Vcg

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

March 19, 2015

Contents

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

theory *SetUtils*
imports
Main

begin

2 Equality

An inference (introduction) 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-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 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*: $\{\ f\ x \mid x\ .\ x \in S\ \} = f\ `\ S$
by *auto*

5 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*

6 Miscellaneous

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

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

lemma *lm01*: **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 trivial-implies-finite order-refl subset-singletonD*
by (*metis*(*no-types*))

lemma *lm02*: **assumes** *finite X*
shows *trivial* $X{=}($*card* $X \leq 1)$
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 *lm03*: **shows** *trivial* $\{x\}$
by (*metis order-refl the-elem-eq trivial-def*)

lemma *lm04*: **assumes** *trivial* X $\{x\} \subseteq X$
shows $\{x\} = X$
using *singleton-sub-trivial-uniq assms* **by** (*metis subset-antisym trivial-def*)

lemma *lm05*: **assumes** $\neg$ *trivial* X *trivial* T
shows $X - T \neq \{\}$
using *assms* **by** (*metis Diff-iff empty-iff subsetI trivial-subset*)

lemma *lm06*: **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 lm02*)

lemma *lm07*: *trivial* $(A \times B) = ($*finite* $(A \times B)$ & *card* $A * ($*card* $B) \leq 1)$
using *lm01 lm06* **by** *blast*

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

lemma *trivial-cartesian*: **assumes** *trivial* X *trivial* Y
shows *trivial* $(X \times Y)$
using *assms lm07 One-nat-def Sigma-empty1 Sigma-empty2 card-empty card-insert-if finite-SigmaI trivial-implies-finite nat-1-eq-mult-iff order-refl subset-singletonD trivial-def trivial-empty*
by (*metis* (*full-types*))

lemma *trivial-same*: *trivial* $X = (\forall x1 \in X.\ \forall x2 \in X.\ x1 = x2)$
using *trivial-def trivial-imp-no-distinct ex-in-conv insertCI subsetI subset-singletonD trivial-singleton*
by (*metis* (*no-types*, *hide-lams*))

lemma *lm08*: **assumes** $(Pow\ X \subseteq \{\{\},X\})$
shows *trivial X*
unfolding *trivial-same* **using** *assms* **by** *auto*

lemma *lm09*: **assumes** *trivial X*
shows $(Pow\ X \subseteq \{\{\},X\})$
using *assms trivial-same* **by** *fast*

lemma *lm10*: $trivial\ X = (Pow\ X \subseteq \{\{\},X\})$
using *lm08 lm09* **by** *metis*

lemma *lm11*: $(\{x\} \times UNIV) \cap P = \{x\} \times (P \text{ `` } \{x\})$
by *fast*

lemma *lm12*: $(x,y) \in P = (y \in P\text{``}\{x\})$
by *simp*

lemma *lm13*: **assumes** *inj-on f A inj-on f B*
shows $inj\text{-}on\ f\ (A \cup B) = (f\text{`}(A-B) \cap (f\text{`}(B-A)) = \{\})$
using *assms inj-on-Un* **by** (*metis*)

lemma *injection-union*: **assumes** *inj-on f A inj-on f B* $(f\text{`}A) \cap (f\text{`}B) = \{\}$
shows $inj\text{-}on\ f\ (A \cup B)$
using *assms lm13* **by** *fast*

lemma *lm14*: $(Pow\ X = \{X\}) = (X=\{\})$
by *auto*

end

7 Partitions of sets

theory *Partitions*
imports
Main
SetUtils

begin

We define the set of all partitions of a set (*all-partitions*) in textbook style, as well as a computable function *all-partitions-list* to algorithmically compute this set (then represented as a list). This function is suitable for code generation. We prove the equivalence of the two definition in order to ensure that the generated code correctly implements the original textbook-style definition. For further background on the overall approach, see Caminati, Kerber, Lange, Rowat: Proving soundness of combinatorial Vickrey auctions and generating verified executable code, 2013.

P is a family of non-overlapping sets.

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

A subfamily of a non-overlapping family is also a non-overlapping family

lemma *subset-is-non-overlapping*:
assumes *subset*: $P \subseteq Q$ **and**
non-overlapping: *is-non-overlapping* Q
shows *is-non-overlapping* 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** *non-overlapping* **unfolding** *is-non-overlapping-def* **by** *force*
}
then show *?thesis* **unfolding** *is-non-overlapping-def* **by** *force*
qed

The family that results from removing one element from an equivalence class of a non-overlapping family is not otherwise a member of the family.

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

Inserting into a non-overlapping family P a set X, which is disjoint with the set partitioned by P, yields another non-overlapping family.

lemma *non-overlapping-extension1*:
fixes P::*'a set set*
and X::*'a set*
assumes *partition*: *is-non-overlapping* P
and *disjoint*: $X \cap \bigcup\ P = \{\}$
and *non-empty*: $X \neq \{\}$
shows *is-non-overlapping* (*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 ∩ Z ≠ {}
      then show Y = Z
        using Y-Z-in-ext-P partition disjoint
        unfolding is-non-overlapping-def
        by fast
    next
      assume Y = Z
      then show Y ∩ Z ≠ {}
        using Y-Z-in-ext-P partition non-empty
        unfolding is-non-overlapping-def
        by auto
    qed
  }
  then show ?thesis unfolding is-non-overlapping-def by force
qed
```

An element of a non-overlapping family has no intersection with any other of its elements.

```
lemma disj-eq-classes:
  fixes P::'a set set
    and X::'a set
  assumes is-non-overlapping P
      and X ∈ P
  shows X ∩ ⋃ (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-non-overlapping-def by fast
    then have x ∈ {} using other-eq-class by fast
  }
  then show ?thesis by blast
qed
```

The empty set is not element of a non-overlapping family.

```
lemma no-empty-in-non-overlapping:
  assumes is-non-overlapping p
  shows {} ∉ p

  using assms is-non-overlapping-def by fast
```

P is a partition of the set A. The infix notation takes the form "noun-verb-object"

