# VCG

M. B. Caminati, C. Lange, M. Kerber, C. Rowat

November 26, 2014

## Contents

# 1 Additional material that we would have expected in Set.thy

**theory** *SetUtils*
**imports**
*Main*

**begin**

# 2 Equality

An inference rule that combines $[\![?A \subseteq ?B; ?B \subseteq ?A]\!] \Longrightarrow ?A = ?B$ and $(\bigwedge x.\ x \in ?A \Longrightarrow x \in ?B) \Longrightarrow ?A \subseteq ?B$ to a single step

**lemma** *equalitySubsetI*: $(\bigwedge x\ .\ x \in A \Longrightarrow x \in B) \Longrightarrow (\bigwedge x\ .\ x \in B \Longrightarrow x \in A) \Longrightarrow A = B$ **by** *blast*

# 3 Trivial sets

A trivial set (i.e. singleton or empty), as in Mizar

**definition** *trivial* **where** $trivial\ x = (x \subseteq \{the\text{-}elem\ x\})$

The empty set is trivial.

**lemma** *trivial-empty*: *trivial* $\{\}$ **unfolding** *trivial-def* **by** (*rule empty-subsetI*)

A singleton set is trivial.

**lemma** *trivial-singleton*: *trivial* $\{x\}$ **unfolding** *trivial-def* **by** *simp*

If there are no two different elements in a set, it is trivial.

**lemma** *no-distinct-imp-trivial*:
  **assumes** $\forall\ x\ y\ .\ x \in X \wedge y \in X \longrightarrow x = y$
  **shows** *trivial X*

**unfolding** *trivial-def*
**proof**
  **fix** $x{::}'a$
  **assume** *x-in-X*: $x \in X$
  **with** *assms* **have** *uniq*: $\forall\ y \in X\ .\ x = y$ **by** *force*
  **have** $X = \{x\}$
  **proof** (*rule equalitySubsetI*)
    **fix** $x'{::}'a$
    **assume** $x' \in X$
    **with** *uniq* **show** $x' \in \{x\}$ **by** *simp*
  **next**
    **fix** $x'{::}'a$
    **assume** $x' \in \{x\}$
    **with** *x-in-X* **show** $x' \in X$ **by** *simp*
  **qed**
  **then show** $x \in \{\textit{the-elem}\ X\}$ **by** *simp*
**qed**

If there exists a unique $x$ with some property, then the set of all such $x$ is trivial.

**lemma** *ex1-imp-trivial*:
  **assumes** $\exists!\ x\ .\ P\ x$
  **shows** *trivial* $\{\ x\ .\ P\ x\ \}$
**proof** −
  **from** *assms* **have** $\forall\ a\ b\ .\ a \in \{\ x\ .\ P\ x\ \} \wedge b \in \{\ x\ .\ P\ x\ \} \longrightarrow a = b$ **by** *blast*
  **then show** *?thesis* **by** (*rule no-distinct-imp-trivial*)
**qed**

If a trivial set has a singleton subset, the latter is unique.

**lemma** *singleton-sub-trivial-uniq*:
  **fixes** *x X*
  **assumes** $\{x\} \subseteq X$
    **and** *trivial X*
  **shows** $x = \textit{the-elem}\ X$

**using** *assms* **unfolding** *trivial-def* **by** *fast*

Any subset of a trivial set is trivial.

**lemma** *trivial-subset*: **fixes** *X Y* **assumes** *trivial Y* **assumes** $X \subseteq Y$
**shows** *trivial X*

**using** *assms* **unfolding** *trivial-def* **by** (*metis* (*full-types*) *subset-empty subset-insertI2 subset-singletonD*)

There are no two different elements in a trivial set.

**lemma** *trivial-imp-no-distinct*:
**assumes** *triv*: *trivial* $X$
**and** $x$: $x \in X$
**and** $y$: $y \in X$
**shows** $x = y$

**using** *assms*
**by** (*metis empty-subsetI insert-subset singleton-sub-trivial-uniq*)

# 4 The image of a set under a function

an equivalent notation for the image of a set, using set comprehension

**lemma** *image-Collect-mem*: $\{ f\, x \mid x .\ x \in S \} = f \text{ ` } S$ **by** *auto*

# 5 Set difference

Subtracting a proper subset from a set yields another proper subset.

**lemma** *Diff-psubset-is-psubset*:
**assumes** $A \neq \{\}$
**and** $A \subset B$
**shows** $B - A \subset B$

**using** *assms*
**by** *blast*

**lemma** *card-diff-gt-0*:
**assumes** *finite* $B$
**and** *card* $A >$ *card* $B$
**shows** *card* $(A - B) > 0$
**using** *assms*
**by** (*metis diff-card-le-card-Diff le-0-eq neq0-conv zero-less-diff*)

# 6 Big Union

An element is in the union of a family of sets if it is in one of the family's member sets.

**lemma** *Union-member*: $(\exists\ S \in F .\ x \in S) \longleftrightarrow x \in \bigcup\ F$ **by** *blast*

When a set of elements $A$ is non-empty, and a function $f$ returns a non-empty set for at least one member of $A$, the union of the image of $A$ under $f$ is non-empty, too.

**lemma** *Union-map-non-empty*:
**assumes** $x \in A$

**and** $f\,x \neq \{\}$
**shows** $\bigcup\ (f\ `\ A) \neq \{\}$
**proof** −
**from** *assms(1)* **have** $f\ `\ A \neq \{\}$ **by** *fast*
**with** *assms* **show** *?thesis* **by** *force*
**qed**

Two alternative notations for the big union operator involving set comprehension are equivalent.

**lemma** *Union-set-compr-eq*: $(\bigcup\ x{\in}A\ .\ f\,x) = \bigcup\ \{\ f\,x \mid x\ .\ x \in A\ \}$
**proof** (*rule equalitySubsetI*)
**fix** $y$
**assume** $y \in (\bigcup\ x{\in}A\ .\ f\,x)$
**then obtain** $z$ **where** $z \in \{\ f\,x \mid x\ .\ x \in A\ \}$ **and** $y \in z$ **by** *blast*
**then show** $y \in \bigcup\ \{\ f\,x \mid x\ .\ x \in A\ \}$ **by** (*rule UnionI*)
**next**
**fix** $y$
**assume** $y \in \bigcup\ \{\ f\,x \mid x\ .\ x \in A\ \}$
**then show** $y \in (\bigcup\ x{\in}A\ .\ f\,x)$ **by** *force*
**qed**

**lemma** *Union-map-compr-eq1*:
**fixes** $x{::}'a$
**and** $f{::}'b \Rightarrow 'a\ set$
**and** $P{::}'b\ set$
**shows** $x \in (\bigcup\ \{f\ Y \mid Y\ .\ Y \in P\}) \longleftrightarrow (\exists\ Y \in P\ .\ x \in f\ Y)$

**proof** −
**have** $x \in (\bigcup\ \{f\ Y \mid Y\ .\ Y \in P\}) \longleftrightarrow x \in (\bigcup\ (f\ `\ P))$ **by** (*simp add: image-Collect-mem*)
**also have** $\ldots \longleftrightarrow (\exists\ y \in (f\ `\ P)\ .\ x \in y)$ **by** (*rule Union-iff*)
**also have** $\ldots \longleftrightarrow (\exists\ y\ .\ y \in (f\ `\ P)\ \&\ x \in y)$ **by** *force*
**also have** $\ldots \longleftrightarrow (\exists\ y \in (f\ `\ P)\ .\ x \in y)$ **by** *blast*
**also have** $\ldots \longleftrightarrow (\exists\ Y \in P\ .\ x \in (f\ Y))$ **by** *force*
**finally show** *?thesis* **.**
**qed**

**lemma** *ll69*: **assumes** *trivial* $t$ $t \cap X \neq \{\}$ **shows** $t \subseteq X$ **using** *trivial-def assms in-mono* **by** *fast*

**lemma** *lm54*: **assumes** *trivial* $X$ **shows** *finite* $X$
**using** *assms* **by** (*metis finite.simps subset-singletonD trivial-def*)

**lemma** *lm001a*: **assumes** *trivial* $(A \times B)$ **shows** (*finite* $(A{\times}B)$ & *card* $A * ($*card* $B) \leq 1$)

**using** *trivial-def assms One-nat-def card-cartesian-product card-empty card-insert-disjoint empty-iff finite.emptyI le0 lm54 order-refl subset-singletonD* **by** (*metis*(*no-types*))

**lemma** *ll97*: **assumes** *finite X* **shows** *trivial X*=(*card X* $\leq$ *1*) (**is** *?LH*=*?RH*)
**using** *assms One-nat-def card-empty card-insert-if card-mono card-seteq empty-iff empty-subsetI*
*finite.cases finite.emptyI finite-insert insert-mono trivial-def trivial-singleton*
**by** (*metis*(*no-types*))

**lemma** *ll10*: **shows** *trivial* $\{x\}$ **by** (*metis order-refl the-elem-eq trivial-def*)

**lemma** *ll11*: **assumes** *trivial X* $\{x\} \subseteq X$ **shows** $\{x\}=X$
**using** *singleton-sub-trivial-uniq assms* **by** (*metis subset-antisym trivial-def*)

**lemma** *ll26*: **assumes** $\neg$ *trivial X trivial T* **shows** $X-T \neq \{\}$
**using** *assms* **by** (*metis Diff-iff empty-iff subsetI trivial-subset*)

**lemma** *lm001b*: **assumes** (*finite* ($A \times B$) & *card A* $*$ (*card B*) $\leq$ *1*) **shows** *trivial* ($A \times B$)
**unfolding** *trivial-def* **using** *trivial-def assms* **by** (*metis card-cartesian-product ll97*)

**lemma** *lm001*: *trivial* ($A \times B$)=(*finite* ($A \times B$) & *card A* $*$ (*card B*) $\leq$ *1*) **using** *lm001a lm001b* **by** *blast*

**lemma** *lm01*: *trivial X* = ($\forall$ *x1* $\in$ *X*. $\forall$ *x2* $\in$ *X*. *x1*=*x2*) **unfolding** *trivial-def* **using** *trivial-def*
**by** (*metis no-distinct-imp-trivial trivial-imp-no-distinct*)

**lemma** *lm009a*: **assumes** (*Pow X* $\subseteq \{\{\},X\}$) **shows** *trivial X* **unfolding** *lm01* **using** *assms* **by** *auto*

**lemma** *lm009b*: **assumes** *trivial X* **shows** (*Pow X* $\subseteq \{\{\},X\}$) **using** *assms lm01* **by** *fast*

**lemma** *lm009*: *trivial X* = (*Pow X* $\subseteq \{\{\},X\}$) **using** *lm009a lm009b* **by** *metis*

**lemma** *lm007*: *trivial X* = ($X=\{\}$ $\vee$ $X=\{$*the-elem X*$\}$)
**by** (*metis subset-singletonD trivial-def trivial-empty trivial-singleton*)

**lemma** *ll40*: **assumes** *trivial X trivial Y* **shows** *trivial* ($X \times Y$)
**using** *assms lm001 One-nat-def Sigma-empty1 Sigma-empty2 card-empty card-insert-if finite-SigmaI*
*lm54 nat-1-eq-mult-iff order-refl subset-singletonD trivial-def trivial-empty*
**by** (*metis* (*full-types*))

**lemma** *lm002*: ($\{x\} \times$ *UNIV*) $\cap$ $P$ = $\{x\} \times$ ($P$ `` $\{x\}$) **by** *fast*

**lemma** *lm00*: $(x,y) \in P$ = ($y \in P$``$\{x\}$) **by** *simp*

**lemma** *lm010*: **assumes** *inj-on f A inj-on f B* **shows** *inj-on f* $(A \cup B) = (f`(A-B)$ $\cap$ $(f`(B-A))=\{\})$
**using** *assms inj-on-Un* **by** (*metis*)

**lemma** *lm010b*: **assumes** *inj-on f A inj-on f B* $f`A \cap (f`B)=\{\}$ **shows** *inj-on f* $(A$ $\cup$ $B)$
**using** *assms lm010* **by** *fast*

**lemma** *lm008*: $(Pow\ X = \{X\}) = (X=\{\})$ **by** *auto*

**end**

# 7 Partitions of sets

**theory** *Partitions*
**imports**
*Main*
*SetUtils*

**begin**

$P$ is a partition of some set.

**definition** *is-partition* **where**
*is-partition* $P = (\forall\ X{\in}P\ .\ \forall\ Y{\in}\ P\ .\ (X \cap Y \neq \{\} \longleftrightarrow X = Y))$

A subset of a partition is also a partition (but, note: only of a subset of the original set)

**lemma** *subset-is-partition*:
**assumes** *subset*: $P \subseteq Q$
**and** *partition*: *is-partition* $Q$
**shows** *is-partition* $P$

**proof** −
**{**
**fix** $X$ $Y$ **assume** $X \in P \wedge Y \in P$
**then have** $X \in Q \wedge Y \in Q$ **using** *subset* **by** *fast*
**then have** $X \cap Y \neq \{\} \longleftrightarrow X = Y$ **using** *partition* **unfolding** *is-partition-def* **by** *force*
**}**
**then show** *?thesis* **unfolding** *is-partition-def* **by** *force*
**qed**

The set that results from removing one element from an equivalence class of a partition is not otherwise a member of the partition.

**lemma** *remove-from-eq-class-preserves-disjoint*:
**fixes** *elem*::$'a$
**and** $X$::$'a$ *set*

**and** $P::'a\ set\ set$
**assumes** *partition*: *is-partition* $P$
**and** *eq-class*: $X \in P$
**and** *elem*: $elem \in X$
**shows** $X - \{elem\} \notin P$
**using** *assms*
*Int-Diff is-partition-def*
**by** (*metis Diff-disjoint Diff-eq-empty-iff Int-absorb2 insert-Diff-if insert-not-empty*)

Inserting into a partition $P$ a set $X$, which is disjoint with the set partitioned by $P$, yields another partition.

**lemma** *partition-extension1*:
**fixes** $P::'a\ set\ set$
**and** $X::'a\ set$
**assumes** *partition*: *is-partition* $P$
**and** *disjoint*: $X \cap \bigcup P = \{\}$
**and** *non-empty*: $X \neq \{\}$
**shows** *is-partition* (*insert* $X$ $P$)
**proof** –
**{**
**fix** $Y::'a\ set$ **and** $Z::'a\ set$
**assume** *Y-Z-in-ext-P*: $Y \in insert\ X\ P \wedge Z \in insert\ X\ P$
**have** $Y \cap Z \neq \{\} \longleftrightarrow Y = Z$
**proof**
**assume** $Y \cap Z \neq \{\}$
**then show** $Y = Z$
**using** *Y-Z-in-ext-P partition disjoint*
**unfolding** *is-partition-def*
**by** *fast*
**next**
**assume** $Y = Z$
**then show** $Y \cap Z \neq \{\}$
**using** *Y-Z-in-ext-P partition non-empty*
**unfolding** *is-partition-def*
**by** *auto*
**qed**
**}**
**then show** *?thesis* **unfolding** *is-partition-def* **by** *force*
**qed**

An equivalence class of a partition has no intersection with any of the other equivalence classes.

**lemma** *disj-eq-classes*:
**fixes** $P::'a\ set\ set$
**and** $X::'a\ set$
**assumes** *is-partition* $P$
**and** $X \in P$
**shows** $X \cap \bigcup (P - \{X\}) = \{\}$
**proof** –

```
{
  fix x::'a
  assume x-in-two-eq-classes: x ∈ X ∩ ⋃ (P − {X})
  then obtain Y where other-eq-class: Y ∈ P − {X} ∧ x ∈ Y by blast
  have x ∈ X ∩ Y ∧ Y ∈ P
    using x-in-two-eq-classes other-eq-class by force
  then have X = Y using assms is-partition-def by fast
  then have x ∈ {} using other-eq-class by fast
}
then show ?thesis by blast
qed
```

In a partition there is no empty equivalence class.

```
lemma no-empty-eq-class:
  assumes is-partition p
  shows {} ∉ p

  using assms is-partition-def by fast
```

$P$ is a partition of the set $A$.

```
definition is-partition-of (infix partitions 75)
where is-partition-of P A = (⋃ P = A ∧ is-partition P)
```

No partition of a non-empty set is empty.

```
lemma non-empty-imp-non-empty-partition:
  assumes A ≠ {}
    and is-partition-of P A
  shows P ≠ {}
using assms
unfolding is-partition-of-def
by fast
```

Every element of a partitioned set ends up in an equivalence class.

```
lemma elem-in-eq-class:
  assumes in-set: x ∈ A
    and part: is-partition-of P A
  obtains X where x ∈ X and X ∈ P
using part in-set
unfolding is-partition-of-def is-partition-def
by (auto simp add: UnionE)
```

Every element of the difference of a set $A$ and another set $B$ ends up in an equivalence class of a partition of $A$, but this equivalence class will never be $\{B\}$.

```
lemma diff-elem-in-eq-class:
  assumes x: x ∈ A − B
    and part: is-partition-of P A
  shows ∃ S ∈ P − { B } . x ∈ S
```

**proof** −
  **from** *part* $x$ **obtain** $X$ **where** $x \in X$ **and** $X \in P$
    **by** (*metis Diff-iff elem-in-eq-class*)
  **with** $x$ **have** $X \neq B$ **by** *fast*
  **with** ‹$x \in X$› ‹$X \in P$› **show** *?thesis* **by** *blast*
**qed**

Every element of a partitioned set ends up in exactly one equivalence class.

**lemma** *elem-in-uniq-eq-class*:
  **assumes** *in-set*: $x \in A$
    **and** *part*: *is-partition-of* $P$ $A$
  **shows** $\exists!\ X \in P\ .\ x \in X$
**proof** −
  **from** *assms* **obtain** $X$ **where** $*$: $X \in P \wedge x \in X$
    **by** (*rule elem-in-eq-class*) *blast*
  **moreover** **{**
    **fix** $Y$ **assume** $Y \in P \wedge x \in Y$
    **then have** $Y = X$
      **using** *part in-set* $*$
      **unfolding** *is-partition-of-def is-partition-def*
      **by** (*metis disjoint-iff-not-equal*)
  **}**
  **ultimately show** *?thesis* **by** (*rule ex1I*)
**qed**

A non-empty set is a partition of itself.

**lemma** *set-partitions-itself*:
  **assumes** $A \neq \{\}$
  **shows** *is-partition-of* $\{A\}$ $A$ **unfolding** *is-partition-of-def is-partition-def*

**proof**
  **show** $\bigcup\ \{A\} = A$ **by** *simp*
  **{**
    **fix** $X$ $Y$
    **assume** $X \in \{A\}$
    **then have** $X = A$ **by** (*rule singletonD*)
    **assume** $Y \in \{A\}$
    **then have** $Y = A$ **by** (*rule singletonD*)
    **from** ‹$X = A$› ‹$Y = A$› **have** $X \cap Y \neq \{\} \longleftrightarrow X = Y$ **using** *assms* **by** *simp*
  **}**
  **then show** $\forall\ X \in \{A\}\ .\ \forall\ Y \in \{A\}\ .\ X \cap Y \neq \{\} \longleftrightarrow X = Y$ **by** *force*
**qed**

The empty set is a partition of the empty set.

**lemma** *emptyset-part-emptyset1*:
  **shows** *is-partition-of* $\{\}$ $\{\}$
  **unfolding** *is-partition-of-def is-partition-def* **by** *fast*

Any partition of the empty set is empty.

**lemma** *emptyset-part-emptyset2*:
  **assumes** *is-partition-of* $P$ $\{\}$
  **shows** $P = \{\}$
  **using** *assms is-partition-def is-partition-of-def* **by** *fast*

classical set-theoretical definition of "all partitions of a set $A$"

**definition** *all-partitions* **where**
*all-partitions* $A = \{P$ . *is-partition-of* $P$ $A\}$

A finite set has finitely many partitions.

**lemma** *finite-all-partitions*:
  **assumes** *finite* $A$
  **shows** *finite* (*all-partitions* $A$)
**unfolding** *all-partitions-def is-partition-of-def is-partition-def*

**proof**
  **have** *finite* (*Pow* (*Pow* $A$)) **using** *assms* **by** *simp*
  **moreover have** $\{ P . \bigcup P = A \} \subseteq$ *Pow* (*Pow* $A$)
  **proof**
    **fix** $P$ **assume** $P \in \{ P . \bigcup P = A \}$
    **then show** $P \in$ *Pow* (*Pow* $A$) **by** *blast*
  **qed**
  **ultimately have** *finite* $\{ P . \bigcup P = A \}$ **by** (*rule rev-finite-subset*)

  **then show** *finite* $\{P . \bigcup P = A\} \vee$ *finite* $\{P. \forall X \in P . \forall Y \in P . (X \cap Y \neq \{\}) \longleftrightarrow X = Y\}$ **..**
**qed**

The set of all partitions of the empty set only contains the empty set. We need this to prove the base case of *all-partitions-paper-equiv-alg*.

**lemma** *emptyset-part-emptyset3*:
  **shows** *all-partitions* $\{\} = \{\{\}\}$
  **unfolding** *all-partitions-def*
  **using** *emptyset-part-emptyset1 emptyset-part-emptyset2*
  **by** *fast*

inserts an element into a specified set inside the given set of sets

**definition** *insert-into-member* :: $'a \Rightarrow 'a$ *set set* $\Rightarrow 'a$ *set* $\Rightarrow 'a$ *set set*
**where** *insert-into-member new-el Sets S* = *insert* ($S \cup \{$*new-el*$\}$) (*Sets* $- \{S\}$)

Using *insert-into-member* to insert a fresh element, which is not a member of the set $S$ being partitioned, into an equivalence class of a partition yields another partition (of – we don't prove this here – the set $S \cup \{$*new-el*$\}$).

**lemma** *partition-extension2*:
  **fixes** *new-el*$::'a$

**and** $P$::$'a$ *set set*
**and** $X$::$'a$ *set*
**assumes** *partition*: *is-partition* $P$
**and** *eq-class*: $X \in P$
**and** *new*: *new-el* $\notin \bigcup$ $P$
**shows** *is-partition* (*insert-into-member new-el* $P$ $X$)
**proof** −
**let** $?Y$ = *insert new-el* $X$
**have** *rest-is-partition*: *is-partition* ($P - \{X\}$)
**using** *partition subset-is-partition* **by** *blast*
**have** ∗: $X \cap \bigcup$ ($P - \{X\}$) = {}
**using** *partition eq-class* **by** (*rule disj-eq-classes*)
**from** ∗ **have** *non-empty*: $?Y \neq \{\}$ **by** *blast*
**from** ∗ **have** *disjoint*: $?Y \cap \bigcup$ ($P - \{X\}$) = {} **using** *new* **by** *force*
**have** *is-partition* (*insert* $?Y$ ($P - \{X\}$))
**using** *rest-is-partition disjoint non-empty* **by** (*rule partition-extension1*)
**then show** *?thesis* **unfolding** *insert-into-member-def* **by** *simp*
**qed**

inserts an element into a specified set inside the given list of sets – the list variant of *insert-into-member*

The rationale for this variant and for everything that depends on it is: While it is possible to computationally enumerate "all partitions of a set" as an $'a$ *set set set*, we need a list representation to apply further computational functions to partitions. Because of the way we construct partitions (using functions such as *all-coarser-partitions-with* below) it is not sufficient to simply use $'a$ *set set list*, but we need $'a$ *set list list*. This is because it is hard to impossible to convert a set to a list, whereas it is easy to convert a *list* to a *set*.

**definition** *insert-into-member-list*
:: $'a \Rightarrow 'a$ *set list* $\Rightarrow 'a$ *set* $\Rightarrow 'a$ *set list*
**where** *insert-into-member-list new-el Sets S* = ($S \cup$ {*new-el*}) # (*remove1 S Sets*)

*insert-into-member-list* and *insert-into-member* are equivalent (as in returning the same set).

**lemma** *insert-into-member-list-alt*:
**fixes** *new-el*::$'a$
**and** *Sets*::$'a$ *set list*
**and** $S$::$'a$ *set*
**assumes** *distinct Sets*
**shows** *set* (*insert-into-member-list new-el Sets S*) = *insert-into-member new-el* (*set Sets*) $S$
**unfolding** *insert-into-member-list-def insert-into-member-def*
**using** *assms*
**by** *simp*

an alternative characterisation of the set partitioned by a partition obtained by inserting an element into an equivalence class of a given partition (if $P$

*is* a partition)

**lemma** *insert-into-member-partition1*:
  **fixes** *elem*::′*a*
    **and** *P*::′*a set set*
    **and** *eq-class*::′*a set*

  **shows** $\bigcup$ *insert-into-member elem P eq-class* = $\bigcup$ *insert* (*eq-class* ∪ {*elem*}) (*P* − {*eq-class*})

    **unfolding** *insert-into-member-def*
    **by** *fast*

Assuming that $P$ is a partition of a set $S$, and *new-el* ∉ $S$, this function yields all possible partitions of $S$ ∪ {*new-el*} that are coarser than $P$ (i.e. not splitting equivalence classes that already exist in $P$). These comprise one partition with an equivalence class {*new-el*} and all other equivalence classes unchanged, as well as all partitions obtained by inserting *new-el* into one equivalence class of $P$ at a time.

**definition** *coarser-partitions-with* ::′*a* ⇒ ′*a set set* ⇒ ′*a set set set*
**where** *coarser-partitions-with new-el P* =
  *insert*
  (∗ *Let P be a partition of a set Set,*
    *and suppose new-el* ∉ *Set, i.e.* {*new-el*} ∉ *P,*
    *then the following constructs a partition of 'Set* ∪ {*new-el*}*' obtained by*
    *inserting a new equivalence class* {*new-el*} *and leaving all previous equivalence classes unchanged.* ∗)
  (*insert* {*new-el*} *P*)
  (∗ *Let P be a partition of a set Set,*
    *and suppose new-el* ∉ *Set,*
    *then the following constructs*
    *the set of those partitions of 'Set* ∪ {*new-el*}*' obtained by*
    *inserting new-el into one equivalence class of P at a time.* ∗)
  ((*insert-into-member new-el P*) ‘ *P*)

the list variant of *coarser-partitions-with*

**definition** *coarser-partitions-with-list* ::′*a* ⇒ ′*a set list* ⇒ ′*a set list list*
**where** *coarser-partitions-with-list new-el P* =
  (∗ *Let P be a partition of a set Set,*
    *and suppose new-el* ∉ *Set, i.e.* {*new-el*} ∉ *set P,*
    *then the following constructs a partition of 'Set* ∪ {*new-el*}*' obtained by*
    *inserting a new equivalence class* {*new-el*} *and leaving all previous equivalence classes unchanged.* ∗)
  ({*new-el*} # *P*)
  #
  (∗ *Let P be a partition of a set Set,*
    *and suppose new-el* ∉ *Set,*
    *then the following constructs*
    *the set of those partitions of 'Set* ∪ {*new-el*}*' obtained by*

*inserting new-el into one equivalence class of P at a time.* ∗)
(*map* ((*insert-into-member-list new-el P*)) *P*)

*coarser-partitions-with-list* and *coarser-partitions-with* are equivalent.

**lemma** *coarser-partitions-with-list-alt*:
**assumes** *distinct P*
**shows** *set* (*map set* (*coarser-partitions-with-list new-el P*)) = *coarser-partitions-with new-el* (*set P*)
**proof** −
**have** *set* (*map set* (*coarser-partitions-with-list new-el P*)) = *set* (*map set* (({*new-el*} # *P*) # (*map* ((*insert-into-member-list new-el P*)) *P*)))
**unfolding** *coarser-partitions-with-list-def* **..**
**also have** ... = *insert* (*insert* {*new-el*} (*set P*)) ((*set* ∘ (*insert-into-member-list new-el P*)) ‘ *set P*)
**by** *simp*
**also have** ... = *insert* (*insert* {*new-el*} (*set P*)) ((*insert-into-member new-el* (*set P*)) ‘ *set P*)
**using** *assms insert-into-member-list-alt* **by** (*metis comp-apply*)
**finally show** *?thesis* **unfolding** *coarser-partitions-with-def* **.**
**qed**

Any member of the set of coarser partitions of a given partition, obtained by inserting a given fresh element into each of its equivalence classes, actually is a partition.

**lemma** *partition-extension3*:
**fixes** *elem*::′*a*
**and** *P*::′*a set set*
**and** *Q*::′*a set set*
**assumes** *P-partition*: *is-partition P*
**and** *new-elem*: *elem* ∉ ⋃ *P*
**and** *Q-coarser*: *Q* ∈ *coarser-partitions-with elem P*
**shows** *is-partition Q*
**proof** −
**let** *?q* = *insert* {*elem*} *P*
**have** *Q-coarser-unfolded*: *Q* ∈ *insert ?q* (*insert-into-member elem P* ‘ *P*)
**using** *Q-coarser*
**unfolding** *coarser-partitions-with-def*
**by** *fast*
**show** *?thesis*
**proof** (*cases Q* = *?q*)
**case** *True*
**then show** *?thesis*
**using** *P-partition new-elem partition-extension1*
**by** *fastforce*
**next**
**case** *False*
**then have** *Q* ∈ (*insert-into-member elem P*) ‘ *P* **using** *Q-coarser-unfolded* **by** *fastforce*
**then show** *?thesis* **using** *partition-extension2 P-partition new-elem* **by** *fast*

**qed**
**qed**

Let $P$ be a partition of a set $S$, and *elem* an element (which may or may not be in $S$ already). Then, any member of *coarser-partitions-with elem P* is a set of sets whose union is $S \cup \{elem\}$, i.e. it satisfies a necessary criterion for being a partition of $S \cup \{elem\}$.

**lemma** *coarser-partitions-covers*:
  **fixes** *elem*::*'a*
    **and** *P*::*'a set set*
    **and** *Q*::*'a set set*
  **assumes** $Q \in$ *coarser-partitions-with elem P*
  **shows** $\bigcup Q =$ *insert elem* $(\bigcup P)$
**proof** −
  **let** *?S* $= \bigcup P$
  **have** *Q-cases*: $Q \in$ (*insert-into-member elem P*) ' $P \vee Q =$ *insert* {*elem*} $P$
    **using** *assms* **unfolding** *coarser-partitions-with-def* **by** *fast*
  **{**
    **fix** *eq-class* **assume** *eq-class-in-P*: *eq-class* $\in P$
   **have** $\bigcup$ *insert* (*eq-class* $\cup$ {*elem*}) ($P$ − {*eq-class*}) = *?S* $\cup$ (*eq-class* $\cup$ {*elem*})
      **using** *insert-into-member-partition1*
     **by** (*metis Sup-insert Un-commute Un-empty-right Un-insert-right insert-Diff-single*)
    **with** *eq-class-in-P* **have** $\bigcup$ *insert* (*eq-class* $\cup$ {*elem*}) ($P$ − {*eq-class*}) = *?S*
$\cup$ {*elem*} **by** *blast*
    **then have** $\bigcup$ *insert-into-member elem P eq-class* = *?S* $\cup$ {*elem*}
      **using** *insert-into-member-partition1*
      **by** (*rule subst*)
  **}**
  **then show** *?thesis* **using** *Q-cases* **by** *blast*
**qed**

Removes the element *elem* from every set in $P$, and removes from $P$ any remaining empty sets. This function is intended to be applied to partitions, i.e. *elem* occurs in at most one set. *partition-without e* reverses *coarser-partitions-with e*. *coarser-partitions-with* is one-to-many, while this is one-to-one, so we can think of a tree relation, where coarser partitions of a set $S \cup \{elem\}$ are child nodes of one partition of $S$.

**definition** *partition-without* :: *'a* ⇒ *'a set set* ⇒ *'a set set*
**where** *partition-without elem P* = ($\lambda X$ . $X$ − {*elem*}) ' $P$ − {{}}

alternative characterisation of the set partitioned by the partition obtained by removing an element from a given partition using *partition-without*

**lemma** *partition-without-covers*:
  **fixes** *elem*::*'a*
    **and** *P*::*'a set set*
  **shows** $\bigcup$ *partition-without elem P* = $\bigcup P$ − {*elem*}
**proof** −
  **have** $\bigcup$ *partition-without elem P* = $\bigcup$ (($\lambda x$ . $x$ − {*elem*}) ' $P$ − {{}})

**unfolding** *partition-without-def* **by** *fast*
**also have** ... = $\bigcup$ $P$ − {*elem*} **by** *blast*
**finally show** *?thesis* **.**
**qed**

Any equivalence class of the partition obtained by removing an element *elem* from an original partition $P$ using *partition-without* equals some equivalence class of $P$, reduced by *elem*.

**lemma** *super-eq-class*:
**assumes** $X \in$ *partition-without elem P*
**obtains** $Z$ **where** $Z \in P$ **and** $X = Z - \{elem\}$
**proof** −
**from** *assms* **have** $X \in (\lambda X . X - \{elem\})$ ‘ $P - \{\{\}\}$ **unfolding** *partition-without-def*
**.**
**then obtain** $Z$ **where** *Z-in-P*: $Z \in P$ **and** *Z-sup*: $X = Z - \{elem\}$
**by** (*metis* (*lifting*) *Diff-iff image-iff*)
**then show** *?thesis* **..**
**qed**

The set of sets obtained by removing an element from a partition actually is another partition.

**lemma** *partition-without-is-partition*:
**fixes** *elem*::′*a*
**and** *P*::′*a set set*
**assumes** *is-partition P*
**shows** *is-partition* (*partition-without elem P*) (**is** *is-partition ?Q*)
**proof** −
**have** $\forall$ *X1* $\in$ *?Q*. $\forall$ *X2* $\in$ *?Q*. *X1* $\cap$ *X2* $\neq$ {} $\longleftrightarrow$ *X1* = *X2*
**proof**
**fix** *X1* **assume** *X1-in-Q*: *X1* $\in$ *?Q*
**then obtain** *Z1* **where** *Z1-in-P*: *Z1* $\in$ $P$ **and** *Z1-sup*: *X1* = *Z1* − {*elem*}
**by** (*rule super-eq-class*)
**have** *X1-non-empty*: *X1* $\neq$ {} **using** *X1-in-Q partition-without-def* **by** *fast*
**show** $\forall$ *X2* $\in$ *?Q*. *X1* $\cap$ *X2* $\neq$ {} $\longleftrightarrow$ *X1* = *X2*
**proof**
**fix** *X2* **assume** *X2* $\in$ *?Q*
**then obtain** *Z2* **where** *Z2-in-P*: *Z2* $\in$ $P$ **and** *Z2-sup*: *X2* = *Z2* − {*elem*}
**by** (*rule super-eq-class*)
**have** *X1* $\cap$ *X2* $\neq$ {} $\longrightarrow$ *X1* = *X2*
**proof**
**assume** *X1* $\cap$ *X2* $\neq$ {}
**then have** *Z1* $\cap$ *Z2* $\neq$ {} **using** *Z1-sup Z2-sup* **by** *fast*
**then have** *Z1* = *Z2* **using** *Z1-in-P Z2-in-P assms* **unfolding** *is-partition-def*
**by** *fast*
**then show** *X1* = *X2* **using** *Z1-sup Z2-sup* **by** *fast*
**qed**
**moreover have** *X1* = *X2* $\longrightarrow$ *X1* $\cap$ *X2* $\neq$ {} **using** *X1-non-empty* **by** *auto*
**ultimately show** (*X1* $\cap$ *X2* $\neq$ {}) $\longleftrightarrow$ *X1* = *X2* **by** *blast*
**qed**

**qed**
**then show** *?thesis* **unfolding** *is-partition-def* **.**
**qed**

*coarser-partitions-with elem* is the "inverse" of *partition-without elem*.

**lemma** *coarser-partitions-inv-without*:
**fixes** *elem*::'*a*
**and** *P*::'*a set set*
**assumes** *partition*: *is-partition P*
**and** *elem*: *elem* $\in \bigcup$ *P*
**shows** *P* $\in$ *coarser-partitions-with elem* (*partition-without elem P*)
(**is** *P* $\in$ *coarser-partitions-with elem ?Q*)
**proof** −
**let** *?remove-elem* = $\lambda X$ . $X - \{elem\}$
**obtain** *Y*
**where** *elem-eq-class*: *elem* $\in$ *Y* **and** *elem-eq-class'*: *Y* $\in$ *P* **using** *elem* **..**
**let** *?elem-neq-classes* = *P* − {*Y*}
**have** *P-wrt-elem*: *P* = *?elem-neq-classes* ∪ {*Y*} **using** *elem-eq-class'* **by** *blast*
**let** *?elem-eq* = *Y* − {*elem*}
**have** *Y-elem-eq*: *?remove-elem* ‘ {*Y*} = {*?elem-eq*} **by** *fast*

**have** *elem-neq-classes-part*: *is-partition ?elem-neq-classes*
**using** *subset-is-partition partition*
**by** *blast*
**have** *elem-eq-wrt-P*: *?elem-eq* ∈ *?remove-elem* ‘ *P* **using** *elem-eq-class'* **by** *blast*

{
**fix** *W* **assume** *W-eq-class*: *W* ∈ *?elem-neq-classes*
**then have** *elem* ∉ *W*
**using** *elem-eq-class elem-eq-class' partition is-partition-def*
**by** *fast*
**then have** *?remove-elem W* = *W* **by** *simp*
}
**then have** *elem-neq-classes-id*: *?remove-elem* ‘ *?elem-neq-classes* = *?elem-neq-classes*
**by** *fastforce*

**have** *Q-unfolded*: *?Q* = *?remove-elem* ‘ *P* − {{}}
**unfolding** *partition-without-def*
**using** *image-Collect-mem*
**by** *blast*
**also have** ... = *?remove-elem* ‘ (*?elem-neq-classes* ∪ {*Y*}) − {{}} **using**
*P-wrt-elem* **by** *presburger*
**also have** ... = *?elem-neq-classes* ∪ {*?elem-eq*} − {{}}
**using** *Y-elem-eq elem-neq-classes-id image-Un* **by** *metis*
**finally have** *Q-wrt-elem*: *?Q* = *?elem-neq-classes* ∪ {*?elem-eq*} − {{}} **.**

**have** *?elem-eq* = {} ∨ *?elem-eq* ∉ *P*
**using** *elem-eq-class elem-eq-class' partition Diff-Int-distrib2 Diff-iff empty-Diff*
*insert-iff*

**unfolding** *is-partition-def* **by** *metis*
  **then have** *?elem-eq* $\notin$ *P*
    **using** *partition no-empty-eq-class*
    **by** *metis*
  **then have** *elem-neq-classes*: *?elem-neq-classes* $-$ {*?elem-eq*} $=$ *?elem-neq-classes* **by** *fastforce*

  **show** *?thesis*
  **proof** *cases*
    **assume** *?elem-eq* $\notin$ *?Q*
    **then have** *?elem-eq* $\in$ {{}}
      **using** *elem-eq-wrt-P Q-unfolded*
      **by** (*metis DiffI*)
    **then have** *Y-singleton*: $Y$ $=$ {*elem*} **using** *elem-eq-class* **by** *fast*
    **then have** *?Q* $=$ *?elem-neq-classes* $-$ {{}}
      **using** *Q-wrt-elem*
      **by** *force*
    **then have** *?Q* $=$ *?elem-neq-classes*
      **using** *no-empty-eq-class elem-neq-classes-part*
      **by** *blast*
    **then have** *insert* {*elem*} *?Q* $=$ *P*
      **using** *Y-singleton elem-eq-class'*
      **by** *fast*
    **then show** *?thesis* **unfolding** *coarser-partitions-with-def* **by** *auto*
  **next**
    **assume** *True*: $\neg$ *?elem-eq* $\notin$ *?Q*
     **hence** *Y'*: *?elem-neq-classes* $\cup$ {*?elem-eq*} $-$ {{}} $=$ *?elem-neq-classes* $\cup$ {*?elem-eq*}
      **using** *no-empty-eq-class partition partition-without-is-partition*
      **by** *force*
     **have** *insert-into-member elem* ({*?elem-eq*} $\cup$ *?elem-neq-classes*) *?elem-eq* $=$ *insert* (*?elem-eq* $\cup$ {*elem*}) (({*?elem-eq*} $\cup$ *?elem-neq-classes*) $-$ {*?elem-eq*})
      **unfolding** *insert-into-member-def* **..**
      **also have** $\ldots$ $=$ ({} $\cup$ *?elem-neq-classes*) $\cup$ {*?elem-eq* $\cup$ {*elem*}} **using** *elem-neq-classes* **by** *force*
    **also have** $\ldots$ $=$ *?elem-neq-classes* $\cup$ {$Y$} **using** *elem-eq-class* **by** *blast*
   **finally have** *insert-into-member elem* ({*?elem-eq*} $\cup$ *?elem-neq-classes*) *?elem-eq* $=$ *?elem-neq-classes* $\cup$ {$Y$} **.**
    **then have** *?elem-neq-classes* $\cup$ {$Y$} $=$ *insert-into-member elem ?Q ?elem-eq*
      **using** *Q-wrt-elem Y' partition-without-def*
      **by** *force*
   **then have** {$Y$} $\cup$ *?elem-neq-classes* $\in$ *insert-into-member elem ?Q* ` *?Q* **using** *True* **by** *blast*
    **then have** {$Y$} $\cup$ *?elem-neq-classes* $\in$ *coarser-partitions-with elem ?Q* **unfolding** *coarser-partitions-with-def* **by** *simp*
    **then show** *?thesis* **using** *P-wrt-elem* **by** *simp*
  **qed**
**qed**

Given a set $Ps$ of partitions, this is intended to compute the set of all coarser partitions (given an extension element) of all partitions in *Ps*.

**definition** *all-coarser-partitions-with* :: *'a ⇒ 'a set set set ⇒ 'a set set set*
**where** *all-coarser-partitions-with elem Ps* = $\bigcup$ (*coarser-partitions-with elem* ‘ *Ps*)

the list variant of *all-coarser-partitions-with*

**definition** *all-coarser-partitions-with-list* :: *'a ⇒ 'a set list list ⇒ 'a set list list*
**where** *all-coarser-partitions-with-list elem Ps* = *concat* (*map* (*coarser-partitions-with-list elem*) *Ps*)

*all-coarser-partitions-with-list* and *all-coarser-partitions-with* are equivalent.

**lemma** *all-coarser-partitions-with-list-alt*:
  **fixes** *elem*::*'a*
    **and** *Ps*::*'a set list list*
  **assumes** *distinct*: ∀ *P* ∈ *set Ps* . *distinct P*
  **shows** *set* (*map set* (*all-coarser-partitions-with-list elem Ps*)) = *all-coarser-partitions-with elem* (*set* (*map set Ps*))
    (**is** *?list-expr* = *?set-expr*)
**proof** −
  **have** *?list-expr* = *set* (*map set* (*concat* (*map* (*coarser-partitions-with-list elem*) *Ps*)))
    **unfolding** *all-coarser-partitions-with-list-def* **..**
  **also have** ... = *set* ‘ ($\bigcup$ *x* ∈ (*coarser-partitions-with-list elem*) ‘ (*set Ps*) . *set x*) **by** *simp*

  **also have** ... = *set* ‘ ($\bigcup$ *x* ∈ { *coarser-partitions-with-list elem P* | *P* . *P* ∈ *set Ps* } . *set x*)
    **by** (*simp add*: *image-Collect-mem*)
  **also have** ... = $\bigcup$ { *set* (*map set* (*coarser-partitions-with-list elem P*)) | *P* . *P* ∈ *set Ps* } **by** *auto*
  **also have** ... = $\bigcup$ { *coarser-partitions-with elem* (*set P*) | *P* . *P* ∈ *set Ps* }
    **using** *distinct coarser-partitions-with-list-alt* **by** *fast*
  **also have** ... = $\bigcup$ (*coarser-partitions-with elem* ‘ (*set* ‘ (*set Ps*))) **by** (*simp add*: *image-Collect-mem*)
  **also have** ... = $\bigcup$ (*coarser-partitions-with elem* ‘ (*set* (*map set Ps*))) **by** *simp*
  **also have** ... = *?set-expr* **unfolding** *all-coarser-partitions-with-def* **..**
  **finally show** *?thesis* **.**
**qed**

all partitions of a set (given as list)

**fun** *all-partitions-set* :: *'a list ⇒ 'a set set set*
**where**
*all-partitions-set* [] = {{}} |
*all-partitions-set* (*e* # *X*) = *all-coarser-partitions-with e* (*all-partitions-set X*)

all partitions of a set (given as list)

**fun** *all-partitions-list* :: *'a list ⇒ 'a set list list*
**where**
*all-partitions-list* [] = [[]] |

*all-partitions-list* $(e \# X) =$ *all-coarser-partitions-with-list e* $($*all-partitions-list* $X)$

A list of partitions coarser than a given partition in list representation (constructed with *coarser-partitions-with* is distinct under certain conditions.

**lemma** *coarser-partitions-with-list-distinct*:
  **fixes** *ps*
  **assumes** *ps-coarser*: *ps* $\in$ *set* (*coarser-partitions-with-list x Q*)
    **and** *distinct*: *distinct Q*
    **and** *partition*: *is-partition* (*set Q*)
    **and** *new*: $\{x\} \notin$ *set Q*
  **shows** *distinct ps*
**proof** −
 **have** *set* (*coarser-partitions-with-list x Q*) = *insert* ($\{x\}$ # *Q*) (*set* (*map* (*insert-into-member-list x Q*) *Q*))
   **unfolding** *coarser-partitions-with-list-def* **by** *simp*
 **with** *ps-coarser* **have** *ps* $\in$ *insert* ($\{x\}$ # *Q*) (*set* (*map* ((*insert-into-member-list x Q*)) *Q*)) **by** *blast*
 **then show** *?thesis*
 **proof**
   **assume** *ps* = $\{x\}$ # *Q*
   **with** *distinct* **and** *new* **show** *?thesis* **by** *simp*
 **next**
   **assume** *ps* $\in$ *set* (*map* (*insert-into-member-list x Q*) *Q*)
   **then obtain** *X* **where** *X-in-Q*: *X* $\in$ *set Q* **and** *ps-insert*: *ps* = *insert-into-member-list x Q X* **by** *auto*
   **from** *ps-insert* **have** *ps* = ($X \cup \{x\}$) # (*remove1 X Q*) **unfolding** *insert-into-member-list-def* **.**
    **also have** ... = ($X \cup \{x\}$) # (*removeAll X Q*) **using** *distinct* **by** (*metis distinct-remove1-removeAll*)
   **finally have** *ps-list*: *ps* = ($X \cup \{x\}$) # (*removeAll X Q*) **.**

   **have** *distinct-tl*: $X \cup \{x\} \notin$ *set* (*removeAll X Q*)
   **proof**
     **from** *partition* **have** *partition'*: $\forall x \in$ *set Q*. $\forall y \in$ *set Q*. $(x \cap y \neq \{\}) = (x = y)$ **unfolding** *is-partition-def* **.**
     **assume** $X \cup \{x\} \in$ *set* (*removeAll X Q*)
    **with** *X-in-Q partition* **show** *False* **by** (*metis partition' inf-sup-absorb member-remove no-empty-eq-class remove-code*(*1*))
   **qed**
    **with** *ps-list distinct* **show** *?thesis* **by** (*metis* (*full-types*) *distinct.simps*(*2*) *distinct-removeAll*)
 **qed**
**qed**

The paper-like definition *all-partitions* and the algorithmic definition *all-partitions-list* are equivalent.

