hmset\_zhmsetE*:  
assumes *m\_le\_n*:  $M \leq N$   
obtains *A B C* **where**  $M = zhmset\_of\ A + C$  **and**  $N = zhmset\_of\ B + C$  **and**  $A \leq B$   
by (*rule less\_eq\_mset\_zmsetE[OF m\_le\_n[folded zhmsetmset\_le]]*)  
(*metis hmsetmset\_le hmultiset.sel ZHMSet\_plus zhmsetmset\_inverse*)

**instantiation** *zhmultiset* :: *ab\_group\_add*  
**begin**

**lift-definition** *uminus\_zhmultiset* ::  $zhmultiset \Rightarrow zhmultiset$  **is**  $\lambda A. - A$  .

**lemmas** *ZHMSet\_uminus* = *uminus\_zhmultiset.abs\_eq[symmetric]*

**lemmas** *zhmsetmset\_uminus* = *uminus\_zhmultiset.rep\_eq*

**instance**  
by (*intro\_classes; transfer; simp*)

end

## 6.5 Infimum and Supremum

**instance** *zhmultiset* :: *ordered\_cancel\_comm\_monoid\_add*  
by (*intro\_classes; transfer*) (*auto simp: add\_left\_mono*)

**instance** *zhmultiset* :: *ordered\_ab\_group\_add*  
by (*intro\_classes; transfer; simp*)

```

instantiation zhmultiset :: distrib_lattice
begin

definition inf_zhmultiset :: zhmultiset  $\Rightarrow$  zhmultiset  $\Rightarrow$  zhmultiset where
  inf_zhmultiset A B = (if A < B then A else B)

definition sup_zhmultiset :: zhmultiset  $\Rightarrow$  zhmultiset  $\Rightarrow$  zhmultiset where
  sup_zhmultiset A B = (if B > A then B else A)

instance
  by intro_classes (auto simp: inf_zhmultiset_def sup_zhmultiset_def)

end

end

```

## 7 Syntactic Ordinals in Cantor Normal Form

```

theory Syntactic_Ordinal
imports Hereditary_Multiset HOL-Library.Product_Order HOL-Library.Extended_Nat
begin

```

### 7.1 Natural (Hessenberg) Product

```

instantiation hmultiset :: comm_semiring_1
begin

abbreviation  $\omega\_exp$  :: hmultiset  $\Rightarrow$  hmultiset ( $\omega^\wedge$ ) where
   $\omega^\wedge \equiv \lambda m. HMSet \{ \#m \# \}$ 

definition one_hmultiset :: hmultiset where
  1 =  $\omega^\wedge 0$ 

abbreviation  $\omega$  :: hmultiset where
   $\omega \equiv \omega^\wedge 1$ 

definition times_hmultiset :: hmultiset  $\Rightarrow$  hmultiset  $\Rightarrow$  hmultiset where
  A * B = HMSet (image_mset (case_prod (+)) (hmsetmset A  $\times$   $\#$  hmsetmset B))

lemma hmsetmset_times:
  hmsetmset (m * n) = image_mset (case_prod (+)) (hmsetmset m  $\times$   $\#$  hmsetmset n)
  unfolding times_hmultiset_def by simp

instance
proof (intro_classes, goal_cases assoc comm one distrib_plus zeroL zeroR zero_one)
  case (assoc a b c)
  thus ?case
    by (auto simp: times_hmultiset_def Times_mset_image_mset1 Times_mset_image_mset2
      Times_mset_assoc ac_simps intro: multiset.map_cong)
next
  case (comm a b)
  thus ?case
    unfolding times_hmultiset_def
    by (subst product_swap_mset[symmetric]) (auto simp: ac_simps intro: multiset.map_cong)
next
  case (one a)
  thus ?case
    by (auto simp: one_hmultiset_def times_hmultiset_def Times_mset_single_left)
next
  case (distrib_plus a b c)
  thus ?case
    by (auto simp: plus_hmultiset_def times_hmultiset_def)
next

```

```

case (zeroL a)
thus ?case
  by (auto simp: times_hmultiset_def)
next
case (zeroR a)
thus ?case
  by (auto simp: times_hmultiset_def)
next
case zero_one
thus ?case
  by (auto simp: one_hmultiset_def)
qed

```

end

```

lemma empty_times_left_hmset[simp]: HMSet {#} * M = 0
  by (simp add: times_hmultiset_def)

```

```

lemma empty_times_right_hmset[simp]: M * HMSet {#} = 0
  by (metis mult_zero_right zero_hmultiset_def)

```

```

lemma singleton_times_left_hmset[simp]:  $\omega^{\wedge}M * N = \text{HMSet } (\text{image\_mset } ((+) M) (\text{hmsetmset } N))$ 
  by (simp add: times_hmultiset_def Times_mset_single_left)

```

```

lemma singleton_times_right_hmset[simp]:  $N * \omega^{\wedge}M = \text{HMSet } (\text{image\_mset } ((+) M) (\text{hmsetmset } N))$ 
  by (metis mult.commute singleton_times_left_hmset)

```

## 7.2 Inequalities

```

definition plus_nmultiset :: unit nmultiset  $\Rightarrow$  unit nmultiset  $\Rightarrow$  unit nmultiset where
  plus_nmultiset X Y = Rep_hmultiset (Abs_hmultiset X + Abs_hmultiset Y)

```

```

lemma plus_nmultiset_mono:
  assumes less: (X, Y) < (X', Y') and no_elem: no_elem X no_elem Y no_elem X' no_elem Y'
  shows plus_nmultiset X Y < plus_nmultiset X' Y'
  using less[unfolded less_le_not_le] no_elem
  by (auto simp: plus_nmultiset_def plus_hmultiset_def less_multiset_extDM_less less_eq_nmultiset_def
    union_less_mono type_definition.Abs_inverse[OF type_definition_hmultiset, simplified]
    elim!: no_elem.cases)

```

```

lemma plus_hmultiset_transfer[transfer_rule]:
  (rel_fun pcr_hmultiset (rel_fun pcr_hmultiset pcr_hmultiset)) plus_nmultiset (+)
  unfolding rel_fun_def plus_nmultiset_def pcr_hmultiset_def nmultiset.rel_eq eq_OO cr_hmultiset_def
  by (auto simp: type_definition.Rep_inverse[OF type_definition_hmultiset])

```

```

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

```

proof -

```

obtain Y X where
  Y_nemp: Y  $\neq$  {#} and Y_sub_N: Y  $\subseteq$  # N and M_eq: M = N - Y + X and
  ex_Y:  $\forall x. x \in \# X \longrightarrow (\exists y. y \in \# Y \wedge x < y)$ 
  using less[unfolded less_multisetDM] by blast

```

```

let ?X = X  $\times$  # Z
let ?Y = Y  $\times$  # Z

```

```

show ?thesis
  unfolding less_multisetDM
proof (intro exI conjI)
  show M  $\times$  # Z = N  $\times$  # Z - ?Y + ?X
    unfolding M_eq by (auto simp: Sigma_mset_Diff_distrib1)
next
  obtain y where y:  $\forall x. x \in \# X \longrightarrow y x \in \# Y \wedge x < y x$ 

```

```

using ex_Y by moura

show  $\forall x. x \in\# ?X \longrightarrow (\exists y. y \in\# ?Y \wedge x < y)$ 
proof (intro allI impI)
  fix x
  assume x  $\in\# ?X$ 
  thus  $\exists y. y \in\# ?Y \wedge x < y$ 
    using y by (intro exI[of _ (y (fst x), snd x)]) (auto simp: less_le_not_le)
qed
qed (auto simp: Z_nemp Y_nemp Y_sub_N Sigma_mset_mono)
qed

lemma times_hmultiset_monoL:
  a < b  $\implies$  0 < c  $\implies$  a * c < b * c for a b c :: hmultiset
by (cases a, cases b, cases c, hypsubst_thin,
  unfold times_hmultiset_def zero_hmultiset_def hmultiset.sel, transfer,
  auto simp: less_multiset_extDM_less multiset.pred_set
  intro!: image_mset_strict_mono Times_mset_monoL elim!: plus_nmultiset_mono)

instance hmultiset :: linordered_semiring_strict
by intro_classes (subst (1 2) mult.commute, (fact times_hmultiset_monoL)+)

lemma mult_le_mono1_hmset: i  $\leq$  j  $\implies$  i * k  $\leq$  j * k for i j k :: hmultiset
by (simp add: mult_right_mono)

lemma mult_le_mono2_hmset: i  $\leq$  j  $\implies$  k * i  $\leq$  k * j for i j k :: hmultiset
by (simp add: mult_left_mono)

lemma mult_le_mono_hmset: i  $\leq$  j  $\implies$  k  $\leq$  l  $\implies$  i * k  $\leq$  j * l for i j k l :: hmultiset
by (simp add: mult_mono)

lemma less_iff_add1_le_hmset: m < n  $\iff$  m + 1  $\leq$  n for m n :: hmultiset
proof (cases m n rule: hmultiset.exhaust[case_product hmultiset.exhaust])
  case (HMSet_HMSet m0 n0)
  note m = this(1) and n = this(2)

  show ?thesis
proof (simp add: m n one_hmultiset_def plus_hmultiset_def order.order_iff_strict
  less_multiset_extDM_less, intro iffI)
  assume m0_lt_n0: m0 < n0
  note
    m0_ne_n0 = m0_lt_n0[unfolded less_multiset_HO, THEN conjunct1] and
    ex_n0_gt_m0 = m0_lt_n0[unfolded less_multiset_HO, THEN conjunct2, rule_format]

  {
  assume zero_m0_gt_n0: add_mset 0 m0 > n0
  note
    n0_ne_0m0 = zero_m0_gt_n0[unfolded less_multiset_HO, THEN conjunct1] and
    ex_0m0_gt_n0 = zero_m0_gt_n0[unfolded less_multiset_HO, THEN conjunct2, rule_format]

  {
  fix y
  assume m0y_lt_n0y: count m0 y < count n0 y

  have  $\exists x > y. \text{count } n0 \ x < \text{count } m0 \ x$ 
  proof (cases count (add_mset 0 m0) y < count n0 y)
    case True
    then obtain aa where
      aa_gt_y: aa > y and
      count_n0aa_lt_count_0m0aa: count n0 aa < count (add_mset 0 m0) aa
    using ex_0m0_gt_n0 by blast
  have aa  $\neq$  0
  by (rule gr_implies_not_zero_hmset[OF aa_gt_y])
  }
  }
  }
  }

```



```

    hence count (add_mset 0 m0) aa = count m0 aa
      by simp
    thus ?thesis
      using count_n0aa_lt_count_0m0aa aa_gt_y by auto
  next
    case not_0m0_y_lt_n0y: False
    hence y_eq_0: y = 0
      by (metis count_add_mset m0y_lt_n0y)
    have sm0y_eq_n0y: Suc (count m0 y) = count n0 y
      using m0y_lt_n0y not_0m0_y_lt_n0y count_add_mset[of 0 _ 0] unfolding y_eq_0 by simp

    obtain bb where count n0 bb < count (add_mset 0 m0) bb
      using lt_imp_ex_count_lt[OF zero_m0_gt_n0] by blast
    hence n0bb_lt_m0bb: count n0 bb < count m0 bb
      unfolding count_add_mset by (metis (full_types) less_irrefl_nat sm0y_eq_n0y y_eq_0)
    hence bb ≠ 0
      using sm0y_eq_n0y y_eq_0 by auto
    thus ?thesis
      unfolding y_eq_0 using n0bb_lt_m0bb not_gr_zero_hmset by blast
  qed
}
hence n0 < m0
  unfolding less_multiset_HO using m0_ne_n0 by blast
hence False
  using m0_lt_n0 by simp
}
thus add_mset 0 m0 < n0 ∨ add_mset 0 m0 = n0
  using antisym_conv3 by blast
next
  assume add_mset 0 m0 < n0 ∨ add_mset 0 m0 = n0
  thus m0 < n0
    using dual_order.strict_trans le_multiset_right_total by blast
  qed
qed

lemma zero_less_iff_1_le_hmset: 0 < n ↔ 1 ≤ n for n :: hmultiset
  by (rule less_iff_add1_le_hmset[of 0, simplified])

lemma less_add_1_iff_le_hmset: m < n + 1 ↔ m ≤ n for m n :: hmultiset
  by (rule less_iff_add1_le_hmset[of m n + 1, simplified])

instance hmultiset :: ordered_cancel_comm_semiring
  by intro_classes (simp add: mult_le_mono2_hmset)

instance hmultiset :: zero_less_one
  by intro_classes (simp add: zero_less_iff_neq_zero_hmset)

instance hmultiset :: linordered_semiring_1_strict
  by intro_classes

instance hmultiset :: bounded_lattice_bot
  by intro_classes

instance hmultiset :: linordered_nonzero_semiring
  by intro_classes simp

instance hmultiset :: semiring_no_zero_divisors
  by intro_classes (use mult_pos_pos not_gr_zero_hmset in blast)

lemma lt_1_iff_eq_0_hmset: M < 1 ↔ M = 0 for M :: hmultiset
  by (simp add: less_iff_add1_le_hmset)

lemma zero_less_mult_iff_hmset[simp]: 0 < m * n ↔ 0 < m ∧ 0 < n for m n :: hmultiset

```

using `mult_eq_0_iff not_gr_zero_hmset` by `blast`

**lemma** `one_le_mult_iff_hmset[simp]`:  $1 \leq m * n \iff 1 \leq m \wedge 1 \leq n$  for  $m n :: \text{hmultiset}$   
by `(metis lt_1_iff_eq_0_hmset mult_eq_0_iff not_le)`

**lemma** `mult_less_cancel2_hmset[simp]`:  $m * k < n * k \iff 0 < k \wedge m < n$  for  $k m n :: \text{hmultiset}$   
by `(metis gr_zeroI_hmset leD leI le_cases mult_right_mono mult_zero_right times_hmultiset_monoL)`

**lemma** `mult_less_cancel1_hmset[simp]`:  $k * m < k * n \iff 0 < k \wedge m < n$  for  $k m n :: \text{hmultiset}$   
by `(simp add: mult.commute[of k])`

**lemma** `mult_le_cancel1_hmset[simp]`:  $k * m \leq k * n \iff (0 < k \implies m \leq n)$  for  $k m n :: \text{hmultiset}$   
by `(simp add: linorder_not_less[symmetric], auto)`

**lemma** `mult_le_cancel2_hmset[simp]`:  $m * k \leq n * k \iff (0 < k \implies m \leq n)$  for  $k m n :: \text{hmultiset}$   
by `(simp add: linorder_not_less[symmetric], auto)`

**lemma** `mult_le_cancel_left1_hmset`:  $y > 0 \implies x \leq x * y$  for  $x y :: \text{hmultiset}$   
by `(metis zero_less_iff_1_le_hmset mult.commute mult.left_neutral mult_le_cancel2_hmset)`

**lemma** `mult_le_cancel_left2_hmset`:  $y \leq 1 \implies x * y \leq x$  for  $x y :: \text{hmultiset}$   
by `(metis mult.commute mult.left_neutral mult_le_cancel2_hmset)`

**lemma** `mult_le_cancel_right1_hmset`:  $y > 0 \implies x \leq y * x$  for  $x y :: \text{hmultiset}$   
by `(subst mult.commute) (fact mult_le_cancel_left1_hmset)`

**lemma** `mult_le_cancel_right2_hmset`:  $y \leq 1 \implies y * x \leq x$  for  $x y :: \text{hmultiset}$   
by `(subst mult.commute) (fact mult_le_cancel_left2_hmset)`

**lemma** `le_square_hmset`:  $m \leq m * m$  for  $m :: \text{hmultiset}$   
using `mult_le_cancel_left1_hmset` by `force`

**lemma** `le_cube_hmset`:  $m \leq m * (m * m)$  for  $m :: \text{hmultiset}$   
using `mult_le_cancel_left1_hmset` by `force`

**lemma**  
`less_imp_minus_plus_hmset`:  $m < n \implies k < k - m + n$  and  
`le_imp_minus_plus_hmset`:  $m \leq n \implies k \leq k - m + n$  for  $k m n :: \text{hmultiset}$   
by `(meson add_less_cancel_left leD le_minus_plus_same_hmset less_le_trans not_le_imp_less)+`

**lemma** `gt_0_lt_mult_gt_1_hmset`:  
fixes  $m n :: \text{hmultiset}$   
assumes  $m > 0$  and  $n > 1$   
shows  $m < m * n$   
using `assms` by `(metis mult.right_neutral mult_less_cancel1_hmset)`

**instance** `hmultiset` :: `linordered_comm_semiring_strict`  
by `intro_classes simp`

### 7.3 Embedding of Natural Numbers

**lemma** `of_nat_hmset`:  $\text{of\_nat } n = \text{HMSet } (\text{replicate\_mset } n \ 0)$   
by `(induct n) (auto simp: zero_hmultiset_def one_hmultiset_def plus_hmultiset_def)`

**lemma** `of_nat_inject_hmset[simp]`:  $(\text{of\_nat } m :: \text{hmultiset}) = \text{of\_nat } n \iff m = n$   
unfolding `of_nat_hmset` by `simp`