```
definition is-partition-of (infix partitions 75)
```

where *is-partition-of* P A = ($\bigcup$ P = A $\wedge$ *is-non-overlapping* P)

No partition of a non-empty set is empty.

lemma *non-empty-imp-non-empty-partition*:
  assumes $A \neq \{\}$
    and P *partitions* A
  shows $P \neq \{\}$
  using *assms* **unfolding** *is-partition-of-def* **by** *fast*

Every element of a partitioned set ends up in one element in the partition.

lemma *elem-in-partition*:
  assumes *in-set*: $x \in A$
    and *part*: P *partitions* A
  obtains X **where** $x \in X$ **and** $X \in P$
  using *part in-set* **unfolding** *is-partition-of-def is-non-overlapping-def* **by** (*auto simp add*: *UnionE*)

Every element of the difference of a set A and another set B ends up in an element of a partition of A, but not in an element of the partition of $\{B\}$.

lemma *diff-elem-in-partition*:
  assumes *x*: $x \in A - B$
    and *part*: P *partitions* A
  shows $\exists$ $S \in P - \{ B \}$. $x \in S$

proof −
  from *part x* **obtain** X **where** $x \in X$ **and** $X \in P$
    by (*metis Diff-iff elem-in-partition*)
  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 set.

lemma *elem-in-uniq-set*:
  assumes *in-set*: $x \in A$
    and *part*: P *partitions* 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-partition*) *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-non-overlapping-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 $\{A\}$ *partitions* A **unfolding** *is-partition-of-def is-non-overlapping-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 $\{\}$ *partitions* $\{\}$
  unfolding *is-partition-of-def is-non-overlapping-def* **by** *fast*

Any partition of the empty set is empty.

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

Classical set-theoretical definition of "all partitions of a set A"

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

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 new$_elintoaspecifiedsetSinsideagivenfamilyofsets$

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 a non-overlapping family of sets yields another non-overlapping family.

lemma *non-overlapping-extension2*:
fixes *new-el*::$'a$
and P::$'a$ *set set*
and X::$'a$ *set*
assumes *non-overlapping*: *is-non-overlapping* P
and *class-element*: $X \in P$
and *new*: *new-el* $\notin \bigcup$ P
shows *is-non-overlapping* (*insert-into-member new-el* P X)
proof −
let $?Y$ = *insert new-el* X
have *rest-is-non-overlapping*: *is-non-overlapping* $(P - \{X\})$
using *non-overlapping subset-is-non-overlapping* **by** *blast*
have $*$: $X \cap \bigcup (P - \{X\}) = \{\}$
using *non-overlapping class-element* **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-non-overlapping* (*insert* $?Y$ $(P - \{X\})$)
using *rest-is-non-overlapping disjoint non-empty* **by** (*rule non-overlapping-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-equivalence*:
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 characterization 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 *set*::*'a set*
shows $\bigcup$ *insert-into-member elem P set* = $\bigcup$ *insert* (*set* $\cup$ {*elem*}) (P − {*set*})

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

Assuming that P is a partition of a set S, and *new-el* $\notin S$, the function defined below yields all possible partitions of $S \cup$ {*new-el*} that are coarser than P (i.e. not splitting classes that already exist in P). These comprise one partition with a class {*new-el*} and all other classes unchanged, as well as all partitions obtained by inserting *new-el* into one class of P at a time. While we use the definition to build coarser partitions of an existing partition P, the definition itself does not require P to be a partition.

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 class {*new-el*} *and leaving all previous 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 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 class {*new-el*} *and leaving all previous 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 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-equivalence*:
  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-equivalence* **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 classes, is $\text{non}_o verlapping$.

lemma *non-overlapping-extension3*:
  fixes *elem*::*'a*
    and *P*::*'a set set*
    and *Q*::*'a set set*
  assumes *P-non-overlapping*: *is-non-overlapping P*
      and *new-elem*: *elem* ∉ ⋃ *P*
      and *Q-coarser*: *Q* ∈ *coarser-partitions-with elem P*
  shows *is-non-overlapping 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-non-overlapping new-elem non-overlapping-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** *non-overlapping-extension2 P-non-overlapping 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 one of the necessary criteria 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* = (λX . X – {*elem*}) ‘ P – {{}}

alternative characterization 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* = $\bigcup$ ((λx . x – {*elem*}) ‘ P – {{}})
    unfolding *partition-without-def* **by** *fast*
  also have ... = $\bigcup$ P – {*elem*} **by** *blast*

finally show *?thesis* **.**
qed

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

lemma *super-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 class of sets obtained by removing an element from a non-overlapping class is another non-overlapping clas.

lemma *non-overlapping-without-is-non-overlapping*:
  fixes *elem*::′*a*
    and *P*::′*a set set*
  assumes *is-non-overlapping P*
  shows *is-non-overlapping* (*partition-without elem P*) (**is** *is-non-overlapping ?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-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-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-non-overlapping-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-non-overlapping-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 *non-overlapping*: *is-non-overlapping 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* $\cup \{Y\}$ **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-non-overlapping ?elem-neq-classes*
    using *subset-is-non-overlapping non-overlapping*
    by *blast*
  have *elem-eq-wrt-P*: *?elem-eq* $\in$ *?remove-elem* ‘ *P* **using** *elem-eq-class'* **by** *blast*

  {
    fix *W* **assume** *W-eq-class*: $W \in$ *?elem-neq-classes*
    then have $elem \notin W$
      using *elem-eq-class elem-eq-class' non-overlapping is-non-overlapping-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* $\cup \{Y\}) - \{\{\}\}$ **using**
P-wrt-elem **by** *presburger*
  also have ... = *?elem-neq-classes* $\cup \{$*?elem-eq*$\} - \{\{\}\}$
    using *Y-elem-eq elem-neq-classes-id image-Un* **by** *metis*
  finally have *Q-wrt-elem*: *?Q* = *?elem-neq-classes* $\cup \{$*?elem-eq*$\} - \{\{\}\}$ **.**

  have *?elem-eq* $= \{\} \vee$ *?elem-eq* $\notin P$
      using *elem-eq-class elem-eq-class' non-overlapping Diff-Int-distrib2 Diff-iff*
empty-Diff insert-iff
unfolding *is-non-overlapping-def* **by** *metis*
  then have *?elem-eq* $\notin P$

using *non-overlapping no-empty-in-non-overlapping*
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-in-non-overlapping 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-in-non-overlapping non-overlapping non-overlapping-without-is-non-overlapping*
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 ... = ({} $\cup$ *?elem-neq-classes*) $\cup$ {*?elem-eq* $\cup$ {*elem*}} **using** *elem-neq-classes* **by** *force*
also have ... = *?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* :: $'a \Rightarrow 'a$ *set set set* $\Rightarrow 'a$ *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* :: $'a \Rightarrow 'a$ *set list list* $\Rightarrow 'a$ *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-equivalence*:
fixes *elem*::$'a$
and *Ps*::$'a$ *set list list*
assumes *distinct*: $\forall$ $P \in$ *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* ‘ ($\bigcup$ $x \in$ (*coarser-partitions-with-list elem*) ‘ (*set Ps*) . *set x*) **by** *simp*
also have ... = *set* ‘ ($\bigcup$ $x \in$ { *coarser-partitions-with-list elem P* | *P* . $P \in$ *set Ps* } . *set x*)
by (*simp add*: *image-Collect-mem*)
also have ... = $\bigcup$ { *set* (*map set* (*coarser-partitions-with-list elem P*)) | *P* . $P \in$ *set Ps* } **by** *auto*
also have ... = $\bigcup$ { *coarser-partitions-with elem* (*set P*) | *P* . $P \in$ *set Ps* }
using *distinct coarser-partitions-with-list-equivalence* **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) in form of a set

fun *all-partitions-set* :: $'a$ *list* $\Rightarrow 'a$ *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) in form of a list

fun *all-partitions-list* :: $'a$ *list* $\Rightarrow 'a$ *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*: $ps \in set\ (coarser\text{-}partitions\text{-}with\text{-}list\ x\ Q)$
    and *distinct*: *distinct Q*
    and *partition*: *is-non-overlapping* (*set Q*)
    and *new*: $\{x\} \notin set\ Q$
  shows *distinct ps*
proof −
  have $set\ (coarser\text{-}partitions\text{-}with\text{-}list\ x\ Q) = insert\ (\{x\}\ \#\ Q)\ (set\ (map\ (insert\text{-}into\text{-}member\text{-}list\ x\ Q)\ Q))$
    unfolding *coarser-partitions-with-list-def* **by** *simp*
  with *ps-coarser* **have** $ps \in insert\ (\{x\}\ \#\ Q)\ (set\ (map\ ((insert\text{-}into\text{-}member\text{-}list\ x\ Q))\ Q))$ **by** *blast*
  then show *?thesis*
  proof
    assume $ps = \{x\}\ \#\ Q$
    with *distinct* **and** *new* **show** *?thesis* **by** *simp*
  next
    assume $ps \in set\ (map\ (insert\text{-}into\text{-}member\text{-}list\ x\ Q)\ Q)$
    then obtain *X* **where** *X-in-Q*: $X \in set\ Q$ **and** *ps-insert*: $ps = insert\text{-}into\text{-}member\text{-}list\ x\ Q\ X$ **by** *auto*
    from *ps-insert* **have** $ps = (X \cup \{x\})\ \#\ (remove1\ X\ Q)$ **unfolding** *insert-into-member-list-def* **.**
    also have $\ldots = (X \cup \{x\})\ \#\ (removeAll\ X\ Q)$ **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-non-overlapping-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-in-non-overlapping remove-code*(*1*))
    qed
    with *ps-list distinct* **show** *?thesis* **by** (*metis* (*full-types*) *distinct.simps*(*2*) *distinct-removeAll*)
  qed
qed

The classical definition *all-partitions* and the algorithmic (constructive) definition *all-partitions-list* are equivalent.

lemma *all-partitions-equivalence′*:
  fixes *xs*::′*a list*
  shows $distinct\ xs \Longrightarrow$
    $((set\ (map\ set\ (all\text{-}partitions\text{-}list\ xs)) =$
    $all\text{-}partitions\ (set\ xs)) \wedge (\forall\ ps \in set\ (all\text{-}partitions\text{-}list\ xs)\ .\ distinct\ ps))$

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 ∀ *ps* ∈ *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*: ∀ *ps* ∈ *set* (*all-partitions-list xs*) . *distinct ps* **by** *simp*

  have *distinct-xs*: *distinct xs* **using** *Cons.prems* **by** *simp*
  have *x-notin-xs*: *x* ∉ *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*: ⋃ *?P-without-x* = ⋃ *P* − {*x*} **using** *partition-without-covers*
.

    assume *P* ∈ *all-partitions* (*set* (*x* # *xs*))
   then have *is-partition-of*: *P partitions* (*set* (*x* # *xs*)) **unfolding** *all-partitions-def*
..
   then have *is-non-overlapping*: *is-non-overlapping P* **unfolding** *is-partition-of-def*
by *simp*
   from *is-partition-of* **have** *P-covers*: ⋃ *P* = *set* (*x* # *xs*) **unfolding** *is-partition-of-def*
by *simp*

    have *?P-without-x partitions* (*set xs*)
      unfolding *is-partition-of-def*
     using *is-non-overlapping non-overlapping-without-is-non-overlapping 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* ∈ *set* (*map set* (*all-partitions-list xs*))
      unfolding *all-partitions-def* **by** *fast*
    have *P* ∈ *coarser-partitions-with x ?P-without-x*
      using *coarser-partitions-inv-without is-non-overlapping P-covers*
      by (*metis List.set-simps*(*2*) *insertI1*)
    then have *P* ∈ ⋃ (*coarser-partitions-with x* ‘ *set* (*map set* (*all-partitions-list*
xs)))
      using *p-list* **by** *blast*
    then have *P* ∈ *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-equivalence 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-equivalence* **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 Q *partitions* (*set xs*) **unfolding** *all-partitions-def* **..**
then have *is-non-overlapping* Q **and** *Q-covers*: $\bigcup$ Q = *set xs*
unfolding *is-partition-of-def* **by** *simp-all*
then have *P-partition*: *is-non-overlapping* P
using *non-overlapping-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 P *partitions* (*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-non-overlapping* (*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 classical definition *all-partitions* and the algorithmic (constructive) definition *all-partitions-list* are equivalent. This is a front-end 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-equivalence'* **by** *blast*

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

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

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 = (λ*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 ⋀*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*]:
  $m \leq n \longleftrightarrow$ (*of-nat m* :: *integer*) $\leq$ *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* $\circ$ *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* $\Rightarrow$ *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* ⇒ $'a$ ⇒ $'b$ ⇒ $'b$
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 λ functions.