**lemma** *all-partitions-paper-equiv-alg'*:
  **fixes** *xs*::*'a list*

**shows** *distinct xs* $\Longrightarrow$ ((*set* (*map set* (*all-partitions-list xs*)) = *all-partitions* (*set xs*)) $\wedge$ ($\forall$ *ps* $\in$ *set* (*all-partitions-list xs*) . *distinct ps*))
**proof** (*induct xs*)
  **case** *Nil*
  **have** *set* (*map set* (*all-partitions-list* [])) = *all-partitions* (*set* [])
    **unfolding** *List.set-simps*(*1*) *emptyset-part-emptyset3* **by** *simp*

  **moreover have** $\forall$ *ps* $\in$ *set* (*all-partitions-list* []) . *distinct ps* **by** *fastforce*
  **ultimately show** *?case* **..**
**next**
  **case** (*Cons x xs*)
  **from** *Cons.prems Cons.hyps*
    **have** *hyp-equiv*: *set* (*map set* (*all-partitions-list xs*)) = *all-partitions* (*set xs*) **by** *simp*
  **from** *Cons.prems Cons.hyps*
    **have** *hyp-distinct*: $\forall$ *ps* $\in$ *set* (*all-partitions-list xs*) . *distinct ps* **by** *simp*

  **have** *distinct-xs*: *distinct xs* **using** *Cons.prems* **by** *simp*
  **have** *x-notin-xs*: *x* $\notin$ *set xs* **using** *Cons.prems* **by** *simp*

  **have** *set* (*map set* (*all-partitions-list* (*x* # *xs*))) = *all-partitions* (*set* (*x* # *xs*))
  **proof** (*rule equalitySubsetI*)
    **fix** *P*::*'a set set*
    **let** *?P-without-x* = *partition-without x P*
    **have** *P-partitions-exc-x*: $\bigcup$ *?P-without-x* = $\bigcup$ *P* $-$ {*x*} **using** *partition-without-covers* **.**

    **assume** *P* $\in$ *all-partitions* (*set* (*x* # *xs*))
    **then have** *is-partition-of*: *is-partition-of P* (*set* (*x* # *xs*)) **unfolding** *all-partitions-def* **..**
    **then have** *is-partition*: *is-partition P* **unfolding** *is-partition-of-def* **by** *simp*
    **from** *is-partition-of* **have** *P-covers*: $\bigcup$ *P* = *set* (*x* # *xs*) **unfolding** *is-partition-of-def* **by** *simp*

    **have** *is-partition-of ?P-without-x* (*set xs*)
      **unfolding** *is-partition-of-def*
    **using** *is-partition partition-without-is-partition partition-without-covers P-covers x-notin-xs*
      **by** (*metis Diff-insert-absorb List.set-simps*(*2*))
    **with** *hyp-equiv* **have** *p-list*: *?P-without-x* $\in$ *set* (*map set* (*all-partitions-list xs*))
      **unfolding** *all-partitions-def* **by** *fast*
    **have** *P* $\in$ *coarser-partitions-with x ?P-without-x*
      **using** *coarser-partitions-inv-without is-partition P-covers*
      **by** (*metis List.set-simps*(*2*) *insertI1*)
    **then have** *P* $\in$ $\bigcup$ (*coarser-partitions-with x* ` *set* (*map set* (*all-partitions-list xs*)))
      **using** *p-list* **by** *blast*
    **then have** *P* $\in$ *all-coarser-partitions-with x* (*set* (*map set* (*all-partitions-list xs*)))

**unfolding** *all-coarser-partitions-with-def* **by** *fast*
**then show** $P \in$ *set* (*map set* (*all-partitions-list* ($x$ # *xs*)))
**using** *all-coarser-partitions-with-list-alt hyp-distinct*
**by** (*metis all-partitions-list.simps*(*2*))
**next**
**fix** $P$::'*a set set*
**assume** $P$: $P \in$ *set* (*map set* (*all-partitions-list* ($x$ # *xs*)))

**have** *set* (*map set* (*all-partitions-list* ($x$ # *xs*))) = *set* (*map set* (*all-coarser-partitions-with-list* $x$ (*all-partitions-list xs*)))
**by** *simp*
**also have** ... = *all-coarser-partitions-with* $x$ (*set* (*map set* (*all-partitions-list* *xs*)))
**using** *distinct-xs hyp-distinct all-coarser-partitions-with-list-alt* **by** *fast*
**also have** ... = *all-coarser-partitions-with* $x$ (*all-partitions* (*set xs*))
**using** *distinct-xs hyp-equiv* **by** *auto*
**finally have** *P-set*: *set* (*map set* (*all-partitions-list* ($x$ # *xs*))) = *all-coarser-partitions-with* $x$ (*all-partitions* (*set xs*)) **.**

**with** $P$ **have** $P \in$ *all-coarser-partitions-with* $x$ (*all-partitions* (*set xs*)) **by** *fast*
**then have** $P \in \bigcup$ (*coarser-partitions-with* $x$ ‘ (*all-partitions* (*set xs*)))
**unfolding** *all-coarser-partitions-with-def* **.**
**then obtain** $Y$
**where** *P-in-Y*: $P \in Y$
**and** *Y-coarser*: $Y \in$ *coarser-partitions-with* $x$ ‘ (*all-partitions* (*set xs*)) **..**
**from** *Y-coarser* **obtain** $Q$
**where** *Q-part-xs*: $Q \in$ *all-partitions* (*set xs*)
**and** *Y-coarser'*: $Y$ = *coarser-partitions-with* $x$ $Q$ **..**
**from** *P-in-Y Y-coarser'* **have** *P-wrt-Q*: $P \in$ *coarser-partitions-with* $x$ $Q$ **by** *fast*
**then have** $Q \in$ *all-partitions* (*set xs*) **using** *Q-part-xs* **by** *simp*
**then have** *is-partition-of* $Q$ (*set xs*) **unfolding** *all-partitions-def* **..**
**then have** *is-partition* $Q$ **and** *Q-covers*: $\bigcup$ $Q$ = *set xs*
**unfolding** *is-partition-of-def* **by** *simp-all*
**then have** *P-partition*: *is-partition* $P$
**using** *partition-extension3 P-wrt-Q x-notin-xs* **by** *fast*
**have** $\bigcup$ $P$ = *set xs* $\cup$ $\{x\}$
**using** *Q-covers P-in-Y Y-coarser' coarser-partitions-covers* **by** *fast*
**then have** $\bigcup$ $P$ = *set* ($x$ # *xs*)
**using** *x-notin-xs P-wrt-Q Q-covers*
**by** (*metis List.set-simps*(*2*) *insert-is-Un sup-commute*)
**then have** *is-partition-of* $P$ (*set* ($x$ # *xs*))
**using** *P-partition* **unfolding** *is-partition-of-def* **by** *blast*
**then show** $P \in$ *all-partitions* (*set* ($x$ # *xs*)) **unfolding** *all-partitions-def* **..**
**qed**
**moreover have** $\forall$ *ps* $\in$ *set* (*all-partitions-list* ($x$ # *xs*)) . *distinct ps*
**proof**
**fix** *ps*::'*a set list* **assume** *ps-part*: *ps* $\in$ *set* (*all-partitions-list* ($x$ # *xs*))

**have** *set* (*all-partitions-list* ($x$ # *xs*)) = *set* (*all-coarser-partitions-with-list* $x$ (*all-partitions-list xs*))
by *simp*
**also have** … = *set* (*concat* (*map* (*coarser-partitions-with-list* $x$) (*all-partitions-list xs*)))
**unfolding** *all-coarser-partitions-with-list-def* ..
**also have** … = $\bigcup$ ((*set* $\circ$ (*coarser-partitions-with-list* $x$)) ' (*set* (*all-partitions-list xs*)))
**by** *simp*
**finally have** *all-parts-unfolded*: *set* (*all-partitions-list* ($x$ # *xs*)) = $\bigcup$ ((*set* $\circ$ (*coarser-partitions-with-list* $x$)) ' (*set* (*all-partitions-list xs*))) .

**with** *ps-part* **obtain** *qs*
**where** *qs*: *qs* $\in$ *set* (*all-partitions-list xs*)
**and** *ps-coarser*: *ps* $\in$ *set* (*coarser-partitions-with-list x qs*)
**using** *UnionE comp-def imageE* **by** *auto*

**from** *qs* **have** *set qs* $\in$ *set* (*map set* (*all-partitions-list* (*xs*))) **by** *simp*
**with** *distinct-xs hyp-equiv* **have** *qs-hyp*: *set qs* $\in$ *all-partitions* (*set xs*) **by** *fast*
**then have** *qs-part*: *is-partition* (*set qs*)
**using** *all-partitions-def is-partition-of-def*
**by** (*metis mem-Collect-eq*)
**then have** *distinct-qs*: *distinct qs*
**using** *qs distinct-xs hyp-distinct* **by** *fast*

**from** *Cons.prems* **have** $x \notin$ *set xs* **by** *simp*
**then have** *new*: $\{x\} \notin$ *set qs*
**using** *qs-hyp*
**unfolding** *all-partitions-def is-partition-of-def*
**by** (*metis* (*lifting*, *mono-tags*) *UnionI insertI1 mem-Collect-eq*)

**from** *ps-coarser distinct-qs qs-part new*
**show** *distinct ps* **by** (*rule coarser-partitions-with-list-distinct*)
**qed**
**ultimately show** *?case* ..
**qed**

The paper-like definition *all-partitions* and the algorithmic definition *all-partitions-list* are equivalent. This is a frontend theorem derived from *distinct ?xs* $\Longrightarrow$ *set* (*map set* (*all-partitions-list ?xs*)) = *all-partitions* (*set ?xs*) $\wedge$ ($\forall$ *ps*$\in$*set* (*all-partitions-list ?xs*). *distinct ps*); it does not make the auxiliary statement about partitions being distinct lists.

**theorem** *all-partitions-paper-equiv-alg*:
**fixes** *xs*::'*a list*
**shows** *distinct xs* $\Longrightarrow$ *set* (*map set* (*all-partitions-list xs*)) = *all-partitions* (*set xs*)
**using** *all-partitions-paper-equiv-alg'* **by** *blast*

The function that we will be using in practice to compute all partitions of a set, a set-oriented frontend to *all-partitions-list*

**definition** *all-partitions-alg* :: $'a$::*linorder set* $\Rightarrow$ $'a$ *set list list*
**where** *all-partitions-alg* $X$ = *all-partitions-list* (*sorted-list-of-set* $X$)

**corollary** *mm90*[*code-unfold*]:
  **fixes** $X$
  **assumes** *finite* $X$
  **shows** *all-partitions* $X$ = *set* (*map set* (*all-partitions-alg* $X$))
    **unfolding** *all-partitions-alg-def*
  **using** *assms* **by** (*metis all-partitions-paper-equiv-alg′ sorted-list-of-set*)

**lemma** *remove-singleton-eq-class-from-part*:
  **assumes** *singleton-eq-class*: $\{X\} \subseteq P$
      **and** *part*: *is-partition* $P$
  **shows** $(P - \{X\}) \cap \{Y \cup X\} = \{\}$
    **using** *assms* **unfolding** *is-partition-def*
   **by** (*metis Diff-disjoint Diff-iff Int-absorb2 Int-insert-right-if0 Un-upper2 empty-Diff insert-subset subset-refl*)

If new elements are added to a set, for any partition $P$ of the original set, we can obtain a partition $Q$ of the enlarged set by adding the new elements as a new equivalence class, and each equivalence class in $P$ is a subset of one equivalence class in $Q$.

**lemma** *exists-partition-of-strictly-larger-set*:
  **assumes** *part*: $P$ *partitions* $A$
      **and** *new*: $B \cap A = \{\}$
      **and** *non-empty*: $B \neq \{\}$
  **shows** $(P \cup \{B\})$ *partitions* $(A \cup B) \wedge (\forall\ X \in P\ .\ \exists\ Y \in P \cup \{B\}\ .\ X \subseteq Y)$
**proof**
  **show** $(P \cup \{B\})$ *partitions* $(A \cup B)$
    **unfolding** *is-partition-of-def is-partition-def*
  **proof**
    **from** *part* **have** $\bigcup\ P = A$ **unfolding** *is-partition-of-def* **..**

    **show** $\bigcup\ (P \cup \{B\}) = A \cup B$
    **proof** −
      **from** *part* **have** $\bigcup\ P = A$ **unfolding** *is-partition-of-def* **..**
      **then show** *?thesis* **by** *auto*
    **qed**
    **show** $\forall\ X \in P \cup \{B\}\ .\ \forall\ Y \in P \cup \{B\}\ .\ (X \cap Y \neq \{\} \longleftrightarrow X = Y)$
    **proof**
      **fix** $X$ **assume** *X-class*: $X \in P \cup \{B\}$
      **show** $\forall\ Y \in P \cup \{B\}\ .\ (X \cap Y \neq \{\} \longleftrightarrow X = Y)$
**by** (*metis Un-insert-right X-class assms*(*1*) *assms*(*2*) *assms*(*3*) *is-partition-def*

*is-partition-of-def partition-extension1 sup-bot.right-neutral*)

**qed**
**qed**
**show** $\forall\ X \in P\ .\ \exists\ Y \in P \cup \{B\}\ .\ X \subseteq Y$
**proof**
**fix** $X$ **assume** $X \in P$
**then have** $X \in P \cup \{B\}$ **by** (*rule UnI1*)
**then show** $\exists\ Y \in P \cup \{B\}\ .\ X \subseteq Y$ **by** *blast*
**qed**
**qed**

If zero or more new elements are added to a set, one can obtain for any partition $P$ of the original set a partition $Q$ of the enlarged set such that each equivalence class in $P$ is a subset of one equivalence class in $Q$.

**lemma** *exists-partition-of-larger-set*:
**assumes** *part*: $P$ *partitions* $A$
**and** *new*: $B \cap A = \{\}$
**shows** $\exists\ Q\ .\ Q$ *partitions* $(A \cup B) \wedge (\forall\ X \in P\ .\ \exists\ Y \in Q\ .\ X \subseteq Y)$
**proof** *cases*
**assume** $B = \{\}$
**with** *part* **have** $P$ *partitions* $(A \cup B) \wedge (\forall\ X \in P\ .\ \exists\ Y \in P\ .\ X \subseteq Y)$
**unfolding** *is-partition-of-def* **by** *auto*
**then show** *?thesis* **by** *fast*
**next**
**assume** *non-empty*: $B \neq \{\}$
**with** *part new* **have** $(P \cup \{B\})$ *partitions* $(A \cup B) \wedge (\forall\ X \in P\ .\ \exists\ Y \in P \cup \{B\}\ .\ X \subseteq Y)$
**by** (*rule exists-partition-of-strictly-larger-set*)
**then show** *?thesis* **by** *blast*
**qed**

**end**

# 8 Avoidance of pattern matching on natural numbers

**theory** *Code-Abstract-Nat*
**imports** *Main*
**begin**

When natural numbers are implemented in another than the conventional inductive *0/Suc* representation, it is necessary to avoid all pattern matching on natural numbers altogether. This is accomplished by this theory (up to a certain extent).

## 8.1 Case analysis

Case analysis on natural numbers is rephrased using a conditional expression:

**lemma** [*code*, *code-unfold*]:
  *case-nat* = ($\lambda$*f g n*. *if n* = *0 then f else g* (*n* − *1*))
  **by** (*auto simp add*: *fun-eq-iff dest*!: *gr0-implies-Suc*)

## 8.2 Preprocessors

The term *Suc n* is no longer a valid pattern. Therefore, all occurrences of this term in a position where a pattern is expected (i.e. on the left-hand side of a code equation) must be eliminated. This can be accomplished – as far as possible – by applying the following transformation rule:

**lemma** *Suc-if-eq*:
  **assumes** $\bigwedge$*n*. *f* (*Suc n*) ≡ *h n*
  **assumes** *f 0* ≡ *g*
  **shows** *f n* ≡ *if n* = *0 then g else h* (*n* − *1*)
  **by** (*rule eq-reflection*) (*cases n*, *insert assms*, *simp-all*)

The rule above is built into a preprocessor that is plugged into the code generator.

**setup** ‹‹

```
let

val Suc-if-eq = Thm.incr-indexes 1 @{thm Suc-if-eq};

fun remove-suc ctxt thms =
  let
    val thy = Proof-Context.theory-of ctxt;
    val vname = singleton (Name.variant-list (map fst
      (fold (Term.add-var-names o Thm.full-prop-of) thms []))) n;
    val cv = cterm-of thy (Var ((vname, 0), HOLogic.natT));
    val lhs-of = snd o Thm.dest-comb o fst o Thm.dest-comb o cprop-of;
    val rhs-of = snd o Thm.dest-comb o cprop-of;
    fun find-vars ct = (case term-of ct of
       (Const (@{const-name Suc}, -) $ Var -) => [(cv, snd (Thm.dest-comb ct))]
      | - $ - =>
        let val (ct1, ct2) = Thm.dest-comb ct
        in
          map (apfst (fn ct => Thm.apply ct ct2)) (find-vars ct1) @
          map (apfst (Thm.apply ct1)) (find-vars ct2)
        end
      | - => []);
    val eqs = maps
      (fn thm => map (pair thm) (find-vars (lhs-of thm))) thms;
    fun mk-thms (thm, (ct, cv')) =
      let
```

```
        val thm' =
          Thm.implies-elim
           (Conv.fconv-rule (Thm.beta-conversion true)
             (Drule.instantiate'
               [SOME (ctyp-of-term ct)] [SOME (Thm.lambda cv ct),
                 SOME (Thm.lambda cv' (rhs-of thm)), NONE, SOME cv']
               Suc-if-eq)) (Thm.forall-intr cv' thm)
      in
        case map-filter (fn thm'' =>
            SOME (thm'', singleton
              (Variable.trade (K (fn [thm'''] => [thm''' RS thm']))
                (Variable.global-thm-context thm'')) thm'')
          handle THM - => NONE) thms of
            [] => NONE
          | thmps =>
              let val (thms1, thms2) = split-list thmps
              in SOME (subtract Thm.eq-thm (thm :: thms1) thms @ thms2) end
      end
  in get-first mk-thms eqs end;

fun eqn-suc-base-preproc thy thms =
  let
    val dest = fst o Logic.dest-equals o prop-of;
    val contains-suc = exists-Const (fn (c, -) => c = @{const-name Suc});
  in
    if forall (can dest) thms andalso exists (contains-suc o dest) thms
    then thms |> perhaps-loop (remove-suc thy) |> (Option.map o map) Drule.zero-var-indexes
       else NONE
  end;

val eqn-suc-preproc = Code-Preproc.simple-functrans eqn-suc-base-preproc;

in

  Code-Preproc.add-functrans (eqn-Suc, eqn-suc-preproc)

end;
⟩⟩
```

**end**

# 9 Implementation of natural numbers by target-language integers

**theory** *Code-Target-Nat*
**imports** *Code-Abstract-Nat*
**begin**

## 9.1 Implementation for *nat*

**context**
**includes** *natural.lifting integer.lifting*
**begin**

**lift-definition** *Nat* :: *integer* ⇒ *nat*
  **is** *nat*
  .

**lemma** [*code-post*]:
  *Nat 0* = *0*
  *Nat 1* = *1*
  *Nat* (*numeral k*) = *numeral k*
  **by** (*transfer*, *simp*)+

**lemma** [*code-abbrev*]:
  *integer-of-nat* = *of-nat*
  **by** *transfer rule*

**lemma** [*code-unfold*]:
  *Int.nat* (*int-of-integer k*) = *nat-of-integer k*
  **by** *transfer rule*

**lemma** [*code abstype*]:
  *Code-Target-Nat.Nat* (*integer-of-nat n*) = *n*
  **by** *transfer simp*

**lemma** [*code abstract*]:
  *integer-of-nat* (*nat-of-integer k*) = *max 0 k*
  **by** *transfer auto*

**lemma** [*code-abbrev*]:
  *nat-of-integer* (*numeral k*) = *nat-of-num k*
  **by** *transfer* (*simp add*: *nat-of-num-numeral*)

**lemma** [*code abstract*]:
  *integer-of-nat* (*nat-of-num n*) = *integer-of-num n*
  **by** *transfer* (*simp add*: *nat-of-num-numeral*)

**lemma** [*code abstract*]:
  *integer-of-nat 0* = *0*
  **by** *transfer simp*

**lemma** [*code abstract*]:
  *integer-of-nat 1* = *1*
  **by** *transfer simp*

**lemma** [*code*]:
  *Suc n* = *n* + *1*

**by** *simp*

**lemma** [*code abstract*]:
*integer-of-nat* ($m + n$) = *of-nat* $m$ + *of-nat* $n$
**by** *transfer simp*

**lemma** [*code abstract*]:
*integer-of-nat* ($m - n$) = *max 0* (*of-nat* $m$ − *of-nat* $n$)
**by** *transfer simp*

**lemma** [*code abstract*]:
*integer-of-nat* ($m * n$) = *of-nat* $m$ * *of-nat* $n$
**by** *transfer* (*simp add*: *of-nat-mult*)

**lemma** [*code abstract*]:
*integer-of-nat* ($m$ *div* $n$) = *of-nat* $m$ *div of-nat* $n$
**by** *transfer* (*simp add*: *zdiv-int*)

**lemma** [*code abstract*]:
*integer-of-nat* ($m$ *mod* $n$) = *of-nat* $m$ *mod of-nat* $n$
**by** *transfer* (*simp add*: *zmod-int*)

**lemma** [*code*]:
*Divides.divmod-nat* $m$ $n$ = ($m$ *div* $n$, $m$ *mod* $n$)
**by** (*fact divmod-nat-div-mod*)

**lemma** [*code*]:
*HOL.equal* $m$ $n$ = *HOL.equal* (*of-nat* $m$ :: *integer*) (*of-nat* $n$)
**by** *transfer* (*simp add*: *equal*)

**lemma** [*code*]:
$m \le n \longleftrightarrow$ (*of-nat* $m$ :: *integer*) $\le$ *of-nat* $n$
**by** *simp*

**lemma** [*code*]:
$m < n \longleftrightarrow$ (*of-nat* $m$ :: *integer*) $<$ *of-nat* $n$
**by** *simp*

**lemma** *num-of-nat-code* [*code*]:
*num-of-nat* = *num-of-integer* ∘ *of-nat*
**by** *transfer* (*simp add*: *fun-eq-iff*)

**end**

**lemma** (**in** *semiring-1*) *of-nat-code-if*:
*of-nat* $n$ = (*if* $n = 0$ *then 0*
*else let*
($m$, $q$) = *divmod-nat* $n$ *2*;
$m'$ = *2* * *of-nat* $m$

*in if q = 0 then m′ else m′ + 1*)
**proof** −
**from** *mod-div-equality* **have** ∗: *of-nat n = of-nat* (*n div 2* ∗ *2* + *n mod 2*) **by** *simp*
**show** *?thesis*
**by** (*simp add*: *Let-def divmod-nat-div-mod of-nat-add* [*symmetric*])
(*simp add*: ∗ *mult.commute of-nat-mult add.commute*)
**qed**

**declare** *of-nat-code-if* [*code*]

**definition** *int-of-nat* :: *nat* ⇒ *int* **where**
[*code-abbrev*]: *int-of-nat* = *of-nat*

**lemma** [*code*]:
*int-of-nat n* = *int-of-integer* (*of-nat n*)
**by** (*simp add*: *int-of-nat-def*)

**lemma** [*code abstract*]:
*integer-of-nat* (*nat k*) = *max 0* (*integer-of-int k*)
**including** *integer.lifting* **by** *transfer auto*

**lemma** *term-of-nat-code* [*code*]:
— Use *nat-of-integer* in term reconstruction instead of *Code-Target-Nat.Nat* such that reconstructed terms can be fed back to the code generator
*term-of-class.term-of n* =
*Code-Evaluation.App*
(*Code-Evaluation.Const* (*STR* ″*Code-Numeral.nat-of-integer*″)
(*typerep.Typerep* (*STR* ″*fun*″)
[*typerep.Typerep* (*STR* ″*Code-Numeral.integer*″) [],
*typerep.Typerep* (*STR* ″*Nat.nat*″) []]))
(*term-of-class.term-of* (*integer-of-nat n*))
**by** (*simp add*: *term-of-anything*)

**lemma** *nat-of-integer-code-post* [*code-post*]:
*nat-of-integer 0* = *0*
*nat-of-integer 1* = *1*
*nat-of-integer* (*numeral k*) = *numeral k*
**including** *integer.lifting* **by** (*transfer*, *simp*)+

**code-identifier**
**code-module** *Code-Target-Nat* ⇀
(*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 10 Additional operators on relations, going beyond Relations.thy, and properties of these operators

**theory** *RelationOperators*
**imports**
  *Main*
  *SetUtils*
  *~~/src/HOL/Library/Code-Target-Nat*

**begin**

# 11 evaluating a relation as a function

If an input has a unique image element under a given relation, return that element; otherwise return a fallback value.

**fun** *eval-rel-or* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a \Rightarrow 'b \Rightarrow 'b$
**where** *eval-rel-or R a z* = (*let im* = *R* `` $\{a\}$ *in if card im* = *1 then the-elem im else z*)

right-uniqueness of a relation: the image of a *trivial* set (i.e. an empty or singleton set) under the relation is trivial again. This is the set-theoretical way of characterizing functions, as opposed to $\lambda$ functions.

**definition** *runiq* :: $('a \times 'b)$ *set* $\Rightarrow$ *bool* **where**
*runiq R* = ($\forall$ *X* . *trivial X* $\longrightarrow$ *trivial* (*R* `` *X*))

# 12 restriction

restriction of a relation to a set (usually resulting in a relation with a smaller domain)

**definition** *restrict*

:: $('a \times 'b)$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $('a \times 'b)$ *set* (**infix** || *75*)
**where** *R* || *X* = *X* $\times$ *Range R* $\cap$ *R*

extensional characterisation of the pairs within a restricted relation

**lemma** *restrict-ext*: *R* || *X* = $\{(x, y) \mid x\ y\ .\ x \in X \wedge (x, y) \in R\}$
**unfolding** *restrict-def* **using** *Range-iff* **by** *blast*

alternative statement of *?R* || *?X* = $\{(x, y) \mid x\ y.\ x \in ?X \wedge (x, y) \in ?R\}$ without explicitly naming the pair's components

**lemma** *restrict-ext'*: *R* || *X* = $\{p\ .\ fst\ p \in X \wedge p \in R\}$
**proof** −
  **have** *R* || *X* = $\{(x, y) \mid x\ y\ .\ x \in X \wedge (x, y) \in R\}$ **by** (*rule restrict-ext*)

**also have** $\ldots = \{\ p\ .\ \mathit{fst}\ p \in X \wedge p \in R\ \}$ **by** *force*
**finally show** *?thesis* **.**
**qed**

Restricting a relation to the empty set yields the empty set.

**lemma** *restrict-empty*: $P \mid\mid \{\} = \{\}$ **unfolding** *restrict-def* **by** *simp*

A restriction is a subrelation of the original relation.

**lemma** *restriction-is-subrel*: $P \mid\mid X \subseteq P$ **using** *restrict-def* **by** *blast*

Restricting a relation only has an effect within its domain.

**lemma** *restriction-within-domain*: $P \mid\mid X = P \mid\mid (X \cap (\mathit{Domain}\ P))$ **unfolding** *restrict-def* **by** *fast*

alternative characterisation of the restriction of a relation to a singleton set

**lemma** *restrict-to-singleton*: $P \mid\mid \{x\} = \{x\} \times P \text{ `` } \{x\}$ **unfolding** *restrict-def* **by** *fast*

# 13 relation outside some set

For a set-theoretical relation $R$ and an "exclusion" set $X$, return those tuples of $R$ whose first component is not in $X$. In other words, exclude $X$ from the domain of $R$.

**definition** *Outside* :: $('a \times 'b)\ \mathit{set} \Rightarrow 'a\ \mathit{set} \Rightarrow ('a \times 'b)\ \mathit{set}$ (**infix** *outside* *75*)
**where** $R\ \mathit{outside}\ X = R - (X \times \mathit{Range}\ R)$

Considering a relation outside some set $X$ reduces its domain by $X$.

**lemma** *outside-reduces-domain*: $\mathit{Domain}\ (P\ \mathit{outside}\ X) = \mathit{Domain}\ P - X$
**unfolding** *Outside-def* **by** *fast*

Considering a relation outside a singleton set $\{x\}$ reduces its domain by $x$.

**corollary** *Domain-outside-singleton*:
  **assumes** $\mathit{Domain}\ R = \mathit{insert}\ x\ A$
    **and** $x \notin A$
  **shows** $\mathit{Domain}\ (R\ \mathit{outside}\ \{x\}) = A$
**using** *assms*
**using** *outside-reduces-domain*
**by** (*metis Diff-insert-absorb*)

For any set, a relation equals the union of its restriction to that set and its pairs outside that set.

**lemma** *outside-union-restrict*: $P = P\ \mathit{outside}\ X \cup P \mid\mid X$
**unfolding** *Outside-def restrict-def* **by** *fast*

The range of a relation $R$ outside some exclusion set $X$ is a subset of the image of the domain of $R$, minus $X$, under $R$.

**lemma** *Range-outside-sub-Image-Domain*: *Range* (*R outside X*) $\subseteq$ *R* `` (*Domain R* $-$ *X*)
**using** *Outside-def Image-def Domain-def Range-def* **by** *blast*

Considering a relation outside some set doesn't enlarge its range.

**lemma** *Range-outside-sub*:
  **assumes** *Range R* $\subseteq$ *Y*
  **shows** *Range* (*R outside X*) $\subseteq$ *Y*
**using** *assms*
**using** *outside-union-restrict*
**by** (*metis Range-mono inf-sup-ord*(*3*) *subset-trans*)

# 14 flipping pairs of relations

flipping a pair: exchanging first and second component

**definition** *flip* **where** *flip tup* = (*snd tup*, *fst tup*)

Flipped pairs can be found in the converse relation.

**lemma** *flip-in-conv*:
  **assumes** *tup* $\in$ *R*
  **shows** *flip tup* $\in R^{-1}$
**using** *assms* **unfolding** *flip-def* **by** *simp*

Flipping a pair twice doesn't change it.

**lemma** *flip-flip*: *flip* (*flip tup*) = *tup*
**by** (*metis flip-def fst-conv snd-conv surjective-pairing*)

Flipping all pairs in a relation yields the converse relation.

**lemma** *flip-conv*: *flip* ` *R* = $R^{-1}$
**proof** –
  **have** *flip* ` *R* = { *flip tup* | *tup* . *tup* $\in$ *R* } **by** (*metis image-Collect-mem*)
  **also have** ... = { *tup* . *tup* $\in R^{-1}$ } **using** *flip-in-conv* **by** (*metis converse-converse flip-flip*)
  **also have** ... = $R^{-1}$ **by** *simp*
  **finally show** *?thesis* **.**
**qed**

Summing over all pairs of a relation is the same as summing over all pairs of the converse relation after flipping them.

**lemma** *setsum-rel-comm*:
  **fixes** *R*::(*'a* $\times$ *'b*) *set*
    **and** *f*::*'a* $\Rightarrow$ *'b* $\Rightarrow$ *'c*::*comm-monoid-add*
  **shows** ($\sum$ (*x*, *y*) $\in$ *R* . *f x y*) = ($\sum$ (*y'*, *x'*) $\in R^{-1}$ . *f x' y'*)
**proof** –

  **have** *inj-on flip* ($R^{-1}$)
    **by** (*metis flip-flip inj-on-def*)

**moreover have** $R = \mathit{flip}$ ‘ $(R^{-1})$
**by** (*metis converse-converse flip-conv*)
**moreover have** $\bigwedge$ *tup* . *tup* $\in R^{-1} \Longrightarrow f$ (*snd tup*) (*fst tup*) $= f$ (*fst* (*flip tup*)) (*snd* (*flip tup*))
**by** (*metis flip-def fst-conv snd-conv*)
**ultimately have** ($\sum$ *tup* $\in R$ . $f$ (*fst tup*) (*snd tup*)) $=$ ($\sum$ *tup* $\in R^{-1}$ . $f$ (*snd tup*) (*fst tup*))
**using** *setsum.reindex-cong* **by** (*metis* (*erased*, *lifting*))
**then show** *?thesis*
**by** (*metis* (*mono-tags*) *setsum.cong split-beta*)
**qed**

## 15 evaluation as a function

Evaluates a relation $R$ for a single argument, as if it were a function. This will only work if $R$ is a total function, i.e. if the image is always a singleton set.

**fun** *eval-rel* :: $('a \times 'b)$ *set* $\Rightarrow 'a \Rightarrow 'b$ (**infix** ,, *75*)
**where** $R$ ,, $a$ $=$ *the-elem* ($R$ “ $\{a\}$)

## 16 paste

the union of two binary relations $P$ and $Q$, where pairs from $Q$ override pairs from $P$ when their first components coincide. This is particularly useful when P, Q are *runiq*, and one wants to preserve that property.

**definition** *paste* (**infix** $+*$ *75*)
**where** $P +* Q = (P$ *outside Domain* $Q) \cup Q$

If a relation $P$ is a subrelation of another relation $Q$ on $Q$'s domain, pasting $Q$ on $P$ is the same as forming their union.

**lemma** *paste-subrel*: **assumes** $P \mid\mid$ *Domain* $Q \subseteq Q$ **shows** $P +* Q = P \cup Q$
**unfolding** *paste-def* **using** *assms outside-union-restrict* **by** *blast*

Pasting two relations with disjoint domains is the same as forming their union.

**lemma** *paste-disj-domains*: **assumes** *Domain* $P \cap$ *Domain* $Q = \{\}$ **shows** $P +* Q = P \cup Q$
**unfolding** *paste-def Outside-def*
**using** *assms*
**by** *fast*

A relation $P$ is equivalent to pasting its restriction to some set $X$ on $P$ *outside* $X$.

**lemma** *paste-outside-restrict*: $P = (P$ *outside* $X) +* (P \mid\mid X)$
**proof** −

**have** *Domain* (*P outside X*) ∩ *Domain* (*P* || *X*) = {}
**unfolding** *Outside-def restrict-def* **by** *fast*
**moreover have** *P* = *P outside X* ∪ *P* || *X* **by** (*rule outside-union-restrict*)
**ultimately show** *?thesis* **using** *paste-disj-domains* **by** *metis*
**qed**

The domain of two pasted relations equals the union of their domains.

**lemma** *paste-Domain*: *Domain*(*P* +∗ *Q*)=*Domain P*∪*Domain Q* **unfolding** *paste-def Outside-def* **by** *blast*

Pasting two relations yields a subrelation of their union.

**lemma** *paste-sub-Un*: *P* +∗ *Q* ⊆ *P* ∪ *Q* **unfolding** *paste-def Outside-def* **by** *fast*

The range of two pasted relations is a subset of the union of their ranges.

**lemma** *paste-Range*: *Range* (*P* +∗ *Q*) ⊆ *Range P* ∪ *Range Q*

**using** *paste-sub-Un* **by** *blast*

**end**

# 17 Additional properties of relations, and operators on relations, as they have been defined by Relations.thy

**theory** *RelationProperties*
**imports**
*Main*
*RelationOperators*
*SetUtils*
*Conditionally-Complete-Lattices*

**begin**

# 18 right-uniqueness

**lemma** *injflip*: *inj-on flip A* **by** (*metis flip-flip inj-on-def*)

**lemma** *lm003*: *card P* = *card* (*P*ˆ−*1*) **using** *assms card-image flip-conv injflip* **by** *metis*

**lemma** *nn56*: *card X*=*1* = (*X*={*the-elem X*})
**by** (*metis One-nat-def card-Suc-eq card-empty empty-iff the-elem-eq*)

**lemma** *lm007b*: *trivial X* = (*X*={} ∨ *card X*=*1*) **using**
*nn56 order-refl subset-singletonD trivial-def trivial-empty* **by** (*metis*(*no-types*))

**lemma** *lm004*: *trivial* $P$ = *trivial* $(P\hat{\ }{-1})$ **using** *trivial-def subset-singletonD subset-refl subset-insertI nn56 converse-inject converse-empty lm003* **by** *metis*

**lemma** *lll85*: *Range* $(P||X) = P``X$ **unfolding** *restrict-def* **by** *blast*
**lemma** *lll02*: $(P \mid\mid X) \mid\mid Y = P \mid\mid (X \cap Y)$

**unfolding** *restrict-def* **by** *fast*
**lemma** *ll41*: *Domain* $(R||X)$ = *Domain* $R \cap X$ **using** *restrict-def* **by** *fastforce*

A subrelation of a right-unique relation is right-unique.

**lemma** *subrel-runiq*: **assumes** *runiq* $Q$ $P \subseteq Q$ **shows** *runiq* $P$
**using** *assms runiq-def* **by** (*metis Image-mono subsetI trivial-subset*)

**lemma** *lll31*: **assumes** *runiq* $P$ **shows** *inj-on fst* $P$
**unfolding** *inj-on-def* **using** *assms runiq-def trivial-def trivial-imp-no-distinct the-elem-eq surjective-pairing subsetI Image-singleton-iff* **by** (*metis*(*no-types*))

alternative characterisation of right-uniqueness: the image of a singleton set is *trivial*, i.e. an empty or singleton set.

**lemma** *runiq-alt*: *runiq* $R \longleftrightarrow (\forall\ x\ .$ *trivial* $(R\ ``\ \{x\}))$
**unfolding** *runiq-def* **using** *Image-empty lm007 the-elem-eq* **by** (*metis*(*no-types*))

an alternative definition of right-uniqueness in terms of *op* ,,

**lemma** *runiq-wrt-eval-rel*: *runiq* $R = (\forall x\ .\ R\ ``\ \{x\} \subseteq \{R\ ,,\ x\})$ **by** (*metis eval-rel.simps runiq-alt trivial-def*)
**lemma** *l31*: **assumes** *runiq* $f$ **assumes** $(x,y)\in f$ **shows** $y=f,,x$ **using** *assms runiq-wrt-eval-rel subset-singletonD Image-singleton-iff equals0D singletonE* **by** *fast*
**lemma** *runiq-basic*: *runiq* $R \longleftrightarrow (\forall\ x\ y\ y'\ .\ (x,\ y) \in R \wedge (x,\ y') \in R \longrightarrow y = y')$
**unfolding** *runiq-alt lm01* **by** *blast*

**lemma** *ll71*: **assumes** *runiq* $f$ **shows** $f``(f\hat{\ }{-1}``Y) \subseteq Y$
**using** *assms runiq-basic ImageE converse-iff subsetI* **by** (*metis*(*no-types*))

**lemma** *ll68*: **assumes** *runiq* $f$ $y1 \in$ *Range* $f$ **shows**
$(f\hat{\ }{-1}\ ``\ \{y1\} \cap f\hat{\ }{-1}\ ``\ \{y2\} \neq \{\}) = (f\hat{\ }{-1}``\{y1\}=f\hat{\ }{-1}``\{y2\})$
**using** *assms ll71* **by** *fast*

**lemma** *converse-Image*:
  **assumes** *runiq*: *runiq* $R$
    **and** *runiq-conv*: *runiq* $(R\hat{\ }{-1})$
**shows** $(R\hat{\ }{-1})\ ``\ R\ ``\ X \subseteq X$ **using** *assms* **by** (*metis converse-converse ll71*)

**lemma** *lll32*: **assumes** *inj-on fst* $P$ **shows** *runiq* $P$ **unfolding** *runiq-basic* **using** *assms fst-conv inj-on-def old.prod.inject* **by** (*metis*(*no-types*))

**lemma** *lll33*: *runiq* $P$=*inj-on fst* $P$ **using** *lll31 lll32* **by** *blast*

**lemma** *disj-Un-runiq*: **assumes** *runiq P runiq Q Domain P* $\cap$ (*Domain Q*) = {}
**shows** *runiq* (*P Un Q*)
**using** *assms lll33 fst-eq-Domain lm010b* **by** *metis*

**lemma** *runiq-paste1*: **assumes** *runiq Q runiq* (*P outside Domain Q*) **shows** *runiq* (*P* +* *Q*)
**unfolding** *paste-def* **using** *assms disj-Un-runiq Diff-disjoint Un-commute outside-reduces-domain*
**by** (*metis* (*poly-guards-query*))

**corollary** *runiq-paste2*: **assumes** *runiq Q runiq P* **shows** *runiq* (*P* +* *Q*)
**using** *assms runiq-paste1 subrel-runiq Diff-subset Outside-def* **by** (*metis*)

**lemma** *l14*: *runiq* {(*x*,*f x*)| *x*. *P x*} **unfolding** *runiq-basic* **by** *fast*

**lemma** *runiq-alt2*: *runiq R* = ($\forall$ *x* $\in$ *Domain R*. *trivial* (*R* `` {*x*}))
**by** (*metis Domain.DomainI Image-singleton-iff lm01 runiq-alt*)

**lemma** *lm013*: **assumes** *x* $\in$ *Domain R runiq R* **shows** *card* (*R*``{*x*})=*1*
**using** *assms runiq-alt2 lm007b* **by** (*metis DomainE Image-singleton-iff empty-iff*)

The image of a singleton set under a right-unique relation is a singleton set.

**lemma** *Image-runiq-eq-eval*: **assumes** *x* $\in$ *Domain R runiq R* **shows** *R* `` {*x*} = {*R* ,, *x*}
**using** *assms lm013* **by** (*metis eval-rel.simps nn56*)

the image of a singleton set under a right-unique relation is *trivial*, i.e. an empty or singleton set.

If all images of singleton sets under a relation are *trivial*, i.e. an empty or singleton set, the relation is right-unique.

**lemma** *Image-within-runiq-domain*:
  **fixes** *x R*
  **assumes** *runiq R*
  **shows** *x* $\in$ *Domain R* $\longleftrightarrow$ ($\exists$ *y* . *R* `` {*x*} = {*y*}) **using** *assms Image-runiq-eq-eval*
**by** *fast*

**lemma** *runiq-imp-singleton-image'*:
  **assumes** *runiq*: *runiq R*
    **and** *dom*: *x* $\in$ *Domain R*
  **shows** *the-elem* (*R* `` {*x*}) = (*THE y* . (*x*, *y*) $\in$ *R*) (**is** *the-elem* (*R* `` {*x*}) = *?y*)
**unfolding** *the-elem-def*
**using** *assms Image-singleton-iff Image-within-runiq-domain singletonD singletonI*
**by** (*metis*)

**lemma** *runiq-conv-imp-singleton-preimage'*:
  **assumes** *runiq-conv*: *runiq* ($R^{-1}$)

**and** *ran*: $y \in Range\ R$
**shows** *the-elem* $((R^{-1})$ ‘‘ $\{y\}) = (THE\ x\ .\ (x,\ y) \in R)$

**proof** −
**from** *ran* **have** *dom*: $y \in Domain\ (R^{-1})$ **by** *simp*
**with** *runiq-conv* **have** *the-elem* $((R^{-1})$ ‘‘ $\{y\}) = (THE\ x\ .\ (y,\ x) \in (R^{-1}))$ **by** (*rule runiq-imp-singleton-image′*)
**also have** ... $= (THE\ x\ .\ (x,\ y) \in R)$ **by** *simp*
**finally show** *?thesis* **.**
**qed**

another alternative definition of right-uniqueness in terms of *op* ,,

**lemma** *runiq-wrt-eval-rel′*:
**fixes** $R :: ('a \times 'b)\ set$
**shows** *runiq* $R \longleftrightarrow (\forall x \in Domain\ R\ .\ R$ ‘‘ $\{x\} = \{R\ ,,\ x\})$ **unfolding** *runiq-wrt-eval-rel* **by** *fast*

**lemma** *runiq-wrt-ex1*:
*runiq* $R \longleftrightarrow (\forall\ a \in Domain\ R\ .\ \exists!\ b\ .\ (a,\ b) \in R)$
**using** *runiq-basic* **by** (*metis Domain.DomainI Domain.cases*)

**lemma** *runiq-imp-THE-right-comp*:
**fixes** $a$ **and** $b$
**assumes** *runiq*: *runiq* $R$
**and** *aRb*: $(a,\ b) \in R$
**shows** $b = (THE\ b\ .\ (a,\ b) \in R)$ **using** *assms* **by** (*metis runiq-basic the-equality*)

**lemma** *runiq-imp-THE-right-comp′*:
**assumes** *runiq*: *runiq* $R$
**and** *in-Domain*: $a \in Domain\ R$
**shows** $(a,\ THE\ b.\ (a,\ b) \in R) \in R$
**proof** −
**from** *in-Domain* **obtain** $b$ **where** $*$: $(a,\ b) \in R$ **by** *force*
**with** *runiq* **have** $b = (THE\ b\ .\ (a,\ b) \in R)$ **by** (*rule runiq-imp-THE-right-comp*)
**with** $*$ **show** *?thesis* **by** *simp*
**qed**

**lemma** *THE-right-comp-imp-runiq*:
**assumes** $\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow b = (THE\ b\ .\ (a,\ b) \in R)$
**shows** *runiq* $R$
**using** *assms DomainE runiq-wrt-ex1* **by** *metis*

another alternative definition of right-uniqueness in terms of *The*

**lemma** *runiq-wrt-THE*:
*runiq* $R \longleftrightarrow (\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow b = (THE\ b\ .\ (a,\ b) \in R))$
**proof**
**assume** *runiq* $R$
**then show** $\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow b = (THE\ b\ .\ (a,\ b) \in R)$ **by** (*metis runiq-imp-THE-right-comp*)

**next**
  **assume** $\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow b = (\mathit{THE}\ b\ .\ (a,\ b) \in R)$
  **then show** *runiq R* **by** (*rule THE-right-comp-imp-runiq*)
**qed**

**lemma** *runiq-conv-imp-THE-left-comp*:
  **assumes** *runiq-conv*: *runiq* $(R^{-1})$ **and** *aRb*: $(a,\ b) \in R$
  **shows** $a = (\mathit{THE}\ a\ .\ (a,\ b) \in R)$
**proof** −
  **from** *aRb* **have** $(b,\ a) \in R^{-1}$ **by** *simp*
  **with** *runiq-conv* **have** $a = (\mathit{THE}\ a\ .\ (b,\ a) \in R^{-1})$ **by** (*rule runiq-imp-THE-right-comp*)
  **then show** *?thesis* **by** *fastforce*
**qed**

**lemma** *runiq-conv-imp-THE-left-comp'*:
  **assumes** *runiq-conv*: *runiq* $(R^{-1})$
    **and** *in-Range*: $b \in \mathit{Range}\ R$
  **shows** $(\mathit{THE}\ a.\ (a,\ b) \in R,\ b) \in R$
**proof** −
  **from** *in-Range* **obtain** $a$ **where** $*$: $(a,\ b) \in R$ **by** *force*
  **with** *runiq-conv* **have** $a = (\mathit{THE}\ a\ .\ (a,\ b) \in R)$ **by** (*rule runiq-conv-imp-THE-left-comp*)
  **with** $*$ **show** *?thesis* **by** *simp*
**qed**

**lemma** *THE-left-comp-imp-runiq-conv*:
  **assumes** $\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow a = (\mathit{THE}\ a\ .\ (a,\ b) \in R)$
  **shows** *runiq* $(R^{-1})$
**proof** −
  **from** *assms* **have** $\forall\ b\ a\ .\ (b,\ a) \in R^{-1} \longrightarrow a = (\mathit{THE}\ a\ .\ (b,\ a) \in R^{-1})$ **by** *auto*
  **then show** *?thesis* **by** (*rule THE-right-comp-imp-runiq*)
**qed**

**lemma** *runiq-conv-wrt-THE*:
  *runiq* $(R^{-1}) \longleftrightarrow (\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow a = (\mathit{THE}\ a\ .\ (a,\ b) \in R))$
**proof** −
  **have** *runiq* $(R^{-1}) \longleftrightarrow (\forall\ a\ b\ .\ (a,\ b) \in R^{-1} \longrightarrow b = (\mathit{THE}\ b\ .\ (a,\ b) \in R^{-1}))$ **by** (*rule runiq-wrt-THE*)
  **also have** $\ldots \longleftrightarrow (\forall\ a\ b\ .\ (a,\ b) \in R \longrightarrow a = (\mathit{THE}\ a\ .\ (a,\ b) \in R))$ **by** *auto*
  **finally show** *?thesis* **.**
**qed**

**lemma** *lm022*: **assumes** *trivial f* **shows** *runiq f* **using** *assms* **by** (*metis* (*erased*, *hide-lams*) *lm01 runiq-basic snd-conv*)

A singleton relation is right-unique.