**lemma** `of_nat_minus_hmset`:  $\text{of\_nat } (m - n) = (\text{of\_nat } m :: \text{hmultiset}) - \text{of\_nat } n$   
unfolding `of_nat_hmset` `minus_hmultiset_def` by `simp`

**lemma** `plus_of_nat_plus_of_nat_hmset`:  
 $k + \text{of\_nat } m + \text{of\_nat } n = k + \text{of\_nat } (m + n)$  for  $k :: \text{hmultiset}$   
by `simp`

**lemma** *plus\_of\_nat\_minus\_of\_nat\_hmset*:  
**fixes**  $k :: \text{hmultiset}$   
**assumes**  $n \leq m$   
**shows**  $k + \text{of\_nat } m - \text{of\_nat } n = k + \text{of\_nat } (m - n)$   
**using** *assms* **by** (*metis add.left\_commute add\_diff\_cancel\_left' le\_add\_diff\_inverse of\_nat\_add*)

**lemma** *of\_nat\_lt\_omega[simp]*:  $\text{of\_nat } n < \omega$   
**by** (*auto simp: of\_nat\_hmset zero\_less\_iff\_neq\_zero\_hmset less\_multiset\_ext\_DM\_less*)

**lemma** *of\_nat\_ne\_omega[simp]*:  $\text{of\_nat } n \neq \omega$   
**by** (*simp add: neq\_iff*)

**lemma** *of\_nat\_less\_hmset[simp]*:  $(\text{of\_nat } M :: \text{hmultiset}) < \text{of\_nat } N \longleftrightarrow M < N$   
**unfolding** *of\_nat\_hmset less\_multiset\_ext\_DM\_less* **by** *simp*

**lemma** *of\_nat\_le\_hmset[simp]*:  $(\text{of\_nat } M :: \text{hmultiset}) \leq \text{of\_nat } N \longleftrightarrow M \leq N$   
**unfolding** *of\_nat\_hmset order\_le\_less less\_multiset\_ext\_DM\_less* **by** *simp*

**lemma** *of\_nat\_times\_omega\_exp*:  $\text{of\_nat } n * \omega^{\hat{m}} = \text{HMSet } (\text{replicate\_mset } n \ m)$   
**by** (*induct n*) (*simp\_all add: hmsetmset\_plus\_one\_hmultiset\_def*)

**lemma** *omega\_exp\_times\_of\_nat*:  $\omega^{\hat{m}} * \text{of\_nat } n = \text{HMSet } (\text{replicate\_mset } n \ m)$   
**using** *of\_nat\_times\_omega\_exp* **by** *simp*

## 7.4 Embedding of Extended Natural Numbers

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

**lemma** *hmset\_of\_enat\_0[simp]*:  $\text{hmset\_of\_enat } 0 = 0$   
**by** (*simp add: zero\_enat\_def*)

**lemma** *hmset\_of\_enat\_1[simp]*:  $\text{hmset\_of\_enat } 1 = 1$   
**by** (*simp add: one\_enat\_def del: One\_nat\_def*)

**lemma** *hmset\_of\_enat\_of\_nat[simp]*:  $\text{hmset\_of\_enat } (\text{of\_nat } n) = \text{of\_nat } n$   
**using** *of\_nat\_eq\_enat* **by** *auto*

**lemma** *hmset\_of\_enat\_numeral[simp]*:  $\text{hmset\_of\_enat } (\text{numeral } n) = \text{numeral } n$   
**by** (*simp add: numeral\_eq\_enat*)

**lemma** *hmset\_of\_enat\_le\_omega[simp]*:  $\text{hmset\_of\_enat } n \leq \omega$   
**using** *of\_nat\_lt\_omega* [*THEN less\_imp\_le*] **by** (*cases n*) *auto*

**lemma** *hmset\_of\_enat\_eq\_omega\_iff[simp]*:  $\text{hmset\_of\_enat } n = \omega \longleftrightarrow n = \infty$   
**by** (*cases n*) *auto*

## 7.5 Head Omega

**definition** *head\_omega* ::  $\text{hmultiset} \Rightarrow \text{hmultiset}$  **where**  
*head\_omega*  $M = (\text{if } M = 0 \text{ then } 0 \text{ else } \omega^{\wedge}(\text{Max } (\text{set\_mset } (\text{hmsetmset } M))))$

**lemma** *head\_omega\_subseteq*:  $\text{hmsetmset } (\text{head\_omega } M) \subseteq\# \text{hmsetmset } M$   
**unfolding** *head\_omega\_def* **by** *simp*

**lemma** *head\_omega\_eq\_0\_iff[simp]*:  $\text{head\_omega } m = 0 \longleftrightarrow m = 0$   
**unfolding** *head\_omega\_def zero\_hmultiset\_def* **by** *simp*

**lemma** *head\_omega\_0[simp]*:  $\text{head\_omega } 0 = 0$   
**by** *simp*

**lemma** *head\_omega\_1[simp]*:  $\text{head\_omega } 1 = 1$   
**unfolding** *head\_omega\_def one\_hmultiset\_def* **by** *simp*

**lemma** *head\_ω\_of\_nat*[simp]:  $\text{head}_\omega (\text{of\_nat } n) = (\text{if } n = 0 \text{ then } 0 \text{ else } 1)$   
**unfolding** *head\_ω\_def one\_hmultiset\_def of\_nat\_hmset* **by** *simp*

**lemma** *head\_ω\_numeral*[simp]:  $\text{head}_\omega (\text{numeral } n) = 1$   
**by** (*metis head\_ω\_of\_nat of\_nat\_numeral zero\_neq\_numeral*)

**lemma** *head\_ω\_ω*[simp]:  $\text{head}_\omega \omega = \omega$   
**unfolding** *head\_ω\_def* **by** *simp*

**lemma** *le\_imp\_head\_ω\_le*:  
**assumes** *m\_le\_n*:  $m \leq n$   
**shows**  $\text{head}_\omega m \leq \text{head}_\omega n$

**proof** –

**have** *le\_in\_le\_max*:  $\bigwedge a M N. M \leq N \implies a \in \# M \implies a \leq \text{Max} (\text{set\_mset } N)$   
**by** (*metis (no\_types) Max\_ge finite\_set\_mset le\_less less\_eq\_multiset\_HO linorder\_not\_less mem\_Collect\_eq neq0\_conv order\_trans set\_mset\_def*)

**show** *?thesis*

**using** *m\_le\_n* **unfolding** *head\_ω\_def*

**by** (*cases m, cases n,*  
*auto simp del: hmsetmset\_le simp: head\_ω\_def hmsetmset\_le[symmetric] zero\_hmultiset\_def,*  
*metis Max\_in dual\_order.antisym finite\_set\_mset le\_in\_le\_max le\_less set\_mset\_eq\_empty\_iff*)

**qed**

**lemma** *head\_ω\_lt\_imp\_lt*:  $\text{head}_\omega m < \text{head}_\omega n \implies m < n$   
**unfolding** *head\_ω\_def hmsetmset\_less[symmetric]*  
**by** (*rule all\_lt\_Max\_imp\_lt\_mset, auto simp: zero\_hmultiset\_def split: if\_splits*)

**lemma** *head\_ω\_plus*[simp]:  $\text{head}_\omega (m + n) = \sup (\text{head}_\omega m) (\text{head}_\omega n)$

**proof** (*cases m n rule: hmultiset.exhaust[case\_product hmultiset.exhaust]*)

**case** *m\_n*: (*HMSet\_HMSet M N*)

**show** *?thesis*

**proof** (*cases Max\_mset M < Max\_mset N*)

**case** *True*

**thus** *?thesis*

**unfolding** *m\_n head\_ω\_def sup\_hmultiset\_def zero\_hmultiset\_def plus\_hmultiset\_def*

**by** (*simp add: Max.union max\_def dual\_order.strict\_implies\_order*)

**next**

**case** *False*

**thus** *?thesis*

**unfolding** *m\_n head\_ω\_def sup\_hmultiset\_def zero\_hmultiset\_def plus\_hmultiset\_def*

**by** *simp (metis False Max.union finite\_set\_mset leI max\_def set\_mset\_eq\_empty\_iff sup commute)*

**qed**

**qed**

**lemma** *head\_ω\_times*[simp]:  $\text{head}_\omega (m * n) = \text{head}_\omega m * \text{head}_\omega n$

**proof** (*cases m = 0 ∨ n = 0*)

**case** *False*

**hence** *m\_nz*:  $m \neq 0$  **and** *n\_nz*:  $n \neq 0$

**by** *simp+*

**define**  $\delta$  **where**  $\delta = \text{hmsetmset } m$

**define**  $\varepsilon$  **where**  $\varepsilon = \text{hmsetmset } n$

**have** *δ\_nemp*:  $\delta \neq \{\#\}$

**unfolding** *δ\_def* **using** *m\_nz* **by** *simp*

**have** *ε\_nemp*:  $\varepsilon \neq \{\#\}$

**unfolding** *ε\_def* **using** *n\_nz* **by** *simp*

**let** *?D* = *set\_mset*  $\delta$

**let** *?E* = *set\_mset*  $\varepsilon$

**let** *?DE* =  $\{z. \exists x \in ?D. \exists y \in ?E. z = x + y\}$

```

have max_D_in:  $Max\ ?D \in\ ?D$ 
  using  $\delta\_nemp$  by simp
have max_E_in:  $Max\ ?E \in\ ?E$ 
  using  $\varepsilon\_nemp$  by simp

have  $Max\ ?DE = Max\ ?D + Max\ ?E$ 
proof (rule order_antisym, goal_cases le ge)
  case le
  have  $\bigwedge x\ y. x \in\ ?D \implies y \in\ ?E \implies x + y \leq Max\ ?D + Max\ ?E$ 
    by (simp add: add_mono)
  hence mem_imp_le:  $\bigwedge z. z \in\ ?DE \implies z \leq Max\ ?D + Max\ ?E$ 
    by auto
  show ?case
    by (intro mem_imp_le Max_in, simp, use  $\delta\_nemp\ \varepsilon\_nemp$  in fast)
next
  case ge
  have  $\{z. \exists x \in\ \{Max\ ?D\}. \exists y \in\ \{Max\ ?E\}. z = x + y\} \subseteq \{z. \exists x \in\ \#\ \delta. \exists y \in\ \#\ \varepsilon. z = x + y\}$ 
    using max_D_in max_E_in by fast
  thus ?case
    by simp
qed
thus ?thesis
  unfolding  $\delta\_def\ \varepsilon\_def$  by (auto simp: head_omega_def image_def times_hmultiset_def)
qed auto

```

## 7.6 More Inequalities and Some Equalities

```

lemma zero_lt_omega[simp]:  $0 < \omega$ 
  by (metis of_nat_lt_omega of_nat_0)

lemma one_lt_omega[simp]:  $1 < \omega$ 
  by (metis enat_defs(2) hmultiset_of_enat.simps(1) hmultiset_of_enat_1 of_nat_lt_omega)

lemma numeral_lt_omega[simp]: numeral  $n < \omega$ 
  using hmultiset_of_enat_numeral[symmetric] hmultiset_of_enat.simps(1) of_nat_lt_omega numeral_eq_enat
  by presburger

lemma one_le_omega[simp]:  $1 \leq \omega$ 
  by (simp add: less_imp_le)

lemma of_nat_le_omega[simp]: of_nat  $n \leq \omega$ 
  by (simp add: le_less)

lemma numeral_le_omega[simp]: numeral  $n \leq \omega$ 
  by (simp add: less_imp_le)

lemma not_omega_lt_1[simp]:  $\neg \omega < 1$ 
  by (simp add: not_less)

lemma not_omega_lt_of_nat[simp]:  $\neg \omega < \text{of\_nat } n$ 
  by (simp add: not_less)

lemma not_omega_lt_numeral[simp]:  $\neg \omega < \text{numeral } n$ 
  by (simp add: not_less)

lemma not_omega_le_1[simp]:  $\neg \omega \leq 1$ 
  by (simp add: not_le)

lemma not_omega_le_of_nat[simp]:  $\neg \omega \leq \text{of\_nat } n$ 
  by (simp add: not_le)

lemma not_omega_le_numeral[simp]:  $\neg \omega \leq \text{numeral } n$ 
  by (simp add: not_le)

```

**lemma** *zero\_ne\_ω*[simp]:  $0 \neq \omega$   
**by** (*metis not\_ω\_le\_1 zero\_le\_hmset*)

**lemma** *one\_ne\_ω*[simp]:  $1 \neq \omega$   
**using** *not\_ω\_le\_1* **by** *force*

**lemma** *numeral\_ne\_ω*[simp]: *numeral*  $n \neq \omega$   
**by** (*metis not\_ω\_le\_numeral numeral\_le\_ω*)

**lemma**  
*ω\_ne\_0*[simp]:  $\omega \neq 0$  **and**  
*ω\_ne\_1*[simp]:  $\omega \neq 1$  **and**  
*ω\_ne\_of\_nat*[simp]:  $\omega \neq \text{of\_nat } m$  **and**  
*ω\_ne\_numeral*[simp]:  $\omega \neq \text{numeral } n$   
**using** *zero\_ne\_ω one\_ne\_ω of\_nat\_ne\_ω numeral\_ne\_ω* **by** *metis+*

**lemma**  
*hmset\_of\_enat\_inject*[simp]: *hmset\_of\_enat*  $m = \text{hmset_of_enat } n \iff m = n$  **and**  
*hmset\_of\_enat\_less*[simp]: *hmset\_of\_enat*  $m < \text{hmset_of_enat } n \iff m < n$  **and**  
*hmset\_of\_enat\_le*[simp]: *hmset\_of\_enat*  $m \leq \text{hmset_of_enat } n \iff m \leq n$   
**by** (*cases m; cases n; simp*)**+**

**lemma** *lt\_ω\_imp\_ex\_of\_nat*:  
**assumes** *M\_lt\_ω*:  $M < \omega$   
**shows**  $\exists n. M = \text{of\_nat } n$

**proof** –

**have** *M\_lt\_single\_1*: *hmsetmset*  $M < \{\#1\#$   
**by** (*rule M\_lt\_ω[unfolded hmsetmset\_less[symmetric] less\_multiset\_ext\_DM\_less hmultiset.sel]*)

**have**  $N = 0$  **if**  $N \in \# \text{hmsetmset } M$  **for**  $N$

**proof** –

**have**  $0 < \text{count } (\text{hmsetmset } M) \ N$

**using** *that* **by** *auto*

**hence**  $N < 1$

**by** (*metis (no\_types) M\_lt\_single\_1 count\_single gr\_implies\_not0 less\_eq\_multiset\_HO less\_one neq\_iff\_not\_le*)

**thus** *?thesis*

**by** (*simp add: lt\_1\_iff\_eq\_0\_hmset*)

**qed**

**then obtain**  $n$  **where** *hmmM*:  $M = \text{HMSet } (\text{replicate\_mset } n \ 0)$

**using** *ex\_replicate\_mset\_if\_all\_elems\_eq* **by** (*metis hmultiset.collapse*)

**show** *?thesis*

**unfolding** *hmmM of\_nat\_hmset* **by** *blast*

**qed**

**lemma** *le\_ω\_imp\_ex\_hmset\_of\_enat*:

**assumes** *M\_le\_ω*:  $M \leq \omega$

**shows**  $\exists n. M = \text{hmset_of_enat } n$

**proof** (*cases M = ω*)

**case** *True*

**thus** *?thesis*

**by** (*metis hmset\_of\_enat.simps(2)*)

**next**

**case** *False*

**thus** *?thesis*

**using** *M\_le\_ω lt\_ω\_imp\_ex\_of\_nat* **by** (*metis hmset\_of\_enat.simps(1) le\_less*)

**qed**

**lemma** *lt\_ω\_lt\_ω\_imp\_times\_lt\_ω*:  $M < \omega \implies N < \omega \implies M * N < \omega$

**by** (*metis lt\_ω\_imp\_ex\_of\_nat of\_nat\_lt\_ω of\_nat\_mult*)

**lemma** *times\_ω\_minus\_of\_nat*[simp]:  $m * \omega - \text{of\_nat } n = m * \omega$

**by** (*auto intro!: Diff\_triv\_mset simp: times\_hmultiset\_def minus\_hmultiset\_def*)

*Times\_mset\_single\_right\_of\_nat\_hmset\_disjunct\_not\_in\_image\_def*)

**lemma** *times\_ω\_minus\_numeral[simp]*:  $m * \omega - \text{numeral } n = m * \omega$   
**by** (*metis of\_nat\_numeral times\_ω\_minus\_of\_nat*)

**lemma** *ω\_minus\_of\_nat[simp]*:  $\omega - \text{of\_nat } n = \omega$   
**using** *times\_ω\_minus\_of\_nat[of 1]* **by** (*metis mult.left\_neutral*)

**lemma** *ω\_minus\_1[simp]*:  $\omega - 1 = \omega$   
**using** *ω\_minus\_of\_nat[of 1]* **by** *simp*

**lemma** *ω\_minus\_numeral[simp]*:  $\omega - \text{numeral } n = \omega$   
**using** *times\_ω\_minus\_numeral[of 1]* **by** (*metis mult.left\_neutral*)

