Euler's Partition Theorem

Lukas Bulwahn

December 17, 2016

Abstract

Euler's Partition Theorem states that the number of partitions with only distinct parts is equal to the number of partitions with only odd parts. The combinatorial proof follows John Harrison's pre-existing HOL Light formalization [1]. To understand the rough idea of the proof, I read the lecture notes of the MIT course 18.312 on Algebraic Combinatorics [2] by Gregg Musiker. This theorem is the 45th theorem of the Top 100 Theorems list.

Contents

1 (Finite) multisets **3**
1.1 The type of multisets . . . 3
1.2 Representing multisets . . . 3
1.3 Basic operations . . . 5
1.3.1 Conversion to set and membership . . . 5
1.3.2 Union . . . 7
1.3.3 Difference . . . 8
1.3.4 Equality of multisets . . . 9
1.3.5 Pointwise ordering induced by count . . . 11
1.3.6 Intersection and bounded union . . . 14
1.3.7 Additional intersection facts . . . 15
1.3.8 Additional bounded union facts . . . 17
1.3.9 Subset is an order . . . 18
1.4 Replicate and repeat operations . . . 18
1.4.1 Simprocs . . . 19
1.4.2 Conditionally complete lattice . . . 20
1.4.3 Filter (with comprehension syntax) . . . 22
1.4.4 Size . . . 23
1.5 Induction and case splits . . . 25
1.5.1 Strong induction and subset induction for multisets . 25
1.6 The fold combinator . . . 26
1.7 Image . . . 27
1.8 Further conversions . . . 29

1.9 More properties of the replicate and repeat operations 32
1.10 Big operators 33
1.11 Alternative representations 39
1.11.1 Lists 39
1.12 The multiset order 41
1.12.1 Well-foundedness 41
1.12.2 Closure-free presentation 42
1.13 The multiset extension is cancellative for multiset union . . . 42
1.14 Quasi-executable version of the multiset extension 43
1.14.1 Partial-order properties 43
1.14.2 Monotonicity of multiset union 44
1.14.3 Termination proofs with multiset orders 44
1.15 Legacy theorem bindings 45
1.16 Naive implementation using lists 46
1.17 BNF setup 49
1.18 Size setup 51

2 Additions to Isabelle's Main Theories **51**
2.1 Addition to Finite-Set Theory 52
2.2 Additions to Groups-Big Theory 52
2.3 Addition to Set-Interval Theory 52
2.4 Additions to Multiset Theory 52

3 Number Partitions **53**
3.1 Number Partitions as $nat \Rightarrow nat$ Functions 53
3.2 Bounds and Finiteness of Number Partitions 53
3.3 Operations of Number Partitions 54
3.4 Number Partitions as Multisets on Natural Numbers 55
3.4.1 Relationship to Definition on Functions 55

4 Euler's Partition Theorem **56**
4.1 Preliminaries 56
4.1.1 Additions to Divides Theory 56
4.1.2 Additions to Groups-Big Theory 56
4.1.3 Additions to Set-Interval Theory 56
4.1.4 Additions to Nat Theory or Power Theory 56
4.1.5 Additions to Finite-Set Theory 56
4.2 Binary Encoding of Natural Numbers 57
4.3 Decomposition of a Number into a Power of Two and an Odd Number 58
4.4 Partitions With Only Distinct and Only Odd Parts 59
4.5 Euler's Partition Theorem 60

1 (Finite) multisets

theory *Multiset*
imports *Main*
begin

1.1 The type of multisets

definition $multiset = \{f :: 'a \Rightarrow nat.\ finite\ \{x.\ f\ x > 0\}\}$

typedef $'a\ multiset = multiset :: ('a \Rightarrow nat)\ set$
  morphisms *count Abs-multiset*
  ⟨*proof*⟩

setup-lifting *type-definition-multiset*

lemma *multiset-eq-iff*: $M = N \longleftrightarrow (\forall a.\ count\ M\ a = count\ N\ a)$
  ⟨*proof*⟩

lemma *multiset-eqI*: $(\bigwedge x.\ count\ A\ x = count\ B\ x) \Longrightarrow A = B$
  ⟨*proof*⟩

Preservation of the representing set *multiset*.

lemma *const0-in-multiset*: $(\lambda a.\ 0) \in multiset$
  ⟨*proof*⟩

lemma *only1-in-multiset*: $(\lambda b.\ if\ b = a\ then\ n\ else\ 0) \in multiset$
  ⟨*proof*⟩

lemma *union-preserves-multiset*: $M \in multiset \Longrightarrow N \in multiset \Longrightarrow (\lambda a.\ M\ a + N\ a) \in multiset$
  ⟨*proof*⟩

lemma *diff-preserves-multiset*:
  assumes $M \in multiset$
  shows $(\lambda a.\ M\ a - N\ a) \in multiset$
⟨*proof*⟩

lemma *filter-preserves-multiset*:
  assumes $M \in multiset$
  shows $(\lambda x.\ if\ P\ x\ then\ M\ x\ else\ 0) \in multiset$
⟨*proof*⟩

lemmas *in-multiset = const0-in-multiset only1-in-multiset*
  union-preserves-multiset diff-preserves-multiset filter-preserves-multiset

1.2 Representing multisets

Multiset enumeration

instantiation *multiset* :: (*type*) *cancel-comm-monoid-add*
begin

lift-definition *zero-multiset* :: *'a multiset* **is** *λa. 0*
⟨*proof*⟩

abbreviation *Mempty* :: *'a multiset* ({#}) **where**
Mempty ≡ 0

lift-definition *plus-multiset* :: *'a multiset ⇒ 'a multiset ⇒ 'a multiset* **is** *λM N.* (*λa. M a + N a*)
⟨*proof*⟩

lift-definition *minus-multiset* :: *'a multiset ⇒ 'a multiset ⇒ 'a multiset* **is** *λ M N. λa. M a − N a*
⟨*proof*⟩

instance
⟨*proof*⟩

end

context
begin

qualified definition *is-empty* :: *'a multiset ⇒ bool* **where**
[*code-abbrev*]: *is-empty A ⟷ A = {#}*

end

lemma *add-mset-in-multiset*:
assumes *M*: ‹*M ∈ multiset*›
shows ‹(*λb. if b = a then Suc* (*M b*) *else M b*) *∈ multiset*›
⟨*proof*⟩

lift-definition *add-mset* :: *'a ⇒ 'a multiset ⇒ 'a multiset* **is**
λa M b. if b = a then Suc (*M b*) *else M b*
⟨*proof*⟩

syntax
-multiset :: *args ⇒ 'a multiset* ({#(-)#})
translations
{#*x*, *xs*#} == *CONST add-mset x* {#*xs*#}
{#*x*#} == *CONST add-mset x* {#}

lemma *count-empty* [*simp*]: *count* {#} *a = 0*
⟨*proof*⟩

lemma *count-add-mset* [*simp*]:

count (*add-mset b A*) *a* = (*if b* = *a then Suc* (*count A a*) *else count A a*)
⟨*proof*⟩

lemma *count-single*: *count* {#*b*#} *a* = (*if b* = *a then 1 else 0*)
⟨*proof*⟩

lemma
add-mset-not-empty [*simp*]: ‹*add-mset a A* ≠ {#}› **and**
empty-not-add-mset [*simp*]: {#} ≠ *add-mset a A*
⟨*proof*⟩

lemma *add-mset-add-mset-same-iff* [*simp*]:
add-mset a A = *add-mset a B* ⟷ *A* = *B*
⟨*proof*⟩

lemma *add-mset-commute*:
add-mset x (*add-mset y M*) = *add-mset y* (*add-mset x M*)
⟨*proof*⟩

1.3 Basic operations

1.3.1 Conversion to set and membership

definition *set-mset* :: *'a multiset* ⇒ *'a set*
where *set-mset M* = {*x*. *count M x* > *0*}

abbreviation *Melem* :: *'a* ⇒ *'a multiset* ⇒ *bool*
where *Melem a M* ≡ *a* ∈ *set-mset M*

notation
Melem (*op* ∈#) **and**
Melem ((-/ ∈# -) [*51*, *51*] *50*)

notation (*ASCII*)
Melem (*op* :#) **and**
Melem ((-/ :# -) [*51*, *51*] *50*)

abbreviation *not-Melem* :: *'a* ⇒ *'a multiset* ⇒ *bool*
where *not-Melem a M* ≡ *a* ∉ *set-mset M*

notation
not-Melem (*op* ∉#) **and**
not-Melem ((-/ ∉# -) [*51*, *51*] *50*)

notation (*ASCII*)
not-Melem (*op* ~:#) **and**
not-Melem ((-/ ~:# -) [*51*, *51*] *50*)

context
begin

qualified abbreviation *Ball* :: *'a multiset* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*
where *Ball M* ≡ *Set.Ball* (*set-mset M*)

qualified abbreviation *Bex* :: *'a multiset* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*
where *Bex M* ≡ *Set.Bex* (*set-mset M*)

end

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

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

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

lemma *count-eq-zero-iff*:
count M x = *0* ⟷ *x* ∉# *M*
⟨*proof*⟩

lemma *not-in-iff*:
x ∉# *M* ⟷ *count M x* = *0*
⟨*proof*⟩

lemma *count-greater-zero-iff* [*simp*]:
count M x > *0* ⟷ *x* ∈# *M*
⟨*proof*⟩

lemma *count-inI*:
assumes *count M x* = *0* ⟹ *False*
shows *x* ∈# *M*
⟨*proof*⟩

lemma *in-countE*:
assumes *x* ∈# *M*
obtains *n* **where** *count M x* = *Suc n*
⟨*proof*⟩

lemma *count-greater-eq-Suc-zero-iff* [*simp*]:
count M x ≥ *Suc 0* ⟷ *x* ∈# *M*
⟨*proof*⟩

lemma *count-greater-eq-one-iff* [*simp*]:
count M x ≥ *1* ⟷ *x* ∈# *M*

⟨*proof*⟩

lemma *set-mset-empty* [*simp*]:
set-mset {#} = {}
⟨*proof*⟩

lemma *set-mset-single*:
set-mset {#*b*#} = {*b*}
⟨*proof*⟩

lemma *set-mset-eq-empty-iff* [*simp*]:
set-mset M = {} ⟷ *M* = {#}
⟨*proof*⟩

lemma *finite-set-mset* [*iff*]:
finite (*set-mset M*)
⟨*proof*⟩

lemma *set-mset-add-mset-insert* [*simp*]: ‹*set-mset* (*add-mset a A*) = *insert a* (*set-mset A*)›
⟨*proof*⟩

lemma *multiset-nonemptyE* [*elim*]:
assumes *A* ≠ {#}
obtains *x* **where** *x* ∈# *A*
⟨*proof*⟩

1.3.2 Union

lemma *count-union* [*simp*]:
count (*M* + *N*) *a* = *count M a* + *count N a*
⟨*proof*⟩

lemma *set-mset-union* [*simp*]:
set-mset (*M* + *N*) = *set-mset M* ∪ *set-mset N*
⟨*proof*⟩

lemma *union-mset-add-mset-left* [*simp*]:
add-mset a A + *B* = *add-mset a* (*A* + *B*)
⟨*proof*⟩

lemma *union-mset-add-mset-right* [*simp*]:
A + *add-mset a B* = *add-mset a* (*A* + *B*)
⟨*proof*⟩

lemma *add-mset-add-single*: ‹*add-mset a A* = *A* + {#*a*#}›
⟨*proof*⟩

1.3.3 Difference

instance *multiset* :: (*type*) *comm-monoid-diff*
⟨*proof*⟩

lemma *count-diff* [*simp*]:
count (*M* − *N*) *a* = *count M a* − *count N a*
⟨*proof*⟩

lemma *add-mset-diff-bothsides*:
‹*add-mset a M* − *add-mset a A* = *M* − *A*›
⟨*proof*⟩

lemma *in-diff-count*:
a ∈# *M* − *N* ⟷ *count N a* < *count M a*
⟨*proof*⟩

lemma *count-in-diffI*:
assumes ⋀*n*. *count N x* = *n* + *count M x* ⟹ *False*
shows *x* ∈# *M* − *N*
⟨*proof*⟩

lemma *in-diff-countE*:
assumes *x* ∈# *M* − *N*
obtains *n* **where** *count M x* = *Suc n* + *count N x*
⟨*proof*⟩

lemma *in-diffD*:
assumes *a* ∈# *M* − *N*
shows *a* ∈# *M*
⟨*proof*⟩

lemma *set-mset-diff*:
set-mset (*M* − *N*) = {*a*. *count N a* < *count M a*}
⟨*proof*⟩

lemma *diff-empty* [*simp*]: *M* − {#} = *M* ∧ {#} − *M* = {#}
⟨*proof*⟩

lemma *diff-cancel*: *A* − *A* = {#}
⟨*proof*⟩

lemma *diff-union-cancelR*: *M* + *N* − *N* = (*M*::′*a multiset*)
⟨*proof*⟩

lemma *diff-union-cancelL*: *N* + *M* − *N* = (*M*::′*a multiset*)
⟨*proof*⟩

lemma *diff-right-commute*:
fixes *M N Q* :: ′*a multiset*

shows $M - N - Q = M - Q - N$
⟨*proof*⟩

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

lemma *insert-DiffM* [*simp*]: $x \in\# M \Longrightarrow add\text{-}mset\ x\ (M - \{\#x\#\}) = M$
⟨*proof*⟩

lemma *insert-DiffM2*: $x \in\# M \Longrightarrow (M - \{\#x\#\}) + \{\#x\#\} = M$
⟨*proof*⟩

lemma *diff-union-swap*: $a \neq b \Longrightarrow add\text{-}mset\ b\ (M - \{\#a\#\}) = add\text{-}mset\ b\ M - \{\#a\#\}$
⟨*proof*⟩

lemma *diff-add-mset-swap* [*simp*]: $b \notin\# A \Longrightarrow add\text{-}mset\ b\ M - A = add\text{-}mset\ b\ (M - A)$
⟨*proof*⟩

lemma *diff-union-swap2* [*simp*]: $y \in\# M \Longrightarrow add\text{-}mset\ x\ M - \{\#y\#\} = add\text{-}mset\ x\ (M - \{\#y\#\})$
⟨*proof*⟩

lemma *diff-diff-add-mset* [*simp*]: $(M::'a\ multiset) - N - P = M - (N + P)$
⟨*proof*⟩

lemma *diff-union-single-conv*:
$a \in\# J \Longrightarrow I + J - \{\#a\#\} = I + (J - \{\#a\#\})$
⟨*proof*⟩

lemma *mset-add* [*elim?*]:
assumes $a \in\# A$
obtains B **where** $A = add\text{-}mset\ a\ B$
⟨*proof*⟩

lemma *union-iff*:
$a \in\# A + B \longleftrightarrow a \in\# A \vee a \in\# B$
⟨*proof*⟩