**corollary** *runiq-singleton-rel*: *runiq* $\{(x,\ y)\}$ (**is** *runiq ?R*)
**using** *trivial-singleton lm022* **by** *fast*

The empty relation is right-unique

**lemma** *runiq-emptyrel*: *runiq* {} **using** *trivial-empty lm022* **by** *blast*

alternative characterisation of the fact that, if a relation $R$ is right-unique, its evaluation $R$ ,, $x$ on some argument $x$ in its domain, occurs in $R$'s range.

**lemma** *eval-runiq-rel*:
**assumes** *domain*: $x \in$ *Domain* $R$
**and** *runiq*: *runiq* $R$
**shows** $(x, R,,x) \in R$
**using** *assms* **by** (*metis l31 runiq-wrt-ex1*)

Evaluating a right-unique relation as a function on the relation's domain yields an element from its range.

**lemma** *eval-runiq-in-Range*:
**assumes** *runiq* $R$
**and** $a \in$ *Domain* $R$
**shows** $R$ ,, $a \in$ *Range* $R$
**using** *assms* **by** (*metis Range-iff eval-runiq-rel*)

right-uniqueness of a restricted relation expressed using basic set theory

**lemma** *runiq-restrict*: *runiq* $(R \mid\mid X) \longleftrightarrow (\forall\ x \in X\ .\ \forall\ y\ y'\ .\ (x, y) \in R \wedge (x, y') \in R \longrightarrow y = y')$
**proof** −
**have** *runiq* $(R \mid\mid X) \longleftrightarrow (\forall\ x\ y\ y'\ .\ (x, y) \in R \mid\mid X \wedge (x, y') \in R \mid\mid X \longrightarrow y = y')$
**by** (*rule runiq-basic*)
**also have** $\ldots \longleftrightarrow (\forall\ x\ y\ y'\ .\ (x, y) \in \{\ p\ .\ fst\ p \in X \wedge p \in R\ \} \wedge (x, y') \in \{\ p\ .\ fst\ p \in X \wedge p \in R\ \} \longrightarrow y = y')$
**using** *restrict-ext'* **by** *blast*
**also have** $\ldots \longleftrightarrow (\forall\ x \in X\ .\ \forall\ y\ y'\ .\ (x, y) \in R \wedge (x, y') \in R \longrightarrow y = y')$ **by** *auto*
**finally show** *?thesis* **.**
**qed**

## 18.1 paste

Pasting a singleton relation on some other right-unique relation $R$ yields a right-unique relation if the single element of the singleton's domain is not yet in the domain of $R$.

**lemma** *runiq-paste3*:
**assumes** *runiq* $R$
**and** $x \notin$ *Domain* $R$
**shows** *runiq* $(R +* \{(x, y)\})$
**using** *assms runiq-paste2 runiq-singleton-rel* **by** *metis*

## 18.2 difference

Removing one pair from a right-unique relation still leaves it right-unique.

**lemma** *runiq-except*:
  **assumes** *runiq R*
  **shows** *runiq* $(R - \{tup\})$
**using** *assms*
**by** (*rule subrel-runiq*) *fast*

**lemma** *runiq-Diff-singleton-Domain*:
  **assumes** *runiq*: *runiq R*
    **and** *in-rel*: $(x, y) \in R$
  **shows** $x \notin$ *Domain* $(R - \{(x, y)\})$

**using** *assms DomainE Domain-Un-eq UnI1 Un-Diff-Int member-remove remove-def runiq-wrt-ex1*
**by** *metis*

## 18.3 converse

The inverse image of the image of a singleton set under some relation is the same singleton set, if both the relation and its converse are right-unique and the singleton set is in the relation's domain.

**lemma** *converse-Image-singleton-Domain*:
  **assumes** *runiq*: *runiq R*
    **and** *runiq-conv*: *runiq* $(R^{-1})$
    **and** *domain*: $x \in$ *Domain R*
**shows** $R^{-1}$ `` $R$ `` $\{x\} = \{x\}$
**proof** –
  **have** *sup*: $\{x\} \subseteq R^{-1}$ `` $R$ `` $\{x\}$ **using** *domain* **by** *fast*
  **have** *trivial* $(R$ `` $\{x\})$ **using** *runiq domain* **by** (*metis runiq-def trivial-singleton*)
  **then have** *trivial* $(R^{-1}$ `` $R$ `` $\{x\})$
    **using** *assms runiq-def* **by** *blast*
  **then show** *?thesis*
    **using** *sup* **by** (*metis singleton-sub-trivial-uniq subset-antisym trivial-def*)
**qed**

The inverse image of the image of a singleton set under some relation is the same singleton set or empty, if both the relation and its converse are right-unique.

**corollary** *converse-Image-singleton*:
  **assumes** *runiq R*
    **and** *runiq* $(R^{-1})$
  **shows** $R^{-1}$ `` $R$ `` $\{x\} \subseteq \{x\}$
**using** *assms converse-Image-singleton-Domain* **by** *fast*

The inverse image of the image of a set under some relation is a subset of that set, if both the relation and its converse are right-unique.

**lemma** *disj-Domain-imp-disj-Image*: **assumes** *Domain* $R \cap X \cap Y = \{\}$
  **assumes** *runiq R*
    **and** *runiq* $(R^{-1})$

**shows** $R$ ‘‘ $X \cap R$ ‘‘ $Y = \{\}$
**using** *assms* **unfolding** *runiq-basic* **by** *blast*

**lemma** *runiq-imp-Dom-rel-Range*:
**assumes** $x \in$ *Domain R*
**and** *runiq R*
**shows** (*THE y* . $(x, y) \in R) \in$ *Range R*
**using** *assms*
**by** (*metis Range.intros runiq-imp-THE-right-comp runiq-wrt-ex1*)

**lemma** *runiq-conv-imp-Range-rel-Dom*:
**assumes** *y-Range*: $y \in$ *Range R*
**and** *runiq-conv*: *runiq* $(R^{-1})$
**shows** (*THE x* . $(x, y) \in R) \in$ *Domain R*
**proof** −
**from** *y-Range* **have** $y \in$ *Domain* $(R^{-1})$ **by** *simp*
**then have** (*THE x* . $(y, x) \in R^{-1}) \in$ *Range* $(R^{-1})$ **using** *runiq-conv* **by** (*rule runiq-imp-Dom-rel-Range*)
**then show** *?thesis* **by** *simp*
**qed**

The converse relation of two pasted relations is right-unique, if the relations have disjoint domains and ranges, and if their converses are both right-unique.

**lemma** *runiq-converse-paste*:
**assumes** *runiq-P-conv*: *runiq* $(P^{-1})$
**and** *runiq-Q-conv*: *runiq* $(Q^{-1})$
**and** *disj-D*: *Domain P* $\cap$ *Domain Q* $= \{\}$
**and** *disj-R*: *Range P* $\cap$ *Range Q* $= \{\}$
**shows** *runiq* $((P +* Q)^{-1})$
**proof** −
**have** $P +* Q = P \cup Q$ **using** *disj-D* **by** (*rule paste-disj-domains*)
**then have** $(P +* Q)^{-1} = P^{-1} \cup Q^{-1}$ **by** *auto*
**also have** $\ldots = P^{-1} +* Q^{-1}$ **using** *disj-R paste-disj-domains Domain-converse* **by** *metis*
**finally show** *?thesis* **using** *runiq-P-conv runiq-Q-conv runiq-paste2* **by** *auto*
**qed**

The converse relation of a singleton relation pasted on some other relation $R$ is right-unique, if the singleton pair is not in *Domain R* $\times$ *Range R*, and if $R^{-1}$ is right-unique.

**lemma** *runiq-converse-paste-singleton*:
**assumes** *runiq*: *runiq* $(R^{-1})$
**and** *y-notin-R*: $y \notin$ *Range R*
**and** *x-notin-D*: $x \notin$ *Domain R*
**shows** *runiq* $((R +* \{(x,y)\})^{-1})$
**proof** −
**have** $\{(x,y)\}^{-1} = \{(y,x)\}$ **by** *fastforce*
**then have** *runiq* $(\{(x,y)\}^{-1})$ **using** *runiq-singleton-rel* **by** *metis*

**moreover have** *Domain R* $\cap$ *Domain* $\{(x,y)\} = \{\}$ **and** *Range R* $\cap$ (*Range* $\{(x,y)\})=\{\}$
**using** *y-notin-R x-notin-D* **by** *simp-all*
**ultimately show** *?thesis* **using** *runiq runiq-converse-paste* **by** *blast*
**qed**

If a relation is known to be right-unique, it is easier to know when we can evaluate it like a function, using *eval-rel-or*.

**lemma** *eval-runiq-rel-or*:
**assumes** *runiq R*
**shows** *eval-rel-or R a z* = (*if a* $\in$ *Domain R then the-elem* (*R* `` $\{a\}$) *else z*)
**proof** –
**from** *assms* **have** *card* (*R* `` $\{a\}$) = *1* $\longleftrightarrow$ *a* $\in$ *Domain R*

**using** *Image-within-runiq-domain card-Suc-eq card-empty ex-in-conv One-nat-def* **by** *metis*
**then show** *?thesis* **by** *force*
**qed**

## 19 injectivity

A relation $R$ is injective on its domain iff any two domain elements having the same image are equal. This definition on its own is of limited utility, as it does not assume that $R$ is a function, i.e. right-unique.

**definition** *injective* :: ($'a \times 'b$) *set* $\Rightarrow$ *bool*
**where** *injective R* $\longleftrightarrow$ ($\forall$ *a* $\in$ *Domain R* . $\forall$ *b* $\in$ *Domain R* . *R* `` $\{a\}$ = *R* `` $\{b\}$ $\longrightarrow$ *a* = *b*)

If both a relation and its converse are right-unique, it is injective on its domain.

**lemma** *runiq-and-conv-imp-injective*:
**assumes** *runiq*: *runiq R*
**and** *runiq-conv*: *runiq* ($R^{-1}$)
**shows** *injective R*
**proof** –
{
**fix** *a* **assume** *a-Dom*: *a* $\in$ *Domain R*
**fix** *b* **assume** *b-Dom*: *b* $\in$ *Domain R*
**have** *R* `` $\{a\}$ = *R* `` $\{b\}$ $\longrightarrow$ *a* = *b*
**proof**
**assume** *eq-Im*: *R* `` $\{a\}$ = *R* `` $\{b\}$
**from** *runiq a-Dom* **obtain** *Ra* **where** *Ra*: *R* `` $\{a\}$ = $\{Ra\}$ **by** (*metis Image-runiq-eq-eval*)
**from** *runiq b-Dom* **obtain** *Rb* **where** *Rb*: *R* `` $\{b\}$ = $\{Rb\}$ **by** (*metis Image-runiq-eq-eval*)
**from** *eq-Im Ra Rb* **have** *eq-Im'*: *Ra* = *Rb* **by** *simp*
**from** *eq-Im' Ra a-Dom runiq-conv* **have** *a'*: ($R^{-1}$) `` $\{Ra\}$ = $\{a\}$

using *converse-Image-singleton-Domain runiq* **by** *metis*
**from** *eq-Im′ Rb b-Dom runiq-conv* **have** $b'$: $(R^{-1})$ `` $\{Rb\} = \{b\}$
using *converse-Image-singleton-Domain runiq* **by** *metis*
**from** *eq-Im′ a′ b′* **show** $a = b$ **by** *simp*
**qed**
**}**
**then show** *?thesis* **unfolding** *injective-def* **by** *blast*
**qed**

the set of all injective functions from $X$ to $Y$.

**definition** *injections* :: $'a$ *set* $\Rightarrow$ $'b$ *set* $\Rightarrow$ $('a \times 'b)$ *set set*
**where** *injections X Y* = $\{R$ . *Domain R* $= X \wedge$ *Range R* $\subseteq Y \wedge$ *runiq R* $\wedge$ *runiq* $(R^{-1})\}$

introduction rule that establishes the injectivity of a relation

**lemma** *injectionsI*:
**fixes** $R$::$('a \times 'b)$ *set*
**assumes** *Domain R* $= X$
**and** *Range R* $\subseteq Y$
**and** *runiq R*
**and** *runiq* $(R^{-1})$
**shows** $R \in$ *injections X Y*
**using** *assms* **unfolding** *injections-def* **using** *CollectI* **by** *blast*

the set of all injective partial functions (including total ones) from $X$ to $Y$.

**definition** *partial-injections* :: $'a$ *set* $\Rightarrow$ $'b$ *set* $\Rightarrow$ $('a \times 'b)$ *set set*
**where** *partial-injections X Y* = $\{R$ . *Domain R* $\subseteq X \wedge$ *Range R* $\subseteq Y \wedge$ *runiq R* $\wedge$ *runiq* $(R^{-1})\}$

Given a relation $R$, an element $x$ of the relation's domain type and a set $Y$ of the relation's range type, this function constructs the list of all superrelations of $R$ that extend $R$ by a pair $(x, y)$ for some $y$ not yet covered by $R$.

**fun** *sup-rels-from-alg* :: $('a \times 'b$::*linorder*$)$ *set* $\Rightarrow$ $'a \Rightarrow$ $'b$ *set* $\Rightarrow$ $('a \times 'b)$ *set list*
**where**
*sup-rels-from-alg R x Y* = [ $R$ +∗ $\{(x,y)\}$ . $y \leftarrow$ *sorted-list-of-set* $(Y -$ *Range R*$)$ ]

set-based variant of *sup-rels-from-alg*

**definition** *sup-rels-from* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a \Rightarrow$ $'b$ *set* $\Rightarrow$ $('a \times 'b)$ *set set*
**where** *sup-rels-from R x Y* = $\{$ $R$ +∗ $\{(x, y)\}$ | $y$ . $y \in Y -$ *Range R* $\}$

On finite sets, *sup-rels-from-alg* and *sup-rels-from* are equivalent.

**lemma** *sup-rels-from-paper-equiv-alg*:
**assumes** *finite Y*
**shows** *set* (*sup-rels-from-alg R x Y*) = *sup-rels-from R x Y*
**proof** −
**have** *distinct* (*sorted-list-of-set* $(Y -$ *Range R*$)$) **using** *assms* **by** *simp*

**then have** *set* [ $R$ +∗ $\{(x,y)\}$ . $y \leftarrow$ *sorted-list-of-set* ($Y$ − *Range* $R$) ] = { $R$ +∗ $\{(x,y)\}$ | $y$ . $y \in$ *set* (*sorted-list-of-set* ($Y$ − *Range* $R$)) } **by** *auto*
**moreover have** *set* (*sorted-list-of-set* ($Y$ − *Range* $R$)) = $Y$ − *Range* $R$ **using** *assms* **by** *simp*
**ultimately show** *?thesis* **unfolding** *sup-rels-from-def* **by** *simp*
**qed**

the list of all injective functions (represented as relations) from one set (represented as a list) to another set

**fun** *injections-alg* :: $'a$ *list* ⇒ $'b$::*linorder set* ⇒ ($'a \times 'b$) *set list*
**where** *injections-alg* [] $Y$ = [{}] |
*injections-alg* ($x$ # *xs*) $Y$ = *concat* [ [ $R$ +∗ $\{(x,y)\}$ . $y \leftarrow$ *sorted-list-of-set* ($Y$ − *Range* $R$) ]
. $R \leftarrow$ *injections-alg xs* $Y$ ]

the set-theoretic variant of the recursive rule of *injections-alg*

**lemma** *injections-paste*:
**assumes** *new*: $x \notin A$
**shows** *injections* (*insert* $x$ $A$) $Y$ = ($\bigcup$ { *sup-rels-from* $P$ $x$ $Y$ | $P$ . $P \in$ *injections* $A$ $Y$ })
**proof** (*rule equalitySubsetI*)
**fix** $R$
**assume** $R \in$ *injections* (*insert* $x$ $A$) $Y$
**then have** *injections-unfolded*: *Domain* $R$ = *insert* $x$ $A$ ∧ *Range* $R \subseteq Y$ ∧ *runiq* $R$ ∧ *runiq* ($R^{-1}$)
**unfolding** *injections-def* **by** *simp*
**then have** *Domain*: *Domain* $R$ = *insert* $x$ $A$
**and** *Range*: *Range* $R \subseteq Y$
**and** *runiq*: *runiq* $R$
**and** *runiq-conv*: *runiq* ($R^{-1}$) **by** *simp-all*

**let** *?P* = $R$ *outside* $\{x\}$
**have** *subrel*: *?P* $\subseteq R$ **unfolding** *Outside-def* **by** *fast*
**have** *subrel-conv*: *?P*$^{-1} \subseteq R^{-1}$ **using** *subrel* **by** *blast*

**from** *Domain new* **have** *Domain-pre*: *Domain ?P* = $A$ **by** (*rule Domain-outside-singleton*)
**have** *P-inj*: *?P* $\in$ *injections* $A$ $Y$
**proof** (*rule injectionsI*)
**show** *Domain ?P* = $A$ **by** (*rule Domain-pre*)
**show** *Range ?P* $\subseteq Y$ **using** *Range* **by** (*rule Range-outside-sub*)
**show** *runiq ?P* **using** *runiq subrel* **by** (*rule subrel-runiq*)
**show** *runiq* (*?P*$^{-1}$) **using** *runiq-conv subrel-conv* **by** (*rule subrel-runiq*)
**qed**

**obtain** $y$ **where** $y$: $R$ `` $\{x\} = \{y\}$ **using** *Image-runiq-eq-eval Domain runiq* **by** (*metis insertI1*)
**from** $y$ *Range* **have** $y \in Y$ **by** *fast*
**moreover have** $y \notin$ *Range ?P*

**proof**
**assume** *assm*: $y \in Range\ ?P$
**then obtain** $x'$ **where** *x'-Domain*: $x' \in Domain\ ?P$ **and** *x'-P-y*: $(x', y) \in ?P$ **by** *fast*
**have** *x'-img*: $x' \in R^{-1}$ `` $\{y\}$ **using** *subrel x'-P-y* **by** *fast*
**have** *x-img*: $x \in R^{-1}$ `` $\{y\}$ **using** *y* **by** *fast*
**have** $x' \neq x$
**proof** −
**from** *x'-Domain* **have** $x' \in A$ **using** *Domain-pre* **by** *fast*
**with** *new* **show** *?thesis* **by** *fast*
**qed**
**have** *trivial* $(R^{-1}$ `` $\{y\})$ **using** *runiq-conv* **by** (*metis runiq-alt*)
**then have** $x' = x$ **using** *x'-img x-img* **by** (*rule trivial-imp-no-distinct*)
**with** ‹$x' \neq x$› **show** *False* **..**
**qed**
**ultimately have** *y-in*: $y \in Y - Range\ ?P$ **by** (*rule DiffI*)

**from** *y* **have** *x-rel*: $R \mid\mid \{x\} = \{(x, y)\}$ **unfolding** *restrict-def* **by** *blast*
**from** *x-rel* **have** *Dom-restrict*: $Domain\ (R \mid\mid \{x\}) = \{x\}$ **by** *simp*
**from** *x-rel* **have** *P-paste'*: $?P +* \{(x, y)\} = ?P \cup R \mid\mid \{x\}$
**using** *outside-union-restrict paste-outside-restrict* **by** *metis*
**from** *Dom-restrict Domain-pre new* **have** $Domain\ ?P \cap Domain\ (R \mid\mid \{x\}) = \{\}$ **by** *simp*
**then have** $?P +* (R \mid\mid \{x\}) = ?P \cup (R \mid\mid \{x\})$ **by** (*rule paste-disj-domains*)
**then have** *P-paste*: $?P +* \{(x, y)\} = R$ **using** *P-paste' outside-union-restrict* **by** *blast*

**from** *P-inj y-in P-paste* **have** $\exists\ P \in injections\ A\ Y\ .\ \exists\ y \in Y - Range\ P\ .\ R = P +* \{(x, y)\}$ **by** *blast*

**then have** $\exists\ Q \in \{\ sup\text{-}rels\text{-}from\ P\ x\ Y \mid P\ .\ P \in injections\ A\ Y\ \}\ .\ R \in Q$
**unfolding** *sup-rels-from-def* **by** *auto*
**then show** $R \in \bigcup\ \{\ sup\text{-}rels\text{-}from\ P\ x\ Y \mid P\ .\ P \in injections\ A\ Y\ \}$
**using** *Union-member* **by** (*rule rev-iffD1*)
**next**
**fix** $R$
**assume** $R \in \bigcup\ \{\ sup\text{-}rels\text{-}from\ P\ x\ Y \mid P\ .\ P \in injections\ A\ Y\ \}$
**then have** $\exists\ Q \in \{\ sup\text{-}rels\text{-}from\ P\ x\ Y \mid P\ .\ P \in injections\ A\ Y\ \}\ .\ R \in Q$
**using** *Union-member* **by** (*rule rev-iffD2*)
**then obtain** $P$ **and** $y$ **where** $P$: $P \in injections\ A\ Y$
**and** $y$: $y \in Y - Range\ P$
**and** $R$: $R = P +* \{(x, y)\}$
**unfolding** *sup-rels-from-def* **by** *auto*
**then have** *P-unfolded*: $Domain\ P = A \wedge Range\ P \subseteq Y \wedge runiq\ P \wedge runiq\ (P^{-1})$
**unfolding** *injections-def* **by** (*simp add*: *CollectE*)
**then have** *Domain-pre*: $Domain\ P = A$
**and** *Range-pre*: $Range\ P \subseteq Y$
**and** *runiq-pre*: $runiq\ P$

and *runiq-conv-pre*: *runiq* $(P^{-1})$ **by** *simp-all*

**show** $R \in$ *injections* (*insert x A*) *Y*
**proof** (*rule injectionsI*)
**show** *Domain R* = *insert x A*
**proof** −
**have** *Domain R* = *Domain P* ∪ *Domain* $\{(x,y)\}$ **using** *paste-Domain R* **by** *metis*
**also have** ... = $A \cup \{x\}$ **using** *Domain-pre* **by** *simp*
**finally show** *?thesis* **by** *auto*
**qed**

**show** *Range R* ⊆ *Y*
**proof** −
**have** *Range R* ⊆ *Range P* ∪ *Range* $\{(x,y)\}$ ∧ *Range P* ∪ *Range* $\{(x,y)\}$ ⊆ $Y \cup \{y\}$
**using** *paste-Range R Range-pre* **by** *force*
**then show** *?thesis* **using** *y* **by** *auto*
**qed**

**show** *runiq R*
**using** *runiq-pre R runiq-singleton-rel runiq-paste2* **by** *fast*

**show** *runiq* $(R^{-1})$
**using** *runiq-conv-pre R y new* **and** *runiq-converse-paste-singleton DiffE Domain-pre*
**by** *metis*
**qed**
**qed**

There are finitely many injective function from a finite set to another finite set.

**lemma** *finite-injections*:
**fixes** *X*::*'a set*
**and** *Y*::*'b set*
**assumes** *finite X*
**and** *finite Y*
**shows** *finite* (*injections X Y*)
**proof** (*rule rev-finite-subset*)
**from** *assms* **show** *finite* (*Pow* $(X \times Y)$) **by** *simp*
**moreover show** *injections X Y* ⊆ *Pow* $(X \times Y)$
**proof**
**fix** *R* **assume** $R \in$ (*injections X Y*)
**then have** *Domain R* = *X* ∧ *Range R* ⊆ *Y* **unfolding** *injections-def* **by** *simp*
**then have** $R \subseteq X \times Y$ **by** *fast*
**then show** $R \in$ *Pow* $(X \times Y)$ **by** *simp*
**qed**
**qed**

The paper-like definition *injections* and the algorithmic definition *injections-alg* are equivalent.

**theorem** *injections-equiv*:
  **fixes** *xs*::′*a list*
    **and** *Y*::′*b*::*linorder set*
  **assumes** *non-empty*: *card Y* > *0*
  **shows** *distinct xs* $\Longrightarrow$ (*set* (*injections-alg xs Y*)::(′*a* × ′*b*) *set set*) = *injections* (*set xs*) *Y*
**proof** (*induct xs*)
  **case** *Nil*
  **have** *set* (*injections-alg* [] *Y*) = {{}::(′*a* × ′*b*) *set*} **by** *simp*
  **also have** . . . = *injections* (*set* []) *Y*
  **proof** −
    **have** {{}} = {*R*::((′*a* × ′*b*) *set*) . *Domain R* = {} ∧ *Range R* ⊆ *Y* ∧ *runiq R* ∧ *runiq* ($R^{-1}$)} (**is** *?LHS* = *?RHS*)
    **proof**
      **have** *Domain* {} = {} **by** (*rule Domain-empty*)
      **moreover have** *Range* {} ⊆ *Y* **by** *simp*
      **moreover note** *runiq-emptyrel*
      **moreover have** *runiq* ($\{\}^{-1}$) **by** (*simp add*: *runiq-emptyrel*)
      **ultimately have** *Domain* {} = {} ∧ *Range* {} ⊆ *Y* ∧ *runiq* {} ∧ *runiq* ($\{\}^{-1}$) **by** *blast*

      **then have** {} ∈ {*R* . *Domain R* = {} ∧ *Range R* ⊆ *Y* ∧ *runiq R* ∧ *runiq* ($R^{-1}$)} **by** (*rule CollectI*)
      **then show** *?LHS* ⊆ *?RHS* **using** *empty-subsetI insert-subset* **by** *fast*
    **next**
      **show** *?RHS* ⊆ *?LHS*
      **proof**
        **fix** *R*
        **assume** *R* ∈ {*R*::((′*a* × ′*b*) *set*) . *Domain R* = {} ∧ *Range R* ⊆ *Y* ∧ *runiq R* ∧ *runiq* ($R^{-1}$)}
        **then show** *R* ∈ {{}} **by** (*simp add*: *Domain-empty-iff*)
      **qed**
    **qed**
    **also have** . . . = *injections* (*set* []) *Y*
      **unfolding** *injections-def* **by** *simp*
    **finally show** *?thesis* **.**
  **qed**
  **finally show** *?case* **.**
**next**
  **case** (*Cons x xs*)

  **from** *non-empty* **have** *finite Y* **by** (*rule card-ge-0-finite*)

  **have** *set* (*injections-alg* (*x* # *xs*) *Y*) = (⋃ { *set* (*sup-rels-from-alg R x Y*) | *R*

. $R \in$ *injections* (*set xs*) $Y$ })
  **using** *Cons.hyps Cons.prems* **by** (*simp add*: *image-Collect-mem*)

**also have** ... = ($\bigcup$ { *sup-rels-from* $R$ $x$ $Y$ | $R$ . $R \in$ *injections* (*set xs*) $Y$ })
  **using** ‹*finite* $Y$› *sup-rels-from-paper-equiv-alg* **by** *fast*

**also have** ... = *injections* (*set* ($x$ # *xs*)) $Y$ **using** *Cons.prems* **by** (*simp add*: *injections-paste*)

**finally show** *?case* **.**
**qed**

**lemma** *Image-within-domain'*: **fixes** $x$ $R$ **shows** $x \in$ *Domain* $R$ = ($R$ `` $\{x\} \neq \{\}$) **by** *blast*

**end**

# 20 Common discrete functions

**theory** *Discrete*
**imports** *Main*
**begin**

## 20.1 Discrete logarithm

**fun** *log* :: *nat* ⇒ *nat* **where**
  [*simp del*]: *log n* = (*if n* < *2 then 0 else Suc* (*log* (*n div 2*)))

**lemma** *log-zero* [*simp*]:
  *log 0* = *0*
  **by** (*simp add*: *log.simps*)

**lemma** *log-one* [*simp*]:
  *log 1* = *0*
  **by** (*simp add*: *log.simps*)

**lemma** *log-Suc-zero* [*simp*]:
  *log* (*Suc 0*) = *0*
  **using** *log-one* **by** *simp*

**lemma** *log-rec*:
  $n \geq 2 \Longrightarrow$ *log n* = *Suc* (*log* (*n div 2*))
  **by** (*simp add*: *log.simps*)

**lemma** *log-twice* [*simp*]:
  $n \neq 0 \Longrightarrow$ *log* ($2 * n$) = *Suc* (*log n*)
  **by** (*simp add*: *log-rec*)

**lemma** *log-half* [*simp*]:
  *log* (*n div 2*) = *log n* − *1*
**proof** (*cases n* < *2*)
  **case** *True*
  **then have** *n* = *0* ∨ *n* = *1* **by** *arith*
  **then show** *?thesis* **by** (*auto simp del*: *One-nat-def*)
**next**
  **case** *False* **then show** *?thesis* **by** (*simp add*: *log-rec*)
**qed**

**lemma** *log-exp* [*simp*]:
  *log* (*2* ˆ *n*) = *n*
  **by** (*induct n*) *simp-all*

**lemma** *log-mono*:
  *mono log*
**proof**
  **fix** *m n* :: *nat*
  **assume** *m* ≤ *n*
  **then show** *log m* ≤ *log n*
  **proof** (*induct m arbitrary*: *n rule*: *log.induct*)
    **case** (*1 m*)
    **then have** *mn2*: *m div 2* ≤ *n div 2* **by** *arith*
    **show** *log m* ≤ *log n*
    **proof** (*cases m* < *2*)
      **case** *True*
      **then have** *m* = *0* ∨ *m* = *1* **by** *arith*
      **then show** *?thesis* **by** (*auto simp del*: *One-nat-def*)
    **next**
      **case** *False*
      **with** *mn2* **have** *m* ≥ *2* **and** *n* ≥ *2* **by** *auto arith*
      **from** *False* **have** *m2-0*: *m div 2* ≠ *0* **by** *arith*
      **with** *mn2* **have** *n2-0*: *n div 2* ≠ *0* **by** *arith*
      **from** *False 1.hyps mn2* **have** *log* (*m div 2*) ≤ *log* (*n div 2*) **by** *blast*
      **with** *m2-0 n2-0* **have** *log* (*2* ∗ (*m div 2*)) ≤ *log* (*2* ∗ (*n div 2*)) **by** *simp*
      **with** *m2-0 n2-0* ‹*m* ≥ *2*› ‹*n* ≥ *2*› **show** *?thesis* **by** (*simp only*: *log-rec* [*of m*] *log-rec* [*of n*]) *simp*
    **qed**
  **qed**
**qed**

## 20.2 Discrete square root

**definition** *sqrt* :: *nat* ⇒ *nat*
**where**
  *sqrt n* = *Max* {*m*. $m^2$ ≤ *n*}

**lemma** *sqrt-aux*:

**fixes** $n$ :: *nat*
**shows** *finite* $\{m.\ m^2 \leq n\}$ **and** $\{m.\ m^2 \leq n\} \neq \{\}$
**proof** −
**{ fix** $m$
**assume** $m^2 \leq n$
**then have** $m \leq n$
**by** (*cases m*) (*simp-all add*: *power2-eq-square*)
**} note** $** = $ *this*
**then have** $\{m.\ m^2 \leq n\} \subseteq \{m.\ m \leq n\}$ **by** *auto*
**then show** *finite* $\{m.\ m^2 \leq n\}$ **by** (*rule finite-subset*) *rule*
**have** $0^2 \leq n$ **by** *simp*
**then show** $*$: $\{m.\ m^2 \leq n\} \neq \{\}$ **by** *blast*
**qed**

**lemma** [*code*]:
*sqrt* $n = $ *Max* (*Set.filter* $(\lambda m.\ m^2 \leq n)$ $\{0..n\}$)
**proof** −
**from** *power2-nat-le-imp-le* [*of - n*] **have** $\{m.\ m \leq n \wedge m^2 \leq n\} = \{m.\ m^2 \leq n\}$ **by** *auto*
**then show** *?thesis* **by** (*simp add*: *sqrt-def Set.filter-def*)
**qed**

**lemma** *sqrt-inverse-power2* [*simp*]:
*sqrt* $(n^2) = n$
**proof** −
**have** $\{m.\ m \leq n\} \neq \{\}$ **by** *auto*
**then have** *Max* $\{m.\ m \leq n\} \leq n$ **by** *auto*
**then show** *?thesis*
**by** (*auto simp add*: *sqrt-def power2-nat-le-eq-le intro*: *antisym*)
**qed**

**lemma** *mono-sqrt*: *mono sqrt*
**proof**
**fix** $m\ n$ :: *nat*
**have** $*$: $0 * 0 \leq m$ **by** *simp*
**assume** $m \leq n$
**then show** *sqrt* $m \leq$ *sqrt* $n$
**by** (*auto intro*!: *Max-mono* ‹$0 * 0 \leq m$› *finite-less-ub simp add*: *power2-eq-square sqrt-def*)
**qed**

**lemma** *sqrt-greater-zero-iff* [*simp*]:
*sqrt* $n > 0 \longleftrightarrow n > 0$
**proof** −
**have** $*$: $0 <$ *Max* $\{m.\ m^2 \leq n\} \longleftrightarrow (\exists\, a \in \{m.\ m^2 \leq n\}.\ 0 < a)$
**by** (*rule Max-gr-iff*) (*fact sqrt-aux*)+
**show** *?thesis*
**proof**
**assume** $0 <$ *sqrt* $n$

then have $0 < Max\ \{m.\ m^2 \leq n\}$ **by** (*simp add*: *sqrt-def*)
with $*$ **show** $0 < n$ **by** (*auto dest*: *power2-nat-le-imp-le*)
**next**
**assume** $0 < n$
**then have** $1^2 \leq n \wedge 0 < (1::nat)$ **by** *simp*
**then have** $\exists\, q.\ q^2 \leq n \wedge 0 < q$ **..**
**with** $*$ **have** $0 < Max\ \{m.\ m^2 \leq n\}$ **by** *blast*
**then show** $0 < sqrt\ n$ **by** (*simp add*: *sqrt-def*)
**qed**
**qed**

**lemma** *sqrt-power2-le* [*simp*]:
$(sqrt\ n)^2 \leq n$
**proof** (*cases* $n > 0$)
**case** *False* **then show** *?thesis* **by** (*simp add*: *sqrt-def*)
**next**
**case** *True* **then have** $sqrt\ n > 0$ **by** *simp*
**then have** *mono* (*times* ($Max\ \{m.\ m^2 \leq n\}$)) **by** (*auto intro*: *mono-times-nat simp add*: *sqrt-def*)
**then have** $*$: $Max\ \{m.\ m^2 \leq n\} * Max\ \{m.\ m^2 \leq n\} = Max\ (times\ (Max\ \{m.\ m^2 \leq n\})\ `\ \{m.\ m^2 \leq n\})$
**using** *sqrt-aux* [*of n*] **by** (*rule mono-Max-commute*)
**have** $Max\ (op * (Max\ \{m.\ m * m \leq n\})\ `\ \{m.\ m * m \leq n\}) \leq n$
**apply** (*subst Max-le-iff*)
**apply** (*metis* (*mono-tags*) *finite-imageI finite-less-ub le-square*)
**apply** *simp*
**apply** (*metis le0 mult-0-right*)
**apply** *auto*
**proof** −
**fix** $q$
**assume** $q * q \leq n$
**show** $Max\ \{m.\ m * m \leq n\} * q \leq n$
**proof** (*cases* $q > 0$)
**case** *False* **then show** *?thesis* **by** *simp*
**next**
**case** *True* **then have** *mono* (*times q*) **by** (*rule mono-times-nat*)
**then have** $q * Max\ \{m.\ m * m \leq n\} = Max\ (times\ q\ `\ \{m.\ m * m \leq n\})$
**using** *sqrt-aux* [*of n*] **by** (*auto simp add*: *power2-eq-square intro*: *mono-Max-commute*)
**then have** $Max\ \{m.\ m * m \leq n\} * q = Max\ (times\ q\ `\ \{m.\ m * m \leq n\})$ **by** (*simp add*: *ac-simps*)
**then show** *?thesis* **apply** *simp*
**apply** (*subst Max-le-iff*)
**apply** *auto*
**apply** (*metis* (*mono-tags*) *finite-imageI finite-less-ub le-square*)
**apply** (*metis* ‹$q * q \leq n$›)
**using** ‹$q * q \leq n$› **by** (*metis le-cases mult-le-mono1 mult-le-mono2 order-trans*)
**qed**
**qed**

**with** $*$ **show** *?thesis* **by** (*simp add*: *sqrt-def power2-eq-square*)
**qed**

**lemma** *sqrt-le*:
  *sqrt n* $\leq$ *n*
  **using** *sqrt-aux* [*of n*] **by** (*auto simp add*: *sqrt-def intro*: *power2-nat-le-imp-le*)

**hide-const** (**open**) *log sqrt*

**end**

# 21 Indicator Function

**theory** *Indicator-Function*
**imports** *Complex-Main*
**begin**

**definition** *indicator S x* = (*if x* $\in$ *S then 1 else 0*)

**lemma** *indicator-simps*[*simp*]:
  *x* $\in$ *S* $\Longrightarrow$ *indicator S x* = *1*
  *x* $\notin$ *S* $\Longrightarrow$ *indicator S x* = *0*
  **unfolding** *indicator-def* **by** *auto*

**lemma** *indicator-pos-le*[*intro*, *simp*]: (*0*::'*a*::*linordered-semidom*) $\leq$ *indicator S x*
  **and** *indicator-le-1*[*intro*, *simp*]: *indicator S x* $\leq$ (*1*::'*a*::*linordered-semidom*)
  **unfolding** *indicator-def* **by** *auto*

**lemma** *indicator-abs-le-1*: |*indicator S x*| $\leq$ (*1*::'*a*::*linordered-idom*)
  **unfolding** *indicator-def* **by** *auto*

**lemma** *indicator-eq-0-iff*: *indicator A x* = (*0*::-::*zero-neq-one*) $\longleftrightarrow$ *x* $\notin$ *A*
  **by** (*auto simp*: *indicator-def*)

**lemma** *indicator-eq-1-iff*: *indicator A x* = (*1*::-::*zero-neq-one*) $\longleftrightarrow$ *x* $\in$ *A*
  **by** (*auto simp*: *indicator-def*)

**lemma** *split-indicator*: *P* (*indicator S x*) $\longleftrightarrow$ ((*x* $\in$ *S* $\longrightarrow$ *P 1*) $\wedge$ (*x* $\notin$ *S* $\longrightarrow$ *P 0*))
  **unfolding** *indicator-def* **by** *auto*

**lemma** *split-indicator-asm*: *P* (*indicator S x*) $\longleftrightarrow$ ($\neg$ (*x* $\in$ *S* $\wedge$ $\neg$ *P 1* $\vee$ *x* $\notin$ *S* $\wedge$ $\neg$ *P 0*))
  **unfolding** *indicator-def* **by** *auto*

**lemma** *indicator-inter-arith*: *indicator* (*A* $\cap$ *B*) *x* = *indicator A x* $*$ (*indicator B x*::'*a*::*semiring-1*)
  **unfolding** *indicator-def* **by** (*auto simp*: *min-def max-def*)

**lemma** *indicator-union-arith*: *indicator* (*A* ∪ *B*) *x* = *indicator A x* + *indicator B x* − *indicator A x* ∗ (*indicator B x*::′*a*::*ring-1*)
**unfolding** *indicator-def* **by** (*auto simp*: *min-def max-def*)

**lemma** *indicator-inter-min*: *indicator* (*A* ∩ *B*) *x* = *min* (*indicator A x*) (*indicator B x*::′*a*::*linordered-semidom*)
**and** *indicator-union-max*: *indicator* (*A* ∪ *B*) *x* = *max* (*indicator A x*) (*indicator B x*::′*a*::*linordered-semidom*)
**unfolding** *indicator-def* **by** (*auto simp*: *min-def max-def*)

**lemma** *indicator-disj-union*: *A* ∩ *B* = {} ⟹ *indicator* (*A* ∪ *B*) *x* = (*indicator A x* + *indicator B x*::′*a*::*linordered-semidom*)
**by** (*auto split*: *split-indicator*)

**lemma** *indicator-compl*: *indicator* (− *A*) *x* = *1* − (*indicator A x*::′*a*::*ring-1*)
**and** *indicator-diff*: *indicator* (*A* − *B*) *x* = *indicator A x* ∗ (*1* − *indicator B x*::′*a*::*ring-1*)
**unfolding** *indicator-def* **by** (*auto simp*: *min-def max-def*)

**lemma** *indicator-times*: *indicator* (*A* × *B*) *x* = *indicator A* (*fst x*) ∗ (*indicator B* (*snd x*)::′*a*::*semiring-1*)
**unfolding** *indicator-def* **by** (*cases x*) *auto*

**lemma** *indicator-sum*: *indicator* (*A* <+> *B*) *x* = (*case x of Inl x* ⇒ *indicator A x* | *Inr x* ⇒ *indicator B x*)
**unfolding** *indicator-def* **by** (*cases x*) *auto*

**lemma**
**fixes** *f* :: ′*a* ⇒ ′*b*::*semiring-1* **assumes** *finite A*
**shows** *setsum-mult-indicator*[*simp*]: (∑ *x* ∈ *A*. *f x* ∗ *indicator B x*) = (∑ *x* ∈ *A* ∩ *B*. *f x*)
**and** *setsum-indicator-mult*[*simp*]: (∑ *x* ∈ *A*. *indicator B x* ∗ *f x*) = (∑ *x* ∈ *A* ∩ *B*. *f x*)
**unfolding** *indicator-def*
**using** *assms* **by** (*auto intro*!: *setsum.mono-neutral-cong-right split*: *split-if-asm*)

**lemma** *setsum-indicator-eq-card*:
**assumes** *finite A*
**shows** (*SUM x* : *A*. *indicator B x*) = *card* (*A Int B*)
**using** *setsum-mult-indicator*[*OF assms*, *of* %*x*. *1*::*nat*]
**unfolding** *card-eq-setsum* **by** *simp*

**lemma** *setsum-indicator-scaleR*[*simp*]:
*finite A* ⟹
(∑ *x* ∈ *A*. *indicator* (*B x*) (*g x*) $*_R$ *f x*) = (∑ *x* ∈ {*x*∈*A*. *g x* ∈ *B x*}. *f x*::′*a*::*real-vector*)
**using** *assms* **by** (*auto intro*!: *setsum.mono-neutral-cong-right split*: *split-if-asm simp*: *indicator-def*)

**lemma** *LIMSEQ-indicator-incseq*:
**assumes** *incseq A*
**shows** $(\lambda i.\ indicator\ (A\ i)\ x :: 'a :: \{topological\text{-}space,\ one,\ zero\}) \ {-}{-}{-}{-}{>}$ $indicator\ (\bigcup i.\ A\ i)\ x$
**proof** *cases*
**assume** $\exists i.\ x \in A\ i$
**then obtain** $i$ **where** $x \in A\ i$
**by** *auto*
**then have**
$\bigwedge n.\ (indicator\ (A\ (n + i))\ x :: 'a) = 1$
$(indicator\ (\bigcup\ i.\ A\ i)\ x :: 'a) = 1$
**using** *incseqD*[*OF* ‹*incseq A*›, *of i n* + *i* **for** *n*] ‹$x \in A\ i$› **by** (*auto simp*: *indicator-def*)
**then show** *?thesis*
**by** (*rule-tac LIMSEQ-offset*[*of* - *i*]) (*simp add*: *tendsto-const*)
**qed** (*auto simp*: *indicator-def tendsto-const*)

**lemma** *LIMSEQ-indicator-UN*:
$(\lambda k.\ indicator\ (\bigcup\ i{<}k.\ A\ i)\ x :: 'a :: \{topological\text{-}space,\ one,\ zero\}) \ {-}{-}{-}{-}{>}$ $indicator\ (\bigcup i.\ A\ i)\ x$
**proof** –
**have** $(\lambda k.\ indicator\ (\bigcup\ i{<}k.\ A\ i)\ x{::}'a) \ {-}{-}{-}{-}{>}\ indicator\ (\bigcup k.\ \bigcup\ i{<}k.\ A\ i)$ $x$
**by** (*intro LIMSEQ-indicator-incseq*) (*auto simp*: *incseq-def intro*: *less-le-trans*)
**also have** $(\bigcup k.\ \bigcup\ i{<}k.\ A\ i) = (\bigcup i.\ A\ i)$
**by** *auto*
**finally show** *?thesis* **.**
**qed**

**lemma** *LIMSEQ-indicator-decseq*:
**assumes** *decseq A*
**shows** $(\lambda i.\ indicator\ (A\ i)\ x :: 'a :: \{topological\text{-}space,\ one,\ zero\}) \ {-}{-}{-}{-}{>}$ $indicator\ (\bigcap i.\ A\ i)\ x$
**proof** *cases*
**assume** $\exists i.\ x \notin A\ i$
**then obtain** $i$ **where** $x \notin A\ i$
**by** *auto*
**then have**
$\bigwedge n.\ (indicator\ (A\ (n + i))\ x :: 'a) = 0$
$(indicator\ (\bigcap i.\ A\ i)\ x :: 'a) = 0$
**using** *decseqD*[*OF* ‹*decseq A*›, *of i n* + *i* **for** *n*] ‹$x \notin A\ i$› **by** (*auto simp*: *indicator-def*)
**then show** *?thesis*
**by** (*rule-tac LIMSEQ-offset*[*of* - *i*]) (*simp add*: *tendsto-const*)
**qed** (*auto simp*: *indicator-def tendsto-const*)

**lemma** *LIMSEQ-indicator-INT*:
$(\lambda k.\ indicator\ (\bigcap i{<}k.\ A\ i)\ x :: 'a :: \{topological\text{-}space,\ one,\ zero\}) \ {-}{-}{-}{-}{>}$ $indicator\ (\bigcap i.\ A\ i)\ x$

**proof** −
  **have** ($\lambda k.$ *indicator* ($\bigcap i<k.$ $A$ $i$) $x$::$'a$) $----{>}$ *indicator* ($\bigcap k.$ $\bigcap i<k.$ $A$ $i$) $x$
    **by** (*intro LIMSEQ-indicator-decseq*) (*auto simp*: *decseq-def intro*: *less-le-trans*)
  **also have** ($\bigcap k.$ $\bigcap$ $i<k.$ $A$ $i$) = ($\bigcap i.$ $A$ $i$)
    **by** *auto*
  **finally show** *?thesis* **.**
**qed**

**lemma** *indicator-add*:
  $A \cap B = \{\} \Longrightarrow$ (*indicator* $A$ $x$::-::*monoid-add*) + *indicator* $B$ $x$ = *indicator* ($A$ $\cup$ $B$) $x$
  **unfolding** *indicator-def* **by** *auto*

**lemma** *of-real-indicator*: *of-real* (*indicator* $A$ $x$) = *indicator* $A$ $x$
  **by** (*simp split*: *split-indicator*)

**lemma** *real-of-nat-indicator*: *real* (*indicator* $A$ $x$ :: *nat*) = *indicator* $A$ $x$
  **by** (*simp split*: *split-indicator*)

**lemma** *abs-indicator*: |*indicator* $A$ $x$ :: $'a$::*linordered-idom*| = *indicator* $A$ $x$
  **by** (*simp split*: *split-indicator*)

**lemma** *mult-indicator-subset*:
  $A \subseteq B \Longrightarrow$ *indicator* $A$ $x$ $*$ *indicator* $B$ $x$ = (*indicator* $A$ $x$ :: $'a$::{*comm-semiring-1*})
  **by** (*auto split*: *split-indicator simp*: *fun-eq-iff*)

**lemma** *indicator-sums*:
  **assumes** $\bigwedge i\ j.\ i \neq j \Longrightarrow A\ i \cap A\ j = \{\}$
  **shows** ($\lambda i.$ *indicator* ($A$ $i$) $x$::*real*) *sums* *indicator* ($\bigcup i.$ $A$ $i$) $x$
**proof** *cases*
  **assume** $\exists\, i.\ x \in A\ i$
  **then obtain** $i$ **where** $i$: $x \in A\ i$ **..**
  **with** *assms* **have** ($\lambda i.$ *indicator* ($A$ $i$) $x$::*real*) *sums* ($\sum i{\in}\{i\}.$ *indicator* ($A$ $i$) $x$)
    **by** (*intro sums-finite*) (*auto split*: *split-indicator*)
  **also have** ($\sum i{\in}\{i\}.$ *indicator* ($A$ $i$) $x$) = *indicator* ($\bigcup i.$ $A$ $i$) $x$
    **using** $i$ **by** (*auto split*: *split-indicator*)
  **finally show** *?thesis* **.**
**qed** *simp*

**end**

# 22 Locus where a function or a list (of linord type) attains its maximum value

**theory** *Argmax*
**imports** *Main*

**begin**

the subset of elements of a set where a function reaches its maximum

**fun** *argmax* :: $('a \Rightarrow 'b::linorder) \Rightarrow 'a\ set \Rightarrow 'a\ set$
**where** *argmax f A* = $\{\ x \in A\ .\ f\ x = Max\ (f\ `\ A)\ \}$

**lemma** *mm79*: *argmax f A* = $A \cap f\ -`\ \{Max\ (f\ `\ A)\}$ **by** *force*
**lemma** *mm86b*: **assumes** $y \in f`A$ **shows** $A \cap f\ -`\ \{y\} \neq \{\}$ **using** *assms* **by** *blast*

The arg max of a function over a non-empty set is non-empty.

