

Formalization of Countable Multisets

Mathias Schack Rabing Dmitriy Traytel

June 25, 2026

Abstract

We define *countable multisets* as a generalization of finite multisets. Countable multisets are equivalently represented as functions from elements to extended natural numbers ($\mathbb{N} \cup \{\infty\}$) returning non-zero values for countably many elements, or as quotients of lazy lists modulo infinitary permutations. We register countable multisets as a bounded natural functor (BNF), enabling nested (co)recursion through them in (co)datatypes. We further define the subtype of *countably infinite multisets* and register it as a BNF, too.

Contents

1	Infinite Sums over Extended Natural Numbers	2
1.1	Preliminaries	2
1.2	Infinite Summation	4
2	Miscellaneous (Mostly About Lazy Lists)	7
3	enat as a Codatatype	8
4	Counting in Lazy Lists	11
5	Lazy Lists of Natural Numbers	12
6	Permutations of Lazy Lists	13
7	Same-Count Equivalence Relation	14
8	Countable Multisets as a Quotient	15
9	Lazy List Interleaving	15
10	Countable Multisets as a Subtype	20
11	Countably Infinite Multisets	27

1 Infinite Sums over Extended Natural Numbers

This is a theory of infinite sums of extended natural numbers defined as limits of finite sums. The goal is to make reasoning about these infinite sums almost as easy as that about finite sums.

theory *Infinite-Sums-Enat*
imports *HOL-Library.Countable-Set HOL-Library.Extended-Nat*
begin

1.1 Preliminaries

lemma *enat-pm-iff*:

$$\begin{aligned} \bigwedge a b c. b \leq c &\implies (a::enat) + b \leq c \iff a \leq c - b \\ \bigwedge a b c. a \leq c &\implies (a::enat) + b \leq c \iff b \leq c - a \\ \bigwedge a b c. a \leq b &\implies c \leq b \implies (a::enat) \leq b - c \iff c \leq b - a \end{aligned}$$

<proof>

lemma *disjoint-finite-aux*:

$$\forall i \in I. \forall j \in I. i \neq j \implies A i \cap A j = \{\} \implies B \subseteq \bigcup (A \text{ ' } I) \implies \text{finite } B \implies \text{finite } \{i \in I. B \cap A i \neq \{\}\}$$

<proof>

lemma *incl-UNION-aux*: $B \subseteq \bigcup (A \text{ ' } I) \implies B = \bigcup ((\lambda i. (B \cap A i)) \text{ ' } \{i \in I. B \cap A i \neq \{\}\})$

<proof>

lemma *incl-UNION-aux2*: $B \subseteq \bigcup (A \text{ ' } I) \iff B = \bigcup ((\lambda i. (B \cap A i)) \text{ ' } I)$

<proof>

lemma *sum-singl[simp]*: $\text{sum } f \{a\} = f a$

<proof>

lemma *sum-two[simp]*: $a1 \neq a2 \implies \text{sum } f \{a1, a2\} = f a1 + f a2$

<proof>

lemma *sum-three[simp]*: $a1 \neq a2 \implies a1 \neq a3 \implies a2 \neq a3 \implies$

$$\text{sum } f \{a1, a2, a3\} = f a1 + f a2 + f a3$$

<proof>

lemma *Sup-leq*:

$$A \neq \{\} \implies \forall a \in A. \exists b \in B. (a::enat) \leq b \implies \text{Sup } A \leq \text{Sup } B$$

<proof>

lemma *Sup-image-leq*:

$$A \neq \{\} \implies \forall a \in A. \exists b \in B. (f a::enat) \leq g b \implies$$

$$\text{Sup } (f \text{ ' } A) \leq \text{Sup } (g \text{ ' } B)$$

<proof>

lemma *Sup-cong*:

assumes $A \neq \{\}$ $\vee B \neq \{\}$ $\forall a \in A. \exists b \in B. (a::\text{enat}) \leq b \vee b \in B. \exists a \in A. (b::\text{enat}) \leq a$

shows $\text{Sup } A = \text{Sup } B$

<proof>

lemma *Sup-image-cong*:

$A \neq \{\} \vee B \neq \{\} \implies \forall a \in A. \exists b \in B. (f a::\text{enat}) \leq g b \implies \forall b \in B. \exists a \in A. (g b::\text{enat}) \leq f a \implies$

$\text{Sup } (f ' A) = \text{Sup } (g ' B)$

<proof>

lemma *Sup-congL*:

$A \neq \{\} \implies \forall a \in A. \exists b \in B. (a::\text{enat}) \leq b \implies \forall b \in B. b \leq \text{Sup } A \implies \text{Sup } A = \text{Sup } B$

<proof>

lemma *Sup-image-congL*:

$A \neq \{\} \implies \forall a \in A. \exists b \in B. (f a::\text{enat}) \leq g b \implies \forall b \in B. g b \leq \text{Sup } (f ' A) \implies \text{Sup } (f ' A) = \text{Sup } (g ' B)$

<proof>

lemma *Sup-congR*:

$B \neq \{\} \implies \forall a \in A. a \leq \text{Sup } B \implies \forall b \in B. \exists a \in A. (b::\text{enat}) \leq a \implies \text{Sup } A = \text{Sup } B$

<proof>

lemma *Sup-image-congR*:

$B \neq \{\} \implies \forall a \in A. f a \leq \text{Sup } (g ' B) \implies \forall b \in B. \exists a \in A. (g b::\text{enat}) \leq f a \implies \text{Sup } (f ' A) = \text{Sup } (g ' B)$

<proof>

lemma *Sup-eq-0-iff*: $\text{Sup } (A :: \text{enat set}) = 0 \iff (\forall a \in A. a = 0)$

<proof>

lemma *plus-Sup-commute*:

assumes $f1: \{f1 b1 \mid b1. \varphi1 b1\} \neq \{\}$ **and**

$f2: \{f2 b2 \mid b2. \varphi2 b2\} \neq \{\}$

shows

$\text{Sup } \{(f1 b1::\text{enat}) \mid b1. \varphi1 b1\} + \text{Sup } \{f2 b2 \mid b2. \varphi2 b2\} =$

$\text{Sup } \{f1 b1 + f2 b2 \mid b1 b2. \varphi1 b1 \wedge \varphi2 b2\}$ (**is** $?L1 + ?L2 = ?R$)

<proof>

lemma *plus-Sup-commute'*:

assumes $f1: A1 \neq \{\}$ **and** $f2: A2 \neq \{\}$

shows $\text{Sup } A1 + \text{Sup } A2 = \text{Sup } \{(a1::\text{enat}) + a2 \mid a1 a2. a1 \in A1 \wedge a2 \in$

$A2\}$
 $\langle proof \rangle$

lemma *plus-SupR*: $A \neq \{\}$ $\implies Sup A + (b::enat) = Sup \{a + b \mid a. a \in A\}$
 $\langle proof \rangle$

lemma *plus-SupL*: $A \neq \{\}$ $\implies (b::enat) + Sup A = Sup \{b + a \mid a. a \in A\}$
 $\langle proof \rangle$

lemma *sum-mono3*:
 $finite B \implies A \subseteq B \implies (\bigwedge a. a \in A \implies (f a::enat) \leq g a) \implies$
 $sum f A \leq sum g B$
 $\langle proof \rangle$

lemma *sum-Sup-commute*:
fixes $h :: 'a \Rightarrow enat$
assumes *finite J and* $\forall i \in J. \{h b \mid b. \varphi i b\} \neq \{\}$
shows $sum (\lambda i. Sup \{h b \mid b. \varphi i b\}) J =$
 $Sup \{sum (\lambda i. h (b i)) J \mid b. \forall i \in J. \varphi i (b i)\}$
 $\langle proof \rangle$

1.2 Infinite Summation

definition *isum* :: $('a \Rightarrow enat) \Rightarrow 'a \text{ set} \Rightarrow enat$ **where**
 $isum f A \equiv Sup (sum f ` \{B \mid B . B \subseteq A \wedge finite B\})$

lemma *isum-subset-mono*: $A \subseteq B \implies isum f A \leq isum f B$
 $\langle proof \rangle$

lemma *isum-eq-sum*:
 $finite A \implies isum f A = sum f A$
 $\langle proof \rangle$

lemma *isum-cong*:
assumes $A = B$ **and** $\bigwedge x. x \in B \implies g x = h x$
shows $isum g A = isum h B$
 $\langle proof \rangle$

lemma *isum-mono*:

assumes $\bigwedge x. x \in A \implies g\ x \leq h\ x$
shows $isum\ g\ A \leq isum\ h\ A$
<proof>

lemma *isum-mono'*:

assumes $A \subseteq B$
shows $isum\ g\ A \leq isum\ g\ B$
<proof>

lemma *isum-empty[simp]*: $isum\ g\ \{\} = 0$

<proof>

lemma *isum-const-zero[simp]*: $isum\ (\lambda x. 0)\ A = 0$

<proof>

lemma *isum-const-zero'*: $\forall x \in A. g\ x = 0 \implies isum\ g\ A = 0$

<proof>

lemma *isum-eq-0-iff*: $isum\ f\ A = 0 \iff (\forall a \in A. f\ a = 0)$

<proof>

lemma *isum-reindex*: $inj\text{-on}\ h\ A \implies isum\ g\ (h\ ` A) = isum\ (g \circ h)\ A$

<proof>

lemma *isum-reindex-cong*: $inj\text{-on}\ l\ B \implies A = l\ ` B \implies$

$(\bigwedge x. x \in B \implies g\ (l\ x) = h\ x) \implies isum\ g\ A = isum\ h\ B$