1.3.4 Equality of multisets

lemma *single-eq-single* [*simp*]: $\{\#a\#\} = \{\#b\#\} \longleftrightarrow a = b$
⟨*proof*⟩

lemma *union-eq-empty* [*iff*]: $M + N = \{\#\} \longleftrightarrow M = \{\#\} \wedge N = \{\#\}$
⟨*proof*⟩

lemma *empty-eq-union* [*iff*]: $\{\#\} = M + N \longleftrightarrow M = \{\#\} \wedge N = \{\#\}$
⟨*proof*⟩

lemma *multi-self-add-other-not-self* [*simp*]: $M = \textit{add-mset}\ x\ M \longleftrightarrow \textit{False}$
⟨*proof*⟩

lemma *add-mset-remove-trivial* [*simp*]: ‹*add-mset x M* − {#*x*#} = *M*›
⟨*proof*⟩

lemma *diff-single-trivial*: $\neg\ x \in\#\ M \Longrightarrow M - \{\#x\#\} = M$
⟨*proof*⟩

lemma *diff-single-eq-union*: $x \in\#\ M \Longrightarrow M - \{\#x\#\} = N \longleftrightarrow M = \textit{add-mset}\ x\ N$
⟨*proof*⟩

lemma *union-single-eq-diff*: $\textit{add-mset}\ x\ M = N \Longrightarrow M = N - \{\#x\#\}$
⟨*proof*⟩

lemma *union-single-eq-member*: $\textit{add-mset}\ x\ M = N \Longrightarrow x \in\#\ N$
⟨*proof*⟩

lemma *add-mset-remove-trivial-If*:
$\textit{add-mset}\ a\ (N - \{\#a\#\}) = (\textit{if}\ a \in\#\ N\ \textit{then}\ N\ \textit{else}\ \textit{add-mset}\ a\ N)$
⟨*proof*⟩

lemma *add-mset-remove-trivial-eq*: ‹*N* = *add-mset a* (*N* − {#*a*#}) ⟷ *a* ∈# *N*›
⟨*proof*⟩

lemma *union-is-single*:
$M + N = \{\#a\#\} \longleftrightarrow M = \{\#a\#\} \wedge N = \{\#\} \vee M = \{\#\} \wedge N = \{\#a\#\}$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *single-is-union*: $\{\#a\#\} = M + N \longleftrightarrow \{\#a\#\} = M \wedge N = \{\#\} \vee M = \{\#\} \wedge \{\#a\#\} = N$
⟨*proof*⟩

lemma *add-eq-conv-diff*:
$\textit{add-mset}\ a\ M = \textit{add-mset}\ b\ N \longleftrightarrow M = N \wedge a = b \vee M = \textit{add-mset}\ b\ (N - \{\#a\#\}) \wedge N = \textit{add-mset}\ a\ (M - \{\#b\#\})$
(**is** *?lhs* ⟷ *?rhs*)

⟨*proof*⟩

lemma *add-mset-eq-single* [*iff*]: $\textit{add-mset}\ b\ M = \{\#a\#\} \longleftrightarrow b = a \wedge M = \{\#\}$
⟨*proof*⟩

lemma *single-eq-add-mset* [*iff*]: $\{\#a\#\} = \textit{add-mset}\ b\ M \longleftrightarrow b = a \wedge M = \{\#\}$
⟨*proof*⟩

lemma *insert-noteq-member*:
assumes *BC*: $\textit{add-mset}\ b\ B = \textit{add-mset}\ c\ C$
and *bnotc*: $b \neq c$
shows $c \in\# B$
⟨*proof*⟩

lemma *add-eq-conv-ex*:
$(\textit{add-mset}\ a\ M = \textit{add-mset}\ b\ N) =$
$(M = N \wedge a = b \vee (\exists K.\ M = \textit{add-mset}\ b\ K \wedge N = \textit{add-mset}\ a\ K))$
⟨*proof*⟩

lemma *multi-member-split*: $x \in\# M \Longrightarrow \exists A.\ M = \textit{add-mset}\ x\ A$
⟨*proof*⟩

lemma *multiset-add-sub-el-shuffle*:
assumes $c \in\# B$
and $b \neq c$
shows $\textit{add-mset}\ b\ (B - \{\#c\#\}) = \textit{add-mset}\ b\ B - \{\#c\#\}$
⟨*proof*⟩

lemma *add-mset-eq-singleton-iff* [*iff*]:
$\textit{add-mset}\ x\ M = \{\#y\#\} \longleftrightarrow M = \{\#\} \wedge x = y$
⟨*proof*⟩

1.3.5 Pointwise ordering induced by count

definition $\textit{subseteq-mset} :: 'a\ \textit{multiset} \Rightarrow 'a\ \textit{multiset} \Rightarrow \textit{bool}$ (**infix** $\subseteq\#$ *50*)
where $A \subseteq\# B = (\forall a.\ \textit{count}\ A\ a \leq \textit{count}\ B\ a)$

definition $\textit{subset-mset} :: 'a\ \textit{multiset} \Rightarrow 'a\ \textit{multiset} \Rightarrow \textit{bool}$ (**infix** $\subset\#$ *50*)
where $A \subset\# B = (A \subseteq\# B \wedge A \neq B)$

abbreviation (*input*) $\textit{supseteq-mset} :: 'a\ \textit{multiset} \Rightarrow 'a\ \textit{multiset} \Rightarrow \textit{bool}$ (**infix** $\supseteq\#$ *50*)
where $\textit{supseteq-mset}\ A\ B \equiv B \subseteq\# A$

abbreviation (*input*) $\textit{supset-mset} :: 'a\ \textit{multiset} \Rightarrow 'a\ \textit{multiset} \Rightarrow \textit{bool}$ (**infix** $\supset\#$ *50*)
where $\textit{supset-mset}\ A\ B \equiv B \subset\# A$

notation (*input*)
$\textit{subseteq-mset}$ (**infix** $\leq\#$ *50*) **and**
$\textit{supseteq-mset}$ (**infix** $\geq\#$ *50*)

notation (*ASCII*)

subseteq-mset (**infix** <=# *50*) **and**
subset-mset (**infix** <# *50*) **and**
supseteq-mset (**infix** >=# *50*) **and**
supset-mset (**infix** ># *50*)

interpretation *subset-mset*: *ordered-ab-semigroup-add-imp-le op + op − op ⊆# op ⊂#*
⟨*proof*⟩

interpretation *subset-mset*: *ordered-ab-semigroup-monoid-add-imp-le op + 0 op − op ≤# op <#*
⟨*proof*⟩

lemma *mset-subset-eqI*:
(⋀*a*. *count A a* ≤ *count B a*) ⟹ *A* ⊆# *B*
⟨*proof*⟩

lemma *mset-subset-eq-count*:
A ⊆# *B* ⟹ *count A a* ≤ *count B a*
⟨*proof*⟩

lemma *mset-subset-eq-exists-conv*: (*A*::'*a multiset*) ⊆# *B* ⟷ (∃ *C*. *B* = *A* + *C*)
⟨*proof*⟩

interpretation *subset-mset*: *ordered-cancel-comm-monoid-diff op + 0 op ≤# op <# op −*
⟨*proof*⟩

declare *subset-mset.add-diff-assoc*[*simp*] *subset-mset.add-diff-assoc2*[*simp*]

lemma *mset-subset-eq-mono-add-right-cancel*: (*A*::'*a multiset*) + *C* ⊆# *B* + *C* ⟷ *A* ⊆# *B*
⟨*proof*⟩

lemma *mset-subset-eq-mono-add-left-cancel*: *C* + (*A*::'*a multiset*) ⊆# *C* + *B* ⟷ *A* ⊆# *B*
⟨*proof*⟩

lemma *mset-subset-eq-mono-add*: (*A*::'*a multiset*) ⊆# *B* ⟹ *C* ⊆# *D* ⟹ *A* + *C* ⊆# *B* + *D*
⟨*proof*⟩

lemma *mset-subset-eq-add-left*: (*A*::'*a multiset*) ⊆# *A* + *B*
⟨*proof*⟩

lemma *mset-subset-eq-add-right*: *B* ⊆# (*A*::'*a multiset*) + *B*
⟨*proof*⟩

lemma *single-subset-iff* [*simp*]:

$\{\#a\#\} \subseteq\# M \longleftrightarrow a \in\# M$
⟨*proof*⟩

lemma *mset-subset-eq-single*: $a \in\# B \Longrightarrow \{\#a\#\} \subseteq\# B$
⟨*proof*⟩

lemma *mset-subset-eq-add-mset-cancel*: ‹*add-mset a A* $\subseteq\#$ *add-mset a B* $\longleftrightarrow$ *A* $\subseteq\#$ *B*›
⟨*proof*⟩

lemma *multiset-diff-union-assoc*:
fixes *A B C D* :: *'a multiset*
shows $C \subseteq\# B \Longrightarrow A + B - C = A + (B - C)$
⟨*proof*⟩

lemma *mset-subset-eq-multiset-union-diff-commute*:
fixes *A B C D* :: *'a multiset*
shows $B \subseteq\# A \Longrightarrow A - B + C = A + C - B$
⟨*proof*⟩

lemma *diff-subset-eq-self*[*simp*]:
(*M*::*'a multiset*) − *N* $\subseteq\#$ *M*
⟨*proof*⟩

lemma *mset-subset-eqD*:
assumes $A \subseteq\# B$ **and** $x \in\# A$
shows $x \in\# B$
⟨*proof*⟩

lemma *mset-subsetD*:
$A \subset\# B \Longrightarrow x \in\# A \Longrightarrow x \in\# B$
⟨*proof*⟩

lemma *set-mset-mono*:
$A \subseteq\# B \Longrightarrow$ *set-mset A* $\subseteq$ *set-mset B*
⟨*proof*⟩

lemma *mset-subset-eq-insertD*:
add-mset x A $\subseteq\# B \Longrightarrow x \in\# B \wedge A \subset\# B$
⟨*proof*⟩

lemma *mset-subset-insertD*:
add-mset x A $\subset\# B \Longrightarrow x \in\# B \wedge A \subset\# B$
⟨*proof*⟩

lemma *mset-subset-of-empty*[*simp*]: $A \subset\# \{\#\} \longleftrightarrow$ *False*
⟨*proof*⟩

lemma *empty-subset-add-mset*[*simp*]: $\{\#\} <\#$ *add-mset x M*

⟨*proof*⟩

lemma *empty-le*: $\{\#\} \subseteq\# A$
⟨*proof*⟩

lemma *insert-subset-eq-iff*:
$add\text{-}mset\ a\ A \subseteq\# B \longleftrightarrow a \in\# B \wedge A \subseteq\# B - \{\#a\#\}$
⟨*proof*⟩

lemma *insert-union-subset-iff*:
$add\text{-}mset\ a\ A \subset\# B \longleftrightarrow a \in\# B \wedge A \subset\# B - \{\#a\#\}$
⟨*proof*⟩

lemma *subset-eq-diff-conv*:
$A - C \subseteq\# B \longleftrightarrow A \subseteq\# B + C$
⟨*proof*⟩

lemma *multi-psub-of-add-self* [*simp*]: $A \subset\# add\text{-}mset\ x\ A$
⟨*proof*⟩

lemma *multi-psub-self*: $A \subset\# A = False$
⟨*proof*⟩

lemma *mset-subset-add-mset* [*simp*]: $add\text{-}mset\ x\ N \subset\# add\text{-}mset\ x\ M \longleftrightarrow N \subset\# M$
⟨*proof*⟩

lemma *mset-subset-diff-self*: $c \in\# B \Longrightarrow B - \{\#c\#\} \subset\# B$
⟨*proof*⟩

lemma *Diff-eq-empty-iff-mset*: $A - B = \{\#\} \longleftrightarrow A \subseteq\# B$
⟨*proof*⟩

lemma *add-mset-subseteq-single-iff* [*iff*]: $add\text{-}mset\ a\ M \subseteq\# \{\#b\#\} \longleftrightarrow M = \{\#\} \wedge a = b$
⟨*proof*⟩

1.3.6 Intersection and bounded union

definition *inf-subset-mset* :: *'a multiset* ⇒ *'a multiset* ⇒ *'a multiset* (**infixl** ∩# *70*) **where**
multiset-inter-def: $inf\text{-}subset\text{-}mset\ A\ B = A - (A - B)$

interpretation *subset-mset*: *semilattice-inf inf-subset-mset op* ⊆# *op* ⊂#
⟨*proof*⟩

definition *sup-subset-mset* :: *'a multiset* ⇒ *'a multiset* ⇒ *'a multiset*(**infixl** ∪# *70*)
where $sup\text{-}subset\text{-}mset\ A\ B = A + (B - A)$ — FIXME irregular fact name

interpretation *subset-mset*: *semilattice-sup sup-subset-mset op* $\subseteq\#$ *op* $\subset\#$
⟨*proof*⟩

interpretation *subset-mset*: *bounded-lattice-bot op* $\cap\#$ *op* $\subseteq\#$ *op* $\subset\#$
op $\cup\#$ $\{\#\}$
⟨*proof*⟩

1.3.7 Additional intersection facts

lemma *multiset-inter-count* [*simp*]:
fixes A B :: $'a$ *multiset*
shows *count* $(A \cap\# B)$ x = *min* (*count* A x) (*count* B x)
⟨*proof*⟩

lemma *set-mset-inter* [*simp*]:
set-mset $(A \cap\# B)$ = *set-mset* $A \cap$ *set-mset* B
⟨*proof*⟩

lemma *diff-intersect-left-idem* [*simp*]:
$M - M \cap\# N = M - N$
⟨*proof*⟩

lemma *diff-intersect-right-idem* [*simp*]:
$M - N \cap\# M = M - N$
⟨*proof*⟩

lemma *multiset-inter-single*[*simp*]: $a \neq b \Longrightarrow \{\#a\#\} \cap\# \{\#b\#\} = \{\#\}$
⟨*proof*⟩

lemma *multiset-union-diff-commute*:
assumes $B \cap\# C = \{\#\}$
shows $A + B - C = A - C + B$
⟨*proof*⟩

lemma *disjunct-not-in*:
$A \cap\# B = \{\#\} \longleftrightarrow (\forall a.\ a \notin\# A \vee a \notin\# B)$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

lemma *inter-mset-empty-distrib-right*: $A \cap\# (B + C) = \{\#\} \longleftrightarrow A \cap\# B = \{\#\} \wedge A \cap\# C = \{\#\}$
⟨*proof*⟩

lemma *inter-mset-empty-distrib-left*: $(A + B) \cap\# C = \{\#\} \longleftrightarrow A \cap\# C = \{\#\} \wedge B \cap\# C = \{\#\}$
⟨*proof*⟩

lemma *add-mset-inter-add-mset*[*simp*]:
add-mset a $A \cap\#$ *add-mset* a B = *add-mset* a $(A \cap\# B)$