definition *runiq* :: ($'a \times 'b$) *set* ⇒ *bool*
where *runiq R* = (∀ *X* . *trivial X* ⟶ *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 $R \mathbin{||} X = (X \times \mathit{Range}\ R) \cap R$

extensional characterization of the pairs within a restricted relation

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

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

lemma *restrict-ext'*: $R \mathbin{||} X = \{p\ .\ \mathit{fst}\ p \in X \wedge p \in R\}$
proof $-$
have $R \mathbin{||} X = \{(x, y) \mid x\ y\ .\ x \in X \wedge (x, y) \in R\}$ **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*: $P \mathbin{||} \{\} = \{\}$
unfolding *restrict-def* **by** *simp*

A restriction is a subrelation of the original relation.

lemma *restriction-is-subrel*: $P \mathbin{||} X \subseteq P$
using *restrict-def* **by** *blast*

Restricting a relation only has an effect within its domain.

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

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

lemma *restrict-to-singleton*: $P \mathbin{||} \{x\} = \{x\} \times (P \text{ `` } \{x\})$
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* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $('a \times 'b)$ *set* (**infix** *outside* *75*)
where R *outside* $X = R - (X \times \mathit{Range}\ R)$

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

lemma *outside-reduces-domain*: *Domain* (*P* *outside* *X*) = (*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 *Domain* *R* = *insert* *x* *A*
and $x \notin A$
shows *Domain* (*R* *outside* $\{x\}$) = *A*
using *assms* *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* *outside* *X*) ∪ (*P* || *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*) ⊆ *R* `` (*Domain* *R* − *X*)
using *Outside-def* *Image-def* *Domain-def* *Range-def* **by** *blast*

Considering a relation outside some set does not enlarge its range.

lemma *Range-outside-sub*:
assumes *Range* *R* ⊆ *Y*
shows *Range* (*R* *outside* *X*) ⊆ *Y*
using *assms* *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* ∈ *R*
shows *flip* *tup* ∈ 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* ∈ *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

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 right-unique, 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 || *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 || X)
proof −
have *Domain* (P *outside* X) $\cap$ *Domain* (P || X) = $\{\}$
unfolding *Outside-def restrict-def* **by** *fast*
moreover have P = P *outside* $X \cup P$ || 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 *lm01*: *card* P = *card* (Pˆ−*1*)
 using *assms card-image flip-conv injflip* **by** *metis*

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

lemma *lm02*: *trivial* X = (X={} ∨ *card* X=*1*)
 using *cardinalityOneTheElemIdentity order-refl subset-singletonD trivial-def trivial-empty*
by (*metis*(*no-types*))

lemma *lm03*: *trivial* P = *trivial* (Pˆ−*1*)
 using *trivial-def subset-singletonD subset-refl subset-insertI cardinalityOneTheElemIdentity converse-inject*
 converse-empty lm01
 by *metis*

lemma *restrictedRange*: *Range* $(P||X) = P``X$
 unfolding *restrict-def* **by** *blast*

lemma *doubleRestriction*: $((P \mid\mid X) \mid\mid Y) = (P \mid\mid (X \cap Y))$
 unfolding *restrict-def* **by** *fast*

lemma *restrictedDomain*: *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 *rightUniqueInjectiveOnFirstImplication*:
 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 characterization of right-uniqueness: the image of a singleton set is *trivial*, i.e. an empty or a singleton set.

lemma *runiq-alt*: *runiq* $R \longleftrightarrow (\forall\ x\ .$ *trivial* $(R\ ``\ \{x\}))$
 unfolding *runiq-def*
 using *Image-empty trivial-empty-or-singleton 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 *rightUniquePair*:
 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 trivial-same* **by** *blast*

lemma *rightUniqueFunctionAfterInverse*:
 assumes *runiq* f
 shows $f``(f\hat{}{-1}``Y) \subseteq Y$

using *assms runiq-basic ImageE converse-iff subsetI* **by** (*metis*(*no-types*))

lemma *lm04*:
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 rightUniqueFunctionAfterInverse* **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 rightUniqueFunctionAfterInverse*)

lemma *lm05*:
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 *rightUniqueInjectiveOnFirst*: (*runiq P*) = (*inj-on fst P*)
using *rightUniqueInjectiveOnFirstImplication lm05* **by** *blast*

lemma *disj-Un-runiq*:
assumes *runiq P runiq Q* (*Domain P*) $\cap$ (*Domain Q*) = $\{\}$
shows *runiq* ($P \cup Q$)
using *assms rightUniqueInjectiveOnFirst fst-eq-Domain injection-union* **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 *rightUniqueRestrictedGraph*: *runiq* $\{(x,f\ x)|\ x.\ P\ x\}$
unfolding *runiq-basic* **by** *fast*

lemma *rightUniqueSetCardinality*:
assumes $x \in$ *Domain R runiq R*
shows *card* (R“$\{x\}$)=*1*

using *assms lm02 DomainE Image-singleton-iff empty-iff*
by (*metis runiq-alt*)

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 rightUniqueSetCardinality*
by (*metis eval-rel.simps cardinalityOneTheElemIdentity*)

lemma *lm06*:
assumes *trivial f*
shows *runiq f*
using *assms trivial-subset-non-empty runiq-basic snd-conv*
by *fastforce*

A singleton relation is right-unique.

corollary *runiq-singleton-rel*: *runiq* $\{(x,\ y)\}$
using *trivial-singleton lm06* **by** *fast*

The empty relation is right-unique

lemma *runiq-emptyrel*: *runiq* $\{\}$
using *trivial-empty lm06* **by** *blast*

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*)

alternative characterization 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. Note that we need runiq R in order to get a definite value for $R\ ,,\ x$

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

18.1 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 images of two disjoint sets under an injective function are disjoint.

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-converse-paste-singleton*:
  assumes *runiq* (Pˆ−*1*) $y \notin$(*Range P*)
  shows *runiq* ((P +∗ $\{(x,y)\})^{-1}$)
  (**is** *?u* (*?P*ˆ−*1*))
proof −
  have (*?P*) $\subseteq P \cup \{(x,y)\}$ **using** *assms* **by** (*metis paste-sub-Un*)
  then have *?P*ˆ−*1* $\subseteq$ Pˆ−*1* $\cup$ ($\{(x,y)\}$ˆ−*1*) **by** *blast*
  moreover have ... = Pˆ−*1* $\cup \{(y,x)\}$ **by** *fast*
  moreover have *Domain* (Pˆ−*1*) $\cap$ *Domain* $\{(y,x)\} = \{\}$ **using** *assms*(*2*) **by** *auto*
  ultimately moreover have *?u* (Pˆ−*1* $\cup \{(y,x)\}$) **using** *assms*(*1*) **by** (*metis disj-Un-runiq runiq-singleton-rel*)
  ultimately show *?thesis* **by** (*metis subrel-runiq*)
qed

19 injectivity

The following is a classical definition of the set of all injective functions from X to Y.

definition *injections* :: *'a set* ⇒ *'b set* ⇒ (*'a* × *'b*) *set set*

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

The following definition is a constructive (computational) characterization of the set of all injections X Y, represented by a list. That is, we define the list of all injective functions (represented as relations) from one set (represented as a list) to another set. We formally prove the equivalence of the constructive and the classical definition in Universes.thy.

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

lemma *Image-within-domain'*:
fixes $x\ R$
shows $(x \in \mathit{Domain}\ R) = (R\ ``\ \{x\} \neq \{\})$
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 ≥ *2* ⟹ *log n* = *Suc* (*log* (*n div 2*))
  by (*simp add*: *log.simps*)

lemma *log-twice* [*simp*]:
  n ≠ *0* ⟹ *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 $m \le n$
then show $log\ m \le log\ n$
proof (*induct m arbitrary*: *n rule*: *log.induct*)
case (*1 m*)
then have *mn2*: $m\ div\ 2 \le n\ div\ 2$ **by** *arith*
show $log\ m \le log\ n$
proof (*cases* $m < 2$)
case *True*
then have $m = 0 \lor m = 1$ **by** *arith*
then show *?thesis* **by** (*auto simp del*: *One-nat-def*)
next
case *False*
with *mn2* **have** $m \ge 2$ **and** $n \ge 2$ **by** *auto arith*
from *False* **have** *m2-0*: $m\ div\ 2 \ne 0$ **by** *arith*
with *mn2* **have** *n2-0*: $n\ div\ 2 \ne 0$ **by** *arith*
from *False 1.hyps mn2* **have** $log\ (m\ div\ 2) \le log\ (n\ div\ 2)$ **by** *blast*
with *m2-0 n2-0* **have** $log\ (2 * (m\ div\ 2)) \le log\ (2 * (n\ div\ 2))$ **by** *simp*
with *m2-0 n2-0* ‹$m \ge 2$› ‹$n \ge 2$› **show** *?thesis* **by** (*simp only*: *log-rec* [*of m*] *log-rec* [*of n*]) *simp*
qed
qed
qed

20.2 Discrete square root

definition *sqrt* :: $nat \Rightarrow nat$
where
$sqrt\ n = Max\ \{m.\ m^2 \le n\}$

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

lemma [*code*]:
$sqrt\ n = Max\ (Set.filter\ (\lambda m.\ m^2 \le n)\ \{0..n\})$

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* $\langle q * q \leq n \rangle$)
using $\langle q * q \leq n \rangle$ **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* ∈ *S then 1 else 0*)

lemma *indicator-simps*[*simp*]:
x ∈ *S* ⟹ *indicator S x* = *1*
x ∉ *S* ⟹ *indicator S x* = *0*
unfolding *indicator-def* **by** *auto*

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

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

lemma *indicator-eq-0-iff*: *indicator A x* = (*0*::-::*zero-neq-one*) ⟷ *x* ∉ *A*
by (*auto simp*: *indicator-def*)

lemma *indicator-eq-1-iff*: *indicator A x* = (*1*::-::*zero-neq-one*) ⟷ *x* ∈ *A*
by (*auto simp*: *indicator-def*)

lemma *split-indicator*: *P* (*indicator S x*) ⟷ ((*x* ∈ *S* ⟶ *P 1*) ∧ (*x* ∉ *S* ⟶ *P 0*))
unfolding *indicator-def* **by** *auto*

lemma *split-indicator-asm*: *P* (*indicator S x*) ⟷ (¬ (*x* ∈ *S* ∧ ¬ *P 1* ∨ *x* ∉ *S* ∧ ¬ *P 0*))
unfolding *indicator-def* **by** *auto*

lemma *indicator-inter-arith*: *indicator* (*A* ∩ *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* $(A \times B)$ 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* $x \Rightarrow$ *indicator* A x | *Inr* $x \Rightarrow$ *indicator* B x)
unfolding *indicator-def* **by** (*cases* x) *auto*

lemma
fixes f :: $'a \Rightarrow 'b$::*semiring-1* **assumes** *finite* A
shows *setsum-mult-indicator*[*simp*]: $(\sum x \in A.\ f\ x * \mathit{indicator}\ B\ x) = (\sum x \in A \cap B.\ f\ x)$
and *setsum-indicator-mult*[*simp*]: $(\sum x \in A.\ \mathit{indicator}\ B\ x * f\ x) = (\sum x \in A \cap B.\ f\ x)$
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 $A \Longrightarrow$
$(\sum x \in A.\ \mathit{indicator}\ (B\ x)\ (g\ x) *_R f\ x) = (\sum x \in \{x{\in}A.\ g\ x \in B\ x\}.\ f\ x$::$'a$::*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 (λi. *indicator* (A i) x :: $'a$ :: {*topological-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*:
$A \cap B = \{\} \Longrightarrow (indicator\ A\ x{::}\text{-}{::}monoid\text{-}add) + indicator\ B\ x = indicator\ (A \cup B)\ x$
unfolding *indicator-def* **by** *auto*

lemma *of-real-indicator*: $of\text{-}real\ (indicator\ A\ x) = indicator\ A\ x$
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 (λi. *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** (λi. *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

Structural induction is used in proofs on lists.

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

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

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

lemma *argmaxLemma*: *argmax f A* = { $x \in A$. *f x* = *Max* (*f* ‘ *A*) }
    using *argmax-def* **by** *simp*

lemma *lm01*: *argmax f A* = $A \cap f$ −‘ {*Max* (*f* ‘ *A*)}

by *force*

lemma *lm02*: **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 lm01 lm02*
by (*metis*(*no-types*))

The previous definition of argmax operates on sets. In the following we define a corresponding notion on lists. To this end, we start with defining a filter predicate and are looking for 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*
where *filterpositions2 P l* = [*n. n* ← [*0..<size l*], *P* (*l!n*)]

definition *maxpositions*
where *maxpositions l* = *filterpositions2* (%*x* . *x* ≥ *Max* (*set l*)) *l*

lemma *lm03*: *maxpositions l* = [*n. n*←[*0..<size l*], *l!n* ≥ *Max*(*set l*)]
using *assms* **unfolding** *maxpositions-def filterpositions2-def* **by** *fastforce*

definition *argmaxList*
where *argmaxList f l* = *map* (*nth l*) (*maxpositions* (*map f l*))

lemma *lm04*: [*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* (λ*n. if n* ∈ *set l then if P n then* [*n*] *else* [] *else* []) *l*) **by** *presburger*
qed

lemma *lm05*: [*n* . *n* <− [*0*..<*m*], *P n*] = [*n* . *n* <− [*0*..<*m*], *n* ∈ *set* [*0*..<*m*], *P n*]
  using *lm04* **by** *fast*

lemma *lm06*: **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*: ∀ *x xs*. ([*f n* . *n* <− *xs*, *Q* (*f n*)] = [*x* <− (*map f xs*). *Q x*] ⟶
    [*f n* . *n* <− (*x*#*xs*), *Q* (*f n*)] = [*x* <− (*map f* (*x*#*xs*)). *Q x*])
  using *assms* **by** *simp*

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 structInduct* **by** *fast*

lemma *lm07*: **fixes** *f l*
    shows *maxpositions* (*map f l*) =
      [*n* . *n* <− [*0*..<*size l*], *f* (*l*!*n*) ≥ *Max* (*f*‘(*set l*))]
    (**is** *maxpositions* (*?fl*) = -)
proof −
 have *maxpositions ?fl* =
 [*n*. *n* <− [*0*..<*size ?fl*], *n*∈ *set*[*0*..<*size ?fl*], *?fl*!*n* ≥ *Max* (*set ?fl*)]
 using *lm04* **unfolding** *filterpositions2-def maxpositions-def* **.**
 also have ... =
 [*n* . *n* <− [*0*..<*size l*], (*n*<*size l*), (*?fl*!*n* ≥ *Max* (*set ?fl*))] **by** *simp*
 also have ... =
 [*n* . *n* <− [*0*..<*size l*], (*n*<*size l*) ∧ (*f* (*l*!*n*) ≥ *Max* (*set ?fl*))]
 using *nth-map* **by** (*metis* (*poly-guards-query*, *hide-lams*)) **also have** ... =