<proof>

lemma *isum-reindex-cong'*:

assumes $(\bigwedge x\ y. x \in A \implies y \in A \implies x \neq y \implies h\ x = h\ y \implies g\ (h\ x) = 0)$

shows $isum\ g\ (h\ ` A) = isum\ (g \circ h)\ A$

<proof>

lemma *isum-zeros-cong*:

assumes $\bigwedge i. i \in T - S \implies h\ i = 0$ **and** $\bigwedge i. i \in S - T \implies g\ i = 0$

and $\bigwedge x. x \in S \cap T \implies g\ x = h\ x$

shows $isum\ g\ S = isum\ h\ T$

<proof>

lemma *isum-zeros-congL*:

$S \subseteq T \implies \forall i \in T - S. g\ i = 0 \implies \text{isum } g\ S = \text{isum } g\ T$
<proof>

lemma *isum-zeros-congR*:

$S \subseteq T \implies \forall i \in T - S. g\ i = 0 \implies \text{isum } g\ T = \text{isum } g\ S$
<proof>

lemma *isum-singl[simp]*: $\text{isum } f\ \{a\} = f\ a$

<proof>

lemma *isum-two[simp]*: $a1 \neq a2 \implies \text{isum } f\ \{a1, a2\} = f\ a1 + f\ a2$

<proof>

lemma *isum-three[simp]*: $a1 \neq a2 \implies a1 \neq a3 \implies a2 \neq a3 \implies$

$\text{isum } f\ \{a1, a2, a3\} = f\ a1 + f\ a2 + f\ a3$

<proof>

lemma *in-le-isum*: $a \in A \implies f\ a \leq \text{isum } f\ A$

<proof>

lemma *isum-eq-singl*:

assumes $fx: f\ a = x$ **and** $f: \forall a'. a' \neq a \implies f\ a' = 0$ **and** $a: a \in A$
shows $\text{isum } f\ A = x$

<proof>

lemma *isum-le-singl*:

assumes $fx: f\ a \leq x$ **and** $f: \forall a'. a' \neq a \implies f\ a' = 0$ **and** $a: a \in A$
shows $\text{isum } f\ A \leq x$

<proof>

lemma *isum-insert[simp]*: $a \notin A \implies \text{isum } f\ (\text{insert } a\ A) = \text{isum } f\ A + f\ a$

<proof>

lemma *isum-UNION*:

assumes $dsj: \forall i \in I. \forall j \in I. i \neq j \implies A\ i \cap A\ j = \{\}$

shows $\text{isum } g\ (\bigcup (A\ ' I)) = \text{isum } (\lambda i. \text{isum } g\ (A\ i))\ I$

<proof>

lemma *isum-Un[simp]*:

assumes $A1 \cap A2 = \{\}$

shows $isum\ f\ (A1\ \cup\ A2) = isum\ f\ A1 + isum\ f\ A2$
 ⟨proof⟩

lemma *isum-Sigma*:

$isum\ (\lambda(a,b).\ f\ a\ b)\ (Sigma\ A\ Bs) = isum\ (\lambda a.\ isum\ (f\ a)\ (Bs\ a))\ A$
 ⟨proof⟩

lemma *isum-Times*: $isum\ (\lambda(a,b).\ f\ a\ b)\ (A\ \times\ B) = isum\ (\lambda a.\ isum\ (f\ a)\ B)\ A$
 ⟨proof⟩

lemma *isum-swap*: $isum\ (\lambda a.\ isum\ (f\ a)\ B)\ A = isum\ (\lambda b.\ isum\ (\lambda a.\ f\ a\ b)\ A)\ B$
 (is ?L = ?R)
 ⟨proof⟩

lemma *isum-plus*:

shows $isum\ (\lambda a.\ f1\ a + f2\ a)\ A = isum\ f1\ A + isum\ f2\ A$
 ⟨proof⟩

end

theory *Countable-Multiset*

imports

HOL-Library.Countable-Set-Type

HOL-Library.Extended-Nat

Coinductive.Coinductive-List

HOL-Library.BNF-Corec

Infinite-Sums-Enat

begin

2 Miscellaneous (Mostly About Lazy Lists)

lemma *bij-betw-singl-remap*: $\langle\text{bij-betw}\ \pi\ A\ B \implies x \in A \implies y \in B \implies$
 $\text{bij-betw}\ (\pi(\text{inv-into}\ A\ \pi\ y := \pi\ x))\ (A - \{x\})\ (B - \{y\})\rangle$
 ⟨proof⟩

lemma *ldropWhile-eq-LCons-iff*: $\langle\text{ldropWhile}\ P\ lxs = LCons\ x\ lxs' \iff (\neg P\ x \wedge$
 $(\exists xs.\ lxs = \text{lappend}\ (\text{llist-of}\ xs)\ (LCons\ x\ lxs')) \wedge (\forall x \in \text{set}\ xs.\ P\ x))\rangle$
 ⟨proof⟩

lemma *ltakeWhile-ldropWhile-decomp*:

assumes $\langle x \in \text{lset}\ lxs \rangle$

shows $\langle lxs = \text{lappend}\ (\text{ltakeWhile}\ ((\neq)\ x)\ lxs)\ (LCons\ x\ (\text{ttl}\ (\text{ldropWhile}\ ((\neq)\ x)\ lxs))) \rangle$

⟨proof⟩

lemma *lzip-lmap-same*: $\langle \text{lzip}\ (\text{lmap}\ f\ lxs)\ (\text{lmap}\ g\ lxs) = \text{lmap}\ (\lambda x.\ (f\ x,\ g\ x))\ lxs \rangle$

<proof>

lemma *llength-not-lnull*: $\langle \neg \text{lnull } lxs \implies \text{llength } lxs = e\text{Suc } (\text{llength } (\text{ltl } lxs)) \rangle$
<proof>

primrec *ltaken* :: $\langle \text{nat} \Rightarrow 'a \text{ llist} \Rightarrow 'a \text{ list} \rangle$ **where**
 $\langle \text{ltaken } 0 \text{ } lxs = [] \rangle$
 $\mid \langle \text{ltaken } (\text{Suc } i) \text{ } lxs = (\text{case } lxs \text{ of } LNil \Rightarrow [] \mid LCons \ x \ lxs \Rightarrow x \ \# \ \text{ltaken } i \text{ } lxs) \rangle$

lemma *nth-ltaken*: $\langle m < n \implies n \leq \text{llength } lxs \implies \text{nth } (\text{ltaken } n \text{ } lxs) \ m = \text{lnth } lxs \ m \rangle$
<proof>

lemma *set-ltaken*: $\langle \text{set } (\text{ltaken } n \text{ } lxs) \subseteq \text{lset } lxs \rangle$
<proof>

lemma *length-ltaken*: $\langle \text{length } (\text{ltaken } n \text{ } lxs) = (\text{if } \text{enat } n \leq \text{llength } lxs \text{ then } n \text{ else } \text{the-enat } (\text{llength } lxs)) \rangle$
<proof>

lemma *set-ltaken-conv*: $\langle n \leq \text{llength } lxs \implies \text{set } (\text{ltaken } n \text{ } lxs) = \text{lnth } lxs \ \{0..<n\} \rangle$
<proof>

lemma *ltaken-lappend*:
 $\langle \text{ltaken } n \ (\text{lappend } lxs \ lys) = (\text{case } \text{llength } lxs \text{ of } \infty \Rightarrow \text{ltaken } n \text{ } lxs \mid \text{enat } m \Rightarrow \text{ltaken } n \text{ } lxs \ @ \ \text{ltaken } (n - m) \ lys) \rangle$
<proof>

lemma *ltaken-LNil[simp]*: $\langle \text{ltaken } i \ LNil = [] \rangle$
<proof>

3 enat as a Codatatype

codatatype *en* = *eZ* | *eS* (*ep*: *en*)
where $\langle \text{ep } eZ = eZ \rangle$

coinductive *is-einf* **where**
 $\langle \text{is-einf } n \implies \text{is-einf } (eS \ n) \rangle$

inductive *is-fin* **where**
 $\langle \text{is-fin } eZ \rangle$
 $\mid \langle \text{is-fin } n \implies \text{is-fin } (eS \ n) \rangle$

lemma *not-is-fin-is-einf*: $\langle \neg \text{is-fin } n \implies \text{is-einf } n \rangle$
<proof>

lemma *is-fin-not-is-einf*: $\langle \text{is-fin } n \implies \neg \text{is-einf } n \rangle$
<proof>

primcorec *empf* **where**

⟨*empf* = *eS empf*⟩

lemma *empf-eS-iff*: ⟨*empf* = *eS x* \longleftrightarrow *x* = *empf*⟩

⟨*proof*⟩

lemma *is-empf-empf*: ⟨*is-empf empf*⟩

⟨*proof*⟩

lemma *is-empf-eq-empf*: ⟨*is-empf n* \implies *n* = *empf*⟩

⟨*proof*⟩

fun *nat-to-en* **where**

⟨*nat-to-en 0* = *eZ*⟩

| ⟨*nat-to-en (Suc n)* = *eS (nat-to-en n)*⟩

lemma *is-fin-ex-nat-to-en*: ⟨*is-fin n* \implies $\exists m. n$ = *nat-to-en m*⟩

⟨*proof*⟩

lemma *inj-nat-to-en*: ⟨*nat-to-en x* = *nat-to-en y* \implies *x* = *y*⟩

⟨*proof*⟩

lemma *nat-to-en-not-empf*: ⟨*nat-to-en n* = *empf* \implies *False*⟩

⟨*proof*⟩

definition *enat-to-en* **where**

⟨*enat-to-en n* = (*case n of enat n* \Rightarrow *nat-to-en n* | - \Rightarrow *empf*)⟩

lemma *inj-enat-to-en*: ⟨*inj enat-to-en*⟩

⟨*proof*⟩

lemma *range-enat-to-en*: ⟨*n* \in *range enat-to-en*⟩

⟨*proof*⟩

lemma *type-definition-enat*: ⟨*type-definition enat-to-en (inv enat-to-en) UNIV*⟩