**corollary** *argmax-non-empty-iff*: **assumes** *finite X X* $\neq \{\}$ **shows** *argmax f X* $\neq\{\}$
**using** *assms Max-in finite-imageI image-is-empty mm79 mm86b* **by** (*metis*(*no-types*))

We want the elements of a list satisfying a given predicate; but, rather than returning them directly, we return the (sorted) list of their indices. This is done, in different ways, by *filterpositions* and *filterpositions2*.

**definition** *filterpositions*

:: ('*a* => *bool*) => '*a list* => *nat list*
**where** *filterpositions P l* = *map snd* (*filter* (*P o fst*) (*zip l* (*upt 0* (*size l*))))

**definition** *filterpositions2*
:: ('*a* => *bool*) => '*a list* => *nat list*
**where** *filterpositions2 P l* = [*n*. *n* ← [*0*..<*size l*], *P* (*l*!*n*)]

**definition** *maxpositions* :: '*a*::*linorder list* => *nat list* **where**
*maxpositions l* = *filterpositions2* (%*x* . *x* ≥ *Max* (*set l*)) *l*

**lemma** *ll5*: *maxpositions l* = [*n*. *n*←[*0*..<*size l*], *l*!*n* ≥ *Max*(*set l*)]
**using** *assms* **unfolding** *maxpositions-def filterpositions2-def* **by** *fastforce*

**definition** *argmaxList*
:: ('*a* => ('*b*::*linorder*)) => '*a list* => '*a list*
**where** *argmaxList f l* = *map* (*nth l*) (*maxpositions* (*map f l*))

**lemma** [*n* . *n* <− [*0*..<*m*], (*n* ∈ *set* [*0*..<*m*] & *P n*)]
= [*n* . *n* <− [*0*..<*m*], *n* ∈ *set* [*0*..<*m*], *P n*] **by** *meson*

**lemma** *ll7b*: [*n* . *n* <− *l*, *P n*] = [*n* . *n* <− *l*, *n* ∈ *set l*, *P n*]
**proof** −

**have** *map* (λ*uu*. *if P uu then* [*uu*] *else* []) *l* =
*map* (λ*uu*. *if uu* ∈ *set l then if P uu then* [*uu*] *else* [] *else* []) *l* **by** *simp*
**thus** *concat* (*map* (λ*n*. *if P n then* [*n*] *else* []) *l*) =

*concat* (*map* ($\lambda n$. *if* $n \in$ *set* $l$ *then if* $P$ $n$ *then* [$n$] *else* [] *else* []) $l$) **by** *presburger*
**qed**

**lemma** *ll7*: [$n$ . $n$ <− [*0*..<$m$], $P$ $n$] = [$n$ . $n$ <− [*0*..<$m$], $n \in$ *set* [*0*..<$m$], $P$ $n$]
**using** *ll7b* **by** *fast*

**lemma** *ll10*: **fixes** $f$ $m$ $P$ **shows** (*map* $f$ [$n$ . $n$ <− [*0*..<$m$], $P$ $n$]) = [ $f$ $n$ . $n$ <− [*0*..<$m$], $P$ $n$]
**by** (*induct* $m$) *auto*

**lemma** *map-commutes-a*: [$f$ $n$ . $n$ <− [], $Q$ ($f$ $n$)] = [$x$ <− (*map* $f$ []). $Q$ $x$] **by** *simp*

**lemma** *map-commutes-b*: $\forall$ $x$ *xs*. ([$f$ $n$ . $n$ <− *xs*, $Q$ ($f$ $n$)] = [$x$ <− (*map* $f$ *xs*). $Q$ $x$]
$\longrightarrow$ [$f$ $n$ . $n$ <− ($x$#*xs*), $Q$ ($f$ $n$)] = [$x$ <− (*map* $f$ ($x$#*xs*)). $Q$ $x$]) **using** *assms* **by** *simp*

**lemma** *myStructInduct*: **assumes** $P$ [] $\forall$ $x$ *xs*. $P$ (*xs*) $\longrightarrow$ $P$ ($x$#*xs*) **shows** $P$ $l$
**using** *assms list-nonempty-induct* **by** (*metis*)

**lemma** *map-commutes*: **fixes** $f$::'*a* => '*b* **fixes** $Q$::'*b* => *bool* **fixes** *xs*::'*a list*
**shows** [$f$ $n$ . $n$ <− *xs*, $Q$ ($f$ $n$)] = [$x$ <− (*map* $f$ *xs*). $Q$ $x$]
**using** *map-commutes-a map-commutes-b myStructInduct* **by** *fast*

**lemma** *ll9*: **fixes** $f$ $l$ **shows** *maxpositions* (*map* $f$ $l$) =
[$n$ . $n$ <− [*0*..<*size* $l$], $f$ ($l$!$n$) $\geq$ *Max* ($f$‘(*set* $l$))] (**is** *maxpositions* (*?fl*) = -)
**proof** −
  **have** *maxpositions ?fl* =
  [$n$. $n$ <− [*0*..<*size ?fl*], $n\in$ *set*[*0*..<*size ?fl*], *?fl*!$n$ $\geq$ *Max* (*set ?fl*)]
  **using** *ll7b* **unfolding** *filterpositions2-def maxpositions-def* **.**
  **also have** ... =
  [$n$ . $n$ <− [*0*..<*size* $l$], ($n$<*size* $l$), (*?fl*!$n$ $\geq$ *Max* (*set ?fl*))] **by** *simp*
  **also have** ... =
  [$n$ . $n$ <− [*0*..<*size* $l$], ($n$<*size* $l$) $\wedge$ ($f$ ($l$!$n$) $\geq$ *Max* (*set ?fl*))]
  **using** *nth-map* **by** (*metis* (*poly-guards-query*, *hide-lams*)) **also have** ... =
  [$n$ . $n$ <− [*0*..<*size* $l$], ($n\in$ *set* [*0*..<*size* $l$]),($f$ ($l$!$n$) $\geq$ *Max* (*set ?fl*))]
  **using** *atLeastLessThan-iff le0 set-upt* **by** (*metis*(*no-types*))
  **also have** ... =
  [$n$ . $n$ <− [*0*..<*size* $l$], $f$ ($l$!$n$) $\geq$ *Max* (*set ?fl*)] **using** *ll7* **by** *presburger*
  **finally show** *?thesis* **by** *auto*
**qed**

**lemma** *ll11*: **fixes** $f$ $l$ **shows** *argmaxList* $f$ $l$ = [ $l$!$n$ . $n$ <− [*0*..<*size* $l$], $f$ ($l$!$n$) $\geq$ *Max* ($f$‘(*set* $l$))]
**unfolding** *ll9 argmaxList-def* **by** (*metis ll10*)

**theorem** *argmaxadequacy*:

**fixes** *f*::*'a => ('b::linorder)* **fixes** *l*::*'a list* **shows**
*argmaxList f l* = [ *x* <− *l*. *f x* ≥ *Max* (*f*‘(*set l*))] (**is** *?lh*=-)
**proof** −
  **let** *?P*=% *y*::(*'b*::*linorder*) . *y* ≥ *Max* (*f*‘(*set l*))
  **let** *?mh*=[*nth l n* . *n* <− [*0*..<*size l*], *?P* (*f* (*nth l n*))]
  **let** *?rh*=[ *x* <− (*map* (*nth l*) [*0*..<*size l*]). *?P* (*f x*)]
  **have** *?lh* = *?mh* **using** *ll11* **by** *fast*
  **also have** ... = *?rh* **using** *map-commutes* **by** *fast*
  **also have** ...= [*x* <− *l*. *?P* (*f x*)] **using** *map-nth* **by** *metis*
  **finally show** *?thesis* **by** *force*
**qed**

**end**

# 23 Toolbox of various definitions and theorems about sets, relations and lists

**theory** *MiscTools*

**imports**
*RelationProperties*
*~~/src/HOL/Library/Discrete*
*Main*
*RelationOperators*
*~~/src/HOL/Library/Code-Target-Nat*
*~~/src/HOL/Library/Indicator-Function*
*Argmax*

**begin**

# 24 Facts and notations about relations, sets and functions.

**notation** *paste* (**infix** +< *75*)

+< abbreviation permits to shorten the notation for altering a function in a single point.

**abbreviation** *singlepaste* **where** *singlepaste F f* == *F* +∗ {(*fst f*, *snd f*)}
**notation** *singlepaste* (**infix** +< *75*)

−− abbreviation permits to shorten the notation for considering a function outside a single point.

**abbreviation** *singleoutside* (**infix** −− *75*) **where** *f* −− *x* ≡ *f outside* {*x*}

*ler-ni* inverts *in-rel*

**abbreviation** *ler-ni* **where** *ler-ni* $r == (\bigcup x.\ (\{x\} \times (r\ x -` \{True\})))$

Turns a HOL function into a set-theoretical function

**definition**
$Graph\ f = \{(x, f\ x) \mid x\ .\ True\}$

Inverts *Graph* (which is equivalently done by *op* ,,).

**definition** $toFunction\ R = (\lambda\ x\ .\ (R\ ,,\ x))$

**lemma** $toFunction = eval\text{-}rel$ **using** *toFunction-def eval-rel-def* **by** *blast*

**lemma** *lll40*: $(P \cup Q) \mid\mid X = (P \mid\mid X) \cup (Q \mid\mid X)$ **unfolding** *restrict-def* **using** *assms* **by** *blast*

Behaves like P +* Q (paste), but without enlarging P's Domain. Compare with *fun-upd*

**definition** *update* **where** $update\ P\ Q = P +* (Q \mid\mid (Domain\ P))$
**notation** *update* (**infix** +^ *75*)

**definition** *runiqer*
$::('a \times 'b)\ set => ('a \times 'b)\ set$

**where** $runiqer\ R = \{\ (x,\ THE\ y.\ y \in R\ ``\ \{x\}) \mid x.\ x \in Domain\ R\ \}$

Like *Graph*, but with a built-in restriction to a given set *X*. This makes it more computable than the equivalent *Graph f* || *X*. Duplicates the eponymous definition found in *Function-Order*, which is otherwise unneeded.

**definition** *graph* **where** $graph\ X\ f = \{(x, f\ x) \mid x.\ x \in X\}$

**lemma** *lm024a*: **assumes** *runiq R* **shows** $R \supseteq graph\ (Domain\ R)\ (toFunction\ R)$

**unfolding** *graph-def toFunction-def*
**using** *assms graph-def toFunction-def eval-runiq-rel* **by** *fastforce*

**lemma** *lm024b*: **assumes** *runiq R* **shows** $R \subseteq graph\ (Domain\ R)\ (toFunction\ R)$

**unfolding** *graph-def toFunction-def*
**using** *assms eval-runiq-rel runiq-basic Domain.DomainI mem-Collect-eq subrelI* **by** *fastforce*

**lemma** *lm024*: **assumes** *runiq R* **shows** $R = graph\ (Domain\ R)\ (toFunction\ R)$
**using** *assms lm024a lm024b* **by** *fast*

**lemma** *ll37*: $runiq(graph\ X\ f)\ \&\ Domain(graph\ X\ f) = X$ **unfolding** *graph-def* **using** *l14* **by** *fast*

**abbreviation** *eval-rel2* $(R::('a \times ('b\ set))\ set)\ (x::'a) == \bigcup\ (R``\{x\})$

**notation** *eval-rel2* (**infix** ,,, *75*)

**lemma** *lll82*: **assumes** *runiq* ($f$::(('$a$ × ('$b$ *set*)) *set*)) $x$ ∈ *Domain* $f$ **shows** $f$,,$x$ = $f$,,,$x$
**using** *assms Image-runiq-eq-eval cSup-singleton* **by** *metis*

**lemma** *ll36*: *Graph* $f$=*graph UNIV* $f$ **unfolding** *Graph-def graph-def* **by** *simp*

**lemma** *lm04*: *graph* ($X$ ∩ $Y$) $f$ ⊆ *graph* $X$ $f$ || $Y$ **unfolding** *graph-def*
**using** *Int-iff mem-Collect-eq restrict-ext subrelI* **by** *auto*

**lemma** *ll14*: **assumes** $x$∈*Domain* $f$ *runiq* $f$ **shows** $f$,,$x$ ∈ *Range* $f$
**using** *assms* **by** (*metis Range-iff eval-runiq-rel*)

**definition** *runiqs* **where** *runiqs*={$f$. *runiq* $f$}

**lemma** *l37a*: ($P$ *outside* $X$) *outside* $Y$=$P$ *outside* ($X$∪$Y$) **unfolding** *Outside-def*
**by** *blast*

**corollary** *l37*: $(P\ outside\ X)\ outside\ X = P\ outside\ X$ **using** *l37a* **by** *force*

**lemma** *l38a*: **assumes** $X \cap Domain\ P \subseteq Domain\ Q$ **shows**
$P +* Q = (P\ outside\ X) +* Q$ **unfolding** *paste-def Outside-def* **using** *assms* **by** *blast*

**corollary** *l38*: $P +* Q = (P\ outside\ (Domain\ Q)) +* Q$ **using** *l38a* **by** *fast*

**corollary** *l39*: $R = (R\ outside\ \{x\}) \cup (\{x\} \times (R\ ``\ \{x\}))$
**using** *restrict-to-singleton outside-union-restrict* **by** *metis*

**corollary** *l40*: $R = (R\ outside\ \{x\}) +* (\{x\} \times (R\ ``\ \{x\}))$
**by** (*metis paste-outside-restrict restrict-to-singleton*)

**lemma** *ll83*: $R \subseteq R +* (\{x\} \times (R``\{x\}))$ **using**
*l40 l38 paste-def Outside-def* **by** *fast*

**lemma** *ll82*: $R \supseteq R{+*}(\{x\} \times (R``\{x\}))$ **by** (*metis*
*Un-least Un-upper1 outside-union-restrict paste-def restrict-to-singleton restriction-is-subrel*)

**corollary** *ll84*: $R = R +* (\{x\} \times (R``\{x\}))$ **using** *ll82 ll83* **by** *force*

**lemma** *lll59*: **assumes** *trivial* $Y$ **shows** *runiq* $(X \times Y)$ **using** *assms*
*runiq-def Image-subset ll84 trivial-subset ll83* **by** (*metis*(*no-types*))

**lemma** *mm14b*: *runiq* $((X \times \{x\}) +* (Y \times \{y\}))$ **using** *assms lll59 trivial-singleton runiq-paste2* **by** *metis*

**lemma** *lll11b*: $P\ ||\ (X \cap Y) \subseteq P||X\ \&\ P\ outside\ (X \cup Y) \subseteq P\ outside\ X$ **using**

*Outside-def restrict-def Sigma-Un-distrib1 Un-upper1 inf-mono Diff-mono subset-refl* **by** (*metis* (*lifting*) *Sigma-mono inf-le1*)

**lemma** *lll11*: $P\ ||\ X \subseteq P||(X \cup Y)\ \&\ P\ outside\ X \subseteq P\ outside\ (X \cap Y)$
**using** *lll11b distrib-sup-le sup-idem*
*le-inf-iff subset-antisym sup-commute*
**by** (*metis sup-ge1*)

**lemma** *lll84*: $P``(X \cap Domain\ P) = P``X$ **by** *blast*

**lemma** *nn57*: **assumes** *card* $X=1$ $X \subseteq Y$ **shows** *Union* $X \in Y$ **using** *assms nn56* **by** (*metis cSup-singleton insert-subset*)
**lemma** *nn57b*: **assumes** *card* $X=1$ $X \subseteq Y$ **shows** *the-elem* $X \in Y$ **using** *assms*

**by** (*metis* (*full-types*) *insert-subset nn56*)

**lemma** *ll52*: $P\ outside\ (X \cup Y) = (P\ outside\ X)\ outside\ Y$ **unfolding** *Outside-def* **by** *blast*

**corollary** *ll52b*: (*R outside X1*) *outside X2* = (*R outside X2*) *outside X1* **by** (*metis ll52 sup-commute*)
**lemma assumes** *card* $X=1$ **shows** $X=\{$*the-elem X*$\}$ **using** *assms card-eq-SucD the-elem-eq* **by** *fastforce*
**lemma assumes** *card* $X \geq 1$ $\forall x \in X.\ y > x$ **shows** $y >$ *Max X* **using** *assms*
**by** (*metis* (*poly-guards-query*) *Max-in One-nat-def card-eq-0-iff lessI not-le*)

**lemma** *mm80a*: **assumes** *finite X* $mx \in X$ $f\ x < f\ mx$ **shows**$x \notin$ *argmax f X*
**using** *assms not-less* **by** *fastforce*

**lemma** *mm80d*: **assumes** *finite X* $mx \in X$ $\forall x \in X-\{mx\}.\ f\ x < f\ mx$ **shows** *argmax f X* $\subseteq \{mx\}$
**using** *assms mk-disjoint-insert* **by** *force*

**lemma** *mm80*: **assumes** *finite X* $mx \in X$ $\forall x \in X-\{mx\}.\ f\ x < f\ mx$ **shows** *argmax f X* $= \{mx\}$
**using** *assms mm80d* **by** (*metis argmax-non-empty-iff equals0D subset-singletonD*)

**corollary** *mm80c*: (*finite X* & $mx \in X$ & $(\forall aa \in X-\{mx\}.\ f\ aa < f\ mx)) \longrightarrow$ *argmax f X* $= \{mx\}$
**using** *assms mm80* **by** *metis*

**corollary** *mm80b*: **assumes** *finite X* $mx \in X$ $\forall x \in X.\ x \neq mx \longrightarrow f\ x < f\ mx$ **shows** *argmax f X* $= \{mx\}$
**using** *assms mm80* **by** (*metis Diff-iff insertI1*)

**lemma** *mm75f*: **assumes** $f \circ g=id$ **shows** *inj-on g UNIV* **using** *assms*
**by** (*metis inj-on-id inj-on-imageI2*)

**lemma** *mm74a*: **assumes** *inj-on f X* **shows** *inj-on* (*image f*) (*Pow X*)
**using** *assms inj-on-image-eq-iff inj-onI PowD* **by** (*metis* (*mono-tags*, *lifting*))

**lemma** *mm74*: **assumes** *inj-on f Y* $X \subseteq Y$ **shows** *inj-on* (*image f*) (*Pow X*)
**using** *assms mm74a* **by** (*metis subset-inj-on*)

**definition** *finestpart* **where** *finestpart X* = (%*x*. *insert x* {}) ‘ *X*

**lemma** *ll64*: *finestpart X* $= \{\{x\}|x\ .\ x \in X\}$ **unfolding** *finestpart-def* **by** *blast*

**lemma** *mm75*: $X=\bigcup$ (*finestpart X*) **using** *ll64* **by** *auto*
**lemma** *mm75b*: *Union* $\circ$ *finestpart* = *id* **using** *finestpart-def mm75* **by** *fastforce*
**lemma** *mm75c*: *inj-on Union* (*finestpart* ‘ *UNIV*) **using** *assms mm75b* **by** (*metis inj-on-id inj-on-imageI*)
**lemma** *mm31*: **assumes** $X \neq Y$ **shows** $\{\{x\}|\ x.\ x \in X\} \neq \{\{x\}|\ x.\ x \in Y\}$ **using** *assms* **by** *auto*
**corollary** *mm31b*: *inj-on finestpart UNIV* **using** *mm31 ll64* **by** (*metis* (*lifting*, *no-types*) *injI*)

**lemma** *mm55m*: $\{Y.\ EX\ y.((Y \in finestpart\ y)\ \&\ (y \in Range\ a))\} = \bigcup (finestpart`(Range\ a))$ **by** *auto*

**lemma** *mm55l*: $Range\ \{(fst\ pair,\ Y)|\ Y\ pair.\ Y \in finestpart\ (snd\ pair)\ \&\ pair \in a\} =$
$\{Y.\ EX\ y.\ ((Y \in finestpart\ y)\ \&\ (y \in Range\ a))\}$ **by** *auto*

**lemma** *mm55j*: $\{(fst\ pair,\ \{y\})|\ y\ pair.\ y \in snd\ pair\ \&\ pair \in a\} =$
$\{(fst\ pair,\ Y)|\ Y\ pair.\ Y \in finestpart\ (snd\ pair)\ \&\ pair \in a\}$
**using** *finestpart-def* **by** *fastforce*

**lemma** *mm55b*: $\{(fst\ pair,\ \{y\})|\ y.\ y \in\ snd\ pair\} = \{fst\ pair\} \times \{\{y\}|\ y.\ y \in snd\ pair\}$ **by** *fastforce*

**lemma** *mm71*: $x \in X = (\{x\} \in finestpart\ X)$ **using** *finestpart-def* **by** *force*

**lemma** *nn43*: $\{(x,X)\}-\{(x,\{\})\} = \{x\}\times(\{X\}-\{\{\}\})$ **by** *blast*

**lemma** *nn11*: **assumes** $\bigcup\ P = X$ **shows** $P \subseteq Pow\ X$ **using** *assms* **by** *blast*

**lemma** *mm85*: $argmax\ f\ \{x\} = \{x\}$ **using** *argmax-def* **by** *auto*

**lemma** *lm64*: **assumes** *finite X* **shows** *set* (*sorted-list-of-set X*)$=X$ **using** *assms* **by** *simp*

**lemma** *lll23*: **assumes** *finite A* **shows** $setsum\ f\ A = setsum\ f\ (A \cap B) + setsum\ f\ (A - B)$ **using**
*assms* **by** (*metis DiffD2 Int-iff Un-Diff-Int Un-commute finite-Un setsum.union-inter-neutral*)

**lemma** *ll54*: **assumes** $(Domain\ P \subseteq Domain\ Q)$ **shows** $(P +* Q=Q)$
**unfolding** *paste-def Outside-def* **using** *assms* **by** *fast*

**lemma** *ll55*: **assumes** $(P +* Q=Q)$ **shows** $(Domain\ P \subseteq Domain\ Q)$
**using** *assms paste-def Outside-def* **by** *blast*

**lemma** *ll56*: $(Domain\ P \subseteq Domain\ Q) = (P +* Q=Q)$ **using** *ll54 ll55* **by** *metis*

**lemma** $(P||(Domain\ Q)) +* Q = Q$ **by** (*metis Int-lower2 ll41 ll56*)

**lemma** *lll00*: $P||X = P\ outside\ (Domain\ P - X)$
**using** *Outside-def restrict-def* **by** *fastforce*
**lemma** *lll01b*: $P\ outside\ X \subseteq P\ ||\ ((Domain\ P)-X)$
**using** *lll00 lll11* **by** (*metis Int-commute ll41 outside-reduces-domain*)

**lemma** *lll06a*: **shows** $Domain\ (P\ outside\ X) \cap Domain\ (Q\ ||\ X) \subseteq \{\}$ **using** *lll00* **by**
(*metis Diff-disjoint Domain-empty-iff Int-Diff inf-commute ll41 outside-reduces-domain restrict-empty subset-empty*)

**lemma** *lll06b*: **shows** $(P\ outside\ X) \cap (Q\ ||\ X) = \{\}$ **using** *lll06a* **by** *fast*

**lemma** *lll06c*: **shows** $(P\ outside\ (X \cup Y)) \cap (Q\ ||\ (X)) = \{\}\ \&$
$(P\ outside\ (X)) \cap (Q\ ||\ (X \cap Z)) = \{\}$
**using** *assms Outside-def restrict-def lll06b lll11b* **by** *fast*

**lemma** *lll01*: **shows** $P\ outside\ X = P\ ||\ (Domain\ P\ -X)$
**using** *Outside-def restrict-def lll01b* **by** *fast*

**lemma** *lll99*: $R``(X-Y) = (R||X)``(X-Y)$ **using** *restrict-def* **by** *blast*

**lemma** *lll79*: **assumes** $\bigcup\ XX \subseteq X\ x \in XX\ x \neq \{\}$ **shows** $x \cap X \neq \{\}$ **using** *assms* **by** *blast*

**lemma** *lm66*: **assumes** $\forall\, l \in set\ (g1\ G).\ set\ (g2\ l\ N) = f2\ (set\ l)\ N$ **shows**
$set\ [set\ (g2\ l\ N).\ l <-\ g1\ G] = \{f2\ P\ N|\ P.\ P \in set\ (map\ set\ (g1\ G))\}$ **using** *assms* **by** *auto*
**lemma** *lm66b*: $(\forall\, l \in set\ (g1\ G).\ set\ (g2\ l\ N) = f2\ (set\ l)\ N)\ -->$
$\{f2\ P\ N|\ P.\ P \in set\ (map\ set\ (g1\ G))\} = set\ [set\ (g2\ l\ N).\ l <-\ g1\ G]$ **by** *auto*

**lemma** *lll86*: **assumes** $X \cap Y=\{\}$ **shows** $R``X = (R\ outside\ Y)``X$
**using** *assms Outside-def Image-def* **by** *blast*

**lemma** *lm02*: $argmax\ f\ A = \{\ x \in A\ .\ f\ x = Max\ (f\ `\ A)\ \}$ **using** *assms* **by** *simp*

**abbreviation** *mylog n* == (*if* (*n* ≠ *0*) *then* (*Discrete.log n*) *else* (−*1*))
**abbreviation** *Card X* == *mylog* (*card* (*Pow X*))
**lemma assumes** *finite X* **shows** *Card X* = *card X* (**is** *?L=?R*) **using** *assms*
**proof** −
**have** *Card X=Discrete.log* (*card* (*Pow X*)) **using** *assms* **by** *auto*
**moreover have** ... = *Discrete.log* (*2*ˆ*card X*) **using** *assms* **by** (*metis* (*poly-guards-query*) *card-Pow*)
**ultimately show** *?thesis* **by** *fastforce*
**qed**

**lemma assumes** ¬ (*finite X*) **shows** *Card X*=−*1* **using** *assms* **by** *simp*

**lemma** *lll77*: **assumes** *Range P* ∩ (*Range Q*)={} *runiq* (*P*ˆ−*1*) *runiq* (*Q*ˆ−*1*)
**shows** *runiq* ((*P* ∪ *Q*)ˆ−*1*)
**using** *assms* **by** (*metis Domain-converse converse-Un disj-Un-runiq*)

**lemma** *lll77b*: **assumes** *Range P* ∩ (*Range Q*)={} *runiq* (*P*ˆ−*1*) *runiq* (*Q*ˆ−*1*)
**shows** *runiq* ((*P* +∗ *Q*)ˆ−*1*)
**using** *lll77 assms subrel-runiq* **by** (*metis converse-converse converse-subset-swap paste-sub-Un*)

**lemma** *l32*: *runiq* $R$ = (∀ $x$ . *trivial* ($R$``{$x$}))
**using** *assms* **by** (*metis runiq-alt*)

**lemma** *lm014*: **assumes** *runiq* $R$ **shows** *card* ($R$ `` {$a$}) = *1* ⟷ $a$ ∈ *Domain* $R$
**using** *assms card-Suc-eq One-nat-def* **by** (*metis Image-within-domain′ Suc-neq-Zero assms lm013*)

**lemma** *inj-on* (%$a$. ((*fst* $a$, *fst* (*snd* $a$)), *snd* (*snd* $a$))) *UNIV*
**by** (*metis* (*lifting*, *mono-tags*) *Pair-fst-snd-eq Pair-inject injI*)
**lemma** *nn27*: **assumes** *finite* $X$ $x$ > *Max* $X$ **shows** $x$ ∉ $X$ **using** *assms Max.coboundedI* **by** (*metis leD*)

**lemma** *mm86*: **assumes** *finite* $A$ $A$ ≠ {} **shows** *Max* ($f$`$A$) ∈ $f$`$A$
**using** *assms* **by** (*metis Max-in finite-imageI image-is-empty*)

**lemma** *argmax* $f$ $A$ ⊆ $f$ −` {*Max* ($f$ ` $A$)} **by** *force*

**lemma** *mm78*: *argmax* $f$ $A$ = $A$ ∩{ $x$ . $f$ $x$ = *Max* ($f$ ` $A$) } **by** *auto*

**lemma** *nn60*: ($x$ ∈ *argmax* $f$ $X$) = ($x$ ∈ $X$ & $f$ $x$ = *Max* {$f$ *xx*| *xx*. *xx* ∈ $X$})
**using** *argmax.simps image-Collect-mem mem-Collect-eq*
**by** (*metis* (*mono-tags*, *lifting*))

**corollary** *nn59*: **assumes** *finite* $g$ **shows** *setsum* $f$ $g$ = *setsum* $f$ ($g$ *outside* $X$) + (*setsum* $f$ ($g$||$X$))
**unfolding** *Outside-def restrict-def* **using** *assms add.commute inf-commute lll23*
**by** (*metis*)

**lemma** *mm51*: *Range* −` {{}} = {{}} **by** *auto*

**lemma** *mm47*: (∀ *pair* ∈ $a$. *finite* (*snd pair*)) = (∀ $y$ ∈ *Range* $a$. *finite* $y$) **by** *fastforce*

**lemma** *mm38e*: *fst* ` $P$ = *snd* ` ($P$^−*1*) **by** *force*
**lemma** *mm38d*: *fst* $z$ =*snd* (*flip* $z$) & *snd* $z$=*fst* (*flip* $z$) **unfolding** *flip-def* **by** *simp*
**lemma** *flip-flip2*: *flip* ∘ *flip*=*id* **using** *flip-flip* **by** *fastforce*
**lemma** *mm38f*: *fst*=(*snd*∘*flip*) **using** *mm38d* **by** *fastforce*
**lemma** *mm38g*: *snd*=(*fst*∘*flip*) **using** *mm38d* **by** *fastforce*
**lemma** *mm38h*: *inj-on fst* $P$ = *inj-on* (*snd*∘*flip*) $P$ **using** *mm38f* **by** *metis*
**lemma** *mm38c*: *inj-on fst* $P$ = *inj-on snd* ($P$^−*1*)
**using** *mm38h flip-conv* **by** (*metis converse-converse inj-on-imageI mm38g*)

**lemma** *mm39*: **assumes** *runiq* ($a$^−*1*) **shows** *setsum* (*card* ∘ *snd*) $a$ = *setsum card* (*Range* $a$)
**using** *assms mm38c converse-converse lll31 setsum.reindex snd-eq-Range* **by** *metis*

**lemma** *mm29*: **assumes** $X \neq \{\}$ **shows** *finestpart* $X \neq \{\}$ **using** *assms finestpart-def* **by** *blast*

**lemma assumes** *inj-on* $g$ $X$ **shows** *setsum* $f$ $(g`X)$ $=$ *setsum* $(f \circ g)$ $X$ **using** *assms* **by** (*metis setsum.reindex*)

**lemma** *mm60*: **assumes** *runiq* $R$ $z \in R$ **shows** $R,,$(*fst* $z$) $=$ *snd* $z$ **using** *assms* **by** (*metis l31 surjective-pairing*)

**lemma** *mm59*: **assumes** *runiq* $R$ **shows** *setsum* (*toFunction* $R$) (*Domain* $R$) $=$ *setsum snd* $R$ **using** *assms toFunction-def setsum.reindex-cong mm60 lll31* **by** (*metis* (*no-types*) *fst-eq-Domain*)

**corollary** *mm59b*: **assumes** *runiq* $(f||X)$ **shows** *setsum* (*toFunction* $(f||X)$) ($X$ $\cap$ *Domain* $f$) $=$ *setsum snd* $(f||X)$ **using** *assms mm59* **by** (*metis Int-commute ll41*)

**lemma** *lll85b*: *Range* ($R$ *outside* $X$) $=$ $R``$(*Domain* $R$ $-$ $X$) **using** *assms* **by** (*metis Diff-idemp ImageE Range.intros Range-outside-sub-Image-Domain lll01 lll99 order-class.order.antisym subsetI*)

**lemma** $(R||X)$ `` $X$ $=$ $R``X$ **using** *Int-absorb lll02 lll85* **by** *metis*
**lemma** *mm61*: **assumes** $x \in$ *Domain* $(f||X)$ **shows** $(f||X)``\{x\}$ $=$ $f``\{x\}$ **using** *assms* *lll02 lll85 Int-empty-right Int-iff Int-insert-right-if1 ll41* **by** *metis*
**lemma** *mm61b*: **assumes** $x \in X \cap$ *Domain* $f$ *runiq* $(f||X)$ **shows** $(f||X),,x = f,,x$

**using** *assms lll02 lll85 Int-empty-right Int-iff Int-insert-right-if1 eval-rel.simps* **by** *metis*

**lemma** *mm61c*: **assumes** *runiq* $(f||X)$ **shows** *setsum* (*toFunction* $(f||X)$) ($X$ $\cap$ *Domain* $f$) $=$ *setsum* (*toFunction* $f$) ($X$ $\cap$ *Domain* $f$) **using** *assms setsum.cong mm61b toFunction-def* **by** *metis*
**corollary** *mm59c*: **assumes** *runiq* $(f||X)$ **shows** *setsum* (*toFunction* $f$) ($X$ $\cap$ *Domain* $f$) $=$ *setsum snd* $(f||X)$ **using** *assms mm59b mm61c* **by** *fastforce*

**corollary assumes** *runiq* $(f||X)$ **shows** *setsum* (*toFunction* $(f||X)$) ($X$ $\cap$ *Domain* $f$) $=$ *setsum snd* $(f||X)$ **using** *assms mm59 ll41 Int-commute* **by** *metis*
**lemma** *mm26*: *card* (*finestpart* $X$) $=$ *card* $X$ **using** *finestpart-def* **by** (*metis* (*lifting*) *card-image inj-on-inverseI the-elem-eq*)
**corollary** *mm26b*: *finestpart* $\{\}$ $=$ $\{\}$ & *card* $\circ$ *finestpart* $=$ *card* **using** *mm26 finestpart-def* **by** *fastforce*

**lemma** *mm40*: *finite* (*finestpart* $X$) $=$ *finite* $X$ **using** *assms finestpart-def mm26b*

**by** (*metis card-eq-0-iff empty-is-image finite.simps mm26*)

**lemma** *finite* $\circ$ *finestpart* = *finite* **using** *mm40* **by** *fastforce*

**lemma** *lll34*: **assumes** *runiq P* **shows** *card* (*Domain P*) = *card P*
**using** *assms lll33 card-image* **by** (*metis Domain-fst*)

**lemma** *mm43*: **assumes** *runiq f* **shows** *finite* (*Domain f*) = *finite f*
**using** *assms Domain-empty-iff card-eq-0-iff finite.emptyI lll34* **by** *metis*

**lemma** *mm24*: *setsum* ((*curry f*) *x*) *Y* = *setsum f* ($\{x\}$ $\times$ *Y*)
**proof** −
**let** *?f*=% *y*. (*x*, *y*) **let** *?g*=(*curry f*) *x* **let** *?h*=*f*
**have** *inj-on ?f Y* **by** (*metis*(*no-types*) *Pair-inject inj-onI*)
**moreover have** $\{x\}$ $\times$ *Y* = *?f* ‘ *Y* **by** *fast*
**moreover have** $\forall$ *y*. *y* $\in$ *Y* $\longrightarrow$ *?g y* = *?h* (*?f y*) **by** *simp*
**ultimately show** *?thesis* **using** *setsum.reindex-cong* **by** *metis*
**qed**

**lemma** *mm24b*: *setsum* (%*y*. *f* (*x*,*y*)) *Y* = *setsum f* ($\{x\}\times Y$)
**using** *mm24 Sigma-cong curry-def setsum.cong* **by** *fastforce*

**corollary** *mm12*: **assumes** *finite X* **shows** *setsum f X* = *setsum f* (*X*−*Y*) + (*setsum f* (*X* $\cap$ *Y*))
**using** *assms Diff-iff IntD2 Un-Diff-Int finite-Un inf-commute setsum.union-inter-neutral*
**by** *metis*

**lemma** *ll50*: (*P* +∗ *Q*)‘‘(*Domain Q*$\cap$*X*)=*Q*‘‘(*Domain Q*$\cap$*X*)
**unfolding** *paste-def Outside-def Image-def Domain-def* **by** *blast*

**corollary** (*P* +∗ *Q*)‘‘(*X*$\cap$(*Domain Q*))=*Q*‘‘*X* **using** *Int-commute ll50* **by** (*metis lll84*)

**corollary** *mm19*: **assumes** *X* $\cap$ *Domain Q* = {} (**is** *X* $\cap$ *?dq*={}) **shows** (*P* +∗ *Q*) ‘‘ *X* = (*P outside ?dq*)‘‘ *X*
**using** *assms paste-def* **by** *fast*

**lemma** *mm20*: **assumes** *X*$\cap$*Y*={} **shows** (*P outside Y*)‘‘*X*=*P*‘‘*X* **using** *assms Outside-def* **by** *blast*

**corollary** *mm19b*: **assumes** *X*$\cap$*Domain Q*={} **shows** (*P* +∗ *Q*)‘‘*X*=*P*‘‘*X* **using** *assms mm19 mm20* **by** *metis*

**lemma** *mm23*: **assumes** *finite X finite Y card*(*X*$\cap$*Y*)=*card X* **shows** *X*$\subseteq$*Y* **using** *assms*
**by** (*metis Int-lower1 Int-lower2 card-seteq order-refl*)

**lemma** *mm23b*: **assumes** *finite X finite Y card X* =*card Y* **shows** (*card* (*X*$\cap$*Y*)=*card X*)=(*X*=*Y*)
**using** *assms mm23* **by** (*metis card-seteq le-iff-inf order-refl*)

**lemma** *l16*: **assumes** *P xx* **shows**
$\{(x,f\ x)|\ x.\ P\ x\},,xx=f\ xx$
**proof** –
**let** *?F*=$\{(x,f\ x)|\ x.\ P\ x\}$ **let** *?X*=*?F*‘‘$\{xx\}$
**have** *?X*=$\{f\ xx\}$ **using** *Image-def assms* **by** *blast* **thus** *?thesis* **by** *fastforce*
**qed**

**lemma** *ll33*: **assumes** $x \in X$ **shows** *graph X f* $,,x=f\ x$
**unfolding** *graph-def* **using** *assms l16* **by** (*metis* (*mono-tags*) *Gr-def*)

**lemma** *ll28*: *Graph f* $,,x=f\ x$ **using** *UNIV-I ll33 ll36* **by** (*metis*(*no-types*))

**lemma** *toFunction* (*Graph f*)=*f* (**is** *?L*=-)
**proof**–**{fix** *x* **have** *?L x*=*f x* **unfolding** *toFunction-def ll28* **by** *metis***}thus** *?thesis* **by** *blast* **qed**

**lemma** *nn29*: *R outside X* $\subseteq$ *R* **by** (*metis outside-union-restrict subset-Un-eq sup-left-idem*)

**lemma** *nn30a*: *Range*(*f outside X*) $\supseteq$ (*Range f*)–(*f*‘‘*X*) **using** *assms Outside-def* **by** *blast*

**lemma** *lll71b*: **assumes** *runiq P* **shows** $P^{-1}$‘‘((*Range P*)–*X*)$\cap$(($P^{-1}$)‘‘*X*)=$\{\}$
**using** *assms ll71* **by** *blast*

**lemma** *lll78*: **assumes** *runiq* ($P^{-1}$) **shows** *P*‘‘(*Domain P* – *X*)$\cap$(*P*‘‘*X*)=$\{\}$
**using** *assms ll71* **by** *fast*

**lemma** *nn30b*: **assumes** *runiq f runiq* (*f*ˆ–*1*) **shows** *Range*(*f outside X*)$\subseteq$(*Range f*)–(*f*‘‘*X*)
**using** *assms Diff-triv lll78 lll85b Diff-iff ImageE Range-iff subsetI* **by** *metis*
**lemma** *nn30*: **assumes** *runiq f runiq* (*f*ˆ–*1*) **shows** *Range*(*f outside X*)=(*Range f*)–(*f*‘‘*X*)
**using** *assms nn30a nn30b* **by** (*metis order-class.order.antisym*)

**lemma** *lm42*: $(\forall x{\in}X.\ \forall y{\in}Y.\ x{\cap}y=\{\})=(\bigcup X\cap(\bigcup\ Y)=\{\})$ **by** *blast*

**lemma** *Domain* ((*a outside* $(X \cup \{i\})$) $\cup$ ($\{(i, \bigcup$ (*a*‘‘$(X \cup \{i\})$))$\}$ – $\{(i,\{\})\}$) )
$\subseteq$ *Domain a* – $X \cup \{i\}$ **using** *assms Outside-def* **by** *auto*

**lemma** $(R - ((X{\cup}\{i\}){\times}(Range\ R)))$ = (*R outside X*) *outside* $\{i\}$ **using** *Outside-def*

**by** (*metis ll52*)

**lemma** $\{(i,\ x)\} - \{(i,y)\} = \{i\} \times (\{x\}{-}\{y\})$ **by** *fast*

**lemma** *lm44*: $\{x\}{-}\{y\}{=}\{\} = (x{=}y)$ **by** *auto*

**lemma assumes** $R \neq \{\}$ *Domain R* $\cap\ X \neq \{\}$ **shows** *R*‘‘$X \neq \{\}$ **using** *assms*

**by** *blast*

**lemma** *R''{}={}* **by** (*metis Image-empty*)

**lemma** *lm56*: *R* ⊆ (*Domain R*) × (*Range R*) **by** *auto*

**lemma** *lm57*: (*finite* (*Domain Q*) & *finite* (*Range Q*)) = *finite Q* **using** *rev-finite-subset finite-SigmaI lm56 finite-Domain finite-Range* **by** *metis*

**lemma** *lll41*: **assumes** *finite* (⋃ *XX*) **shows** ∀ *X* ∈ *XX*. *finite X* **using** *assms* **by** (*metis Union-upper finite-subset*)

**lemma** *ll57*: **fixes** *a::real* **fixes** *b c* **shows** *a*∗*b* − *a*∗*c*=*a*∗(*b*−*c*) **using** *assms* **by** (*metis real-scaleR-def real-vector.scale-right-diff-distrib*)

**lemma** *lll62*: **fixes** *a::real* **fixes** *b c* **shows** *a*∗*b* − *c*∗*b*=(*a*−*c*)∗*b* **using** *assms ll57* **by** (*metis comm-semiring-1-class.normalizing-semiring-rules*(*7*))

**lemma** *ll71b*: **assumes** *runiq f X* ⊆ (*f*ˆ−*1*)''*Y* **shows** *f''X* ⊆ *Y* **using** *assms ll71* **by** (*metis Image-mono order-refl subset-trans*)

**lemma** *l31b*: **assumes** *y* ∈ *f''*{*x*} *runiq f* **shows** *f*,,*x* = *y* **using** *assms* **by** (*metis Image-singleton-iff l31*)

# 25 Indicator function in set-theoretical form.

**abbreviation** *Outside' X f* == *f outside X*
**abbreviation** *Chi X Y* == (*Y* × {*0::nat*}) +∗ (*X* × {*1*})
**notation** *Chi* (**infix** <|| *80*)
**abbreviation** *chii X Y* == *toFunction* (*X* <|| *Y*)
**notation** *chii* (**infix** <| *80*)
**abbreviation** *chi X* == *indicator X*

**lemma** *mm13*: *runiq* (*X* <|| *Y*) **by** (*metis lll59 runiq-paste2 trivial-singleton*)

**lemma** *mm14c*: **assumes** *x* ∈ *X* **shows** *1* ∈ (*X* <|| *Y*) '' {*x*} **using** *assms toFunction-def*
*paste-def Outside-def runiq-def mm14b* **by** *blast*

**lemma** *mm14d*: **assumes** *x* ∈ *Y*−*X* **shows** *0* ∈ (*X* <|| *Y*) '' {*x*} **using** *assms toFunction-def*
*paste-def Outside-def runiq-def mm14b* **by** *blast*

**lemma** *mm14e*: **assumes** *x* ∈ *X* ∪ *Y* **shows** (*X* <|| *Y*),,*x* = *chi X x* (**is** *?L=?R*)
**using**
*assms mm14b mm14c mm14d l31b* **by** (*metis DiffI Un-iff indicator-simps*(*1*) *indicator-simps*(*2*))