⟨*proof*⟩

lemma *add-mset-disjoint* [*simp*]:
add-mset a A ∩# *B* = {#} ⟷ *a* ∉# *B* ∧ *A* ∩# *B* = {#}
{#} = *add-mset a A* ∩# *B* ⟷ *a* ∉# *B* ∧ {#} = *A* ∩# *B*
⟨*proof*⟩

lemma *disjoint-add-mset* [*simp*]:
B ∩# *add-mset a A* = {#} ⟷ *a* ∉# *B* ∧ *B* ∩# *A* = {#}
{#} = *A* ∩# *add-mset b B* ⟷ *b* ∉# *A* ∧ {#} = *A* ∩# *B*
⟨*proof*⟩

lemma *inter-add-left1*: ¬ *x* ∈# *N* ⟹ (*add-mset x M*) ∩# *N* = *M* ∩# *N*
⟨*proof*⟩

lemma *inter-add-left2*: *x* ∈# *N* ⟹ (*add-mset x M*) ∩# *N* = *add-mset x* (*M* ∩# (*N* − {#*x*#}))
⟨*proof*⟩

lemma *inter-add-right1*: ¬ *x* ∈# *N* ⟹ *N* ∩# (*add-mset x M*) = *N* ∩# *M*
⟨*proof*⟩

lemma *inter-add-right2*: *x* ∈# *N* ⟹ *N* ∩# (*add-mset x M*) = *add-mset x* ((*N* − {#*x*#}) ∩# *M*)
⟨*proof*⟩

lemma *disjunct-set-mset-diff*:
assumes *M* ∩# *N* = {#}
shows *set-mset* (*M* − *N*) = *set-mset M*
⟨*proof*⟩

lemma *at-most-one-mset-mset-diff*:
assumes *a* ∉# *M* − {#*a*#}
shows *set-mset* (*M* − {#*a*#}) = *set-mset M* − {*a*}
⟨*proof*⟩

lemma *more-than-one-mset-mset-diff*:
assumes *a* ∈# *M* − {#*a*#}
shows *set-mset* (*M* − {#*a*#}) = *set-mset M*
⟨*proof*⟩

lemma *inter-iff*:
a ∈# *A* ∩# *B* ⟷ *a* ∈# *A* ∧ *a* ∈# *B*
⟨*proof*⟩

lemma *inter-union-distrib-left*:
A ∩# *B* + *C* = (*A* + *C*) ∩# (*B* + *C*)
⟨*proof*⟩

lemma *inter-union-distrib-right*:
$C + A \cap\# B = (C + A) \cap\# (C + B)$
⟨*proof*⟩

lemma *inter-subset-eq-union*:
$A \cap\# B \subseteq\# A + B$
⟨*proof*⟩

1.3.8 Additional bounded union facts

lemma *sup-subset-mset-count* [*simp*]: — FIXME irregular fact name
count $(A \cup\# B)$ $x = max$ $(count\ A\ x)$ $(count\ B\ x)$
⟨*proof*⟩

lemma *set-mset-sup* [*simp*]:
set-mset $(A \cup\# B) =$ *set-mset* $A \cup$ *set-mset* B
⟨*proof*⟩

lemma *sup-union-left1* [*simp*]: $\neg\ x \in\# N \Longrightarrow (add\text{-}mset\ x\ M) \cup\# N = add\text{-}mset\ x\ (M \cup\# N)$
⟨*proof*⟩

lemma *sup-union-left2*: $x \in\# N \Longrightarrow (add\text{-}mset\ x\ M) \cup\# N = add\text{-}mset\ x\ (M \cup\# (N - \{\#x\#\}))$
⟨*proof*⟩

lemma *sup-union-right1* [*simp*]: $\neg\ x \in\# N \Longrightarrow N \cup\# (add\text{-}mset\ x\ M) = add\text{-}mset\ x\ (N \cup\# M)$
⟨*proof*⟩

lemma *sup-union-right2*: $x \in\# N \Longrightarrow N \cup\# (add\text{-}mset\ x\ M) = add\text{-}mset\ x\ ((N - \{\#x\#\}) \cup\# M)$
⟨*proof*⟩

lemma *sup-union-distrib-left*:
$A \cup\# B + C = (A + C) \cup\# (B + C)$
⟨*proof*⟩

lemma *union-sup-distrib-right*:
$C + A \cup\# B = (C + A) \cup\# (C + B)$
⟨*proof*⟩

lemma *union-diff-inter-eq-sup*:
$A + B - A \cap\# B = A \cup\# B$
⟨*proof*⟩

lemma *union-diff-sup-eq-inter*:
$A + B - A \cup\# B = A \cap\# B$
⟨*proof*⟩

lemma *add-mset-union*:
‹*add-mset a A* ∪# *add-mset a B* = *add-mset a* (*A* ∪# *B*)›
⟨*proof*⟩

1.3.9 Subset is an order

interpretation *subset-mset*: *order op* ⊆# *op* ⊂# ⟨*proof*⟩

1.4 Replicate and repeat operations

definition *replicate-mset* :: *nat* ⇒ *'a* ⇒ *'a multiset* **where**
replicate-mset n x = (*add-mset x* ^^ *n*) {#}

lemma *replicate-mset-0*[*simp*]: *replicate-mset 0 x* = {#}
⟨*proof*⟩

lemma *replicate-mset-Suc* [*simp*]: *replicate-mset* (*Suc n*) *x* = *add-mset x* (*replicate-mset n x*)
⟨*proof*⟩

lemma *count-replicate-mset*[*simp*]: *count* (*replicate-mset n x*) *y* = (*if y* = *x then n else 0*)
⟨*proof*⟩

fun *repeat-mset* :: *nat* ⇒ *'a multiset* ⇒ *'a multiset* **where**
repeat-mset 0 - = {#} |
repeat-mset (*Suc n*) *A* = *A* + *repeat-mset n A*

lemma *count-repeat-mset* [*simp*]: *count* (*repeat-mset i A*) *a* = *i* ∗ *count A a*
⟨*proof*⟩

lemma *repeat-mset-right* [*simp*]: *repeat-mset a* (*repeat-mset b A*) = *repeat-mset* (*a* ∗ *b*) *A*
⟨*proof*⟩

lemma *left-diff-repeat-mset-distrib'*: ‹*repeat-mset* (*i* − *j*) *u* = *repeat-mset i u* − *repeat-mset j u*›
⟨*proof*⟩

lemma *left-add-mult-distrib-mset*:
repeat-mset i u + (*repeat-mset j u* + *k*) = *repeat-mset* (*i*+*j*) *u* + *k*
⟨*proof*⟩

lemma *repeat-mset-distrib*:
repeat-mset (*m* + *n*) *A* = *repeat-mset m A* + *repeat-mset n A*
⟨*proof*⟩

lemma *repeat-mset-distrib2*[*simp*]:
repeat-mset n (*A* + *B*) = *repeat-mset n A* + *repeat-mset n B*

⟨*proof*⟩

lemma *repeat-mset-replicate-mset*[*simp*]:
repeat-mset n {#*a*#} = *replicate-mset n a*
⟨*proof*⟩

lemma *repeat-mset-distrib-add-mset*[*simp*]:
repeat-mset n (*add-mset a A*) = *replicate-mset n a* + *repeat-mset n A*
⟨*proof*⟩

lemma *repeat-mset-empty*[*simp*]: *repeat-mset n* {#} = {#}
⟨*proof*⟩

1.4.1 Simprocs

lemma *mset-diff-add-eq1*:
j ≤ (*i*::*nat*) ⟹ ((*repeat-mset i u* + *m*) − (*repeat-mset j u* + *n*)) = ((*repeat-mset* (*i*−*j*) *u* + *m*) − *n*)
⟨*proof*⟩

lemma *mset-diff-add-eq2*:
i ≤ (*j*::*nat*) ⟹ ((*repeat-mset i u* + *m*) − (*repeat-mset j u* + *n*)) = (*m* − (*repeat-mset* (*j*−*i*) *u* + *n*))
⟨*proof*⟩

lemma *mset-eq-add-iff1*:
j ≤ (*i*::*nat*) ⟹ (*repeat-mset i u* + *m* = *repeat-mset j u* + *n*) = (*repeat-mset* (*i*−*j*) *u* + *m* = *n*)
⟨*proof*⟩

lemma *mset-eq-add-iff2*:
i ≤ (*j*::*nat*) ⟹ (*repeat-mset i u* + *m* = *repeat-mset j u* + *n*) = (*m* = *repeat-mset* (*j*−*i*) *u* + *n*)
⟨*proof*⟩

lemma *mset-subseteq-add-iff1*:
j ≤ (*i*::*nat*) ⟹ (*repeat-mset i u* + *m* ⊆# *repeat-mset j u* + *n*) = (*repeat-mset* (*i*−*j*) *u* + *m* ⊆# *n*)
⟨*proof*⟩

lemma *mset-subseteq-add-iff2*:
i ≤ (*j*::*nat*) ⟹ (*repeat-mset i u* + *m* ⊆# *repeat-mset j u* + *n*) = (*m* ⊆# *repeat-mset* (*j*−*i*) *u* + *n*)
⟨*proof*⟩

lemma *mset-subset-add-iff1*:
j ≤ (*i*::*nat*) ⟹ (*repeat-mset i u* + *m* ⊂# *repeat-mset j u* + *n*) = (*repeat-mset* (*i*−*j*) *u* + *m* ⊂# *n*)
⟨*proof*⟩

lemma *mset-subset-add-iff2*:
$i \leq (j::nat) \Longrightarrow (repeat\text{-}mset\ i\ u + m \subset\# repeat\text{-}mset\ j\ u + n) = (m \subset\# repeat\text{-}mset\ (j-i)\ u + n)$
⟨*proof*⟩

⟨*ML*⟩

1.4.2 Conditionally complete lattice

instantiation *multiset* :: (*type*) *Inf*
begin

lift-definition *Inf-multiset* :: *'a multiset set* ⇒ *'a multiset* **is**
$\lambda A\ i.\ if\ A = \{\}\ then\ 0\ else\ Inf\ ((\lambda f.\ f\ i)\ `\ A)$
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *Inf-multiset-empty*: $Inf\ \{\} = \{\#\}$
⟨*proof*⟩

lemma *count-Inf-multiset-nonempty*: $A \neq \{\} \Longrightarrow count\ (Inf\ A)\ x = Inf\ ((\lambda X.\ count\ X\ x)\ `\ A)$
⟨*proof*⟩

instantiation *multiset* :: (*type*) *Sup*
begin

definition *Sup-multiset* :: *'a multiset set* ⇒ *'a multiset* **where**
$Sup\text{-}multiset\ A = (if\ A \neq \{\} \wedge subset\text{-}mset.bdd\text{-}above\ A\ then$
$\quad Abs\text{-}multiset\ (\lambda i.\ Sup\ ((\lambda X.\ count\ X\ i)\ `\ A))\ else\ \{\#\})$

lemma *Sup-multiset-empty*: $Sup\ \{\} = \{\#\}$
⟨*proof*⟩

lemma *Sup-multiset-unbounded*: $\neg subset\text{-}mset.bdd\text{-}above\ A \Longrightarrow Sup\ A = \{\#\}$
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *bdd-above-multiset-imp-bdd-above-count*:
assumes *subset-mset.bdd-above* (*A* :: *'a multiset set*)

shows *bdd-above* ((λ*X*. *count X x*) ‘ *A*)
⟨*proof*⟩

lemma *bdd-above-multiset-imp-finite-support*:
assumes *A* ≠ {} *subset-mset.bdd-above* (*A* :: ′*a multiset set*)
shows *finite* (⋃ *X*∈*A*. {*x*. *count X x* > *0*})
⟨*proof*⟩

lemma *Sup-multiset-in-multiset*:
assumes *A* ≠ {} *subset-mset.bdd-above A*
shows (λ*i*. *SUP X*:*A*. *count X i*) ∈ *multiset*
⟨*proof*⟩

lemma *count-Sup-multiset-nonempty*:
assumes *A* ≠ {} *subset-mset.bdd-above A*
shows *count* (*Sup A*) *x* = (*SUP X*:*A*. *count X x*)
⟨*proof*⟩

interpretation *subset-mset*: *conditionally-complete-lattice Inf Sup op* ∩# *op* ⊆# *op* ⊂# *op* ∪#
⟨*proof*⟩

lemma *set-mset-Inf*:
assumes *A* ≠ {}
shows *set-mset* (*Inf A*) = (⋂ *X*∈*A*. *set-mset X*)
⟨*proof*⟩

lemma *in-Inf-multiset-iff*:
assumes *A* ≠ {}
shows *x* ∈# *Inf A* ⟷ (∀ *X*∈*A*. *x* ∈# *X*)
⟨*proof*⟩

lemma *in-Inf-multisetD*: *x* ∈# *Inf A* ⟹ *X* ∈ *A* ⟹ *x* ∈# *X*
⟨*proof*⟩

lemma *set-mset-Sup*:
assumes *subset-mset.bdd-above A*
shows *set-mset* (*Sup A*) = (⋃ *X*∈*A*. *set-mset X*)
⟨*proof*⟩

lemma *in-Sup-multiset-iff*:
assumes *subset-mset.bdd-above A*
shows *x* ∈# *Sup A* ⟷ (∃ *X*∈*A*. *x* ∈# *X*)
⟨*proof*⟩

lemma *in-Sup-multisetD*:
assumes *x* ∈# *Sup A*
shows ∃ *X*∈*A*. *x* ∈# *X*

⟨*proof*⟩

interpretation *subset-mset*: *distrib-lattice op* ∩# *op* ⊆# *op* ⊂# *op* ∪#
⟨*proof*⟩

1.4.3 Filter (with comprehension syntax)

Multiset comprehension

lift-definition *filter-mset* :: ('*a* ⇒ *bool*) ⇒ '*a multiset* ⇒ '*a multiset*
is λ*P M*. λ*x*. *if P x then M x else 0*
⟨*proof*⟩

syntax (*ASCII*)
-MCollect :: *pttrn* ⇒ '*a multiset* ⇒ *bool* ⇒ '*a multiset* (*(1{#- :# -./ -#})*)
syntax
-MCollect :: *pttrn* ⇒ '*a multiset* ⇒ *bool* ⇒ '*a multiset* (*(1{#- ∈# -./ -#})*)
translations
{#*x* ∈# *M*. *P*#} == *CONST filter-mset* (λ*x*. *P*) *M*

lemma *count-filter-mset* [*simp*]:
count (*filter-mset P M*) *a* = (*if P a then count M a else 0*)
⟨*proof*⟩

lemma *set-mset-filter* [*simp*]:
set-mset (*filter-mset P M*) = {*a* ∈ *set-mset M*. *P a*}
⟨*proof*⟩

lemma *filter-empty-mset* [*simp*]: *filter-mset P* {#} = {#}
⟨*proof*⟩

lemma *filter-single-mset*: *filter-mset P* {#*x*#} = (*if P x then* {#*x*#} *else* {#})
⟨*proof*⟩

lemma *filter-union-mset* [*simp*]: *filter-mset P* (*M* + *N*) = *filter-mset P M* + *filter-mset P N*
⟨*proof*⟩

lemma *filter-diff-mset* [*simp*]: *filter-mset P* (*M* − *N*) = *filter-mset P M* − *filter-mset P N*
⟨*proof*⟩

lemma *filter-inter-mset* [*simp*]: *filter-mset P* (*M* ∩# *N*) = *filter-mset P M* ∩# *filter-mset P N*
⟨*proof*⟩

lemma *filter-sup-mset*[*simp*]: *filter-mset P* (*A* ∪# *B*) = *filter-mset P A* ∪# *filter-mset P B*
⟨*proof*⟩

lemma *filter-mset-add-mset* [*simp*]:
filter-mset P (*add-mset x A*) =
(*if P x then add-mset x* (*filter-mset P A*) *else filter-mset P A*)
⟨*proof*⟩

lemma *multiset-filter-subset*[*simp*]: *filter-mset f M* ⊆# *M*
⟨*proof*⟩

lemma *multiset-filter-mono*:
assumes *A* ⊆# *B*
shows *filter-mset f A* ⊆# *filter-mset f B*
⟨*proof*⟩

lemma *filter-mset-eq-conv*:
filter-mset P M = *N* ⟷ *N* ⊆# *M* ∧ (∀ *b*∈#*N*. *P b*) ∧ (∀ *a*∈#*M* − *N*. ¬ *P a*)
(**is** *?P* ⟷ *?Q*)
⟨*proof*⟩

lemma *filter-filter-mset*: *filter-mset P* (*filter-mset Q M*) = {#*x* ∈# *M*. *Q x* ∧ *P x*#}
⟨*proof*⟩

lemma
filter-mset-True[*simp*]: {#*y* ∈# *M*. *True*#} = *M* **and**
filter-mset-False[*simp*]: {#*y* ∈# *M*. *False*#} = {#}
⟨*proof*⟩