⟨*proof*⟩

setup-lifting *type-definition-enat*

lift-definition *corec-enat* :: ⟨('a \Rightarrow *bool*) \Rightarrow ('a \Rightarrow *bool*) \Rightarrow ('a \Rightarrow *enat*) \Rightarrow ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow *enat*⟩

is *corec-en* ⟨*proof*⟩

lemma *eZ-transfer[transfer-rule]*: ⟨*pcr-enat eZ 0*⟩

⟨*proof*⟩

lemma *eS-transfer[transfer-rule]*: ⟨*rel-fun pcr-enat pcr-enat eS eSuc*⟩

⟨*proof*⟩

lemma *ep-nat-to-en*: $\langle ep \ (nat\text{-to-en} \ n) = nat\text{-to-en} \ (n - Suc \ 0) \rangle$
<proof>

lemma *ep-transfer*[*transfer-rule*]: $\langle rel\text{-fun} \ pcr\text{-enat} \ pcr\text{-enat} \ ep \ epred \rangle$
<proof>

lemma *corec-enat-code*[*code*]:
 $\langle corec\text{-enat} \ zer \ stop \ end \ cnt \ a =$
 $\ (if \ zer \ a \ then \ 0 \ else \ eSuc \ (if \ stop \ a \ then \ end \ a \ else \ corec\text{-enat} \ zer \ stop \ end \ cnt$
 $\ (cnt \ a)) \rangle$
<proof>

instantiation *en* :: *comm-monoid-add*
begin

definition *zero-en* **where** $\langle zero\text{-en} = eZ \rangle$

primcorec *plus-en* **where**
 $\langle plus\text{-en} \ e1 \ e2 = (case \ e1 \ of \ eZ \Rightarrow \ e2 \ | \ eS \ e1' \Rightarrow \ eS \ (plus\text{-en} \ e1' \ e2)) \rangle$

friend-of-corec *plus* :: $\langle en \Rightarrow en \Rightarrow en \rangle$ **where**
 $\langle e1 + e2 = (case \ e1 \ of \ eZ \Rightarrow (case \ e2 \ of \ eZ \Rightarrow eZ \ | \ eS \ e2' \Rightarrow eS \ e2') \ | \ eS \ e1'$
 $\Rightarrow eS \ (e1' + e2)) \rangle$
<proof>

lemma *plus-einf-left*[*simp*]: $\langle einf + e = einf \rangle$
<proof>

lemma *eZ-left-neutral*[*simp*]: $\langle eZ + e = e \rangle$
<proof>

lemma *eZ-right-neutral*[*simp*]: $\langle e + eZ = e \rangle$
<proof>

lemma *plus-eS-left*[*simp*]: $\langle eS \ e1 + e2 = eS \ (e1 + e2) \rangle$
<proof>

lemma *plus-eS-right*[*simp*]: $\langle e1 + eS \ e2 = eS \ (e1 + e2) \rangle$
<proof>

instance
<proof>

end

lemma *plus-einf-right*[*simp*]: $\langle e + einf = einf \rangle$
<proof>

lemma *nat-to-en-plus*[*simp*]: $\langle nat\text{-to-en} \ (m + n) = nat\text{-to-en} \ m + nat\text{-to-en} \ n \rangle$
<proof>

lemma *ezero-transfer*[*transfer-rule*]: $\langle pcr\text{-enat} \ 0 \ 0 \rangle$

$\langle \text{proof} \rangle$

lemma *epplus-transfer*[*transfer-rule*]: $\langle \text{rel-fun pcr-enat (rel-fun pcr-enat pcr-enat)}$
 $(+) (+)$
 $\langle \text{proof} \rangle$

lemma *case-en-transfer*[*transfer-rule*]: $\langle \text{rel-fun R (rel-fun (rel-fun pcr-enat R) (rel-fun}$
 $\text{pcr-enat R)) case-en co.case-enat} \rangle$
 $\langle \text{proof} \rangle$

corec *lsum-en where*

$\langle \text{lsum-en lxs} = (\text{case ldropWhile ((=) 0) lxs of LNil} \Rightarrow 0 \mid \text{LCons } e \text{ lxs} \Rightarrow eS (\text{ep}$
 $e + \text{lsum-en lxs})) \rangle$

lift-definition *lsum* :: $\langle \text{enat llist} \Rightarrow \text{enat} \rangle$
is *lsum-en* $\langle \text{proof} \rangle$

lemmas *lsum-code*[*code*] = *lsum-en.code*[*transferred*]

lemma *lsum-code-alt*:

$\langle \text{lsum lxs} = (\text{case ldropWhile ((=) 0) lxs of LNil} \Rightarrow 0 \mid \text{LCons } e \text{ lxs} \Rightarrow e + \text{lsum}$
 $\text{lxs}) \rangle$
 $\langle \text{proof} \rangle$

4 Counting in Lazy Lists

primcorec *count-llist-en where*

$\langle \text{count-llist-en lxs } x = (\text{if } x \in \text{lset lxs then } eS (\text{count-llist-en (ltl (ldropWhile ((}\neq$
 $x) \text{lxs})) } x) \text{ else } eZ) \rangle$

lift-definition *count-llist* :: $\langle 'a \text{ llist} \Rightarrow 'a \Rightarrow \text{enat} \rangle$
is *count-llist-en* $\langle \text{proof} \rangle$

lemmas *count-llist-code*[*code*] = *count-llist-en.code*[*transferred*]

lemmas *count-llist-sel*[*simp*] = *count-llist-en.sel*[*transferred*]

lemmas *count-llist-ctr* = *count-llist-en.ctr*[*transferred*]

lemmas *enat-coinduct-strong*[*case-names eq-enat, coinduct type*] = *en.coinduct-strong*[*transferred*]

lift-definition *enat-cong* :: $\langle (\text{enat} \Rightarrow \text{enat} \Rightarrow \text{bool}) \Rightarrow \text{enat} \Rightarrow \text{enat} \Rightarrow \text{bool} \rangle$ **is**
 $\langle \text{en.congclp} \rangle$ $\langle \text{proof} \rangle$

lemmas *enat-coinduct-upto*[*case-names eq-enat*] = *en.coinduct-upto*[*transferred*]

lemmas *enat-cong-intros* = *en.cong-intros*[*transferred*]

lifting-update *enat.lifting*

lifting-forget *enat.lifting*

lemma *in-lset-iff-count-llist*: $\langle x \in \text{lset lxs} \iff \text{count-llist lxs } x \neq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *count-llist-zero-iff*: $\langle \text{count-llist } lxs \ x = 0 \longleftrightarrow x \notin \text{lset } lxs \rangle$
<proof>

lemma *count-llist-alt*: $\langle \text{count-llist } lxs \ x = \text{llength } (\text{lfilter } ((=) \ x) \ lxs) \rangle$
<proof>

lemma *count-llist-lappend*:
 $\langle \text{count-llist } (\text{lappend } lxs \ lys) \ x = \text{count-llist } lxs \ x + (\text{if } \text{lfinite } lxs \ \text{then } \text{count-llist } lys \ x \ \text{else } 0) \rangle$
<proof>

lemma *count-llist-LNil[simp]*: $\langle \text{count-llist } LNil \ x = 0 \rangle$
<proof>

lemma *count-llist-LCons[simp]*: $\langle \text{count-llist } (LCons \ y \ lxs) \ x =$
 $(\text{if } x = y \ \text{then } \text{eSuc } (\text{count-llist } lxs \ x) \ \text{else } \text{count-llist } lxs \ x) \rangle$
<proof>

5 Lazy Lists of Natural Numbers

primcorec *lupt where*

$\langle \text{lupt } i \ j = (\text{if } \text{enat } i \geq j \ \text{then } LNil \ \text{else } LCons \ i \ (\text{lupt } (\text{Suc } i) \ j)) \rangle$

lemma *lset-luptD[OF - refl]*:
assumes $\langle k \in \text{lset } lxs \rangle \langle lxs = \text{lupt } i \ j \rangle$
shows $\langle i \leq k \rangle \langle k < j \rangle$
<proof>

lemma *lset-luptI*: $\langle i \leq k \implies \text{enat } k < j \implies k \in \text{lset } (\text{lupt } i \ j) \rangle$
<proof>

lemma *lset-lupt*: $\langle \text{lset } (\text{lupt } i \ j) = \{k. i \leq k \wedge \text{enat } k < j\} \rangle$
<proof>

lemma *llength-lupt[simp]*: $\langle \text{llength } (\text{lupt } i \ j) = j - i \rangle$
<proof>

lemma *lnth-lupt*: $\langle k < j - \text{enat } i \implies \text{lnth } (\text{lupt } i \ j) \ k = i + k \rangle$
<proof>

lemma *lmap-lupt-Suc*: $\langle \text{lmap } f \ (\text{lupt } (\text{Suc } i) \ (\text{eSuc } j)) = \text{lmap } (f \ o \ \text{Suc}) \ (\text{lupt } i \ j) \rangle$
<proof>

lemma *llist-conv-lmap-lupt*:
 $\langle lxs = \text{lmap } (\text{lnth } lxs) \ (\text{lupt } 0 \ (\text{llength } lxs)) \rangle$
<proof>

lemma *ldistinct-lupt[simp]*: $\langle \text{ldistinct } (\text{lupt } i \ j) \rangle$
<proof>

lemma *lzip-lupt*: $\langle \text{lzip } (\text{lupt } i \ j) \ (\text{lupt } i \ k) = \text{lmap } (\lambda x. (x, x)) \ (\text{lupt } i \ (\text{min } j \ k)) \rangle$
 $\langle \text{proof} \rangle$