**lemma** *mm14f*: **assumes** *x* ∈ *X* ∪ *Y* **shows** (*X* <| *Y*) *x* = *chi X x* (**is** *?L=?R*)
**using** *assms toFunction-def mm14e* **by** *metis*

**corollary** *mm15b*: *setsum* $(X <| Y)$ $(X \cup Y)$ = *setsum* (*chi* $X$) $(X \cup Y)$
**using** *mm14f setsum.cong* **by** *metis*

**corollary** *lmm02*: **assumes** $!x{:}X.\ f\ x = g\ x$ **shows** *setsum* $f$ $X$ = *setsum* $g$ $X$
**using** *assms*
**by** (*metis* (*poly-guards-query*) *setsum.cong*)
**corollary** *lm02b*: **assumes** $!x{:}X.\ f\ x = g\ x\ Y \subseteq X$ **shows** *setsum* $f$ $Y$ = *setsum* $g$ $Y$ **using** *assms lmm02*
**by** (*metis contra-subsetD*)

**corollary** *mm15*: **assumes** $Z \subseteq X \cup Y$ **shows** *setsum* $(X <| Y)$ $Z$ = *setsum* (*chi* $X$) $Z$
**proof** −
**have** $!x{:}Z.(X{<|}Y)\ x{=}$(*chi* $X$) $x$ **using** *assms mm14f in-mono* **by** *metis* **thus** *?thesis* **using** *lmm02* **by** *blast*
**qed**

**corollary** *mm16*: *setsum* (*chi* $X$) $(Z - X) = 0$ **by** *simp*

**corollary** *mm17*: **assumes** $Z \subseteq X \cup Y$ **shows** *setsum* $(X <| Y)$ $(Z - X) = 0$
**using** *assms mm16 mm15 Diff-iff in-mono subsetI* **by** *metis*

**corollary** *mm18*: **assumes** *finite* $Z$ **shows** *setsum* $(X <| Y)$ $Z$ = *setsum* $(X <| Y)$ $(Z - X)$
+(*setsum* $(X <| Y)$ $(Z \cap X)$) **using** *mm12 assms* **by** *blast*

**corollary** *mm18b*: **assumes** $Z \subseteq X \cup Y$ *finite* $Z$ **shows** *setsum* $(X <| Y)$ $Z$ = *setsum* $(X <| Y)$ $(Z \cap X)$
**using** *assms mm12 mm17 comm-monoid-add-class.add.left-neutral* **by** *metis*

**corollary** *mm21*: **assumes** *finite* $Z$ **shows** *setsum* (*chi* $X$) $Z$ = *card* $(X \cap Z)$
**using** *assms*
*setsum-indicator-eq-card* **by** (*metis Int-commute*)

**corollary** *mm22*: **assumes** $Z \subseteq X \cup Y$ *finite* $Z$ **shows** *setsum* $(X <| Y)$ $Z$ = *card* $(Z \cap X)$
**using** *assms mm21* **by** (*metis mm15 setsum-indicator-eq-card*)

**corollary** *mm28*: **assumes** $Z \subseteq X \cup Y$ *finite* $Z$ **shows** (*setsum* $(X <| Y)$ $X$) − (*setsum* $(X <| Y)$ $Z$) =
*card* $X$ − *card* $(Z \cap X)$ **using** *assms mm22* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

**corollary** *mm28b*: **assumes** $Z \subseteq X \cup Y$ *finite* $Z$ **shows** *int* (*setsum* $(X <| Y)$ $X$) − *int* (*setsum* $(X <| Y)$ $Z$) =
*int* (*card* $X$) − *int* (*card* $(Z \cap X)$) **using** *assms mm22* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

**lemma** *mm28c*: *int* ($n$::*nat*) = *real* $n$ **by** *simp*

**corollary** *mm28d*: **assumes** $Z \subseteq X \cup Y$ *finite Z* **shows**
*real* (*setsum* ($X$ <| $Y$) $X$) − *real* (*setsum* ($X$ <| $Y$) $Z$) = *real* (*card X*) − *real* (*card* ($Z \cap X$))
**using** *assms mm22* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

**lemma** *mm84c*: **assumes** $\exists\ n \in \{0..<size\ l\}.\ P\ (l!n)$ **shows** $[n.\ n \leftarrow [0..<size\ l],\ P\ (l!n)] \neq []$
**using** *assms* **by** *auto*
**lemma** *mm84d*: **assumes** $ll \in set\ (l::'a\ list)$ **shows** $\exists\ n \in (nth\ l)\ -`\ (set\ l).\ ll = l!n$
**using** *assms*(1) **by** (*metis in-set-conv-nth vimageI2*)
**lemma** *mm84e*: **assumes** $ll \in set\ (l::'a\ list)$ **shows** $\exists\ n.\ ll = l!n\ \&\ n < size\ l\ \&\ n >= 0$
**using** *assms in-set-conv-nth* **by** (*metis le0*)

**lemma** $(nth\ l) -`\ (set\ l) \supseteq \{0..<size\ l\}$ **using** *assms* **by** *fastforce*
**lemma** *mm84f*: **assumes** $P -`\ \{True\} \cap set\ l \neq \{\}$ **shows** $\exists\ n \in \{0..<size\ l\}.$ $P\ (l!n)$
**using** *assms mm84e* **by** *fastforce*

**lemma** *mm84g*: **assumes** $P -`\ \{True\} \cap set\ l \neq \{\}$ **shows** $[n.\ n \leftarrow [0..<size\ l],$ $P\ (l!n)] \neq []$
**using** *assms filterpositions2-def mm84f mm84c* **by** *metis*

**lemma** *nn06*: $(nth\ l)\ `\ set\ ([n.\ n \leftarrow [0..<size\ l], (\%x.\ x{\in}X)\ (l!n)]) \subseteq X{\cap}set\ l$ **by** *force*
**corollary** *nn06b*: $(nth\ l)`\ set\ (filterpositions2\ (\%x.(x{\in}X))\ l) \subseteq X \cap\ set\ l$
**unfolding** *filterpositions2-def* **using** *nn06* **by** *fast*

**lemma** $(n{\in}\{0..<N\}) = ((n::nat) < N)$ **using** *atLeast0LessThan lessThan-iff* **by** *metis*
**lemma** *nn01*: **assumes** $X \subseteq \{0..<size\ list\}$ **shows** $(nth\ list)`X \subseteq set\ list$
**using** *assms atLeastLessThan-def atLeast0LessThan lessThan-iff* **by** *auto*
**lemma** *mm99*: $set\ ([n.\ n \leftarrow [0..<size\ l],\ P\ (l!n)]) \subseteq \{0..<size\ l\}$ **by** *force*
**lemma** *mm99b*: $set\ (filterpositions2\ pre\ list) \subseteq \{0..<size\ list\}$ **using** *filterpositions2-def mm99* **by** *metis*

**lemma** *mm55n*: **assumes** $X \subseteq Y$ **shows** $finestpart\ X \subseteq finestpart\ Y$ **using** *assms finestpart-def* **by** (*metis image-mono*)
**corollary** *mm55o*: **assumes** $x \in X$ **shows** $finestpart\ x \subseteq finestpart\ (\bigcup\ X)$ **using** *assms*
*mm55n* **by** (*metis Union-upper*)
**lemma** *mm55p*: $\bigcup\ (finestpart\ `\ XX) \subseteq finestpart\ (\bigcup\ XX)$ **using** *assms finestpart-def*

*mm55o* **by** *force*
**lemma** *mm55q*: $\bigcup\ (finestpart\ `\ XX) \supseteq finestpart\ (\bigcup\ XX)$ (**is** $?L \supseteq ?R$)
**unfolding** *finestpart-def* **using** *finestpart-def* **by** *auto*

**corollary** *mm55r*: $\bigcup\ (finestpart\ `\ XX) = finestpart\ (\bigcup\ XX)$ **using** *mm55p mm55q* **by** *fast*

**lemma** *mm55h*: **assumes** *runiq a* **shows** $\{(x,\ \{y\})|\ x\ y.\ y \in \bigcup\ (a``\{x\})\ \&\ x \in Domain\ a\} =$
$\{(x,\ \{y\})|\ x\ y.\ y \in a,,x\ \&\ x \in Domain\ a\}$ **using** *assms Image-runiq-eq-eval*
**by** (*metis* (*lifting*, *no-types*) *cSup-singleton*)

## 25.1 Computing all the permutations of a list

**abbreviation** *rotateLeft* == *rotate*
**abbreviation** *rotateRight n l* == *rotateLeft* (*size l* − (*n mod* (*size l*))) *l*
**abbreviation** *insertAt x l n* == *rotateRight n* (*x*#(*rotateLeft n l*))

**fun** *perm2* **where**

*perm2* [] = (%*n*. []) |
*perm2* (*x*#*l*) = (%*n*. *insertAt x* ((*perm2 l*) (*n div* (*1*+*size l*))) (*n mod* (*1*+*size l*)))

**abbreviation** *takeAll pre list* == *map* (*nth list*) (*filterpositions2 pre list*)

**lemma** *mm83*: **assumes** $l \neq []$ **shows** *perm2 l n* $\neq$ []
**using** *assms perm2-def perm2.simps*(*2*) *rotate-is-Nil-conv* **by** (*metis neq-Nil-conv*)
**lemma** *mm98*: *set* (*takeAll pre list*) = ((*nth list*) ‘ *set* (*filterpositions2 pre list*))
**by** *simp*

**corollary** *nn06c*: *set* (*takeAll* (%*x*.($x \in X$)) *l*) $\subseteq$ $X \cap$*set l* **using** *nn06b mm98* **by** *metis*

**corollary** *nn02*: *set* (*takeAll pre list*) $\subseteq$ *set list* **using** *mm99b mm98 nn01* **by** *metis*
**lemma** *nn03*: *set* (*insertAt x l n*) = $\{x\} \cup$ *set l* **by** *simp*
**lemma** *nn04a*: $\forall$ *n*. *set* (*perm2* [] *n*) = *set* [] **by** *simp*
**lemma** *nn04b*: **assumes** $\forall$ *n*. (*set* (*perm2 l n*) = *set l*) **shows** *set* (*perm2* (*x*#*l*) *n*) = $\{x\} \cup$ *set l*
**using** *assms perm2-def nn03* **by** *force*
**corollary** *nn04*: $\forall$ *n*. *set* (*perm2 l n*) = *set l*

**proof** (*induct l*)
**let** *?P*=%*l*. ($\forall$ *n*. *set* (*perm2 l n*)=*set l*)
**show** *?P* [] **using** *nn04a* **by** *force* **next let** *?P*=%*l*. ($\forall$ *n*. *set* (*perm2 l n*)=*set l*)
**fix** *x* **fix** *l* **assume** *?P l* **then show** *?P* (*x*#*l*) **by** *force*
**qed**

**corollary** *nn05a*: *set* (*perm2* (*takeAll* (%*x*.($x \in X$)) *l*) *n*) $\subseteq X \cap$ *set l* **using** *nn06c nn04* **by** *metis*

## 26 A more computable version of *toFunction*.

**abbreviation** *toFunctionWithFallback R fallback* == (% *x*. *if* ($R$“$\{x\}$=$\{R$,,$x\}$) *then* (*R*,,*x*) *else fallback*)
**notation** *toFunctionWithFallback* (**infix** *Else 75*)
**abbreviation** *setsum′ R X* == *setsum* (*R Else 0*) *X*
**abbreviation** *setsum′′ R X* == *setsum* (*toFunction R*) ($X \cap$ *Domain R*)
**abbreviation** *setsum′′′ R X* == *setsum′ R* ($X\cap$*Domain R*)
**abbreviation** *setsum′′′′ R X* == *setsum* (%*x*. *setsum id* ($R$“$\{x\}$)) *X*

**lemma** *nn47*: **assumes** *runiq f x* $\in$ *Domain f* **shows** (*f Else 0*) *x* = (*toFunction f*) *x* **using** *assms*
**by** (*metis Image-runiq-eq-eval toFunction-def*)

**lemma** *nn48b*: **assumes** *runiq f* **shows** *setsum* (*f Else 0*) ($X\cap$(*Domain f*)) = *setsum* (*toFunction f*) ($X\cap$(*Domain f*))
**using** *assms setsum.cong nn47* **by** *fastforce*

**lemma** *nn51*: **assumes** $Y \subseteq f-`\{0\}$ **shows** *setsum f* $Y=0$ **using** *assms*
**by** (*metis set-rev-mp setsum.neutral vimage-singleton-eq*)

**lemma** *nn49*: **assumes** $Y \subseteq f-`\{0\}$ *finite X* **shows** *setsum f X* = *setsum f* $(X-Y)$
**using** *assms Int-lower2 comm-monoid-add-class.add.right-neutral inf.boundedE inf.orderE mm12 nn51*
**by** (*metis*(*no-types*))

**lemma** *nn50*: $-(Domain\ f) \subseteq (f\ Else\ 0)-`\{0\}$ **by** *fastforce*

**corollary** *nn52*: **assumes** *finite X* **shows** *setsum* (*f Else 0*) *X*=*setsum* (*f Else 0*) $(X \cap Domain\ f)$
**proof**− **have** $X \cap Domain\ f = X-(-Domain\ f)$ **by** *simp* **thus** *?thesis* **using** *assms nn50 nn49* **by** *fastforce* **qed**
**corollary** *nn52b*: **assumes** *finite X* **shows** *setsum* (*f Else 0*) $(X \cap Domain\ f)$=*setsum* (*f Else 0*) *X*
(**is** *?L*=*?R*) **proof** − **have** *?R*=*?L* **using** *assms* **by** (*rule nn52*) **thus** *?thesis* **by** *simp* **qed**

**corollary** *nn48c*: **assumes** *finite X runiq f* **shows**
*setsum* (*f Else 0*) *X* = *setsum* (*toFunction f*) $(X \cap Domain\ f)$ (**is** *?L*=*?R*)
**proof** −
**have** *?R* = *setsum* (*f Else 0*) $(X \cap Domain\ f)$ **using** *assms*(*2*) *nn48b* **by** *fastforce*
**moreover have** ... = *?L* **using** *assms*(*1*) **by** (*rule nn52b*) **ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *nn53*: *setsum* (*f Else 0*) *X* = *setsum′ f X* **by** *fast*

**corollary** *nn48d*: **assumes** *finite X runiq f* **shows** *setsum* (*toFunction f*) $(X \cap Domain\ f)$ = *setsum′ f X*
**using** *assms nn53 nn48c* **by** *fastforce*
**lemma** *argmax* (*setsum′ b*) = (*argmax* ∘ *setsum′*) *b* **by** *simp*

**lemma** *lm015*: **assumes** *runiq R runiq* $(R\hat{}-1)$ **shows** $R``A \cap (R``B) = R``(A \cap B)$

**using** *assms lll33 converse-Image* **by** *force*

**lemma** *lm40*: **assumes** *runiq* $(R\hat{}-1)$ *runiq R* $X1 \cap X2 = \{\}$ **shows** $R``X1 \cap (R``X2) = \{\}$
**using** *assms* **by** (*metis disj-Domain-imp-disj-Image inf-assoc inf-bot-right*)

**lemma** *ll70*: **assumes** *runiq f trivial Y* **shows** *trivial* $(f `` (f\hat{}-1 `` Y))$
**using** *assms* **by** (*metis ll71 trivial-subset*)

**lemma** *lm020*: **assumes** *trivial X* **shows** *card* $(Pow\ X) \in \{1,2\}$ **using** *lm007 card-Pow*
*Pow-empty assms lm54 nn56 power-one-right the-elem-eq* **by** (*metis insert-iff*)

**lemma** *lm017*: **assumes** *card* (*Pow A*)=*1* **shows** *A*={} **using** *assms*
**by** (*metis Pow-bottom Pow-top nn56 singletonD*)

**lemma** *lm022*: (¬ (*finite A*)) = (*card* (*Pow A*) = *0*) **by** *auto*

**corollary** *lm022b*: (*finite A*) = (*card* (*Pow A*) ≠ *0*) **using** *lm022* **by** *metis*

**lemma** *lm016*: **assumes** *card* (*Pow A*) ≠ *0* **shows** *card A*=*Discrete.log* (*card* (*Pow A*)) **using** *assms*
*log-exp card-Pow* **by** (*metis card-infinite finite-Pow-iff*)

**lemma** *lm018*: **assumes** *card* (*Pow A*)=*2* **shows** *card A*=*1* **using** *assms lm016*
**by** (*metis*(*no-types*) *comm-semiring-1-class.normalizing-semiring-rules*(*33*) *log-exp zero-neq-numeral*)

**lemma** *lm019*: **assumes** *card* (*Pow X*)=*1* ∨ *card* (*Pow X*)=*2* **shows** *trivial X*
**using** *assms lm007 lm017 lm018 nn56* **by** *metis*

**lemma** *lm021*: *trivial A* = (*card* (*Pow A*) ∈ {*1*,*2*}) **using** *lm019 lm020* **by** *blast*

**lemma assumes** *R*⊆*f runiq f Domain f* = *Domain R* **shows** *runiq R*
**using** *assms* **by** (*metis subrel-runiq*)

**lemma** *ll81a*: **assumes** *f* ⊆ *g runiq g Domain f* = *Domain g* **shows** *g* ⊆ *f*
**using** *assms Domain-iff contra-subsetD runiq-wrt-ex1 subrelI*
**by** (*metis* (*full-types*,*hide-lams*))

**lemma** *ll81*: **assumes** *R*⊆*f runiq f Domain f* ⊆ *Domain R* **shows** *f*=*R*
**using** *assms ll81a* **by** (*metis Domain-mono dual-order.antisym*)

**lemma** *lm06*: *graph X f* = *Graph f* || *X* **using** *inf-top.left-neutral ll36 ll37 ll41 ll81 lm04 restriction-is-subrel subrel-runiq subset-iff* **by** (*metis* (*erased*, *lifting*))
**lemma** *lm05*: *graph* (*X* ∩ *Y*) *f* = *graph X f* || *Y* **using** *lll02 lm06* **by** *metis*
**lemma** *mm65*:{(*x*, *f x*)| *x*. *x* ∈ *X2*} || *X1* = {(*x*, *f x*)| *x*. *x* ∈ *X2* ∩ *X1*} **using** *graph-def lm05* **by** *metis*

**lemma** *mm10*: **assumes** *runiq f X* ⊆ *Domain f* **shows** *graph X* (*toFunction f*) = (*f*||*X*)
**proof** −
  **have** ⋀*v w*. (*v*::′*a set*) ⊆ *w* ⟶ *w* ∩ *v* = *v* **by** (*simp add*: *Int-commute inf.absorb1*)
  **thus** *graph X* (*toFunction f*) = *f* || *X* **by** (*metis assms*(*1*) *assms*(*2*) *lll02 lm024 lm06*)
**qed**

**lemma** *l4*: (*Graph f*) ‘‘ *X* = *f* ‘ *X* **unfolding** *Graph-def image-def* **by** *auto*

**lemma** *lm025*: **assumes** *X* ⊆ *Domain f runiq f* **shows** *f*‘‘*X* = (*eval-rel f*)‘*X*

**using** *assms l4* **by** (*metis lll85 lm06 mm10 toFunction-def*)

**end**

# 27 Definitions about those Combinatorial Auctions which are strict (i.e., which assign all the available goods)

**theory** *StrictCombinatorialAuction*
**imports** *Complex-Main*
*Partitions*
*MiscTools*

**begin**

# 28 Types

**type-synonym** *index* = *nat*
**type-synonym** *participant* = *index*
**type-synonym** *good* = *nat*
**type-synonym** *goods* = *nat set*
**type-synonym** *price* = *real*

**type-synonym** *bids3* = ((*participant* × *goods*) × *price*) *set*
**type-synonym** *bids* = *participant* ⇒ *goods* ⇒ *price*
**type-synonym** *allocation-rel* = (*goods* × *participant*) *set*
**type-synonym** *allocation* = (*participant* × *goods*) *set*

**type-synonym** *payments* = *participant* ⇒ *price*
**type-synonym** *altbids* = (*participant* × *goods*) ⇒ *price*
**type-synonym** *bidvector*=*altbids*
**abbreviation** *altbids* (*b*::*bids*) == *split b*

**abbreviation** *proceeds* (*b*::*altbids*) (*allo*::*allocation*) == *setsum b allo*

**abbreviation** *participants* **where** *participants* (*a*::*allocation*) == *Domain a*
**abbreviation** *goods*::*allocation* => *goods* **where** *goods* (*allo*::*allocation*) == ⋃ (*Range allo*)

# 29 Allocations

**fun** *possible-allocations-rel*
**where** *possible-allocations-rel G N* = *Union* { *injections Y N* | *Y* . *Y* ∈ *all-partitions G* }

**abbreviation** *is-partition-of'* $P$ $A$ == ($\bigcup$ $P$ = $A$ $\wedge$ *is-partition* $P$)

**abbreviation** *all-partitions'* $A$ == {$P$ . *is-partition-of'* $P$ $A$}

**abbreviation** *injections'* $X$ $Y$ == {$R$ . *Domain* $R$ = $X$ $\wedge$ *Range* $R \subseteq Y$ $\wedge$ *runiq* $R$ $\wedge$ *runiq* ($R^{-1}$)}

**abbreviation** *possible-allocations-rel'* $G$ $N$ == *Union*{*injections'* $Y$ $N$ | $Y$ . $Y \in$ *all-partitions'* $G$}

**abbreviation** *possibleAllocationsRel* **where**
*possibleAllocationsRel* $N$ $G$ == *converse* ‘ (*possible-allocations-rel* $G$ $N$)

**notepad**
**begin**
  **fix** *Rs*::('$a \times$ '$b$) *set set*
  **fix** *Sss*::'$a$ *set set*
  **fix** $P$::'$a$ *set* $\Rightarrow$ ('$a \times$ '$b$) *set set*

  **have** { $R$ . $\exists$ $Y \in$ *Sss* . $R \in P$ $Y$ } = $\bigcup$ { $P$ $Y$ | $Y$ . $Y \in$ *Sss* }
  **using** *Collect-cong Union-eq mem-Collect-eq* **by** *blast*
**end**

algorithmic version of *possible-allocations-rel*

**fun** *possible-allocations-alg* :: *goods* $\Rightarrow$ *participant set* $\Rightarrow$ *allocation-rel list*
**where** *possible-allocations-alg* $G$ $N$ =
*concat* [ *injections-alg* $Y$ $N$ . $Y \leftarrow$ *all-partitions-alg* $G$ ]
**abbreviation** *possibleAllocationsAlg* $N$ $G$ ==
(*map converse* (*possible-allocations-alg* $G$ $N$))
**abbreviation** *possibleAllocationsAlg2* $N$ $G$ ==
*converse* ‘ ($\bigcup$ *set* [*set* (*injections-alg* $l$ $N$) . $l \leftarrow$ *all-partitions-list* $G$])
**abbreviation** *possibleAllocationsAlg3* $N$ $G$ ==
*map converse* (*concat* [(*injections-alg* $l$ $N$) . $l \leftarrow$ *all-partitions-list* $G$])
**lemma** *lm01*: *set* (*possibleAllocationsAlg3* $N$ $G$) = *possibleAllocationsAlg2* $N$ $G$
**using** *assms* **by** *auto*

# 30 VCG mechanism

**abbreviation** *winningAllocationsRel* $N$ $G$ $b$ ==
*argmax* (*setsum* $b$) (*possibleAllocationsRel* $N$ $G$)

**abbreviation** *winningAllocationRel* $N$ $G$ $t$ $b$ == $t$ (*winningAllocationsRel* $N$ $G$ $b$)

**abbreviation** *winningAllocationsAlg* $N$ $G$ $b$ == *argmaxList* (*proceeds* $b$) (*possibleAllocationsAlg3* $N$ $G$)

**definition** *winningAllocationAlg* $N$ $G$ $t$ $b$ == $t$ (*winningAllocationsAlg* $N$ $G$ $b$)

payments

the maximum sum of bids of all bidders except bidder $n$'s bid, computed over all possible allocations of all goods, i.e. the value reportedly generated by value maximization problem when solved without n's bids

**abbreviation** *alpha N G b n == Max ((setsum b)'(possibleAllocationsRel (N−{n}) G))*

**abbreviation** *remainingValueRel N G t b n == setsum b (winningAllocationRel N G t b −− n)*

**abbreviation** *paymentsRel N G t == alpha N G − remainingValueRel N G t*

**abbreviation** *remainingValueAlg N G t b n == proceeds b (winningAllocationAlg N G t b −− n)*

**abbreviation** *alphaAlg N G b n == Max ((proceeds b)'(set (possibleAllocationsAlg3 (N−{n}) (G::- list))))*
**definition** *paymentsAlg N G t == alphaAlg N G − remainingValueAlg N G t*

# 31 Uniform tie breaking: definitions

To each allocation we associate the bid in which each participant bids for a set of goods the cardinality of the intersection of that set with the set she gets in the given allocation. By construction, the revenue of an auction run using this bid is maximal on the given allocation, and this maximal is unique. We can then use the bid constructed this way *tiebids'* to break ties by running an auction having the same form as a normal auction (that is why we use the adjective "uniform"), only with this special bid vector.

**abbreviation** *omega pair == {fst pair} × (finestpart (snd pair))*
**abbreviation** *pseudoAllocation allocation == ⋃ (omega ' allocation)*

**abbreviation** *bidMaximizedBy allocation N G ==*
*(∗ (N × finestpart G) × {0::price} +∗ ((pseudoAllocation allocation) × {1}) ∗) pseudoAllocation allocation <|| ((N × (finestpart G)))*
**abbreviation** *maxbid' a N G == toFunction (bidMaximizedBy a N G)*
**abbreviation** *partialCompletionOf bids pair == (pair, setsum (%g. bids (fst pair, g)) (finestpart (snd pair)))*
**abbreviation** *test bids pair == setsum (%g. bids (fst pair, g)) (finestpart (snd pair))*
**abbreviation** *LinearCompletion bids N G == (partialCompletionOf bids) ' (N × (Pow G − {{}}))*
**abbreviation** *linearCompletion' bids N G == toFunction (LinearCompletion bids N G)*
**abbreviation** *tiebids' a N G == linearCompletion' (maxbid' a N G) N G*
**abbreviation** *Tiebids a N G == LinearCompletion (real∘maxbid' a N G) N G*
**abbreviation** *chosenAllocation' N G bids random ==*

*hd*(*perm2* (*takeAll* (%*x*. *x*∈(*winningAllocationsRel N* (*set G*) *bids*)) (*possibleAllocationsAlg3 N G*)) *random*)
**abbreviation** *resolvingBid′ N G bids random* == *tiebids′* (*chosenAllocation′ N G bids random*) *N* (*set G*)

**end**

# 32 Sets of injections, partitions, allocations expressed as suitable subsets of the corresponding universes

**theory** *Universes*

**imports**
*~~/src/HOL/Library/Code-Target-Nat*
*StrictCombinatorialAuction*
*MiscTools*
*~~/src/HOL/Library/Indicator-Function*

**begin**

# 33 Preliminary lemmas

**lemma** *lm63*: **assumes** *Y* ∈ *set* (*all-partitions-alg X*) **shows** *distinct Y*
**using** *assms distinct-sorted-list-of-set all-partitions-alg-def all-partitions-paper-equiv-alg′*
**by** *metis*

**lemma** *lm65*: **assumes** *finite G* **shows** *all-partitions G* = *set* ‘ (*set* (*all-partitions-alg G*))
**using** *assms lm64 all-partitions-alg-def all-partitions-paper-equiv-alg distinct-sorted-list-of-set image-set* **by** *metis*

**lemma assumes** *Y* ∈ *set* (*all-partitions-alg G*) *card N* > *0 finite N finite G*
**shows** *injections* (*set Y*) *N* = *set* (*injections-alg Y N*)
**using** *assms injections-equiv lm63* **by** *metis*

**lemma** *lm67*: **assumes** *l* ∈ *set* (*all-partitions-list G*) *distinct G* **shows** *distinct l*
**using** *assms all-partitions-list-def* **by** (*metis all-partitions-paper-equiv-alg′*)
**lemma** *lm68*: **assumes** *card N* > *0 distinct G* **shows**
∀ *l* ∈ *set* (*all-partitions-list G*). *set* (*injections-alg l N*) = *injections* (*set l*) *N*
**using** *lm67 injections-equiv assms* **by** *blast*

**lemma** *lm69*: **assumes** *card N* > *0 distinct G*
**shows** {*injections P N*| *P*. *P* ∈ *all-partitions* (*set G*)} =
*set* [*set* (*injections-alg l N*) . *l* ← *all-partitions-list G*] **using** *assms lm66 lm68 lm66b*
**proof** −

**let** *?g1=all-partitions-list* **let** *?f2=injections* **let** *?g2=injections-alg*
**have** $\forall$ *l* $\in$ *set* (*?g1 G*). *set* (*?g2 l N*) = *?f2* (*set l*) *N* **using** *assms lm68* **by** *blast*
**then have** *set* [*set* (*?g2 l N*). *l* <− *?g1 G*] = {*?f2 P N*| *P*. *P* $\in$ *set* (*map set* (*?g1 G*))} **apply** (*rule lm66*) **done**
**moreover have** ... = {*?f2 P N*| *P*. *P* $\in$ *all-partitions* (*set G*)} **using** *all-partitions-paper-equiv-alg assms* **by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *lm70*: **assumes** *card N* > *0 distinct G* **shows**
*Union* {*injections P N*| *P*. *P* $\in$ *all-partitions* (*set G*)} =
*Union* (*set* [*set* (*injections-alg l N*) . *l* $\leftarrow$ *all-partitions-list G*]) (**is** *Union ?L* = *Union ?R*)
**proof** − **have** *?L* = *?R* **using** *assms* **by** (*rule lm69*) **thus** *?thesis* **by** *presburger*
**qed**

**corollary** *lm70b*: **assumes** *card N* > *0 distinct G* **shows**
*possibleAllocationsRel N* (*set G*) = *possibleAllocationsAlg2 N G* (**is** *?L* = *?R*) **using** *assms lm70*
*possible-allocations-rel-def*
**proof** −
**let** *?LL*=$\bigcup$ {*injections P N*| *P*. *P* $\in$ *all-partitions* (*set G*)}
**let** *?RR*=$\bigcup$ (*set* [*set* (*injections-alg l N*) . *l* $\leftarrow$ *all-partitions-list G*])
**have** *?LL* = *?RR* **using** *assms* **apply** (*rule lm70*) **done**
**then have** *converse* ‘ *?LL* = *converse* ‘ *?RR* **by** *presburger*
**thus** *?thesis* **using** *possible-allocations-rel-def* **by** *force*
**qed**

# 34 Definitions of various subsets of *UNIV*.

**abbreviation** *isChoice R* == $\forall$ *x*. *R*‘‘{*x*} $\subseteq$ *x*
**abbreviation** *dualOutside R Y* == *R* − (*Domain R* $\times$ *Y*)
**notation** *dualOutside* (**infix** |− *75*)
**notation** *Outside* (**infix** −| *75*)

**abbreviation** *partitionsUniverse* == {*X*. *is-partition X*}
**lemma** *partitionsUniverse* $\subseteq$ *Pow UNIV* **by** *simp*
**abbreviation** *partitionValuedUniverse* == $\bigcup$ *P* $\in$ *partitionsUniverse*. *Pow* (*UNIV* $\times$ *P*)
**lemma** *partitionValuedUniverse* $\subseteq$ *Pow* (*UNIV* $\times$ (*Pow UNIV*)) **by** *simp*
**abbreviation** *injectionsUniverse* == {*R*. (*runiq R*) & (*runiq* (*R*ˆ−*1*))}
**abbreviation** *allocationsUniverse* == *injectionsUniverse* $\cap$ *partitionValuedUniverse*
**abbreviation** *totalRels X Y* == {*R*. *Domain R* = *X* & *Range R* $\subseteq$ *Y*}
**abbreviation** *strictCovers G* == *Union* −‘ {*G*}

## 35 Results about the sets defined in the previous section

**lemma** *lm01a*: *partitionsUniverse* $\subseteq$ $\{P-\{\{\}\}|\ P.\ \bigcap P \in \{\bigcup P,\{\}\}\}$ **unfolding** *is-partition-def* **by** *auto*
**lemma** *lm04*: **assumes** $!x1 : X.\ (x1 \neq \{\}\ \&\ (!\ x2 : X-\{x1\}.\ x1 \cap x2=\{\}))$ **shows** *is-partition X*
**unfolding** *is-partition-def* **using** *assms* **by** *fast*
**lemma** *lm72*: **assumes** $\forall x \in X.\ t\ x \in x$ **shows** *isChoice* (*graph X t*) **using** *assms* **by** (*metis Image-within-domain′ empty-subsetI insert-subset ll33 ll37 runiq-wrt-eval-rel subset-trans*)

**lemma** $R\ |-\ Y = (R\,\hat{}-1\ -|\ Y)\,\hat{}-1$ **using** *Outside-def* **by** *auto*
**lemma** *lm24*: *injections′ X Y* = *injections X Y* **using** *injections-def* **by** *metis*
**lemma** *lm25*: *injections′ X Y* $\subseteq$ *injectionsUniverse* **by** *fast*
**lemma** *lm25b*: *injections X Y* $\subseteq$ *injectionsUniverse* **using** *injections-def* **by** *blast*
**lemma** *lm26*: *injections′ X Y* = *totalRels X Y* $\cap$ *injectionsUniverse* **by** *fastforce*

**lemma** *lm47*: **assumes** $a \in$ *possibleAllocationsRel N G* **shows**
$a\,\hat{}-1 \in$ *injections* (*Range a*) *N* & *Range a partitions G* & *Domain a* $\subseteq$ *N*
**unfolding** *injections-def* **using** *assms all-partitions-def injections-def* **by** *fastforce*

**lemma** *lll80*: **assumes** *is-partition XX YY* $\subseteq$ *XX* **shows** (*XX* − *YY*) *partitions* ($\bigcup$ *XX* − $\bigcup$ *YY*)
**using** *is-partition-of-def is-partition-def assms*
**proof** −
  **let** *?xx=XX* − *YY* **let** *?X*=$\bigcup$ *XX* **let** *?Y*=$\bigcup$ *YY*
  **let** *?x=?X* − *?Y*
  **have** $\forall\ y \in YY.\ \forall\ x \in ?xx.\ y \cap x=\{\}$ **using** *assms is-partition-def* **by** (*metis Diff-iff set-rev-mp*)
  **then have** $\bigcup$ *?xx* $\subseteq$ *?x* **using** *assms* **by** *blast*
  **then have** $\bigcup$ *?xx* = *?x* **by** *blast*
  **moreover have** *is-partition ?xx* **using** *subset-is-partition* **by** (*metis Diff-subset assms*(*1*))
  **ultimately**
  **show** *?thesis* **using** *is-partition-of-def* **by** *blast*
**qed**

**lemma** *lll81a*: **assumes** $a \in$ *possible-allocations-rel G N* **shows**
*runiq a* & *runiq* ($a^{-1}$) & (*Domain a*) *partitions G* & *Range a* $\subseteq$ *N*
**proof** −
  **obtain** *Y* **where**
  *0*: $a \in$ *injections Y N* & $Y \in$ *all-partitions G* **using** *assms possible-allocations-rel-def* **by** *auto*
  **show** *?thesis* **using** *0 injections-def all-partitions-def mem-Collect-eq* **by** *fastforce*
**qed**

**lemma** *lll81b*: **assumes** *runiq a runiq* ($a^{-1}$) (*Domain a*) *partitions G Range a* $\subseteq$ *N*

**shows** *a* ∈ *possible-allocations-rel G N*
**proof** −
**have** *a* ∈ *injections* (*Domain a*) *N* **unfolding** *injections-def* **using** *assms*(*1*) *assms*(*2*) *assms*(*4*) **by** *blast*
**moreover have** *Domain a* ∈ *all-partitions G* **using** *assms*(*3*) *all-partitions-def* **by** *fast*
**ultimately show** *?thesis* **using** *assms*(*1*) *possible-allocations-rel-def* **by** *auto*
**qed**

**lemma** *lll81*: *a* ∈ *possible-allocations-rel G N* ⟷
*runiq a* & *runiq* ($a^{-1}$) & (*Domain a*) *partitions G* & *Range a* ⊆ *N*
**using** *lll81a lll81b* **by** *blast*

**corollary assumes** *runiq* (*P*ˆ−*1*) **shows** *Range* (*P outside X*) ∩ *Range* (*P* || *X*)={}
**using** *assms lll78* **by** (*metis lll01 lll85*)

**lemma** *lm10*: *possible-allocations-rel′ G N* ⊆ *injectionsUniverse*
**using** *assms* **by** *force*

**lemma** *lm09*: *possible-allocations-rel G N* ⊆ {*a. Range a* ⊆ *N* & *Domain a* ∈ *all-partitions G*}
**using** *assms possible-allocations-rel-def injections-def* **by** *fastforce*

**lemma** *lm11*: *injections X Y* = *injections′ X Y* **using** *injections-def*
**by** *metis*

**lemma** *lm12*: *all-partitions X* = *all-partitions′ X* **using** *all-partitions-def is-partition-of-def*

**by** *auto*

**lemma** *lm13*: *possible-allocations-rel′ A B* = *possible-allocations-rel A B* (**is** *?A*=*?B*)
**proof** −
**have** *?B*=⋃ { *injections Y B* | *Y* . *Y* ∈ *all-partitions A* }
**using** *possible-allocations-rel-def* **by** *auto*
**moreover have** ... = *?A* **using** *injections-def lm12* **by** *metis*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *lm14*: *possible-allocations-rel G N* ⊆ *injectionsUniverse* ∩ {*a. Range a* ⊆ *N* & *Domain a* ∈ *all-partitions G*}
**using** *assms lm09 lm10 possible-allocations-rel-def injections-def* **by** *fastforce*

**lemma** *lm15*: *possible-allocations-rel G N* ⊇ *injectionsUniverse* ∩ {*a. Domain a* ∈ *all-partitions G* & *Range a* ⊆ *N*}
**using** *possible-allocations-rel-def injections-def* **by** *auto*

**lemma** *lm16*: *converse* ‘ *injectionsUniverse* = *injectionsUniverse* **by** *auto*

**lemma** *lm17*: *possible-allocations-rel G N* = *injectionsUniverse* ∩ {*a*. *Domain a* ∈ *all-partitions G* & *Range a* ⊆ *N*}
**using** *lm14 lm15* **by** *blast*

**lemma** *lm18*: *converse*‘(*A* ∩ *B*)=*converse*‘*A* ∩ (*converse*‘*B*) **by** *force*

**lemma** *lm19*: *possibleAllocationsRel N G* = *injectionsUniverse* ∩ {*a*. *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions G*}
**proof** −
**let** *?A*=*possible-allocations-rel G N* **let** *?c*=*converse* **let** *?I*=*injectionsUniverse*
**let** *?P*=*all-partitions G* **let** *?d*=*Domain* **let** *?r*=*Range*
**have** *?c*‘*?A* = (*?c*‘*?I*) ∩ *?c*‘ ({*a*. *?r a* ⊆ *N* & *?d a* ∈ *?P*}) **using** *lm17* **by** *fastforce*
**moreover have** ... = (*?c*‘*?I*) ∩ {*aa*. *?d aa* ⊆ *N* & *?r aa* ∈ *?P*} **by** *fastforce*
**moreover have** ... = *?I* ∩ {*aa*. *?d aa* ⊆ *N* & *?r aa* ∈ *?P*} **using** *lm16* **by** *metis*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**corollary** *lm19c*: *a* ∈ *possibleAllocationsRel N G* =
(*a* ∈ *injectionsUniverse* & *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions G*)
**using** *lm19 Int-Collect Int-iff* **by** (*metis* (*lifting*))

**corollary** *lm19d*: **assumes** *a* ∈ *possibleAllocationsRel N1 G* **shows**
*a* ∈ *possibleAllocationsRel* (*N1* ∪ *N2*) *G*
**proof** −
**have** *Domain a* ⊆ *N1* ∪ *N2* **using** *assms*(*1*) *lm19c* **by** (*metis le-supI1*)
**moreover have** *a* ∈ *injectionsUniverse* & *Range a* ∈ *all-partitions G*
**using** *assms lm19c* **by** *blast* **ultimately show** *?thesis* **using** *lm19c* **by** *blast*
**qed**

**corollary** *lm19b*: *possibleAllocationsRel N1 G* ⊆ *possibleAllocationsRel* (*N1* ∪ *N2*) *G*
**using** *lm19d* **by** (*metis subsetI*)

**lemma assumes** *x*≠{} **shows** *is-partition* {*x*} **unfolding** *is-partition-def* **using** *assms is-partition-def* **by** *force*

**lemma** *lm20d*: **assumes** ⋃ *P1* ∩ (⋃ *P2*) = {} *is-partition P1 is-partition P2 X* ∈ *P1* ∪ *P2 Y* ∈ *P1* ∪ *P2*
*X* ∩ *Y* ≠ {} **shows** (*X* = *Y*) **unfolding** *is-partition-def* **using** *assms is-partition-def* **by** *fast*

**lemma** *lm20e*: **assumes** ⋃ *P1* ∩ (⋃ *P2*) = {} *is-partition P1 is-partition P2 X* ∈ *P1* ∪ *P2 Y* ∈ *P1* ∪ *P2*
(*X* = *Y*) **shows** *X* ∩ *Y* ≠ {} **unfolding** *is-partition-def* **using** *assms is-partition-def* **by** *fast*

**lemma** *lm20*: **assumes** ⋃ *P1* ∩ (⋃ *P2*) = {} *is-partition P1 is-partition P2*
**shows** *is-partition* (*P1* ∪ *P2*) **unfolding** *is-partition-def* **using** *assms lm20d*

*lm20e* **by** *metis*

**lemma** *lm21*: *Range Q* ∪ (*Range* (*P outside* (*Domain Q*))) = *Range* (*P* +∗ *Q*)
**unfolding** *paste-def Range-Un-eq Un-commute* **by** (*metis*(*no-types*))

**lemma** *lll77c*: **assumes** *a1* ∈ *injectionsUniverse a2* ∈ *injectionsUniverse Range a1* ∩ (*Range a2*)={}
*Domain a1* ∩ (*Domain a2*) = {} **shows** *a1* ∪ *a2* ∈ *injectionsUniverse*
**using** *assms disj-Un-runiq* **by** (*metis* (*no-types*) *Domain-converse converse-Un mem-Collect-eq*)

**lemma** *lm22*: **assumes** *R* ∈ *partitionValuedUniverse* **shows** *is-partition* (*Range R*)
**using** *assms*
**proof** −
  **obtain** *P* **where**
  *0*: *P* ∈ *partitionsUniverse* & *R* ⊆ *UNIV* × *P* **using** *assms* **by** *blast*
  **have** *Range R* ⊆ *P* **using** *0* **by** *fast*
  **then show** *?thesis* **using** *0 mem-Collect-eq subset-is-partition* **by** (*metis*)
**qed**

**lemma** *lm23*: **assumes** *a1* ∈ *allocationsUniverse a2* ∈ *allocationsUniverse* ⋃ (*Range a1*) ∩ (⋃ (*Range a2*))={}
*Domain a1* ∩ (*Domain a2*) = {} **shows** *a1* ∪ *a2* ∈ *allocationsUniverse*
**proof** −
  **let** *?a*=*a1* ∪ *a2* **let** *?b1*=*a1*ˆ−*1* **let** *?b2*=*a2*ˆ−*1* **let** *?r*=*Range* **let** *?d*=*Domain*
  **let** *?I*=*injectionsUniverse* **let** *?P*=*partitionsUniverse* **let** *?PV*=*partitionValuedUniverse*
**let** *?u*=*runiq*
  **let** *?b*=*?a*ˆ−*1* **let** *?p*=*is-partition*
  **have** *?p* (*?r a1*) & *?p* (*?r a2*) **using** *assms lm22* **by** *blast* **then**
  **moreover have** *?p* (*?r a1* ∪ *?r a2*) **using** *assms* **by** (*metis lm20*)
  **then moreover have** (*?r a1* ∪ *?r a2*) ∈ *?P* **by** *simp*
  **moreover have** *?r ?a* = (*?r a1* ∪ *?r a2*) **using** *assms* **by** *fast*
  **ultimately moreover have** *?p* (*?r ?a*) **using** *lm20 assms* **by** *fastforce*
  **then moreover have** *?a* ∈ *?PV* **using** *assms* **by** *fast*
  **moreover have** *?r a1* ∩ (*?r a2*) ⊆ *Pow* (⋃ (*?r a1*) ∩ (⋃ (*?r a2*))) **by** *auto*
  **ultimately moreover have** {} ∉ (*?r a1*) & {} ∉ (*?r a2*) **using** *is-partition-def*
**by** (*metis Int-empty-left*)
  **ultimately moreover have** *?r a1* ∩ (*?r a2*) = {} **using** *assms lm22 is-partition-def*
**by** *auto*
  **ultimately moreover have** *?a* ∈ *?I* **using** *lll77c assms* **by** *fastforce*
  **ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lm27*: **assumes** *a* ∈ *injectionsUniverse* **shows** *a* − *b* ∈ *injectionsUniverse*
**using** *assms*
**by** (*metis* (*lifting*) *Diff-subset converse-mono mem-Collect-eq subrel-runiq*)

**lemma** *lm30b*: {*a*. *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions G*} =

$(Range -` (all\text{-}partitions\ G)) \cap (Domain -`(Pow\ N))$
**by** *fastforce*

**lemma** *lm30*: *possibleAllocationsRel N G* = *injectionsUniverse* ∩ ((*Range* −‘ (*all-partitions G*))
∩ (*Domain* −‘(*Pow N*)))
**using** *lm19 lm30b* **by** *metis*

**lemma** *lm28a*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows** (*a*ˆ−*1* ∈ *injections* (*Range a*) *N* & *Range a* ∈ *all-partitions G*)
**using** *assms*
**by** (*metis* (*mono-tags*, *hide-lams*) *lm19c lm47*)

**lemma** *lm28c*: **assumes** *a*ˆ−*1* ∈ *injections* (*Range a*) *N Range a* ∈ *all-partitions G*
**shows** *a* ∈ *possibleAllocationsRel N G* **using** *assms image-iff* **by** *fastforce*

**lemma** *lm28*: *a* ∈ *possibleAllocationsRel N G* = (*a*ˆ−*1* ∈ *injections* (*Range a*) *N* & *Range a* ∈ *all-partitions G*)
**using** *lm28a lm28c* **by** *metis*

**lemma** *lm28d*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows** (*a* ∈ *injections* (*Domain a*) (*Range a*)
& *Range a* ∈ *all-partitions G* & *Domain a* ⊆ *N*) **using** *assms lm28a*
**by** (*metis* (*erased*, *lifting*) *Domain-converse converse-converse injectionsI injections-def mem-Collect-eq order-refl*)

**lemma** *lm28e*: **assumes** *a* ∈ *injections* (*Domain a*) (*Range a*)
*Range a* ∈ *all-partitions G Domain a* ⊆ *N* **shows** *a* ∈ *possibleAllocationsRel N G*
**using** *assms mem-Collect-eq lm19c injections-def* **by** (*metis* (*erased*, *lifting*))