 [*n* . *n* <− [*0*..<*size l*], (*n*∈ *set* [*0*..<*size l*]),(*f* (*l*!*n*) ≥ *Max* (*set ?fl*))]
 using *atLeastLessThan-iff le0 set-upt* **by** (*metis*(*no-types*))
 also have ... =
 [*n* . *n* <− [*0*..<*size l*], *f* (*l*!*n*) ≥ *Max* (*set ?fl*)] **using** *lm05* **by** *presburger*
 finally show *?thesis* **by** *auto*
qed

lemma *lm08*: **fixes** *f l*
    shows *argmaxList f l* =
      [*l*!*n* . *n* <− [*0*..<*size l*], *f* (*l*!*n*) ≥ *Max* (*f*‘(*set l*))]

unfolding *lm07 argmaxList-def* **by** (*metis lm06*)

The theorem expresses that argmaxList is the list of arguments greater equal the Max of the list.

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** *lm08* **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 f in a single point by giving a pair (a, b) so that the new function has value b with argument a.

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

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

abbreviation *singleoutside* (**infix** $--$ *75*)
  where $f\ --\ x \equiv f\ outside\ \{x\}$

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 *lm001*:
  $((P \cup Q) \mid\mid X) = ((P \mid\mid X) \cup (Q \mid\mid X))$
  unfolding *restrict-def* **using** *assms* **by** *blast*

update behaves like P +* Q (paste), but without enlarging P's Domain. update is the set theoretic equivalent of the lambda function update *fun-upd*

definition *update*
  where $update\ P\ Q = P\ +\!*\ (Q \mid\mid (Domain\ P))$
  notation *update* (**infix** $+\hat{}$ *75*)

definition *runiqer* $::\ ('a \times 'b)\ set => ('a \times 'b)\ set$
  where $runiqer\ R = \{\ (x,\ THE\ y.\ y \in R\ ``\ \{x\})\mid x.\ x \in Domain\ R\ \}$

graph is like *Graph*, but with a built-in restriction to a given set X. This makes it computable for finite X, whereas $Graph\ f \mid\mid X$ is not computable. Duplicates the eponymous definition found in *Function-Order*, which is otherwise not needed.

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

lemma *lm002*:
  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 *lm003*:
  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 *lm004*:
assumes *runiq* R
shows $R =$ *graph* (*Domain* R) (*toFunction* R)
using *assms lm002 lm003* **by** *fast*

lemma *domainOfGraph*:
runiq(*graph* X f) & *Domain*(*graph* X f)=X
unfolding *graph-def*
using *rightUniqueRestrictedGraph* **by** *fast*

abbreviation *eval-rel2* (R::('a × ('b *set*)) *set*) (x::'a) == $\bigcup$ (R``{x})
notation *eval-rel2* (**infix** ,,, *75*)

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

lemma *lm005*:
Graph f=*graph UNIV* f
unfolding *Graph-def graph-def* **by** *simp*

lemma *graphIntersection*:
graph ($X \cap Y$) $f \subseteq$ ((*graph* X f) || Y)
unfolding *graph-def*
using *Int-iff mem-Collect-eq restrict-ext subrelI* **by** *auto*

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

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

corollary *lm006*:
((P *outside* X) *outside* X) = P *outside* X
using *outsideOutside* **by** *force*

lemma *lm007*:
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 *lm008*:
$P +* Q = (P$ *outside* $(Domain\ Q)) +* Q$
using *lm007* **by** *fast*

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

lemma *lm009*:
$P = P \cup \{x\}\times P``\{x\}$
using *assms* **by** (*metis outsideUnion sup.right-idem*)

corollary *lm010*:
$R = (R$ *outside* $\{x\}) +* (\{x\} \times (R$ `` $\{x\}))$
by (*metis paste-outside-restrict restrict-to-singleton*)

lemma *lm011*:
$R \subseteq R +* (\{x\} \times (R``\{x\}))$
using *lm010 lm008 paste-def Outside-def* **by** *fast*

lemma *lm012*:
$R \supseteq R+*(\{x\} \times (R``\{x\}))$
by (*metis Un-least Un-upper1 outside-union-restrict paste-def restrict-to-singleton restriction-is-subrel*)

lemma *lm013*:
$R = R +* (\{x\} \times (R``\{x\}))$
using *lm011 lm012* **by** *force*

lemma *rightUniqueTrivialCartes*:
assumes *trivial* Y
shows *runiq* $(X \times Y)$
using *assms runiq-def Image-subset lm013 trivial-subset lm011* **by** (*metis*(*no-types*))

lemma *lm014*:
runiq $((X \times \{x\}) +* (Y \times \{y\}))$
using *assms rightUniqueTrivialCartes trivial-singleton runiq-paste2* **by** *metis*

lemma *lm015*:
$(P \mid\mid (X \cap Y)) \subseteq (P\mid\mid X)$ & 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 *lm016*:
$P \mid\mid X \subseteq (P\mid\mid(X \cup Y))$ & P *outside* $X \subseteq P$ *outside* $(X \cap Y)$
using *lm015 distrib-sup-le sup-idem le-inf-iff subset-antisym sup-commute*

by (*metis sup-ge1*)

lemma *lm017*:
P``(X ∩ Domain P) = P``X
by *blast*

lemma *cardinalityOneSubset*:
assumes *card X=1* **and** *X ⊆ Y*
shows *Union X ∈ Y*
using *assms cardinalityOneTheElemIdentity* **by** (*metis cSup-singleton insert-subset*)

lemma *cardinalityOneTheElem*:
assumes *card X=1 X ⊆ Y*
shows *the-elem X ∈ Y*
using *assms* **by** (*metis* (*full-types*) *insert-subset cardinalityOneTheElemIdentity*)

lemma *lm018*:
(*R outside X1*) *outside X2* = (*R outside X2*) *outside X1*
by (*metis outsideOutside sup-commute*)

25 ordered relations

lemma *lm019*:
assumes *card X≥1* ∀ *x∈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 *lm020*:
assumes *finite X mx ∈ X f x < f mx*
shows*x ∉ argmax f X*
using *assms not-less* **by** *fastforce*

lemma *lm021*:
assumes *finite X mx ∈ X* ∀ *x ∈ X−{mx}. f x < f mx*
shows *argmax f X ⊆ {mx}*
using *assms mk-disjoint-insert* **by** *force*

lemma *lm022*:
assumes *finite X mx ∈ X* ∀ *x ∈ X−{mx}. f x < f mx*
shows *argmax f X = {mx}*
using *assms lm021* **by** (*metis argmax-non-empty-iff equals0D subset-singletonD*)

corollary *argmaxProperty*:
(*finite X* & *mx ∈ X* & (∀ *aa ∈ X−{mx}. f aa < f mx*)) ⟶ *argmax f X = {mx}*
using *assms lm022* **by** *metis*

corollary *lm023*:
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 lm022* **by** (*metis Diff-iff insertI1*)

lemma *lm024*:
assumes $f \circ g = id$
shows *inj-on g UNIV* **using** *assms*
by (*metis inj-on-id inj-on-imageI2*)

lemma *lm025*:
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 *injectionPowerset*:
assumes *inj-on f Y X* $\subseteq$ *Y*
shows *inj-on* (*image f*) (*Pow X*)
using *assms lm025* **by** (*metis subset-inj-on*)

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

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

lemma *finestPartUnion*:
$X = \bigcup$ (*finestpart X*)
using *finestPart* **by** *auto*

lemma *lm026*:
Union $\circ$ *finestpart* = *id*
using *finestpart-def finestPartUnion* **by** *fastforce*

lemma *lm027*:
inj-on Union (*finestpart* ‘ *UNIV*)
using *assms lm026* **by** (*metis inj-on-id inj-on-imageI*)

lemma *nonEqualitySetOfSets*:
assumes $X \neq Y$
shows $\{\{x\}|\ x.\ x \in X\} \neq \{\{x\}|\ x.\ x \in Y\}$
using *assms* **by** *auto*

corollary *lm028*:
inj-on finestpart UNIV
using *nonEqualitySetOfSets finestPart* **by** (*metis* (*lifting*, *no-types*) *injI*)

lemma *unionFinestPart*:
$\{Y \mid Y.\ EX\ x.((Y \in \mathit{finestpart}\ x)\ \&\ (x \in X))\} = \bigcup(\mathit{finestpart}`X)$
by *auto*

lemma *rangeSetOfPairs*:
$\mathit{Range}\ \{(\mathit{fst\ pair},\ Y)|\ Y\ \mathit{pair}.\ Y \in \mathit{finestpart}\ (\mathit{snd\ pair})\ \&\ \mathit{pair} \in X\} =$
$\{Y.\ EX\ x.\ ((Y \in \mathit{finestpart}\ x)\ \&\ (x \in \mathit{Range}\ X))\}$
by *auto*

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

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

lemma *lm029*:
$x \in X = (\{x\} \in \mathit{finestpart}\ X)$
using *finestpart-def* **by** *force*

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

lemma *lm030*:
assumes $\bigcup\ P = X$
shows $P \subseteq \mathit{Pow}\ X$
using *assms* **by** *blast*

lemma *lm031*:
$\mathit{argmax}\ f\ \{x\} = \{x\}$
using *argmax-def* **by** *auto*

lemma *sortingSameSet*:
assumes *finite X*
shows $\mathit{set}\ (\mathit{sorted\text{-}list\text{-}of\text{-}set}\ X) = X$
using *assms* **by** *simp*

lemma *lm032*:
assumes *finite A*
shows $\mathit{setsum}\ f\ A = \mathit{setsum}\ f\ (A \cap B) + \mathit{setsum}\ f\ (A - B)$

using *assms* **by** (*metis DiffD2 Int-iff Un-Diff-Int Un-commute finite-Un setsum.union-inter-neutral*)

corollary *setsumOutside*:
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 lm032* **by** (*metis*)

lemma *lm033*:
assumes (*Domain P* $\subseteq$ *Domain Q*)
shows (*P* +∗ *Q*) = *Q*
unfolding *paste-def Outside-def* **using** *assms* **by** *fast*

lemma *lm034*:
assumes (*P* +∗ *Q*=*Q*)
shows (*Domain P* $\subseteq$ *Domain Q*)
using *assms paste-def Outside-def* **by** *blast*

lemma *lm035*:
(*Domain P* $\subseteq$ *Domain Q*) = (*P* +∗ *Q*=*Q*)
using *lm033 lm034* **by** *metis*

lemma
(*P*||(*Domain Q*)) +∗ *Q* = *Q*
by (*metis Int-lower2 restrictedDomain lm035*)

lemma *lm036*:
P||*X* = *P outside* (*Domain P* − *X*)
using *Outside-def restrict-def* **by** *fastforce*

lemma *lm037*:
(*P outside X*) $\subseteq$ *P* || ((*Domain P*)−*X*)
using *lm036 lm016* **by** (*metis Int-commute restrictedDomain outside-reduces-domain*)

lemma *lm038*:
Domain (*P outside X*) ∩ *Domain* (*Q* || *X*) = {}
using *lm036*
by (*metis Diff-disjoint Domain-empty-iff Int-Diff inf-commute restrictedDomain outside-reduces-domain restrict-empty*)

lemma *lm039*:
(*P outside X*) ∩ (*Q* || *X*) = {}
using *lm038* **by** *fast*

lemma *lm040*:
(*P outside* (*X* ∪ *Y*)) ∩ (*Q* || *X*) = {} & (*P outside X*) ∩ (*Q* || (*X* ∩ *Z*)) = {}

using *assms Outside-def restrict-def lm039 lm015* **by** *fast*

lemma *lm041*:
P outside X $=$ *P* || ((*Domain P*) $-$ *X*)
using *Outside-def restrict-def lm037* **by** *fast*

lemma *lm042*:
$R``(X-Y) = (R||X)``(X-Y)$
using *restrict-def* **by** *blast*

lemma *lm043*:
assumes $\bigcup$ *XX* $\subseteq$ *X x* $\in$ *XX x* $\neq$ {}
shows *x* $\cap$ *X* $\neq$ {}
using *assms* **by** *blast*

lemma *lm044*:
assumes $\forall$ *l* $\in$ *set L1. set L2* = *f2* (*set l*) *N*
shows *set* [*set L2. l* <$-$ *L1*] = {*f2 P N*| *P. P* $\in$ *set* (*map set L1*)}
using *assms* **by** *auto*

lemma *setVsList*:
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 *lm045*:
($\forall$ *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 *lm046*:
assumes *X* $\cap$ *Y* = {}
shows $R``X$ = (*R outside Y*)$``X$
using *assms Outside-def Image-def* **by** *blast*

lemma *lm047*:
assumes (*Range P*) $\cap$ (*Range Q*) = {} *runiq* ($P\hat{\ }{-1}$) *runiq* ($Q\hat{\ }{-1}$)
shows *runiq* (($P \cup Q$)$\hat{\ }{-1}$)
using *assms* **by** (*metis Domain-converse converse-Un disj-Un-runiq*)

lemma *lm048*:
assumes (*Range P*) $\cap$ (*Range Q*) = {} *runiq* ($P\hat{\ }{-1}$) *runiq* ($Q\hat{\ }{-1}$)
shows *runiq* ((P +* Q)$\hat{\ }{-1}$)
using *lm047 assms subrel-runiq* **by** (*metis converse-converse converse-subset-swap paste-sub-Un*)

lemma *lm049*:
assumes *runiq R*
shows *card* $(R$ `` $\{a\}) = 1 \longleftrightarrow a \in$ *Domain R*
using *assms card-Suc-eq One-nat-def*
by (*metis Image-within-domain′ Suc-neq-Zero assms rightUniqueSetCardinality*)

lemma *lm050*:
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 *lm051*:
assumes *finite X* $x >$ *Max X*
shows $x \notin X$
using *assms Max.coboundedI* **by** (*metis leD*)

lemma *lm052*:
assumes *finite A* $A \neq \{\}$
shows *Max* (*f*‘*A*) $\in$ *f*‘*A*
using *assms* **by** (*metis Max-in finite-imageI image-is-empty*)

lemma *lm053*:
argmax f A $\subseteq$ *f* −‘ {*Max* (*f* ‘ *A*)}
by *force*

lemma *lm054*:
argmax f A = *A* ∩ { *x* . *f x* = *Max* (*f* ‘ *A*) }
by *auto*

lemma *lm055*:
($x \in$ *argmax f X*) = ($x \in X$ & *f x* = *Max* (*f* ‘ *X*))
using *argmax.simps mem-Collect-eq* **by** (*metis* (*mono-tags*, *lifting*))

lemma *rangeEmpty*:
Range −‘ {{}} = {{}}
by *auto*

lemma *finitePairSecondRange*:
(∀ *pair* ∈ *R*. *finite* (*snd pair*)) = (∀ *y* ∈ *Range R*. *finite y*)
by *fastforce*

lemma *lm056*:
fst ‘ *P* = *snd* ‘ (*P*^−*1*)
by *force*

lemma *lm057*:
fst pair = *snd* (*flip pair*) & *snd pair* = *fst* (*flip pair*)

unfolding *flip-def* **by** *simp*

lemma *flip-flip2*:
flip ∘ *flip* = *id*
using *flip-flip* **by** *fastforce*

lemma *lm058*:
fst = (*snd*∘*flip*)
using *lm057* **by** *fastforce*

lemma *lm059*:
snd = (*fst*∘*flip*)
using *lm057* **by** *fastforce*

lemma *lm060*:
inj-on fst P = *inj-on* (*snd*∘*flip*) *P*
using *lm058* **by** *metis*

lemma *lm062*:
inj-on fst P = *inj-on snd* (*P*ˆ−*1*)
using *lm060 flip-conv* **by** (*metis converse-converse inj-on-imageI lm059*)

lemma *setsumPairsInverse*:
assumes *runiq* (*P*ˆ−*1*)
shows *setsum* (*f* ∘ *snd*) *P* = *setsum f* (*Range P*)
using *assms lm062 converse-converse rightUniqueInjectiveOnFirst rightUniqueInjectiveOnFirst setsum.reindex snd-eq-Range*
by *metis*

lemma *notEmptyFinestpart*:
assumes *X* ≠ {}
shows *finestpart X* ≠ {}
using *assms finestpart-def* **by** *blast*

lemma *lm063*:
assumes *inj-on g X*
shows *setsum f* (*g*‘*X*) = *setsum* (*f* ∘ *g*) *X*
using *assms* **by** (*metis setsum.reindex*)

lemma *functionOnFirstEqualsSecond*:
assumes *runiq R z* ∈ *R*
shows *R*,,(*fst z*) = *snd z*
using *assms* **by** (*metis rightUniquePair surjective-pairing*)

lemma *lm064*:
assumes *runiq R*
shows *setsum* (*toFunction R*) (*Domain R*) = *setsum snd R*
using *assms toFunction-def setsum.reindex-cong functionOnFirstEqualsSecond rightUniqueInjectiveOnFirst*

by (*metis* (*no-types*) *fst-eq-Domain*)

corollary *lm065*:
assumes *runiq* $(f||X)$
shows *setsum* (*toFunction* $(f||X)$) ($X \cap$ *Domain* f) = *setsum snd* $(f||X)$
using *assms lm064* **by** (*metis Int-commute restrictedDomain*)

lemma *lm066*:
Range (R *outside* X) = R``((*Domain* R) $- X$)
using *assms*
by (*metis Diff-idemp ImageE Range.intros Range-outside-sub-Image-Domain lm041 lm042 order-class.order.antisym subsetI*)

lemma *lm067*:
$(R||X)$ `` $X = R$``X
using *Int-absorb doubleRestriction restrictedRange* **by** *metis*

lemma *lm068*:
assumes $x \in$ *Domain* $(f||X)$
shows $(f||X)$``$\{x\} = f$``$\{x\}$
using *assms doubleRestriction restrictedRange Int-empty-right Int-iff Int-insert-right-if1 restrictedDomain*
by *metis*

lemma *lm069*:
assumes $x \in X \cap$ *Domain* f *runiq* $(f||X)$
shows $(f||X),,x = f,,x$
using *assms doubleRestriction restrictedRange Int-empty-right Int-iff Int-insert-right-if1 eval-rel.simps*
by *metis*

lemma *lm070*:
assumes *runiq* $(f||X)$
shows *setsum* (*toFunction* $(f||X)$) ($X \cap$ *Domain* f) = *setsum* (*toFunction* f) (X $\cap$ *Domain* f)
using *assms setsum.cong lm069 toFunction-def* **by** *metis*

corollary *setsumRestrictedToDomainInvariant*:
assumes *runiq* $(f||X)$
shows *setsum* (*toFunction* f) ($X \cap$ *Domain* f) = *setsum snd* $(f||X)$
using *assms lm065 lm070* **by** *fastforce*

corollary *setsumRestrictedOnFunction*:
assumes *runiq* $(f||X)$
shows *setsum* (*toFunction* $(f||X)$) ($X \cap$ *Domain* f) = *setsum snd* $(f||X)$
using *assms lm064 restrictedDomain Int-commute* **by** *metis*

lemma *cardFinestpart*:
card (*finestpart* X) = *card* X

using *finestpart-def* **by** (*metis* (*lifting*) *card-image inj-on-inverseI the-elem-eq*)

corollary *lm071*:
finestpart {} = {} & *card* ∘ *finestpart* = *card*
using *cardFinestpart finestpart-def* **by** *fastforce*

lemma *finiteFinestpart*:
finite (*finestpart X*) = *finite X*
using *finestpart-def lm071*
by (*metis card-eq-0-iff empty-is-image finite.simps cardFinestpart*)

lemma *lm072*:
finite ∘ *finestpart* = *finite*
using *finiteFinestpart* **by** *fastforce*

lemma *finestpartSubset*:
assumes $X \subseteq Y$
shows *finestpart X* ⊆ *finestpart Y*
using *assms finestpart-def* **by** (*metis image-mono*)

corollary *lm073*:
assumes $x \in X$
shows *finestpart x* ⊆ *finestpart* (⋃ *X*)
using *assms finestpartSubset* **by** (*metis Union-upper*)

lemma *lm074*:
⋃ (*finestpart* ‘ *XX*) ⊆ *finestpart* (⋃ *XX*)
using *finestpart-def lm073* **by** *force*

lemma *lm075*:
⋃ (*finestpart* ‘ *XX*) ⊇ *finestpart* (⋃ *XX*)
(**is** *?L* ⊇ *?R*)
unfolding *finestpart-def* **using** *finestpart-def* **by** *auto*

corollary *commuteUnionFinestpart*:
⋃ (*finestpart* ‘ *XX*) = *finestpart* (⋃ *XX*)
using *lm074 lm075* **by** *fast*

lemma *unionImage*:
assumes *runiq a*
shows {(*x*, {*y*})| *x y*. *y* ∈ ⋃ (*a*‘‘{*x*}) & *x* ∈ *Domain a*} =
{(*x*, {*y*})| *x y*. *y* ∈ *a*,,*x* & *x* ∈ *Domain a*}
using *assms Image-runiq-eq-eval*
by (*metis* (*lifting*, *no-types*) *cSup-singleton*)

lemma *lm076*:
assumes *runiq P*
shows *card* (*Domain P*) = *card P*
using *assms rightUniqueInjectiveOnFirst card-image* **by** (*metis Domain-fst*)

lemma *finiteDomainImpliesFinite*:
  assumes *runiq f*
  shows *finite* (*Domain f*) = *finite f*
  using *assms Domain-empty-iff card-eq-0-iff finite.emptyI lm076* **by** *metis*

lemma *sumCurry*:
  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 *lm077*:
  setsum (%*y*. *f* (*x*,*y*)) Y = *setsum f* ($\{x\}\times Y$)
  using *sumCurry Sigma-cong curry-def setsum.cong* **by** *fastforce*

corollary *lm078*:
  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 *lm079*:
  (P +∗ Q)‘‘(*Domain* $Q \cap X$) = Q‘‘(*Domain* $Q \cap X$)
  unfolding *paste-def Outside-def Image-def Domain-def* **by** *blast*

corollary *lm080*:
  (P +∗ Q)‘‘($X \cap$(*Domain* Q)) = Q‘‘X
  using *Int-commute lm079* **by** (*metis lm017*)

corollary *lm081*:
  assumes $X \cap$ (*Domain* Q) = {}
  shows (P +∗ Q) ‘‘ X = (P *outside* (*Domain* Q))‘‘ X
  using *assms paste-def* **by** *fast*

lemma *lm082*:
  assumes $X \cap Y$ = {}
  shows (P *outside* Y)‘‘X=P‘‘X
  using *assms Outside-def* **by** *blast*

corollary *lm083*:
  assumes $X \cap$ (*Domain* Q) = {}
  shows (P +∗ Q)‘‘X=P‘‘X

using *assms lm081 lm082* **by** *metis*

lemma *lm084*:
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 *cardinalityIntersectionEquality*:
assumes *finite X finite Y card X* = *card Y*
shows (*card* $(X \cap Y)$ = *card X*) = $(X = Y)$
using *assms lm084* **by** (*metis card-seteq le-iff-inf order-refl*)

lemma *lm085*:
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 *graphEqImage*:
assumes $x \in X$
shows *graph X f* ,,x = $f\ x$
unfolding *graph-def* **using** *assms lm085* **by** (*metis* (*mono-tags*) *Gr-def*)

lemma *lm086*:
Graph f ,,x = $f\ x$
using *UNIV-I graphEqImage lm005* **by** (*metis*(*no-types*))

lemma *lm087*:
toFunction (*Graph f*) = f (**is** *?L*=-)
proof –
{**fix** x **have** *?L x*=$f\ x$ **unfolding** *toFunction-def lm086* **by** *metis*}
thus *?thesis* **by** *blast*
qed

lemma *lm088*:
R outside X $\subseteq$ *R*
by (*metis outside-union-restrict subset-Un-eq sup-left-idem*)

lemma *lm089*:
Range(*f outside X*) $\supseteq$ (*Range f*)–(*f*‘‘*X*)
using *assms Outside-def* **by** *blast*

lemma *lm090*:
assumes *runiq P*
shows $(P^{-1}$‘‘((*Range P*)$-Y$)) $\cap$ $((P^{-1})$‘‘$Y)$ = $\{\}$
using *assms rightUniqueFunctionAfterInverse* **by** *blast*

lemma *lm091*:
  assumes *runiq* (P^{-1})
  shows $(P``((Domain\ P) - X)) \cap (P``X) = \{\}$
  using *assms rightUniqueFunctionAfterInverse* **by** *fast*

lemma *lm092*:
  assumes *runiq f runiq* $(f\hat{\ }{-1})$
  shows $Range(f\ outside\ X) \subseteq (Range\ f){-}(f``X)$
  using *assms Diff-triv lm091 lm066 Diff-iff ImageE Range-iff subsetI* **by** *metis*

lemma *rangeOutside*:
  assumes *runiq f runiq* $(f\hat{\ }{-1})$
  shows $Range(f\ outside\ X) = (Range\ f){-}(f``X)$
  using *assms lm089 lm092* **by** (*metis order-class.order.antisym*)

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

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

lemma *lm093*:
  assumes $R \neq \{\}$ $Domain\ R \cap X \neq \{\}$
  shows $R``X \neq \{\}$
  using *assms* **by** *blast*

lemma *lm094*:
  $R``\{\}{=}\{\}$
  by (*metis Image-empty*)

lemma *lm095*:
  $R \subseteq (Domain\ R) \times (Range\ R)$
  by *auto*

lemma *finiteRelationCharacterization*:
  (*finite* (*Domain Q*) & *finite* (*Range Q*)) = *finite Q*
  using *rev-finite-subset finite-SigmaI lm095 finite-Domain finite-Range* **by** *metis*

lemma *familyUnionFiniteEverySetFinite*:
  assumes *finite* $(\bigcup\ XX)$
  shows $\forall X \in XX.$ *finite X*
  using *assms* **by** (*metis Union-upper finite-subset*)

lemma *lm096*:
  assumes *runiq f* $X \subseteq (f\hat{\ }{-1})``Y$
  shows $f``X \subseteq Y$

using *assms rightUniqueFunctionAfterInverse* **by** (*metis Image-mono order-refl subset-trans*)

lemma *lm097*:
assumes $y \in f``\{x\}$ *runiq f*
shows $f,,x = y$
using *assms* **by** (*metis Image-singleton-iff rightUniquePair*)

26 Indicator function in set-theoretical form.

abbreviation
Outside' X f == *f outside X*

abbreviation
Chi X Y == ($Y \times \{0::nat\}$) +* ($X \times \{1\}$)
notation *Chi* (**infix** <|| *80*)

abbreviation
chii X Y == *toFunction* (*X* <|| *Y*)
notation *chii* (**infix** <| *80*)

abbreviation
chi X == *indicator X*

lemma *lm098*:
runiq (*X* <|| *Y*)
by (*rule lm014*)

lemma *lm099*:
assumes $x \in X$
shows $1 \in (X <|| Y)$ `` $\{x\}$
using *assms toFunction-def paste-def Outside-def runiq-def lm014* **by** *blast*

lemma *lm100*:
assumes $x \in Y-X$
shows $0 \in (X <|| Y)$ `` $\{x\}$
using *assms toFunction-def paste-def Outside-def runiq-def lm014* **by** *blast*

lemma *lm101*:
assumes $x \in X \cup Y$
shows (*X* <|| *Y*),,*x* = *chi X x* (**is** *?L=?R*)
using *assms lm014 lm099 lm100 lm097*
by (*metis DiffI Un-iff indicator-simps*(*1*) *indicator-simps*(*2*))

lemma *lm102*:
assumes $x \in X \cup Y$
shows (*X* <| *Y*) *x* = *chi X x*
using *assms toFunction-def lm101* **by** *metis*

corollary *lm103*:
setsum $(X <| Y)$ $(X \cup Y)$ = *setsum* (*chi* X) $(X \cup Y)$
using *lm102 setsum.cong* **by** *metis*

corollary *lm104*:
assumes $\forall x \in X.\ f\ x = g\ x$
shows *setsum* $f\ X$ = *setsum* $g\ X$
using *assms* **by** (*metis* (*poly-guards-query*) *setsum.cong*)

corollary *lm105*:
assumes $\forall x \in X.\ f\ x = g\ x$ $Y \subseteq X$
shows *setsum* $f\ Y$ = *setsum* $g\ Y$
using *assms lm104* **by** (*metis contra-subsetD*)

corollary *lm106*:
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 lm102 in-mono* **by** *metis*
thus *?thesis* **using** *lm104* **by** *blast*
qed

corollary *lm107*:
setsum (*chi* X) $(Z - X) = 0$
by *simp*

corollary *lm108*:
assumes $Z \subseteq X \cup Y$
shows *setsum* $(X <| Y)$ $(Z - X) = 0$
using *assms lm107 lm106 Diff-iff in-mono subsetI* **by** *metis*

corollary *lm109*:
assumes *finite* Z
shows *setsum* $(X <| Y)$ Z = *setsum* $(X <| Y)$ $(Z - X)$ + (*setsum* $(X <| Y)$ $(Z \cap X)$)
using *lm078 assms* **by** *blast*

corollary *lm110*:
assumes $Z \subseteq X \cup Y$ *finite* Z
shows *setsum* $(X <| Y)$ Z = *setsum* $(X <| Y)$ $(Z \cap X)$
using *assms lm078 lm108 comm-monoid-add-class.add.left-neutral* **by** *metis*

corollary *lm111*:
assumes *finite* Z
shows *setsum* (*chi* X) Z = *card* $(X \cap Z)$
using *assms setsum-indicator-eq-card* **by** (*metis Int-commute*)

corollary *lm112*:

assumes $Z \subseteq X \cup Y$ *finite* Z
shows *setsum* $(X <| \ Y)\ Z = $ *card* $(Z \cap X)$
using *assms lm111* **by** (*metis lm106 setsum-indicator-eq-card*)

corollary *subsetCardinality*:
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 lm112* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

corollary *differenceSetsumVsCardinality*:
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 lm112* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

lemma *lm113*:
int (*n*::*nat*) = *real n*
by *simp*

corollary *differenceSetsumVsCardinalityReal*:
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 lm112* **by** (*metis Int-absorb2 Un-upper1 card-infinite equalityE setsum.infinite*)

27 Lists

lemma *lm114*:
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 *lm115*:
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 *lm116*:
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 *lm117*:
  assumes P −‘ {*True*} ∩ *set l* ≠ {}
  shows ∃ *n* ∈ {*0..<size l*}. *P* (*l*!*n*)
  using *assms lm116* **by** *fastforce*

lemma *nonEmptyListFiltered*:
  assumes P −‘ {*True*} ∩ *set l* ≠ {}
  shows [*n*. *n* ← [*0..<size l*], *P* (*l*!*n*)] ≠ []
  using *assms filterpositions2-def lm117 lm114* **by** *metis*

lemma *lm118*:
  (*nth l*) ‘ *set* ([*n*. *n* ← [*0..<size l*], (%*x*. *x*∈*X*) (*l*!*n*)]) ⊆ *X*∩*set l*
  by *force*

corollary *lm119*:
  (*nth l*)‘ *set* (*filterpositions2* (%*x*.(*x*∈*X*)) *l*) ⊆ *X* ∩ *set l*
  unfolding *filterpositions2-def* **using** *lm118* **by** *fast*

lemma *lm120*:
  (*n*∈{*0..<N*}) = ((*n*::*nat*) < *N*)
  using *atLeast0LessThan lessThan-iff* **by** *metis*

lemma *lm121*:
  assumes *X* ⊆ {*0..<size list*}
  shows (*nth list*)‘*X* ⊆ *set list*
  using *assms atLeastLessThan-def atLeast0LessThan lessThan-iff* **by** *auto*

lemma *lm122*:
  set ([*n*. *n* ← [*0..<size l*], *P* (*l*!*n*)]) ⊆ {*0..<size l*}
  by *force*

lemma *lm123*:
  set (*filterpositions2 pre list*) ⊆ {*0..<size list*}
  using *filterpositions2-def lm122* **by** *metis*

27.1 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 P list == *map* (*nth list*) (*filterpositions2 P list*)

lemma *permutationNotEmpty*:
  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 *lm124*:
  set (*takeAll P list*) = ((*nth list*) ‘ *set* (*filterpositions2 P list*))
  by *simp*

corollary *listIntersectionWithSet*:
  set (*takeAll* (%*x*.(*x*∈*X*)) *l*) ⊆ (*X* ∩ *set l*)
  using *lm119 lm124* **by** *metis*

corollary *lm125*:
  set (*takeAll P list*) ⊆ *set list*
  using *lm123 lm124 lm121* **by** *metis*

lemma *lm126*:
  set (*insertAt x l n*) = {*x*} ∪ *set l*
  by *simp*

lemma *lm127*:
  ∀ *n*. *set* (*perm2* [] *n*) = *set* []
  by *simp*

lemma *lm128*:
  assumes ∀ *n*. (*set* (*perm2 l n*) = *set l*)
  shows *set* (*perm2* (*x*#*l*) *n*) = {*x*} ∪ *set l*
  using *assms perm2-def lm126* **by** *force*

corollary *permutationInvariance*:
  ∀ *n*. *set* (*perm2* (*l*::'*a list*) *n*) = *set l*
proof (*induct l*)

let *?P* = *%l::('a list)*. (∀ *n*. *set* (*perm2 l n*) = *set l*)
show *?P* [] **using** *lm127* **by** *force*
fix *x* **fix** *l*
assume *?P l* **then**
show *?P* (*x#l*) **by** *force*
qed

corollary *takeAllPermutation*:
set (*perm2* (*takeAll* (*%x.*(*x*∈*X*)) *l*) *n*) ⊆ *X* ∩ *set l*
using *listIntersectionWithSet permutationInvariance* **by** *metis*

28 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*

lemma *lm129*:
assumes *runiq f x* ∈ *Domain f*
shows (*f Else 0*) *x* = (*toFunction f*) *x*
using *assms* **by** (*metis Image-runiq-eq-eval toFunction-def*)

lemma *lm130*:
assumes *runiq f*
shows *setsum* (*f Else 0*) (*X*∩(*Domain f*)) = *setsum* (*toFunction f*) (*X*∩(*Domain f*))
using *assms setsum.cong lm129* **by** *fastforce*

lemma *lm131*:
assumes *Y* ⊆ *f*−‘{*0*}
shows *setsum f Y* = *0*
using *assms* **by** (*metis set-rev-mp setsum.neutral vimage-singleton-eq*)

lemma *lm132*:
assumes *Y* ⊆ *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 lm078 lm131*
by (*metis*(*no-types*))

lemma *lm133*:
−(*Domain f*) ⊆ (*f Else 0*)−‘{*0*}
by *fastforce*

corollary *lm134*:
  assumes *finite X*
  shows *setsum* (*f Else 0*) *X* = *setsum* (*f Else 0*) (*X*∩*Domain f*)
proof −
  have *X*∩*Domain f*=*X*−(−*Domain f*) **by** *simp*
  thus *?thesis* **using** *assms lm133 lm132* **by** *fastforce*
qed

corollary *lm135*:
  assumes *finite X*
  shows *setsum* (*f Else 0*) (*X*∩*Domain f*) = *setsum* (*f Else 0*) *X*
  (**is** *?L*=*?R*)
proof −
  have *?R*=*?L* **using** *assms* **by** (*rule lm134*)
  thus *?thesis* **by** *simp*
qed

corollary *lm136*:
  assumes *finite X runiq f*
  shows *setsum* (*f Else 0*) *X* = *setsum* (*toFunction f*) (*X*∩*Domain f*)
  (**is** *?L*=*?R*)
proof −
  have *?R* = *setsum* (*f Else 0*) (*X*∩*Domain f*) **using** *assms*(*2*) *lm130* **by** *fastforce*