6 Permutations of Lazy Lists

definition $\langle \text{lpermute } \pi \ \text{lhs} = \text{lmap } (\lambda i. \text{lnth } \text{lhs} \ (\pi \ i)) \ (\text{lupt } 0 \ (\text{llength } \text{lhs})) \rangle$

abbreviation $\langle \text{lbij-on } \pi \ \text{lhs} \equiv \text{bij-betw } \pi \ \{k. \text{enat } k < \text{llength } \text{lhs}\} \ \{k. \text{enat } k < \text{llength } \text{lhs}\} \rangle$

lemma *lset-lpermute*:

assumes $\langle \text{lbij-on } \pi \ \text{lhs} \rangle$

shows $\langle \text{lset } (\text{lpermute } \pi \ \text{lhs}) = \text{lset } \text{lhs} \rangle$

$\langle \text{proof} \rangle$

lemma *llength-lpermute[simp]*: $\langle \text{llength } (\text{lpermute } \pi \ \text{lhs}) = \text{llength } \text{lhs} \rangle$

$\langle \text{proof} \rangle$

lemma *lnth-lpermute[simp]*: $\langle \text{lbij-on } \pi \ \text{lhs} \implies i < \text{llength } \text{lhs} \implies \text{lnth } (\text{lpermute } \pi \ \text{lhs}) \ i = \text{lnth } \text{lhs} \ (\pi \ i) \rangle$

$\langle \text{proof} \rangle$

lemma *lfilter-permute*: $\langle \text{ldistinct } \text{lhs} \implies \text{ldistinct } \text{lhs} \implies \text{bij-betw } \pi \ (\text{lset } \text{lhs}) \ (\text{lset } \text{lhs}) \implies \forall x \in \text{lset } \text{lhs}. P \ x \longleftrightarrow Q \ (\pi \ x) \implies$

$\text{llength } (\text{lfilter } P \ \text{lhs}) = \text{llength } (\text{lfilter } Q \ \text{lhs}) \rangle$

$\langle \text{proof} \rangle$

lemma *count-llist-lpermute*:

$\langle \text{lbij-on } \pi \ \text{lhs} \implies \text{count-llist } (\text{lpermute } \pi \ \text{lhs}) \ x = \text{count-llist } \text{lhs} \ x \rangle$

$\langle \text{proof} \rangle$

lemma *lpermute-lzip*: $\langle \text{lbij-on } \pi \ \text{lhs} \implies \text{llength } \text{lhs} = \text{llength } \text{lhs} \implies \text{lpermute } \pi \ (\text{lzip } \text{lhs} \ \text{lhs}) = \text{lzip } (\text{lpermute } \pi \ \text{lhs}) \ (\text{lpermute } \pi \ \text{lhs}) \rangle$

$\langle \text{proof} \rangle$

lemma *exist-nth-occurrence-in-llist*:

$\langle \text{count-llist } \text{lhs} \ x > n \implies \exists i < \text{llength } \text{lhs}. \text{lnth } \text{lhs} \ i = x \wedge \text{count-list } (\text{ltaken } i \ \text{lhs}) \ x = n \rangle$

$\langle \text{proof} \rangle$

function *lfind-index where*

$\langle \text{lfind-index } n \ x \ \text{lhs} = (\text{if } \text{count-llist } \text{lhs} \ x \leq \text{enat } n \ \text{then } i \ \text{else}$

$\text{if } \text{lhd } \text{lhs} = x \ \text{then } \text{case } n \ \text{of } 0 \Rightarrow i \mid \text{Suc } m \Rightarrow \text{lfind-index } m \ x \ (\text{Suc } i) \ (\text{tl } \text{lhs})$

$\text{else } \text{lfind-index } n \ x \ (\text{Suc } i) \ (\text{tl } \text{lhs})) \rangle$

$\langle \text{proof} \rangle$

termination

$\langle \text{proof} \rangle$

declare *lfind-index.simps[simp del]*

lemma *lfind-index-ge[simp]*: $\langle \text{lfind-index } n \ x \ j \ \text{lbs} \geq j \rangle$
 $\langle \text{proof} \rangle$

lemma *lfind-index-less[simp]*: $\langle \text{count-llist } \text{lbs} \ x \ > \ \text{enat } n \implies \text{lfind-index } n \ x \ j \ \text{lbs} < \text{llength } \text{lbs} + j \rangle$
 $\langle \text{proof} \rangle$

lemma *lfind-index-lnth*:
 $\langle \text{count-llist } \text{lbs} \ x \ > \ \text{enat } n \implies \text{lnth } \text{lbs} \ (\text{lfind-index } n \ x \ j \ \text{lbs} - j) = x \rangle$
 $\langle \text{proof} \rangle$

lemma *ltaken-LCons*:
 $\langle \text{ltaken } i \ (\text{LCons } x \ \text{lbs}) = (\text{case } i \ \text{of } 0 \Rightarrow [] \mid \text{Suc } j \Rightarrow x \ \# \ \text{ltaken } j \ \text{lbs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *lfind-index-count-list*:
 $\langle \text{count-llist } \text{lbs} \ x \ > \ n \implies \text{count-list } (\text{ltaken } (\text{lfind-index } n \ x \ j \ \text{lbs} - j) \ \text{lbs}) \ x = n \rangle$
 $\langle \text{proof} \rangle$

lemma *lnth-equalityI*: $\langle \text{llength } \text{lbs} = \text{llength } \text{lys} \implies (\bigwedge i. \ \text{enat } i < \text{llength } \text{lbs} \implies \text{lnth } \text{lbs} \ i = \text{lnth } \text{lys} \ i) \implies \text{lbs} = \text{lys} \rangle$
 $\langle \text{proof} \rangle$

lemma *lfind-index-inject*:
 $\langle \text{count-llist } \text{lbs} \ x \ > \ \text{enat } n \implies \text{count-llist } \text{lbs} \ y \ > \ \text{enat } m \implies \text{lfind-index } n \ x \ j \ \text{lbs} = \text{lfind-index } m \ y \ j \ \text{lbs} \implies n = m \wedge x = y \rangle$
 $\langle \text{proof} \rangle$

lemma *count-list-ltaken-less*:
 $\langle i < \text{llength } \text{lbs} \implies \text{count-list } (\text{ltaken } i \ \text{lbs}) \ (\text{lnth } \text{lbs} \ i) < \text{count-llist } \text{lbs} \ (\text{lnth } \text{lbs} \ i) \rangle$
 $\langle \text{proof} \rangle$

lemma *count-list-inject*:
 $\langle \text{enat } i < \text{llength } \text{lbs} \implies \text{enat } j < \text{llength } \text{lbs} \implies \text{count-list } (\text{ltaken } i \ \text{lbs}) \ (\text{lnth } \text{lbs} \ j) = \text{count-list } (\text{ltaken } j \ \text{lbs}) \ (\text{lnth } \text{lbs} \ j) \implies \text{lnth } \text{lbs} \ i = \text{lnth } \text{lbs} \ j \implies i = j \rangle$
 $\langle \text{proof} \rangle$

7 Same-Count Equivalence Relation

definition $\langle \text{eq-cmset } \text{lbs} \ \text{lys} = (\forall x. \ \text{count-llist } \text{lbs} \ x = \text{count-llist } \text{lys} \ x) \rangle$

lemma *lset-eq-cmset*: $\langle \text{eq-cmset } \text{lbs} \ \text{lys} \implies \text{lset } \text{lbs} = \text{lset } \text{lys} \rangle$
 $\langle \text{proof} \rangle$

lemma *eq-cmset-llength*: $\langle \text{eq-cmset } \text{lbs} \ \text{lys} \implies \text{llength } \text{lbs} = \text{llength } \text{lys} \rangle$
 $\langle \text{proof} \rangle$

lemma *eq-cmset-alt*: $\langle \text{eq-cmset } lxs' \ lxs \longleftrightarrow (\exists \pi. \text{lbij-on } \pi \ lxs \wedge \text{lpermute } \pi \ lxs = lxs') \rangle$
 $\langle \text{proof} \rangle$

lemma *eq-cmset-lzip-left*:
assumes $\langle \text{eq-cmset } lxs' \ lxs \rangle \langle \text{llength } lys = \text{llength } lxs \rangle$
shows $\langle \exists lys'. \text{eq-cmset } (\text{lzip } lxs' \ lys') (\text{lzip } lxs \ lys) \wedge \text{llength } lys' = \text{llength } lxs' \rangle$
 $\langle \text{proof} \rangle$

lemma *eq-cmset-lmap*:
assumes $\langle \text{eq-cmset } lxs \ lys \rangle$
shows $\langle \text{eq-cmset } (\text{lmap } f \ lxs) (\text{lmap } f \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *llist-all2-reorder-left-invariance*:
assumes *rel*: $\langle \text{llist-all2 } R \ lxs \ lys \rangle$ **and** *ms-x*: $\langle \text{eq-cmset } lxs' \ lxs \rangle$
shows $\langle \exists lys'. \text{llist-all2 } R \ lxs' \ lys' \wedge \text{eq-cmset } lys' \ lys \rangle$
 $\langle \text{proof} \rangle$