**lemma** *hmset\_of\_enat\_minus\_enat[simp]*:  $\text{hmset\_of\_enat } (m - \text{enat } n) = \text{hmset\_of\_enat } m - \text{of\_nat } n$   
**by** (*cases m*) (*auto simp: of\_nat\_minus\_hmset*)

**lemma** *of\_nat\_lt\_hmset\_of\_enat\_iff*:  $\text{of\_nat } m < \text{hmset\_of\_enat } n \iff \text{enat } m < n$   
**by** (*metis hmset\_of\_enat.simps(1) hmset\_of\_enat\_less*)

**lemma** *of\_nat\_le\_hmset\_of\_enat\_iff*:  $\text{of\_nat } m \leq \text{hmset\_of\_enat } n \iff \text{enat } m \leq n$   
**by** (*metis hmset\_of\_enat.simps(1) hmset\_of\_enat\_le*)

**lemma** *hmset\_of\_enat\_lt\_iff\_ne\_infinity*:  $\text{hmset\_of\_enat } x < \omega \iff x \neq \infty$   
**by** (*cases x; simp*)

**lemma** *minus\_diff\_sym\_hmset*:  $m - (m - n) = n - (n - m)$  **for**  $m \ n :: \text{hmultiset}$   
**unfolding** *minus\_hmultiset\_def* **by** *simp* (*metis multiset\_inter\_def subset\_mset.inf\_aci(1)*)

**lemma** *diff\_plus\_sym\_hmset*:  $(c - b) + b = (b - c) + c$  **for**  $b \ c :: \text{hmultiset}$

**proof** –

**have** *f1*:  $\bigwedge h \ ha :: \text{hmultiset}. h - (ha + h) = 0$   
**by** (*simp add: add.commute*)

**have** *f2*:  $\bigwedge h \ ha \ hb :: \text{hmultiset}. h + ha - (h - hb) = hb + ha - (hb - h)$   
**by** (*metis (no\_types) add\_diff\_cancel\_right minus\_diff\_sym\_hmset*)

**have**  $\bigwedge h \ ha \ hb :: \text{hmultiset}. h + (ha + hb) - hb = h + ha$   
**by** (*metis (no\_types) add.assoc add\_diff\_cancel\_right'*)

**then show** *?thesis*

**using** *f2 f1* **by** (*metis (no\_types) add.commute add.right\_neutral diff\_diff\_add\_hmset*)

**qed**

**lemma** *times\_diff\_plus\_sym\_hmset*:  $a * (c - b) + a * b = a * (b - c) + a * c$  **for**  $a \ b \ c :: \text{hmultiset}$   
**by** (*metis distrib\_left diff\_plus\_sym\_hmset*)

**lemma** *times\_of\_nat\_minus\_left*:

$(\text{of\_nat } m - \text{of\_nat } n) * l = \text{of\_nat } m * l - \text{of\_nat } n * l$  **for**  $l :: \text{hmultiset}$   
**by** (*induct n m rule: diff\_induct*) (*auto simp: ring\_distrib*)

**lemma** *times\_of\_nat\_minus\_right*:

$l * (\text{of\_nat } m - \text{of\_nat } n) = l * \text{of\_nat } m - l * \text{of\_nat } n$  **for**  $l :: \text{hmultiset}$   
**by** (*metis times\_of\_nat\_minus\_left mult.commute*)

**lemma** *lt\_ω\_imp\_times\_minus\_left*:  $m < \omega \implies n < \omega \implies (m - n) * l = m * l - n * l$   
**by** (*metis lt\_ω\_imp\_ex\_of\_nat times\_of\_nat\_minus\_left*)

**lemma** *lt\_ω\_imp\_times\_minus\_right*:  $m < \omega \implies n < \omega \implies l * (m - n) = l * m - l * n$   
**by** (*metis lt\_ω\_imp\_ex\_of\_nat times\_of\_nat\_minus\_right*)

**lemma** *hmset\_pair\_decompose*:

$\exists k \ n1 \ n2. m1 = k + n1 \wedge m2 = k + n2 \wedge (\text{head}_\omega \ n1 \neq \text{head}_\omega \ n2 \vee n1 = 0 \wedge n2 = 0)$

**proof** –

**define** *n1* **where**  $n1: n1 = m1 - m2$

**define** *n2* **where**  $n2: n2 = m2 - m1$

```

define k where k1: k = m1 - n1

have k2: k = m2 - n2
  using k1 unfolding n1 n2 by (simp add: minus_diff_sym_hmset)

have m1 = k + n1
  unfolding k1
  by (metis (no_types) n1 add_diff_cancel_left add.commute add_diff_cancel_right' diff_add_zero
    diff_diff_add minus_diff_sym_hmset)
moreover have m2 = k + n2
  unfolding k2
  by (metis n2 add.commute add_diff_cancel_left add_diff_cancel_left' add_diff_cancel_right'
    diff_add_zero diff_diff_add diff_zero k2 minus_diff_sym_hmset)
moreover have hd_n: head_ω n1 ≠ head_ω n2 if n1_or_n2_nz: n1 ≠ 0 ∨ n2 ≠ 0
proof (cases n1 = 0 n2 = 0 rule: bool.exhaust[case_product bool.exhaust])
  case False_False
  note n1_nz = this(1)[simplified] and n2_nz = this(2)[simplified]

define δ1 where δ1 = hmsetmset n1
define δ2 where δ2 = hmsetmset n2

have δ1_inter_δ2: δ1 ∩# δ2 = {#}
  unfolding δ1_def δ2_def n1 n2 minus_hmultiset_def by (simp add: diff_intersect_sym_diff)

have δ1_ne: δ1 ≠ {#}
  unfolding δ1_def using n1_nz by simp
have δ2_ne: δ2 ≠ {#}
  unfolding δ2_def using n2_nz by simp

have max_δ1: Max (set_mset δ1) ∈# δ1
  using δ1_ne by simp
have max_δ2: Max (set_mset δ2) ∈# δ2
  using δ2_ne by simp
have max_δ1_ne_δ2: Max (set_mset δ1) ≠ Max (set_mset δ2)
  using δ1_inter_δ2 disjunct_not_in max_δ1 max_δ2 by force

show ?thesis
  using n1_nz n2_nz
  by (cases n1 rule: hmultiset.exhaust_sel, cases n2 rule: hmultiset.exhaust_sel,
    auto simp: head_ω_def zero_hmultiset_def max_δ1_ne_δ2[unfolded δ1_def δ2_def])
qed (use n1_or_n2_nz in (auto simp: head_ω_def))
ultimately show ?thesis
  by blast
qed

lemma hmset_pair_decompose_less:
  assumes m1_lt_m2: m1 < m2
  shows ∃ k n1 n2. m1 = k + n1 ∧ m2 = k + n2 ∧ head_ω n1 < head_ω n2
proof -
  obtain k n1 n2 where
    m1: m1 = k + n1 and
    m2: m2 = k + n2 and
    hds: head_ω n1 ≠ head_ω n2 ∨ n1 = 0 ∧ n2 = 0
  using hmset_pair_decompose[of m1 m2] by blast
{
  assume n1 = 0 and n2 = 0
  hence m1 = m2
    unfolding m1 m2 by simp
  hence False
    using m1_lt_m2 by simp
}
moreover

```



```

{
  assume head_ω n1 > head_ω n2
  hence n1 > n2
    by (rule head_ω_lt_imp_lt)
  hence m1 > m2
    unfolding m1 m2 by simp
  hence False
    using m1_lt_m2 by simp
}
ultimately show ?thesis
  using m1 m2 hds by (blast elim: neqE)
qed

lemma hmset_pair_decompose_less_eq:
  assumes m1 ≤ m2
  shows ∃ k n1 n2. m1 = k + n1 ∧ m2 = k + n2 ∧ (head_ω n1 < head_ω n2 ∨ n1 = 0 ∧ n2 = 0)
  using assms
  by (metis add_cancel_right_right hmset_pair_decompose_less order.not_eq_order_implies_strict)

lemma mono_cross_mult_less_hmset:
  fixes Aa A Ba B :: hmultiset
  assumes A_lt: A < Aa and B_lt: B < Ba
  shows A * Ba + B * Aa < A * B + Aa * Ba
proof -
  obtain j m1 m2 where A: A = j + m1 and Aa: Aa = j + m2 and hd_m: head_ω m1 < head_ω m2
    by (metis hmset_pair_decompose_less[OF A_lt])
  obtain k n1 n2 where B: B = k + n1 and Ba: Ba = k + n2 and hd_n: head_ω n1 < head_ω n2
    by (metis hmset_pair_decompose_less[OF B_lt])

  have hd_lt: head_ω (m1 * n2 + m2 * n1) < head_ω (m1 * n1 + m2 * n2)
  proof simp
    have ∧h ha :: hmultiset. 0 < h ∨ ¬ ha < h
    by force
    hence ¬ head_ω m2 * head_ω n2 ≤ sup (head_ω m1 * head_ω n2) (head_ω m2 * head_ω n1)
    using hd_m hd_n sup_hmultiset_def by auto
    thus sup (head_ω m1 * head_ω n2) (head_ω m2 * head_ω n1)
      < sup (head_ω m1 * head_ω n1) (head_ω m2 * head_ω n2)
    by (meson leI sup.bounded_iff)
  qed
  show ?thesis
    unfolding A Aa B Ba ring_distrib by (simp add: algebra_simps head_ω_lt_imp_lt[OF hd_lt])
qed

lemma triple_cross_mult_hmset:
  An * (Bn * Cn + Bp * Cp - (Bn * Cp + Cn * Bp))
  + (Cn * (An * Bp + Bn * Ap - (An * Bn + Ap * Bp))
  + (Ap * (Bn * Cp + Cn * Bp - (Bn * Cn + Bp * Cp))
  + Cp * (An * Bn + Ap * Bp - (An * Bp + Bn * Ap))) =
  An * (Bn * Cp + Cn * Bp - (Bn * Cn + Bp * Cp))
  + (Cn * (An * Bn + Ap * Bp - (An * Bp + Bn * Ap))
  + (Ap * (Bn * Cn + Bp * Cp - (Bn * Cp + Cn * Bp))
  + Cp * (An * Bp + Bn * Ap - (An * Bn + Ap * Bp)))
for Ap An Bp Bn Cp Cn Dp Dn :: hmultiset
apply (simp add: algebra_simps)
apply (unfold add.assoc[symmetric])

apply (rule add_right_cancel[THEN iffD1, of _ Cp * (An * Bp + Ap * Bn)])
apply (unfold add.assoc)
apply (subst times_diff_plus_sym_hmset)
apply (unfold add.assoc[symmetric])
apply (subst (12) add commute)
apply (subst (11) add commute)
apply (unfold add.assoc[symmetric])

```

```

apply (rule add_right_cancel[THEN iffD1, of _ Cn * (An * Bn + Ap * Bp)])
apply (unfold add.assoc)
apply (subst times_diff_plus_sym_hmset)
apply (unfold add.assoc[symmetric])
apply (subst (14) add.commute)
apply (subst (13) add.commute)
apply (unfold add.assoc[symmetric])

apply (rule add_right_cancel[THEN iffD1, of _ Ap * (Bn * Cn + Bp * Cp)])
apply (unfold add.assoc)
apply (subst times_diff_plus_sym_hmset)
apply (unfold add.assoc[symmetric])
apply (subst (16) add.commute)
apply (subst (15) add.commute)
apply (unfold add.assoc[symmetric])

apply (rule add_right_cancel[THEN iffD1, of _ An * (Bn * Cp + Bp * Cn)])
apply (unfold add.assoc)
apply (subst times_diff_plus_sym_hmset)
apply (unfold add.assoc[symmetric])
apply (subst (18) add.commute)
apply (subst (17) add.commute)
apply (unfold add.assoc[symmetric])

by (simp add: algebra_simps)

```

## 7.7 Conversions to Natural Numbers

**definition** *offset\_hmset* :: *hmultiset*  $\Rightarrow$  *nat* **where**  
*offset\_hmset* M = count (hmsetmset M) 0

**lemma** *offset\_hmset\_of\_nat*[simp]: *offset\_hmset* (of\_nat n) = n  
**unfolding** *offset\_hmset\_def* of\_nat\_hmset **by** simp

**lemma** *offset\_hmset\_numeral*[simp]: *offset\_hmset* (numeral n) = numeral n  
**unfolding** *offset\_hmset\_def* **by** (metis *offset\_hmset\_def* *offset\_hmset\_of\_nat* of\_nat\_numeral)

**definition** *sum\_coefs* :: *hmultiset*  $\Rightarrow$  *nat* **where**  
*sum\_coefs* M = size (hmsetmset M)

**lemma** *sum\_coefs\_distrib\_plus*[simp]: *sum\_coefs* (M + N) = *sum\_coefs* M + *sum\_coefs* N  
**unfolding** *plus\_hmultiset\_def* *sum\_coefs\_def* **by** simp

**lemma** *sum\_coefs\_gt\_0*: *sum\_coefs* M > 0  $\longleftrightarrow$  M > 0  
**by** (auto simp: *sum\_coefs\_def* zero\_hmultiset\_def hmsetmset\_less[symmetric] less\_multiset\_extDM\_less  
nonempty\_has\_size[symmetric])

## 7.8 An Example

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

**lemma** *ludwig\_waldmann\_less*:  
**fixes**  $\alpha 1 \alpha 2 \beta 1 \beta 2 \gamma \delta :: \text{hmultiset}$   
**assumes**  
 $\alpha \beta 2 \gamma \text{lt} \alpha \beta 1 \gamma$ :  $\alpha 2 + \beta 2 * \gamma < \alpha 1 + \beta 1 * \gamma$  **and**  
 $\beta 2 \text{le} \beta 1$ :  $\beta 2 \leq \beta 1$  **and**  
 $\gamma \text{lt} \delta$ :  $\gamma < \delta$   
**shows**  $\alpha 2 + \beta 2 * \delta < \alpha 1 + \beta 1 * \delta$   
**proof** –  
**obtain**  $\beta 0 \beta 2a \beta 1a$  **where**  
 $\beta 1$ :  $\beta 1 = \beta 0 + \beta 1a$  **and**  
 $\beta 2$ :  $\beta 2 = \beta 0 + \beta 2a$  **and**

```

hd_β2a_vs_β1a: head_ω β2a < head_ω β1a ∨ β2a = 0 ∧ β1a = 0
using hmset_pair_decompose_less_eq[OF β2_le_β1] by blast

obtain η γ a δ a where
  γ: γ = η + γ a and
  δ: δ = η + δ a and
  hd_γa_lt_δa: head_ω γ a < head_ω δ a
using hmset_pair_decompose_less[OF γ_lt_δ] by blast

have α2 + β0 * γ + β2a * γ = α2 + β2 * γ
  unfolding β2 by (simp add: add.commute add.left_commute distrib_left mult.commute)
also have ... < α1 + β1 * γ
  by (rule αβ2γ_lt_αβ1γ)
also have ... = α1 + β0 * γ + β1a * γ
  unfolding β1 by (simp add: add.commute add.left_commute distrib_left mult.commute)
finally have *: α2 + β2a * γ < α1 + β1a * γ
  by (metis add_less_cancel_right semiring_normalization_rules(23))

have α2 + β2 * δ = α2 + β0 * δ + β2a * δ
  unfolding β2 by (simp add: ab_semigroup_add_class.add_ac(1) distrib_right)
also have ... = α2 + β0 * δ + β2a * η + β2a * δ a
  unfolding δ by (simp add: distrib_left semiring_normalization_rules(25))
also have ... ≤ α2 + β0 * δ + β2a * η + β2a * δ a + β2a * γ a
  by simp
also have ... = α2 + β2a * γ + β0 * δ + β2a * δ a
  unfolding γ distrib_left add.assoc[symmetric] by (simp add: semiring_normalization_rules(23))
also have ... < α1 + β1a * γ + β0 * δ + β2a * δ a
  using * by simp
also have ... = α1 + β1a * η + β1a * γ a + β0 * η + β0 * δ a + β2a * δ a
  unfolding γ δ distrib_left add.assoc[symmetric] by (rule refl)
also have ... ≤ α1 + β1a * η + β0 * η + β0 * δ a + β1a * δ a
proof -
  have β1a * γ a + β2a * δ a ≤ β1a * δ a
  proof (cases β2a = 0 ∧ β1a = 0)
    case False
    hence head_ω β2a < head_ω β1a
    using hd_β2a_vs_β1a by blast
    hence head_ω (β1a * γ a + β2a * δ a) < head_ω (β1a * δ a)
    using hd_γa_lt_δa by (auto intro: gr_zeroI_hmset simp: sup_hmultiset_def)
    hence β1a * γ a + β2a * δ a < β1a * δ a
    by (rule head_ω_lt_imp_lt)
  thus ?thesis
  by simp
qed simp
thus ?thesis
by simp
qed
finally show ?thesis
  unfolding β1 δ
  by (simp add: distrib_left distrib_right add.assoc[symmetric] semiring_normalization_rules(23))
qed
end

```

## 8 Signed Syntactic Ordinals in Cantor Normal Form

```

theory Signed_Syntactic_Ordinal
imports Signed_Hereditary_Multiset Syntactic_Ordinal
begin

```

### 8.1 Natural (Hessenberg) Product

```

instantiation zhmultiset :: comm_ring_1

```

**begin**

**abbreviation**  $\omega_z\_exp :: \text{hmultiset} \Rightarrow \text{zhmultiset} (\omega_z \hat{\ })$  **where**  
 $\omega_z \hat{\ } \equiv \lambda m. \text{ZHMSet} \{\#m\}_z$

**lift-definition**  $\text{one\_zhmultiset} :: \text{zhmultiset} \text{ is } \{\#0\}_z$  .

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

**lemma**  $\omega_z\_as\_ \omega: \omega_z = \text{zhmset\_of } \omega$   
**by** *simp*

**lift-definition**  $\text{times\_zhmultiset} :: \text{zhmultiset} \Rightarrow \text{zhmultiset} \Rightarrow \text{zhmultiset}$  **is**  
 $\lambda M N.$   
 $\text{zmset\_of } (\text{hmsetmset } (\text{HMSet } (\text{mset\_pos } M) * \text{HMSet } (\text{mset\_pos } N)))$   
 $- \text{zmset\_of } (\text{hmsetmset } (\text{HMSet } (\text{mset\_pos } M) * \text{HMSet } (\text{mset\_neg } N)))$   
 $+ \text{zmset\_of } (\text{hmsetmset } (\text{HMSet } (\text{mset\_neg } M) * \text{HMSet } (\text{mset\_neg } N)))$   
 $- \text{zmset\_of } (\text{hmsetmset } (\text{HMSet } (\text{mset\_neg } M) * \text{HMSet } (\text{mset\_pos } N)))$  .