  moreover have ... = *?L* **using** *assms*(*1*) **by** (*rule lm135*)
  ultimately show *?thesis* **by** *presburger*
qed

lemma *lm137*:
  setsum (*f Else 0*) *X* = *setsum′ f X*
  by *fast*

corollary *lm138*:
  assumes *finite X runiq f*
  shows *setsum* (*toFunction f*) (*X*∩*Domain f*) = *setsum′ f X*
  using *assms lm137 lm136* **by** *fastforce*

lemma *lm139*:
  argmax (*setsum′ b*) = (*argmax* ∘ *setsum′*) *b*
  by *simp*

29 cardinalities of sets.

lemma *lm140*:
  assumes *runiq R runiq* (*R*^−*1*)
  shows (*R*``*A*) ∩ (*R*``*B*) = *R*``(*A*∩*B*)
  using *assms rightUniqueInjectiveOnFirst converse-Image* **by** *force*

lemma *intersectionEmptyRelationIntersectionEmpty*:

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 *lm141*:
assumes *runiq f trivial Y*
shows *trivial* $(f \text{ `` } (f\hat{}{-1} \text{ `` } Y))$
using *assms* **by** (*metis rightUniqueFunctionAfterInverse trivial-subset*)

lemma *lm142*:
assumes *trivial X*
shows *card* $(Pow\ X) \in \{1,2\}$
using *trivial-empty-or-singleton card-Pow Pow-empty assms trivial-implies-finite cardinalityOneTheElemIdentity power-one-right the-elem-eq*
by (*metis insert-iff*)

lemma *lm143*:
assumes *card* $(Pow\ A) = 1$
shows $A = \{\}$
using *assms* **by** (*metis Pow-bottom Pow-top cardinalityOneTheElemIdentity singletonD*)

lemma *lm144*:
$(\neg\ (finite\ A)) = (card\ (Pow\ A) = 0)$
by *auto*

corollary *lm145*:
$(finite\ A) = (card\ (Pow\ A) \neq 0)$
using *lm144* **by** *metis*

lemma *lm146*:
assumes *card* $(Pow\ A) \neq 0$
shows *card* $A = Discrete.log\ (card\ (Pow\ A))$
using *assms log-exp card-Pow* **by** (*metis card-infinite finite-Pow-iff*)

lemma *lm147*:
assumes *card* $(Pow\ A) = 2$
shows *card* $A = 1$
using *assms lm146*
by (*metis*(*no-types*) *comm-semiring-1-class.normalizing-semiring-rules*(*33*) *log-exp zero-neq-numeral*)

lemma *lm148*:
assumes *card* $(Pow\ X) = 1 \lor card\ (Pow\ X) = 2$
shows *trivial X*
using *assms trivial-empty-or-singleton lm143 lm147 cardinalityOneTheElemIdentity* **by** *metis*

lemma *lm149*:
trivial A = (*card* (*Pow A*) $\in$ {*1*,*2*})
using *lm148 lm142* **by** *blast*

lemma *lm150*:
assumes *R* $\subseteq$ *f runiq f Domain f* = *Domain R*
shows *runiq R*
using *assms* **by** (*metis subrel-runiq*)

lemma *lm151*:
assumes *f* $\subseteq$ *g runiq g Domain f* = *Domain g*
shows *g* $\subseteq$ *f*
using *assms Domain-iff contra-subsetD runiq-wrt-ex1 subrelI*
by (*metis* (*full-types*,*hide-lams*))

lemma *lm152*:
assumes *R* $\subseteq$ *f runiq f Domain f* $\subseteq$ *Domain R*
shows *f* = *R*
using *assms lm151* **by** (*metis Domain-mono dual-order.antisym*)

lemma *lm153*:
graph X f = (*Graph f*) || *X*
using *inf-top.left-neutral lm005 domainOfGraph restrictedDomain lm152 graphIntersection restriction-is-subrel subrel-runiq subset-iff*
by (*metis* (*erased*, *lifting*))

lemma *lm154*:
graph (*X* $\cap$ *Y*) *f* = (*graph X f*) || *Y*
using *doubleRestriction lm153* **by** *metis*

lemma *restrictionVsIntersection*:
{(*x*, *f x*)| *x*. *x* $\in$ *X2*} || *X1* = {(*x*, *f x*)| *x*. *x* $\in$ *X2* $\cap$ *X1*}
using *graph-def lm154* **by** *metis*

lemma *lm155*:
assumes *runiq f X* $\subseteq$ *Domain f*
shows *graph X* (*toFunction f*) = (*f*||*X*)
proof −
have $\bigwedge v\ w.\ (v{::}'a\ set) \subseteq w \longrightarrow w \cap v = v$ **by** (*simp add*: *Int-commute inf.absorb1*)
thus *graph X* (*toFunction f*) = *f* || *X* **by** (*metis assms*(*1*) *assms*(*2*) *doubleRestriction lm004 lm153*)
qed

lemma *lm156*:
(*Graph f*) ‘‘ *X* = *f* ‘ *X*
unfolding *Graph-def image-def* **by** *auto*

lemma *lm157*:

assumes $X \subseteq$ *Domain f runiq f*
shows $f``X = (\textit{eval-rel } f)`X$
using *assms lm156* **by** (*metis restrictedRange lm153 lm155 toFunction-def*)

lemma *cardOneImageCardOne*:
assumes *card* $A = 1$
shows *card* $(f`A) = 1$
using *assms card-image card-image-le*
proof −
have *finite* $(f`A)$ **using** *assms One-nat-def Suc-not-Zero card-infinite finite-imageI*

by (*metis*(*no-types*))
moreover have $f`A \neq \{\}$ **using** *assms* **by** *fastforce*
moreover have *card* $(f`A) \leq 1$ **using** *assms card-image-le One-nat-def Suc-not-Zero card-infinite*
by (*metis*)
ultimately show *?thesis* **by** (*metis assms image-empty image-insert cardinalityOneTheElemIdentity the-elem-eq*)
qed

lemma *cardOneTheElem*:
assumes *card* $A = 1$
shows *the-elem* $(f`A) = f$ (*the-elem* A)
using *assms image-empty image-insert the-elem-eq* **by** (*metis cardinalityOneTheElemIdentity*)

abbreviation
swap f == *curry* ((*split f*) ∘ *flip*)

lemma *lm158*:
finite X = ($X \in$ *range set*)
by (*metis List.finite-set finite-list image-iff rangeI*)

lemma *lm159*:
finite = (%X. $X \in$*range set*)
using *lm158* **by** *metis*

lemma *lm160*:
swap f = (%x. %y. $f\ y\ x$)
by (*metis comp-eq-dest-lhs curry-def flip-def fst-conv old.prod.case snd-conv*)

30 some easy properties on real numbers

lemma *lm161*:
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 *lm162*:
fixes a::*real*
fixes b c
shows $a*b - c*b=(a-c)*b$
using *assms lm161* **by** (*metis comm-semiring-1-class.normalizing-semiring-rules*(*7*))

end

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

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

begin

32 Types

type-synonym *index* = *integer*
type-synonym *participant* = *index*
type-synonym *good* = *nat*
type-synonym *goods* = *good 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 *bidvector* = (*participant* × *goods*) ⇒ *price*
abbreviation *bidvector* (*b*::*bids*) == *split b*
abbreviation *proceeds* (*b*::*bidvector*) (*allo*::*allocation*) == *setsum b allo*
abbreviation *winnersOfAllo* (*a*::*allocation*) == *Domain a*
abbreviation *allocatedGoods* (*allo*::*allocation*) == $\bigcup$ (*Range allo*)

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

abbreviation $\textit{is-partition-of}'\ P\ A == (\bigcup\ P = A \wedge \textit{is-non-overlapping}\ P)$
abbreviation $\textit{all-partitions}'\ A == \{P\ .\ \textit{is-partition-of}'\ P\ A\}$
abbreviation $\textit{injections}'\ X\ Y == \{R\ .\ \textit{Domain}\ R = X \wedge \textit{Range}\ R \subseteq Y \wedge \textit{runiq}\ R \wedge \textit{runiq}\ (R^{-1})\}$
abbreviation $\textit{possible-allocations-rel}'\ G\ N == \textit{Union}\{\textit{injections}'\ Y\ N \mid Y\ .\ Y \in \textit{all-partitions}'\ G\}$
abbreviation *possibleAllocationsRel* **where**
possibleAllocationsRel N G == converse ‘ (possible-allocations-rel G N)

algorithmic version of *possible-allocations-rel*

fun *possible-allocations-alg* :: *goods* ⇒ *participant set* ⇒ *allocation-rel list*
where *possible-allocations-alg G N* =
concat [*injections-alg Y N* . *Y* ← *all-partitions-alg G*]

abbreviation *possibleAllocationsAlg N G* ==
map converse (*concat* [(*injections-alg l N*) . *l* ← *all-partitions-list G*])

33 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*) (*possibleAllocationsAlg N G*)

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

payments

alpha is 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 when solved without n's bids

abbreviation *alpha N G b n* == *Max* ((*setsum b*)‘(*possibleAllocationsRel* ($N-\{n\}$) *G*))

abbreviation *alphaAlg N G b n* == *Max* ((*proceeds b*)‘(*set* (*possibleAllocationsAlg* ($N-\{n\}$) (*G*::- *list*))))

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

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

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

definition *paymentsAlg N G t* == (*alphaAlg N G*) − (*remainingValueAlg N G t*)

end

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

theory *Universes*

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

begin

35 Preliminary lemmas

lemma *lm63*: **assumes** $Y \in$ *set* (*all-partitions-alg X*) **shows** *distinct Y*
using *assms distinct-sorted-list-of-set all-partitions-alg-def all-partitions-equivalence′*
by *metis*

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

36 Definitions of various subsets of *UNIV*.

abbreviation *isChoice R* == $\forall x.\ R``\{x\} \subseteq x$
abbreviation *partitionsUniverse* == {*X*. *is-non-overlapping X*}
lemma *partitionsUniverse* ⊆ *Pow UNIV*
by *simp*

abbreviation *partitionValuedUniverse* == $\bigcup$ *P* ∈ *partitionsUniverse*. *Pow* (*UNIV* × *P*)
lemma *partitionValuedUniverse* ⊆ *Pow* (*UNIV* × (*Pow UNIV*))

by *simp*

abbreviation *injectionsUniverse* == $\{R.\ (runiq\ R)\ \&\ (runiq\ (R\hat{}{-1}))\}$

abbreviation *allocationsUniverse* == *injectionsUniverse* ∩ *partitionValuedUniverse*
abbreviation *totalRels X Y* == $\{R.\ Domain\ R = X\ \&\ Range\ R \subseteq Y\}$

37 Results about the sets defined in the previous section

lemma *lm04*: **assumes** $\forall\ x1 \in X.\ (x1 \neq \{\}\ \&\ (\forall\ x2 \in X-\{x1\}.\ x1 \cap x2 = \{\}))$
shows *is-non-overlapping X*
unfolding *is-non-overlapping-def* **using** *assms* **by** *fast*

lemma *lm72*: **assumes** $\forall x \in X.\ f\ x \in x$
shows *isChoice* (*graph X f*) **using** *assms*
by (*metis Image-within-domain′ empty-subsetI insert-subset graphEqImage domainOfGraph*
runiq-wrt-eval-rel subset-trans)

lemma *lm24*: *injections* = *injections′* **using** *injections-def* **by** (*metis*(*no-types*))
lemma *lm25*: *injections′ X Y* ⊆ *injectionsUniverse* **by** *fast*
lemma *lm25b*: *injections X Y* ⊆ *injectionsUniverse* **using** *injections-def* **by** *blast*
lemma *lm26*: *injections′ X Y* = *totalRels X Y* ∩ *injectionsUniverse* **by** *fastforce*

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

lemma *lll80*: **assumes** *is-non-overlapping XX YY* ⊆ *XX*
shows (*XX* − *YY*) *partitions* (⋃ *XX* − ⋃ *YY*)
proof −
let *?xx=XX − YY* **let** *?X=⋃ XX* **let** *?Y=⋃ YY*
let *?x=?X − ?Y*
have ∀ *y* ∈ *YY*. ∀ *x*∈*?xx*. *y* ∩ *x*={} **using** *assms is-non-overlapping-def* **by** (*metis Diff-iff set-rev-mp*)
then have ⋃ *?xx* ⊆ *?x* **using** *assms* **by** *blast*
then have ⋃ *?xx* = *?x* **by** *blast*
moreover have *is-non-overlapping ?xx* **using** *subset-is-non-overlapping* **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 $a \in$ *possible-allocations-rel G N*
proof −
have $a \in$ *injections* (*Domain a*) *N* **unfolding** *injections-def* **using** *assms*(*1*) *assms*(*2*) *assms*(*4*) **by** *blast*
moreover have *Domain a* $\in$ *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*: $a \in$ *possible-allocations-rel G N* $\longleftrightarrow$
runiq a & *runiq* (a^{-1}) & (*Domain a*) *partitions G* & *Range a* $\subseteq$ *N*
using *lll81a lll81b* **by** *blast*

lemma *lm10*: *possible-allocations-rel'* *G N* $\subseteq$ *injectionsUniverse*
using *assms* **by** *force*

lemma *lm09*: *possible-allocations-rel G N* $\subseteq$ $\{a.$ (*Range a*) $\subseteq$ *N* & (*Domain a*) $\in$ *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*=$\bigcup$ { *injections Y B* | *Y* . $Y \in$ *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 *lm17a*: *possible-allocations-rel G N* $\subseteq$
injectionsUniverse $\cap$ {*a. Range a* $\subseteq$ *N* & *Domain a* $\in$ *all-partitions G*}
using *assms lm09 lm10 possible-allocations-rel-def injections-def* **by** *fastforce*

lemma *lm17b*: *possible-allocations-rel G N* $\supseteq$
injectionsUniverse $\cap$ {*a. Domain a* $\in$ *all-partitions G* & *Range a* $\subseteq$ *N*}
using *possible-allocations-rel-def injections-def* **by** *auto*

lemma *lm17*: *possible-allocations-rel G N* =
injectionsUniverse $\cap$ {*a. Domain a* $\in$ *all-partitions G* & *Range a* $\subseteq$ *N*}
using *lm17a lm17b* **by** *blast*

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

lemma *lm18*: *converse‘*(*A* $\cap$ *B*) = (*converse‘A*) $\cap$ (*converse‘B*)
by *force*

lemma *lm19*: *possibleAllocationsRel N G* =
injectionsUniverse $\cap$ {*a. Domain a* $\subseteq$ *N* & *Range a* $\in$ *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*) $\cap$ *?c‘* ({*a. ?r a* $\subseteq$ *N* & *?d a* $\in$ *?P*}) **using** *lm17* **by** *fastforce*
moreover have ... = (*?c‘?I*) $\cap$ {*aa. ?d aa* $\subseteq$ *N* & *?r aa* $\in$ *?P*} **by** *fastforce*
moreover have ... = *?I* $\cap$ {*aa. ?d aa* $\subseteq$ *N* & *?r aa* $\in$ *?P*} **using** *lm16* **by** *metis*
ultimately show *?thesis* **by** *presburger*
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-non-overlapping-def* **by** *blast*

corollary *lm50*: *possibleAllocationsRel N G* $\subseteq$ *allocationsUniverse*
using *lm48 lm49* **by** (*metis* (*lifting*, *mono-tags*) *inf.bounded-iff*)

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

corollary *lm19d*: **assumes** $a \in$ *possibleAllocationsRel N1 G*
shows $a \in$ *possibleAllocationsRel* (*N1* $\cup$ *N2*) *G*
proof –
have *Domain a* $\subseteq$ *N1* $\cup$ *N2* **using** *assms*(*1*) *lm19c* **by** (*metis le-supI1*)
moreover have $a \in$ *injectionsUniverse* & *Range a* $\in$ *all-partitions G*
using *assms lm19c* **by** *blast* **ultimately show** *?thesis* **using** *lm19c* **by** *blast*
qed

corollary *lm19b*: *possibleAllocationsRel N1 G* $\subseteq$ *possibleAllocationsRel* (*N1* $\cup$ *N2*) *G*
using *lm19d* **by** (*metis subsetI*)

lemma *lm20d*: **assumes** ($\bigcup$ *P1*) $\cap$ ($\bigcup$ *P2*) = {}
is-non-overlapping P1 is-non-overlapping P2
$X \in$ *P1* $\cup$ *P2* $Y \in$ *P1* $\cup$ *P2* $X \cap Y \neq$ {}
shows ($X = Y$)
unfolding *is-non-overlapping-def* **using** *assms is-non-overlapping-def*
by *fast*

lemma *lm20e*: **assumes** ($\bigcup$ *P1*) $\cap$ ($\bigcup$ *P2*) = {}
is-non-overlapping P1 is-non-overlapping P2
$X \in$ *P1* $\cup$ *P2* $Y \in$ *P1* $\cup$ *P2* ($X = Y$)
shows $X \cap Y \neq$ {}
unfolding *is-non-overlapping-def* **using** *assms is-non-overlapping-def*
by *fast*