8 Countable Multisets as a Quotient

quotient-type *'a cmset* = $\langle 'a \ \text{llist} \rangle / \text{eq-cmset}$
 $\langle \text{proof} \rangle$

lift-bnf (*cmset*: *'a*) *cmset*
for *map*: *cmimage* *rel*: *cmrel*
 $\langle \text{proof} \rangle$

9 Lazy List Interleaving

context notes $[[\text{function-internals}]]$
begin
partial-function (*llist*) *lflat* **where**
 $\langle \text{lflat } lxs = (\text{case } lxs \ \text{of } LNil \Rightarrow LNil \mid LCons \ x \ lxs \Rightarrow \text{lappend } (\text{llist-of } x) (\text{lflat } lxs)) \rangle$
end

lemma *ltaken-ldropn-decomp*: $\langle lxs = \text{lappend } (\text{llist-of } (\text{ltaken } n \ lxs)) (\text{ldropn } n \ lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *count-llist-llist-of[simp]*: $\langle \text{count-llist } (\text{llist-of } xs) \ x = \text{count-list } xs \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *lmap-lfilter-swap*: $\langle \forall x \in \text{lset } lxs. P \ x \longleftrightarrow Q \ (f \ x) \implies \text{lmap } f \ (\text{lfilter } P \ lxs) = \text{lfilter } Q \ (\text{lmap } f \ lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *lsum-lmap-zero*: $\langle \text{lsum} (\text{lmap} (\lambda z. 0) \text{ lxs}) = 0 \rangle$
<proof>

lemma *lsum-LNil[simp]*: $\langle \text{lsum} \text{ LNil} = 0 \rangle$
<proof>

lemma *lsum-LCons[simp]*: $\langle \text{lsum} (\text{LCons } x \text{ lxs}) = x + \text{lsum } \text{lxs} \rangle$
<proof>

lemma *count-llist-lmap-const*:
 $\langle \text{count-llist} (\text{lmap} (\lambda z. a) \text{ lxs}) \ x = (\text{if } x = a \text{ then } \text{llength } \text{lxs} \text{ else } 0) \rangle$
<proof>

lemma *lsum-lmap-all-but-one-0*: $\langle x \in \text{lset } \text{lxs} \implies \text{ldistinct } \text{lxs} \implies \text{lsum} (\text{lmap} (\lambda z. \text{if } z = x \text{ then } y \text{ else } 0) \text{ lxs}) = y \rangle$
<proof>

lemma *ltaken-Suc*:
 $\langle \text{ltaken} (\text{Suc } i) \text{ lxs} = (\text{if } i < \text{llength } \text{lxs} \text{ then } \text{ltaken } i \text{ lxs} \ @ \ [\text{lnth } \text{lxs } i] \text{ else } \text{ltaken } i \text{ lxs}) \rangle$
<proof>

lemma *ltaken-all-same*: $\langle \text{enat } n \geq \text{llength } \text{lxs} \implies \text{enat } m \geq \text{llength } \text{lxs} \implies \text{ltaken } n \text{ lxs} = \text{ltaken } m \text{ lxs} \rangle$
<proof>

lemma *lsum-mono*: $\langle \text{lprefix } \text{lxs } \text{lys} \implies \text{lsum } \text{lxs} \leq \text{lsum } \text{lys} \rangle$
<proof>

lemma *Sup-enat-remove0*: $\langle \exists x \in X. x > 0 \implies \text{Sup} (X :: \text{enat set}) = \text{Sup} (X - \{0\}) \rangle$
<proof>

lemma *lSup-eq-lappend*: $\langle \text{Complete-Partial-Order.chain } \text{lprefix } Y \implies \exists \text{ lys} \in Y. \text{lprefix} (\text{llist-of } \text{xs}) \ \text{lys} \implies \text{lSup } Y = \text{lappend} (\text{llist-of } \text{xs}) (\text{lSup} (\text{ldropn} (\text{length } \text{xs}) ' Y)) \rangle$
<proof>

lemma *lsum-0D*: $\langle x \in \text{lset } \text{lxs} \implies \text{lsum } \text{lxs} = 0 \implies x = 0 \rangle$
<proof>

lemma *lsum-0I*: $\langle \forall x \in \text{lset } \text{lxs}. x = 0 \implies \text{lsum } \text{lxs} = 0 \rangle$
<proof>

lemma *lsum-0-iff*: $\langle \text{lsum } \text{lxs} = 0 \iff (\forall x \in \text{lset } \text{lxs}. x = 0) \rangle$
<proof>

lemma *lsum-lappend-lfinite*: $\langle \text{lfinite } \text{lxs} \implies \text{lsum} (\text{lappend } \text{lxs } \text{lys}) = \text{lsum } \text{lxs} +$

$lsum\ lys$
 $\langle proof \rangle$

lemma $lsum-lappend$: $\langle lsum (lappend\ lxs\ lys) = (if\ lfinite\ lxs\ then\ lsum\ lxs + lsum\ lys\ else\ lsum\ lxs) \rangle$
 $\langle proof \rangle$

lemma $lsum-llist-of[simp]$: $\langle lsum (llist-of\ xs) = sum-list\ xs \rangle$
 $\langle proof \rangle$

lemma $lsum-cont$:
 $\langle Complete-Partial-Order.chain\ lprefix\ Y \implies Y \neq \{\} \implies lsum (lSup\ Y) = \sqcup (lsum\ `Y) \rangle$
 $\langle proof \rangle$

lemma $mcont2mcont-lsum[THEN\ lfp.mcont2mcont,\ simp,\ cont-intro]$:
shows $mcont-lsum$: $\langle mcont\ lSup (lprefix)\ Sup (\leq)\ lsum \rangle$
 $\langle proof \rangle$

lemma $lsum-lfilter-nonzero$: $\langle lsum (lfilter ((\neq)\ 0)\ lxs) = lsum\ lxs \rangle$
 $\langle proof \rangle$

abbreviation $\langle LSUM\ lxs\ f \equiv lsum (lmap\ f\ lxs) \rangle$

abbreviation $\langle lenats \equiv lupt\ 0\ \infty \rangle$

lemma $LSUM-lfilter$: $\langle LSUM (lfilter (\lambda x. f\ x \neq 0)\ lxs) f = LSUM\ lxs\ f \rangle$
 $\langle proof \rangle$

lemma $LSUM-count-lfilter$:
 $\langle lsum (lmap (\lambda lxs. count-llist\ lxs\ x) (lfilter (\lambda x. \neg\ lnull\ x)\ lxs)) = lsum (lmap (\lambda lxs. count-llist\ lxs\ x)\ lxs) \rangle$
 $\langle proof \rangle$

lemma $lflat-LNil[simp]$: $\langle lflat\ LNil = LNil \rangle$
 $\langle proof \rangle$

lemma $lflat-LCons[simp]$: $\langle lflat (LCons\ xs\ lxs) = lappend (llist-of\ xs) (lflat\ lxs) \rangle$
 $\langle proof \rangle$

lemma $count-llist-mono$:
assumes $\langle lprefix\ lxs\ lys \rangle$
shows $\langle count-llist\ lxs\ x \leq count-llist\ lys\ x \rangle$
 $\langle proof \rangle$

lemma $ldropWhile-not-lnull-alt$:
 $\langle ldropWhile ((\neq)\ x) `Y \cap \{lxs. \neg\ lnull\ lxs\} = ldropWhile ((\neq)\ x) ` \{lxs \in Y. x \in lset\ lxs\} \rangle$

⟨proof⟩

lemma *count-llist-cont*: ⟨Complete-Partial-Order.chain lprefix Y \implies Y \neq {} \implies
count-llist (lSup Y) x = (\sqcup lxs \in Y. count-llist lxs x)⟩
⟨proof⟩

lemma *mcont2mcont-count-llist*[THEN lfp.mcont2mcont, simp, cont-intro]:
shows *mcont-count-llist*: ⟨mcont lSup (lprefix) Sup (\leq) (λ lxs. count-llist lxs x)⟩
⟨proof⟩

lemma *mono2mono-lflat*[THEN llist.mono2mono, simp, cont-intro]:
shows *mono-lflat*: ⟨monotone lprefix lprefix lflat⟩
⟨proof⟩

lemma *mcont2mcont-lflat*[THEN llist.mcont2mcont, simp, cont-intro]:
shows *mcont-lflat*: ⟨mcont lSup lprefix lSup lprefix lflat⟩
⟨proof⟩

lemma *lflat-LNil-iff*: ⟨lflat lxs = LNil \longleftrightarrow (\forall xs \in lset lxs. xs = [])⟩
⟨proof⟩

lemma *count-llist-lflat*: ⟨count-llist (lflat lxs) x = lsum (lmap (λ xs. count-list xs
x) lxs)⟩
⟨proof⟩

lemma *lset-lflat*: ⟨lset (lflat lxs) = (\bigcup xs \in lset lxs. set xs)⟩
⟨proof⟩

definition *lvertical lxs i* = (if enat i < llength lxs then ltaken (Suc i) (lnth lxs
i) else [])

definition *lhorizontal lxs j* = List.map-filter (λ lxs. if enat j < llength lxs then
Some (lnth lxs j) else None) (ltaken j lxs)

definition *lmerge where*

⟨lmerge lxs = lflat (lmap (λ i. lvertical lxs i @ lhorizontal lxs i) (lupt 0 ∞))⟩

lemma *set-ltaken-conv'*: ⟨set (ltaken n lxs) = (case llength lxs of enat m \Rightarrow lnth
lxs ' {0.. \min n m} | - \Rightarrow lnth lxs ' {0.. ∞ })⟩
⟨proof⟩

lemma *lset-lmerge*: ⟨lset (lmerge lxs) = (\bigcup lxs \in lset lxs. lset lxs)⟩
⟨proof⟩

lemma *lsum-lmap-add*: ⟨lsum (lmap (λ x. f x + g x) lxs) = lsum (lmap f lxs) +
lsum (lmap g lxs)⟩
⟨proof⟩

lemma *count-list-alt*: ⟨count-list xs x = card {j. j < length xs \wedge xs ! j = x}⟩
⟨proof⟩

lemma *count-list-lvertical*:

$\langle \text{count-list } (\text{lvertical } \text{lxss } i) \ x = (\text{if } i < \text{llength } \text{lxss} \text{ then card } \{j. j \leq i \wedge j < \text{llength } (\text{lnth } \text{lxss } i) \wedge \text{lnth } (\text{lnth } \text{lxss } i) \ j = x\} \text{ else } 0) \rangle$
 $\langle \text{proof} \rangle$

lemma *llength-less-lfinite[simp]*: $\langle \text{llength } \text{lxss} < \text{enat } j \implies \text{lfinite } \text{lxss} \rangle$
 $\langle \text{proof} \rangle$

lemma *ltaken-all*: $\langle \text{llength } \text{lxss} < \text{enat } j \implies \text{ltaken } j \ \text{lxss} = \text{list-of } \text{lxss} \rangle$
 $\langle \text{proof} \rangle$

lemma *count-list-map-filter*:

$\langle \text{count-list } (\text{List.map-filter } f \ xs) \ x = \text{card } \{i. i < \text{length } xs \wedge f \ (xs \ ! \ i) = \text{Some } x\} \rangle$
 $\langle \text{proof} \rangle$

lemma *count-list-lhorizontal*:

$\langle \text{count-list } (\text{lhorizontal } \text{lxss } j) \ x = \text{card } \{i. i < j \wedge i < \text{llength } \text{lxss} \wedge j < \text{llength } (\text{lnth } \text{lxss } i) \wedge \text{lnth } (\text{lnth } \text{lxss } i) \ j = x\} \rangle$
 $\langle \text{proof} \rangle$

lemma *LSUM-extend*: **assumes** $\langle \text{lprefix } \text{xs } \text{lys} \rangle$

$\langle \forall i < \text{llength } \text{xs}. f \ (\text{lnth } \text{xs } i) = g \ (\text{lnth } \text{lys } i) \rangle$

$\langle \forall i < \text{llength } \text{lys}. i \geq \text{llength } \text{xs} \longrightarrow g \ (\text{lnth } \text{lys } i) = 0 \rangle$

shows $\langle \text{LSUM } \text{xs } f = \text{LSUM } \text{lys } g \rangle$

$\langle \text{proof} \rangle$

lemma *lupt-0[simp]*: $\langle \text{lupt } i \ 0 = \text{LNil} \rangle$
 $\langle \text{proof} \rangle$

lemma *lprefix-lupt*: $\langle j \leq k \implies \text{lprefix } (\text{lupt } i \ j) \ (\text{lupt } i \ k) \rangle$
 $\langle \text{proof} \rangle$

lemma *if-card-else-0*: $\langle (\text{if } P \ z \text{ then card } \{x. Q \ x \ z\} \text{ else } 0) = \text{card } \{x. P \ z \wedge Q \ x \ z\} \rangle$
 $\langle \text{proof} \rangle$

lemma *LSUM-llist-of[simp]*: $\langle \text{LSUM } (\text{llist-of } \text{xs}) \ f = \text{sum-list } (\text{map } f \ \text{xs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *card-sum-list*: $\langle (\bigwedge i. P \ i \implies i \leq n) \implies$

$\text{card } \{i :: \text{nat}. P \ i\} = \text{sum-list } (\text{map } (\lambda i. \text{if } P \ i \text{ then } 1 \text{ else } 0) \ [0 \ ..< \ \text{Suc } n]) \rangle$

$\langle \text{proof} \rangle$

lemma *llist-of-lupt*: $\langle \text{llist-of } [i \ ..< \ j] = \text{lupt } i \ j \rangle$
 $\langle \text{proof} \rangle$

lemma *enat-sum-list*: $\langle \text{enat } (\text{sum-list } \text{xs}) = \text{sum-list } (\text{map } \text{enat } \text{xs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *card-LSUM*:

assumes $\langle \bigwedge i. P\ i \implies i \leq n \rangle$

shows $\langle \text{enat} (\text{card} \{i :: \text{nat}. P\ i\}) = \text{LSUM } \text{lenats} (\lambda i. \text{if } P\ i \text{ then } 1 \text{ else } 0) \rangle$

<proof>

lemma *LSUM-extend-lenats*:

$\langle \text{LSUM } lxs\ f = \text{LSUM } \text{lenats} (\lambda i. \text{if } \text{enat } i < \text{llength } lxs \text{ then } f (\text{lnth } lxs\ i) \text{ else } 0) \rangle$

<proof>

lemma *LCons-eq-lmap-conv*:

$\langle \text{LCons } y\ lxs = \text{lmap } f\ lxs = (\exists x\ lxs'. lxs = \text{LCons } x\ lxs' \wedge y = f\ x \wedge lxs = \text{lmap } f\ lxs') \rangle$

<proof>

lemma *lmap-eq-lappend*: $\langle \text{lfinite } lxs \implies \text{lmap } f\ lxs = \text{lappend } lxs\ lxs \longleftrightarrow$

$(\exists lxs\ lxs'. lxs = \text{lappend } lxs\ lxs' \wedge lxs = \text{lmap } f\ lxs \wedge lxs = \text{lmap } f\ lxs') \rangle$

<proof>

lemma *lmap-eq-lappend*:

$\langle \text{lmap } f\ lxs = \text{lmap } f\ lxs \longleftrightarrow (\exists lxs'. lxs = \text{lmap } f\ lxs' \wedge \text{lmap } f\ lxs = lxs') \rangle$

<proof>

lemma *neq-LCons-conv*: $\langle (\forall x\ lxs. lxs \neq \text{LCons } x\ lxs) \longleftrightarrow lxs = \text{LNil} \rangle$

<proof>

lemma *epred-iadd*: $\langle \text{epred } (a + b) = (\text{if } a = 0 \text{ then } \text{epred } b \text{ else } \text{epred } a + b) \rangle$

<proof>

lemma *LSUM-isum*: $\langle \text{ldistinct } lxs \implies \text{LSUM } lxs\ f = \text{isum } f (\text{lset } lxs) \rangle$

<proof>

lemma *set-partition-subset*: $\langle A \subseteq B \implies B = A \cup (B - A) \rangle$

<proof>

lemma *count-lmap-isum*: $\langle \text{count-lmap } lxs\ x = \text{isum } (\lambda i. \text{if } \text{enat } i < \text{llength } lxs \wedge \text{lnth } lxs\ i = x \text{ then } 1 \text{ else } 0) \text{ UNIV} \rangle$

<proof>

lemma *count-lmap-lmerge*: $\langle \text{count-lmap } (\text{lmerge } lxs) x = \text{lsum } (\text{lmap } (\lambda lxs. \text{count-lmap } lxs\ x) lxs) \rangle$

<proof>

10 Countable Multisets as a Subtype

lift-definition *cmcount* :: $\langle 'a\ \text{cmset} \Rightarrow 'a \Rightarrow \text{enat} \rangle$ **is** *count-lmap*

<proof>

lift-definition *cmempty* :: $\langle 'a \text{ cmset} \rangle$ **is** *LNil* $\langle \text{proof} \rangle$

lemma *cmcount-cmempty[simp]*: $\langle \text{cmcount cmempty } x = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *countable-nonzero-cmcount*:
fixes *M* :: $\langle 'a \text{ cmset} \rangle$
shows $\langle \text{countable } \{x. \text{cmcount } M \ x \neq 0\} \rangle$
 $\langle \text{proof} \rangle$

lemma *countable-exists-llist*:
assumes $\langle \text{countable } X \rangle$
shows $\langle \exists \text{ lxs. lset lxs} = X \wedge \text{ldistinct lxs} \rangle$
 $\langle \text{proof} \rangle$

lemma *in-range-cmcount*:
assumes $\langle \text{countable } \{x :: 'a. f \ x \neq 0\} \rangle$
shows $\langle f \in \text{range cmcount} \rangle$
 $\langle \text{proof} \rangle$

lemma *inj-cmcount*: $\langle \text{inj cmcount} \rangle$
 $\langle \text{proof} \rangle$

lemma *type-definition-cmset*: $\langle \text{type-definition cmcount (inv cmcount) } \{f. \text{countable } \{x. f \ x \neq 0\}\} \rangle$
 $\langle \text{proof} \rangle$

corec (*friend*) *linterleave* **where**
 $\langle \text{linterleave lxs lys} = (\text{case (lxs, lys) of}$
 $(\text{LCons } x \ \text{lxs}', \text{LCons } y \ \text{lys}') \Rightarrow \text{LCons } x \ (\text{LCons } y \ (\text{linterleave lxs}' \ \text{lys}'))$
 $| (\text{LCons } x \ \text{lxs}', \text{LNil}) \Rightarrow \text{LCons } x \ \text{lxs}'$
 $| (\text{LNil}, \text{LCons } y \ \text{lys}') \Rightarrow \text{LCons } y \ \text{lys}'$
 $| (\text{LNil}, \text{LNil}) \Rightarrow \text{LNil}) \rangle$

simps-of-case *linterleave-simps[simp]*: *linterleave.code[unfolded prod.case]*

lemma *linterleave-LNil[simp]*:
 $\langle \text{linterleave LNil lys} = \text{lys} \rangle$
 $\langle \text{linterleave lys LNil} = \text{lys} \rangle$
 $\langle \text{proof} \rangle$

lemma *linterleave-LCons1[simp]*:
 $\langle \text{linterleave (LCons } x \ \text{lxs}) \ \text{lys} = \text{LCons } x \ (\text{linterleave lys lxs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *lset-linterleave1*:
 $\langle x \in \text{lset (linterleave lxs lys)} \implies$
 $x \in \text{lset lxs} \cup \text{lset lys} \rangle$
 $\langle \text{proof} \rangle$

lemma *lset-linterleave2*:

$\langle x \in \text{lset } lxs \implies x \in \text{lset } (\text{linterleave } lxs \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *lset-linterleave3*:

$\langle x \in \text{lset } lys \implies$
 $x \in \text{lset } (\text{linterleave } lxs \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *lset-linterleave[simp]*:

$\langle \text{lset } (\text{linterleave } lxs \ lys) = \text{lset } lxs \cup \text{lset } lys \rangle$
 $\langle \text{proof} \rangle$

lemma *ldistinct-linterleave*: $\langle \text{ldistinct } lxs \implies \text{ldistinct } lys \implies \text{lset } lxs \cap \text{lset } lys = \{\} \implies \text{ldistinct } (\text{linterleave } lxs \ lys) \rangle$
 $\langle \text{proof} \rangle$

coinductive *linfinite where*

$\langle \text{linfinite } lxs \implies \text{linfinite } (LCons \ x \ lxs) \rangle$

inductive *linfinite-cong for R where*

$\langle R \ lxs \implies \text{linfinite-cong } R \ lxs \rangle$
 $| \langle \text{linfinite } lxs \implies \text{linfinite-cong } R \ lxs \rangle$
 $| \langle \text{linfinite-cong } R \ lxs \implies \text{linfinite-cong } R \ (LCons \ x \ lxs) \rangle$

lemma *linfinite-coinduct-upto[consumes 1, case-names linfinite]*:

assumes $\langle X \ lxs \rangle \langle \bigwedge lys. X \ lys \implies \exists lxs \ x. lys = LCons \ x \ lxs \wedge \text{linfinite-cong } X \ lxs \rangle$

shows $\langle \text{linfinite } lxs \rangle$
 $\langle \text{proof} \rangle$

inductive-cases *linfinite-LNilE[elim!]*: $\langle \text{linfinite } LNil \rangle$

inductive-cases *linfinite-LConsE[elim!]*: $\langle \text{linfinite } (LCons \ x \ lxs) \rangle$

lemma *linfinite-linterleaveL*: $\langle \text{linfinite } lxs \implies \text{linfinite } (\text{linterleave } lxs \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-linterleaveR*: $\langle \text{linfinite } lys \implies \text{linfinite } (\text{linterleave } lxs \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *lfinite-imp-not-linfinite*: $\langle \text{lfinite } lxs \implies \neg \text{linfinite } lxs \rangle$
 $\langle \text{proof} \rangle$

lemma *not-lfinite-imp-linfinite*: $\langle \neg \text{lfinite } lxs \implies \text{linfinite } lxs \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-eq-not-lfinite*: $\langle \text{linfinite } lxs \longleftrightarrow \neg \text{lfinite } lxs \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-eq-llength*: $\langle \text{linfinite } lxs \longleftrightarrow \text{llength } lxs = \infty \rangle$
 $\langle \text{proof} \rangle$

lemma *llength-linterleave[simp]*: $\langle \text{llength } (\text{linterleave } lxs \ lys) = \text{llength } lxs + \text{llength } lys \rangle$
 $\langle \text{proof} \rangle$

lemma *lnth-linterleave-swap*:
 $\langle \text{lnth } (\text{linterleave } lxs \ lys) \ i \notin \text{lset } lys \implies i < \text{llength } (\text{linterleave } lxs \ lys) \implies$
 $\exists j < \text{min } (\text{Suc } i) \ (\text{llength } lxs). \ \text{lnth } (\text{linterleave } lxs \ lys) \ i = \text{lnth } lxs \ j \rangle$
 $\langle \text{proof} \rangle$

lemma *ldropWhile-eq-ldropn*:
 $\langle \exists x \in \text{lset } lxs. \ \neg P \ x \implies \exists n. \ \text{enat } n = \text{llength } (\text{ltakeWhile } P \ lxs) \wedge \text{ldropWhile } P \ lxs = \text{ldropn } n \ lxs \rangle$
 $\langle \text{proof} \rangle$

lemma *ltl-linterleave[simp]*: $\langle \neg \text{lnull } lxs \implies \text{ltl } (\text{linterleave } lxs \ lys) = \text{linterleave } lys \ (\text{ltl } lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-ltl[simp]*: $\langle \text{linfinite } lxs \implies \text{linfinite } (\text{ltl } lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-ldropn[simp]*: $\langle \text{linfinite } lxs \implies \text{linfinite } (\text{ldropn } n \ lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *linfinite-not-lnull*: $\langle \text{linfinite } lxs \implies \neg \text{lnull } lxs \rangle$
 $\langle \text{proof} \rangle$

lemma *ldropn-linterleave*: $\langle \text{linfinite } lxs \implies \text{linfinite } lys \implies \text{ldropn } n \ (\text{linterleave } lxs \ lys) =$
 $\text{if } \exists m. \ n = 2 * m \ \text{then } \text{linterleave } (\text{ldropn } (n \ \text{div } 2) \ lxs) \ (\text{ldropn } (n \ \text{div } 2) \ lys)$
 else
 $\text{linterleave } (\text{ldropn } (n \ \text{div } 2) \ lys) \ (\text{ldropn } (\text{Suc } (n \ \text{div } 2)) \ lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma *llength-ltakeWhile-linterleave-ge*:
 $\langle \text{enat } (2 * m) \leq \text{llength } (\text{ltakeWhile } ((\neq) \ x) \ (\text{linterleave } lxs \ lys)) \implies$
 $x \notin \text{set } (\text{ltaken } m \ lxs) \wedge x \notin \text{set } (\text{ltaken } m \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *llength-ltakeWhile-linterleave-eq*:
 $\langle \text{enat } (2 * m) = \text{llength } (\text{ltakeWhile } ((\neq) \ x) \ (\text{linterleave } lxs \ lys)) \implies$
 $x \notin \text{set } (\text{ltaken } m \ lxs) \wedge x \notin \text{set } (\text{ltaken } m \ lys) \rangle$
 $\langle \text{proof} \rangle$

lemma *llength-ltakeWhile-linterleave-ge-Suc*:
 $\langle \text{linfinite } lxs \implies \text{linfinite } lys \implies$

$enat (Suc (2 * m)) \leq llength (ltakeWhile ((\neq) x) (linterleave lxs lys)) \implies$
 $x \notin set (ltaken (Suc m) lxs) \wedge x \notin set (ltaken m lys)$
 ⟨proof⟩

lemma *llength-ltakeWhile-linterleave-eq-Suc*:

$\langle lfinite lxs \implies lfinite lys \implies$
 $enat (Suc (2 * m)) = llength (ltakeWhile ((\neq) x) (linterleave lxs lys)) \implies$
 $x \notin set (ltaken (Suc m) lxs) \wedge x \notin set (ltaken m lys) \rangle$
 ⟨proof⟩

lemma *count-llist-linterleave*:

$\langle count-llist (linterleave lxs lys) x = count-llist lxs x + count-llist lys x \rangle$
 ⟨proof⟩

lemma *lfinite-linterleave[simp]*: $\langle lfinite (linterleave lxs lys) \longleftrightarrow lfinite lxs \wedge lfinite lys \rangle$

⟨proof⟩

lift-definition *cmadd* :: $\langle 'a\ cmset \Rightarrow 'a\ cmset \Rightarrow 'a\ cmset \rangle$ **is** *linterleave*

⟨proof⟩

lemma *cmcount-cmadd[simp]*: $\langle cmcount (cmadd M N) x = cmcount M x + cmcount N x \rangle$

⟨proof⟩

lemma *count-llist-lmap*: $\langle count-llist (lmap f lxs) b = isum (count-llist lxs) \{a. f a = b\} \rangle$

⟨proof⟩

lemma *cmcount-cmimage[simp]*: $\langle cmcount (cmimage f M) b = isum (cmcount M) \{a. f a = b\} \rangle$

⟨proof⟩

lift-definition *cmfinite* :: $\langle 'a\ cmset \Rightarrow bool \rangle$ **is** *lfinite*

⟨proof⟩

lemma *Sup-enat-eq-inf*: $\langle Sup (A :: enat\ set) = \infty \longleftrightarrow \infty \in A \vee infinite\ A \rangle$

⟨proof⟩

lemma *sum-eq-inf*: $\langle finite\ A \implies sum (f :: - \Rightarrow enat) A = \infty \longleftrightarrow (\exists a \in A. f a = \infty) \rangle$

⟨proof⟩

lemma *inf-eq-sum*: $\langle finite\ A \implies \infty = sum (f :: - \Rightarrow enat) A \longleftrightarrow (\exists a \in A. f a = \infty) \rangle$