1.4.4 Size

definition *wcount* **where** *wcount f M* = (λ*x*. *count M x* ∗ *Suc* (*f x*))

lemma *wcount-union*: *wcount f* (*M* + *N*) *a* = *wcount f M a* + *wcount f N a*
⟨*proof*⟩

lemma *wcount-add-mset*:
wcount f (*add-mset x M*) *a* = (*if x* = *a then Suc* (*f a*) *else 0*) + *wcount f M a*
⟨*proof*⟩

definition *size-multiset* :: ('*a* ⇒ *nat*) ⇒ '*a multiset* ⇒ *nat* **where**
size-multiset f M = *sum* (*wcount f M*) (*set-mset M*)

lemmas *size-multiset-eq* = *size-multiset-def*[*unfolded wcount-def*]

instantiation *multiset* :: (*type*) *size*
begin

definition *size-multiset* **where**
size-multiset-overloaded-def: *size-multiset* = *Multiset.size-multiset* (λ-. *0*)
instance ⟨*proof*⟩

end

lemmas *size-multiset-overloaded-eq* =
size-multiset-overloaded-def[*THEN fun-cong*, *unfolded size-multiset-eq*, *simplified*]

lemma *size-multiset-empty* [*simp*]: *size-multiset f* {#} = *0*
⟨*proof*⟩

lemma *size-empty* [*simp*]: *size* {#} = *0*
⟨*proof*⟩

lemma *size-multiset-single* : *size-multiset f* {#*b*#} = *Suc* (*f b*)
⟨*proof*⟩

lemma *size-single*: *size* {#*b*#} = *1*
⟨*proof*⟩

lemma *sum-wcount-Int*:
finite A ⟹ *sum* (*wcount f N*) (*A* ∩ *set-mset N*) = *sum* (*wcount f N*) *A*
⟨*proof*⟩

lemma *size-multiset-union* [*simp*]:
size-multiset f (*M* + *N*::'*a multiset*) = *size-multiset f M* + *size-multiset f N*
⟨*proof*⟩

lemma *size-multiset-add-mset* [*simp*]:
size-multiset f (*add-mset a M*) = *Suc* (*f a*) + *size-multiset f M*
⟨*proof*⟩

lemma *size-add-mset* [*simp*]: *size* (*add-mset a A*) = *Suc* (*size A*)
⟨*proof*⟩

lemma *size-union* [*simp*]: *size* (*M* + *N*::'*a multiset*) = *size M* + *size N*
⟨*proof*⟩

lemma *size-multiset-eq-0-iff-empty* [*iff*]:
size-multiset f M = *0* ⟷ *M* = {#}
⟨*proof*⟩

lemma *size-eq-0-iff-empty* [*iff*]: (*size M* = *0*) = (*M* = {#})
⟨*proof*⟩

lemma *nonempty-has-size*: (*S* ≠ {#}) = (*0* < *size S*)
⟨*proof*⟩

lemma *size-eq-Suc-imp-elem*: *size M* = *Suc n* ⟹ ∃ *a*. *a* ∈# *M*
⟨*proof*⟩

lemma *size-eq-Suc-imp-eq-union*:
assumes *size M = Suc n*
shows $\exists a\ N.\ M = add\text{-}mset\ a\ N$
⟨*proof*⟩

lemma *size-mset-mono*:
fixes *A B :: 'a multiset*
assumes $A \subseteq\# B$
shows *size A ≤ size B*
⟨*proof*⟩

lemma *size-filter-mset-lesseq*[*simp*]: *size* (*filter-mset f M*) ≤ *size M*
⟨*proof*⟩

lemma *size-Diff-submset*:
$M \subseteq\# M' \Longrightarrow size\ (M' - M) = size\ M' - size(M::'a\ multiset)$
⟨*proof*⟩

1.5 Induction and case splits

theorem *multiset-induct* [*case-names empty add*, *induct type*: *multiset*]:
assumes *empty*: *P* {#}
assumes *add*: $\bigwedge x\ M.\ P\ M \Longrightarrow P\ (add\text{-}mset\ x\ M)$
shows *P M*
⟨*proof*⟩

lemma *multi-nonempty-split*: $M \neq \{\#\} \Longrightarrow \exists A\ a.\ M = add\text{-}mset\ a\ A$
⟨*proof*⟩

lemma *multiset-cases* [*cases type*]:
obtains (*empty*) *M* = {#}
| (*add*) *x N* **where** *M* = *add-mset x N*
⟨*proof*⟩

lemma *multi-drop-mem-not-eq*: $c \in\# B \Longrightarrow B - \{\#c\#\} \neq B$
⟨*proof*⟩

lemma *multiset-partition*: $M = \{\#\ x{\in}\#M.\ P\ x\ \#\} + \{\#\ x{\in}\#M.\ \neg\ P\ x\ \#\}$
⟨*proof*⟩

lemma *mset-subset-size*: $(A::'a\ multiset) \subset\# B \Longrightarrow size\ A < size\ B$
⟨*proof*⟩

lemma *size-1-singleton-mset*: *size M = 1* $\Longrightarrow \exists a.\ M = \{\#a\#\}$
⟨*proof*⟩

1.5.1 Strong induction and subset induction for multisets

Well-foundedness of strict subset relation

lemma *wf-subset-mset-rel*: *wf* {(*M*, *N* :: *'a multiset*). *M* ⊂# *N*}
⟨*proof*⟩

lemma *full-multiset-induct* [*case-names less*]:
assumes *ih*: ⋀*B*. ∀(*A*::*'a multiset*). *A* ⊂# *B* ⟶ *P A* ⟹ *P B*
shows *P B*
⟨*proof*⟩

lemma *multi-subset-induct* [*consumes 2*, *case-names empty add*]:
assumes *F* ⊆# *A*
and *empty*: *P* {#}
and *insert*: ⋀*a F*. *a* ∈# *A* ⟹ *P F* ⟹ *P* (*add-mset a F*)
shows *P F*
⟨*proof*⟩

1.6 The fold combinator

definition *fold-mset* :: (*'a* ⇒ *'b* ⇒ *'b*) ⇒ *'b* ⇒ *'a multiset* ⇒ *'b*
where
fold-mset f s M = *Finite-Set.fold* (λ*x*. *f x* ^^ *count M x*) *s* (*set-mset M*)

lemma *fold-mset-empty* [*simp*]: *fold-mset f s* {#} = *s*
⟨*proof*⟩

context *comp-fun-commute*
begin

lemma *fold-mset-add-mset* [*simp*]: *fold-mset f s* (*add-mset x M*) = *f x* (*fold-mset f s M*)
⟨*proof*⟩

corollary *fold-mset-single*: *fold-mset f s* {#*x*#} = *f x s*
⟨*proof*⟩

lemma *fold-mset-fun-left-comm*: *f x* (*fold-mset f s M*) = *fold-mset f* (*f x s*) *M*
⟨*proof*⟩

lemma *fold-mset-union* [*simp*]: *fold-mset f s* (*M* + *N*) = *fold-mset f* (*fold-mset f s M*) *N*
⟨*proof*⟩

lemma *fold-mset-fusion*:
assumes *comp-fun-commute g*
and ∗: ⋀*x y*. *h* (*g x y*) = *f x* (*h y*)
shows *h* (*fold-mset g w A*) = *fold-mset f* (*h w*) *A*
⟨*proof*⟩

end

lemma *union-fold-mset-add-mset*: $A + B = \textit{fold-mset add-mset } A\ B$
⟨*proof*⟩

A note on code generation: When defining some function containing a subterm *fold-mset F*, code generation is not automatic. When interpreting locale *left-commutative* with *F*, the would be code thms for *fold-mset* become thms like $\textit{fold-mset } F\ z\ \{\#\} = z$ where *F* is not a pattern but contains defined symbols, i.e. is not a code thm. Hence a separate constant with its own code thms needs to be introduced for *F*. See the image operator below.

1.7 Image

definition *image-mset* :: $('a \Rightarrow 'b) \Rightarrow 'a\ multiset \Rightarrow 'b\ multiset$ **where**
$\textit{image-mset } f = \textit{fold-mset } (\textit{add-mset} \circ f)\ \{\#\}$

lemma *comp-fun-commute-mset-image*: *comp-fun-commute* $(\textit{add-mset} \circ f)$
⟨*proof*⟩

lemma *image-mset-empty* [*simp*]: $\textit{image-mset } f\ \{\#\} = \{\#\}$
⟨*proof*⟩

lemma *image-mset-single*: $\textit{image-mset } f\ \{\#x\#\} = \{\#f\ x\#\}$
⟨*proof*⟩

lemma *image-mset-union* [*simp*]: $\textit{image-mset } f\ (M + N) = \textit{image-mset } f\ M + \textit{image-mset } f\ N$
⟨*proof*⟩

corollary *image-mset-add-mset* [*simp*]:
$\textit{image-mset } f\ (\textit{add-mset } a\ M) = \textit{add-mset } (f\ a)\ (\textit{image-mset } f\ M)$
⟨*proof*⟩

lemma *set-image-mset* [*simp*]: $\textit{set-mset } (\textit{image-mset } f\ M) = \textit{image } f\ (\textit{set-mset } M)$
⟨*proof*⟩

lemma *size-image-mset* [*simp*]: $\textit{size } (\textit{image-mset } f\ M) = \textit{size } M$
⟨*proof*⟩

lemma *image-mset-is-empty-iff* [*simp*]: $\textit{image-mset } f\ M = \{\#\} \longleftrightarrow M = \{\#\}$
⟨*proof*⟩

lemma *image-mset-If*:
$\textit{image-mset } (\lambda x.\ \textit{if } P\ x\ \textit{then } f\ x\ \textit{else } g\ x)\ A =$
$\quad \textit{image-mset } f\ (\textit{filter-mset } P\ A) + \textit{image-mset } g\ (\textit{filter-mset } (\lambda x.\ \neg P\ x)\ A)$
⟨*proof*⟩

lemma *image-mset-Diff*:
assumes $B \subseteq\# A$

shows *image-mset f (A − B) = image-mset f A − image-mset f B*
⟨*proof*⟩

lemma *count-image-mset*:
count (image-mset f A) x = (∑ y∈f −‘ {x} ∩ set-mset A. count A y)
⟨*proof*⟩

lemma *image-mset-subseteq-mono*: *A ⊆# B ⟹ image-mset f A ⊆# image-mset f B*
⟨*proof*⟩

syntax (*ASCII*)
-comprehension-mset :: *'a ⇒ 'b ⇒ 'b multiset ⇒ 'a multiset* (({#-/. - :# -#}))
syntax
-comprehension-mset :: *'a ⇒ 'b ⇒ 'b multiset ⇒ 'a multiset* (({#-/. - ∈# -#}))
translations
{#*e*. *x* ∈# *M*#} ⇌ *CONST image-mset* (λ*x*. *e*) *M*

syntax (*ASCII*)
-comprehension-mset' :: *'a ⇒ 'b ⇒ 'b multiset ⇒ bool ⇒ 'a multiset* (({#-/ | -:# -./ -#}))
syntax
-comprehension-mset' :: *'a ⇒ 'b ⇒ 'b multiset ⇒ bool ⇒ 'a multiset* (({#-/ | -∈# -./ -#}))
translations
{#*e* | *x*∈#*M*. *P*#} ↽ {#*e*. *x* ∈# {# *x*∈#*M*. *P*#}#}

This allows to write not just filters like {#*x* ∈# *M*. *x* < *c*#} but also images like {#*x* + *x*. *x* ∈# *M*#} and {#*x*+*x*|*x*∈#*M*. *x*<*c*#}, where the latter is currently displayed as {#*x* + *x*. *x* ∈# {#*x* ∈# *M*. *x* < *c*#}#}.

lemma *in-image-mset*: *y ∈# {#f x. x ∈# M#} ⟷ y ∈ f ‘ set-mset M*
⟨*proof*⟩

functor *image-mset*: *image-mset*
⟨*proof*⟩

declare
image-mset.id [*simp*]
image-mset.identity [*simp*]

lemma *image-mset-id*[*simp*]: *image-mset id x = x*
⟨*proof*⟩

lemma *image-mset-cong*: *(⋀x. x ∈# M ⟹ f x = g x) ⟹ {#f x. x ∈# M#} = {#g x. x ∈# M#}*
⟨*proof*⟩

lemma *image-mset-cong-pair*:
(∀ x y. (x, y) ∈# M ⟶ f x y = g x y) ⟹ {#f x y. (x, y) ∈# M#} = {#g x

$y.\ (x,\ y) \in\#\ M\#\}$
⟨*proof*⟩

1.8 Further conversions

primrec *mset* :: *'a list* ⇒ *'a multiset* **where**
mset [] = {#} |
mset (*a* # *x*) = *add-mset a* (*mset x*)

lemma *in-multiset-in-set*:
x ∈# *mset xs* ⟷ *x* ∈ *set xs*
⟨*proof*⟩

lemma *count-mset*:
count (*mset xs*) *x* = *length* (*filter* (λ*y*. *x* = *y*) *xs*)
⟨*proof*⟩

lemma *mset-zero-iff*[*simp*]: (*mset x* = {#}) = (*x* = [])
⟨*proof*⟩

lemma *mset-zero-iff-right*[*simp*]: ({#} = *mset x*) = (*x* = [])
⟨*proof*⟩

lemma *mset-single-iff*[*iff*]: *mset xs* = {#*x*#} ⟷ *xs* = [*x*]
⟨*proof*⟩

lemma *mset-single-iff-right*[*iff*]: {#*x*#} = *mset xs* ⟷ *xs* = [*x*]
⟨*proof*⟩

lemma *set-mset-mset*[*simp*]: *set-mset* (*mset xs*) = *set xs*
⟨*proof*⟩

lemma *set-mset-comp-mset* [*simp*]: *set-mset* ∘ *mset* = *set*
⟨*proof*⟩

lemma *size-mset* [*simp*]: *size* (*mset xs*) = *length xs*
⟨*proof*⟩

lemma *mset-append* [*simp*]: *mset* (*xs* @ *ys*) = *mset xs* + *mset ys*
⟨*proof*⟩

lemma *mset-filter*: *mset* (*filter P xs*) = {#*x* ∈# *mset xs*. *P x* #}
⟨*proof*⟩

lemma *mset-rev* [*simp*]:
mset (*rev xs*) = *mset xs*
⟨*proof*⟩

lemma *surj-mset*: *surj mset*

⟨*proof*⟩

lemma *distinct-count-atmost-1*:
  distinct x = (∀ *a*. *count* (*mset x*) *a* = (*if a* ∈ *set x then 1 else 0*))
⟨*proof*⟩

lemma *mset-eq-setD*:
  assumes *mset xs* = *mset ys*
  shows *set xs* = *set ys*
⟨*proof*⟩

lemma *set-eq-iff-mset-eq-distinct*:
  distinct x ⟹ *distinct y* ⟹
    (*set x* = *set y*) = (*mset x* = *mset y*)
⟨*proof*⟩

lemma *set-eq-iff-mset-remdups-eq*:
   (*set x* = *set y*) = (*mset* (*remdups x*) = *mset* (*remdups y*))
⟨*proof*⟩

lemma *mset-compl-union* [*simp*]: *mset* [*x*←*xs*. *P x*] + *mset* [*x*←*xs*. ¬*P x*] = *mset xs*
  ⟨*proof*⟩

lemma *nth-mem-mset*: *i* < *length ls* ⟹ (*ls* ! *i*) ∈# *mset ls*
⟨*proof*⟩

lemma *mset-remove1*[*simp*]: *mset* (*remove1 a xs*) = *mset xs* − {#*a*#}
  ⟨*proof*⟩

lemma *mset-eq-length*:
  assumes *mset xs* = *mset ys*
  shows *length xs* = *length ys*
  ⟨*proof*⟩

lemma *mset-eq-length-filter*:
  assumes *mset xs* = *mset ys*
  shows *length* (*filter* (λ*x*. *z* = *x*) *xs*) = *length* (*filter* (λ*y*. *z* = *y*) *ys*)
  ⟨*proof*⟩

lemma *fold-multiset-equiv*:
  assumes *f*: ⋀*x y*. *x* ∈ *set xs* ⟹ *y* ∈ *set xs* ⟹ *f x* ∘ *f y* = *f y* ∘ *f x*
    and *equiv*: *mset xs* = *mset ys*
  shows *List.fold f xs* = *List.fold f ys*
  ⟨*proof*⟩

lemma *mset-insort* [*simp*]: *mset* (*insort x xs*) = *add-mset x* (*mset xs*)
  ⟨*proof*⟩

lemma *mset-map*[*simp*]: *mset* (*map f xs*) = *image-mset f* (*mset xs*)
⟨*proof*⟩