lemma *lm20*: **assumes** ($\bigcup$ *P1*) $\cap$ ($\bigcup$ *P2*) = {} *is-non-overlapping P1 is-non-overlapping P2*
shows *is-non-overlapping* (*P1* $\cup$ *P2*)
unfolding *is-non-overlapping-def* **using** *assms lm20d lm20e* **by** *metis*

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

lemma *lll77c*: **assumes** *a1* $\in$ *injectionsUniverse a2* $\in$ *injectionsUniverse*
(*Range a1*) $\cap$ (*Range a2*)={} (*Domain a1*) $\cap$ (*Domain a2*) = {}
shows *a1* $\cup$ *a2* $\in$ *injectionsUniverse*
using *assms disj-Un-runiq*
by (*metis* (*no-types*) *Domain-converse converse-Un mem-Collect-eq*)

lemma *lm22*: **assumes** $R \in$ *partitionValuedUniverse*
shows *is-non-overlapping* (*Range R*)
proof –
obtain *P* **where**
0: $P \in$ *partitionsUniverse* & $R \subseteq$ *UNIV* $\times$ *P* **using** *assms* **by** *blast*
have *Range R* $\subseteq$ *P* **using** *0* **by** *fast*
then show *?thesis* **using** *0 mem-Collect-eq subset-is-non-overlapping* **by** (*metis*)
qed

lemma *lm23*: **assumes** *a1* ∈ *allocationsUniverse a2* ∈ *allocationsUniverse*
$(\bigcup$ *(Range a1)*$) \cap (\bigcup$ *(Range a2)*$) = \{\}$
(Domain a1) ∩ *(Domain a2)* = {}
shows *a1* ∪ *a2* ∈ *allocationsUniverse*
proof −
let *?a=a1* ∪ *a2* **let** *?b1=a1ˆ−1* **let** *?b2=a2ˆ−1* **let** *?r=Range* **let** *?d=Domain*
let *?I=injectionsUniverse* **let** *?P=partitionsUniverse* **let** *?PV=partitionValuedUniverse*
let *?u=runiq*
let *?b=?aˆ−1* **let** *?p=is-non-overlapping*
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* $(\bigcup$ (*?r a1*) ∩ $(\bigcup$ (*?r a2*))) **by** *auto*
ultimately moreover have {} ∉ (*?r a1*) & {} ∉ (*?r a2*) **using** *is-non-overlapping-def*
by (*metis Int-empty-left*)
ultimately moreover have *?r a1* ∩ (*?r a2*) = {} **using** *assms lm22 is-non-overlapping-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*} =
(*Domain* −‘(*Pow N*)) ∩ (*Range* −‘ (*all-partitions G*))
by *fastforce*

lemma *lm30*: *possibleAllocationsRel N G* =
injectionsUniverse ∩ ((*Range* −‘ (*all-partitions G*)) ∩ (*Domain* −‘(*Pow*
N)))
using *lm19 lm30b* **by** (*metis* (*no-types*) *Int-commute*)

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

lemma *lm28a*: **assumes** *a* ∈ *possibleAllocationsRel N G*

shows $(a\hat{\ }{-1} \in$ *injections* $($*Range* $a)$ N &
Range $a \in$ *all-partitions* $G)$
using *assms*
by (*metis* (*mono-tags*, *hide-lams*) *lm19c lm47*)

lemma *lm28c*: **assumes** $a\hat{\ }{-1} \in$ *injections* $($*Range* $a)$ N *Range* $a \in$ *all-partitions* G
shows $a \in$ *possibleAllocationsRel* N G
using *assms image-iff* **by** *fastforce*

lemma *lm28*: $a \in$ *possibleAllocationsRel* N G $=$
$(a\hat{\ }{-1} \in$ *injections* $($*Range* $a)$ N & *Range* $a \in$ *all-partitions* $G)$
using *lm28a lm28c*
by *metis*

lemma *lm28d*: **assumes** $a \in$ *possibleAllocationsRel* N G
shows $a \in$ *injections* $($*Domain* $a)$ $($*Range* $a)$ &
Range $a \in$ *all-partitions* G &
Domain $a \subseteq N$
using *assms mem-Collect-eq injections-def lm19c order-refl*
by (*metis* (*mono-tags*, *lifting*))

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

lemma *lm28b*: $a \in$ *possibleAllocationsRel* N G $=$
$(a \in$ *injections* $($*Domain* $a)$ $($*Range* $a)$ &
Range $a \in$ *all-partitions* G &
Domain $a \subseteq N)$
using *lm28d lm28e*
by *metis*

lemma *lm32*: **assumes** $a \in$ *partitionValuedUniverse*
shows $a - b \in$ *partitionValuedUniverse*
using *assms subset-is-non-overlapping*
by *fast*

lemma *lm35*: **assumes** $a \in$ *allocationsUniverse*
shows $a - b \in$ *allocationsUniverse*
using *assms lm27 lm32*
by *auto*

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

lemma *lm34*: **assumes** *a* ∈ *allocationsUniverse*
shows *a* ∈ *possibleAllocationsRel* (*Domain a*) (⋃ (*Range a*))
proof −
let *?r*=*Range* **let** *?p*=*is-non-overlapping* **let** *?P*=*all-partitions* **have** *?p* (*?r a*)
using
assms lm22 Int-iff **by** *blast* **then have** *?r a* ∈ *?P* (⋃ (*?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*} ∈ *partitionsUniverse*) = (*X* ≠ {})
using *is-non-overlapping-def*
by *fastforce*

lemma *lm36b*: {(*x*, *X*)} − {(*x*, {})} ∈ *partitionValuedUniverse*
using *lm36*
by *auto*

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

lemma *allocationUniverseProperty*: {(*x*,*X*)} − {(*x*,{})} ∈ *allocationsUniverse*
using *lm36b lm37 lm27 Int-iff*
by (*metis* (*no-types*))

lemma *lm41*: **assumes** *is-non-overlapping PP is-non-overlapping* (*Union PP*)
shows *is-non-overlapping* (*Union* ‘ *PP*)
proof −
let *?p*=*is-non-overlapping* **let** *?U*=*Union* **let** *?P2*=*?U PP* **let** *?P1*=*?U* ‘ *PP*
have
0: ∀ *X*∈*?P1*. ∀ *Y* ∈ *?P1*. (*X* ∩ *Y* = {} ⟶ *X* ≠ *Y*) **using** *assms is-non-overlapping-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* ∈ *?P1* & *Y*∈*?P1* & *X* ≠ *Y*
then obtain *XX YY* **where**
1: *X* = *?U XX* & *Y*=*?U YY* & *XX*∈*PP* & *YY*∈*PP* **by** *blast*
then have *XX* ⊆ *Union PP* & *YY* ⊆ *Union PP* & *XX* ∩ *YY* = {}
using *2 1 is-non-overlapping-def assms*(*1*) *Sup-upper* **by** *metis*
then moreover have ∀ *x*∈*XX*. ∀ *y*∈*YY*. *x* ∩ *y* = {} **using** *1 assms*(*2*) *is-non-overlapping-def*
by (*metis IntI empty-iff subsetCE*)
ultimately have *X* ∩ *Y*={} **using** *assms 0 1 2 is-non-overlapping-def* **by** *auto*

}
then show *?thesis* **using** *0 is-non-overlapping-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-non-overlapping* **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-non-overlapping* **by** *metis* **have**
$?a1 \in allocationsUniverse$ & $?a2 \in allocationsUniverse$ **using** *allocationUniverseProperty 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*
Diff-disjoint intersectionEmptyRelationIntersectionEmpty **by** *metis*
ultimately moreover have $?r\ ?a1 \cap ?Yi = \{\}$ & $?Yi \neq \{\}$ **by** *blast*
ultimately moreover have $?p\ \{?r\ ?a1,\ ?Yi\}$ **unfolding** *is-non-overlapping-def* **using**
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-non-overlapping*)
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-non-overlapping-def*
by (*metis set-rev-mp*)
ultimately have *?U* (*?r ?a1*) $\cap$ *?Y* = {} **using** *unionIntersectionEmpty*
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** *allocationUniverseProperty* **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

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

lemma *lm45*: **assumes** *Domain a* $\cap X \neq \{\}$ $a \in$ *allocationsUniverse*
shows $\bigcup$($a``X$) $\neq \{\}$
proof $-$
let *?p=is-non-overlapping* **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-non-overlapping* **by** *blast*
moreover have $a``X \neq \{\}$ **using** *assms* **by** *fast*
ultimately show *?thesis* **by** (*metis Union-member all-not-in-conv no-empty-in-non-overlapping*)
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 *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*)
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 outside* ($X \cup \{i\}$) $\cup$ $\{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\})$ $\in$ *allocationsUniverse*
using *lm43b* **by** *fastforce*
hence *a outside* ($X \cup \{i\}$) $\cup$ $\{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\})$ $\in$ *allocationsUniverse*
by (*metis 0*)
thus *a outside* ($X \cup \{i\}$) $\cup$ $\{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\})$ $\in$ *allocationsUniverse* $\wedge$ $\bigcup$ *Range* (*a outside* ($X \cup \{i\}$) $\cup$ $\{i\} \times (\{\bigcup(a `` (X \cup \{i\}))\} - \{\{\}\})$)
$= \bigcup$ *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 *bidMonotonicity b i* == ($\forall$ *t t'*. (*trivial t* & *trivial t'* & *Union t* $\subseteq$ *Union t'*) $\longrightarrow$
setsum b ($\{i\}\times t$) $\leq$ *setsum b* ($\{i\}\times t'$))

lemma *lm46*: **assumes** *bidMonotonicity 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 *lm39*: **assumes** *XX* $\in$ *partitionValuedUniverse*
shows $\{\}\notin$ *Range XX*
using *assms mem-Collect-eq no-empty-in-non-overlapping* **by** *auto*

corollary *lm39b*: **assumes** *a* $\in$ *possibleAllocationsRel N G*
shows $\{\}\notin$ *Range a*
using *assms lm39 lm50* **by** *blast*

lemma *lm40*: **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 *lm40b*: **assumes** *a* $\in$ *possibleAllocationsRel N G*
shows *Domain a* $\subseteq$ *N* & *Range a* $\subseteq$ *Pow G* $-$ $\{\{\}\}$
using *assms lm40 insert-Diff-single lm39b subset-insert lm47* **by** *metis*

corollary *lm40c*: **assumes** *a* $\in$ *possibleAllocationsRel N G*
shows *a* $\subseteq$ *N* $\times$ (*Pow G* $-$ $\{\{\}\}$)
using *assms lm40b* **by** *blast*

corollary *lm40e*: *possibleAllocationsRel N G* $\subseteq$ *Pow* ($N\times$(*Pow G*$-\{\{\}\}$))
using *lm40c* **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** *bidMonotonicity* (*b*::- => *real*) *i*
$a \in$ *allocationsUniverse*
Domain a $\cap X \neq \{\}$
finite a
shows *setsum b* (*a outside X*) $\leq$
setsum b (*a outside* $(X \cup \{i\}) \cup (\{i\} \times \{\bigcup(a``(X\cup\{i\}))\})$)
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* $(X \cup \{i\}) \cup (\{i\} \times \{?U(a``(X \cup \{i\}))\})$
have $a \in$ *injectionsUniverse* **using** *assms* **by** *fast* **then have**
0: *?u a* **by** *simp*
moreover have $?R \subseteq a$ & $?R--i \subseteq a$ **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* $(\{i\} \times (?R``\{i\}))$ **using** *runiq-def*
by (*metis trivial-cartesian trivial-singleton*)
moreover have $\forall X.$ $(?R -- i) \cap (\{i\} \times X) = \{\}$ **using** *outside-reduces-domain* **by** *force*
ultimately have
1: *finite* $(?R--i)$ & *finite* $(\{i\} \times (?R``\{i\}))$ & $(?R -- i) \cap (\{i\} \times (?R``\{i\})) = \{\}$ &
finite $(\{i\} \times \{?U(a``(X \cup \{i\}))\})$ & $(?R -- i) \cap (\{i\} \times \{?U(a``(X \cup \{i\}))\}) = \{\}$

using *Outside-def trivial-implies-finite* **by** *fast*
have $?R = (?R -- i) \cup (\{i\} \times ?R``\{i\})$ **by** (*metis outsideUnion*)
then have *setsum b ?R* = *setsum b* $(?R -- i)$ + *setsum b* $(\{i\} \times (?R``\{i\}))$
using *1 setsum.union-disjoint* **by** (*metis* (*lifting*) *setsum.union-disjoint*)
moreover have *setsum b* $(\{i\} \times (?R``\{i\})) \leq$ *setsum b* $(\{i\} \times \{?U(a``(X \cup \{i\}))\})$ **using** *lm46*
assms(*1*) *0* **by** *metis*
ultimately have *setsum b ?R* $\leq$ *setsum b* $(?R -- i)$ + *setsum b* $(\{i\} \times \{?U(a``(X \cup \{i\}))\})$ **by** *linarith*
moreover have ... = *setsum b* $(?R -- i \cup (\{i\} \times \{?U(a``(X \cup \{i\}))\}))$
using *1 setsum.union-disjoint* **by** *auto*
moreover have ... = *setsum b ?aa* **by** (*metis outsideOutside*)
ultimately show *?thesis* **by** *linarith*
qed

lemma *lm55*: **assumes** *finite X XX* $\in$ *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* $\in$ *possibleAllocationsRel N G*
shows *finite a*
using *assms finiteRelationCharacterization rev-finite-subset* **by** (*metis lm28b lm55*)

lemma *lm59*: **assumes** *finite N finite G*
shows *finite* (*possibleAllocationsRel N G*)
proof –
have *finite* $(Pow(N \times (Pow\ G - \{\{\}\})))$ **using** *assms finite-Pow-iff* **by** *blast*
then show *?thesis* **using** *lm40e rev-finite-subset* **by** (*metis*(*no-types*))
qed

corollary *lm53*: **assumes** *bidMonotonicity* (*b*::- => *real*) *i*
a ∈ *possibleAllocationsRel N G*
i ∈ *N*−*X*
Domain a ∩ *X* ≠ {}
finite N
finite G
shows *Max* ((*setsum b*)‘(*possibleAllocationsRel* (*N*−*X*) *G*)) ≥
setsum b (*a outside X*)
proof −
let *?aa*=*a outside* (*X* ∪ {*i*}) ∪ ({*i*} × {⋃(*a*‘‘(*X*∪{*i*}))})
have *bidMonotonicity* (*b*::- => *real*) *i* **using** *assms*(*1*) **by** *fast*
moreover have *a* ∈ *allocationsUniverse* **using** *assms*(*2*) *lm50* **by** *blast*
moreover have *Domain a* ∩ *X* ≠ {} **using** *assms*(*4*) **by** *auto*
moreover have *finite a* **using** *assms lm58* **by** *metis* **ultimately have**
0: *setsum b* (*a outside X*) ≤ *setsum b ?aa* **by** (*rule lm52*)
have *?aa* ∈ *possibleAllocationsRel* (*N*−*X*) (⋃ (*Range a*)) **using** *assms lm51* **by** *metis*
moreover have ⋃ (*Range a*) = *G* **using** *assms lm47 is-partition-of-def* **by** *metis*
ultimately have *setsum b ?aa* ∈ (*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-Diff finite-imageI*)
ultimately have *setsum b ?aa* ≤ *Max* ((*setsum b*)‘(*possibleAllocationsRel* (*N*−*X*) *G*)) **by** *auto*
then show *?thesis* **using** *0* **by** *linarith*
qed

lemma *cardinalityPreservation*: **assumes** *finite XX* ∀ *X* ∈ *XX*. *finite X is-non-overlapping XX*
shows *card* (⋃ *XX*) = *setsum card XX*
using *assms is-non-overlapping-def card-Union-disjoint* **by** *fast*

corollary *lm33b*: **assumes** *XX partitions X finite X finite XX*
shows *card* (⋃ *XX*) = *setsum card XX*
using *assms cardinalityPreservation* **by** (*metis is-partition-of-def familyUnionFiniteEverySetFinite*)

lemma *setsumUnionDisjoint1*: **assumes** ∀ *A*∈*C*. *finite A* ∀ *A*∈*C*. ∀ *B*∈*C*. *A* ≠ *B* ⟶ *A Int B* = {}
shows *setsum f* (*Union C*) = *setsum* (*setsum f*) *C*
using *assms setsum.Union-disjoint* **by** *fastforce*

corollary *setsumUnionDisjoint2*: **assumes** ∀ *x*∈*X*. *finite x is-non-overlapping X*
shows *setsum f* (⋃ *X*) = *setsum* (*setsum f*) *X*
using *assms setsumUnionDisjoint1 is-non-overlapping-def*
by *fast*

corollary *setsumUnionDisjoint3*: **assumes** $\forall x \in X.$ *finite x X partitions XX*
shows *setsum f XX* = *setsum* (*setsum f*) *X*
using *assms*
by (*metis is-partition-of-def setsumUnionDisjoint2*)

corollary *setsum-associativity*: **assumes** *finite x X partitions x*
shows *setsum f x* = *setsum* (*setsum f*) *X*
using *assms setsumUnionDisjoint3*
by (*metis is-partition-of-def familyUnionFiniteEverySetFinite*)

lemma *lm19e*: **assumes** $a \in$ *allocationsUniverse Domain a* $\subseteq$ *N* ($\bigcup$ *Range a*) = *G*