**lemmas**  $\text{zhmsetmset\_times} = \text{times\_zhmultiset.rep\_eq}$

**instance**

**proof** (*intro\_classes, goal\_cases mult\_assoc mult\_comm mult\_1 distrib zero\_neq\_one*)

**case** (*mult\_assoc a b c*)

**show** *?case*

**by** (*transfer,*

*simp add: algebra\_simps zmset\_of\_plus[symmetric] hmsetmset\_plus[symmetric] HMSet\_diff,*  
*rule triple\_cross\_mult\_hmset*)

**next**

**case** (*mult\_comm a b*)

**show** *?case*

**by** *transfer (auto simp: algebra\_simps)*

**next**

**case** (*mult\_1 a*)

**show** *?case*

**by** *transfer (auto simp: algebra\_simps mset\_pos\_neg\_partition[symmetric])*

**next**

**case** (*distrib a b c*)

**show** *?case*

**by** (*simp add: times\_zhmultiset\_def ZHMSet\_plus[symmetric] zmset\_of\_plus[symmetric]*  
*hmsetmset\_plus[symmetric] algebra\_simps hmset\_pos\_plus hmset\_neg\_plus*)  
(*simp add: mult.commute[of \_ hmset\_pos c] mult.commute[of \_ hmset\_neg c]*  
*add.commute[of hmset\_neg c \* M hmset\_pos c \* N for M N]*  
*add.assoc[symmetric] ring\_distrib(1)[symmetric] hmset\_pos\_neg\_dual*)

**next**

**case** *zero\_neq\_one*

**show** *?case*

**unfolding** *zero\_zhmultiset\_def one\_zhmultiset\_def* **by** *simp*

**qed**

**end**

**lemma**  $\text{zhmset\_of } 1: \text{zhmset\_of } 1 = 1$

**by** (*simp add: one\_hmultiset\_def one\_zhmultiset\_def*)

**lemma**  $\text{zhmset\_of\_times}: \text{zhmset\_of } (A * B) = \text{zhmset\_of } A * \text{zhmset\_of } B$

**by** *transfer simp*

**lemma**  $\text{zhmset\_of\_prod\_list}$ :

$\text{zhmset\_of } (\text{prod\_list } Ms) = \text{prod\_list } (\text{map } \text{zhmset\_of } Ms)$

**by** (*induct Ms*) (*auto simp: one\_hmultiset\_def one\_zhmultiset\_def zhmset\_of\_times*)

## 8.2 Embedding of Natural Numbers

**lemma** *of\_nat\_zhmset*:  $of\_nat\ n = zhmset\_of\ (of\_nat\ n)$   
**by** (*induct n*) (*auto simp: zero\_zhmset\_def zhmset\_of\_plus zhmset\_of\_1*)

**lemma** *of\_nat\_inject\_zhmset*[*simp*]:  $(of\_nat\ m :: zhmset) = of\_nat\ n \longleftrightarrow m = n$   
**unfolding** *of\_nat\_zhmset* **by** *simp*

**lemma** *plus\_of\_nat\_plus\_of\_nat\_zhmset*:  
 $k + of\_nat\ m + of\_nat\ n = k + of\_nat\ (m + n)$  **for**  $k :: zhmset$   
**by** *simp*

**lemma** *plus\_of\_nat\_minus\_of\_nat\_zhmset*:  
**fixes**  $k :: zhmset$   
**assumes**  $n \leq m$   
**shows**  $k + of\_nat\ m - of\_nat\ n = k + of\_nat\ (m - n)$   
**using** *assms* **by** (*simp add: of\_nat\_diff*)

**lemma** *of\_nat\_lt\_omega\_z*[*simp*]:  $of\_nat\ n < \omega_z$   
**unfolding** *omega\_z\_as\_omega* **using** *of\_nat\_lt\_omega of\_nat\_zhmset zhmset\_of\_less* **by** *presburger*

**lemma** *of\_nat\_ne\_omega\_z*[*simp*]:  $of\_nat\ n \neq \omega_z$   
**by** (*metis of\_nat\_lt\_omega\_z mset\_le\_asym mset\_lt\_single\_iff*)

## 8.3 Embedding of Extended Natural Numbers

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

**lemma** *zhmset\_of\_enat\_0*[*simp*]:  $zhmset\_of\_enat\ 0 = 0$   
**by** (*simp add: zero\_enat\_def*)

**lemma** *zhmset\_of\_enat\_1*[*simp*]:  $zhmset\_of\_enat\ 1 = 1$   
**by** (*simp add: one\_enat\_def del: One\_nat\_def*)

**lemma** *zhmset\_of\_enat\_of\_nat*[*simp*]:  $zhmset\_of\_enat\ (of\_nat\ n) = of\_nat\ n$   
**using** *of\_nat\_eq\_enat* **by** *auto*

**lemma** *zhmset\_of\_enat\_numeral*[*simp*]:  $zhmset\_of\_enat\ (numeral\ n) = numeral\ n$   
**by** (*simp add: numeral\_eq\_enat*)

**lemma** *zhmset\_of\_enat\_le\_omega\_z*[*simp*]:  $zhmset\_of\_enat\ n \leq \omega_z$   
**using** *of\_nat\_lt\_omega\_z* [*THEN less\_imp\_le*] **by** (*cases n*) *auto*

**lemma** *zhmset\_of\_enat\_eq\_omega\_z\_iff*[*simp*]:  $zhmset\_of\_enat\ n = \omega_z \longleftrightarrow n = \infty$   
**by** (*cases n*) *auto*

## 8.4 Inequalities and Some (Dis)equalities

**instance** *zhmset* :: *zero\_less\_one*  
**by** (*intro\_classes, transfer, transfer, auto*)

**instantiation** *zhmset* :: *linordered\_idom*  
**begin**

**definition** *sgn\_zhmset* ::  $zhmset \Rightarrow zhmset$  **where**  
 $sgn\_zhmset\ M = (if\ M = 0\ then\ 0\ else\ if\ M > 0\ then\ 1\ else\ -1)$

**definition** *abs\_zhmset* ::  $zhmset \Rightarrow zhmset$  **where**  
 $abs\_zhmset\ M = (if\ M < 0\ then\ -\ M\ else\ M)$

**lemma** *gt\_0\_times\_gt\_0\_imp*:  
**fixes**  $a\ b :: zhmset$

```

assumes  $a\_gt0: a > 0$  and  $b\_gt0: b > 0$ 
shows  $a * b > 0$ 
proof –
  show ?thesis
    using  $a\_gt0$   $b\_gt0$ 
    by (subst (asm) (2 4) zhmsset_pos_neg_partition, simp, transfer,
      simp del: HMSet_less add: algebra_simps zmsset_of_plus[symmetric] hmsetmset_plus[symmetric]
      zmsset_of_less HMSet_less[symmetric])
      (rule mono_cross_mult_less_hmset)
qed

```

**instance**

```

proof intro_classes
  fix  $a$   $b$   $c :: \text{zhmultiset}$ 

```

**assume**

```

 $a\_lt\_b: a < b$  and
 $zero\_lt\_c: 0 < c$ 

```

```

have  $c * b < c * b + c * (b - a)$ 
  using  $gt\_0\_times\_gt\_0\_imp$  by (simp add:  $a\_lt\_b$   $zero\_lt\_c$ )
hence  $c * a + c * (b - a) < c * b + c * (b - a)$ 
  by (simp add: right_diff_distrib')
thus  $c * a < c * b$ 
  by simp

```

**qed** (auto simp: sgn\_zhmultiset\_def abs\_zhmultiset\_def)

**end**

```

lemma  $le\_zhmsset\_of\_pos: M \leq \text{zhmsset\_of} (\text{hmset\_pos } M)$ 
  by (simp add: less_eq_zhmultiset.rep_eq mset_pos_supset subset_eq_imp_le_zmset)

```

```

lemma  $minus\_zhmsset\_of\_pos\_le: - \text{zhmsset\_of} (\text{hmset\_neg } M) \leq M$ 
  by (metis le_zhmsset_of_pos minus_le_iff mset_pos_uminus zhmssetmset_uminus)

```

```

lemma  $\text{zhmsset\_of\_nonneg}[simp]: \text{zhmsset\_of } M \geq 0$ 
  by (metis hmsetmset_0 zero_le_hmset zero_zhmultiset_def zhmsset_of_le zmset_of_empty)

```

**lemma**

```

fixes  $n :: \text{zhmultiset}$ 
assumes  $0 \leq m$ 
shows
   $le\_add1\_hmset: n \leq n + m$  and
   $le\_add2\_hmset: n \leq m + n$ 
using assms by simp+

```

**lemma**  $less\_iff\_add1\_le\_zhmsset: m < n \iff m + 1 \leq n$  **for**  $m$   $n :: \text{zhmultiset}$

**proof**

```

assume  $m\_lt\_n: m < n$ 
show  $m + 1 \leq n$ 
proof –
  obtain  $hh :: \text{hmultiset}$  and  $zz :: \text{zhmultiset}$  and  $hha :: \text{hmultiset}$  where
     $f1: m = \text{zhmsset\_of } hh + zz \wedge n = \text{zhmsset\_of } hha + zz \wedge hh < hha$ 
  using less_hmset_zhmssetE[OF  $m\_lt\_n$ ] by metis
  hence  $\text{zhmsset\_of} (hh + 1) \leq \text{zhmsset\_of } hha$ 
  by (metis (no_types) less_iff_add1_le_hmset zhmsset_of_le)
  thus ?thesis
    using  $f1$  by (simp add: zhmsset_of_1 zhmsset_of_plus)

```

**qed**

**qed** simp

```

lemma  $gt\_0\_lt\_mult\_gt\_1\_zhmsset:$ 
  fixes  $m$   $n :: \text{zhmultiset}$ 

```

```

assumes  $m > 0$  and  $n > 1$ 
shows  $m < m * n$ 
using assms by simp

lemma zero_less_iff_1_le_zhmset:  $0 < n \longleftrightarrow 1 \leq n$  for  $n :: \text{zhmultiset}$ 
by (rule less_iff_add1_le_zhmset[of 0, simplified])

lemma less_add_1_iff_le_hmset:  $m < n + 1 \longleftrightarrow m \leq n$  for  $m\ n :: \text{zhmultiset}$ 
by (rule less_iff_add1_le_zhmset[of m n + 1, simplified])

lemma nonneg_le_mult_right_mono_zhmset:
fixes  $x\ y\ z :: \text{zhmultiset}$ 
assumes  $x: 0 \leq x$  and  $y: 0 < y$  and  $z: x \leq z$ 
shows  $x \leq y * z$ 
using x zero_less_iff_1_le_zhmset[THEN iffD1, OF y] z
by (meson dual_order.trans leD mult_less_cancel_right2 not_le_imp_less)

instance hmultiset :: ordered_cancel_comm_semiring
by intro_classes

instance hmultiset :: linordered_semiring_1_strict
by intro_classes

instance hmultiset :: bounded_lattice_bot
by intro_classes

instance hmultiset :: zero_less_one
by intro_classes

instance hmultiset :: linordered_nonzero_semiring
by intro_classes

instance hmultiset :: semiring_no_zero_divisors
by intro_classes

lemma zero_lt_omega_z[simp]:  $0 < \omega_z$ 
by (metis of_nat_lt_omega_z of_nat_0)

lemma one_lt_omega_z[simp]:  $1 < \omega_z$ 
by (metis enat_defs(2) zhmset_of_enat.simps(1) zhmset_of_enat_1 of_nat_lt_omega_z)

lemma numeral_lt_omega_z[simp]: numeral  $n < \omega_z$ 
using zhmset_of_enat_numeral[symmetric] zhmset_of_enat.simps(1) of_nat_lt_omega_z numeral_eq_enat
by presburger

lemma one_le_omega_z[simp]:  $1 \leq \omega_z$ 
by (simp add: less_imp_le)

lemma of_nat_le_omega_z[simp]: of_nat  $n \leq \omega_z$ 
by (simp add: le_less)

lemma numeral_le_omega_z[simp]: numeral  $n \leq \omega_z$ 
by (simp add: less_imp_le)

lemma not_omega_z_lt_1[simp]:  $\neg \omega_z < 1$ 
by (simp add: not_less)

lemma not_omega_z_lt_of_nat[simp]:  $\neg \omega_z < \text{of\_nat } n$ 
by (simp add: not_less)

lemma not_omega_z_lt_numeral[simp]:  $\neg \omega_z < \text{numeral } n$ 
by (simp add: not_less)

```

**lemma** *not\_ω<sub>z</sub>\_le\_1*[simp]:  $\neg \omega_z \leq 1$   
**by** (*simp add: not\_le*)

**lemma** *not\_ω<sub>z</sub>\_le\_of\_nat*[simp]:  $\neg \omega_z \leq \text{of\_nat } n$   
**by** (*simp add: not\_le*)

**lemma** *not\_ω<sub>z</sub>\_le\_numeral*[simp]:  $\neg \omega_z \leq \text{numeral } n$   
**by** (*simp add: not\_le*)

**lemma** *zero\_ne\_ω<sub>z</sub>*[simp]:  $0 \neq \omega_z$   
**using** *zero\_lt\_ω<sub>z</sub>* **by** *linarith*

**lemma** *one\_ne\_ω<sub>z</sub>*[simp]:  $1 \neq \omega_z$   
**using** *not\_ω<sub>z</sub>\_le\_1* **by** *force*

**lemma** *numeral\_ne\_ω<sub>z</sub>*[simp]:  $\text{numeral } n \neq \omega_z$   
**by** (*metis not\_ω<sub>z</sub>\_le\_numeral numeral\_le\_ω<sub>z</sub>*)

**lemma**  
*ω<sub>z</sub>\_ne\_0*[simp]:  $\omega_z \neq 0$  **and**  
*ω<sub>z</sub>\_ne\_1*[simp]:  $\omega_z \neq 1$  **and**  
*ω<sub>z</sub>\_ne\_of\_nat*[simp]:  $\omega_z \neq \text{of\_nat } m$  **and**  
*ω<sub>z</sub>\_ne\_numeral*[simp]:  $\omega_z \neq \text{numeral } n$   
**using** *zero\_ne\_ω<sub>z</sub> one\_ne\_ω<sub>z</sub> of\_nat\_ne\_ω<sub>z</sub> numeral\_ne\_ω<sub>z</sub>* **by** *metis+*

**lemma**  
*zhmset\_of\_enat\_inject*[simp]:  $\text{zhmset\_of\_enat } m = \text{zhmset\_of\_enat } n \longleftrightarrow m = n$  **and**  
*zhmset\_of\_enat\_lt\_iff\_lt*[simp]:  $\text{zhmset\_of\_enat } m < \text{zhmset\_of\_enat } n \longleftrightarrow m < n$  **and**  
*zhmset\_of\_enat\_le\_iff\_le*[simp]:  $\text{zhmset\_of\_enat } m \leq \text{zhmset\_of\_enat } n \longleftrightarrow m \leq n$   
**by** (*cases m; cases n; simp*)**+**

**lemma** *of\_nat\_lt\_zhmset\_of\_enat\_iff*:  $\text{of\_nat } m < \text{zhmset\_of\_enat } n \longleftrightarrow \text{enat } m < n$   
**by** (*metis zhmset\_of\_enat.simps(1) zhmset\_of\_enat\_lt\_iff\_lt*)

**lemma** *of\_nat\_le\_zhmset\_of\_enat\_iff*:  $\text{of\_nat } m \leq \text{zhmset\_of\_enat } n \longleftrightarrow \text{enat } m \leq n$   
**by** (*metis zhmset\_of\_enat.simps(1) zhmset\_of\_enat\_le\_iff\_le*)