**lemma** *lm28b*: *a* ∈ *possibleAllocationsRel N G* = (*a* ∈ *injections* (*Domain a*) (*Range a*)
& *Range a* ∈ *all-partitions G* & *Domain a* ⊆ *N*) **using** *lm28d lm28e* **by** *metis*

**lemma** *lm29*: *possibleAllocationsRel N G* ⊇ *injectionsUniverse* ∩ (*Range* −‘ (*all-partitions G*))
∩ (*Domain* −‘(*Pow N*)) **using** *subsetI Int-assoc lm30*
**by** *metis*

**corollary** *lm31*: *possibleAllocationsRel N G* = *injectionsUniverse* ∩ (*Range* −‘ (*all-partitions G*))
∩ (*Domain* −‘(*Pow N*)) **using** *lm30 Int-assoc* **by** (*metis*)

**lemma** *lm32*: **assumes** *a* ∈ *partitionValuedUniverse* **shows** *a* − *b* ∈ *partitionValuedUniverse*
**using** *assms subset-is-partition* **by** *fast*

**lemma** *lm35*: **assumes** *a* ∈ *allocationsUniverse* **shows** *a* − *b* ∈ *allocationsUni-*

*verse* **using** *assms*
*lm27 lm32* **by** *auto*

**lemma** *lm33*: **assumes** $a \in$ *injectionsUniverse* **shows** $a \in$ *injections* (*Domain a*) (*Range a*)
**using** *assms* **by** (*metis* (*lifting*) *injectionsI mem-Collect-eq order-refl*)

**lemma** *lm34*: **assumes** $a \in$ *allocationsUniverse* **shows** $a \in$ *possibleAllocationsRel* (*Domain a*) ($\bigcup$ (*Range a*))
**proof** −
**let** *?r=Range* **let** *?p=is-partition* **let** *?P=all-partitions* **have** *?p* (*?r a*) **using** *assms lm22 Int-iff* **by** *blast* **then have** *?r a* $\in$ *?P* ($\bigcup$ (*?r a*)) **unfolding** *all-partitions-def*

**using** *is-partition-of-def mem-Collect-eq* **by** (*metis*) **then show** *?thesis* **using** *assms IntI Int-lower1 equalityE lm19 mem-Collect-eq set-rev-mp* **by** (*metis* (*lifting*, *no-types*))
**qed**

**lemma** *lm36*: $\{X\} \in$ *partitionsUniverse* = ($X \neq \{\}$) **using** *is-partition-def* **by** *fastforce*

**lemma** *lm36b*: $\{(x,\ X)\} - \{(x,\ \{\})\} \in$ *partitionValuedUniverse* **using** *lm36* **by** *auto*

**lemma** *runiq* $\{(x,X)\}$
**by** (*metis runiq-singleton-rel*)

**lemma** *lm37*: $\{(x,\ X)\} \in$ *injectionsUniverse* **unfolding** *runiq-basic* **using** *runiq-singleton-rel* **by** *blast*

**lemma** *lm38*: $\{(x,X)\} - \{(x,\{\})\} \in$ *allocationsUniverse* **using** *lm36b lm37 lm27 Int-iff* **by** (*metis* (*no-types*))

**lemma** **assumes** *is-partition Y X* $\subseteq$ *Y* **shows** *is-partition X* **using** *assms subset-is-partition* **by** (*metis*(*no-types*))

**lemma** *lm41*: **assumes** *is-partition PP is-partition* (*Union PP*) **shows** *is-partition* (*Union ‘ PP*)
**proof** −
**let** *?p=is-partition* **let** *?U=Union* **let** *?P2=?U PP* **let** *?P1=?U ‘ PP* **have**
*0*: $\forall$ $X \in$*?P1*. $\forall$ $Y \in$ *?P1*. ($X \cap Y = \{\} \longrightarrow X \neq Y$) **using** *assms is-partition-def Int-absorb*
*Int-empty-left UnionI Union-disjoint ex-in-conv imageE* **by** (*metis* (*hide-lams*, *no-types*))
**{**
  **fix** *X Y* **assume**
  *2*: $X \in$ *?P1* & $Y \in$*?P1* & $X \neq Y$
  **then obtain** *XX YY* **where**
  *1*: $X =$ *?U XX* & $Y=$*?U YY* & *XX*$\in$*PP* & *YY*$\in$*PP* **by** *blast*
  **then have** $XX \subseteq$ *Union PP* & $YY \subseteq$ *Union PP* & $XX \cap YY = \{\}$

**using** *2 1 is-partition-def assms*(*1*) *Sup-upper* **by** *metis*
**then moreover have** $\forall\ x{\in}XX.\ \forall\ y{\in}YY.\ x \cap y = \{\}$ **using** *1 assms*(*2*) *is-partition-def*
**by** (*metis IntI empty-iff subsetCE*)
**ultimately have** $X \cap Y{=}\{\}$ **using** *assms 0 1 2 is-partition-def* **by** *auto*
**}**
**then show** *?thesis* **using** *0 is-partition-def* **by** *metis*
**qed**

**lemma** *lm43*: **assumes** $a \in$ *allocationsUniverse* **shows**
$(a - ((X{\cup}\{i\}){\times}(Range\ a))) \cup (\{(i, \bigcup\ (a``(X \cup \{i\})))\} - \{(i,\{\})\}) \in$ *allocationsUniverse* &
$\bigcup\ (Range\ ((a - ((X{\cup}\{i\}){\times}(Range\ a))) \cup (\{(i, \bigcup\ (a``(X \cup \{i\})))\} - \{(i,\{\})\})))$
$= \bigcup(Range\ a)$
**proof** −
**let** *?d=Domain* **let** *?r=Range* **let** *?U=Union* **let** *?p=is-partition* **let** *?P=partitionsUniverse* **let** *?u=runiq*
**let** $?Xi{=}X \cup \{i\}$ **let** $?b{=}?Xi \times (?r\ a)$ **let** $?a1{=}a - ?b$ **let** $?Yi{=}a``?Xi$ **let** $?Y{=}?U\ ?Yi$
**let** $?A2{=}\{(i,\ ?Y)\}$ **let** $?a3{=}\{(i,\{\})\}$ **let** $?a2{=}?A2 - ?a3$ **let** *?aa1=a outside ?Xi*
**let** $?c{=}?a1 \cup ?a2$ **let** $?t1{=}?c \in$ *allocationsUniverse* **have**
*7*: $?U(?r(?a1{\cup}?a2)){=}?U(?r\ ?a1) \cup (?U(?r\ ?a2))$ **by** (*metis Range-Un-eq Union-Un-distrib*)
**have**
*5*: $?U(?r\ a) \subseteq ?U(?r\ ?a1) \cup ?U(a``?Xi)$ & $?U(?r\ ?a1) \cup ?U(?r\ ?a2) \subseteq ?U(?r\ a)$ **by** *blast* **have**
*1*: $?u\ a$ & $?u\ (a\hat{\ }{-}1)$ & $?p\ (?r\ a)$ & $?r\ ?a1 \subseteq ?r\ a$ & $?Yi \subseteq ?r\ a$
**using** *assms Int-iff lm22 mem-Collect-eq* **by** *auto* **then have**
*2*: $?p\ (?r\ ?a1)$ & $?p\ ?Yi$ **using** *subset-is-partition* **by** *metis* **have**
$?a1 \in$ *allocationsUniverse* & $?a2 \in$ *allocationsUniverse* **using** *lm38 assms*(*1*) *lm35* **by** *fastforce* **then have**
$(?a1 = \{\} \vee ?a2 = \{\}) \longrightarrow ?t1$ **using** *Un-empty-left* **by** (*metis* (*lifting*, *no-types*) *Un-absorb2 empty-subsetI*) **moreover have**
$(?a1 = \{\} \vee ?a2 = \{\}) \longrightarrow ?U\ (?r\ a) = ?U\ (?r\ ?a1) \cup ?U\ (?r\ ?a2)$ **by** *fast*
**ultimately have**
*3*: $(?a1 = \{\} \vee ?a2 = \{\}) \longrightarrow$ *?thesis* **using** *7* **by** *presburger*
**{**
**assume**
*0*: $?a1{\neq}\{\}$ & $?a2{\neq}\{\}$ **then have** $?r\ ?a2{\supseteq}\{?Y\}$ **using** *Diff-cancel Range-insert empty-subsetI*
*insert-Diff-single insert-iff insert-subset* **by** (*metis* (*hide-lams*, *no-types*)) **then have**
*6*: $?U\ (?r\ a) = ?U\ (?r\ ?a1) \cup ?U\ (?r\ ?a2)$ **using** *5* **by** *blast*
**have** $?r\ ?a1 \neq \{\}$ & $?r\ ?a2 \neq \{\}$ **using** *0* **by** *auto*
**moreover have** $?r\ ?a1 \subseteq a``(?d\ ?a1)$ **using** *assms* **by** *blast*
**moreover have** $?Yi \cap (a``(?d\ a - ?Xi)) = \{\}$ **using** *assms 0 1 lm40*
**by** (*metis Diff-disjoint*)
**ultimately moreover have** $?r\ ?a1 \cap ?Yi = \{\}$ & $?Yi \neq \{\}$ **by** *blast*
**ultimately moreover have** $?p\ \{?r\ ?a1,\ ?Yi\}$ **unfolding** *is-partition-def* **us-**

**ing**
*IntI Int-commute empty-iff insert-iff subsetI subset-empty* **by** *metis*
**moreover have** $?U\ \{?r\ ?a1,\ ?Yi\} \subseteq ?r\ a$ **by** *auto*
**then moreover have** $?p\ (?U\ \{?r\ ?a1,\ ?Yi\})$ **by** (*metis 1 Outside-def subset-is-partition*)
**ultimately moreover have** $?p\ (?U`\{(?r\ ?a1),\ ?Yi\})$ **using** *lm41* **by** *fast*
**moreover have** $... = \{?U\ (?r\ ?a1),\ ?Y\}$ **by** *force*
**ultimately moreover have** $\forall\ x \in ?r\ ?a1.\ \forall\ y{\in}?Yi.\ x \neq y$
**using** *IntI empty-iff* **by** *metis*
**ultimately moreover have** $\forall\ x \in ?r\ ?a1.\ \forall\ y{\in}?Yi.\ x \cap y = \{\}$ **using** *0 1 2 is-partition-def*
**by** (*metis set-rev-mp*)
**ultimately have** $?U\ (?r\ ?a1) \cap ?Y = \{\}$ **using** *lm42*
**proof** −
**have** $\forall v0.\ v0 \in Range\ (a - (X \cup \{i\}) \times Range\ a) \longrightarrow (\forall v1.\ v1 \in a\ ``\ (X \cup \{i\}) \longrightarrow v0 \cap v1 = \{\})$
**by** (*metis* (*no-types*) $‹\forall x{\in}Range\ (a - (X \cup \{i\}) \times Range\ a).\ \forall y{\in}a\ ``\ (X \cup \{i\}).\ x \cap y = \{\}›$)
**thus** $\bigcup Range\ (a - (X \cup \{i\}) \times Range\ a) \cap \bigcup(a\ ``\ (X \cup \{i\})) = \{\}$ **by** *blast*
**qed then have**
$?U\ (?r\ ?a1) \cap (?U\ (?r\ ?a2)) = \{\}$ **by** *blast*
**moreover have** $?d\ ?a1 \cap (?d\ ?a2) = \{\}$ **by** *blast*
**moreover have** $?a1 \in allocationsUniverse$ **using** *assms*(*1*) *lm35* **by** *blast*
**moreover have** $?a2 \in allocationsUniverse$ **using** *lm38* **by** *fastforce*
**ultimately have** $?a1 \in allocationsUniverse\ \&$
$?a2 \in allocationsUniverse\ \&$
$\bigcup Range\ ?a1 \cap \bigcup Range\ ?a2 = \{\}\ \&\ Domain\ ?a1 \cap Domain\ ?a2 = \{\}$
**by** *blast* **then have**
*?t1* **using** *lm23* **by** *auto*
**then have** *?thesis* **using** *6 7* **by** *presburger*
}
**then show** *?thesis* **using** *3* **by** *linarith*
**qed**

**lemma** *lm45*: **assumes** $Domain\ a \cap X \neq \{\}$ $a \in allocationsUniverse$ **shows** $\bigcup(a``X) \neq \{\}$
**proof** −
**let** *?p=is-partition* **let** *?r=Range*
**have** $?p\ (?r\ a)$ **using** *assms Int-iff lm22* **by** *auto*
**moreover have** $a``X \subseteq ?r\ a$ **by** *fast*
**ultimately have** $?p\ (a``X)$ **using** *assms subset-is-partition* **by** *blast*
**moreover have** $a``X \neq \{\}$ **using** *assms* **by** *fast*
**ultimately show** *?thesis* **by** (*metis Union-member all-not-in-conv no-empty-eq-class*)
**qed**

**corollary** *lm45b*: **assumes** $Domain\ a \cap X \neq \{\}$ $a \in allocationsUniverse$ **shows** $\{\bigcup(a``(X{\cup}\{i\}))\}{-}\{\{\}\} = \{\bigcup(a``(X{\cup}\{i\}))\}$ **using** *assms lm45* **by** *fast*

**corollary** *lm43b*: **assumes** $a \in allocationsUniverse$ **shows**
$(a\ outside\ (X{\cup}\{i\})) \cup (\{i\}{\times}(\{\bigcup(a``(X{\cup}\{i\}))\}{-}\{\{\}\})) \in allocationsUniverse\ \&$

$\bigcup(Range((a\ outside\ (X\cup\{i\}))\ \cup\ (\{i\}\times(\{\bigcup(a``(X\cup\{i\}))\}-\{\{\}\})))) = \bigcup(Range\ a)$
**proof** −
**have** $a - ((X\cup\{i\})\times(Range\ a)) = a\ outside\ (X \cup \{i\})$ **using** *Outside-def* **by** *metis*
**moreover have** $(a - ((X\cup\{i\})\times(Range\ a))) \cup (\{(i, \bigcup (a``(X \cup \{i\})))\} - \{(i,\{\})\}) \in allocationsUniverse$
**using** *assms lm43* **by** *fastforce*
**moreover have** $\bigcup (Range\ ((a - ((X\cup\{i\})\times(Range\ a))) \cup (\{(i, \bigcup (a``(X \cup \{i\})))\} - \{(i,\{\})\}))) = \bigcup(Range\ a)$
**using** *assms lm43* **by** (*metis* (*no-types*))
**ultimately have**
$(a\ outside\ (X\cup\{i\})) \cup (\{(i, \bigcup (a``(X \cup \{i\})))\} - \{(i,\{\})\}) \in allocationsUniverse$ &
$\bigcup (Range\ ((a\ outside\ (X\cup\{i\})) \cup (\{(i, \bigcup (a``(X \cup \{i\})))\} - \{(i,\{\})\}))) = \bigcup(Range\ a)$ **by**
*presburger*
**moreover have** $\{(i, \bigcup (a``(X \cup \{i\})))\} - \{(i,\{\})\} = \{i\} \times (\{\bigcup (a``(X\cup\{i\}))\} - \{\{\}\})$
**by** *fast*
**ultimately show** *?thesis* **by** *auto*
**qed**

**corollary** *lm43c*: **assumes** $a \in allocationsUniverse\ Domain\ a \cap X \neq \{\}$ **shows**
$(a\ outside\ (X\cup\{i\})) \cup (\{i\}\times\{\bigcup(a``(X\cup\{i\}))\}) \in allocationsUniverse$ &
$\bigcup(Range((a\ outside\ (X\cup\{i\})) \cup (\{i\}\times\{\bigcup(a``(X\cup\{i\}))\}))) = \bigcup(Range\ a)$
**using** *assms lm43b lm45b*
**proof** −
**let** $?t1=(a\ outside\ (X\cup\{i\})) \cup (\{i\}\times(\{\bigcup(a``(X\cup\{i\}))\}-\{\{\}\})) \in allocationsUniverse$
**let** $?t2=\bigcup(Range((a\ outside\ (X\cup\{i\})) \cup (\{i\}\times(\{\bigcup(a``(X\cup\{i\}))\}-\{\{\}\})))) = \bigcup(Range\ a)$
**have**
*0*: $a \in allocationsUniverse$ **using** *assms*(*1*) **by** *fast*
**then have** *?t1* & *?t2* **using** *lm43b*
**proof** −

**have** $a \in allocationsUniverse \longrightarrow a -| (X \cup \{i\}) \cup \{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\}) \in allocationsUniverse$
**using** *lm43b* **by** *fastforce*
**hence** $a -| (X \cup \{i\}) \cup \{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\}) \in allocationsUniverse$
**by** (*metis 0*)
**thus** $a -| (X \cup \{i\}) \cup \{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\}) \in allocationsUniverse \wedge \bigcup Range\ (a -| (X \cup \{i\}) \cup \{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\})) = \bigcup Range\ a$
**using** *0* **by** (*metis* (*no-types*) *lm43b*)
**qed**
**moreover have**
*1*: $\{\bigcup(a``(X\cup\{i\}))\}-\{\{\}\} = \{\bigcup(a``(X\cup\{i\}))\}$ **using** *lm45 assms* **by** *fast*
**ultimately show** *?thesis* **by** *auto*
**qed**

**abbreviation** *condition1 b i* == ($\forall$ *t t'*. (*trivial t* & *trivial t'* & *Union t* $\subseteq$ *Union t'*) $\longrightarrow$
*setsum b* ($\{i\}\times t$) $\leq$ *setsum b* ($\{i\}\times t'$))

**abbreviation** *condition1b b i* == $\forall$ *X Y*. *setsum b* ($\{i\}\times\{X\}$) $\leq$ *setsum b* ($\{i\}\times\{X \cup Y\}$)

**lemma** *lm46*: **assumes** *condition1 b i runiq a* **shows**
*setsum b* ($\{i\}\times$((*a outside X*)``$\{i\}$)) $\leq$ *setsum b* ($\{i\}\times\{\bigcup$(*a*``($X\cup\{i\}$))$\}$)
**proof** –
  **let** *?u=runiq* **let** *?I*=$\{i\}$ **let** *?R=a outside X* **let** *?U=Union* **let** *?Xi=X* $\cup$*?I*
  **let** *?t1=?R``?I* **let** *?t2*=$\{$*?U* (*a``?Xi*)$\}$
  **have** *?U* (*?R``?I*) $\subseteq$ *?U* (*?R``*($X\cup$*?I*)) **by** *blast*
  **moreover have** ... $\subseteq$ *?U* (*a``*($X\cup$*?I*)) **using** *Outside-def* **by** *blast*
  **ultimately have** *?U* (*?R``?I*) $\subseteq$ *?U* (*a``*($X\cup$*?I*)) **by** *auto*
  **then have**
  *0*: *?U ?t1* $\subseteq$ *?U ?t2* **by** *auto*
  **have** *?u a* **using** *assms* **by** *fast*
  **moreover have** *?R* $\subseteq$ *a* **using** *Outside-def* **by** *blast* **ultimately**
  **have** *?u ?R* **using** *subrel-runiq* **by** *metis*
  **then have** *trivial ?t1* **by** (*metis runiq-alt*)
  **moreover have** *trivial ?t2* **by** (*metis trivial-singleton*)
  **ultimately show** *?thesis* **using** *assms 0* **by** *blast*
**qed**

**lemma** *lm48*: *possibleAllocationsRel N G* $\subseteq$ *injectionsUniverse* **using** *lm19* **by** *fast*

**lemma** *lm49*: *possibleAllocationsRel N G* $\subseteq$ *partitionValuedUniverse*
**using** *assms lm47 is-partition-of-def is-partition-def* **by** *blast*

**corollary** *lm50*: *possibleAllocationsRel N G* $\subseteq$ *allocationsUniverse* **using** *lm48 lm49*
**by** (*metis* (*lifting*, *mono-tags*) *inf.bounded-iff*)

**lemma** *mm45*: **assumes** *XX* $\in$ *partitionValuedUniverse* **shows** $\{\}$ $\notin$ *Range XX* **using** *assms*
*mem-Collect-eq no-empty-eq-class* **by** *auto*
**corollary** *mm45b*: **assumes** *a* $\in$ *possibleAllocationsRel N G* **shows** $\{\}$ $\notin$ *Range a* **using** *assms mm45*
*lm50* **by** *blast*
**lemma** *mm63*: **assumes** *a* $\in$ *possibleAllocationsRel N G* **shows** *Range a* $\subseteq$ *Pow G*
**using** *assms lm47 is-partition-of-def* **by** (*metis subset-Pow-Union*)
**corollary** *mm63b*: **assumes** *a* $\in$ *possibleAllocationsRel N G* **shows** *Domain a* $\subseteq$ *N* & *Range a* $\subseteq$ *Pow G* $-$ $\{\{\}\}$ **using**
*assms mm63 insert-Diff-single mm45b subset-insert lm47* **by** *metis*

**corollary** *mm63c*: **assumes** $a \in$ *possibleAllocationsRel N G* **shows** $a \subseteq N \times$ (*Pow* $G - \{\{\}\}$)
**using** *assms mm63b* **by** *blast*
**corollary** *mm63e*: *possibleAllocationsRel N G* $\subseteq$ *Pow* ($N\times$(*Pow* $G-\{\{\}\}$)) **using** *mm63c* **by** *blast*

**lemma** *lm51*: **assumes**
$a \in$ *possibleAllocationsRel N G*
$i\in N-X$
*Domain* $a \cap X \neq \{\}$
**shows**
*a outside* $(X \cup \{i\}) \cup (\{i\} \times \{\bigcup(a``(X\cup\{i\}))\}) \in$ *possibleAllocationsRel* $(N-X)$ ($\bigcup$ (*Range a*))
**proof** −
**let** *?R=a outside X* **let** *?I=*$\{i\}$ **let** *?U=Union* **let** *?u=runiq* **let** *?r=Range* **let** *?d=Domain*
**let** *?aa=a outside* $(X \cup \{i\}) \cup (\{i\} \times \{$*?U*$(a``(X\cup\{i\}))\})$ **have**
*1*: $a \in$ *allocationsUniverse* **using** *assms*(*1*) *lm50 set-rev-mp* **by** *blast*
**have** $i \notin X$ **using** *assms* **by** *fast* **then have**
*2*: *?d a* $- X \cup \{i\} =$ *?d a* $\cup \{i\} - X$ **by** *fast*
**have** $a \in$ *allocationsUniverse* **using** *1* **by** *fast* **moreover have** *?d a* $\cap X \neq \{\}$ **using** *assms* **by** *fast*
**ultimately have** *?aa* $\in$ *allocationsUniverse* & *?U* (*?r ?aa*) = *?U* (*?r a*) **apply** (*rule lm43c*) **done**
**then have** *?aa* $\in$ *possibleAllocationsRel* (*?d ?aa*) (*?U* (*?r a*))
**using** *lm34* **by** (*metis* (*lifting*, *mono-tags*))
**then have** *?aa* $\in$ *possibleAllocationsRel* (*?d ?aa* $\cup$ (*?d a* $- X \cup \{i\}$)) (*?U* (*?r a*))
**by** (*metis lm19d*)
**moreover have** *?d a* $- X \cup \{i\} =$ *?d ?aa* $\cup$ (*?d a* $- X \cup \{i\}$) **using** *Outside-def* **by** *auto*
**ultimately have** *?aa* $\in$ *possibleAllocationsRel* (*?d a* $- X \cup \{i\}$) (*?U* (*?r a*)) **by** *simp*
**then have** *?aa* $\in$ *possibleAllocationsRel* (*?d a* $\cup \{i\} - X$) (*?U* (*?r a*)) **using** *2* **by** *simp*
**moreover have** *?d a* $\subseteq N$ **using** *assms*(*1*) *lm19c* **by** *metis*
**then moreover have** (*?d a* $\cup \{i\} - X) \cup (N - X) = N - X$ **using** *assms* **by** *fast*
**ultimately have** *?aa* $\in$ *possibleAllocationsRel* $(N - X)$ (*?U* (*?r a*)) **using** *lm19b*
**by** (*metis* (*lifting*, *no-types*) *in-mono*)
**then show** *?thesis* **by** *fast*
**qed**

**lemma** *lm52*: **assumes**
*condition1* (*b*::- => *real*) *i*
$a \in$ *allocationsUniverse*
*Domain* $a \cap X \neq \{\}$
*finite a* **shows**

*setsum b* (*a outside X*) ≤ *setsum b* (*a outside* (*X* ∪ {*i*}) ∪ ({*i*} × {⋃ (*a*‘‘(*X*∪{*i*}))}))
**proof** −
**let** *?R=a outside X* **let** *?I*={*i*} **let** *?U=Union* **let** *?u=runiq* **let** *?r=Range* **let** *?d=Domain*
**let** *?aa=a outside* (*X* ∪ {*i*}) ∪ ({*i*} × {*?U*(*a*‘‘(*X*∪{*i*}))})
**have** *a* ∈ *injectionsUniverse* **using** *assms* **by** *fast* **then have**
*0*: *?u a* **by** *simp*
**moreover have** *?R* ⊆ *a* & *?R*−−*i* ⊆ *a* **using** *Outside-def* **by** *blast*
**ultimately have** *finite* (*?R* −− *i*) & *?u* (*?R*−−*i*) & *?u ?R* **using** *finite-subset subrel-runiq*
**by** (*metis assms*(*4*))
**then moreover have** *trivial* ({*i*}×(*?R*‘‘{*i*})) **using** *runiq-def*
**by** (*metis ll40 trivial-singleton*)
**moreover have** ∀ *X*. (*?R* −− *i*) ∩ ({*i*}×*X*)={} **using** *outside-reduces-domain* **by** *force*
**ultimately have**
*1*: *finite* (*?R*−−*i*) & *finite* ({*i*}×(*?R*‘‘{*i*})) & (*?R* −− *i*) ∩ ({*i*}×(*?R*‘‘{*i*}))={} &
*finite* ({*i*} × {*?U*(*a*‘‘(*X*∪{*i*}))}) & (*?R* −− *i*) ∩ ({*i*} × {*?U*(*a*‘‘(*X*∪{*i*}))})={}

**using** *Outside-def lm54* **by** *fast*
**have** *?R* = (*?R* −− *i*) ∪ ({*i*}×*?R*‘‘{*i*}) **by** (*metis l39*)
**then have** *setsum b ?R* = *setsum b* (*?R* −− *i*) + *setsum b* ({*i*}×(*?R*‘‘{*i*}))
**using** *1 setsum.union-disjoint* **by** (*metis* (*lifting*) *setsum.union-disjoint*)
**moreover have** *setsum b* ({*i*}×(*?R*‘‘{*i*})) ≤ *setsum b* ({*i*}×{*?U*(*a*‘‘(*X*∪{*i*}))}) **using** *lm46*
*assms*(*1*) *0* **by** *metis*
**ultimately have** *setsum b ?R* ≤ *setsum b* (*?R* −− *i*) + *setsum b* ({*i*}×{*?U*(*a*‘‘(*X*∪{*i*}))}) **by** *linarith*
**moreover have** ... = *setsum b* (*?R* −− *i* ∪ ({*i*} × {*?U*(*a*‘‘(*X*∪{*i*}))}))
**using** *1 setsum.union-disjoint* **by** *auto*
**moreover have** ... = *setsum b ?aa* **by** (*metis ll52*)
**ultimately show** *?thesis* **by** *linarith*
**qed**

**lemma** *lm55*: **assumes** *finite X XX* ∈ *all-partitions X* **shows** *finite XX* **using** *all-partitions-def is-partition-of-def*
**by** (*metis assms*(*1*) *assms*(*2*) *finite-UnionD mem-Collect-eq*)

**lemma** *lm58*: **assumes** *finite N finite G a* ∈ *possibleAllocationsRel N G*
**shows** *finite a* **using** *assms lm57 rev-finite-subset* **by** (*metis lm28b lm55*)

**lemma** *lm59*: **assumes** *finite N finite G* **shows** *finite* (*possibleAllocationsRel N G*)
**proof** −
**have** *finite* (*Pow*(*N*×(*Pow G*−{{}}))) **using** *assms finite-Pow-iff* **by** *blast*
**then show** *?thesis* **using** *mm63e rev-finite-subset* **by** (*metis*(*no-types*))
**qed**

**corollary** *lm53*: **assumes** *condition1* $(b{::}\text{-} => real)$ *i a* $\in$ *possibleAllocationsRel N G i*$\in$*N*$-$*X*
*Domain a* $\cap X \neq \{\}$ *finite N finite G* **shows**
*Max* $((setsum\ b)`(possibleAllocationsRel\ (N-X)\ G)) \geq setsum\ b\ (a\ outside\ X)$
**proof** $-$
**let** *?aa=a outside* $(X \cup \{i\}) \cup (\{i\} \times \{\bigcup(a``(X\cup\{i\}))\})$
**have** *condition1* $(b{::}\text{-} => real)$ *i* **using** *assms(1)* **by** *fast*
**moreover have** $a \in$ *allocationsUniverse* **using** *assms(2) lm50* **by** *blast*
**moreover have** *Domain a* $\cap X \neq \{\}$ **using** *assms(4)* **by** *auto*
**moreover have** *finite a* **using** *assms lm58* **by** *metis* **ultimately have**
*0*: *setsum b* $(a\ outside\ X) \leq$ *setsum b ?aa* **by** $(rule\ lm52)$
**have** *?aa* $\in$ *possibleAllocationsRel* $(N-X)$ $(\bigcup$ $(Range\ a))$ **using** *assms lm51* **by** *metis*
**moreover have** $\bigcup$ $(Range\ a) = G$ **using** *assms lm47 is-partition-of-def* **by** *metis*
**ultimately have** *setsum b ?aa* $\in$ $(setsum\ b)`(possibleAllocationsRel\ (N-X)\ G)$
**by** $(metis\ imageI)$
**moreover have** *finite* $((setsum\ b)`(possibleAllocationsRel\ (N-X)\ G))$ **using** *assms lm59 assms(5,6)*
**by** $(metis\ finite\text{-}Diff\ finite\text{-}imageI)$
**ultimately have** *setsum b ?aa* $\leq$ *Max* $((setsum\ b)`(possibleAllocationsRel\ (N-X)\ G))$ **by** *auto*
**then show** *?thesis* **using** *0* **by** *linarith*
**qed**

**lemma assumes** $f \in$ *partitionValuedUniverse* **shows** $\{\} \notin$ *Range f* **using** *assms* **by** $(metis\ lm22\ no\text{-}empty\text{-}eq\text{-}class)$

**lemma** *mm33*: **assumes** *finite XX* $\forall X \in XX.$ *finite X is-partition XX* **shows** *card* $(\bigcup\ XX) =$ *setsum card XX* **using** *assms is-partition-def card-Union-disjoint* **by** *fast*

**corollary** *mm33b*: **assumes** *XX partitions X finite X finite XX* **shows** *card* $(\bigcup\ XX) =$ *setsum card XX* **using** *assms mm33* **by** $(metis\ is\text{-}partition\text{-}of\text{-}def\ lll41)$

**lemma** *setsum-Union-disjoint-4*: **assumes** $\forall A{\in}C.$ *finite A* $\forall A{\in}C.\ \forall B{\in}C.\ A \neq B \longrightarrow A\ Int\ B = \{\}$
**shows** *setsum f* $(Union\ C) =$ *setsum* $(setsum\ f)$ *C* **using** *assms setsum.Union-disjoint* **by** *fastforce*

**corollary** *setsum-Union-disjoint-2*: **assumes** $\forall x{\in}X.$ *finite x is-partition X* **shows**

*setsum f* $(\bigcup\ X) =$ *setsum* $(setsum\ f)$ *X* **using** *assms setsum-Union-disjoint-4 is-partition-def* **by** *fast*

**corollary** *setsum-Union-disjoint-3*: **assumes** $\forall x{\in}X.$ *finite x X partitions XX* **shows** *setsum f XX* $=$ *setsum* $(setsum\ f)$ *X* **using** *assms* **by** $(metis\ is\text{-}partition\text{-}of\text{-}def\ setsum\text{-}Union\text{-}disjoint\text{-}2)$

**corollary** *setsum-associativity*: **assumes** *finite x X partitions x* **shows** *setsum f x = setsum (setsum f) X* **using** *assms setsum-Union-disjoint-3* **by** (*metis is-partition-of-def lll41*)

**lemma** *lm19e*: **assumes** *a∈allocationsUniverse Domain a⊆N ⋃ Range a=G* **shows**

*a ∈possibleAllocationsRel N G* **using** *assms lm19c lm34* **by** (*metis* (*mono-tags*, *lifting*))

**corollary** *nn24a*: (*allocationsUniverse∩*{*a. Domain a⊆N & ⋃ Range a=G*})*⊆possibleAllocationsRel N G*
**using** *lm19e* **by** *fastforce*
**corollary** *nn24f*: *possibleAllocationsRel N G ⊆* {*a. Domain a⊆N*} **using** *lm47* **by** *blast*
**corollary** *nn24g*: *possibleAllocationsRel N G ⊆* {*a. ⋃ Range a=G*} **using** *is-partition-of-def lm47 mem-Collect-eq subsetI*
**by** (*metis*(*mono-tags*))
**corollary assumes** *a ∈ possibleAllocationsRel N G* **shows** ⋃ *Range a = G* **using** *assms*
**by** (*metis is-partition-of-def lm47*)
**corollary** *nn24e*:
*possibleAllocationsRel N G ⊆ allocationsUniverse &*
*possibleAllocationsRel N G ⊆* {*a. Domain a⊆N & ⋃ Range a=G*} **using** *nn24f nn24g*
*conj-subset-def lm50* **by** (*metis* (*no-types*))

**corollary** *nn24b*: *possibleAllocationsRel N G ⊆ allocationsUniverse∩*{*a. Domain a⊆N & ⋃ Range a=G*}
(**is** *?L ⊆ ?R1 ∩ ?R2*)
**proof** − **have** *?L ⊆ ?R1 & ?L ⊆ ?R2* **by** (*rule nn24e*) **thus** *?thesis* **by** *auto* **qed**

**corollary** *nn24*: *possibleAllocationsRel N G =* (*allocationsUniverse∩*{*a. Domain a⊆N & ⋃ Range a=G*})
(**is** *?L = ?R*)
**proof** −
  **have** *?L ⊆ ?R* **using** *nn24b* **by** *metis* **moreover have** *?R ⊆ ?L* **using** *nn24a* **by** *fast*
  **ultimately show** *?thesis* **by** *force*
**qed**

**corollary** *nn24c*: *b ∈ possibleAllocationsRel N G=*(*b∈allocationsUniverse& Domain b⊆N & ⋃ Range b = G*)
**using** *nn24 Int-Collect* **by** (*metis* (*mono-tags*, *lifting*))

**corollary** *lm35d*: **assumes** *a ∈ allocationsUniverse* **shows** *a outside X ∈ allocationsUniverse* **using** *assms Outside-def*
**by** (*metis* (*lifting*, *mono-tags*) *lm35*)

**end**

# 36 Termination theorem for uniform tie-breaking

**theory** *UniformTieBreaking*

**imports**
*StrictCombinatorialAuction*
*Universes*
*~~/src/HOL/Library/Code-Target-Nat*

**begin**

# 37 Termination theorem for the uniform tie-breaking scheme $\lambda N\ G\ bids\ random.\ linearCompletion'\ (pseudoAllocation\ (hd\ (perm2\ (takeAll\ (\lambda x.\ winningAllocationRel\ N\ (set\ G)\ (op \in x)\ bids)\ (possibleAllocationsAlg3\ N\ G))\ random)) <|\ (N \times finestpart\ (set\ G)))\ N\ (set\ G)$

**corollary** *lm03*: *winningAllocationsRel N G b* $\subseteq$ *possibleAllocationsRel N G*
**using** *lm02 mem-Collect-eq subsetI* **by** *auto*

**lemma** *lm35b*: **assumes** $a \in$ *allocationsUniverse* $c \subseteq a$ **shows** $c \in$ *allocationsUniverse*
**proof** − **have** $c=a-(a-c)$ **using** *assms*(*2*) **by** *blast* **thus** *?thesis* **using** *assms*(*1*) *lm35* **by** (*metis* (*no-types*)) **qed**
**lemma** *lm35c*: **assumes** $a \in$ *allocationsUniverse* **shows** *a outside X* $\in$ *allocationsUniverse*
**using** *assms lm35 Outside-def* **by** (*metis* (*no-types*))

**corollary** *lm38d*: $\{x\}\times(\{X\}-\{\{\}\}) \in$ *allocationsUniverse* **using** *lm38 nn43* **by** *metis*
**corollary** *lm38b*: $\{(x,\{y\})\} \in$ *allocationsUniverse* **using** *lm38 lm44 insert-not-empty*

**proof** −
 **have** $(x, \{y\}) \neq (x, \{\})$ **by** *blast*
 **thus** $\{(x, \{y\})\} \in$ *allocationsUniverse* **by** (*metis* (*no-types*) *insert-Diff-if insert-iff lm38 lm44*)
**qed**
**corollary** *lm38c*: *allocationsUniverse* $\neq \{\}$ **using** *lm38b* **by** *fast*
**corollary** *nn39*: $\{\} \in$ *allocationsUniverse* **using** *lm35b lm38b* **by** (*metis* (*lifting*, *mono-tags*) *empty-subsetI*)
**lemma** *mm87*: **assumes** $G \neq \{\}$ **shows** $\{G\} \in$ *all-partitions G* **using** *all-partitions-def is-partition-of-def*
*is-partition-def assms* **by** *force*
**lemma** *mm88*: **assumes** $n \in N$ **shows** $\{(G,n)\} \in$ *totalRels* $\{G\}$ *N* **using** *assms* **by** *force*

**lemma** *mm89*: **assumes** $n \in N$ **shows** $\{(G,n)\} \in$ *injections* $\{G\}$ $N$
**using** *assms possible-allocations-rel-def injections-def mm87 all-partitions-def is-partition-def is-partition-of-def lm26 mm88 lm37 lm24* **by** *fastforce*
**corollary** *mm90*: **assumes** $G \neq \{\}$ $n \in N$ **shows** $\{(G,n)\} \in$ *possible-allocations-rel* $G$ $N$
**proof** –
  **have** $\{(G,n)\} \in$ *injections* $\{G\}$ $N$ **using** *assms mm89* **by** *fast*
  **moreover have** $\{G\} \in$ *all-partitions* $G$ **using** *assms mm87* **by** *metis*
  **ultimately show** *?thesis* **using** *possible-allocations-rel-def* **by** *auto*
**qed**
**corollary** *mm90b*: **assumes** $N \neq \{\}$ $G \neq \{\}$ **shows** *possibleAllocationsRel* $N$ $G \neq \{\}$
**using** *assms mm90* **by** (*metis* (*hide-lams*, *no-types*) *equals0I image-insert insert-absorb insert-not-empty*)
**corollary** *mm91*: **assumes** $N \neq \{\}$ *finite* $N$ $G \neq \{\}$ *finite* $G$ **shows** *winningAllocationsRel* $N$ $G$ *bids* $\neq \{\}$ & *finite* (*winningAllocationsRel* $N$ $G$ *bids*)
**using** *assms mm90b lm59 argmax-non-empty-iff* **by** (*metis lm03 rev-finite-subset*)

**lemma** *mm52*: *possibleAllocationsRel* $N$ $\{\} \subseteq \{\{\}\}$ **using** *emptyset-part-emptyset3 mm51*
*lm28b mem-Collect-eq subsetI vimage-def* **by** *metis*

**lemma** *mm42*: **assumes** $a \in$ *possibleAllocationsRel* $N$ $G$ *finite* $G$ **shows** *finite* (*Range* $a$)
**using** *assms lm55* **by** (*metis lm28*)

**corollary** *mm44*: **assumes** $a \in$ *possibleAllocationsRel* $N$ $G$ *finite* $G$ **shows** *finite* $a$
**using** *assms mm42 mm43 finite-converse*
**by** (*metis* (*erased*, *hide-lams*) *Range-converse imageE lll81*)

**lemma assumes** $a \in$ *possibleAllocationsRel* $N$ $G$ **shows** $\bigcup$ *Range* $a = G$ **using** *assms*
**by** (*metis is-partition-of-def lm47*)

**lemma** *mm41*: **assumes** $a \in$ *possibleAllocationsRel* $N$ $G$ *finite* $G$ **shows** $\forall$ $y \in$ *Range* $a$. *finite* $y$ **using** *assms is-partition-of-def lm47* **by** (*metis Union-upper rev-finite-subset*)

**corollary** *mm33c*: **assumes** $a \in$ *possibleAllocationsRel* $N$ $G$ *finite* $G$ **shows** *card* $G$ = *setsum card* (*Range* $a$) **using** *assms mm33b mm42 lm47* **by** (*metis is-partition-of-def*)

**lemma** *mm66*: *LinearCompletion bids N G* =
$\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in N \times (Pow\ G-\{\{\}\})\}$ **by** *blast*
**corollary** *mm65b*:
$\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in N \times (Pow\ G-\{\{\}\})\}\ ||\ a =$
$\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in (N \times (Pow\ G - \{\{\}\})) \cap a\}$
**by** (*metis mm65*)
**corollary** *mm66b*: (*LinearCompletion bids N G*) || *a* =
$\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in (N \times (Pow\ G - \{\{\}\})) \cap a\}$
(**is** *?L=?R*) **using** *mm65b mm66*
**proof** −
**let** *?l=LinearCompletion*
**let** $?M=\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in N \times (Pow\ G-\{\{\}\})\}$
**have** *?l bids N G* = *?M* **by** (*rule mm66*)
**then have** *?L* = (*?M* || *a*) **by** *presburger*
**moreover have** ... = *?R* **by** (*rule mm65b*)
**ultimately show** *?thesis* **by** *presburger*
**qed**
**lemma** *mm66c*: (*partialCompletionOf bids*) ‘ $((N \times (Pow\ G - \{\{\}\})) \cap a) =$
$\{(pair,setsum\ (\%g.\ bids\ (fst\ pair,\ g))\ (finestpart\ (snd\ pair)))|pair.\ pair \in (N \times (Pow\ G-\{\{\}\})) \cap a\}$
**by** *blast*
**corollary** *mm66d*: (*LinearCompletion bids N G*) || *a* = (*partialCompletionOf bids*) ‘ $((N \times (Pow\ G - \{\{\}\})) \cap a)$
(**is** *?L=?R*)
**using** *mm66c mm66b*
**proof** −
**let** *?l=LinearCompletion* **let** *?p=partialCompletionOf* **let** *?M*={

(*pair*,*setsum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*)))|*pair*. *pair* ∈ (*N* × (*Pow G* − {{}})) ∩ *a*}
**have** *?L* = *?M* **by** (*rule mm66b*)
**moreover have** ... = *?R* **using** *mm66c* **by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**
**lemma** *mm57*: *inj-on* (*partialCompletionOf bids*) *UNIV* **using** *assms* **by** (*metis* (*lifting*) *fst-conv inj-on-inverseI*)
**corollary** *mm57b*: *inj-on* (*partialCompletionOf bids*) *X* **using** *fst-conv inj-on-inverseI* **by** (*metis* (*lifting*))
**lemma** *mm58*: *setsum snd* (*LinearCompletion bids N G*) =
*setsum* (*snd* ∘ (*partialCompletionOf bids*)) (*N* × (*Pow G* − {{}})) **using** *assms mm57b setsum.reindex* **by** *blast*
**corollary** *mm25*: *snd* (*partialCompletionOf bids pair*)=*setsum bids* (*omega pair*) **using** *mm24* **by** *force*
**corollary** *mm25b*: *snd* ∘ *partialCompletionOf bids* = (*setsum bids*) ∘ *omega* **using** *mm25* **by** *fastforce*

**lemma** *mm27*: **assumes** *finite* (*finestpart* (*snd pair*)) **shows**
*card* (*omega pair*) = *card* (*finestpart* (*snd pair*)) **using** *assms* **by** *force*

**corollary assumes** *finite* (*snd pair*) **shows** *card* (*omega pair*) = *card* (*snd pair*) **using** *assms mm26 card-cartesian-product-singleton* **by** *metis*

**lemma** *mm30*: **assumes** {} ∉ *Range f runiq f* **shows** *is-partition* (*omega* ‘ *f*)

**proof** −
**let** *?X*=*omega* ‘ *f* **let** *?p*=*finestpart*
{ **fix** *y1 y2* **assume** *y1* ∈ *?X* & *y2* ∈ *?X*
**then obtain** *pair1 pair2* **where**
*0*: *y1* = *omega pair1* & *y2* = *omega pair2* & *pair1* ∈ *f* & *pair2* ∈ *f* **by** *blast*
**then moreover have** *snd pair1* ≠ {} & *snd pair1* ≠ {} **using** *assms*
**by** (*metis rev-image-eqI snd-eq-Range*)
**ultimately moreover have** *fst pair1* = *fst pair2* ⟷ *pair1* = *pair2* **using** *assms*
*runiq-basic surjective-pairing* **by** *metis*
**ultimately moreover have** *y1* ∩ *y2* ≠ {} ⟶ *y1* = *y2* **using** *assms 0* **by** *fast*
**ultimately have** *y1* = *y2* ⟷ *y1* ∩ *y2* ≠ {} **using** *assms mm29*
**by** (*metis Int-absorb Times-empty insert-not-empty*)
}
**thus** *?thesis* **using** *is-partition-def* **by** (*metis* (*lifting*, *no-types*) *inf-commute inf-sup-aci*(*1*))
**qed**

**lemma** *mm32*: **assumes** {} ∉ *Range X* **shows** *inj-on omega X*
**proof** −

**let** *?p=finestpart*
**{**
**fix** *pair1 pair2* **assume** *pair1* $\in$ *X* & *pair2* $\in$ *X* **then have**
*0*: *snd pair1* $\neq$ {} & *snd pair2* $\neq$ {} **using** *assms* **by** (*metis Range.intros surjective-pairing*)
**assume** *omega pair1* = *omega pair2* **then moreover have** *?p* (*snd pair1*) = *?p* (*snd pair2*) **by** *blast*
**then moreover have** *snd pair1* = *snd pair2* **by** (*metis ll64 mm31*)
**ultimately moreover have** {*fst pair1*} = {*fst pair2*} **using** *0 mm29* **by** (*metis fst-image-times*)
**ultimately have** *pair1* = *pair2* **by** (*metis prod-eqI singleton-inject*)
**}**
**thus** *?thesis* **by** (*metis* (*lifting*, *no-types*) *inj-onI*)
**qed**

**lemma** *mm36*: **assumes** {} $\notin$ *Range a*
*finite* (*omega ‘ a*) $\forall$ *X* $\in$ *omega ‘ a*. *finite X is-partition* (*omega ‘ a*)
**shows** *card* (*pseudoAllocation a*) = *setsum* (*card* $\circ$ *omega*) *a* (**is** *?L* = *?R*)
**using** *assms mm33 UniformTieBreaking.mm32 setsum.reindex*
**proof** −
**have** *?L* = *setsum card* (*omega ‘ a*) **using** *assms*(*2*,*3*,*4*) **by** (*rule mm33*)
**moreover have** ... = *?R* **using** *assms*(*1*) *mm32 setsum.reindex* **by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *mm35*: *card* (*omega pair*)= *card* (*snd pair*)
**using** *mm26 card-cartesian-product-singleton* **by** *metis*