global-interpretation *mset-set*: *folding add-mset* {#}
defines *mset-set* = *folding.F add-mset* {#}
⟨*proof*⟩

lemma *count-mset-set* [*simp*]:
finite A ⟹ *x* ∈ *A* ⟹ *count* (*mset-set A*) *x* = *1* (**is** *PROP ?P*)
¬ *finite A* ⟹ *count* (*mset-set A*) *x* = *0* (**is** *PROP ?Q*)
x ∉ *A* ⟹ *count* (*mset-set A*) *x* = *0* (**is** *PROP ?R*)
⟨*proof*⟩

lemma *elem-mset-set*[*simp*, *intro*]: *finite A* ⟹ *x* ∈# *mset-set A* ⟷ *x* ∈ *A*
⟨*proof*⟩

lemma *mset-set-Union*:
finite A ⟹ *finite B* ⟹ *A* ∩ *B* = {} ⟹ *mset-set* (*A* ∪ *B*) = *mset-set A* + *mset-set B*
⟨*proof*⟩

lemma *filter-mset-mset-set* [*simp*]:
finite A ⟹ *filter-mset P* (*mset-set A*) = *mset-set* {*x*∈*A*. *P x*}
⟨*proof*⟩

lemma *mset-set-Diff*:
assumes *finite A B* ⊆ *A*
shows *mset-set* (*A* − *B*) = *mset-set A* − *mset-set B*
⟨*proof*⟩

lemma *mset-set-set*: *distinct xs* ⟹ *mset-set* (*set xs*) = *mset xs*
⟨*proof*⟩

context *linorder*
begin

definition *sorted-list-of-multiset* :: *'a multiset* ⇒ *'a list*
where
sorted-list-of-multiset M = *fold-mset insort* [] *M*

lemma *sorted-list-of-multiset-empty* [*simp*]:
sorted-list-of-multiset {#} = []
⟨*proof*⟩

lemma *sorted-list-of-multiset-singleton* [*simp*]:
sorted-list-of-multiset {#*x*#} = [*x*]
⟨*proof*⟩

lemma *sorted-list-of-multiset-insert* [*simp*]:

sorted-list-of-multiset (*add-mset x M*) = *List.insert x* (*sorted-list-of-multiset M*)
⟨*proof*⟩

end

lemma *mset-sorted-list-of-multiset* [*simp*]:
mset (*sorted-list-of-multiset M*) = *M*
⟨*proof*⟩

lemma *sorted-list-of-multiset-mset* [*simp*]:
sorted-list-of-multiset (*mset xs*) = *sort xs*
⟨*proof*⟩

lemma *finite-set-mset-mset-set*[*simp*]:
finite A ⟹ *set-mset* (*mset-set A*) = *A*
⟨*proof*⟩

lemma *mset-set-empty-iff*: *mset-set A* = {#} ⟷ *A* = {} ∨ *infinite A*
⟨*proof*⟩

lemma *infinite-set-mset-mset-set*:
¬ *finite A* ⟹ *set-mset* (*mset-set A*) = {}
⟨*proof*⟩

lemma *set-sorted-list-of-multiset* [*simp*]:
set (*sorted-list-of-multiset M*) = *set-mset M*
⟨*proof*⟩

lemma *sorted-list-of-mset-set* [*simp*]:
sorted-list-of-multiset (*mset-set A*) = *sorted-list-of-set A*
⟨*proof*⟩

lemma *mset-upt* [*simp*]: *mset* [*m*..<*n*] = *mset-set* {*m*..<*n*}
⟨*proof*⟩

lemma *image-mset-map-of*:
distinct (*map fst xs*) ⟹ {#*the* (*map-of xs i*). *i* ∈# *mset* (*map fst xs*)#} = *mset*
(*map snd xs*)
⟨*proof*⟩

lemma *image-mset-mset-set*:
assumes *inj-on f A*
shows *image-mset f* (*mset-set A*) = *mset-set* (*f* ‘ *A*)
⟨*proof*⟩

1.9 More properties of the replicate and repeat operations

lemma *in-replicate-mset*[*simp*]: *x* ∈# *replicate-mset n y* ⟷ *n* > *0* ∧ *x* = *y*

⟨*proof*⟩

lemma *set-mset-replicate-mset-subset*[*simp*]: *set-mset* (*replicate-mset n x*) = (*if n = 0 then* {} *else* {*x*})
⟨*proof*⟩

lemma *size-replicate-mset*[*simp*]: *size* (*replicate-mset n M*) = *n*
⟨*proof*⟩

lemma *count-le-replicate-mset-subset-eq*: *n* ≤ *count M x* ⟷ *replicate-mset n x* ⊆# *M*
⟨*proof*⟩

lemma *filter-eq-replicate-mset*: {#*y* ∈# *D*. *y* = *x*#} = *replicate-mset* (*count D x*) *x*
⟨*proof*⟩

lemma *replicate-count-mset-eq-filter-eq*:
replicate (*count* (*mset xs*) *k*) *k* = *filter* (*HOL.eq k*) *xs*
⟨*proof*⟩

lemma *replicate-mset-eq-empty-iff* [*simp*]:
replicate-mset n a = {#} ⟷ *n* = *0*
⟨*proof*⟩

lemma *replicate-mset-eq-iff*:
replicate-mset m a = *replicate-mset n b* ⟷
m = *0* ∧ *n* = *0* ∨ *m* = *n* ∧ *a* = *b*
⟨*proof*⟩

lemma *repeat-mset-cancel1*: *repeat-mset a A* = *repeat-mset a B* ⟷ *A* = *B* ∨ *a* = *0*
⟨*proof*⟩

lemma *repeat-mset-cancel2*: *repeat-mset a A* = *repeat-mset b A* ⟷ *a* = *b* ∨ *A* = {#}
⟨*proof*⟩

lemma *repeat-mset-eq-empty-iff*: *repeat-mset n A* = {#} ⟷ *n* = *0* ∨ *A* = {#}
⟨*proof*⟩

lemma *image-replicate-mset* [*simp*]:
image-mset f (*replicate-mset n a*) = *replicate-mset n* (*f a*)
⟨*proof*⟩

1.10 Big operators

locale *comm-monoid-mset* = *comm-monoid*
begin

interpretation *comp-fun-commute f*
⟨*proof*⟩

interpretation *comp?*: *comp-fun-commute f* ∘ *g*
⟨*proof*⟩

context
begin

definition *F* :: *'a multiset* ⇒ *'a*
where *eq-fold*: *F M* = *fold-mset f* **1** *M*

lemma *empty* [*simp*]: *F* {#} = **1**
⟨*proof*⟩

lemma *singleton* [*simp*]: *F* {#*x*#} = *x*
⟨*proof*⟩

lemma *union* [*simp*]: *F* (*M* + *N*) = *F M* ∗ *F N*
⟨*proof*⟩

lemma *add-mset* [*simp*]: *F* (*add-mset x N*) = *x* ∗ *F N*
⟨*proof*⟩

lemma *insert* [*simp*]:
shows *F* (*image-mset g* (*add-mset x A*)) = *g x* ∗ *F* (*image-mset g A*)
⟨*proof*⟩

lemma *remove*:
assumes *x* ∈# *A*
shows *F A* = *x* ∗ *F* (*A* − {#*x*#})
⟨*proof*⟩

lemma *neutral*:
∀ *x*∈#*A*. *x* = **1** ⟹ *F A* = **1**
⟨*proof*⟩

lemma *neutral-const* [*simp*]:
F (*image-mset* (λ-. **1**) *A*) = **1**
⟨*proof*⟩ **lemma** *F-image-mset-product*:
F {#*g x j* ∗ *F* {#*g i j*. *i* ∈# *A*#}. *j* ∈# *B*#} =
F (*image-mset* (*g x*) *B*) ∗ *F* {#*F* {#*g i j*. *i* ∈# *A*#}. *j* ∈# *B*#}
⟨*proof*⟩

lemma *commute*:
F (*image-mset* (λ*i*. *F* (*image-mset* (*g i*) *B*)) *A*) =
F (*image-mset* (λ*j*. *F* (*image-mset* (λ*i*. *g i j*) *A*)) *B*)
⟨*proof*⟩

lemma *distrib*: $F\ (\mathit{image\text{-}mset}\ (\lambda x.\ g\ x * h\ x)\ A) = F\ (\mathit{image\text{-}mset}\ g\ A) * F\ (\mathit{image\text{-}mset}\ h\ A)$
⟨*proof*⟩

lemma *union-disjoint*:
$A \cap\# B = \{\#\} \Longrightarrow F\ (\mathit{image\text{-}mset}\ g\ (A \cup\# B)) = F\ (\mathit{image\text{-}mset}\ g\ A) * F\ (\mathit{image\text{-}mset}\ g\ B)$
⟨*proof*⟩

end
end

lemma *comp-fun-commute-plus-mset*[*simp*]: *comp-fun-commute* (*op* + :: *'a multi-set* ⇒ - ⇒ -)
⟨*proof*⟩

declare *comp-fun-commute.fold-mset-add-mset*[*OF comp-fun-commute-plus-mset*, *simp*]

lemma *in-mset-fold-plus-iff*[*iff*]: $x \in\# \mathit{fold\text{-}mset}\ (op\ +)\ M\ NN \longleftrightarrow x \in\# M \vee (\exists N.\ N \in\# NN \wedge x \in\# N)$
⟨*proof*⟩

context *comm-monoid-add*
begin

sublocale *sum-mset*: *comm-monoid-mset plus 0*
defines *sum-mset* = *sum-mset.F* ⟨*proof*⟩

lemma (**in** *semiring-1*) *sum-mset-replicate-mset* [*simp*]:
sum-mset (*replicate-mset n a*) = *of-nat n* ∗ *a*
⟨*proof*⟩

lemma *sum-unfold-sum-mset*:
sum f A = *sum-mset* (*image-mset f* (*mset-set A*))
⟨*proof*⟩

lemma *sum-mset-delta*: *sum-mset* (*image-mset* (λ*x*. *if x* = *y then c else 0*) *A*) = *c* ∗ *count A y*
⟨*proof*⟩

lemma *sum-mset-delta'*: *sum-mset* (*image-mset* (λ*x*. *if y* = *x then c else 0*) *A*) = *c* ∗ *count A y*
⟨*proof*⟩

end

lemma *of-nat-sum-mset* [*simp*]:

of-nat (*sum-mset M*) = *sum-mset* (*image-mset of-nat M*)
⟨*proof*⟩

lemma *sum-mset-0-iff* [*simp*]:
sum-mset M = (*0*::*'a*::*canonically-ordered-monoid-add*)
⟷ (∀ *x* ∈ *set-mset M*. *x* = *0*)
⟨*proof*⟩

lemma *sum-mset-diff*:
fixes *M N* :: (*'a* :: *ordered-cancel-comm-monoid-diff*) *multiset*
shows *N* ⊆# *M* ⟹ *sum-mset* (*M* − *N*) = *sum-mset M* − *sum-mset N*
⟨*proof*⟩

lemma *size-eq-sum-mset*: *size M* = *sum-mset* (*image-mset* (λ-. *1*) *M*)
⟨*proof*⟩

lemma *size-mset-set* [*simp*]: *size* (*mset-set A*) = *card A*
⟨*proof*⟩

lemma *sum-mset-sum-list*: *sum-mset* (*mset xs*) = *sum-list xs*
⟨*proof*⟩

syntax (*ASCII*)
-sum-mset-image :: *pttrn* ⇒ *'b set* ⇒ *'a* ⇒ *'a*::*comm-monoid-add* ((*3SUM -*:#*-.*
-) [*0*, *51*, *10*] *10*)
syntax
-sum-mset-image :: *pttrn* ⇒ *'b set* ⇒ *'a* ⇒ *'a*::*comm-monoid-add* ((*3*∑ *-*∈#*-.*
-) [*0*, *51*, *10*] *10*)
translations
∑ *i* ∈# *A*. *b* ⇌ *CONST sum-mset* (*CONST image-mset* (λ*i*. *b*) *A*)

lemma *sum-mset-distrib-left*:
fixes *f* :: *'a* ⇒ *'b*::*semiring-0*
shows *c* ∗ (∑ *x* ∈# *M*. *f x*) = (∑ *x* ∈# *M*. *c* ∗ *f*(*x*))
⟨*proof*⟩

lemma *sum-mset-distrib-right*:
fixes *f* :: *'a* ⇒ *'b*::*semiring-0*
shows (∑ *b* ∈# *B*. *f b*) ∗ *a* = (∑ *b* ∈# *B*. *f b* ∗ *a*)
⟨*proof*⟩

lemma *sum-mset-constant* [*simp*]:
fixes *y* :: *'b*::*semiring-1*
shows ‹(∑ *x*∈#*A*. *y*) = *of-nat* (*size A*) ∗ *y*›
⟨*proof*⟩

lemma (**in** *ordered-comm-monoid-add*) *sum-mset-mono*:
assumes ⋀*i*. *i* ∈# *K* ⟹ *f i* ≤ *g i*
shows *sum-mset* (*image-mset f K*) ≤ *sum-mset* (*image-mset g K*)

⟨*proof*⟩

lemma *sum-mset-product*:
fixes f :: *'a*::{*comm-monoid-add*,*times*} ⇒ *'b*::*semiring-0*
shows $(\sum i \in\# A.\ f\ i) * (\sum i \in\# B.\ g\ i) = (\sum i{\in}\#A.\ \sum j{\in}\#B.\ f\ i * g\ j)$
⟨*proof*⟩

abbreviation *Union-mset* :: *'a multiset multiset* ⇒ *'a multiset* $(\bigcup\#\text{-}\ [900]\ 900)$
where $\bigcup\#\ MM \equiv$ *sum-mset MM* — FIXME ambiguous notation – could likewise refer to $\bigsqcup\#$

lemma *set-mset-Union-mset*[*simp*]: *set-mset* $(\bigcup\#\ MM) = (\bigcup M \in$ *set-mset MM*. *set-mset M*)
⟨*proof*⟩

lemma *in-Union-mset-iff*[*iff*]: $x \in\# \bigcup\#\ MM \longleftrightarrow (\exists M.\ M \in\# MM \wedge x \in\# M)$
⟨*proof*⟩

lemma *count-sum*:
count (*sum f A*) *x* = *sum* (λ*a*. *count* (*f a*) *x*) *A*
⟨*proof*⟩

lemma *sum-eq-empty-iff*:
assumes *finite A*
shows *sum f A* = {#} $\longleftrightarrow (\forall a{\in}A.\ f\ a = \{\#\})$
⟨*proof*⟩

lemma *Union-mset-empty-conv*[*simp*]: $\bigcup\#\ M = \{\#\} \longleftrightarrow (\forall i{\in}\#M.\ i = \{\#\})$
⟨*proof*⟩

context *comm-monoid-mult*
begin

sublocale *prod-mset*: *comm-monoid-mset times 1*
defines *prod-mset* = *prod-mset.F* ⟨*proof*⟩

lemma *prod-mset-empty*:
prod-mset {#} = *1*
⟨*proof*⟩

lemma *prod-mset-singleton*:
prod-mset {#*x*#} = *x*
⟨*proof*⟩

lemma *prod-mset-Un*:
prod-mset (*A* + *B*) = *prod-mset A* ∗ *prod-mset B*
⟨*proof*⟩

lemma *prod-mset-replicate-mset* [*simp*]:
prod-mset (*replicate-mset n a*) = *a* ^ *n*
⟨*proof*⟩

lemma *prod-unfold-prod-mset*:
prod f A = *prod-mset* (*image-mset f* (*mset-set A*))
⟨*proof*⟩

lemma *prod-mset-multiplicity*:
prod-mset M = *prod* (λ*x*. *x* ^ *count M x*) (*set-mset M*)
⟨*proof*⟩