**lemma** *zhmset\_of\_enat\_lt\_iff\_ne\_infinity*:  $\text{zhmset\_of\_enat } x < \omega_z \longleftrightarrow x \neq \infty$   
**by** (*cases x; simp*)

## 8.5 An Example

A new proof of  $\llbracket ?\alpha 2.0 + ?\beta 2.0 * ?\gamma < ?\alpha 1.0 + ?\beta 1.0 * ?\gamma; ?\beta 2.0 \leq ?\beta 1.0; ?\gamma < ?\delta \rrbracket \implies ?\alpha 2.0 + ?\beta 2.0 * ?\delta < ?\alpha 1.0 + ?\beta 1.0 * ?\delta$ :

**lemma**  
**fixes**  $\alpha 1 \alpha 2 \beta 1 \beta 2 \gamma \delta :: \text{hmultiset}$   
**assumes**  
 $\alpha \beta 2 \gamma \text{\_lt\_} \alpha \beta 1 \gamma$ :  $\alpha 2 + \beta 2 * \gamma < \alpha 1 + \beta 1 * \gamma$  **and**  
 $\beta 2 \text{\_le\_} \beta 1$ :  $\beta 2 \leq \beta 1$  **and**  
 $\gamma \text{\_lt\_} \delta$ :  $\gamma < \delta$   
**shows**  $\alpha 2 + \beta 2 * \delta < \alpha 1 + \beta 1 * \delta$

**proof** –  
**let**  $?z = \text{zhmset\_of}$

**note**  $\alpha \beta 2 \gamma \text{\_lt\_} \alpha \beta 1 \gamma' = \alpha \beta 2 \gamma \text{\_lt\_} \alpha \beta 1 \gamma$  [THEN *zhmset\_of\_less* [THEN *iffD2*],  
*simplified zhmset\_of\_plus zhmset\_of\_times*]

**note**  $\beta 2 \text{\_le\_} \beta 1' = \beta 2 \text{\_le\_} \beta 1$  [THEN *zhmset\_of\_le* [THEN *iffD2*]]

**note**  $\gamma \text{\_lt\_} \delta' = \gamma \text{\_lt\_} \delta$  [THEN *zhmset\_of\_less* [THEN *iffD2*]]

**have**  $?z \alpha 2 + ?z \beta 2 * ?z \delta < ?z \alpha 1 + ?z \beta 1 * ?z \gamma + ?z \beta 2 * (?z \delta - ?z \gamma)$   
**using**  $\alpha \beta 2 \gamma \text{\_lt\_} \alpha \beta 1 \gamma'$  **by** (*simp add: algebra\_simps*)

**also have**  $\dots \leq ?z \alpha 1 + ?z \beta 1 * ?z \gamma + ?z \beta 1 * (?z \delta - ?z \gamma)$   
**using**  $\beta 2 \text{\_le\_} \beta 1' \gamma \text{\_lt\_} \delta'$  **by** *simp*



```

finally show ?thesis
  by (simp add: zmsset_of_less zhmsset_of_times[symmetric] zhmsset_of_plus[symmetric] algebra_simps)
qed

end

```

```

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

```

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

### 9.1 Missing Lemmas about Huffman's Ordinals

```

instantiation ordinal :: order_bot
begin

```

```

definition bot_ordinal :: ordinal where
  bot_ordinal = 0

```

```

instance
  by intro_classes (simp add: bot_ordinal_def)

```

```

end

```

```

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

```

```

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

```

```

lemma from_cnf_less_omega_exp:
  assumes  $\forall k \in \text{set } ks. k < l$ 
  shows from_cnf ks <  $\omega ** l$ 
  using assms by (induct ks) (auto simp: additive_principal.sum_less additive_principal_omega_exp)

```

```

lemma from_cnf_0_iff[simp]: from_cnf ks = 0  $\longleftrightarrow$  ks = []
  by (induct ks) (auto simp: ordinal_plus_not_0)

```

```

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

```

```

lemma subseq_from_cnf_less_eq: Sublist.subseq ks ls  $\implies$  from_cnf ks  $\leq$  from_cnf ls
  by (induct rule: list_emb.induct) (auto intro: ordinal_le_plusL order_trans)

```

### 9.2 Embedding of Syntactic Ordinals into Huffman's Ordinals

```

abbreviation  $\omega_h$  :: hmultiset where
   $\omega_h \equiv$  Syntactic_Ordinal. $\omega$ 

```

```

abbreviation  $\omega_h\_exp$  :: hmultiset  $\Rightarrow$  hmultiset ( $\omega_h \hat{\ }^$ ) where
   $\omega_h \hat{\ }^ \equiv$  Syntactic_Ordinal. $\omega\_exp$ 

```

```

primrec ordinal_of_hmset :: hmultiset  $\Rightarrow$  ordinal where
  ordinal_of_hmset (HMSet M) =
    from_cnf (rev (sorted_list_of_multiset (image_mset ordinal_of_hmset M)))

```

```

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

```

```

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

lemma ordinal_of_hmset_1[simp]: ordinal_of_hmset 1 = 1
  using ordinal_of_hmset_suc[of 0] by simp

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

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

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

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

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

  have l_nz: l ≠ 0
    using k_lt_l by auto

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

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

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

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

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

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

  have a_le_x: a ≤ x
    by (simp add: x_max a_in)
  moreover
  {

```

```

assume a_lt_x: a < x
moreover have x_lt_k: x < k
  unfolding k_eq_xKa by (rule mem_imp_less_HMSet) simp
ultimately have a_lt_k: a < k
  by simp

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

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

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

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

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

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

{
  assume x_eq_y: x = y

  have ordinal_of_hmset (HMSet Ka) < ordinal_of_hmset (HMSet La)
proof (rule ih)
  show {#HMSet Ka, HMSet La#} < {#k, l#}
    unfolding k l
    by (metis add_mset_add_single hmsetmset_less hmultiset.sel k k_eq_xKa l l_eq_yLa
      le_multiset_right_total mset_lt_single_iff union_less_mono)
next

```

```

have  $\omega^{\hat{x}} + \text{HMSet } Ka < \omega^{\hat{y}} + \text{HMSet } La$ 
  using  $k\_lt\_l[\text{unfolded } k\_eq\_xKa \ l\_eq\_yLa]$ 
  by (metis  $\text{HMSet\_plus add.commute add\_mset\_add\_single}$ )
thus  $\text{HMSet } Ka < \text{HMSet } La$ 
  using  $x\_eq\_y$  by simp
qed
hence ?thesis
  unfolding  $k\_eq\_xKa \ l\_eq\_yLa$ 
  by (simp, subst (1 2) sorted_insort_is_snoc, simp_all add: ord_x_max ord_y_max,
    force simp:  $x\_eq\_y$ )
}
moreover
{
  assume  $x\_ne\_y: x \neq y$ 

  have  $x\_lt\_y: x < y$ 
    by (metis  $K \ L \ \text{head\_}\omega\_def \ \text{head\_}\omega\_lt\_imp\_lt \ \text{hmsetmset\_less} \ \text{hmultiset.sel} \ k\_lt\_l \ k\_nz \ l\_nz$ 
      less_imp_not_less mset_lt_single_iff neqE  $x \ x\_ne\_y \ y$ )

  have  $\text{ord\_y\_smax\_K}: \text{ordinal\_of\_hmset } a < \text{ordinal\_of\_hmset } y$  if  $a\_in\_K: a \in\# K$  for  $a$ 
  proof (rule ih)
    show  $\{\#a, y\# \} < \{\#k, l\# \}$ 
      unfolding  $k\_eq\_xKa \ l\_eq\_yLa$  using  $a\_in\_K \ k \ k\_eq\_xKa$ 
      by (metis  $\text{add\_mset\_add\_single mem\_imp\_less\_HMSet mset\_lt\_single\_iff union\_less\_mono}$ 
        union_single_eq_member)
  next
    show  $a < y$ 
      by (metis  $\text{Max\_ge finite\_set\_mset less\_le\_trans not\_less\_iff\_gr\_or\_eq that } x \ x\_lt\_y$ )
  qed

  have  $\text{ordinal\_of\_hmset } k < \text{ordinal\_of\_hmset } (\omega^{\hat{y}})$ 
  proof (cases  $La$ )
    case empty
    show ?thesis
      unfolding  $k$  by (auto intro!: from_cnf_less_omega_exp simp: ord_y_smax_K)
  next
    case  $La: (add \ ya \ Lb)$ 
    show ?thesis
      proof (rule ih)
        show  $\{\#k, \omega^{\hat{y}}\# \} < \{\#k, l\# \}$ 
          unfolding  $l\_eq\_yLa \ La$  by simp
      next
        show  $k < \omega^{\hat{y}}$ 
        proof -
          have  $\bigwedge m. x < \text{Max\_mset } (\text{add\_mset } y \ m)$ 
            by (meson  $\text{Max\_ge finite\_set\_mset less\_le\_trans union\_single\_eq\_member } x \ lt\_y$ )
          then show ?thesis
            by (metis  $K \ x \ \text{head\_}\omega\_def \ \text{head\_}\omega\_lt\_imp\_lt \ \text{hmsetmset\_less} \ \text{hmultiset.sel} \ k\_nz$ 
              mset_lt_single_iff  $x \ lt\_y$ )
        qed
      qed
    qed
  qed
  also have  $\dots \leq \text{ordinal\_of\_hmset } l$ 
    unfolding  $l\_eq\_yLa$ 
    by (auto simp del: from_cnf_simps intro!: subseq_from_cnf_less_eq
      simp: subseq_from_cnf_less_eq sorted_insort_is_snoc ord_y_max)
  ultimately have ?thesis
    by simp
}
ultimately show ?thesis
  by sat
qed
qed

```

**lemma** *ordinal\_of\_hmset\_less[simp]*: *ordinal\_of\_hmset*  $k < \text{ordinal\_of\_hmset } l \longleftrightarrow k < l$   
**using** *less\_imp\_not\_less less\_imp\_ordinal\_of\_hmset\_less neq\_iff* **by** *blast*

**end**

## 10 Termination of McCarthy's 91 Function

**theory** *McCarthy\_91*  
**imports** *HOL-Library.Multiset\_Order*  
**begin**

**lemma** *funpow\_rec*:  $f \overset{\sim}{\sim} n = (\text{if } n = 0 \text{ then } \text{id} \text{ else } f \circ f \overset{\sim}{\sim} (n - 1))$   
**by** (*induct n*) *auto*

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

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

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

**function**  $g :: \text{nat} \Rightarrow \text{int} \Rightarrow \text{int}$  **where**  
 $g\ n\ z = (\text{if } n = 0 \text{ then } z \text{ else if } z > 100 \text{ then } g\ (n - 1)\ (z - 10) \text{ else } g\ (n + 1)\ (z + 11))$   
**by** *pat\_completeness auto*

**termination**

**proof** –

**define**  $lt :: (\text{int} \times \text{int}) \text{ set}$  **where**  
 $lt = \{(a, b). b < a \wedge a \leq 111\}$

**have**  $lt\_trans$ : *trans*  $lt$   
**unfolding** *trans\_def lt\_def* **by** *simp*  
**have**  $lt\_irrefl$ : *irrefl*  $lt$   
**unfolding** *irrefl\_def lt\_def* **by** *simp*

**let**  $?LT = \text{mult } lt$   
**let**  $?T = \lambda(n, z). \tau\ n\ z$   
**let**  $?R = \text{inv\_image } ?LT\ ?T$

**show**  $?thesis$

**proof** (*relation*  $?R$ )

**show**  $wf\ ?R$

**by** (*auto simp: lt\_def intro!: wf\_inv\_image[OF wf\_mult]*  
 $wf\_subset[OF wf\_measure[of \lambda z. \text{nat } (111 - z)]]$ )

**next**

**fix**  $n :: \text{nat}$  **and**  $z :: \text{int}$   
**assume**  $n\_ne\_0$ :  $n \neq 0$

{  
**assume**  $z\_gt\_100$ :  $z > 100$

**have**  $\text{map } (\lambda i. (f \overset{\sim}{\sim} \text{nat } i)\ (z - 10)) [0.. \text{int } n - 2] =$   
 $\text{map } (\lambda i. (f \overset{\sim}{\sim} \text{nat } i)\ z) [1.. \text{int } n - 1]$   
**using**  $n\_ne\_0$

**proof** (*induct n rule: less\_induct*)

**case** (*less n*)

**note**  $ih = \text{this}(1)$  **and**  $n\_ne\_0 = \text{this}(2)$

**show**  $?case$

**proof** (*cases n = 1*)

**case** *True*

```

thus ?thesis
  by simp
next
case False
hence n_ge_2: n ≥ 2
  using n_ne_0 by simp

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

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

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

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

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

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

have
  split_l: [2..int n] = [2..int (n - 1)] @ [int n] and
  split_r: [1..int n - 1] = [1..int (n - 1) - 1] @ [int n - 1]
  using n_ge_2 by (induct n) (auto simp: upto_rec2)
from z_le_100 have f_f_z_11: f (f (z + 11)) = f z
  by (simp add: f_def)
moreover define m where m = n - 2
with n_ge_2 have n = m + 2
  by simp
ultimately have f_repeat: (f ^^ n) (z + 11) = (f ^^ (n - 1)) z
  by (simp add: funpow_Suc_right del: funpow_simps)
with n_ge_2 show ?thesis

```

```

    by (auto intro: ih [of nat (int n - 1)]
        simp: less.hypos split_l split_r nat_add_distrib nat_diff_distrib)
  qed
qed

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

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

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

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

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

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

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

declare g.simps [simp del]

```

end

## 11 Termination of the Hydra Battle

```
theory Hydra_Battle
imports Syntactic_Ordinal
begin
```

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

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

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

```
primrec encode :: lisp  $\Rightarrow$  hmultiset where
  encode Nil = 0
| encode (Cons l r) =  $\omega^{\sim}(encode\ l) + encode\ r$ 
```

```
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 *  $\omega^{\sim}(encode\ y) + encode\ x$ 
  unfolding of_nat_times_omega_exp by (induct n) (auto simp: HMSet_plus[symmetric])
```

```
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))
  by pat_completeness auto
```

```
termination
  by (relation measure ( $\lambda(\_, x). size\ x$ ), rule wf_measure, rename_tac n x, case_tac x, auto)
```

```
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))
  by pat_completeness auto
```

```
termination
```

```
proof -
  let ?R = inv_image {(m, n). m < n} ( $\lambda(n, x). encode\ x$ )
```

```
show ?thesis
```

```
proof (relation ?R)
```

```
  show wf ?R
```

```
    by (rule wf_inv_image) (rule wf)
```

```
next
```

```
  fix n x
```

```
  assume x_cons: x  $\neq$  Nil
```

```
  thus ((n + 1, d n x), n, x)  $\in$  ?R
```

```
    unfolding inv_image_def mem_Collect_eq prod.case
```

```
  proof (induct x)
```

```
    case (Cons l r)
```

```
    note ihl = this(1)
```

```
    show ?case
```

```
    proof (subst d.simps, simp, intro conjI impI)
```



```

    assume l_cons: l ≠ Nil
  {
    assume car l = Nil
    show encode (f n (cdr l) r) < ω^(encode l) + encode r
      using l_cons by (cases l) (auto simp: encode_f[unfolded of_nat_times_ω_exp])
  }
  {
    show encode (d n l) < encode l
      by (rule ihl[OF l_cons])
  }
}
qed
qed simp
qed
qed

declare h.simps[simp del]

end

```

## 12 Termination of the Goodstein Sequence

```

theory Goodstein_Sequence
imports Multiset_More Syntactic_Ordinal
begin

```

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

### 12.1 Lemmas about Division

```

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

```

```

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

```

### 12.2 Hereditary and Nonhereditary Base-*n* Systems

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

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

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

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

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

```

```

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

```

```

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

```

```

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

```

### 12.3 Encoding of Natural Numbers into Ordinals