⟨proof⟩

primcorec mk-chain where

$\langle \text{mk-chain } A \ C = (\text{let } a = (\text{SOME } a. a \in A) \text{ in } LCons \ C \ (\text{mk-chain } (A - \{a\}) \ (\text{insert } a \ C))) \rangle$

lemma in-lset-mk-chainD: $\langle X \in \text{lset } (\text{mk-chain } A \ C) \implies \text{infinite } A \implies \exists B \subseteq A. \text{finite } B \wedge X = C \cup B \rangle$
 $\langle \text{proof} \rangle$

lemma linfinite-mk-chain[simp]: $\langle \text{linfinite } (\text{mk-chain } A \ C) \rangle$
 $\langle \text{proof} \rangle$

coinductive chain for R where

$\langle \text{chain } R \ LNil \rangle$
 $| \langle \text{chain } R \ (LCons \ x \ LNil) \rangle$
 $| \langle R \ x \ y \implies \text{chain } R \ (LCons \ y \ lxs) \implies \text{chain } R \ (LCons \ x \ (LCons \ y \ lxs)) \rangle$

inductive-cases chain-LConsE: $\langle \text{chain } R \ (LCons \ x \ lxs) \rangle$

lemma chain-mk-chain: $\langle \text{infinite } A \implies A \cap C = \{\} \implies \text{chain } (\subset) \ (\text{mk-chain } A \ C) \rangle$
 $\langle \text{proof} \rangle$

lemma chainD:

$\langle \text{chain } R \ lxs \implies \text{enat } (\text{Suc } i) < \text{llength } lxs \implies R \ (\text{lnth } lxs \ i) \ (\text{lnth } lxs \ (\text{Suc } i)) \rangle$
 $\langle \text{proof} \rangle$

lemma chain-transD:

assumes $\langle \text{transp } R \rangle$
shows $\langle \text{chain } R \ lxs \implies i < j \implies \text{enat } j < \text{llength } lxs \implies R \ (\text{lnth } lxs \ i) \ (\text{lnth } lxs \ j) \rangle$
 $\langle \text{proof} \rangle$

lemma chain-cpo-chain: $\langle \text{transp } R \implies \text{chain } R \ lxs \implies \text{Complete-Partial-Order.chain} \ (\text{reflclp } R) \ (\text{lset } lxs) \rangle$
 $\langle \text{proof} \rangle$

lemma reflclp-subset: $\langle \text{reflclp } (\subset) = (\subseteq) \rangle$
 $\langle \text{proof} \rangle$

lemma sum-strict-mono2-enat:

fixes $f :: \langle 'a \Rightarrow \text{enat} \rangle$
assumes $\langle \text{finite } B \rangle \langle A \subseteq B \rangle \langle b \in B - A \rangle \langle f \ b > 0 \rangle \langle \infty \notin f \ ` B \rangle$
shows $\langle \text{sum } f \ A < \text{sum } f \ B \rangle$
 $\langle \text{proof} \rangle$

lemma infinite-lset-chain:

assumes $\langle \text{linfinite } lxs \rangle \langle \text{chain } R \ lxs \rangle \langle \text{irreflp } R \rangle \langle \text{transp } R \rangle$
shows $\langle \text{infinite } (\text{lset } lxs) \rangle$

⟨proof⟩

lemma *isum-eq-inf*: ⟨*isum* $f A = \infty \iff \text{infinite } \{a \in A. f a \neq 0\} \vee (\exists a \in A. f a = \infty)$ ⟩
⟨proof⟩

lemma *cmfinite-alt*: ⟨*cmfinite* $M = ((\exists x. \text{cmcount } M x = \infty) \vee \text{infinite } \{x. \text{cmcount } M x \neq 0\})$ ⟩
⟨proof⟩

lemma *cmset-alt*: ⟨*cmset* $M = \{a. \text{cmcount } M a \neq 0\}$ ⟩
⟨proof⟩

lift-definition *cmconst* :: ⟨ $'a \Rightarrow 'a \text{ cmset}$ ⟩ **is** ⟨repeat⟩ ⟨proof⟩

lemma *cmfinite-cmconst[simp]*: ⟨*cmfinite* (*cmconst* a)⟩
⟨proof⟩

lemma *cmfinite-cmimage*: ⟨*cmfinite* (*cmimage* $f M$) \iff *cmfinite* M ⟩
⟨proof⟩

locale *cmset-as-Quotient*
begin

setup-lifting *Quotient-cmset*

declare *cmempty.transfer*[*transfer-rule*]
declare *cmadd.transfer*[*transfer-rule*]
declare *cmset.map-transfer*[*transfer-rule*]
declare *cmset.set-transfer*[*transfer-rule*]
declare *cmfinite.transfer*[*transfer-rule*]

end

locale *cmset-as-typedef*
begin

setup-lifting *type-definition-cmset*

lemma *cmempty-transfer*[*transfer-rule*]: ⟨*pcr-cmset* $R (\lambda-. 0)$ *cmempty*⟩
⟨proof⟩

lemma *cmadd-transfer*[*transfer-rule*]: ⟨*rel-fun* (*pcr-cmset* R) (*rel-fun* (*pcr-cmset* R) ($\lambda M N x. M x + N x$) *cmadd*)⟩
⟨proof⟩

lemma *cmimage-transfer*[*transfer-rule*]: ⟨*rel-fun* ($=$) (*rel-fun* (*pcr-cmset* ($=$)) (*pcr-cmset* ($=$))) ($\lambda f M b. \text{isum } M \{a. f a = b\}$) *cmimage*⟩

⟨proof⟩

lemma *cmset-transfer*[*transfer-rule*]: ⟨*rel-fun* (*pcr-cmset* (=)) (=) ($\lambda M. \{a. M a \neq 0\}$) *cmset*⟩
⟨proof⟩

lemma *cmfinite-transfer*[*transfer-rule*]: ⟨*rel-fun* (*pcr-cmset* (=)) (=) ($\lambda M. (\exists x. M x = \infty) \vee \text{infinite } \{x. M x \neq 0\}$) *cmfinite*⟩
⟨proof⟩

lift-definition *cmreplace* :: ⟨'a *cmset* \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a *cmset*⟩ **is**
⟨ $\lambda f a b. f(a := \text{epred } (f a), b := \text{eSuc } (f b))$ ⟩
⟨proof⟩

lemma *cmfinite-cmreplace*: ⟨*cmfinite* *M* \Longrightarrow *cmfinite* (*cmreplace* *M* *a* *b*)⟩
⟨proof⟩
end

lifting-update *cmset.lifting*
lifting-forget *cmset.lifting*

end

11 Countably Infinite Multisets

theory *Countably-Infinite-Multiset*
imports *Countable-Multiset*
begin

typedef 'a *cinfmset* = ⟨{*M* :: 'a *cmset*. *cmfinite* *M*}⟩
⟨proof⟩

setup-lifting *type-definition-cinfmset*

lift-bnf (*no-warn-wits*) 'a *cinfmset*
for *map*: *cinfmimage* *rel*: *cinfmrel*
⟨proof⟩

lift-definition *cinfmcount* :: ⟨'a *cinfmset* \Rightarrow 'a \Rightarrow *enat*⟩ **is** *cmcount* ⟨proof⟩

context begin

interpretation *cmset-as-typedef* ⟨proof⟩

lift-definition *cinfmreplace* :: ⟨'a *cinfmset* \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a *cinfmset*⟩ **is** *cmreplace*
⟨proof⟩

lemma *set-cinfmset-alt*: ⟨*set-cinfmset* *xs* = {*a*. *cinfmcount* *xs* *a* \neq 0}⟩
⟨proof⟩

lemma *set-cinfsset-cinfmreplace*:

$\langle \text{set-cinfmtset } (\text{cinfmtreplace } M \ x \ y) \subseteq \text{set-cinfmtset } M \cup \{y\} \rangle$
including *cmset.lifting*
 $\langle \text{proof} \rangle$
end

lemma *inj-cinfmtcount*: $\langle \text{inj cinfmtcount} \rangle$
 $\langle \text{proof} \rangle$

lemma *in-range-cinfmtcount*:
 $\langle \text{countable } \{x. f \ x \neq 0\} \implies (\exists x. f \ x = \infty) \vee \text{infinite } \{x. f \ x \neq 0\} \implies f \in \text{range cinfmtcount} \rangle$
 $\langle \text{proof} \rangle$

lemma *cinfmtcount-cinfmtimage[simp]*:
 $\langle \text{cinfmtcount } (\text{cinfmtimage } f \ M) \ b = \text{isum } (\text{cinfmtcount } M) \ \{a. f \ a = b\} \rangle$
 $\langle \text{proof} \rangle$

lemma *countable-nonzero-cinfmtcount*: $\langle \text{countable } \{x. \text{cinfmtcount } M \ x \neq 0\} \rangle$
 $\langle \text{proof} \rangle$

lemma *infinite-nonzero-cinfmtcount*: $\langle (\exists x. \text{cinfmtcount } M \ x = \infty) \vee \text{infinite } \{x. \text{cinfmtcount } M \ x \neq 0\} \rangle$
 $\langle \text{proof} \rangle$

lemma *type-definition-cinfmtset*: $\langle \text{type-definition cinfmtcount } (\text{inv cinfmtcount}) \ \{f. \text{countable } \{x. f \ x \neq 0\} \wedge ((\exists x. f \ x = \infty) \vee \text{infinite } \{x. f \ x \neq 0\})\} \rangle$
 $\langle \text{proof} \rangle$

lifting-update *cinfmtset.lifting*
lifting-forget *cinfmtset.lifting*

definition *get-cinfmtset* :: $\langle 'a \ \text{cinfmtset} \Rightarrow \text{nat} \Rightarrow 'a \rangle$ **where**
 $\langle \text{get-cinfmtset } M \ i = \text{lth } (\text{rep-cmset } (\text{Rep-cinfmtset } M)) \ i \rangle$

context includes *cinfmtset.lifting* **begin**
interpretation *cmset-as-Quotient* $\langle \text{proof} \rangle$

lemma *get-cinfmtset-inject*:
 $\langle \forall i. \text{get-cinfmtset } M \ i = \text{get-cinfmtset } N \ i \implies M = N \rangle$
 $\langle \text{proof} \rangle$

end

end