lemma *prod-mset-delta*: *prod-mset* (*image-mset* (λ*x*. *if x* = *y then c else 1*) *A*) = *c* ^ *count A y*
⟨*proof*⟩

lemma *prod-mset-delta'*: *prod-mset* (*image-mset* (λ*x*. *if y* = *x then c else 1*) *A*) = *c* ^ *count A y*
⟨*proof*⟩

end

syntax (*ASCII*)
-prod-mset-image :: *pttrn* ⇒ *'b set* ⇒ *'a* ⇒ *'a*::*comm-monoid-mult* ((*3PROD* -:#-. -) [*0*, *51*, *10*] *10*)
syntax
-prod-mset-image :: *pttrn* ⇒ *'b set* ⇒ *'a* ⇒ *'a*::*comm-monoid-mult* ((*3*∏ -∈#-. -) [*0*, *51*, *10*] *10*)
translations
∏*i* ∈# *A*. *b* ⇌ *CONST prod-mset* (*CONST image-mset* (λ*i*. *b*) *A*)

lemma (**in** *comm-monoid-mult*) *prod-mset-subset-imp-dvd*:
assumes *A* ⊆# *B*
shows *prod-mset A dvd prod-mset B*
⟨*proof*⟩

lemma (**in** *comm-monoid-mult*) *dvd-prod-mset*:
assumes *x* ∈# *A*
shows *x dvd prod-mset A*
⟨*proof*⟩

lemma (**in** *semidom*) *prod-mset-zero-iff* [*iff*]:
prod-mset A = *0* ⟷ *0* ∈# *A*
⟨*proof*⟩

lemma (**in** *semidom-divide*) *prod-mset-diff*:
assumes *B* ⊆# *A* **and** *0* ∉# *B*
shows *prod-mset* (*A* − *B*) = *prod-mset A div prod-mset B*
⟨*proof*⟩

lemma (**in** *semidom-divide*) *prod-mset-minus*:
assumes $a \in\# A$ **and** $a \neq 0$
shows *prod-mset* $(A - \{\#a\#\}) =$ *prod-mset* A *div* a
⟨*proof*⟩

lemma (**in** *algebraic-semidom*) *is-unit-prod-mset-iff*:
is-unit (*prod-mset* A) $\longleftrightarrow$ $(\forall x \in\# A.$ *is-unit* $x)$
⟨*proof*⟩

lemma (**in** *normalization-semidom*) *normalize-prod-mset*:
normalize (*prod-mset* A) = *prod-mset* (*image-mset normalize* A)
⟨*proof*⟩

lemma (**in** *normalization-semidom*) *normalized-prod-msetI*:
assumes $\bigwedge a.\ a \in\# A \Longrightarrow$ *normalize* $a = a$
shows *normalize* (*prod-mset* A) = *prod-mset* A
⟨*proof*⟩

lemma *prod-mset-prod-list*: *prod-mset* (*mset xs*) = *prod-list xs*
⟨*proof*⟩

1.11 Alternative representations

1.11.1 Lists

context *linorder*
begin

lemma *mset-insort* [*simp*]:
mset (*insort-key k x xs*) = *add-mset x* (*mset xs*)
⟨*proof*⟩

lemma *mset-sort* [*simp*]:
mset (*sort-key k xs*) = *mset xs*
⟨*proof*⟩

This lemma shows which properties suffice to show that a function f with f $xs = ys$ behaves like sort.

lemma *properties-for-sort-key*:
assumes *mset ys* = *mset xs*
and $\bigwedge k.\ k \in$ *set ys* $\Longrightarrow$ *filter* $(\lambda x.\ f\ k = f\ x)$ *ys* = *filter* $(\lambda x.\ f\ k = f\ x)$ *xs*
and *sorted* (*map f ys*)
shows *sort-key f xs* = *ys*
⟨*proof*⟩

lemma *properties-for-sort*:
assumes *multiset*: *mset ys* = *mset xs*
and *sorted ys*
shows *sort xs* = *ys*

⟨*proof*⟩

lemma *sort-key-inj-key-eq*:
assumes *mset-equal*: *mset xs* = *mset ys*
and *inj-on f* (*set xs*)
and *sorted* (*map f ys*)
shows *sort-key f xs* = *ys*
⟨*proof*⟩

lemma *sort-key-eq-sort-key*:
assumes *mset xs* = *mset ys*
and *inj-on f* (*set xs*)
shows *sort-key f xs* = *sort-key f ys*
⟨*proof*⟩

lemma *sort-key-by-quicksort*:
sort-key f xs = *sort-key f* [*x*←*xs*. *f x* < *f* (*xs* ! (*length xs div 2*))]
@ [*x*←*xs*. *f x* = *f* (*xs* ! (*length xs div 2*))]
@ *sort-key f* [*x*←*xs*. *f x* > *f* (*xs* ! (*length xs div 2*))] (**is** *sort-key f ?lhs* = *?rhs*)
⟨*proof*⟩

lemma *sort-by-quicksort*:
sort xs = *sort* [*x*←*xs*. *x* < *xs* ! (*length xs div 2*)]
@ [*x*←*xs*. *x* = *xs* ! (*length xs div 2*)]
@ *sort* [*x*←*xs*. *x* > *xs* ! (*length xs div 2*)] (**is** *sort ?lhs* = *?rhs*)
⟨*proof*⟩

A stable parametrized quicksort

definition *part* :: ('*b* ⇒ '*a*) ⇒ '*a* ⇒ '*b list* ⇒ '*b list* × '*b list* × '*b list* **where**
part f pivot xs = ([*x* ← *xs*. *f x* < *pivot*], [*x* ← *xs*. *f x* = *pivot*], [*x* ← *xs*. *pivot* < *f x*])

lemma *part-code* [*code*]:
part f pivot [] = ([], [], [])
part f pivot (*x* # *xs*) = (*let* (*lts*, *eqs*, *gts*) = *part f pivot xs*; *x'* = *f x in*
if x' < *pivot then* (*x* # *lts*, *eqs*, *gts*)
else if x' > *pivot then* (*lts*, *eqs*, *x* # *gts*)
else (*lts*, *x* # *eqs*, *gts*))
⟨*proof*⟩

lemma *sort-key-by-quicksort-code* [*code*]:
sort-key f xs =
(*case xs of*
[] ⇒ []
| [*x*] ⇒ *xs*
| [*x*, *y*] ⇒ (*if f x* ≤ *f y then xs else* [*y*, *x*])
| - ⇒
let (*lts*, *eqs*, *gts*) = *part f* (*f* (*xs* ! (*length xs div 2*))) *xs*
in sort-key f lts @ *eqs* @ *sort-key f gts*)

⟨*proof*⟩

end

hide-const (**open**) *part*

lemma *mset-remdups-subset-eq*: *mset* (*remdups xs*) ⊆# *mset xs*
⟨*proof*⟩

lemma *mset-update*:
i < *length ls* ⟹ *mset* (*ls*[*i* := *v*]) = *add-mset v* (*mset ls* − {#*ls* ! *i*#})
⟨*proof*⟩

lemma *mset-swap*:
i < *length ls* ⟹ *j* < *length ls* ⟹
mset (*ls*[*j* := *ls* ! *i*, *i* := *ls* ! *j*]) = *mset ls*
⟨*proof*⟩

1.12 The multiset order

1.12.1 Well-foundedness

definition *mult1* :: ('*a* × '*a*) *set* ⇒ ('*a multiset* × '*a multiset*) *set* **where**
mult1 r = {(*N*, *M*). ∃ *a M0 K*. *M* = *add-mset a M0* ∧ *N* = *M0* + *K* ∧
(∀ *b*. *b* ∈# *K* ⟶ (*b*, *a*) ∈ *r*)}

definition *mult* :: ('*a* × '*a*) *set* ⇒ ('*a multiset* × '*a multiset*) *set* **where**
mult r = (*mult1 r*)$^+$

lemma *mult1I*:
assumes *M* = *add-mset a M0* **and** *N* = *M0* + *K* **and** ⋀*b*. *b* ∈# *K* ⟹ (*b*, *a*)
∈ *r*
shows (*N*, *M*) ∈ *mult1 r*
⟨*proof*⟩

lemma *mult1E*:
assumes (*N*, *M*) ∈ *mult1 r*
obtains *a M0 K* **where** *M* = *add-mset a M0 N* = *M0* + *K* ⋀*b*. *b* ∈# *K* ⟹
(*b*, *a*) ∈ *r*
⟨*proof*⟩

lemma *mono-mult1*:
assumes *r* ⊆ *r'* **shows** *mult1 r* ⊆ *mult1 r'*
⟨*proof*⟩

lemma *mono-mult*:
assumes *r* ⊆ *r'* **shows** *mult r* ⊆ *mult r'*
⟨*proof*⟩

lemma *not-less-empty* [*iff*]: (*M*, {#}) ∉ *mult1 r*

⟨*proof*⟩

lemma *less-add*:
assumes *mult1*: $(N,\ \textit{add-mset}\ a\ M0) \in \textit{mult1}\ r$
shows
$(\exists M.\ (M,\ M0) \in \textit{mult1}\ r \wedge N = \textit{add-mset}\ a\ M) \vee$
$(\exists K.\ (\forall b.\ b \in\# K \longrightarrow (b,\ a) \in r) \wedge N = M0 + K)$
⟨*proof*⟩

lemma *all-accessible*:
assumes *wf r*
shows $\forall M.\ M \in \textit{Wellfounded.acc}\ (\textit{mult1}\ r)$
⟨*proof*⟩

theorem *wf-mult1*: $\textit{wf}\ r \Longrightarrow \textit{wf}\ (\textit{mult1}\ r)$
⟨*proof*⟩

theorem *wf-mult*: $\textit{wf}\ r \Longrightarrow \textit{wf}\ (\textit{mult}\ r)$
⟨*proof*⟩

1.12.2 Closure-free presentation

One direction.

lemma *mult-implies-one-step*:
assumes
trans: *trans r* **and**
MN: $(M,\ N) \in \textit{mult}\ r$
shows $\exists I\ J\ K.\ N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge (\forall k \in \textit{set-mset}\ K.\ \exists j \in \textit{set-mset}\ J.\ (k,\ j) \in r)$
⟨*proof*⟩

lemma *one-step-implies-mult*:
assumes
$J \neq \{\#\}$ **and**
$\forall k \in \textit{set-mset}\ K.\ \exists j \in \textit{set-mset}\ J.\ (k,\ j) \in r$
shows $(I + K,\ I + J) \in \textit{mult}\ r$
⟨*proof*⟩

1.13 The multiset extension is cancellative for multiset union

lemma *mult-cancel*:
assumes *trans s* **and** *irrefl s*
shows $(X + Z,\ Y + Z) \in \textit{mult}\ s \longleftrightarrow (X,\ Y) \in \textit{mult}\ s$ (**is** *?L* $\longleftrightarrow$ *?R*)
⟨*proof*⟩

lemma *mult-cancel-max*:
assumes *trans s* **and** *irrefl s*
shows $(X,\ Y) \in \textit{mult}\ s \longleftrightarrow (X - X \cap\# Y,\ Y - X \cap\# Y) \in \textit{mult}\ s$ (**is** *?L* $\longleftrightarrow$ *?R*)

⟨*proof*⟩

1.14 Quasi-executable version of the multiset extension

Predicate variants of *mult* and the reflexive closure of *mult*, which are executable whenever the given predicate P is. Together with the standard code equations for *op* $\cap\#$ and *op* $-$ this should yield quadratic (with respect to calls to P) implementations of *multp* and *multeqp*.

definition *multp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ **where**
multp P N M =
(*let Z* = *M* $\cap\#$ *N*; *X* = *M* − *Z in*
X ≠ {#} ∧ (*let Y* = *N* − *Z in* (∀ *y* ∈ *set-mset Y*. ∃ *x* ∈ *set-mset X*. *P y x*)))

definition *multeqp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ **where**
multeqp P N M =
(*let Z* = *M* $\cap\#$ *N*; *X* = *M* − *Z*; *Y* = *N* − *Z in*
(∀ *y* ∈ *set-mset Y*. ∃ *x* ∈ *set-mset X*. *P y x*))

lemma *multp-iff*:
assumes *irrefl R* **and** *trans R* **and** [*simp*]: $\bigwedge x\ y.\ P\ x\ y \longleftrightarrow (x, y) \in R$
shows *multp P N M* ⟷ (*N*, *M*) ∈ *mult R* (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

lemma *multeqp-iff*:
assumes *irrefl R* **and** *trans R* **and** $\bigwedge x\ y.\ P\ x\ y \longleftrightarrow (x, y) \in R$
shows *multeqp P N M* ⟷ $(N, M) \in (mult\ R)^=$
⟨*proof*⟩

1.14.1 Partial-order properties

lemma (**in** *preorder*) *mult1-lessE*:
assumes (*N*, *M*) ∈ *mult1* {(*a*, *b*). *a* < *b*}
obtains *a M0 K* **where** *M* = *add-mset a M0 N* = *M0* + *K*
a ∉# *K* $\bigwedge b.\ b \in\# K \Longrightarrow b < a$
⟨*proof*⟩

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

definition *less-multiset* :: *'a multiset* ⇒ *'a multiset* ⇒ *bool*
where *M'* < *M* ⟷ (*M'*, *M*) ∈ *mult* {(*x'*, *x*). *x'* < *x*}

definition *less-eq-multiset* :: *'a multiset* ⇒ *'a multiset* ⇒ *bool*
where *less-eq-multiset M' M* ⟷ *M'* < *M* ∨ *M'* = *M*

instance
⟨*proof*⟩
end — FIXME avoid junk stemming from type class interpretation

lemma *mset-le-irrefl* [*elim*!]:
fixes M :: *'a::preorder multiset*
shows $M < M \Longrightarrow R$
⟨*proof*⟩

1.14.2 Monotonicity of multiset union

lemma *mult1-union*: $(B, D) \in$ *mult1* $r \Longrightarrow (C + B, C + D) \in$ *mult1* r
⟨*proof*⟩

lemma *union-le-mono2*: $B < D \Longrightarrow C + B < C +$ (D::*'a::preorder multiset*)
⟨*proof*⟩

lemma *union-le-mono1*: $B < D \Longrightarrow B + C < D +$ (C::*'a::preorder multiset*)
⟨*proof*⟩

lemma *union-less-mono*:
fixes A B C D :: *'a::preorder multiset*
shows $A < C \Longrightarrow B < D \Longrightarrow A + B < C + D$
⟨*proof*⟩

instantiation *multiset* :: (*preorder*) *ordered-ab-semigroup-add*
begin
instance
⟨*proof*⟩
end