```

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

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

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

```

qed

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

**lemma** *encode\_0*[*simp*]: *encode e 0 = 0*  
**by** (*subst encode.simps*) *simp*

**lemma** *encode\_Suc*:  
*encode e (Suc n) = of\_nat (Suc n mod base) \* ω<sup>^(encode 0 e) + encode (e + 1) (Suc n div base)</sup>*  
**by** (*subst encode.simps*) *simp*

**lemma** *encode\_0\_iff*: *encode e n = 0 ↔ n = 0*

**proof** (*induct n arbitrary: e rule: less\_induct*)  
**case** (*less n*)  
**note** *ih = this*

**show** *?case*

**proof** (*cases n*)

**case** *0*

**thus** *?thesis*

**by** *simp*

**next**

**case** *n: (Suc m)*

**show** *?thesis*

**proof** (*cases n mod base = 0*)

**case** *True*

**hence** *n div base ≠ 0*

**using** *div\_eq\_0\_iff n* **by** *fastforce*

**thus** *?thesis*

**using** *ih[of Suc m div base] n*

**by** (*simp add: encode\_Suc*) (*metis One\_nat\_def base\_ge\_2 div\_eq\_dividend\_iff div\_le\_dividend leD lessI nat\_neq\_iff numeral\_2\_eq\_2*)

**next**

**case** *False*

**thus** *?thesis*

**using** *n plus\_hmultiset\_def* **by** (*simp add: encode\_Suc[unfolded of\_nat\_times\_ω\_exp]*)

**qed**

**qed**

qed

**lemma** *encode\_Suc\_exp*: *encode (Suc e) n = encode e (base \* n)*

**using** *base\_ge\_2*

**by** (*subst (1 2) encode.simps, subst (4) encode.simps, simp add: zero\_hmultiset\_def[symmetric]*)

**lemma** *encode\_exp\_0*: *encode e n = encode 0 (base<sup>e</sup> \* n)*

**by** (*induct e arbitrary: n*) (*simp\_all add: encode\_Suc\_exp mult.assoc mult.commute*)

**lemma** *mem\_hmsetmset\_encodeD*: *M ∈# hmsetmset (encode e n) ⇒ ∃ e' ≥ e. M = encode 0 e'*

**proof** (*induct e n rule: encode.induct*)

**case** (*1 e n*)

**note** *ih = this(1-2)* **and** *M\_in = this(3)*

**show** *?case*

**proof** (*cases n*)

**case** *0*

**thus** *?thesis*

**using** *M\_in* **by** *simp*

**next**

**case** *n: (Suc m)*

{  
**assume** *M ∈# replicate\_mset (n mod base) (encode 0 e)*  
**hence** *?thesis*

```

    by (meson in_replicate_mset order_refl)
  }
moreover
  {
    assume  $M \in\# \text{hmsetmset } (\text{encode } (e + 1) (n \text{ div base}))$ 
    hence ?thesis
    using ih(2) le_add1 n order_trans by blast
  }
ultimately show ?thesis
  using M_in[unfolded n encode_Suc[unfolded of_nat_times_omega_exp], folded n]
  unfolding hmsetmset_plus by auto
qed
qed

```

**lemma** *less\_imp\_encode\_less*:  $n < p \implies \text{encode } e \ n < \text{encode } e \ p$

**proof** (*induct e n arbitrary: p rule: encode.induct*)

**case** (*1 e n*)

**note** *ih = this(1-2)* **and**  $n\_lt\_p = \text{this}(3)$

**show** *?case*

**proof** (*cases n = 0*)

**case** *True*

**thus** *?thesis*

using *n\_lt\_p base\_ge\_2 encode\_0\_iff*[*of e p*] *le\_less* **by** *fastforce*

**next**

**case** *n\_nz: False*

**let** *?Ma = replicate\_mset (n mod base) (encode 0 e)*

**let** *?Na = replicate\_mset (p mod base) (encode 0 e)*

**let** *?Pa = replicate\_mset (n mod base - p mod base) (encode 0 e)*

**have** *HMSet ?Ma + encode (Suc e) (n div base) < HMSet ?Na + encode (Suc e) (p div base)*

**proof** (*cases n mod base < p mod base*)

**case** *mod\_lt: True*

**show** *?thesis*

**by** (*rule add\_less\_le\_mono, simp add: mod\_lt,*

*metis ih(2)*[*of p div base, OF n\_nz*] *Suc\_eq\_plus1 div\_le\_mono le\_less n\_lt\_p*)

**next**

**case** *mod\_ge: False*

**hence** *div\_lt: n div base < p div base*

**by** (*metis add\_le\_cancel\_left div\_le\_mono div\_mult\_mod\_eq le\_neq\_implies\_less less\_imp\_le*  
*n\_lt\_p nat\_neq\_iff*)

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

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

**have** *?M < ?N*

**by** (*auto intro!: ih(2)*[*folded Suc\_eq\_plus1*] *n\_nz div\_lt*)

**then obtain** *X Y* **where**

*X\_nemp: X ≠ {#}* **and**

*X\_sub: X ⊆# ?N* **and**

*M: ?M = ?N - X + Y* **and**

*ex\_gt: ∀ y. y ∈# Y ⟶ (∃ x. x ∈# X ∧ x > y)*

**using** *less\_multiset\_DM* **by** *metis*

{

**fix** *x*

**assume** *x\_in\_X: x ∈# X*

**hence** *x\_in\_N: x ∈# ?N*

**using** *X\_sub* **by** *blast*

**then obtain** *e'* **where**

*e'\_gt: e' > e* **and**

*x: x = encode 0 e'*

```

by (auto simp: Suc_le_eq dest: mem_hmsetmset_encodeD)

have x > encode 0 e
  unfolding x using ih(1)[OF n_nz] e'_gt by (blast dest: Suc_lessD)
}
hence ex_gt_e:  $\exists x \in \# X. x > \text{encode } 0 e$ 
  using X_nemp by auto

have X_sub':  $X \subseteq \# ?Na + ?N$ 
  using X_sub by (simp add: subset_mset.add_increasing)
have mam_eq:  $?Ma + ?M = ?Na + ?N - X + (Y + ?Pa)$ 
proof -
  from mod_ge have ?Ma = ?Na + ?Pa
    by (simp add: replicate_mset_plus [symmetric])
  moreover have  $?Na + ?N - X = ?Na + (?N - X)$ 
    by (meson X_sub multiset_diff_union_assoc)
  ultimately show ?thesis
    by (simp add: M)
qed
have max_X:  $\bigwedge k. k \in \# Y + ?Pa \implies \exists a. a \in \# X \wedge k < a$ 
  using ex_gt mod_ge ex_gt_e by (metis in_replicate_mset union_iff)

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

inductive aligned_e :: nat  $\Rightarrow$  hmultiset  $\Rightarrow$  bool where
  ( $\forall m \in \# \text{hmsetmset } M. m \geq \text{encode } 0 e \implies \text{aligned}_e e M$ )

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

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

  have well2:  $\forall M \in \# \text{hmsetmset } (\text{encode } (\text{Suc } e) (n \text{ div } \text{base})). \text{well\_base}_h M$ 
    using ih(2) well_base_h.cases by (metis Suc_eq_plus1 Zero_not_Suc count_empty div_0
    encode_0_iff hmsetmset_empty_iff in_countE)

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

  show ?case
  proof (rule well_base_h.intros)
    show  $\forall M \in \# \text{hmsetmset } (\text{encode } e n). \text{well\_base}_h M$ 
      by (subst encode.simps[unfolded of_nat_times_omega_exp],
          simp add: zero_hmultiset_def hmsetmset_plus, use ih(1) well2 in blast)
  next
    show well_base (hmsetmset (encode e n))
      using cnt1 base_ge_2

```

by (subst encode.simps[unfolded of\_nat\_times\_omega\_exp],  
simp add: well\_base.simps zero\_hmultiset\_def hmsetmset\_plus,  
metis ih(2) well\_base\_h.simps Suc\_eq\_plus1 less\_numerals\_extra(3) well\_base.simps)

qed  
qed

## 12.4 Decoding of Natural Numbers from Ordinals

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

**lemma** decode\_unfold: decode e M =  $(\sum m \in\# \text{hmsetmset } M. \text{base} \wedge \text{decode } 0 m) \text{ div } \text{base} \wedge e$   
**by** (cases M) simp

**lemma** decode\_0[simp]: decode e 0 = 0  
**unfolding** zero\_hmultiset\_def **by** simp

**inductive** aligned<sub>a</sub> :: nat  $\Rightarrow$  hmultiset  $\Rightarrow$  bool **where**  
 $(\forall m \in\# \text{hmsetmset } M. \text{decode } 0 m \geq e) \Longrightarrow \text{aligned}_a e M$

**lemma** aligned\_a\_0[simp]: aligned<sub>a</sub> 0 M  
**by** (rule aligned\_a.intros) simp

**lemma** aligned\_a\_mono\_exp\_Suc: aligned<sub>a</sub> (Suc e) M  $\Longrightarrow$  aligned<sub>a</sub> e M  
**by** (auto simp: aligned\_a.simps)

**lemma** aligned\_a\_mono\_hmset:  
**assumes** aligned<sub>a</sub> e M **and** hmsetmset M'  $\subseteq\#$  hmsetmset M  
**shows** aligned<sub>a</sub> e M'  
**using** assms **by** (auto simp: aligned\_a.simps)

**lemma** decode\_exp\_shift\_Suc:  
**assumes** align<sub>a</sub>: aligned<sub>a</sub> (Suc e) M  
**shows** decode e M = base \* decode (Suc e) M  
**proof** (subst (1 2) decode\_unfold, subst (1 2) sum\_mset\_distrib\_div\_if\_dvd)  
**note** align' = align<sub>a</sub>[unfolded aligned\_a.simps, simplified, unfolded Suc\_le\_eq]

**show**  $\forall m \in\# \text{hmsetmset } M. \text{base} \wedge \text{Suc } e \text{ dvd } \text{base} \wedge \text{decode } 0 m$   
**using** align' Suc\_leI le\_imp\_power\_dvd **by** blast

**show**  $\forall m \in\# \text{hmsetmset } M. \text{base} \wedge e \text{ dvd } \text{base} \wedge \text{decode } 0 m$   
**using** align' **by** (simp add: le\_imp\_power\_dvd le\_less)

**have** base\_e\_nz: base  $\wedge$  e  $\neq$  0  
**using** base\_ge\_2 **by** simp

**have** mult\_base:  
base  $\wedge$  decode 0 m div base  $\wedge$  e = base \* (base  $\wedge$  decode 0 m div (base \* base  $\wedge$  e))  
**if** m\_in: m  $\in\#$  hmsetmset M **for** m  
**using** m\_in align'  
**by** (subst power\_div\_same\_base[OF base\_e\_nz], force,  
metis Suc\_diff\_Suc Suc\_leI mult\_is\_0 power\_Suc power\_div\_same\_base power\_not\_zero)

**show**  $(\sum m \in\# \text{hmsetmset } M. \text{base} \wedge \text{decode } 0 m \text{ div } \text{base} \wedge e) =$   
base \*  $(\sum m \in\# \text{hmsetmset } M. \text{base} \wedge \text{decode } 0 m \text{ div } \text{base} \wedge \text{Suc } e)$   
**by** (auto simp: sum\_mset\_distrib\_left intro!: arg\_cong[of \_ \_ sum\_mset] image\_mset\_cong  
elim!: mult\_base)

qed

**lemma** decode\_exp\_shift:  
**assumes** aligned<sub>a</sub> e M  
**shows** decode 0 M = base  $\wedge$  e \* decode e M  
**using** assms **by** (induct e) (auto simp: decode\_exp\_shift\_Suc dest: aligned\_a\_mono\_exp\_Suc)

```

lemma decode_plus:
  assumes align_d_M: aligned_d e M
  shows decode e (M + N) = decode e M + decode e N
  using align_d_M[unfolded aligned_d.simps, simplified]
  by (subst (1 2 3) decode_unfold) (auto simp: hmsetmset_plus
    intro!: le_imp_power_dvd div_plus_div_distrib_dvd_left[OF sum_mset_dvd])

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

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

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

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

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

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

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

```

```

    have decode 0 x < decode 0 y
      by (rule ih[OF x_lt_M well_h_x aligned_d_0 aligned_d_0 x_lt_y])
  }
  thus x = y
    using x_in y_in dec_eq by (metis leI less_irrefl_nat order.not_eq_order_implies_strict)
qed
hence well_dec_Ma: well_base (image_mset (decode 0) (hmsetmset Ma))
  by (rule well_base_image_inj[OF well_base_h_imp_well_base[OF well_h_Ma]])

have H_bound:  $\forall m \in \# \text{hmsetmset } Ma. \text{decode } 0 \ m < \text{decode } 0 \ H$ 
proof
  fix m
  assume m_in:  $m \in \# \text{hmsetmset } Ma$ 

  have  $\forall m \in \# \text{hmsetmset } (\text{head } \omega \ Ma). \ m < H$ 
    using hds[unfolded H] using head_omega_def by auto
  hence m_lt_H:  $m < H$ 
    using m_in
  by (metis Max_less_iff_empty_iff_finite_set_mset head_omega_def hmultiset.sel insert_iff
    set_mset_add_mset_insert)

  have m_lt_M:  $m < M$ 
    using mem_hmsetmset_imp_less[OF m_in] by (simp add: M_trans_less_add2_hmset)

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

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

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

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

lemma decode_0_iff: well_base_h M  $\implies$  aligned_d e M  $\implies$  decode e M = 0  $\longleftrightarrow$  M = 0
  by (metis aligned_d_0 decode_0 decode_exp_shift encode_0 less_imp_decode_less mult_0_right neqE
    not_less_zero well_base_h_encode)

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

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

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

```



```

less_imp_decode_less[OF well_base_h_encode aligned_d_0 aligned_d_0] less_imp_encode_less)

show ?thesis
  using ih base_ge_2
  by (subst encode_simps[unfolded of_nat_times_omega_exp])
     (simp add: decode_plus[OF align_d_1[unfolded of_nat_times_omega_exp]]
           decode_exp_shift_Suc[OF align_d_2])
qed simp
qed

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

end

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

```

## 12.5 The Goodstein Sequence and Goodstein's Theorem

```

context
  fixes start :: nat
begin

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

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

  have two_le: 2 ≤ i + 3
    by simp

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

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

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

theorem goodsteins_theorem: ∃ i. goodstein i = 0
proof -

```

```

let ?G = λi. encode (i + 2) 0 (goodstein i)

obtain i where
  ¬ ?G i > ?G (i + 1)
  using wf_iff_no_infinite_down_chain[THEN iffD1, OF wf,
    unfolded not_ex not_all mem_Collect_eq prod.case, rule_format, of ?G]
  by auto
hence goodstein i = 0
  using goodstein_step by (metis add.assoc gr0I one_plus_numeral semiring_norm(3))
thus ?thesis
  by blast
qed

end

end

```

## 13 Towards Decidability of Behavioral Equivalence for Unary PCF

```

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

```

### 13.1 Preliminaries

```

lemma prod_UNIV: UNIV = UNIV × UNIV
  by auto

```

```

lemma infinite_cartesian_productI1: infinite A ⇒ B ≠ {} ⇒ infinite (A × B)
  by (auto dest!: finite_cartesian_productD1)

```

### 13.2 Types

```

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

```

```

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

```

```

primrec dest_fun where
  dest_fun B = []
| dest_fun (T → U) = T # dest_fun U

```

```

definition arity where
  arity T = length (dest_fun T)

```

```

lemma mk_fun_dest_fun[simp]: dest_fun T →→ B = T
  by (induct T) (auto simp: mk_fun_def)

```

```

lemma dest_fun_mk_fun[simp]: dest_fun (Ts →→ T) = Ts @ dest_fun T
  by (induct Ts) (auto simp: mk_fun_def)

```

```

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

```

```

lemma δ_mk_fun: δ (Ts →→ T) = HMSet (hmsetmset (δ T) + mset (map δ Ts))
  by (induct Ts) (auto simp: mk_fun_def)

```

```

lemma type_induct [case_names Fun]:
  assumes
    ( $\bigwedge T. (\bigwedge T1 T2. T = T1 \rightarrow T2 \implies P T1) \implies$ 
      $(\bigwedge T1 T2. T = T1 \rightarrow T2 \implies P T2) \implies P T$ )
  shows  $P T$ 
proof (induct T)
  case  $B$ 
  show  $?case$  by (rule assms) simp_all
next
  case Fun
  show  $?case$  by (rule assms) (insert Fun, simp_all)
qed

```

### 13.3 Terms

```

type-synonym name = string
type-synonym idx = nat
datatype expr =
  Var name * type ( $\langle \_ \rangle$ ) | Bound idx | B bool
  | Seq expr expr (infixr  $? 75$ ) | App expr expr (infixl  $\cdot 75$ )
  | Abs type expr ( $\Lambda \langle \_ \rangle \_$  [100, 100] 800)

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

```

```

notation (output)  $B \langle \_ \rangle$ 
notation (output)  $Bound \langle \_ \rangle$ 

```

```

primrec open :: idx  $\Rightarrow$  expr  $\Rightarrow$  expr  $\Rightarrow$  expr where
  open i t ( $j :: idx$ ) = (if  $i = j$  then  $t$  else  $j$ )
  | open i t  $\langle yU \rangle$  =  $\langle yU \rangle$ 
  | open i t ( $b :: bool$ ) =  $b$ 
  | open i t ( $e1 ? e2$ ) = open i t  $e1 ?$  open i t  $e2$ 
  | open i t ( $e1 \cdot e2$ ) = open i t  $e1 \cdot$  open i t  $e2$ 
  | open i t ( $\Lambda \langle U \rangle e$ ) =  $\Lambda \langle U \rangle$  (open ( $i + 1$ )  $t e$ )

```

```

abbreviation open0  $\equiv$  open 0
abbreviation open_Var i xT  $\equiv$  open i  $\langle xT \rangle$ 
abbreviation open0_Var xT  $\equiv$  open 0  $\langle xT \rangle$ 

```