**corollary** *mm35b*: *card* $\circ$ *omega* = *card* $\circ$ *snd* **using** *mm35* **by** *fastforce*

**corollary** *mm37*: **assumes** {} $\notin$ *Range a* $\forall$ *pair* $\in$ *a*. *finite* (*snd pair*) *finite a runiq a*
**shows** *card* (*pseudoAllocation a*) = *setsum* (*card* $\circ$ *snd*) *a*
**proof** −
**let** *?P=pseudoAllocation* **let** *?c=card*
**have** $\forall$ *pair* $\in$ *a*. *finite* (*omega pair*) **using** *mm40 assms* **by** *blast* **moreover**
**have** *is-partition* (*omega ‘ a*) **using** *assms mm30* **by** *force* **ultimately**
**have** *?c* (*?P a*) = *setsum* (*?c* $\circ$ *omega*) *a* **using** *assms mm36* **by** *force*
**moreover have** ... = *setsum* (*?c* $\circ$ *snd*) *a* **using** *mm35b* **by** *metis*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**corollary** *mm46*: **assumes**
*runiq* (*a*ˆ−*1*) *runiq a finite a* {} $\notin$ *Range a* $\forall$ *pair* $\in$ *a*. *finite* (*snd pair*) **shows** *card* (*pseudoAllocation a*) = *setsum card* (*Range a*) **using** *assms mm39 mm37* **by** *force*

**corollary** *mm48*: **assumes** *a* $\in$ *possibleAllocationsRel N G finite G* **shows**
*card* (*pseudoAllocation a*) = *card G*

**proof** −
**have** {} ∉ *Range a* **using** *assms mm45b* **by** *blast*
**moreover have** ∀ *pair* ∈ *a. finite* (*snd pair*) **using** *assms mm41 mm47* **by** *metis*
**moreover have** *finite a* **using** *assms mm44* **by** *blast*
**moreover have** *runiq a* **using** *assms* **by** (*metis* (*lifting*) *Int-lower1 in-mono lm19 mem-Collect-eq*)
**moreover have** *runiq* ($a\hat{}-1$) **using** *assms* **by** (*metis* (*mono-tags*) *injections-def lm28b mem-Collect-eq*)
**ultimately have** *card* (*pseudoAllocation a*) = *setsum card* (*Range a*) **using** *mm46* **by** *fast*
**moreover have** ... = *card G* **using** *assms mm33c* **by** *metis*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**corollary** *mm49*: **assumes**
*pseudoAllocation aa* ⊆ *pseudoAllocation a* ∪ (*N* × (*finestpart G*)) *finite* (*pseudoAllocation aa*)
**shows** *setsum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation a*) −
(*setsum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation aa*)) =
*card* (*pseudoAllocation a*) − *card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
**using** *mm28 assms*
**by** *blast*

**corollary** *mm49c*: **assumes**
*pseudoAllocation aa* ⊆ *pseudoAllocation a* ∪ (*N* × (*finestpart G*)) *finite* (*pseudoAllocation aa*)
**shows** *int* (*setsum* (*maxbid′ a N G*) (*pseudoAllocation a*)) −
*int* (*setsum* (*maxbid′ a N G*) (*pseudoAllocation aa*)) =
*int* (*card* (*pseudoAllocation a*)) − *int* (*card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))) **using** *mm28b assms*
**by** *blast*

**lemma** *mm50*: *pseudoAllocation* {} = {} **by** *simp*

**corollary** *mm53b*: **assumes** *a* ∈ *possibleAllocationsRel N* {} **shows** (*pseudoAllocation a*)={}
**using** *assms mm52* **by** *blast*

**corollary** *mm53*: **assumes** *a* ∈ *possibleAllocationsRel N G finite G G* ≠ {}
**shows** *finite* (*pseudoAllocation a*)
**proof** −
**have** *card* (*pseudoAllocation a*) = *card G* **using** *assms*(*1*,*2*) *mm48* **by** *blast*
**thus** *finite* (*pseudoAllocation a*) **using** *assms*(*2*,*3*) **by** *fastforce*
**qed**

**corollary** *mm54*: **assumes** *a* ∈ *possibleAllocationsRel N G finite G* **shows** *finite* (*pseudoAllocation a*) **using** *assms finite.emptyI mm53 mm53b* **by** (*metis* (*no-types*))

**lemma** *mm56*: **assumes** $a \in$ *possibleAllocationsRel N G aa* $\in$ *possibleAllocation-sRel N G finite G* **shows**
(*card* (*pseudoAllocation aa* $\cap$ (*pseudoAllocation a*)) = *card* (*pseudoAllocation a*))
=
(*pseudoAllocation a* = *pseudoAllocation aa*) **using** *assms mm48 mm23b*
**proof** −
**let** *?P=pseudoAllocation* **let** *?c=card* **let** *?A=?P a* **let** *?AA=?P aa*
**have** *?c ?A=?c G* & *?c ?AA=?c G* **using** *assms mm48* **by** (*metis* (*lifting*, *mono-tags*))
**moreover have** *finite ?A* & *finite ?AA* **using** *assms mm54* **by** *blast*
**ultimately show** *?thesis* **using** *assms mm23b* **by** (*metis*(*no-types*,*lifting*))
**qed**

**lemma** *mm55*: *omega pair* = {*fst pair*} × {{*y*}| *y*. *y* $\in$ *snd pair*} **using** *finestpart-def ll64* **by** *auto*

**lemma** *mm55c*: *omega pair* = {(*fst pair*, {*y*})| *y*. *y* $\in$ *snd pair*} **using** *mm55 mm55b* **by** *metis*

**lemma** *mm55d*: *pseudoAllocation a* = $\bigcup$ {{(*fst pair*, {*y*})| *y*. *y* $\in$ *snd pair*}| *pair*. *pair* $\in$ *a*}
**using** *mm55c* **by** *blast*
**lemma** *mm55e*: $\bigcup$ {{(*fst pair*, {*y*})| *y*. *y* $\in$ *snd pair*}| *pair*. *pair* $\in$ *a*}=
{(*fst pair*, {*y*})| *y pair*. *y* $\in$ *snd pair* & *pair* $\in$ *a*} **by** *blast*

**corollary** *mm55k*: *pseudoAllocation a* = {(*fst pair*, *Y*)| *Y pair*. *Y* $\in$ *finestpart* (*snd pair*) & *pair* $\in$ *a*}
**using** *mm55j* **by** *blast*

**lemma** *mm55u*: **assumes** *runiq a* **shows**
{(*fst pair*, *Y*)| *Y pair*. *Y* $\in$ *finestpart* (*snd pair*) & *pair* $\in$ *a*} = {(*x*, *Y*)| *Y x*. *Y* $\in$ *finestpart* (*a*,,*x*) & *x* $\in$ *Domain a*}
(**is** *?L=?R*) **using** *assms Domain.DomainI fst-conv mm60 runiq-wrt-ex1 surjective-pairing*
**by** (*metis*(*hide-lams*,*no-types*))

**corollary** *mm55v*: **assumes** *runiq a* **shows** *pseudoAllocation a* = {(*x*, *Y*)| *Y x*. *Y* $\in$ *finestpart* (*a*,,*x*) & *x* $\in$ *Domain a*}
**using** *assms mm55u mm55k* **by** *fastforce*

**corollary** *mm55t*: *Range* (*pseudoAllocation a*) = $\bigcup$ (*finestpart* ‘ (*Range a*))
**using** *mm55k mm55l mm55m* **by** *fastforce*

**corollary** *mm55s*: *Range* (*pseudoAllocation a*) = *finestpart* ($\bigcup$ *Range a*) **using** *mm55r mm55t* **by** *metis*

**lemma** *mm55f*: *pseudoAllocation a* = {(*fst pair*, {*y*})| *y pair*. *y* $\in$ *snd pair* & *pair* $\in$ *a*} **using** *mm55d*
*mm55e* **by** (*metis* (*full-types*))

**lemma** *mm55g*: $\{(fst\ pair,\ \{y\})|\ y\ pair.\ y \in snd\ pair\ \&\ pair \in a\}=$
$\{(x,\ \{y\})|\ x\ y.\ y \in \bigcup\ (a``\{x\})\ \&\ x \in Domain\ a\}$ **by** *auto*

**lemma** *mm55i*: $pseudoAllocation\ a = \{(x,\ \{y\})|\ x\ y.\ y \in \bigcup\ (a``\{x\})\ \&\ x \in Domain\ a\}$ (**is** *?L=?R*)
**proof** −
**have** $?L=\{(fst\ pair,\ \{y\})|\ y\ pair.\ y \in snd\ pair\ \&\ pair \in a\}$ **by** (*rule mm55f*)
**moreover have** ... = *?R* **by** (*rule mm55g*) **ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *mm62*: *runiq* (*LinearCompletion bids N G*) **using** *assms* **by** (*metis graph-def image-Collect-mem ll37*)
**corollary** *mm62b*: *runiq* (*LinearCompletion bids N G* || *a*)
**unfolding** *restrict-def* **using** *mm62 subrel-runiq Int-commute* **by** *blast*
**lemma** *mm64*: $N \times (Pow\ G - \{\{\}\}) = Domain\ (LinearCompletion\ bids\ N\ G)$
**by** *blast*

**corollary** *mm63d*: **assumes** $a \in possibleAllocationsRel\ N\ G$ **shows** $a \subseteq Domain$ (*LinearCompletion bids N G*)
**proof** −
**let** *?p=possibleAllocationsRel* **let** *?L=LinearCompletion*
**have** $a \subseteq N \times (Pow\ G - \{\{\}\})$ **using** *assms mm63c* **by** *metis*
**moreover have** $N \times (Pow\ G - \{\{\}\})=Domain\ (?L\ bids\ N\ G)$ **using** *mm64* **by** *blast*
**ultimately show** *?thesis* **by** *blast*
**qed**

**corollary** *mm59d*: *setsum* (*linearCompletion′ bids N G*) ($a \cap$ (*Domain* (*LinearCompletion bids N G*))) =
*setsum snd* ((*LinearCompletion bids N G*) || *a*) **using** *assms mm59c mm62b* **by** *fast*

**corollary** *mm59e*: **assumes** $a \in possibleAllocationsRel\ N\ G$ **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum snd* ((*LinearCompletion bids N G*) || *a*)
**proof** −
**let** *?l=linearCompletion′* **let** *?L=LinearCompletion*
**have** $a \subseteq Domain\ (?L\ bids\ N\ G)$ **using** *assms* **by** (*rule mm63d*) **then**
**have** $a = a \cap Domain\ (?L\ bids\ N\ G)$ **by** *blast* **then**
**have** *setsum* (*?l bids N G*) *a* = *setsum* (*?l bids N G*) ($a \cap Domain$ (*?L bids N G*)) **by** *presburger*
**thus** *?thesis* **using** *mm59d* **by** *auto*
**qed**

**corollary** *mm59f*: **assumes** $a \in possibleAllocationsRel\ N\ G$ **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum snd* ((*partialCompletionOf bids*) ‘ $((N \times (Pow\ G - \{\{\}\})) \cap a)$)
(**is** *?X=?R*)
**proof** −

**let** *?p=partialCompletionOf* **let** *?L=LinearCompletion* **let** *?l=linearCompletion′*
**let** *?A=N* × (*Pow G* − {{}}) **let** *?inner2*=(*?p bids*)‘(*?A* ∩ *a*) **let** *?inner1*=(*?L bids N G*)||*a*
**have** *?R* = *setsum snd ?inner1* **using** *assms mm66d* **by** (*metis* (*no-types*))
**moreover have** *setsum* (*?l bids N G*) *a* = *setsum snd ?inner1* **using** *assms* **by** (*rule mm59e*)
**ultimately show** *?thesis* **by** *presburger*
**qed**
**corollary** *mm59g*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum snd* ((*partialCompletionOf bids*) ‘ *a*) (**is** *?L=?R*)
**using** *assms mm59f mm63c*
**proof** −
**let** *?p=partialCompletionOf* **let** *?l=linearCompletion′*
**have** *?L* = *setsum snd* ((*?p bids*)‘((*N* × (*Pow G* − {{}}))∩ *a*)) **using** *assms* **by** (*rule mm59f*)
**moreover have** ... = *?R* **using** *assms mm63c Int-absorb1* **by** (*metis* (*no-types*))
**ultimately show** *?thesis* **by** *presburger*
**qed**
**corollary** *mm57c*: *setsum snd* ((*partialCompletionOf bids*) ‘ *a*) = *setsum* (*snd* ∘ (*partialCompletionOf bids*)) *a*
**using** *assms setsum.reindex mm57b* **by** *blast*
**corollary** *mm59h*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum* (*snd* ∘ (*partialCompletionOf bids*)) *a* (**is** *?L=?R*)
**using** *assms mm59g mm57c*
**proof** −
**let** *?p=partialCompletionOf* **let** *?l=linearCompletion′*
**have** *?L* = *setsum snd* ((*?p bids*)‘ *a*) **using** *assms* **by** (*rule mm59g*)
**moreover have** ... = *?R* **using** *assms mm57c* **by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**
**corollary** *mm25c*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum* ((*setsum bids*) ∘ *omega*) *a* (**is** *?L=?R*)
**using** *assms mm59h mm25*
**proof** −
**let** *?inner1=snd* ∘ (*partialCompletionOf bids*) **let** *?inner2*=(*setsum bids*) ∘ *omega*
**let** *?M=setsum ?inner1 a*
**have** *?L* = *?M* **using** *assms* **by** (*rule mm59h*)
**moreover have** *?inner1* = *?inner2* **using** *mm25 assms* **by** *fastforce*
**ultimately show** *?L* = *?R* **using** *assms* **by** *metis*
**qed**

**corollary** *mm25d*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows**
*setsum* (*linearCompletion′ bids N G*) *a* = *setsum* (*setsum bids*) (*omega*‘ *a*)
**using** *assms mm25c setsum.reindex mm32*
**proof** −
**have** {} ∉ *Range a* **using** *assms* **by** (*metis mm45b*)

**then have** *inj-on omega a* **using** *mm32* **by** *blast*
**then have** *setsum* (*setsum bids*) (*omega ‘ a*) = *setsum* ((*setsum bids*) ∘ *omega*) *a*

**by** (*rule setsum.reindex*)
**moreover have** *setsum* (*linearCompletion′ bids N G*) *a* = *setsum* ((*setsum bids*) ∘ *omega*) *a*
**using** *assms mm25c* **by** (*rule Extraction.exE-realizer*)
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *mm67*: **assumes** *finite* (*snd pair*) **shows** *finite* (*omega pair*) **using** *assms*

**by** (*metis finite.emptyI finite.insertI finite-SigmaI mm40*)
**corollary** *mm67b*: **assumes** ∀ *y*∈(*Range a*). *finite y* **shows** ∀ *y*∈(*omega ‘ a*). *finite y*
**using** *assms mm67 imageE mm47* **by** *fast*
**lemma assumes** *a* ∈ *possibleAllocationsRel N G finite G* **shows** ∀ *y* ∈ (*Range a*). *finite y*
**using** *assms* **by** (*metis mm41*)

**corollary** *mm67c*: **assumes** *a* ∈ *possibleAllocationsRel N G finite G* **shows** ∀ *x*∈(*omega ‘ a*). *finite x*
**using** *assms mm67b mm41* **by** (*metis*(*no-types*))

**corollary** *mm30b*: **assumes** *a* ∈ *possibleAllocationsRel N G* **shows** *is-partition* (*omega ‘ a*)
**using** *assms mm30 mm45b image-iff lll81a*
**proof** −
  **have** *runiq a* **by** (*metis* (*no-types*) *assms image-iff lll81a*)
  **moreover have** {} ∉ *Range a* **using** *assms mm45b* **by** *blast*
  **ultimately show** *?thesis* **using** *mm30* **by** *blast*
**qed**

**lemma** *mm68*: **assumes** *a* ∈ *possibleAllocationsRel N G finite G* **shows** *setsum* (*setsum bids*) (*omega‘ a*) = *setsum bids* (⋃ (*omega ‘ a*))
**using** *assms setsum-Union-disjoint-2 mm30b mm67c* **by** (*metis* (*lifting*, *mono-tags*))

**corollary** *mm69*: **assumes** *a* ∈ *possibleAllocationsRel N G finite G* **shows** *setsum* (*linearCompletion′ bids N G*) *a* = *setsum bids* (*pseudoAllocation a*) (**is** *?L* = *?R*)
**using** *assms mm25d mm68*
**proof** −
**have** *?L* = *setsum* (*setsum bids*) (*omega ‘a*) **using** *assms mm25d* **by** *blast*
**moreover have** ... = *setsum bids* (⋃ (*omega ‘ a*)) **using** *assms mm68* **by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *mm73*: *Domain* (*pseudoAllocation a*) ⊆ *Domain a* **by** *auto*
**corollary assumes** *a* ∈ *possibleAllocationsRel N G* **shows** ⋃ *Range a* = *G* **using**

*assms lm47*
*is-partition-of-def* **by** *metis*
**corollary** *mm72*: **assumes** $a \in$ *possibleAllocationsRel N G* **shows** *Range* (*pseudoAllocation* *a*) = *finestpart G*
**using** *assms mm55s lm47 is-partition-of-def* **by** *metis*
**corollary** *mm73b*: **assumes** $a \in$ *possibleAllocationsRel N G* **shows** *Domain* (*pseudoAllocation* *a*) $\subseteq$ *N* &
*Range* (*pseudoAllocation a*) = *finestpart G*
**using** *assms mm73 lm47 mm55s is-partition-of-def subset-trans* **by** (*metis*(*no-types*))
**corollary** *mm73c*: **assumes** $a \in$ *possibleAllocationsRel N G*
**shows** *pseudoAllocation a* $\subseteq$ *N* $\times$ *finestpart G* **using** *assms mm73b*
**proof** −
**let** *?p=pseudoAllocation* **let** *?aa=?p a* **let** *?d=Domain* **let** *?r=Range*
**have** *?d ?aa* $\subseteq$ *N* **using** *assms mm73b* **by** (*metis* (*lifting*, *mono-tags*))
**moreover have** *?r ?aa* $\subseteq$ *finestpart G* **using** *assms mm73b* **by** (*metis* (*lifting*, *mono-tags*) *equalityE*)
**ultimately have** *?d ?aa* $\times$ (*?r ?aa*) $\subseteq$ *N* $\times$ *finestpart G* **by** *auto*
**then show** *?aa* $\subseteq$ *N* $\times$ *finestpart G* **by** *auto*
**qed**

**abbreviation** *mbc pseudo* == $\{(x, \bigcup$ (*pseudo* `` $\{x\}$))| *x*. $x \in$ *Domain pseudo*}

**corollary assumes** $\{\} \notin$ *Range X* **shows** *inj-on* (*image omega*) (*Pow X*) **using** *assms mm74 mm32* **by** *blast*

**lemma** *pseudoAllocation* = *Union* $\circ$ (*image omega*) **by** *force*
**lemma** *mm75d*: **assumes** *runiq a* $\{\} \notin$ *Range a* **shows**
*a* = *mbc* (*pseudoAllocation a*)
**proof** –
**let** *?p*=$\{(x, Y)|$ *Y x*. $Y \in$ *finestpart* (*a,,x*) & $x \in$ *Domain a*}
**let** *?a*=$\{(x, \bigcup$ (*?p* `` $\{x\}$))| *x*. $x \in$ *Domain ?p*}
**have** $\forall x \in$ *Domain a*. *a,,x* $\neq \{\}$ **by** (*metis assms ll14*)
**then have** $\forall x \in$ *Domain a*. *finestpart* (*a,,x*) $\neq \{\}$ **by** (*metis mm29*)
**then have** *Domain a* $\subseteq$ *Domain ?p* **by** *force*
**moreover have** *Domain a* $\supseteq$ *Domain ?p* **by** *fast*
**ultimately have**
*1*: *Domain a* = *Domain ?p* **by** *fast*
**{**
**fix** *z* **assume** $z \in$ *?a*
**then obtain** *x* **where**
$x \in$ *Domain ?p* & *z*=(*x* , $\bigcup$ (*?p* `` $\{x\}$)) **by** *blast*
**then have** $x \in$ *Domain a* & *z*=(*x* , $\bigcup$ (*?p* `` $\{x\}$)) **by** *fast*
**then moreover have** *?p*``$\{x\}$ = *finestpart* (*a,,x*) **using** *assms* **by** *fastforce*
**moreover have** $\bigcup$ (*finestpart* (*a,,x*))= *a,,x* **by** (*metis mm75*)
**ultimately have** $z \in a$ **by** (*metis assms*(*1*) *eval-runiq-rel*)
**}**
**then have**
*3*: *?a* $\subseteq$ *a* **by** *fast*
**{**
**fix** *z* **assume** *0*: $z \in a$ **let** *?x*=*fst z* **let** *?Y*=*a,,?x* **let** *?YY*=*finestpart ?Y*
**have** $z \in a$ & *?x* $\in$ *Domain a* **using** *0* **by** (*metis fst-eq-Domain rev-image-eqI*)
**then**
**have**
*2*:$z \in a$ & *?x* $\in$ *Domain ?p* **using** *1* **by** *presburger* **then**
**have** *?p* `` {*?x*} = *?YY* **by** *fastforce*
**then have** $\bigcup$ (*?p* `` {*?x*}) = *?Y* **by** (*metis mm75*)
**moreover have** *z* = (*?x*, *?Y*) **using** *assms* **by** (*metis 0 mm60 surjective-pairing*)
**ultimately have** $z \in$ *?a* **using** *2* **by** (*metis* (*lifting*, *mono-tags*) *mem-Collect-eq*)
**}**
**then have** *a* = *?a* **using** *3* **by** *blast*
**moreover have** *?p* = *pseudoAllocation a* **using** *mm55v assms* **by** (*metis* (*lifting*, *mono-tags*))
**ultimately show** *?thesis* **by** *auto*
**qed**
**corollary** *mm75dd*: **assumes** $a \in$ *runiqs* $\cap$ *Pow* (*UNIV* $\times$ (*UNIV* $-$ $\{\{\}\}$)))

**shows**
(*mbc* ∘ *pseudoAllocation*) *a* = *id a* **using** *assms mm75d*
**proof** −
**have** *runiq a* **using** *runiqs-def assms* **by** *fast*
**moreover have** {} ∉ *Range a* **using** *assms* **by** *blast*
**ultimately show** *?thesis* **using** *mm75d* **by** *fastforce*
**qed**
**lemma** *mm75e*: *inj-on* (*mbc* ∘ *pseudoAllocation*) (*runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}})))
**using** *assms mm75dd inj-on-def inj-on-id*
**proof** −
**let** *?ne=Pow* (*UNIV* × (*UNIV* − {{}})) **let** *?X=runiqs* ∩ *?ne* **let** *?f=mbc* ∘ *pseudoAllocation*
**have** ∀ *a1* ∈ *?X*. ∀ *a2* ∈ *?X*. *?f a1* = *?f a2* ⟶ *id a1* = *id a2* **using** *mm75dd* **by** *blast* **then**
**have** ∀ *a1* ∈ *?X*. ∀ *a2* ∈ *?X*. *?f a1* = *?f a2* ⟶ *a1* = *a2* **by** *auto*
**thus** *?thesis* **unfolding** *inj-on-def* **by** *blast*
**qed**

**corollary** *mm75g*: *inj-on pseudoAllocation* (*runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}})))
**using** *mm75e inj-on-imageI2* **by** *blast*
**lemma** *mm76*: *injectionsUniverse* ⊆ *runiqs* **using** *runiqs-def Collect-conj-eq Int-lower1* **by** *metis*
**lemma** *mm77*: *partitionValuedUniverse* ⊆ *Pow* (*UNIV* × (*UNIV* − {{}})) **using** *assms is-partition-def* **by** *force*
**corollary** *mm75i*: *allocationsUniverse* ⊆ *runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}}))
**using** *mm76 mm77* **by** *auto*
**corollary** *mm75h*: *inj-on pseudoAllocation allocationsUniverse* **using** *assms mm75g mm75i subset-inj-on* **by** *blast*
**corollary** *mm75j*: *inj-on pseudoAllocation* (*possibleAllocationsRel N G*)
**proof** −
  **have** *possibleAllocationsRel N G* ⊆ *allocationsUniverse* **by** (*metis* (*no-types*) *lm50*)
 **thus** *inj-on pseudoAllocation* (*possibleAllocationsRel N G*) **using** *mm75h subset-inj-on* **by** *blast*
**qed**

**fun** *prova* **where** *prova f X 0 = X | prova f X (Suc n) = f n (prova f X n)*

**fun** *prova2* **where** *prova2 f 0 = UNIV |prova2 f (Suc n) = f n (prova2 f n)*

**fun** *geniter* **where** *geniter f 0 = f 0 | geniter f (Suc n)=(f (Suc n)) o (geniter f n)*

**abbreviation** *pseudodecreasing X Y == card X − 1 ≤ card Y − 2*

**notation** *pseudodecreasing* (**infix** <~ *75*)

**abbreviation** *subList l xl == map (nth l) (takeAll (%x. x ≤ size l) xl)*
**abbreviation** *insertRightOf2 x l n == (subList l (map nat [0..n])) @ [x] @ (subList l (map nat [n+1..size l − 1]))*
**abbreviation** *insertRightOf3 x l n == insertRightOf2 x l (Min {n, size l − 1})*
**definition** *insertRightOf x l n = sublist l {0..<1+n} @ [x] @ sublist l {n+1..< 1+size l}*
**lemma** *set (insertRightOf x l n) = set (sublist l {0..<1+n}) ∪ (set [x]) ∪ set (sublist l {n+1..<1+size l})* **using** *insertRightOf-def*
**by** (*metis append-assoc set-append*)
**lemma** *set l1 ∪ set l2 = set (l1 @ l2)* **by** *simp*

**fun** *permOld*::'*a list* => (*nat* × ('*a list*)) *set* **where**
*permOld* [] = {} | *permOld* (*x*#*l*) =
*graph* {*fact* (*size l*) ..< *1*+*fact* (*1* + (*size l*))}
(%*n*::*nat*. *insertRightOf x* (*permOld l*,,(*n div size l*)) (*n mod* (*size l*)))
+* (*permOld l*)

**fun** *permL* **where**
*permL* [] = (%*n*. [])|
*permL* (*x*#*l*) = (%*n*.
*if* (*fact* (*size l*) < *n* & *n* <= *fact* (*1* + (*size l*)))
*then*
(*insertRightOf3 x* (*permL l* (*n div size l*)) (*n mod* (*size l*)))
*else*
(*x* # (*permL l n*))
)

**lemma** *mm94*: *possibleAllocationsAlg2 N G* = *set* (*possibleAllocationsAlg3 N G*)
**by** *auto*
**lemma** *mm95*: **assumes** *card N* > *0 distinct G* **shows**
*winningAllocationsRel N* (*set G*) *bids* ⊆ *set* (*possibleAllocationsAlg3 N G*)
**using** *assms mm94 lm03 lm70b* **by** (*metis*(*no-types*))
**corollary** *mm96*: **assumes**
*N* ≠ {} *finite N distinct G set G* ≠ {} **shows**
*winningAllocationsRel N* (*set G*) *bids* ∩ *set* (*possibleAllocationsAlg3 N G*) ≠ {}
**using** *assms mm91 mm95*
**proof** −
**let** *?w*=*winningAllocationsRel* **let** *?a*=*possibleAllocationsAlg3*
**let** *?G*=*set G* **have** *card N* > *0* **using** *assms* **by** (*metis card-gt-0-iff*)
**then have** *?w N ?G bids* ⊆ *set* (*?a N G*) **using** *mm95* **by** (*metis assms*(*3*))
**then show** *?thesis* **using** *assms mm91* **by** (*metis List.finite-set le-iff-inf*)
**qed**
**lemma** *mm97*: *X* = (%*x*. *x* ∈ *X*) −'{*True*} **by** *blast*
**corollary** *mm96b*: **assumes**
*N* ≠ {} *finite N distinct G set G* ≠ {} **shows**
(%*x*. *x*∈*winningAllocationsRel N* (*set G*) *bids*)−'{*True*} ∩ *set* (*possibleAllocationsAlg3*
*N G*) ≠ {}
**using** *assms mm96 mm97* **by** *metis*
**lemma** *mm84b*: **assumes** *P* −' {*True*} ∩ *set l* ≠ {} **shows** *takeAll P l* ≠ [] **using**
*assms*
*mm84g filterpositions2-def* **by** (*metis Nil-is-map-conv*)
**corollary** *mm84h*: **assumes** *N* ≠ {} *finite N distinct G set G* ≠ {} **shows**
*takeAll* (%*x*. *x* ∈ *winningAllocationsRel N* (*set G*) *bids*) (*possibleAllocationsAlg3*
*N G*) ≠ []
**using** *assms mm84b mm96b* **by** *metis*

**corollary** *nn05b*: **assumes** *N* ≠ {} *finite N distinct G set G* ≠ {} **shows**
*perm2* (*takeAll* (%*x*. *x* ∈ *winningAllocationsRel N* (*set G*) *bids*) (*possibleAllocationsAlg3*
*N G*)) *n* ≠ []

**using** *assms mm83 mm84h* **by** *metis*
**corollary** *mm82*: **assumes** $N \neq \{\}$ *finite N distinct G set* $G \neq \{\}$ **shows** *chosenAllocation' N G bids random* $\in$ *winningAllocationsRel N* (*set G*) *bids*
**using** *assms nn05a nn05b hd-in-set in-mono Int-def Int-lower1 all-not-in-conv image-set nn04 nn06c set-empty subsetI subset-trans*
**proof** −
**let** *?w=winningAllocationsRel* **let** *?p=possibleAllocationsAlg3* **let** *?G=set G*
**let** *?X=?w N ?G bids* **let** *?l=perm2* (*takeAll* (%*x*.(*x*∈*?X*)) (*?p N G*)) *random*
**have** *set ?l* ⊆ *?X* **using** *nn05a* **by** *fast*
**moreover have** *?l* ≠ [] **using** *assms nn05b* **by** *blast*
**ultimately show** *?thesis* **by** (*metis* (*lifting*, *no-types*) *hd-in-set in-mono*)
**qed**

**lemma** *mm49b*: **assumes** *finite G a* ∈ *possibleAllocationsRel N G aa* ∈ *possibleAllocationsRel N G*
**shows** *real*(*setsum*(*maxbid' a N G*)(*pseudoAllocation a*)) − *setsum*(*maxbid' a N G*)(*pseudoAllocation aa*)
= *real* (*card G*) − *card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
**proof** −
**let** *?p=pseudoAllocation* **let** *?f=finestpart* **let** *?m=maxbid'* **let** *?B=?m a N G*
**have**
*2*: *?p aa* ⊆ *N* × *?f G* **using** *assms mm73c* **by** (*metis* (*lifting*, *mono-tags*)) **then have**
*0*: *?p aa* ⊆ *?p a* ∪ (*N* × *?f G*) **by** *auto* **moreover have**
*1*: *finite* (*?p aa*) **using** *assms mm48 mm54* **by** *blast* **ultimately have**
*real*(*setsum ?B* (*?p a*))−*setsum ?B* (*?p aa*) = *real*(*card* (*?p a*))−*card*(*?p aa* ∩ (*?p a*))
**using** *mm28d* **by** *fast*
**moreover have** ... = *real* (*card G*) − *card* (*?p aa* ∩ (*?p a*)) **using** *assms mm48* **by** (*metis* (*lifting*, *mono-tags*))
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *mm66e*: *LinearCompletion bids N G* = *graph* (*N* × (*Pow G*−{{}})) (*test bids*)
**unfolding** *graph-def* **using** *mm66* **by** *blast*
**lemma** *ll33b*: **assumes** *x*∈*X* **shows** *toFunction* (*graph X f*) *x* = *f x* **using** *assms*

**by** (*metis ll33 toFunction-def*)
**corollary** *ll33c*: **assumes** *pair* ∈ *N* × (*Pow G*−{{}}) **shows** *linearCompletion' bids N G pair=test bids pair*
**using** *assms ll33b mm66e* **by** (*metis*(*mono-tags*))

**lemma** *lm031*: *test* (*real* ∘ ((*bids*:: - => *nat*))) *pair* = *real* (*test bids pair*) (**is** *?L=?R*)
**by** *simp*
**lemma** *lm031b*: **assumes** *pair* ∈ *N* × (*Pow G*−{{}}) **shows**
*linearCompletion'* (*real*∘(*bids*:: - => *nat*)) *N G pair* = *real* (*linearCompletion' bids N G pair*)

**using** *assms ll33c lm031* **by** (*metis*(*no-types*))
**corollary** *lm031c*: **assumes** $X \subseteq N \times (Pow\ G - \{\{\}\})$ **shows** $\forall\, pair \in X.$
*linearCompletion′* (*real* ∘ (*bids*::-=>*nat*)) *N G pair*= (*real* ∘ (*linearCompletion′*
*bids N G*)) *pair*
**using** *assms lm031b*
**proof** −
{ **fix** $esk48_0$ :: ′*a* × ′*b set*
{ **assume** $esk48_0 \in N \times (Pow\ G - \{\{\}\})$
**hence** *linearCompletion′* (*real* ∘ *bids*) *N G* $esk48_0$ = *real* (*linearCompletion′*
*bids N G* $esk48_0$) **using** *lm031b* **by** *blast*
**hence** $esk48_0 \notin X$ ∨ *linearCompletion′* (*real* ∘ *bids*) *N G* $esk48_0$ = (*real* ∘
*linearCompletion′ bids N G*) $esk48_0$ **by** *simp* }
**hence** $esk48_0 \notin X$ ∨ *linearCompletion′* (*real* ∘ *bids*) *N G* $esk48_0$ = (*real* ∘
*linearCompletion′ bids N G*) $esk48_0$ **using** *assms* **by** *blast* }
**thus** ∀ *pair*∈*X*. *linearCompletion′* (*real* ∘ *bids*) *N G pair* = (*real* ∘ *linearCom-
pletion′ bids N G*) *pair* **by** *blast*
**qed**

**corollary** *lm031e*: **assumes** $aa \subseteq N \times (Pow\ G - \{\{\}\})$ **shows**
*setsum* ((*linearCompletion′* (*real* ∘ (*bids*::-=>*nat*)) *N G*)) *aa* = *real* (*setsum* ((*linearCompletion′*
*bids N G*)) *aa*)
(**is** *?L*=*?R*)
**proof** −
**have** ∀ *pair* ∈ *aa*. *linearCompletion′* (*real* ∘ *bids*) *N G pair* = (*real* ∘ (*linearCompletion′*
*bids N G*)) *pair*
**using** *assms* **by** (*rule lm031c*)
**then have** *?L* = *setsum* (*real*∘(*linearCompletion′ bids N G*)) *aa* **using** *setsum.cong*
**by** *force*
**then show** *?thesis* **by** *simp*
**qed**

**corollary** *lm031d*: **assumes** *aa* ∈ *possibleAllocationsRel N G* **shows**
*setsum* ((*linearCompletion′* (*real* ∘ (*bids*::-=>*nat*)) *N G*)) *aa* = *real* (*setsum* ((*linearCompletion′*
*bids N G*)) *aa*)
**using** *assms lm031e mm63c* **by** (*metis*(*lifting*,*mono-tags*))

**corollary** *mm70b*:
**assumes** *finite G a* ∈ *possibleAllocationsRel N G aa* ∈ *possibleAllocationsRel N G*
**shows**
*real* (*setsum* (*tiebids′ a N G*) *a*) − *setsum* (*tiebids′ a N G*) *aa* =
*real* (*card G*) − *card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*)) (**is** *?L*=*?R*)
**proof** −
**let** *?l*=*linearCompletion′* **let** *?m*=*maxbid′* **let** *?s*=*setsum* **let** *?p*=*pseudoAllocation*
**let** *?bb*=*?m a N G* **let** *?b*=*real* ∘ (*?m a N G*)
**have** *real* (*?s ?bb* (*?p a*)) − (*?s ?bb* (*?p aa*)) = *?R* **using** *assms mm49b* **by** *blast*

**then have** *?R* = *real* (*?s ?bb* (*?p a*)) − (*?s ?bb* (*?p aa*)) **by** *presburger*
**have** *?s* (*?l ?b N G*) *aa* = *?s ?b* (*?p aa*) **using** *assms mm69* **by** *blast* **moreover**
**have**

... = *?s ?bb* (*?p aa*) **by** *fastforce*
**moreover have** (*?s* (*?l ?b N G*) *aa*) = *real* (*?s* (*?l ?bb N G*) *aa*) **using** *assms*(*3*)
**by** (*rule lm031d*)

**ultimately have**
*1*: *?R* = *real* (*?s ?bb* (*?p a*)) − (*?s* (*?l ?bb N G*) *aa*)
**by** (*metis* ‹*real* (*card G*) − *real* (*card* (*pseudoAllocation aa* ∩ *pseudoAllocation a*)) = *real* (*setsum* (*pseudoAllocation a* <| (*N* × *finestpart G*)) (*pseudoAllocation a*)) − *real* (*setsum* (*pseudoAllocation a* <| (*N* × *finestpart G*)) (*pseudoAllocation aa*))›)
**have** *?s* (*?l ?b N G*) *a*=(*?s ?b* (*?p a*)) **using** *assms mm69* **by** *blast*
**moreover have** ... = *?s ?bb* (*?p a*) **by** *force*
**moreover have** ... = *real* (*?s ?bb* (*?p a*)) **by** *fast*
**moreover have** *?s* (*?l ?b N G*) *a* = *real* (*?s* (*?l ?bb N G*) *a*) **using** *assms*(*2*)
**by** (*rule lm031d*)
**ultimately have** *?s* (*?l ?bb N G*) *a* = *real* (*?s ?bb* (*?p a*)) **try0**
**by** *presburger*
**thus** *?thesis* **using** *1* **by** *presburger*
**qed**

**corollary** *mm70c*: **assumes** *finite G a* ∈ *possibleAllocationsRel N G aa* ∈ *possibleAllocationsRel N G*
*x*=*real* (*setsum* (*tiebids′ a N G*) *a*) − *setsum* (*tiebids′ a N G*) *aa* **shows**
*x* <= *card G* & *x* ≥ *0* & (*x*=*0* ⟷ *a* = *aa*) & (*aa* ≠ *a* ⟶ *setsum* (*tiebids′ a N G*) *aa* < *setsum* (*tiebids′ a N G*) *a*)
**proof** −
**let** *?p*=*pseudoAllocation* **have** *real* (*card G*) >= *real* (*card G*) − *card* (*?p aa* ∩ (*?p a*)) **by** *force*
**moreover have**
*real* (*setsum* (*tiebids′ a N G*) *a*) − *setsum* (*tiebids′ a N G*) *aa* =
*real* (*card G*) − *card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
**using** *assms mm70b* **by** *blast* **ultimately have**
*4*: *x*=*real*(*card G*)−*card*(*pseudoAllocation aa*∩(*pseudoAllocation a*)) **using** *assms*
**by** *force* **then have**
*1*: *x* ≤ *real* (*card G*) **using** *assms* **by** *linarith* **have**
*0*: *card* (*?p aa*) = *card G* & *card* (*?p a*) = *card G* **using** *assms mm48* **by** *blast*
**moreover have** *finite* (*?p aa*) & *finite* (*?p a*) **using** *assms mm54* **by** *blast* **ultimately have** *card* (*?p aa* ∩ *?p a*) ≤ *card G* **using** *Int-lower2 card-mono* **by** *fastforce* **then have**
*2*: *x* ≥ *0* **using** *assms mm70b 4* **by** *linarith*
**have** *card* (*?p aa* ∩ (*?p a*)) = *card G* ⟷ (*?p aa* = *?p a*)
**using** *0 mm56 4 assms* **by** (*metis* (*lifting*, *mono-tags*))
**moreover have** *?p aa* = *?p a* ⟶ *a* = *aa* **using** *assms mm75j inj-on-def*
**by** (*metis* (*lifting*, *mono-tags*))
**ultimately have** *card* (*?p aa* ∩ (*?p a*)) = *card G* ⟷ (*a*=*aa*) **by** *blast*
**moreover have** *x* = *real* (*card G*) − *card* (*?p aa* ∩ (*?p a*)) **using** *assms mm70b*
**by** *blast*
**ultimately have**
*3*: *x* = *0* ⟷ (*a*=*aa*) **by** *linarith* **then have**

*aa* ≠ *a* ⟶ *setsum* (*tiebids′ a N G*) *aa* < *real* (*setsum* (*tiebids′ a N G*) *a*) **using** *1 2 assms*
**by** *auto*
**thus** *?thesis* **using** *1 2 3* **by** *force*
**qed**

**corollary** *mm70d*: **assumes** *finite G a* ∈ *possibleAllocationsRel N G aa* ∈ *possibleAllocationsRel N G*
*aa* ≠ *a* **shows** *setsum* (*tiebids′ a N G*) *aa* < *setsum* (*tiebids′ a N G*) *a* **using** *assms mm70c* **by** *blast*

**lemma** *mm81*: **assumes**
*N* ≠ {} *finite N distinct G set G* ≠ {}
*aa* ∈ (*possibleAllocationsRel N* (*set G*))−{*chosenAllocation′ N G bids random*}
**shows**
*setsum* (*resolvingBid′ N G bids random*) *aa* < *setsum* (*resolvingBid′ N G bids random*) (*chosenAllocation′ N G bids random*)
**proof** −
**let** *?a=chosenAllocation′ N G bids random* **let** *?p=possibleAllocationsRel* **let** *?G=set G*
**have** *?a* ∈ *winningAllocationsRel N* (*set G*) *bids* **using** *assms mm82* **by** *blast*
**moreover have** *winningAllocationsRel N* (*set G*) *bids* ⊆ *?p N ?G* **using** *assms lm03* **by** *metis*
**ultimately have** *?a* ∈ *?p N ?G* **using** *mm82 assms lm03 set-rev-mp* **by** *blast*
**then show** *?thesis* **using** *assms mm70d* **by** *blast*
**qed**

**abbreviation** *terminatingAuctionRel N G bids random* ==
*argmax* (*setsum* (*resolvingBid′ N G bids random*)) (*argmax* (*setsum bids*) (*possibleAllocationsRel N* (*set G*)))

Termination theorem: it assures that the number of winning allocations is exactly one

**theorem** *mm92*: **assumes**
*N* ≠ {} *distinct G set G* ≠ {} *finite N*
**shows** *terminatingAuctionRel N G* (*bids*) *random* = {*chosenAllocation′ N G bids random*}
**proof** −
**let** *?p=possibleAllocationsRel* **let** *?G=set G*
**let** *?X=argmax* (*setsum bids*) (*?p N ?G*)
**let** *?a=chosenAllocation′ N G bids random* **let** *?b=resolvingBid′ N G bids random*
**let** *?f=setsum ?b* **let** *?ff=setsum ?b*
**let** *?t=terminatingAuctionRel* **have** ∀ *aa*∈(*possibleAllocationsRel N ?G*)−{*?a*}. *?f aa* < *?f ?a*
**using** *assms mm81* **by** *blast* **then have**
*0*: ∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a* **using** *assms mm81* **by** *auto*
**have** *finite N* **using** *assms* **by** *simp* **then**
**have** *finite* (*?p N ?G*) **using** *assms lm59* **by** (*metis List.finite-set*)
**then have** *finite ?X* **using** *assms* **by** (*metis finite-subset lm03*)

**moreover have** *?a* ∈ *?X* **using** *mm82 assms* **by** *blast*
**ultimately have**
*finite ?X* & *?a* ∈ *?X* & (∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a*) **using** *0* **by** *force*
**moreover have** (*finite ?X* & *?a* ∈ *?X* & (∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a*)) ⟶ *argmax ?f ?X* = {*?a*}
**by** (*rule mm80c*)
**ultimately have** {*?a*} = *argmax ?f ?X* **using** *mm80* **by** *presburger*
**moreover have** ... = *?t N G bids random* **by** *simp*
**ultimately show** *?thesis* **by** *presburger*
**qed**

A more computable adaptor from set-theoretical to HOL function, with fallback value

**abbreviation** *toFunctionWithFallback2 R fallback* == (% *x*. *if* (*x* ∈ *Domain R*) *then* (*R*,,*x*) *else fallback*)
**notation** *toFunctionWithFallback2* (**infix** *Elsee 75*)

# 38 Combinatorial auction input examples

**abbreviation** *N00* == {*1*,*2*::*nat*}
**abbreviation** *G00* == [*11*::*nat*, *12*, *13*]
**abbreviation** *A00* == {(*0*,{*10*,*11*::*nat*}), (*1*,{*12*,*13*})}
**abbreviation** *b00* ==
{
((*1*::*int*,{*11*}),*3*),
((*1*,{*12*}),*0*),
((*1*,{*11*,*12*::*nat*}),*4*::*price*),
((*2*,{*11*}),*2*),
((*2*,{*12*}),*2*),
((*2*,{*11*,*12*}),*1*)
}

**end**

# 39 VCG auction: definitions and theorems

**theory** *CombinatorialAuction*

**imports**

*UniformTieBreaking*
*StrictCombinatorialAuction*
*~~/src/HOL/Library/Code-Target-Nat*

**begin**

# 40 Definition of a VCG auction scheme, through the pair $(vcga', vcgp')$

**type-synonym** *bidvector'* = ((*participant* × *goods*) × *price*) *set*
**abbreviation** *Participants b'* == *Domain* (*Domain b'*)
**abbreviation** *addedBidder'* == (−*1*::*int*)
**abbreviation** *allStrictAllocations' N G* == *possibleAllocationsRel N G*
**abbreviation** *allStrictAllocations'' N* Ω == *injectionsUniverse* ∩
{*a*. *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions* Ω}
**abbreviation** *allStrictAllocations''' N G* == *allocationsUniverse*∩{*a*. *Domain a* ⊆ *N* & ⋃ *Range a* = *G*}
**lemma** *lm28*: *allStrictAllocations' N G*=*allStrictAllocations'' N G* &
*allStrictAllocations' N G*=*allStrictAllocations''' N G* **using** *lm19 nn24* **by** *metis*
**lemma** *lm28b*: (*a* ∈ *allStrictAllocations''' N G*) = (*a* ∈ *allocationsUniverse*& *Domain a* ⊆ *N* & ⋃ *Range a* = *G*)
**by** *force*
**abbreviation** *allAllocations' N* Ω ==
(*Outside'* {*addedBidder'*}) ‘ (*allStrictAllocations'* (*N* ∪ {*addedBidder'*}) Ω)
**abbreviation** *allAllocations'' N* Ω ==
(*Outside'* {*addedBidder'*}) ‘ (*allStrictAllocations''* (*N* ∪ {*addedBidder'*}) Ω)
**abbreviation** *allAllocations''' N* Ω ==
(*Outside'* {*addedBidder'*}) ‘ (*allStrictAllocations'''* (*N* ∪ {*addedBidder'*}) Ω)
**lemma** *lm28c*:
*allAllocations' N G* = *allAllocations'' N G* & *allAllocations'' N G* = *allAllocations''' N G*
**using** *assms lm28* **by** *metis*
**corollary** *lm28d*: *allAllocations'* = *allAllocations''* & *allAllocations''* = *allAllocations'''*
& *allAllocations'* = *allAllocations'''* **using** *lm28c* **by** *metis*
**lemma** *lm32*: *allAllocations'* (*N*−{*addedBidder'*}) *G* ⊆ *allAllocations' N G* **using** *Outside-def* **by** *simp*