1.14.3 Termination proofs with multiset orders

lemma *multi-member-skip*: $x \in\# XS \Longrightarrow x \in\# \{\# y \#\} + XS$
and *multi-member-this*: $x \in\# \{\# x \#\} + XS$
and *multi-member-last*: $x \in\# \{\# x \#\}$
⟨*proof*⟩

definition *ms-strict* = *mult pair-less*
definition *ms-weak* = *ms-strict* ∪ *Id*

lemma *ms-reduction-pair*: *reduction-pair* (*ms-strict*, *ms-weak*)
⟨*proof*⟩

lemma *smsI*:
(*set-mset A*, *set-mset B*) ∈ *max-strict* ⟹ (*Z* + *A*, *Z* + *B*) ∈ *ms-strict*
⟨*proof*⟩

lemma *wmsI*:
(*set-mset A*, *set-mset B*) ∈ *max-strict* ∨ *A* = {#} ∧ *B* = {#}
⟹ (*Z* + *A*, *Z* + *B*) ∈ *ms-weak*
⟨*proof*⟩

inductive *pw-leq*

where
pw-leq-empty: *pw-leq* {#} {#}
| *pw-leq-step*: ⟦(*x*,*y*) ∈ *pair-leq*; *pw-leq X Y* ⟧ ⟹ *pw-leq* ({#*x*#} + *X*) ({#*y*#} + *Y*)

lemma *pw-leq-lstep*:
(*x*, *y*) ∈ *pair-leq* ⟹ *pw-leq* {#*x*#} {#*y*#}
⟨*proof*⟩

lemma *pw-leq-split*:
assumes *pw-leq X Y*
shows ∃ *A B Z*. *X* = *A* + *Z* ∧ *Y* = *B* + *Z* ∧ ((*set-mset A*, *set-mset B*) ∈ *max-strict* ∨ (*B* = {#} ∧ *A* = {#}))
⟨*proof*⟩

lemma
assumes *pwleq*: *pw-leq Z Z′*
shows *ms-strictI*: (*set-mset A*, *set-mset B*) ∈ *max-strict* ⟹ (*Z* + *A*, *Z′* + *B*) ∈ *ms-strict*
and *ms-weakI1*: (*set-mset A*, *set-mset B*) ∈ *max-strict* ⟹ (*Z* + *A*, *Z′* + *B*) ∈ *ms-weak*
and *ms-weakI2*: (*Z* + {#}, *Z′* + {#}) ∈ *ms-weak*
⟨*proof*⟩

lemma *empty-neutral*: {#} + *x* = *x x* + {#} = *x*
and *nonempty-plus*: {# *x* #} + *rs* ≠ {#}
and *nonempty-single*: {# *x* #} ≠ {#}
⟨*proof*⟩

⟨*ML*⟩

1.15 Legacy theorem bindings

lemmas *multi-count-eq* = *multiset-eq-iff* [*symmetric*]

lemma *union-commute*: *M* + *N* = *N* + (*M*::′*a multiset*)
⟨*proof*⟩

lemma *union-assoc*: (*M* + *N*) + *K* = *M* + (*N* + (*K*::′*a multiset*))
⟨*proof*⟩

lemma *union-lcomm*: *M* + (*N* + *K*) = *N* + (*M* + (*K*::′*a multiset*))
⟨*proof*⟩

lemmas *union-ac* = *union-assoc union-commute union-lcomm add-mset-commute*

lemma *union-right-cancel*: *M* + *K* = *N* + *K* ⟷ *M* = (*N*::′*a multiset*)
⟨*proof*⟩

lemma *union-left-cancel*: $K + M = K + N \longleftrightarrow M = (N::'a\ multiset)$
⟨*proof*⟩

lemma *multi-union-self-other-eq*: $(A::'a\ multiset) + X = A + Y \Longrightarrow X = Y$
⟨*proof*⟩

lemma *mset-subset-trans*: $(M::'a\ multiset) \subset\# K \Longrightarrow K \subset\# N \Longrightarrow M \subset\# N$
⟨*proof*⟩

lemma *multiset-inter-commute*: $A \cap\# B = B \cap\# A$
⟨*proof*⟩

lemma *multiset-inter-assoc*: $A \cap\# (B \cap\# C) = A \cap\# B \cap\# C$
⟨*proof*⟩

lemma *multiset-inter-left-commute*: $A \cap\# (B \cap\# C) = B \cap\# (A \cap\# C)$
⟨*proof*⟩

lemmas *multiset-inter-ac* =
multiset-inter-commute
multiset-inter-assoc
multiset-inter-left-commute

lemma *mset-le-not-refl*: $\neg M < (M::'a::preorder\ multiset)$
⟨*proof*⟩

lemma *mset-le-trans*: $K < M \Longrightarrow M < N \Longrightarrow K < (N::'a::preorder\ multiset)$
⟨*proof*⟩

lemma *mset-le-not-sym*: $M < N \Longrightarrow \neg N < (M::'a::preorder\ multiset)$
⟨*proof*⟩

lemma *mset-le-asym*: $M < N \Longrightarrow (\neg P \Longrightarrow N < (M::'a::preorder\ multiset)) \Longrightarrow P$
⟨*proof*⟩

⟨*ML*⟩

1.16 Naive implementation using lists

code-datatype *mset*

lemma [*code*]: $\{\#\} = mset\ []$
⟨*proof*⟩

lemma [*code*]: *add-mset x* (*mset xs*) = *mset* (*x # xs*)
⟨*proof*⟩

lemma [*code*]: *Multiset.is-empty* (*mset xs*) ⟷ *List.null xs*

⟨*proof*⟩

lemma *union-code* [*code*]: *mset xs* + *mset ys* = *mset* (*xs* @ *ys*)
⟨*proof*⟩

lemma [*code*]: *image-mset f* (*mset xs*) = *mset* (*map f xs*)
⟨*proof*⟩

lemma [*code*]: *filter-mset f* (*mset xs*) = *mset* (*filter f xs*)
⟨*proof*⟩

lemma [*code*]: *mset xs* − *mset ys* = *mset* (*fold remove1 ys xs*)
⟨*proof*⟩

lemma [*code*]:
mset xs ∩# *mset ys* =
mset (*snd* (*fold* (λ*x* (*ys*, *zs*).
if x ∈ *set ys then* (*remove1 x ys*, *x* # *zs*) *else* (*ys*, *zs*)) *xs* (*ys*, [])))
⟨*proof*⟩

lemma [*code*]:
mset xs ∪# *mset ys* =
mset (*case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, [])))
⟨*proof*⟩

declare *in-multiset-in-set* [*code-unfold*]

lemma [*code*]: *count* (*mset xs*) *x* = *fold* (λ*y*. *if x* = *y then Suc else id*) *xs 0*
⟨*proof*⟩

declare *set-mset-mset* [*code*]

declare *sorted-list-of-multiset-mset* [*code*]

lemma [*code*]: — not very efficient, but representation-ignorant!
mset-set A = *mset* (*sorted-list-of-set A*)
⟨*proof*⟩

declare *size-mset* [*code*]

fun *subset-eq-mset-impl* :: *'a list* ⇒ *'a list* ⇒ *bool option* **where**
subset-eq-mset-impl [] *ys* = *Some* (*ys* ≠ [])
| *subset-eq-mset-impl* (*Cons x xs*) *ys* = (*case List.extract* (*op* = *x*) *ys of*
None ⇒ *None*
| *Some* (*ys1*,-,*ys2*) ⇒ *subset-eq-mset-impl xs* (*ys1* @ *ys2*))

lemma *subset-eq-mset-impl*: (*subset-eq-mset-impl xs ys* = *None* ⟷ ¬ *mset xs* ⊆# *mset ys*) ∧
(*subset-eq-mset-impl xs ys* = *Some True* ⟷ *mset xs* ⊂# *mset ys*) ∧

(*subset-eq-mset-impl xs ys = Some False* ⟶ *mset xs = mset ys*)
⟨*proof*⟩

lemma [*code*]: *mset xs* ⊆# *mset ys* ⟷ *subset-eq-mset-impl xs ys* ≠ *None*
⟨*proof*⟩

lemma [*code*]: *mset xs* ⊂# *mset ys* ⟷ *subset-eq-mset-impl xs ys = Some True*
⟨*proof*⟩

instantiation *multiset* :: (*equal*) *equal*
begin

definition
[*code del*]: *HOL.equal A* (*B* :: *'a multiset*) ⟷ *A = B*
lemma [*code*]: *HOL.equal* (*mset xs*) (*mset ys*) ⟷ *subset-eq-mset-impl xs ys = Some False*
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma [*code*]: *sum-mset* (*mset xs*) = *sum-list xs*
⟨*proof*⟩

lemma [*code*]: *prod-mset* (*mset xs*) = *fold times xs 1*
⟨*proof*⟩

Exercise for the casual reader: add implementations for *op* ≤ and *op* < (multiset order).

Quickcheck generators

definition (**in** *term-syntax*)
msetify :: *'a*::*typerep list* × (*unit* ⇒ *Code-Evaluation.term*)
⇒ *'a multiset* × (*unit* ⇒ *Code-Evaluation.term*) **where**
[*code-unfold*]: *msetify xs = Code-Evaluation.valtermify mset* {·} *xs*

notation *fcomp* (**infixl** ∘> *60*)
notation *scomp* (**infixl** ∘→ *60*)

instantiation *multiset* :: (*random*) *random*
begin

definition
Quickcheck-Random.random i = Quickcheck-Random.random i ∘→ (λ*xs*. *Pair* (*msetify xs*))

instance ⟨*proof*⟩

end

no-notation *fcomp* (**infixl** $\circ>$ *60*)
no-notation *scomp* (**infixl** $\circ\rightarrow$ *60*)

instantiation *multiset* :: (*full-exhaustive*) *full-exhaustive*
begin

definition *full-exhaustive-multiset* :: ('*a multiset* $\times$ (*unit* $\Rightarrow$ *term*) $\Rightarrow$ (*bool* $\times$ *term list*) *option*) $\Rightarrow$ *natural* $\Rightarrow$ (*bool* $\times$ *term list*) *option*
where
full-exhaustive-multiset f i = *Quickcheck-Exhaustive.full-exhaustive* (λ*xs*. *f* (*msetify xs*)) *i*

instance ⟨*proof*⟩

end

hide-const (**open**) *msetify*

1.17 BNF setup

definition *rel-mset* **where**
rel-mset R X Y $\longleftrightarrow$ ($\exists$ *xs ys*. *mset xs* = *X* $\wedge$ *mset ys* = *Y* $\wedge$ *list-all2 R xs ys*)

lemma *mset-zip-take-Cons-drop-twice*:
assumes *length xs* = *length ys j* $\leq$ *length xs*
shows *mset* (*zip* (*take j xs* @ *x* # *drop j xs*) (*take j ys* @ *y* # *drop j ys*)) =
add-mset (*x*,*y*) (*mset* (*zip xs ys*))
⟨*proof*⟩

lemma *ex-mset-zip-left*:
assumes *length xs* = *length ys mset xs'* = *mset xs*
shows $\exists$ *ys'*. *length ys'* = *length xs'* $\wedge$ *mset* (*zip xs' ys'*) = *mset* (*zip xs ys*)
⟨*proof*⟩

lemma *list-all2-reorder-left-invariance*:
assumes *rel*: *list-all2 R xs ys* **and** *ms-x*: *mset xs'* = *mset xs*
shows $\exists$ *ys'*. *list-all2 R xs' ys'* $\wedge$ *mset ys'* = *mset ys*
⟨*proof*⟩

lemma *ex-mset*: $\exists$ *xs*. *mset xs* = *X*
⟨*proof*⟩

inductive *pred-mset* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ '*a multiset* $\Rightarrow$ *bool*
where
pred-mset P {#}
| ⟦*P a*; *pred-mset P M*⟧ $\Longrightarrow$ *pred-mset P* (*add-mset a M*)

bnf *'a multiset*
map: *image-mset*
sets: *set-mset*
bd: *natLeq*
wits: {#}
rel: *rel-mset*
pred: *pred-mset*
⟨*proof*⟩

inductive *rel-mset'*
where
Zero[*intro*]: *rel-mset' R* {#} {#}
| *Plus*[*intro*]: ⟦*R a b*; *rel-mset' R M N*⟧ ⟹ *rel-mset' R* (*add-mset a M*) (*add-mset b N*)

lemma *rel-mset-Zero*: *rel-mset R* {#} {#}
⟨*proof*⟩

declare *multiset.count*[*simp*]
declare *Abs-multiset-inverse*[*simp*]
declare *multiset.count-inverse*[*simp*]
declare *union-preserves-multiset*[*simp*]

lemma *rel-mset-Plus*:
assumes *ab*: *R a b*
and *MN*: *rel-mset R M N*
shows *rel-mset R* (*add-mset a M*) (*add-mset b N*)
⟨*proof*⟩

lemma *rel-mset'-imp-rel-mset*: *rel-mset' R M N* ⟹ *rel-mset R M N*
⟨*proof*⟩

lemma *rel-mset-size*: *rel-mset R M N* ⟹ *size M* = *size N*
⟨*proof*⟩

lemma *multiset-induct2*[*case-names empty addL addR*]:
assumes *empty*: *P* {#} {#}
and *addL*: ⋀*a M N*. *P M N* ⟹ *P* (*add-mset a M*) *N*
and *addR*: ⋀*a M N*. *P M N* ⟹ *P M* (*add-mset a N*)
shows *P M N*
⟨*proof*⟩

lemma *multiset-induct2-size*[*consumes 1*, *case-names empty add*]:
assumes *c*: *size M* = *size N*
and *empty*: *P* {#} {#}
and *add*: ⋀*a b M N a b*. *P M N* ⟹ *P* (*add-mset a M*) (*add-mset b N*)
shows *P M N*
⟨*proof*⟩

lemma *msed-map-invL*:
assumes *image-mset f* (*add-mset a M*) = *N*
shows ∃ *N1*. *N* = *add-mset* (*f a*) *N1* ∧ *image-mset f M* = *N1*
⟨*proof*⟩

lemma *msed-map-invR*:
assumes *image-mset f M* = *add-mset b N*
shows ∃ *M1 a*. *M* = *add-mset a M1* ∧ *f a* = *b* ∧ *image-mset f M1* = *N*
⟨*proof*⟩

lemma *msed-rel-invL*:
assumes *rel-mset R* (*add-mset a M*) *N*
shows ∃ *N1 b*. *N* = *add-mset b N1* ∧ *R a b* ∧ *rel-mset R M N1*
⟨*proof*⟩

lemma *msed-rel-invR*:
assumes *rel-mset R M* (*add-mset b N*)
shows ∃ *M1 a*. *M* = *add-mset a M1* ∧ *R a b* ∧ *rel-mset R M1 N*
⟨*proof*⟩

lemma *rel-mset-imp-rel-mset′*:
assumes *rel-mset R M N*
shows *rel-mset′ R M N*
⟨*proof*⟩

lemma *rel-mset-rel-mset′*: *rel-mset R M N* = *rel-mset′ R M N*
⟨*proof*⟩

The main end product for *rel-mset*: inductive characterization:

lemmas *rel-mset-induct*[*case-names empty add*, *induct pred*: *rel-mset*] =
rel-mset′.induct[*unfolded rel-mset-rel-mset′*[*symmetric*]]

1.18 Size setup

lemma *multiset-size-o-map*:
size-multiset g ∘ *image-mset f* = *size-multiset* (*g* ∘ *f*)
⟨*proof*⟩

⟨*ML*⟩

hide-const (**open**) *wcount*

end

2 Additions to Isabelle's Main Theories

theory *Additions-to-Main*
imports ~~/*src*/*HOL*/*Library*/*Multiset*

begin

2.1 Addition to Finite-Set Theory

lemma *bound-domain-and-range-impl-finitely-many-functions*:
finite $\{f::nat \Rightarrow nat.\ (\forall\, i.\ f\ i \leq n) \wedge (\forall\, i \geq m.\ f\ i = 0)\}$
⟨*proof*⟩

2.2 Additions to Groups-Big Theory

lemma *sum-card-image*:
assumes *finite A*
assumes $\forall\, s \in A.\ \forall\, t \in A.\ s \neq t \longrightarrow (f\ s) \cap (f\ t) = \{\}$
shows *sum card* $(f\ `\ A) = sum\ (\lambda a.\ card\ (f\ a))\ A$
⟨*proof*⟩

lemma *card-Union-image*:
assumes *finite S*
assumes $\forall\, s \in f\ `\ S.$ *finite s*
assumes $\forall\, s \in S.\ \forall\, t \in S.\ s \neq t \longrightarrow (f\ s) \cap (f\ t) = \{\}$
shows *card* $(\bigcup (f\ `\ S)) = sum\ (\lambda x.\ card\ (f\ x))\ S$
⟨*proof*⟩

2.3 Addition to Set-Interval Theory

lemma *sum-atMost-remove-nat*:
assumes $k \leq (n :: nat)$
shows $(\sum i \leq n.\ f\ i) = f\ k + (\sum i \in \{..n\} - \{k\}.\ f\ i)$
⟨*proof*⟩