```

primrec close_Var :: idx  $\Rightarrow$  name  $\times$  type  $\Rightarrow$  expr  $\Rightarrow$  expr where
  close_Var i xT ( $j :: idx$ ) =  $j$ 
  | close_Var i xT  $\langle yU \rangle$  = (if  $xT = yU$  then  $i$  else  $\langle yU \rangle$ )
  | 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 ( $e1 \cdot e2$ ) = close_Var i xT  $e1 \cdot$  close_Var i xT  $e2$ 
  | close_Var i xT ( $\Lambda \langle U \rangle e$ ) =  $\Lambda \langle U \rangle$  (close_Var ( $i + 1$ )  $xT e$ )

```

```

abbreviation close0_Var  $\equiv$  close_Var 0

```

```

primrec fv :: expr  $\Rightarrow$  (name  $\times$  type) fset where
  fv j ( $j :: idx$ ) =  $\{\}$ 
  | fv  $\langle yU \rangle$  =  $\{\langle yU \rangle\}$ 
  | fv ( $b :: bool$ ) =  $\{\}$ 
  | fv ( $e1 ? e2$ ) = fv  $e1 \cup$  | fv  $e2$ 
  | fv ( $e1 \cdot e2$ ) = fv  $e1 \cup$  | fv  $e2$ 
  | fv ( $\Lambda \langle U \rangle e$ ) = fv  $e$ 

```

```

abbreviation fresh x e  $\equiv$   $x \notin$  | fv  $e$ 

```

```

lemma ex_fresh:  $\exists x. (x :: char\ list, T) \notin$  |  $A$ 
proof (rule ccontr, unfold not_ex not_not)
  assume  $\forall x. (x, T) \in$  |  $A$ 

```

**then have** *infinite*  $\{x. (x, T) \mid \in A\}$  (**is** *infinite*  $?P$ )  
**by** (*auto simp add: infinite\_UNIV\_listI*)  
**moreover**  
**have**  $?P \subseteq \text{fst } \text{'fset } A$   
**by** (*force simp: fmember.rep\_eq*)  
**from** *finite\_surj[OF \_ this]* **have** *finite*  $?P$   
**by** *simp*  
**ultimately show** *False* **by** *blast*  
**qed**

**inductive** *lc* **where**

$lc\_Var[simp]: lc \langle xT \rangle$   
 $lc\_B[simp]: lc (b :: bool)$   
 $lc\_Seq: lc e1 \implies lc e2 \implies lc (e1 ? e2)$   
 $lc\_App: lc e1 \implies lc e2 \implies lc (e1 \cdot e2)$   
 $lc\_Abs: (\forall x. (x, T) \mid \notin X \longrightarrow lc (open0\_Var (x, T) e)) \implies lc (\Lambda \langle T \rangle e)$

**declare** *lc.intros[intro]*

**definition** *body*  $T t \equiv (\exists X. \forall x. (x, T) \mid \notin X \longrightarrow lc (open0\_Var (x, T) t))$

**lemma** *lc\_Abs\_iff\_body*:  $lc (\Lambda \langle T \rangle t) \longleftrightarrow \text{body } T t$   
**unfolding** *body\_def* **by** (*subst lc.simps*) *simp*

**lemma** *fv\_open\_Var*:  $\text{fresh } xT t \implies \text{fv } (open\_Var i xT t) \mid \subseteq \text{finsert } xT (\text{fv } t)$   
**by** (*induct t arbitrary: i*) *auto*

**lemma** *fv\_close\_Var[simp]*:  $\text{fv } (close\_Var i xT t) = \text{fv } t \mid - \{ \mid xT \}$   
**by** (*induct t arbitrary: i*) *auto*

**lemma** *close\_Var\_open\_Var[simp]*:  $\text{fresh } xT t \implies \text{close\_Var } i xT (\text{open\_Var } i xT t) = t$   
**by** (*induct t arbitrary: i*) *auto*

**lemma** *open\_Var\_inj*:  $\text{fresh } xT t \implies \text{fresh } xT u \implies \text{open\_Var } i xT t = \text{open\_Var } i xT u \implies t = u$   
**by** (*metis close\_Var\_open\_Var*)

**context** **begin**

**private lemma** *open\_Var\_open\_Var\_close\_Var*:  $i \neq j \implies xT \neq yU \implies \text{fresh } yU t \implies \text{open\_Var } i yU (\text{open\_Var } j zV (\text{close\_Var } j xT t)) = \text{open\_Var } j zV (\text{close\_Var } j xT (\text{open\_Var } i yU t))$   
**by** (*induct t arbitrary: i j*) *auto*

**lemma** *open\_Var\_close\_Var[simp]*:  $lc t \implies \text{open\_Var } i xT (\text{close\_Var } i xT t) = t$

**proof** (*induction t arbitrary: i rule: lc.induct*)

**case** (*lc\_Abs*  $T X e i$ )

**obtain**  $x$  **where**  $x: \text{fresh } (x, T) e (x, T) \neq xT (x, T) \mid \notin X$

**using** *ex\_fresh[of \_ fv e | $\cup$ ] finsert xT X]* **by** *blast*

**with** *lc\_Abs.IH* **have**  $lc (open0\_Var (x, T) e)$

$\text{open\_Var } (i + 1) xT (\text{close\_Var } (i + 1) xT (\text{open0\_Var } (x, T) e)) = \text{open0\_Var } (x, T) e$

**by** *auto*

**with**  $x$  **show**  $?case$

**by** (*auto simp: open\_Var\_open\_Var\_close\_Var*

*dest: fset\_mp[OF fv\_open\_Var, rotated]*

*intro!: open\_Var\_inj[of (x, T) \_ e 0]*)

**qed** *auto*

**end**

**lemma** *close\_Var\_inj*:  $lc t \implies lc u \implies \text{close\_Var } i xT t = \text{close\_Var } i xT u \implies t = u$   
**by** (*metis open\_Var\_close\_Var*)

**primrec** *Apps* (**infixl**  $\cdot$  75) **where**

$f \cdot [] = f$

|  $f \cdot (x \# xs) = f \cdot x \cdot xs$

**lemma** *Apps\_snoc*:  $f \cdot (xs @ [x]) = f \cdot xs \cdot x$   
**by** (*induct xs arbitrary*: *f*) *auto*

**lemma** *Apps\_append*:  $f \cdot (xs @ ys) = f \cdot xs \cdot ys$   
**by** (*induct xs arbitrary*: *f*) *auto*

**lemma** *Apps\_inj[simp]*:  $f \cdot ts = g \cdot ts \iff f = g$   
**by** (*induct ts arbitrary*: *f g*) *auto*

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

**lemma** *Apps\_Var\_eq[simp]*:  $\langle xT \rangle \cdot ss = \langle yU \rangle \cdot ts \iff xT = yU \wedge ss = ts$   
**proof** (*induct ss arbitrary*: *ts rule*: *rev\_induct*)  
**case** *snoc*  
**then show** *?case* **by** (*induct ts rule*: *rev\_induct*) (*auto simp*: *Apps\_snoc*)  
**qed** *auto*

**lemma** *Apps\_Abs\_neq\_Apps[simp, symmetric, simp]*:  
 $\Lambda \langle T \rangle r \cdot t \neq \langle xT \rangle \cdot ss$   
 $\Lambda \langle T \rangle r \cdot t \neq (i :: \text{idx}) \cdot ss$   
 $\Lambda \langle T \rangle r \cdot t \neq (b :: \text{bool}) \cdot ss$   
 $\Lambda \langle T \rangle r \cdot t \neq (e1 \ ? \ e2) \cdot ss$   
**by** (*induct ss rule*: *rev\_induct*) (*auto simp*: *Apps\_snoc*)

**lemma** *App\_Abs\_eq\_Apps\_Abs[simp]*:  $\Lambda \langle T \rangle r \cdot t = \Lambda \langle T' \rangle r' \cdot ss \iff T = T' \wedge r = r' \wedge ss = [t]$   
**by** (*induct ss rule*: *rev\_induct*) (*auto simp*: *Apps\_snoc*)

**lemma** *Apps\_Var\_neq\_Apps\_Abs[simp, symmetric, simp]*:  $\langle xT \rangle \cdot ss \neq \Lambda \langle T \rangle r \cdot ts$   
**proof** (*induct ss arbitrary*: *ts rule*: *rev\_induct*)  
**case** (*snoc a ss*)  
**then show** *?case* **by** (*induct ts rule*: *rev\_induct*) (*auto simp*: *Apps\_snoc*)  
**qed** *simp*

**lemma** *Apps\_Var\_neq\_Apps\_beta[simp, THEN not\_sym, simp]*:  
 $\langle xT \rangle \cdot ss \neq \Lambda \langle T \rangle r \cdot s \cdot ts$   
**by** (*metis Apps\_Var\_neq\_Apps\_Abs Apps\_append Apps\_snoc eq\_Apps\_conv(9)*)

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

**lemma** *fold\_eq\_Bool\_iff[simp]*:

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

**lemma** *fold\_eq\_Fun\_iff[simp]*:  
 $fold (\rightarrow) (rev Ts) T = U \rightarrow V \longleftrightarrow$   
 $(Ts = [] \wedge T = U \rightarrow V \vee (\exists Us. Ts = U \# Us \wedge fold (\rightarrow) (rev Us) T = V))$   
**by** (*induct Ts*) *auto*

## 13.4 Substitution

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

**lemma** *fv\_subst*:  
 $fv\ (subst\ xT\ t\ u) = fv\ u\ |-| \{|xT|\}\ |\cup| (if\ xT\ |\in|\ fv\ u\ then\ fv\ t\ else\ \{||\})$   
**by** (*induct u*) *auto*

**lemma** *subst\_fresh*:  $fresh\ xT\ u \implies subst\ xT\ t\ u = u$   
**by** (*induct u*) *auto*

**context begin**

**private lemma** *open\_open\_id*:  $i \neq j \implies open\ i\ t\ (open\ j\ t'\ u) = open\ j\ t'\ u \implies open\ i\ t\ u = u$   
**by** (*induct u arbitrary: i j*) (*auto 6 0*)

**lemma** *lc\_open\_id*:  $lc\ u \implies open\ k\ t\ u = u$

**proof** (*induct u arbitrary: k rule: lc.induct*)  
**case** (*lc\_Abs T X e*)  
**obtain**  $x$  **where**  $x: fresh\ (x, T)\ e\ (x, T)\ |\notin|\ X$   
**using** *ex\_fresh[of \_ fv e |\cup| X]* **by** *blast*  
**with** *lc\_Abs show ?case*  
**by** (*auto intro: open\_open\_id dest: spec[of \_ x] spec[of \_ Suc k]*)  
**qed** *auto*

**lemma** *subst\_open*:  $lc\ u \implies subst\ xT\ u\ (open\ i\ t\ v) = open\ i\ (subst\ xT\ u\ t)\ (subst\ xT\ u\ v)$   
**by** (*induction v arbitrary: i*) (*auto intro: lc\_open\_id[symmetric]*)

**lemma** *subst\_open\_Var*:  
 $xT \neq yU \implies lc\ u \implies subst\ xT\ u\ (open\_Var\ i\ yU\ v) = open\_Var\ i\ yU\ (subst\ xT\ u\ v)$   
**by** (*auto simp: subst\_open*)

**lemma** *subst\_Apps[simp]*:  
 $subst\ xT\ u\ (f \cdot xs) = subst\ xT\ u\ f \cdot map\ (subst\ xT\ u)\ xs$   
**by** (*induct xs arbitrary: f*) *auto*

**end**

**context begin**

**private lemma** *fresh\_close\_Var\_id*:  $fresh\ xT\ t \implies close\_Var\ k\ xT\ t = t$   
**by** (*induct t arbitrary: k*) *auto*

**lemma** *subst\_close\_Var*:  
 $xT \neq yU \implies fresh\ yU\ u \implies subst\ xT\ u\ (close\_Var\ i\ yU\ t) = close\_Var\ i\ yU\ (subst\ xT\ u\ t)$   
**by** (*induct t arbitrary: i*) (*auto simp: fresh\_close\_Var\_id*)

**end**

**lemma** *subst\_intro*:  $\text{fresh } xT \ t \implies \text{lc } u \implies \text{open0 } u \ t = \text{subst } xT \ u \ (\text{open0\_Var } xT \ t)$   
**by** (*auto simp: subst\_fresh subst\_open*)

**lemma** *lc\_subst[simp]*:  $\text{lc } u \implies \text{lc } t \implies \text{lc } (\text{subst } xT \ t \ u)$   
**proof** (*induct u rule: lc.induct*)  
**case** (*lc\_Abs*  $T \ X \ e$ )  
**then show** *?case*  
**by** (*auto simp: subst\_open\_Var intro!: lc.lc\_Abs[of \_ fv e | $\cup$ |  $X$  | $\cup$ |  $\{xT\}$ ]*)  
**qed auto**

**lemma** *body\_subst[simp]*:  $\text{body } U \ u \implies \text{lc } t \implies \text{body } U \ (\text{subst } xT \ t \ u)$   
**proof** (*subst (asm) body\_def, elim conjE exE*)  
**fix**  $X$   
**assume** [*simp*]:  $\text{lc } t \ \forall x. (x, U) \notin X \longrightarrow \text{lc } (\text{open0\_Var } (x, U) \ u)$   
**show**  $\text{body } U \ (\text{subst } xT \ t \ u)$   
**proof** (*unfold body\_def, intro exI[of \_ finsert xT X] conjI allI impI*)  
**fix**  $x$   
**assume**  $(x, U) \notin \text{finsert } xT \ X$   
**then show**  $\text{lc } (\text{open0\_Var } (x, U) \ (\text{subst } xT \ t \ u))$   
**by** (*auto simp: subst\_open\_Var[symmetric]*)  
**qed**  
**qed**

**lemma** *lc\_open\_Var*:  $\text{lc } u \implies \text{lc } (\text{open\_Var } i \ xT \ u)$   
**by** (*simp add: lc\_open\_id*)

**lemma** *lc\_open[simp]*:  $\text{body } U \ u \implies \text{lc } t \implies \text{lc } (\text{open0 } t \ u)$   
**proof** (*unfold body\_def, elim conjE exE*)  
**fix**  $X$   
**assume** [*simp*]:  $\text{lc } t \ \forall x. (x, U) \notin X \longrightarrow \text{lc } (\text{open0\_Var } (x, U) \ u)$   
**with** *ex\_fresh*[*of \_ fv u | $\cup$ |  $X$ ] **obtain**  $x$  **where** [*simp*]:  $\text{fresh } (x, U) \ u \ (x, U) \notin X$  **by** *blast*  
**show** *?thesis* **by** (*subst subst\_intro[of (x, U)] auto*)  
**qed***

## 13.5 Typing

**inductive** *welltyped* ::  $\text{expr} \Rightarrow \text{type} \Rightarrow \text{bool}$  (**infix** :: 60) **where**  
*welltyped\_Var*[*intro!*]:  $\langle(x, T)\rangle \text{ :: } T$   
*welltyped\_B*[*intro!*]:  $(b \text{ :: } \text{bool}) \text{ :: } \mathcal{B}$   
*welltyped\_Seq*[*intro!*]:  $e1 \text{ :: } \mathcal{B} \implies e2 \text{ :: } \mathcal{B} \implies e1 \ ? \ e2 \text{ :: } \mathcal{B}$   
*welltyped\_App*[*intro!*]:  $e1 \text{ :: } T \rightarrow U \implies e2 \text{ :: } T \implies e1 \cdot e2 \text{ :: } U$   
*welltyped\_Abs*[*intro!*]:  $(\forall x. (x, T) \notin X \longrightarrow \text{open0\_Var } (x, T) \ e \text{ :: } U) \implies \Lambda\langle T \rangle \ e \text{ :: } T \rightarrow U$

**inductive-cases** *welltypedE*[*elim!*]:

$\langle x \rangle \text{ :: } T$   
 $(i \text{ :: } \text{idx}) \text{ :: } T$   
 $(b \text{ :: } \text{bool}) \text{ :: } T$   
 $e1 \ ? \ e2 \text{ :: } T$   
 $e1 \cdot e2 \text{ :: } T$   
 $\Lambda\langle T \rangle \ e \text{ :: } U$

**lemma** *welltyped\_unique*:  $t \text{ :: } T \implies t \text{ :: } U \implies T = U$   
**proof** (*induction t T arbitrary: U rule: welltyped.induct*)  
**case** (*welltyped\_Abs*  $T \ X \ t \ U \ T'$ )  
**from** *welltyped\_Abs.prem*s **show** *?case*  
**proof** (*elim welltypedE*)  
**fix**  $Y \ U'$   
**obtain**  $x$  **where** [*simp*]:  $(x, T) \notin X \ (x, T) \notin Y$   
**using** *ex\_fresh*[*of \_ X | $\cup$ | Y*] **by** *blast*  
**assume** [*simp*]:  $T' = T \rightarrow U' \ \forall x. (x, T) \notin Y \longrightarrow \text{open0\_Var } (x, T) \ t \text{ :: } U'$   
**show**  $T \rightarrow U = T'$   
**by** (*auto intro: conjunct2[OF welltyped\_Abs.IH[rule\_format], rule\_format, of x]*)  
**qed**  
**qed blast+**

**lemma** *welltyped\_lc[simp]*:  $t :: T \implies lc\ t$   
**by** (*induction t T rule: welltyped.induct*) *auto*

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

**lemma** *rename\_welltyped*:  $u :: U \implies subst\ (x, T)\ \langle(y, T)\rangle\ u :: U$   
**by** (*rule welltyped\_subst*) *auto*

**lemma** *welltyped\_Abs\_fresh*:  
**assumes** *fresh (x, T) u open0\_Var (x, T) u :: U*  
**shows**  $\Lambda\langle T \rangle\ u :: T \rightarrow U$   
**proof** (*intro welltyped\_Abs[of \_ fv u] allI impI*)  
**fix** *y*  
**assume** *fresh (y, T) u*  
**with** *assms(2) have*  $subst\ (x, T)\ \langle(y, T)\rangle\ (open0\_Var\ (x, T)\ u) :: U$  (*is ?t :: \_*)  
**by** (*auto intro: rename\_welltyped*)  
**also have**  $?t = open0\_Var\ (y, T)\ u$   
**by** (*subst subst\_intro[symmetric]*) (*auto simp: assms(1)*)  
**finally show**  $open0\_Var\ (y, T)\ u :: U$  .  
**qed**

**lemma** *Apps\_alt*:  $f \cdot ts :: T \longleftrightarrow$   
 $(\exists Ts. f :: fold\ (\rightarrow)\ (rev\ Ts)\ T \wedge list\_all2\ (:::)\ ts\ Ts)$   
**proof** (*induct ts arbitrary: f*)  
**case** (*Cons t ts*)  
**from** *Cons(1)[of f · t] show* *?case*  
**by** (*force simp: list\_all2\_Cons1*)  
**qed simp**

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

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

**primrec** *constant0* **where**  
 $constant0\ \mathcal{B} = Var\ ("bool", \mathcal{B})$   
 $| constant0\ (T \rightarrow U) = \Lambda\langle T \rangle\ (constant0\ U)$

**definition** *constant T*  $= \Lambda\langle \mathcal{B} \rangle\ (close0\_Var\ ("bool", \mathcal{B})\ (constant0\ T))$

**lemma** *fv\_constant0[simp]*:  $fv\ (constant0\ T) = \{("bool", \mathcal{B})\}$   
**by** (*induct T*) *auto*

**lemma** *closed\_constant[simp]*:  $closed\ (constant\ T)$   
**unfolding** *constant\_def* **by** *auto*

**lemma** *welltyped\_constant0[simp]*:  $constant0\ T :: T$   
**by** (*induct T*) (*auto simp: lc\_open\_id*)

**lemma** *lc\_constant0[simp]*:  $lc\ (constant0\ T)$   
**using** *welltyped\_constant0 welltyped\_lc* **by** *blast*

**lemma** *welltyped\_constant[simp]*:  $constant\ T :: \mathcal{B} \rightarrow T$   
**unfolding** *constant\_def* **by** (*auto intro: welltyped\_Abs\_fresh[of "bool"]*)

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



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

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

**lemma** *size\_nth\_arg[simp]*:  $i < ar\ T \implies size\ (T\ !-\ i) < size\ T$   
**by** (*induct* *T* *arbitrary*: *i*) (*force simp: nth\_Cons' nth\_arg\_def gr0\_conv\_Suc*)<sup>+</sup>

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

**theorem**  $\pi\_induct$ [*rotated -2, consumes 2, case\_names 0 Suc*]:  
**assumes**  $\bigwedge T\ i.\ i < ar\ T \implies P\ T\ i\ 0$   
**and**  $\bigwedge T\ i\ j.\ i < ar\ T \implies j < ar\ (T!-i) \implies P\ (T!-i)\ j\ 0 \implies$   
 $(\forall x < ar\ (T!-i!-j).\ P\ (T!-i)\ j\ (x + 1)) \implies P\ T\ i\ (j + 1)$   
**shows**  $i < ar\ T \implies j \leq ar\ (T!-i) \implies P\ T\ i\ j$   
**by** (*induct* *T* *i* *j* *rule: pi.induct*) (*auto intro: assms[simplified]*)

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

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

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

**definition** *variant* *k* *base* = *base* @ *replicate* *k* *CHR* *"\*"*

**lemma** *variant\_inj*: *variant* *i* *base* = *variant* *j* *base*  $\implies i = j$   
**unfolding** *variant\_def* **by** *auto*

**lemma** *variant\_inj2*:  
 $CHR\ ".*" \notin set\ b1 \implies CHR\ ".*" \notin set\ b2 \implies variant\ i\ b1 = variant\ j\ b2 \implies b1 = b2$   
**unfolding** *variant\_def*  
**by** (*auto simp: append\_eq\_append\_conv2*)  
(*metis Nil\_is\_append\_conv hd\_append2 hd\_in\_set hd\_rev last\_ConsR last\_snoc\_replicate\_append\_same rev\_replicate*)<sup>+</sup>

**fun** *E* ::  $type \Rightarrow nat \Rightarrow expr$  **and** *P* ::  $type \Rightarrow nat \Rightarrow nat \Rightarrow expr$  **where**  
 $E\ T\ i = (if\ i < ar\ T\ then\ (let$   
 $Ti = T!-i;$   
 $x = \lambda k.\ (variant\ k\ "x",\ T!-k);$   
 $xs = map\ x\ [0\ ..<\ ar\ T];$   
 $xx\_var = \langle nth\ xs\ i \rangle;$   
 $x\_vars = map\ (\lambda x.\ \langle x \rangle)\ (nth\_drop\ i\ xs);$   
 $yy = ("z",\ \pi\ T\ i\ 0);$   
 $yy\_var = \langle yy \rangle;$   
 $y = \lambda j.\ (variant\ j\ "y",\ \pi\ T\ i\ (j + 1));$   
 $ys = map\ y\ [0\ ..<\ ar\ Ti];$   
 $e = \lambda j.\ \langle y\ j \rangle \cdot (P\ Ti\ j\ 0 \cdot xx\_var \# map\ (\lambda k.\ P\ Ti\ j\ (k + 1) \cdot xx\_var)\ [0\ ..<\ ar\ (Ti!-j)]);$   
 $guards = map\ (\lambda i.\ xx\_var \cdot$   
 $map\ (\lambda j.\ constant\ (Ti!-j) \cdot (if\ i = j\ then\ e\ i \cdot x\_vars\ else\ True))\ [0\ ..<\ ar\ Ti])$   
 $[0\ ..<\ ar\ Ti]$   
 $in\ \Lambda[(yy\ \#\ ys\ @\ xs)]\ (guards\ ??\ (yy\_var \cdot x\_vars))\ else\ constant\ (\varepsilon\ T\ i) \cdot False)$   
 $| P\ T\ i\ 0 =$   
 $(if\ i < ar\ T\ then\ (let$   
 $f = ("f",\ T);$   
 $f\_var = \langle f \rangle;$