**lemma** *lm34*: (*a* ∈ *allocationsUniverse*) = (*a* ∈ *allStrictAllocations'''* (*Domain a*) (⋃ *Range a*))
**by** *blast*
**lemma** *lm35*: **assumes** *N1* ⊆ *N2* **shows** *allStrictAllocations''' N1 G* ⊆ *allStrictAllocations''' N2 G*
**using** *assms* **by** *auto*
**lemma** *lm36*: **assumes** *a* ∈ *allStrictAllocations''' N G* **shows** *Domain* (*a* −− *x*) ⊆ *N*−{*x*}
**using** *assms Outside-def* **by** *fastforce*
**lemma** *lm37*: **assumes** *a* ∈ *allAllocations' N G* **shows** *a* ∈ *allocationsUniverse*
**proof** −
**obtain** *aa* **where** *a*=*aa* −− *addedBidder'* & *aa* ∈ *allStrictAllocations'* (*N*∪{*addedBidder'*}) *G*
**using** *assms* **by** *blast*
**then have** *a* ⊆ *aa* & *aa* ∈ *allocationsUniverse* **unfolding** *Outside-def* **using** *nn24b* **by** *blast*
**then show** *?thesis* **using** *lm35b* **by** *blast*

**qed**
**lemma** *lm38*: **assumes** $a \in$ *allAllocations'* $N$ $G$ **shows** $a \in$ *allStrictAllocations'''* (*Domain a*) ($\bigcup$ *Range a*)
**proof** − **show** *?thesis* **using** *assms lm37* **by** *blast* **qed**
**lemma** **assumes** $N1 \subseteq N2$ **shows** *allAllocations'''* $N1$ $G \subseteq$ *allAllocations'''* $N2$ $G$
**using** *assms lm35 lm36 nn24c lm28b lm28 lm34 lm38 Outside-def* **by** *blast*

**lemma** *lll59b*: *runiq* $(X \times \{y\})$ **using** *lll59* **by** (*metis trivial-singleton*)
**lemma** *lm37b*: $\{x\} \times \{y\} \in$ *injectionsUniverse* **using** *Universes.lm37* **by** *fastforce*
**lemma** *lm40b*: **assumes** $a \in$ *allAllocations'''* $N$ $G$ **shows** $\bigcup$ *Range* $a \subseteq G$ **using** *assms Outside-def* **by** *blast*
**lemma** *lm41*: $a \in$ *allAllocations'''* $N$ $G$ =
(*EX aa. aa* −− (*addedBidder'*)=*a* & *aa*∈*allStrictAllocations'''* ($N \cup \{$*addedBidder'*$\}$) *G*) **by** *blast*

**lemma** *lm18*: $(R +* (\{x\} \times Y))$ −− $x = R$ −− $x$ **unfolding** *Outside-def paste-def* **by** *blast*

**lemma** *lm37e*: **assumes** $a \in$ *allocationsUniverse Domain* $a \subseteq N - \{$*addedBidder'*$\}$ $\bigcup$ *Range* $a \subseteq G$ **shows**
$a \in$ *allAllocations'''* $N$ $G$ **using** *assms lm41*
**proof** −
**let** *?i*=*addedBidder'* **let** *?Y*=$\{G - \bigcup$ *Range a*$\} - \{\{\}\}$ **let** *?b*=$\{$*?i*$\} \times$ *?Y* **let** *?aa*=*a*∪*?b*
**let** *?aa'*=*a* +∗ *?b*
**have**
*1*: $a \in$ *allocationsUniverse* **using** *assms*(*1*) **by** *fast*
**have** *?b* ⊆ {(*?i*,*G*−⋃ *Range a*)} − {(*?i*, {})} **by** *fastforce* **then have**
*2*: *?b* ∈ *allocationsUniverse* **using** *Universes.lm38 lm35b* **by** (*metis*(*no-types*))
**have**
*3*: ⋃ *Range a* ∩ ⋃ (*Range ?b*) = {} **by** *blast* **have**
*4*: *Domain a* ∩ *Domain ?b* ={} **using** *assms* **by** *fast*
**have** *?aa* ∈ *allocationsUniverse* **using** *1 2 3 4* **by** (*rule lm23*)
**then have** *?aa* ∈ *allStrictAllocations'''* (*Domain ?aa*)
(⋃ *Range ?aa*) **unfolding** *lm34* **by** *metis* **then have**
*?aa* ∈ *allStrictAllocations'''* (*N*∪{*?i*}) (⋃ *Range ?aa*) **using** *lm35 assms paste-def* **by** *auto*
**moreover have** *Range ?aa* = *Range a* ∪ *?Y* **by** *blast* **then moreover have**
⋃ *Range ?aa* = *G* **using** *Un-Diff-cancel Un-Diff-cancel2 Union-Un-distrib Union-empty Union-insert*
**by** (*metis* (*lifting*, *no-types*) *assms*(*3*) *cSup-singleton subset-Un-eq*) **moreover have**
*?aa'* = *?aa* **using** *4* **by** (*rule paste-disj-domains*)
**ultimately have** *?aa'* ∈ *allStrictAllocations'''* (*N*∪{*?i*}) *G* **by** *simp*
**moreover have** *Domain ?b* ⊆ {*?i*} **by** *fast*
**have** *?aa'* −− *?i* = *a* −− *?i* **by** (*rule lm18*)
**moreover have** ... = *a* **using** *Outside-def assms*(*2*) **by** *auto*
**ultimately show** *?thesis* **using** *lm41* **by** *auto*
**qed**

**lemma** *lm23*:
*a*∈*allStrictAllocations' N* Ω=(*a*∈*injectionsUniverse* & *Domain a*⊆*N* & *Range a*∈*all-partitions* Ω)
**by** (*metis* (*full-types*) *lm19c*)

**lemma** *lm37n*: **assumes** *a* ∈ *allAllocations''' N G* **shows** *Domain a* ⊆ *N*−{*addedBidder'*} & *a* ∈ *allocationsUniverse*
**proof** −
**let** *?i*=*addedBidder'* **obtain** *aa* **where**
*0*: *a*=*aa* −− *?i* & *aa* ∈ *allStrictAllocations'''* (*N* ∪ {*?i*}) *G* **using** *assms*(*1*) *lm41*
**by** *blast*
**then have** *Domain aa* ⊆ *N* ∪ {*?i*} **using** *lm23* **by** *blast*
**then have** *Domain a* ⊆ *N* − {*?i*} **using** *0 Outside-def* **by** *blast*
**moreover have** *a* ∈ *allAllocations' N G* **using** *assms lm28d* **by** *metis*
**then moreover have** *a* ∈ *allocationsUniverse* **using** *lm37* **by** *blast*
**ultimately show** *?thesis* **by** *blast*
**qed**

**corollary** *lm37c*: **assumes** *a* ∈ *allAllocations''' N G* **shows**
*a* ∈ *allocationsUniverse* & *Domain a* ⊆ *N*−{*addedBidder'*} & ⋃ *Range a* ⊆ *G*
**proof** −
**have** *a* ∈ *allocationsUniverse* **using** *assms lm37n* **by** *blast*
**moreover have** *Domain a* ⊆ *N*−{*addedBidder'*} **using** *assms lm37n* **by** *blast*
**moreover have** ⋃ *Range a* ⊆ *G* **using** *assms lm40b* **by** *blast*
**ultimately show** *?thesis* **by** *blast*
**qed**

**corollary** *lm37d*:
(*a*∈*allAllocations''' N G*)=(*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*addedBidder'*} & ⋃ *Range a* ⊆ *G*)
**using** *lm37c lm37e* **by** (*metis* (*mono-tags*))

**lemma** *lm42*: (*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*addedBidder'*} & ⋃ *Range a* ⊆ *G*) =
(*a*∈*allocationsUniverse*& *a*∈{*aa*. *Domain aa* ⊆ *N*−{*addedBidder'*} & ⋃ *Range aa* ⊆ *G*})
**by** (*metis* (*lifting*, *no-types*) *mem-Collect-eq*)

**corollary** *lm37f*: (*a*∈*allAllocations''' N G*)=
(*a*∈*allocationsUniverse*& *a*∈{*aa*. *Domain aa* ⊆ *N*−{*addedBidder'*} & ⋃ *Range aa* ⊆ *G*}) (**is** *?L* = *?R*)
**proof** −
**have** *?L* = (*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*addedBidder'*} & ⋃ *Range a* ⊆ *G*) **by** (*rule lm37d*)
**moreover have** ... = *?R* **by** (*rule lm42*) **ultimately show** *?thesis* **by** *presburger*
**qed**

**corollary** *lm37g*: *a*∈*allAllocations''' N G*=

($a$∈ (*allocationsUniverse* ∩ {*aa*. *Domain aa* ⊆ *N*−{*addedBidder'*} & ⋃ *Range aa* ⊆ *G*}))
**using** *lm37f* **by** (*metis* (*mono-tags*) *Int-iff*)

**abbreviation** *allAllocations''''* *N G* ==
*allocationsUniverse*∩{*aa*. *Domain aa*⊆*N*−{*addedBidder'*} & ⋃*Range aa*⊆*G*}

**lemma** *lm44*: **assumes** *a* ∈ *allAllocations''''* *N G* **shows** *a* −− *n* ∈ *allAllocations''''* (*N*−{*n*}) *G*
**proof** −
let *?bb*=*addedBidder'* **let** *?d*=*Domain* **let** *?r*=*Range* **let** *?X2*={*aa*. *?d aa*⊆*N*−{*?bb*} & ⋃ *?r aa*⊆*G*}
**let** *?X1*={*aa*. *?d aa*⊆*N*−{*n*}−{*?bb*} & ⋃ *?r aa*⊆*G*}
**have** *a*∈*?X2* **using** *assms*(*1*) **by** *fast* **then have**
*0*: *?d a* ⊆ *N*−{*?bb*} & ⋃ *?r a* ⊆ *G* **by** *blast* **then have** *?d* (*a*−−*n*) ⊆ *N*−{*?bb*}−{*n*}

**using** *outside-reduces-domain* **by** (*metis Diff-mono subset-refl*) **moreover have**

... = *N*−{*n*}−{*?bb*} **by** *fastforce* **ultimately have**
*?d* (*a*−−*n*) ⊆ *N*−{*n*}−{*?bb*} **by** *blast* **moreover have** ⋃ *?r* (*a*−−*n*) ⊆ *G*
**unfolding** *Outside-def* **using** *0* **by** *blast* **ultimately have** *a* −− *n* ∈ *?X1* **by** *fast* **moreover have**
*a*−−*n* ∈ *allocationsUniverse* **using** *assms*(*1*) *Int-iff lm35d* **by** (*metis*(*lifting*,*mono-tags*))

**ultimately show** *?thesis* **by** *blast*
**qed**

**corollary** *lm37h*: *allAllocations'''* *N G*=*allAllocations''''* *N G*
(**is** *?L*=*?R*) **proof** − {**fix** *a* **have** *a* ∈ *?L* = (*a* ∈ *?R*) **by** (*rule lm37g*)} **thus** *?thesis* **by** *blast* **qed**

**lemma** *lm28e*: *allAllocations'*=*allAllocations''* & *allAllocations''*=*allAllocations'''* &
*allAllocations'''*=*allAllocations''''* **using** *lm37h lm28d* **by** *metis*

**corollary** *lm44b*: **assumes** *a* ∈ *allAllocations'* *N G* **shows** *a* −− *n* ∈ *allAllocations'* (*N*−{*n*}) *G*

**proof** −
**let** *?A'*=*allAllocations''''* **have** *a* ∈ *?A'* *N G* **using** *assms lm28e* **by** *metis* **then have** *a* −− *n* ∈ *?A'* (*N*−{*n*}) *G* **by** (*rule lm44*) **thus** *?thesis* **using** *lm28e* **by** *metis*
**qed**

**corollary** *lm37i*: **assumes** *G1* ⊆ *G2* **shows** *allAllocations''''* *N G1* ⊆ *allAllocations''''* *N G2*
**using** *assms* **by** *blast*

**corollary** *lm33*: **assumes** *G1* ⊆ *G2* **shows** *allAllocations'''* *N G1* ⊆ *allAlloca-*

*tions''' N G2*
**using** *assms lm37i lm37h*
**proof** –
**have** *allAllocations''' N G1* = *allAllocations'''' N G1* **by** (*rule lm37h*)
**moreover have** ... ⊆ *allAllocations'''' N G2* **using** *assms*(*1*) **by** (*rule lm37i*)
**moreover have** ... = *allAllocations''' N G2* **using** *lm37h* **by** *metis*
**ultimately show** *?thesis* **by** *auto*
**qed**

**abbreviation** *maximalAllocations'' N Ω b* == *argmax* (*setsum b*) (*allAllocations' N Ω*)

**abbreviation** *maximalStrictAllocations' N G b*==
*argmax* (*setsum b*) (*allStrictAllocations'* ({*addedBidder'*}∪*N*) *G*)

**corollary** *lm43d*: **assumes** $a \in$ *allocationsUniverse* **shows**
($a$ *outside* $(X\cup\{i\})) \cup (\{i\}\times(\{\bigcup(a``(X\cup\{i\}))\}-\{\{\}\})) \in$ *allocationsUniverse* **using** *assms lm43b*
**by** *fastforce*

**lemma** *lm27c*: *addedBidder'* ∉ *int* '*N* **by** *fastforce*

**abbreviation** *randomBids' N Ω b random*==*resolvingBid'* (*N*∪{*addedBidder'*}) *Ω b random*

Here we are showing that our way of randomizing using randomBids' actually breaks the tie: we are left with a singleton after the tiebreaking step.

**theorem** *mm92b*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*card* (*argmax* (*setsum* (*randomBids' N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*))=*1*
(**is** *card ?L*=-) **proof** –
**let** *?n*={*addedBidder'*} **have**
*1*: (*?n* ∪ *N*)≠{} **by** *simp* **have**
*4*: *finite* (*?n*∪*N*) **using** *assms*(*3*) **by** *fast* **have**
*terminatingAuctionRel* (*?n*∪*N*) *G b r* = {*chosenAllocation'* (*?n*∪*N*) *G b r*} **using** *1 assms*(*1*)
*assms*(*2*) *4* **by** (*rule mm92*) **moreover have** *?L* = *terminatingAuctionRel* (*?n*∪*N*) *G b r* **by** *auto*
**ultimately show** *?thesis* **by** *auto*
**qed**

**lemma** *argmax* (*setsum* (*randomBids' N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*) ⊆
*maximalStrictAllocations' N* (*set G*) *b* **by** *auto*

**abbreviation** *vcga' N G b r* == (*the-elem*
(*argmax* (*setsum* (*randomBids' N G b r*)) (*maximalStrictAllocations' N* (*set G*)

*b*))) −− *addedBidder′*

**corollary** *lm58*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*the-elem*
(*argmax* (*setsum* (*randomBids′ N G b r*)) (*maximalStrictAllocations′ N* (*set G*) *b*)) ∈
(*maximalStrictAllocations′ N* (*set G*) *b*) (**is** *the-elem ?X* ∈ *?R*) **using** *assms mm92b lm57*
**proof** −
**have** *card ?X=1* **using** *assms* **by** (*rule mm92b*) **moreover have** *?X* ⊆ *?R* **by** *auto*
**ultimately show** *?thesis* **using** *nn57b* **by** *blast*
**qed**

**corollary** *lm58b*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*vcga′ N G b r* ∈ (*Outside′* {*addedBidder′*})‘(*maximalStrictAllocations′ N* (*set G*) *b*)
**using** *assms lm58* **by** *blast*

**lemma** *lm62*: (*Outside′* {*addedBidder′*})‘(*maximalStrictAllocations′ N G b*) ⊆ *allAllocations′ N G*
**using** *Outside-def* **by** *force*

**theorem** *lm58d*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*vcga′ N G b r* ∈ *allAllocations′ N* (*set G*) (**is** *?a* ∈ *?A*) **using** *assms lm58b lm62*
**proof** − **have** *?a* ∈ (*Outside′* {*addedBidder′*})‘(*maximalStrictAllocations′ N* (*set G*) *b*)
**using** *assms* **by** (*rule lm58b*) **thus** *?thesis* **using** *lm62* **by** *fastforce* **qed**

**corollary** *lm59b*: **assumes** ∀ *X*. *X* ∈ *Range a* ⟶*b* (*addedBidder′*, *X*)=*0 finite a*
**shows**
*setsum b a* = *setsum b* (*a*−−*addedBidder′*)
**proof** −
**let** *?n*=*addedBidder′* **have** *finite* (*a*||{*?n*}) **using** *assms restrict-def* **by** (*metis finite-Int*)
**moreover have** ∀ *z* ∈ *a*||{*?n*}. *b z*=*0* **using** *assms restrict-def* **by** *fastforce*
**ultimately have** *setsum b* (*a*||{*?n*}) = *0* **using** *assms* **by** (*metis setsum.neutral*)
**thus** *?thesis* **using** *nn59 assms*(*2*) **by** (*metis comm-monoid-add-class.add.right-neutral*)
**qed**

**corollary** *lm59c*: **assumes** ∀ *a*∈*A*. *finite a* & (∀ *X*. *X*∈*Range a* ⟶ *b* (*addedBidder′*, *X*)=*0*)
**shows** {*setsum b a*| *a*. *a*∈*A*}={*setsum b* (*a* −− *addedBidder′*)| *a*. *a*∈*A*} **using** *assms lm59b*
**by** (*metis* (*lifting*, *no-types*))
**corollary** *lm58c*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*EX a*. ((*a* ∈ (*maximalStrictAllocations′ N* (*set G*) *b*))
& (*vcga′ N G b r* = *a* −− *addedBidder′*)
& (*a* ∈ *argmax* (*setsum b*) (*allStrictAllocations′* ({*addedBidder′*}∪*N*) (*set G*)))

) (**is** *EX a. - & - & a* ∈ *?X*)
**using** *assms lm58b argmax-def* **by** *fast*

**lemma assumes** *distinct G set G* ≠ {} *finite N* **shows**
∀ *aa*∈*allStrictAllocations′* ({*addedBidder′*}∪*N*) (*set G*). *finite aa*
**using** *assms* **by** (*metis List.finite-set mm44*)

**lemma** *lm61*: **assumes** *distinct G set G* ≠ {} *finite N*
∀ *aa*∈*allStrictAllocations′* ({*addedBidder′*}∪*N*) (*set G*). ∀ *X* ∈ *Range aa*. *b* (*addedBidder′*, *X*)=*0*
(**is** ∀ *aa*∈*?X*. -) **shows** *setsum b* (*vcga′ N G b r*)=*Max*{*setsum b aa*| *aa*. *aa* ∈ *allAllocations′ N* (*set G*)}
**proof** −
**let** *?n*=*addedBidder′* **let** *?s*=*setsum* **let** *?a*=*vcga′ N G b r* **obtain** *a* **where**
*0*: *a* ∈ *maximalStrictAllocations′ N* (*set G*) *b* & *?a*=*a*−−*?n* &
(*a*∈*argmax* (*setsum b*) (*allStrictAllocations′*({*addedBidder′*}∪*N*)(*set G*)))(**is** - & *?a*=- & *a*∈*?Z*)
**using** *assms*(*1*,*2*,*3*) *lm58c* **by** *blast* **have**
*1*: ∀ *aa*∈*?X*. *finite aa* & (∀ *X*. *X*∈*Range aa* ⟶ *b* (*?n*, *X*)=*0*) **using** *assms*(*4*) *List.finite-set mm44* **by** *metis* **have**
*2*: *a* ∈ *?X* **using** *0* **by** *auto* **have** *a* ∈ *?Z* **using** *0* **by** *fast*
**then have** *a* ∈ *?X*∩{*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *mm78* **by** *simp*
**then have** *a* ∈ {*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *mm78* **by** *simp*
**moreover have** *?s b* ‘ *?X* = {*?s b aa*| *aa*. *aa*∈*?X*} **by** *blast*
**ultimately have** *?s b a* = *Max* {*?s b aa*| *aa*. *aa*∈*?X*} **by** *auto*
**moreover have** {*?s b aa*| *aa*. *aa*∈*?X*} = {*?s b* (*aa*−−*?n*)| *aa*. *aa*∈*?X*} **using** *1* **by** (*rule lm59c*)
**moreover have** ... = {*?s b aa*| *aa*. *aa* ∈ *Outside′* {*?n*}‘*?X*} **by** *blast*
**moreover have** ... = {*?s b aa*| *aa*. *aa* ∈ *allAllocations′ N* (*set G*)} **by** *simp*
**ultimately have** *Max* {*?s b aa*| *aa*. *aa* ∈ *allAllocations′ N* (*set G*)} = *?s b a* **by** *presburger*
**moreover have** ... = *?s b* (*a*−−*?n*) **using** *1 2 lm59b* **by** (*metis* (*lifting*, *no-types*))
**ultimately show** *?s b ?a*=*Max*{*?s b aa*| *aa*. *aa* ∈ *allAllocations′ N* (*set G*)} **using** *0* **by** *presburger*
**qed**

Adequacy theorem: the allocation satisfies the standard pen-and-paper specification of a VCG auction. See, for example, [**?**, § 1.2].

**theorem** *lm61b*: **assumes** *distinct G set G* ≠ {} *finite N* ∀ *X*. *b* (*addedBidder′*, *X*)=*0*
**shows** *setsum b* (*vcga′ N G b r*)=*Max*{*setsum b aa*| *aa*. *aa* ∈ *allAllocations′ N* (*set G*)}
**using** *assms lm61* **by** *blast*

**corollary** *lm58e*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*vcga′ N G b r* ∈ *allocationsUniverse* & ⋃ *Range* (*vcga′ N G b r*) ⊆ *set G* **using** *assms lm58b*
**proof** −
**let** *?a*=*vcga′ N G b r* **let** *?n*=*addedBidder′*

**obtain** *a* **where**
*0*: *?a=a −− addedBidder′* & *a* ∈ *maximalStrictAllocations′ N* (*set G*) *b*
**using** *assms lm58b* **by** *blast*
**then moreover have**
*1*: *a* ∈ *allStrictAllocations′* ({*?n*}∪*N*) (*set G*) **by** *auto*
**moreover have** *maximalStrictAllocations′ N* (*set G*) *b* ⊆ *allocationsUniverse*
**by** (*metis* (*lifting*, *mono-tags*) *lm03 lm50 subset-trans*)
**ultimately moreover have** *?a=a −− addedBidder′* & *a* ∈ *allocationsUniverse*
**by** *blast*
**then have** *?a* ∈ *allocationsUniverse* **using** *lm35d* **by** *auto*
**moreover have** ⋃ *Range a= set G* **using** *nn24c 1* **by** *metis*
**then moreover have** ⋃ *Range ?a* ⊆ *set G* **using** *Outside-def 0* **by** *fast*
**ultimately show** *?thesis* **using** *lm35d Outside-def* **by** *blast*
**qed**

**lemma** *vcga′ N G b r = the-elem* ((*argmax* ∘ *setsum*) (*randomBids′ N G b r*) ((*argmax* ∘ *setsum*) *b* (*allStrictAllocations′* ({*addedBidder′*}∪*N*) (*set G*)))) *−− addedBidder′* **by** *simp*

**abbreviation** *vcgp′ N G b r n ==*
*Max* (*setsum b* ‘ (*allAllocations′* (*N*−{*n*}) (*set G*))) − (*setsum b* (*vcga′ N G b r −− n*))

**lemma** *lm63*: **assumes** *x* ∈ *X finite X* **shows** *Max* (*f‘X*) *>= f x* (**is** *?L >= ?R*)
**using** *assms*
**by** (*metis* (*hide-lams*, *no-types*) *Max.coboundedI finite-imageI image-eqI*)

The price paid by any participant is non-negative.

**theorem** *NonnegPrices*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
*vcgp′ N G* (*b*) *r n >=* (*0::price*)
**proof** −
**let** *?a=vcga′ N G b r* **let** *?A=allAllocations′* **let** *?A′=allAllocations′′′′* **let** *?f=setsum b*
**have** *?a* ∈ *?A N* (*set G*) **using** *assms* **by** (*rule lm58d*)
**then have** *?a* ∈ *?A′ N* (*set G*) **using** *lm28e* **by** *metis* **then have** *?a −− n* ∈ *?A′* (*N*−{*n*}) (*set G*) **by** (*rule lm44*)
**then have** *?a −− n* ∈ *?A* (*N*−{*n*}) (*set G*) **using** *lm28e* **by** *metis*
**moreover have** *finite* (*?A* (*N*−{*n*}) (*set G*))
**by** (*metis List.finite-set assms*(*3*) *finite.emptyI finite-Diff finite-Un finite-imageI finite-insert lm59*)
**ultimately have** *Max* (*?f‘*(*?A* (*N*−{*n*}) (*set G*))) *>= ?f* (*?a −− n*) (**is** *?L >= ?R*) **by** (*rule lm63*)
**then have** *?L − ?R >=0* **by** *linarith* **thus** *?thesis* **by** *fast*
**qed**

**lemma** *lm19b*: *allStrictAllocations′ N G = possibleAllocationsRel N G* **using** *assms* **by** (*metis lm19*)
**abbreviation** *strictAllocationsUniverse == allocationsUniverse*

**abbreviation** *Goods bids* == $\bigcup$((*snd*∘*fst*)'*bids*)

**corollary** *lm45*: **assumes** $a \in$ *allAllocations''''* *N G* **shows** *Range a* $\in$ *partition-sUniverse*
**using** *assms* **by** (*metis* (*lifting*, *mono-tags*) *Int-iff lm22 mem-Collect-eq*)

**corollary** *lm45a*: **assumes** $a \in$ *allAllocations'* *N G* **shows** *Range a* $\in$ *partitionsUniverse*
**proof** − **have** $a \in$ *allAllocations''''* *N G* **using** *assms lm28e* **by** *metis* **thus** *?thesis*
**by** (*rule lm45*) **qed**

**corollary assumes**
$N \neq \{\}$ *distinct G set G* $\neq \{\}$ *finite N*
**shows** (*Outside'* {*addedBidder'*}) ' (*terminatingAuctionRel N G* (*bids*) *random*)
=
{*chosenAllocation' N G bids random* −− (*addedBidder'*)} (**is** *?L=?R*) **using** *assms mm92 Outside-def*
**proof** −
**have** *?R* = *Outside'* {*addedBidder'*} ' {*chosenAllocation' N G bids random*} **using** *Outside-def*
**by** *blast*
**moreover have** ... = (*Outside'*{*addedBidder'*})'(*terminatingAuctionRel N G bids random*) **using** *assms mm92*
**by** *blast*
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *terminatingAuctionRel N G b r* =
((*argmax* (*setsum* (*resolvingBid' N G b* (*nat r*)))) ∘ (*argmax* (*setsum b*)))
(*possibleAllocationsRel N* (*set G*)) **by** *force*
**term** (*Union* ∘ (*argmax* (*setsum* (*resolvingBid' N G b* (*nat r*)))) ∘ (*argmax* (*setsum b*)))
(*possibleAllocationsRel N* (*set G*))

**lemma** *maximalStrictAllocations' N G b=winningAllocationsRel* ({*addedBidder'*} ∪ *N*) *G b* **by** *fast*

**lemma** *lm64*: **assumes** $a \in$ *allocationsUniverse*
*n1* $\in$ *Domain a n2* $\in$ *Domain a*
*n1* $\neq$ *n2*
**shows** *a,,n1* ∩ *a,,n2*={} **using** *assms is-partition-def lm22 mem-Collect-eq*
**proof** − **have** *Range a* $\in$ *partitionsUniverse* **using** *assms lm22* **by** *blast*
**moreover have** $a \in$ *injectionsUniverse* & $a \in$ *partitionValuedUniverse* **using** *assms* **by** (*metis* (*lifting*, *no-types*) *IntD1 IntD2*)
**ultimately moreover have** *a,,n1* $\in$ *Range a* **using** *assms*
**by** (*metis* (*mono-tags*) *eval-runiq-in-Range mem-Collect-eq*)
**ultimately moreover have** *a,,n1* $\neq$ *a,,n2* **using**
*assms converse.intros eval-runiq-rel mem-Collect-eq runiq-basic* **by** (*metis* (*lifting*, *no-types*))

**ultimately show** *?thesis* **using** *is-partition-def* **by** (*metis* (*lifting*, *no-types*) *assms*(*3*) *eval-runiq-in-Range mem-Collect-eq*)
**qed**

**lemma** *lm64c*: **assumes** $a \in$ *allocationsUniverse*
*n1* $\in$ *Domain a n2* $\in$ *Domain a*
*n1* $\neq$ *n2*
**shows** *a*,,,*n1* $\cap$ *a*,,,*n2*={} **using** *assms lm64 lll82* **by** *fastforce*

No good is assigned twice.

**theorem** *PairwiseDisjointAllocations*:
**assumes** *distinct G set G* $\neq$ {} *finite N*
*n1* $\in$ *Domain* (*vcga' N G b r*) *n2* $\in$ *Domain* (*vcga' N G b r*) *n1* $\neq$ *n2*
**shows** (*vcga' N G b r*),,,*n1* $\cap$ (*vcga' N G b r*),,,*n2*={}
**proof** −
**have** *vcga' N G b r* $\in$ *allocationsUniverse* **using** *lm58e assms* **by** *blast*
**then show** *?thesis* **using** *lm64c assms* **by** *fast*
**qed**

**lemma assumes** *R*,,,*x* $\neq$ {} **shows** *x* $\in$ *Domain R* **using** *assms*
**proof** − **have** $\bigcup$ (*R*``{*x*}) $\neq$ {} **using** *assms*(*1*) **by** *fast*
**then have** *R*``{*x*} $\neq$ {} **by** *fast* **thus** *?thesis* **by** *blast* **qed**

**lemma assumes** *runiq f* **and** *x* $\in$ *Domain f* **shows** (*f* ,, *x*) $\in$ *Range f* **using** *assms*
**by** (*rule eval-runiq-in-Range*)

Nothing outside the set of goods is allocated.

**theorem** *OnlyGoodsAllocated*: **assumes** *distinct G set G* $\neq$ {} *finite N g* $\in$ (*vcga' N G b r*),,,*n*
**shows** *g* $\in$ *set G*
**proof** −
**let** *?a=vcga' N G b r*
**have** *?a* $\in$ *allocationsUniverse* **using** *assms*(*1*,*2*,*3*) *lm58e* **by** *blast*
**then have** *runiq ?a* **using** *assms*(*1*,*2*,*3*) **by** *blast*
**moreover have** *n* $\in$ *Domain ?a* **using** *assms*(*4*) *eval-rel-def* **by** *fast*
**ultimately moreover have** *?a*,,*n* $\in$ *Range ?a* **using** *eval-runiq-in-Range* **by** *fast*

**ultimately have** *?a*,,,*n* $\in$ *Range ?a* **using** *lll82* **by** *fastforce*
**then have** *g* $\in \bigcup$ *Range ?a* **using** *assms* **by** *blast*
**moreover have** $\bigcup$ *Range ?a* $\subseteq$ *set G* **using** *assms*(*1*,*2*,*3*) *lm58e* **by** *fast*
**ultimately show** *?thesis* **by** *blast*
**qed**

**abbreviation** *allStrictAllocations N G* == *possibleAllocationsAlg3 N G*
**abbreviation** *maximalStrictAllocations N G b*==
*argmax* (*setsum b*) (*set* (*allStrictAllocations* ({*addedBidder'*}$\cup$*N*) *G*))

**abbreviation** *chosenAllocation N G b r* ==

*hd*(*perm2* (*takeAll* (%*x*. *x*∈ (*argmax* ∘ *setsum*) *b* (*set* (*allStrictAllocations N G*)))
(*allStrictAllocations N G*)) *r*)
**abbreviation** *maxbid a N G* == (*bidMaximizedBy a N G*) *Elsee 0*
**abbreviation** *linearCompletion* (*bids*) *N G* ==
(*LinearCompletion bids N G*) *Elsee 0*
**abbreviation** *tiebids a N G* == *linearCompletion* (*maxbid a N G*) *N G*
**abbreviation** *resolvingBid N G bids random* == *tiebids* (*chosenAllocation N G bids random*) *N* (*set G*)
**abbreviation** *randomBids N* Ω *b random*==*resolvingBid* (*N*∪{*addedBidder′*}) Ω *b random*
**definition** *vcga N G b r* == (*the-elem*
(*argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations N G b*))) −−
*addedBidder′*

**abbreviation** *allAllocations N* Ω ==
(*Outside′* {*addedBidder′*}) ‘ *set* (*allStrictAllocations* (*N* ∪ {*addedBidder′*}) Ω)
**definition** *vcgp N G b r n* =
*Max* (*setsum b* ‘ (*allAllocations* (*N*−{*n*}) *G*)) − (*setsum b* (*vcga N G b r* −− *n*))

**lemma** *lm01*: **assumes** *x* ∈ *Domain f* **shows** *toFunction f x* = (*f Elsee 0*) *x*
**by** (*metis assms toFunction-def*)
**lemma** *lm03*: *Domain* (*Y* × {*0*::*nat*}) = *Y* & *Domain* (*X* × {*1*})=*X* **by** *blast*
**lemma** *lm04*: *Domain* (*X* <|| *Y*) = *X* ∪ *Y* **using** *lm03 paste-Domain sup-commute*
**by** *metis*
**corollary** *lm04b*: *Domain* (*bidMaximizedBy a N G*) = *pseudoAllocation a* ∪ *N* × (*finestpart G*) **using** *lm04*
**by** *metis*
**lemma** *lm19*: (*pseudoAllocation a*) ⊆ *Domain* (*bidMaximizedBy a N G*) **by** (*metis lm04 Un-upper1*)

**lemma** *lm02*: **assumes** *x* ∈ (*N* × (*Pow G* − {{}})) **shows**
*linearCompletion′ b N G x*=*linearCompletion b N G x*
**using** *assms lm01 Domain.simps imageI* **by** (*metis*(*no-types*,*lifting*))

**corollary** *lm20*: **assumes** ∀ *x* ∈ *X*. *f x* = *g x* **shows** *setsum f X* = *setsum g X*
**using** *assms setsum.cong* **by** *auto*

**lemma** *lm06*: **assumes** *fst pair* ∈ *N snd pair* ∈ *Pow G* − {{}} **shows** *setsum*
(%*g*.
(*toFunction* (*bidMaximizedBy a N G*))
(*fst pair*, *g*)) (*finestpart* (*snd pair*)) =
*setsum* (%*g*.
((*bidMaximizedBy a N G*) *Elsee 0*)
(*fst pair*, *g*)) (*finestpart* (*snd pair*))
**using** *assms lm01 lm05 lm04 Un-upper1 UnCI UnI1 setsum.cong mm55n Diff-iff Pow-iff in-mono*

**proof** −
**let** *?f1*=%*g*.(*toFunction* (*bidMaximizedBy a N G*))(*fst pair*, *g*)
**let** *?f2*=%*g*.((*bidMaximizedBy a N G*) *Elsee 0*)(*fst pair*, *g*)
**{**
**fix** *g* **assume** $g \in$ *finestpart* (*snd pair*) **then have**
*0*: $g \in$ *finestpart G* **using** *assms mm55n* **by** (*metis Diff-iff Pow-iff in-mono*)
**have** *?f1 g* = *?f2 g*
**proof** −
**have** $\bigwedge x_1\ x_2.\ (x_1, g) \in x_2 \times$ *finestpart G* $\vee\ x_1 \notin x_2$ **by** (*metis 0 mem-Sigma-iff*)

**thus** (*pseudoAllocation a* <| (*N* × *finestpart G*)) (*fst pair*, *g*) = *maxbid a N G* (*fst pair*, *g*)
**by** (*metis* (*no-types*) *lm04 UnCI assms*(*1*) *toFunction-def*)
**qed**
**}**
**thus** *?thesis* **using** *setsum.cong* **by** *simp*
**qed**

**corollary** *lm07*: **assumes** *pair* ∈ *N* × (*Pow G* − {{}}) **shows**
*partialCompletionOf* (*toFunction* (*bidMaximizedBy a N G*)) *pair* =
*partialCompletionOf* ((*bidMaximizedBy a N G*) *Elsee 0*) *pair* **using** *assms lm06*
**proof** −
**have** *fst pair* ∈ *N* **using** *assms* **by** *force*
**moreover have** *snd pair* ∈ *Pow G* − {{}} **using** *assms*(*1*) **by** *force*
**ultimately show** *?thesis* **using** *lm06* **by** *blast*
**qed**

**lemma** *lm08*: **assumes** ∀ *x* ∈ *X*. *f x* = *g x* **shows** *f'X*=*g'X* **using** *assms* **by** (*metis image-cong*)

**corollary** *lm09*: ∀ *pair* ∈ *N* × (*Pow G* − {{}}).
*partialCompletionOf* (*toFunction* (*bidMaximizedBy a N G*)) *pair* =
*partialCompletionOf* ((*bidMaximizedBy a N G*) *Elsee 0*) *pair* **using** *lm07*
**by** *blast*

**corollary** *lm10*:
(*partialCompletionOf* (*toFunction* (*bidMaximizedBy a N G*))) ' (*N* × (*Pow G* − {{}}))=
(*partialCompletionOf* ((*bidMaximizedBy a N G*) *Elsee 0*)) ' (*N* × (*Pow G* − {{}}))
(**is** *?f1* ' *?Z* = *?f2* ' *?Z*)
**proof** −
**have** ∀ *z* ∈ *?Z*. *?f1 z* = *?f2 z* **by** (*rule lm09*) **thus** *?thesis* **by** (*rule lm08*)
**qed**

**corollary** *lm11*: *LinearCompletion* (*toFunction* (*bidMaximizedBy a N G*)) *N G* =
*LinearCompletion* ((*bidMaximizedBy a N G*) *Elsee 0*) *N G* **using** *lm10* **by** *metis*

**corollary** *lm12*: *LinearCompletion* (*maxbid' a N G*) *N G* = *LinearCompletion* (*maxbid a N G*) *N G*

**using** *lm11* **by** *metis*

**lemma** *lm13*: **assumes** $x \in (N \times (Pow\ G - \{\{\}\}))$ **shows**
*linearCompletion′* (*maxbid′ a N G*) *N G x* = *linearCompletion* (*maxbid a N G*) *N G x*
(**is** *?f1 ?g1 N G x* = *?f2 ?g2 N G x*)
**using** *assms lm02 lm12*
**proof** −
**let** *?h1=maxbid′ a N G* **let** *?h2=maxbid a N G* **let** *?hh1=real* ∘ *?h1* **let** *?hh2=real* ∘ *?h2*
**have** *LinearCompletion ?h1 N G* = *LinearCompletion ?h2 N G* **using** *lm12* **by** *metis*
**moreover have** *linearCompletion ?h2 N G*=(*LinearCompletion ?h2 N G*) *Elsee 0* **by** *fast*
**ultimately have** *linearCompletion ?h2 N G=LinearCompletion ?h1 N G Elsee 0* **by** *presburger*
**moreover have** ... *x* = (*toFunction* (*LinearCompletion ?h1 N G*)) *x* **using** *assms* **by** (*metis* (*mono-tags*) *lm01 mm64*)
**ultimately have** *linearCompletion ?h2 N G x* = (*toFunction* (*LinearCompletion ?h1 N G*)) *x*
**by** (*metis* (*lifting*, *no-types*))
**thus** *?thesis* **by** *simp*
**qed**

**corollary** *lm70c*: **assumes** *card N* > *0 distinct G* **shows**
*possibleAllocationsRel N* (*set G*) = *set* (*possibleAllocationsAlg3 N G*)
**using** *assms lm70b StrictCombinatorialAuction.lm01* **by** *metis*

**lemma** *lm24*: **assumes** *card A* > *0 card B* > *0* **shows** *card* ($A \cup B$) > *0*
**using** *assms card-gt-0-iff finite-Un sup-eq-bot-iff* **by** (*metis*(*no-types*))
**corollary** *lm24b*: **assumes** *card A* > *0* **shows** *card* ($\{a\} \cup A$) > *0* **using** *assms lm24*
**by** (*metis card-empty card-insert-disjoint empty-iff finite.emptyI lessI*)

**corollary assumes** *card N* > *0 distinct G* **shows**
*maximalStrictAllocations′ N* (*set G*) *b* = *maximalStrictAllocations N G b*
**using** *assms lm70c lm24b* **by** (*metis*(*no-types*))

**corollary** *lm70d*: **assumes** *card N* > *0 distinct G* **shows**
*allStrictAllocations′ N* (*set G*) = *set* (*allStrictAllocations N G*) **using** *assms lm70c* **by** *blast*

**corollary** *lm70f*: **assumes** *card N* > *0 distinct G* **shows**
*winningAllocationsRel N* (*set G*) *b* =
(*argmax* ∘ *setsum*) *b* (*set* (*allStrictAllocations N G*)) **using** *assms lm70c* **by** (*metis*

*comp-apply*)

**corollary** *lm70g*: **assumes** *card N* > *0 distinct G* **shows** *chosenAllocation′ N G b r* = *chosenAllocation N G b r* **using** *assms lm70f* **by** *metis*

**corollary** *lm13b*: **assumes** $x \in (N \times (Pow\ G - \{\{\}\}))$ **shows** *tiebids′ a N G x* = *tiebids a N G x* (**is** *?L=-*)

**proof** −

**have** *?L* = *linearCompletion′* (*maxbid′ a N G*) *N G x* **by** *fast* **moreover have** ...=

*linearCompletion* (*maxbid a N G*) *N G x* **using** *assms* **by** (*rule lm13*) **ultimately show** *?thesis* **by** *fast*

**qed**

**lemma** *lm14*: **assumes** *card N* > *0 distinct G* $a \subseteq (N \times (Pow\ (set\ G) - \{\{\}\}))$ **shows**

*setsum* (*resolvingBid′ N G b r*) *a* = *setsum* (*resolvingBid N G b r*) *a* (**is** *?L=?R*)

**proof** −

**let** *?c′=chosenAllocation′ N G b r* **let** *?c=chosenAllocation N G b r* **let** *?r′=resolvingBid′ N G b r*

**have** *?c′* = *?c* **using** *assms*(*1*,*2*) **by** (*rule lm70g*) **then**

**have** *?r′* = *tiebids′ ?c N* (*set G*) **by** *presburger*

**moreover have** $\forall x \in a$. *tiebids′ ?c N* (*set G*) *x* = *tiebids ?c N* (*set G*) *x* **using** *assms*(*3*) *lm13b* **by** *blast*

**ultimately have** $\forall x \in a$. *?r′ x* = *resolvingBid N G b r x* **by** *presburger*

**thus** *?thesis* **using** *setsum.cong* **by** *simp*

**qed**

**lemma** *lm15*: *allStrictAllocations′ N G* $\subseteq$ *Pow* $(N \times (Pow\ G - \{\{\}\}))$ **by** (*metis PowI mm63c subsetI*)

**corollary** *lm14b*: **assumes** *card N* > *0 distinct G* $a \in$ *allStrictAllocations′ N* (*set G*)

**shows** *setsum* (*resolvingBid′ N G b r*) *a* = *setsum* (*resolvingBid N G b r*) *a*

**proof** −

**have** $a \subseteq N \times (Pow\ (set\ G) - \{\{\}\})$ **using** *assms*(*3*) *lm15* **by** *blast*

**thus** *?thesis* **using** *assms*(*1*,*2*) *lm14* **by** *blast*

**qed**

**corollary** *lm14c*: **assumes** *finite N distinct G* $a \in$ *maximalStrictAllocations′ N* (*set G*) *b*

**shows** *setsum* (*randomBids′ N G b r*) *a* = *setsum* (*randomBids N G b r*) *a*

**proof** −

**have** *card* ($N \cup \{$*addedBidder′*$\}$)>*0* **using** *assms*(*1*) *sup-eq-bot-iff insert-not-empty* **by** (*metis card-gt-0-iff finite.emptyI finite.insertI finite-UnI*)

**moreover have** *distinct G* **using** *assms*(*2*) **by** *simp*

**moreover have** $a \in$ *allStrictAllocations′* ($N \cup \{$*addedBidder′*$\}$) (*set G*) **using** *assms*(*3*) **by** *fastforce*

**ultimately show** *?thesis* **by** (*rule lm14b*)

**qed**

**lemma** *lm16*: **assumes** $\forall x \in X.\ f\ x = g\ x$ **shows** *argmax f X=argmax g X*
**using** *assms MiscTools.lm02 Collect-cong image-cong*
**by** (*metis*(*no-types*,*lifting*))

**corollary** *mm92c*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
*1=card* (*argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*))
**using** *assms mm92b lm14c*
**proof** –
**have** $\forall$ *a* $\in$ *maximalStrictAllocations' N* (*set G*) *b*.
*setsum* (*randomBids' N G b r*) *a* = *setsum* (*randomBids N G b r*) *a* **using** *assms*(*3*,*1*) *lm14c* **by** *blast*
**then have** *argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*) =
*argmax* (*setsum* (*randomBids' N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*)
**using** *lm16* **by** *blast*
**moreover have** *card ... = 1* **using** *assms* **by** (*rule mm92b*)
**ultimately show** *?thesis* **by** *presburger*
**qed**
**corollary** *lm70e*: **assumes** *finite N distinct G* **shows**
*maximalStrictAllocations' N* (*set G*) *b=maximalStrictAllocations N G b*
**proof** –
**let** *?N*={*addedBidder'*} ∪ *N*
**have** *card ?N>0* **using** *assms*(*1*) **by** (*metis* (*full-types*) *card-gt-0-iff finite-insert insert-is-Un insert-not-empty*)
**thus** *?thesis* **using** *assms*(*2*) *lm70d* **by** *metis*
**qed**
**corollary assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
*1=card* (*argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations N G b*))
**proof** –
**have** *1=card* (*argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*))
**using** *assms* **by** (*rule mm92c*)
**moreover have** *maximalStrictAllocations' N* (*set G*) *b* = *maximalStrictAllocations N G b*
**using** *assms*(*3*,*1*) **by** (*rule lm70e*) **ultimately show** *?thesis* **by** *metis*
**qed**

**lemma** *maximalStrictAllocations' N* (*set G*) *b* $\subseteq$ *Pow* (({*addedBidder'*}∪*N*) $\times$ (*Pow* (*set G*) – {{}}))
**using** *lm15 UniformTieBreaking.lm03 subset-trans* **by** (*metis* (*no-types*))

**lemma** *lm17*: (*image converse*) (*Union X*)=*Union* ((*image converse*) ` *X*) **by** *auto*

**lemma** *possibleAllocationsRel N G* =
*Union* {*converse*‘(*injections Y N*)| *Y*. *Y* ∈ *all-partitions G*}
**by** *auto*

**lemma** *allStrictAllocations′ N* Ω = *Union*{{*a*ˆ−*1*|*a*. *a*∈*injections Y N*}|*Y*. *Y*∈*all-partitions* Ω} **by** *auto*

**end**