2.4 Additions to Multiset Theory

lemma *set-mset-Abs-multiset*:
assumes $f \in multiset$
shows *set-mset* $(Abs\text{-}multiset\ f) = \{x.\ f\ x > 0\}$
⟨*proof*⟩

lemma *sum-mset-sum-count*:
sum-mset $M = (\sum i \in set\text{-}mset\ M.\ count\ M\ i * i)$
⟨*proof*⟩

lemma *sum-mset-eq-sum-on-supersets*:
assumes *finite A set-mset* $M \subseteq A$
shows $(\sum i \in set\text{-}mset\ M.\ count\ M\ i * i) = (\sum i \in A.\ count\ M\ i * i)$
⟨*proof*⟩

end

3 Number Partitions

theory *Number-Partition*
imports *Additions-to-Main*
begin

3.1 Number Partitions as $nat \Rightarrow nat$ Functions

definition *partitions* :: $(nat \Rightarrow nat) \Rightarrow nat \Rightarrow bool$ (**infix** *partitions 50*)
where
p *partitions* $n = ((\forall i.\ p\ i \neq 0 \longrightarrow 1 \leq i \wedge i \leq n) \wedge (\sum i{\leq}n.\ p\ i * i) = n)$

lemma *partitionsI*:
assumes $\bigwedge i.\ p\ i \neq 0 \Longrightarrow 1 \leq i \wedge i \leq n$
assumes $(\sum i{\leq}n.\ p\ i * i) = n$
shows p *partitions* n
⟨*proof*⟩

lemma *partitionsE*:
assumes p *partitions* n
obtains $\bigwedge i.\ p\ i \neq 0 \Longrightarrow 1 \leq i \wedge i \leq n\ (\sum i{\leq}n.\ p\ i * i) = n$
⟨*proof*⟩

lemma *partitions-zero*:
p *partitions* $0 \longleftrightarrow p = (\lambda i.\ 0)$
⟨*proof*⟩

lemma *partitions-one*:
p *partitions* $(Suc\ 0) \longleftrightarrow p = (\lambda i.\ 0)(1 := 1)$
⟨*proof*⟩

3.2 Bounds and Finiteness of Number Partitions

lemma *partitions-imp-finite-elements*:
assumes p *partitions* n
shows *finite* $\{i.\ 0 < p\ i\}$
⟨*proof*⟩

lemma *partitions-imp-multiset*:
assumes p *partitions* n
shows $p \in$ *multiset*
⟨*proof*⟩

lemma *partitions-bounds*:
assumes p *partitions* n
shows $p\ i \leq n$
⟨*proof*⟩

lemma *partitions-parts-bounded*:
assumes p *partitions* n

shows *sum p* $\{..n\} \leq n$
⟨*proof*⟩

lemma *finite-partitions*:
finite $\{p.\ p$ *partitions* $n\}$
⟨*proof*⟩

lemma *finite-partitions-k-parts*:
finite $\{p.\ p$ *partitions* $n \wedge$ *sum* $p\ \{..n\} = k\}$
⟨*proof*⟩

lemma *partitions-remaining-Max-part*:
assumes *p partitions n*
assumes $0 < p\ k$
shows $\forall i.\ n - k < i \wedge i \neq k \longrightarrow p\ i = 0$
⟨*proof*⟩

3.3 Operations of Number Partitions

lemma *partitions-remove1-bounds*:
assumes *partitions*: *p partitions n*
assumes *gr0*: $0 < p\ k$
assumes *neq*: $(p(k := p\ k - 1))\ i \neq 0$
shows $1 \leq i \wedge i \leq n - k$
⟨*proof*⟩

lemma *partitions-remove1*:
assumes *partitions*: *p partitions n*
assumes *gr0*: $0 < p\ k$
shows $p(k := p\ k - 1)$ *partitions* $(n - k)$
⟨*proof*⟩

lemma *partitions-insert1*:
assumes *p*: *p partitions n*
assumes $k > 0$
shows $(p(k := p\ k + 1))$ *partitions* $(n + k)$
⟨*proof*⟩

lemma *count-remove1*:
assumes *p partitions n*
assumes $0 < p\ k$
shows $(\sum i{\leq}n - k.\ (p(k := p\ k - 1))\ i) = (\sum i{\leq}n.\ p\ i) - 1$
⟨*proof*⟩

lemma *count-insert1*:
assumes *p partitions n*
shows *sum* $(p(k := p\ k + 1))\ \{..n + k\} = (\sum i{\leq}n.\ p\ i) + 1$
⟨*proof*⟩

lemma *partitions-decrease1*:
assumes *p*: *p partitions m*
assumes *sum*: *sum p* {..*m*} = *k*
assumes *p 1* = *0*
shows (λ*i*. *p* (*i* + *1*)) *partitions m* − *k*
⟨*proof*⟩

lemma *partitions-increase1*:
assumes *partitions*: *p partitions m* − *k*
assumes *k*: *sum p* {..*m* − *k*} = *k*
shows (λ*i*. *p* (*i* − *1*)) *partitions m*
⟨*proof*⟩

lemma *count-decrease1*:
assumes *p*: *p partitions m*
assumes *sum*: *sum p* {..*m*} = *k*
assumes *p 1* = *0*
shows *sum* (λ*i*. *p* (*i* + *1*)) {..*m* − *k*} = *k*
⟨*proof*⟩

lemma *count-increase1*:
assumes *partitions*: *p partitions m* − *k*
assumes *k*: *sum p* {..*m* − *k*} = *k*
shows (∑ *i*≤*m*. *p* (*i* − *1*)) = *k*
⟨*proof*⟩

3.4 Number Partitions as Multisets on Natural Numbers

definition *number-partition* :: *nat* ⇒ *nat multiset* ⇒ *bool*
where
number-partition n N = (*sum-mset N* = *n* ∧ *0* ∉# *N*)

3.4.1 Relationship to Definition on Functions

lemma *count-partitions-iff*:
count N partitions n ⟷ *number-partition n N*
⟨*proof*⟩

lemma *partitions-iff-Abs-multiset*:
p partitions n ⟷ *finite* {*x*. *0* < *p x*} ∧ *number-partition n* (*Abs-multiset p*)
⟨*proof*⟩

lemma *size-nat-multiset-eq*:
fixes *N* :: *nat multiset*
assumes *number-partition n N*
shows *size N* = *sum* (*count N*) {..*n*}
⟨*proof*⟩

end

4 Euler's Partition Theorem

theory *Euler-Partition*
imports
Main
../Card-Number-Partitions/Number-Partition
begin

4.1 Preliminaries

4.1.1 Additions to Divides Theory

lemma *power-div-nat*:
assumes $c \leq b$
assumes $a > 0$
shows $(a :: nat)\ \hat{}\ b\ div\ a\ \hat{}\ c = a\ \hat{}\ (b - c)$
⟨*proof*⟩

4.1.2 Additions to Groups-Big Theory

lemma *sum-div*:
assumes *finite A*
assumes $\bigwedge a.\ a \in A \Longrightarrow (b::'b::semiring\text{-}div)\ dvd\ f\ a$
shows $(\sum a{\in}A.\ f\ a)\ div\ b = (\sum a{\in}A.\ (f\ a)\ div\ b)$
⟨*proof*⟩

lemma *sum-mod*:
assumes *finite A*
assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a\ mod\ b = (0::'b::\{semiring\text{-}div\})$
shows $(\sum a{\in}A.\ f\ a)\ mod\ b = 0$
⟨*proof*⟩

4.1.3 Additions to Set-Interval Theory

lemma *geometric-sum-2nat*:
$(\sum i{<}n.\ (2::nat)\ \hat{}\ i) = (2\ \hat{}\ n - 1)$
⟨*proof*⟩

4.1.4 Additions to Nat Theory or Power Theory

lemma *n-leq-2-pow-n*:
$n \leq 2\ \hat{}\ n$
⟨*proof*⟩

4.1.5 Additions to Finite-Set Theory

lemma *finite-exponents*:
finite $\{i.\ 2\ \hat{}\ i \leq (n::nat)\}$
⟨*proof*⟩

4.2 Binary Encoding of Natural Numbers

definition *bitset* :: *nat* ⇒ *nat set*
where
  bitset n = {*i*. *odd* (*n div* (2 ^ *i*))}

lemma *in-bitset-bound*:
  b ∈ *bitset n* ⟹ 2 ^ *b* ≤ *n*
⟨*proof*⟩

lemma *in-bitset-bound-weak*:
  b ∈ *bitset n* ⟹ *b* ≤ *n*
⟨*proof*⟩

lemma *finite-bitset*:
  finite (*bitset n*)
⟨*proof*⟩

lemma *bitset-0*:
  bitset 0 = {}
⟨*proof*⟩

lemma *binary-induct* [*case-names zero even odd*]:
  assumes *P* (0 :: *nat*)
  assumes ⋀*n*. *P n* ⟹ *P* (2 ∗ *n*)
  assumes ⋀*n*. *P n* ⟹ *P* (2 ∗ *n* + 1)
  shows ⋀*n*. *P n*
⟨*proof*⟩

lemma *bitset-2n*: *bitset* (2 ∗ *n*) = *Suc* ‘ (*bitset n*)
⟨*proof*⟩

lemma *bitset-Suc*:
  assumes *even n*
  shows *bitset* (*n* + 1) = *insert* 0 (*bitset n*)
⟨*proof*⟩

lemma *bitset-2n1*:
  bitset (2 ∗ *n* + 1) = *insert* 0 (*Suc* ‘ (*bitset n*))
⟨*proof*⟩

lemma *sum-bitset*:
  (∑ *i*∈*bitset n*. 2 ^ *i*) = *n*
⟨*proof*⟩

lemma *binarysum-div*:
  assumes *finite B*
  shows (∑ *i*∈*B*. (2::*nat*) ^ *i*) *div* 2 ^ *j* = (∑ *i*∈*B*. *if i* < *j then* 0 *else* 2 ^ (*i* − *j*))
  (**is** - = (∑ *i*∈-. *?f i*))

⟨*proof*⟩

lemma *odd-iff*:
assumes *finite B*
shows *odd* (∑ *i*∈*B*. *if i* < *x then* (*0*::*nat*) *else* 2 ^ (*i* − *x*)) = (*x* ∈ *B*) (**is** *odd* (∑ *i*∈-. *?s i*) = -)
⟨*proof*⟩

lemma *bitset-sum*:
assumes *finite B*
shows *bitset* (∑ *i*∈*B*. 2 ^ *i*) = *B*
⟨*proof*⟩

4.3 Decomposition of a Number into a Power of Two and an Odd Number

function (*sequential*) *index* :: *nat* ⇒ *nat*
where
index 0 = *0*
| *index n* = (*if odd n then 0 else Suc* (*index* (*n div 2*)))
⟨*proof*⟩

termination
⟨*proof*⟩

function (*sequential*) *oddpart* :: *nat* ⇒ *nat*
where
oddpart 0 = *0*
| *oddpart n* = (*if odd n then n else oddpart* (*n div 2*))
⟨*proof*⟩

termination
⟨*proof*⟩

lemma *odd-oddpart*:
odd (*oddpart n*) ⟷ *n* ≠ *0*
⟨*proof*⟩

lemma *index-oddpart-decomposition*:
n = 2 ^ (*index n*) ∗ *oddpart n*
⟨*proof*⟩

lemma *oddpart-leq*:
oddpart n ≤ *n*
⟨*proof*⟩

lemma *index-oddpart-unique*:
assumes *odd* (*m* :: *nat*) *odd m′*
shows (2 ^ *i* ∗ *m* = 2 ^ *i′* ∗ *m′*) ⟷ (*i* = *i′* ∧ *m* = *m′*)

⟨*proof*⟩

lemma *index-oddpart*:
assumes *odd m*
shows *index* (*2* ^ *i* ∗ *m*) = *i oddpart* (*2* ^ *i* ∗ *m*) = *m*
⟨*proof*⟩

4.4 Partitions With Only Distinct and Only Odd Parts

definition *odd-of-distinct* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat*
where
odd-of-distinct p = (λ*i*. *if odd i then* (∑ *j* | *p* (*2* ^ *j* ∗ *i*) = *1*. *2* ^ *j*) *else 0*)

definition *distinct-of-odd* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat*
where
distinct-of-odd p = (λ*i*. *if index i* ∈ *bitset* (*p* (*oddpart i*)) *then 1 else 0*)

lemma *odd*:
odd-of-distinct p i ≠ *0* ⟹ *odd i*
⟨*proof*⟩

lemma *distinct-distinct-of-odd*:
distinct-of-odd p i ≤ *1*
⟨*proof*⟩

lemma *odd-of-distinct*:
assumes *odd-of-distinct p i* ≠ *0*
assumes ⋀*i*. *p i* ≠ *0* ⟹ *i* ≤ *n*
shows *1* ≤ *i* ∧ *i* ≤ *n*
⟨*proof*⟩

lemma *distinct-of-odd*:
assumes ⋀*i*. *p i* ∗ *i* ≤ *n* ⋀*i*. *p i* ≠ *0* ⟹ *odd i*
assumes *distinct-of-odd p i* ≠ *0*
shows *1* ≤ *i* ∧ *i* ≤ *n*
⟨*proof*⟩

lemma *odd-distinct*:
assumes ⋀*i*. *p i* ≠ *0* ⟹ *odd i*
shows *odd-of-distinct* (*distinct-of-odd p*) = *p*
⟨*proof*⟩

lemma *distinct-odd*:
assumes ⋀*i*. *p i* ≠ *0* ⟹ *1* ≤ *i* ∧ *i* ≤ *n* ⋀*i*. *p i* ≤ *1*
shows *distinct-of-odd* (*odd-of-distinct p*) = *p*
⟨*proof*⟩

lemma *sum-distinct-of-odd*:
assumes ⋀*i*. *p i* ≠ *0* ⟹ *1* ≤ *i* ∧ *i* ≤ *n*

assumes $\bigwedge i.\ p\ i * i \leq n$
assumes $\bigwedge i.\ p\ i \neq 0 \Longrightarrow odd\ i$
shows $(\sum i{\leq}n.\ distinct\text{-}of\text{-}odd\ p\ i * i) = (\sum i{\leq}n.\ p\ i * i)$
⟨*proof*⟩

lemma *leq-n*:
assumes $\forall i.\ 0 < p\ i \longrightarrow 1 \leq i \wedge i \leq (n{::}nat)$
assumes $(\sum i{\leq}n.\ p\ i * i) = n$
shows $p\ i * i \leq n$
⟨*proof*⟩

lemma *distinct-of-odd-in-distinct-partitions*:
assumes $p \in \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \neq 0 \longrightarrow odd\ i)\}$
shows $distinct\text{-}of\text{-}odd\ p \in \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \leq 1)\}$
⟨*proof*⟩

lemma *odd-of-distinct-in-odd-partitions*:
assumes $p \in \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \leq 1)\}$
shows $odd\text{-}of\text{-}distinct\ p \in \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \neq 0 \longrightarrow odd\ i)\}$
⟨*proof*⟩

4.5 Euler's Partition Theorem

theorem *Euler-partition-theorem*:
$card\ \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \leq 1)\} = card\ \{p.\ p\ partitions\ n \wedge (\forall i.\ p\ i \neq 0 \longrightarrow odd\ i)\}$
(**is** *card ?distinct-partitions = card ?odd-partitions*)
⟨*proof*⟩

end

References

[1] J. Harrison. Euler's partition theorem and other elementary partition theorems. https://github.com/jrh13/hol-light/blob/master/100/euler.ml.

[2] G. Musiker. Course 18.312: Algebraic combinatorics, 2009. http://ocw.mit.edu/courses/mathematics/18-312-algebraic-combinatorics-spring-2009/readings-and-lecture-notes/MIT18_312S09_lec10_Patitio.pdf.