```

    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 = T!-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** *Abs\_Nil[simp]*:  $\Lambda[\square] b = b$   
**unfolding** *Abs\_def* **by** *simp*

**lemma** *Abs\_Cons[simp]*:  $\Lambda[(x\#xs)] b = \Lambda(\text{snd } x) (\text{close0\_Var } x (\Lambda[xs] b))$   
**unfolding** *Abs\_def* **by** *simp*

**lemma** *welltyped\_Abs*:  $b :: U \Longrightarrow T = \text{map snd } xTs \rightarrow \rightarrow U \Longrightarrow \Lambda[xTs] b :: T$   
**by** (*hypsubst\_thin*, *induct xTs*) (*auto simp: mk\_fun\_def intro!: welltyped\_Abs\_fresh*)

**lemma** *welltyped\_Apps*:  $\text{list\_all2 } (:::) ts Ts \Longrightarrow f :: Ts \rightarrow \rightarrow U \Longrightarrow f \cdot ts :: U$   
**by** (*induct ts Ts arbitrary: f rule: list\_rel\_induct*) (*auto simp: mk\_fun\_def*)

**lemma** *welltyped\_open\_Var\_close\_Var[intro!]*:  
 $t :: T \Longrightarrow \text{open0\_Var } xT (\text{close0\_Var } xT t) :: T$   
**by** *auto*

**lemma** *welltyped\_Var\_iff[simp]*:  
 $\langle(x, T)\rangle :: U \longleftrightarrow T = U$   
**by** *auto*

**lemma** *welltyped\_bool\_iff[simp]*:  $(b :: \text{bool}) :: T \longleftrightarrow T = \mathcal{B}$   
**by** *auto*

**lemma** *welltyped\_constant0\_iff[simp]*:  $\text{constant0 } T :: U \longleftrightarrow (U = T)$   
**by** (*induct T arbitrary: U*) (*auto simp: ex\_fresh lc\_open\_id*)

**lemma** *welltyped\_constant\_iff[simp]*:  $\text{constant } T :: U \longleftrightarrow (U = \mathcal{B} \rightarrow T)$   
**unfolding** *constant\_def*

**proof** (*intro iffI*, *elim welltypedE*, *hypsubst\_thin*, *unfold type.inject simp\_thms*)

```

fix X U
assume ∀x. (x, B) |≠| X → open0_Var (x, B) (close0_Var ("bool", B) (constant0 T)) :: U
moreover obtain x where (x, B) |≠| X using ex_fresh[of B X] by blast
ultimately have open0_Var (x, B) (close0_Var ("bool", B) (constant0 T)) :: U by simp
then have open0_Var ("bool", B) (close0_Var ("bool", B) (constant0 T)) :: U
  using rename_welltyped[of ⟨open0_Var (x, B) (close0_Var ("bool", B) (constant0 T))⟩
    U x B "bool"]
  by (auto simp: subst_open subst_fresh)
then show U = T by auto
qed (auto intro!: welltyped_Abs_fresh)

```

**lemma** *welltyped\_Seq\_iff[simp]*:  $e1 ? e2 :: T \longleftrightarrow (T = \mathcal{B} \wedge e1 :: \mathcal{B} \wedge e2 :: \mathcal{B})$   
**by** *auto*

**lemma** *welltyped\_Seqs\_iff[simp]*:  $es ?? e :: T \longleftrightarrow$

$((es \neq [] \rightarrow T = \mathcal{B}) \wedge (\forall e \in set\ es.\ e ::: \mathcal{B}) \wedge e ::: T)$   
**by** (induct es arbitrary: e) (auto simp: Seqs\_def)

**lemma** welltyped\_App\_iff[simp]:  $f \cdot t ::: U \longleftrightarrow (\exists T.\ f ::: T \rightarrow U \wedge t ::: T)$   
**by** auto

**lemma** welltyped\_Apps\_iff[simp]:  $f \cdot ts ::: U \longleftrightarrow (\exists Ts.\ f ::: Ts \rightarrow U \wedge list\_all2\ (:::)\ ts\ Ts)$   
**by** (induct ts arbitrary: f) (auto 0 3 simp: mk\_fun\_def list\_all2\_Cons1 intro: exI[of \_ \_ # \_])

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

**lemma** map\_nth\_eq\_drop\_take[simp]:  $j \leq length\ xs \implies map\ (nth\ xs)\ [i..<j] = drop\ i\ (take\ j\ xs)$   
**by** (induct j) (auto simp: take\_Suc\_conv\_app\_nth)

**lemma** dest\_fun\_pi\_0:  $i < ar\ T \implies dest\_fun\ (\pi\ T\ i\ 0) = nth\_drop\ i\ (dest\_fun\ T)$   
**by** auto

**lemma** welltyped\_E:  $E\ T\ i ::: \varepsilon\ T\ i$  **and** welltyped\_P:  $P\ T\ i\ j ::: T \rightarrow \pi\ T\ i\ j$   
**proof** (induct T i **and** T i j rule: E\_P.induct)  
**case** (1 T i)  
**note** P.simps[simp del]  $\pi$ .simps[simp del]  $\varepsilon\_def$ [simp] nth\_drop\_def[simp] nth\_arg\_def[simp]  
**from** 1(1)[OF \_ refl refl refl refl refl refl refl refl refl]  
1(2)[OF \_ refl refl refl refl refl refl refl refl refl]  
**show** ?case  
**by** (auto 0 4 simp: Let\_def o\_def take\_map[symmetric] drop\_map[symmetric]  
list\_all2\_conv\_all\_nth nth\_append min\_def dest\_fun\_pi\_0  $\pi$ .simps[of T i]  
intro!: welltyped\_Abs\_fresh welltyped\_Abs[of \_  $\mathcal{B}$ ])

**next**  
**case** (2 T i)  
**show** ?case  
**by** (auto simp: Let\_def take\_map drop\_map o\_def list\_all2\_conv\_all\_nth nth\_append nth\_Cons'  
nth\_drop\_def nth\_arg\_def  
intro!: welltyped\_constant welltyped\_Abs\_fresh welltyped\_Abs[of \_  $\mathcal{B}$ ])

**next**  
**case** (3 T i j)  
**note** E.simps[simp del]  $\pi$ .simps[simp del] Abss\_Cons[simp del]  $\varepsilon\_def$ [simp]  
nth\_drop\_def[simp] nth\_arg\_def[simp]  
**from** 3(1)[OF \_ refl refl refl refl refl refl refl refl refl refl]  
**show** ?case  
**by** (auto 0 3 simp: Let\_def o\_def take\_map[symmetric] drop\_map[symmetric]  
list\_all2\_conv\_all\_nth nth\_append nth\_Cons' min\_def  $\pi$ .simps[of T i]  
intro!: welltyped\_Abs\_fresh welltyped\_Abs[of \_  $\mathcal{B}$ ])

**qed**

**lemma**  $\delta\_gt\_0$ [simp]:  $T \neq \mathcal{B} \implies HMSets\ \{\#\} < \delta\ T$   
**by** (cases T) auto

**lemma** mset\_nth\_drop\_less:  $i < length\ xs \implies mset\ (nth\_drop\ i\ xs) < mset\ xs$   
**by** (induct xs arbitrary: i) (auto simp: take\_Cons' nth\_drop\_def gr0\_conv\_Suc)

**lemma** map\_nth\_drop:  $i < length\ xs \implies map\ f\ (nth\_drop\ i\ xs) = nth\_drop\ i\ (map\ f\ xs)$   
**by** (induct xs arbitrary: i) (auto simp: take\_Cons' nth\_drop\_def gr0\_conv\_Suc)

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

**lemma** dest\_fun\_alt:  $dest\_fun\ T = map\ (\lambda i.\ T\ !- i)\ [0..<ar\ T]$   
**unfolding** list\_eq\_iff\_nth\_eq nth\_arg\_def **by** auto

**context** notes  $\pi$ .simps[simp del] **notes** One\_nat\_def[simp del] **begin**

**lemma**  $\delta\_pi$ :

```

assumes  $i < ar\ T\ j \leq ar\ (T\ !- i)$ 
shows  $\delta\ (\pi\ T\ i\ j) < \delta\ T$ 
using assms proof (induct  $T\ i\ j$  rule:  $\pi\_induct$ )
  fix  $T\ i$ 
  assume  $i < ar\ T$ 
  then show  $\delta\ (\pi\ T\ i\ 0) < \delta\ T$ 
    by (subst (2) mk_fun_dest_fun[symmetric, of  $T$ ], unfold  $\delta\_mk\_fun$ )
      (auto simp:  $\delta\_mk\_fun\ mset\_map$ [symmetric] take\_map[symmetric] drop\_map[symmetric]  $\pi.simps$ 
        mset_nth_drop_less map_nth_drop simp del: mset\_map)
  next
  fix  $T\ i\ j$ 
  let  $?Ti = T\ !- i$ 
  assume [rule_format, simp]:  $i < ar\ T\ j < ar\ ?Ti\ \delta\ (\pi\ ?Ti\ j\ 0) < \delta\ ?Ti$ 
     $\forall k < ar\ (?Ti\ !- j). \delta\ (\pi\ ?Ti\ j\ (k + 1)) < \delta\ ?Ti$ 
  define  $X$  and  $Y$  and  $M$  where
    [simp]:  $X = \{\#\delta\ ?Ti\#\}$  and
    [simp]:  $Y = \{\#\delta\ (\pi\ ?Ti\ j\ 0)\#\} + \{\#\delta\ (\pi\ ?Ti\ j\ (k + 1)). k \in \# mset\ [0 ..< ar\ (?Ti\ !- j)]\#\}$  and
    [simp]:  $M \equiv \{\#\delta\ z. z \in \# mset\ (nth\_drop\ i\ (dest\_fun\ T))\#\}$ 
  have  $\delta\ (\pi\ T\ i\ (j + 1)) = HMSet\ (Y + M)$ 
    by (auto simp: One_nat_def  $\pi.simps\ \delta\_mk\_fun$ )
  also have  $Y + M < X + M$ 
    unfolding less_multisetDM by (rule exI[of  $X$ ], rule exI[of  $Y$ ]) auto
  also have  $HMSet\ (X + M) = \delta\ T$ 
    unfolding  $M\_def$ 
    by (subst (2) mk_fun_dest_fun[symmetric, of  $T$ ], subst (2) id_take_nth_drop[of  $i\ dest\_fun\ T$ ])
      (auto simp:  $\delta\_mk\_fun\ nth\_arg\_def\ nth\_drop\_def$ )
  finally show  $\delta\ (\pi\ T\ i\ (j + 1)) < \delta\ T$  by simp
qed

end

end

```