shows $a \in$ *possibleAllocationsRel N G*
using *assms lm19c lm34* **by** (*metis* (*mono-tags*, *lifting*))

corollary *nn24a*: (*allocationsUniverse* $\cap$ {*a*. (*Domain a*) $\subseteq$ *N* & ($\bigcup$ *Range a*) = *G*}) $\subseteq$
possibleAllocationsRel N G
using *lm19e* **by** *fastforce*

corollary *nn24f*: *possibleAllocationsRel N G* $\subseteq$ {*a*. (*Domain a*) $\subseteq$ *N*}
using *lm47* **by** *blast*

corollary *nn24g*: *possibleAllocationsRel N G* $\subseteq$ {*a*. ($\bigcup$ *Range a*) = *G*}
using *is-partition-of-def lm47 mem-Collect-eq subsetI*
by (*metis*(*mono-tags*))

corollary *nn24e*: *possibleAllocationsRel N G* $\subseteq$ *allocationsUniverse* &
possibleAllocationsRel N G $\subseteq$ {*a*. (*Domain a*) $\subseteq$ *N* & ($\bigcup$ *Range a*) = *G*}
using *nn24f nn24g conj-subset-def lm50* **by** (*metis* (*no-types*))

corollary *nn24b*: *possibleAllocationsRel N G* $\subseteq$
allocationsUniverse $\cap$ {*a*. (*Domain a*) $\subseteq$ *N* & ($\bigcup$ *Range a*) = *G*}
(**is** *?L* $\subseteq$ *?R1* $\cap$ *?R2*)
proof −
have *?L* $\subseteq$ *?R1* & *?L* $\subseteq$ *?R2* **by** (*rule nn24e*) **thus** *?thesis* **by** *auto*
qed

corollary *nn24*: *possibleAllocationsRel N G* =
(*allocationsUniverse* $\cap$ {*a*. (*Domain a*) $\subseteq$ *N* & ($\bigcup$ *Range a*) = *G*})
(**is** *?L* = *?R*)
proof −
have *?L* $\subseteq$ *?R* **using** *nn24b* **by** *metis* **moreover have** *?R* $\subseteq$ *?L* **using** *nn24a* **by** *fast*
ultimately show *?thesis* **by** *force*
qed

corollary *nn24c*: $a \in$ *possibleAllocationsRel N G* $=$
($a \in$ *allocationsUniverse* & (*Domain a*) $\subseteq N$ & ($\bigcup$ *Range a*) $= G$)
using *nn24 Int-Collect* **by** (*metis* (*mono-tags*, *lifting*))

corollary *lm35d*: **assumes** $a \in$ *allocationsUniverse*
shows *a outside X* $\in$ *allocationsUniverse*
using *assms Outside-def* **by** (*metis* (*lifting*, *mono-tags*) *lm35*)

38 Bridging theorem for injections

lemma *lm84*: *totalRels* {} *Y* = {{}}
by *fast*

lemma *lm85*: {} $\in$ *injectionsUniverse*
by (*metis CollectI converse-empty runiq-emptyrel*)

lemma *lm87*: *injectionsUniverse* $\cap$ (*totalRels* {} *Y*) = {{}}
using *lm84 lm85* **by** *fast*

lemma *lm60*: **assumes** *runiq f* $x \notin$ *Domain f*
shows { $f \cup \{(x, y)\}$ | y . $y \in A$ } $\subseteq$ *runiqs*
unfolding *paste-def runiqs-def*
using *assms runiq-basic* **by** *blast*

lemma *lm95*: *converse* ‘ (*converse* ‘ *X*) = *X*
by *auto*

lemma *lm66*: *runiq* ($f\hat{}-1$) = ($f \in$ *converse*‘*runiqs*)
unfolding *runiqs-def* **by** *auto*

lemma *lm68*: **assumes** *runiq* ($f\hat{}-1$) $A \cap$ *Range f* = {}
shows *converse* ‘ { $f \cup \{(x, y)\}$ | y . $y \in A$ } $\subseteq$ *runiqs*
using *assms lm60* **by** *fast*

lemma *lm68b*: **assumes** $f \in$*converse*‘*runiqs* $A \cap$ *Range f* = {}
shows $\{f \cup \{(x, y)\}|\ y.\ y \in A\} \subseteq$ *converse*‘*runiqs*
(**is** *?l* $\subseteq$ *?r*)
proof –
have *runiq* ($f\hat{}-1$) **using** *assms*(*1*) *lm66* **by** *blast* **then**
have *converse* ‘ *?l* $\subseteq$ *runiqs* **using** *assms*(*2*) **by** (*rule lm68*)
then have *?r* $\supseteq$ *converse*‘(*converse*‘*?l*) **by** *auto*
moreover have *converse*‘(*converse*‘*?l*)=*?l* **by** (*rule lm95*)
ultimately show *?thesis* **by** *simp*
qed

lemma *lm01*: { $R \cup \{(x, y)\}$ | y . $y \in A$ } $\subseteq$ *totalRels* ($\{x\} \cup$ *Domain R*) ($A \cup$ *Range R*)
by *force*

lemma *lm69*: *injectionsUniverse* = *runiqs* $\cap$ *converse*'*runiqs*
unfolding *runiqs-def* **by** *auto*

lemma *lm73*: **assumes** $f \in$ *injectionsUniverse* $x \notin$ *Domain f A* $\cap$ (*Range f*) = {}
shows $\{f \cup \{(x, y)\}|\ y.\ y \in A\} \subseteq$ *injectionsUniverse*
(**is** *?l* $\subseteq$ *?r*)
proof –
have $f \in$ *converse*'*runiqs* **using** *assms*(*1*) *lm69* **by** *blast*
then have *?l* $\subseteq$ *converse*'*runiqs* **using** *assms*(*3*) **by** (*rule lm68b*)
moreover have *?l* $\subseteq$ *runiqs* **using** *assms*(*1*,*2*) *lm60* **by** *force*
ultimately show *?thesis* **using** *lm69* **by** *blast*
qed

lemma *lm26b*: *injections X Y* = *totalRels X Y* $\cap$ *injectionsUniverse*
using *injections-def lm26* **by** *metis*

lemma *lm27b*: **assumes** $f \in$ *injectionsUniverse*
shows *f outside A* $\in$ *injectionsUniverse*
using *assms* **by** (*metis* (*no-types*) *Outside-def lm27*)

lemma *lm91*: **assumes** $R \in$ *totalRels A B*
shows *R outside C* $\in$ *totalRels* ($A-C$) *B*
unfolding *Outside-def* **using** *assms* **by** *blast*

lemma *lm71*: **assumes** $g \in$ *injections' A B*
shows *g outside C* $\in$ *injections'* ($A - C$) *B*
using *assms Outside-def Range-outside-sub lm27 mem-Collect-eq outside-reduces-domain*
by *fastforce*

lemma *lm71b*: **assumes** $g \in$ *injections A B*
shows *g outside C* $\in$ *injections* ($A - C$) *B*
using *assms lm71* **by** (*metis injections-def*)

lemma *lm74*: $\{x\}\times\{y\}=\{(x,y)\}$
by *simp*

lemma *lm75*: **assumes** $x \in$ *Domain f runiq f*
shows $\{x\}\times f``\{x\} = \{(x,f,,x)\}$
using *assms lm74 Image-runiq-eq-eval* **by** *metis*

corollary *lm92*: **assumes** $x \in$ *Domain f runiq f*
shows $f = (f -- x) \cup \{(x,f,,x)\}$
using *assms lm75 outsideUnion* **by** *metis*

lemma *nn30b*: **assumes** $f \in$ *injectionsUniverse*
shows *Range*(*f outside A*) = *Range f* $-$ $f``A$
using *assms mem-Collect-eq rangeOutside* **by** (*metis*)

lemma *lm76*: **assumes** $g \in$ *injections'* X Y $x \in$ *Domain* g
shows $g \in \{g--x \cup \{(x,y)\}|y.\ y \in Y - (Range(g--x))\}$
proof –
let *?f=g−−x* **have** *g∈injectionsUniverse* **using** *assms(1) lm26* **by** *fast* **then**
moreover have
g,,x ∈ g``{x} **using** *assms(2)* **by** (*metis Image-runiq-eq-eval insertI1 mem-Collect-eq*)
ultimately have *g,,x ∈ Y−Range ?f* **using** *nn30b assms(1)* **by** *fast* **moreover**
have *g=?f∪{(x, g,,x)}*
using *assms lm92 mem-Collect-eq* **by** (*metis* (*lifting*)) **ultimately show** *?thesis*
by *blast*
qed

corollary *lm71c*: **assumes** $x \notin X$ $g \in$ *injections* $(\{x\} \cup X)$ Y
shows $g--x \in$ *injections* X Y
using *assms lm71b* **by** (*metis Diff-insert-absorb insert-is-Un*)

corollary *lm77*: **assumes** $x \notin X$ $g \in$ *injections* $(\{x\} \cup X)$ Y
(**is** $g \in$ *injections* $(?X)$ Y)
shows $\exists\ f \in$ *injections* X Y. $g \in \{f \cup \{(x,y)\}|y.\ y \in Y - (Range\ f)\}$
proof –
let *?f=g−−x* **have** *1*: *g∈injections' ?X Y* **using** *assms Universes.lm24* **by** *metis*

have *Domain g=?X* **using** *assms(2) Universes.lm24 mem-Collect-eq* **by** (*metis* (*mono-tags*, *lifting*))
then have *0*: $x \in$ *Domain* g **by** *simp* **then have** $?f \in$ *injections* X Y **using** *assms lm71c* **by** *fast*
moreover have *g∈{?f∪{(x,y)}|y. y∈Y−Range ?f}* **using** *1 0* **by** (*rule lm76*)
ultimately show *?thesis* **by** *blast*
qed

corollary *lm77b*: **assumes** $x \notin X$
shows *injections* $(\{x\} \cup X)$ $Y \subseteq$
$(\bigcup\ f \in$ *injections* X $Y.\ \{f \cup \{(x, y)\}\ |\ y\ .\ y \in Y - (Range\ f)\})$
using *assms lm77* **by** *fast*

lemma *lm73b*: **assumes** $x \notin X$
shows $(\bigcup\ f \in$*injections'* X $Y.\ \{f \cup \{(x, y)\}\ |\ y\ .\ y \in Y - Range\ f\}) \subseteq$
injections' $(\{x\} \cup X)$ Y
using *assms lm73 injections-def lm26b lm01*
proof –
{ **fix** *f* **assume** $f \in$ *injections'* X Y **then have**
0: $f \in$ *injectionsUniverse* & $x \notin$ *Domain* f & *Domain* $f = X$ & *Range* $f \subseteq Y$
using *assms* **by** *fast*
then have $f \in$ *injectionsUniverse* **by** *fast* **moreover have** $x \notin$ *Domain* f **using** *0* **by** *fast*
moreover have *1*: $(Y - Range\ f) \cap Range\ f = \{\}$ **by** *blast*

ultimately have $\{f \cup \{(x, y)\} \mid y . y \in (Y - Range\ f)\} \subseteq injectionsUniverse$ **by** (*rule lm73*)
moreover have $\{f \cup \{(x, y)\} \mid y . y \in (Y - Range\ f)\} \subseteq totalRels\ (\{x\} \cup X)\ Y$ **using** *lm01 0* **by** *force*
ultimately have $\{f \cup \{(x, y)\} \mid y . y \in (Y - Range\ f)\} \subseteq injectionsUniverse \cap totalRels\ (\{x\} \cup X)\ Y$ **by** *auto*}
thus *?thesis* **using** *lm26* **by** *blast*
qed

corollary *lm78*: **assumes** $x \notin X$
shows $(\bigcup f \in injections\ X\ Y.\ \{f \cup \{(x, y)\} \mid y . y \in Y - (Range\ f)\}) =$
injections $(\{x\} \cup X)\ Y$
(**is** *?r=injections ?X -*)
proof –
have
0: $?r = (\bigcup f \in injections'\ X\ Y.\ \{f \cup \{(x, y)\} \mid y . y \in Y - Range\ f\})$ (**is** -=*?r'*) **unfolding** *Universes.lm24* **by** *blast*
have $?r' \subseteq injections'\ ?X\ Y$ **using** *assms* **by** (*rule lm73b*) **moreover have** ... = *injections ?X Y* **unfolding** *Universes.lm24*
by *simp* **ultimately have** $?r \subseteq injections\ ?X\ Y$ **using** *0* **by** *simp*
moreover have $injections\ ?X\ Y \subseteq ?r$ **using** *assms* **by** (*rule lm77b*) **ultimately show** *?thesis* **by** *blast*
qed

lemma *lm89*: **assumes** $\forall\ x.\ (P\ x \longrightarrow (f\ x = g\ x))$
shows $Union\ \{f\ x | x.\ P\ x\} = Union\ \{g\ x \mid x.\ P\ x\}$
using *assms* **try0 by** *blast*

lemma *lm88*: **assumes** $x \notin Domain\ R$
shows $R +* \{(x,y)\} = R \cup \{(x,y)\}$
using *assms*
by (*metis* (*erased*, *lifting*) *Domain-empty Domain-insert Int-insert-right-if0 disjoint-iff-not-equal ex-in-conv paste-disj-domains*)

lemma *lm79*: **assumes** $x \notin X$
shows $(\bigcup f \in injections\ X\ Y.\ \{f +* \{(x, y)\} \mid y . y \in Y - Range\ f\}) =$
$(\bigcup f \in injections\ X\ Y.\ \{f \cup \{(x, y)\} \mid y . y \in Y - Range\ f\})$
(**is** *?l = ?r*)
proof –
have
0: $\forall f \in injections\ X\ Y.\ x \notin Domain\ f$ **unfolding** *injections-def* **using** *assms* **by** *fast* **then have**
1: $?l = Union\ \{\{f +* \{(x, y)\} \mid y . y \in Y - Range\ f\} | f . f \in injections\ X\ Y\ \&\ x \notin Domain\ f\}$
(**is** -=*?l'*) **using** *assms* **by** *auto* **moreover have**
2: $?r = Union\ \{\{f \cup \{(x, y)\} \mid y . y \in Y - Range\ f\} | f . f \in injections\ X\ Y\ \&\ x$

$\notin$ *Domain f*}
(**is** -=*?r'*) **using** *assms 0* **by** *auto* **have** $\forall$ *f*. *f*$\in$*injections X Y* & *x* $\notin$ *Domain f* $\longrightarrow$
{*f* +∗ {(*x*, *y*)} | *y* . *y* $\in$ *Y*−*Range f*}={*f* $\cup$ {(*x*, *y*)} | *y* . *y* $\in$ *Y*−*Range f*}
using *lm88* **by** *force*
then have *?l'*=*?r'* **by** (*rule lm89*) **then show** *?l* = *?r* **using** *1 2* **by** *presburger*
qed

corollary *lm78b*: **assumes** *x* $\notin$ *X*
shows ($\bigcup$ *f* $\in$ *injections X Y*. {*f* +∗ {(*x*, *y*)} | *y* . *y* $\in$ *Y*−*Range f*}) =
injections ({*x*} $\cup$ *X*) *Y*
(**is** *?l*=*?r*)
proof −
have *?l*=($\bigcup$ *f*$\in$*injections X Y*. {*f* $\cup$ {(*x*, *y*)} | *y* . *y* $\in$ *Y*−*Range f*}) **using** *assms* **by** (*rule lm79*)
moreover have ... = *?r* **using** *assms* **by** (*rule lm78*) **ultimately show** *?thesis* **by** *presburger*
qed

lemma *lm81*: *set* [*f* $\cup$ {(*x*,*y*)} . *y* ← (*filter* (%*y*. *y* $\notin$ (*Range f*)) *Y*)] =
{*f* $\cup$ {(*x*,*y*)} | *y* . *y* $\in$ (*set Y*) − (*Range f*)}
by *auto*

lemma *lm82*: **assumes** $\forall$ *x* $\in$ *set L*. *set* (*F x*) = *G x*
shows *set* (*concat* [*F x* . *x* <− *L*]) = ($\bigcup$ *x*$\in$*set L*. *G x*)
using *assms* **by** *force*

lemma *lm83*: *set* (*concat* [[*f* $\cup$ {(*x*,*y*)} . *y* ← (*filter* (%*y*. *y* $\notin$ *Range f*) *Y*)]. *f* ← *F*]) =
($\bigcup$ *f* $\in$ *set F*. {*f* $\cup$ {(*x*,*y*)} | *y* . *y* $\in$ (*set Y*) − (*Range f*)})
by *auto*

lemma *lm81b*: **assumes** *finite Y*
shows *set* [*f* +∗ {(*x*,*y*)} . *y* ← *sorted-list-of-set* (*Y* − (*Range f*))] =
{ *f* +∗ {(*x*,*y*)} | *y* . *y* $\in$ *Y* − (*Range f*)}
using *assms* **by** *auto*

lemma *lm83d*: **assumes** *finite Y*
shows *set* (*concat* [[*f* +∗ {(*x*,*y*)}. *y* ← *sorted-list-of-set*(*Y* − (*Range f*))]. *f* ← *F*]) =
($\bigcup$ *f* $\in$ *set F*.{*f* +∗ {(*x*,*y*)} | *y* . *y* $\in$ *Y* − (*Range f*)})
using *assms lm81b lm82* **by** *auto*

39 computable injections

fun *injectionsAlg*
  where
  injectionsAlg [] (Y::$'a$ *list*) = [{}] |
  injectionsAlg (x#xs) Y =
    concat [[R∪{(x,y)}. y ← (*filter* (%y. y ∉ *Range* R) Y)]
      .R ← *injectionsAlg* xs Y]

corollary *lm83b*: *set* (*injectionsAlg* (x # xs) Y) =
  (⋃ f ∈ *set* (*injectionsAlg* xs Y). {f ∪ {(x,y)} | y . y ∈ (*set* Y) − (*Range* f)})
  using *lm83 injectionsAlg-def* **by** *auto*

corollary *lm83c*: **assumes** *set* (*injectionsAlg* xs Y) = *injections* (*set* xs) (*set* Y)
  shows *set* (*injectionsAlg* (x # xs) Y) =
    (⋃ f ∈ *injections* (*set* xs) (*set* Y). {f ∪ {(x,y)} | y . y ∈ (*set* Y) − (*Range* f)})
  using *assms lm83b* **by** *auto*

We sometimes use parallel *abbreviation* and *definition* for the same object to save typing 'unfolding xxx' each time. There is also different behaviour in the code extraction.

lemma *lm44*: *injections'* {} Y = {{}}
  by (*simp add*: *lm26 lm84 runiq-emptyrel*)

lemma *lm44b*: *injections* {} Y = {{}}
  unfolding *injections-def* **by** (*simp add*: *lm26 lm84 runiq-emptyrel*)

lemma *lm80*: *injectionsAlg* [] Y = [{}]
  using *injectionsAlg-def* **by** *simp*

lemma *lm86*: **assumes** x ∉ *set* xs *set* (*injectionsAlg* xs Y) = *injections* (*set* xs) (*set* Y)
  shows *set* (*injectionsAlg* (x # xs) Y) = *injections* ({x} ∪ *set* xs) (*set* Y)
  (**is** *?l*=*?r*)
proof −
 have *?l* = (⋃ f∈*injections* (*set* xs) (*set* Y). {f ∪ {(x,y)} | y . y ∈ (*set* Y)−*Range* f})
 using *assms*(*2*) **by** (*rule lm83c*) **moreover have** ... = *?r* **using** *assms*(*1*) **by** (*rule lm78*)
 ultimately show *?thesis* **by** *presburger*
qed

lemma *lm86b*: **assumes** x ∉ *set* xs
  set (*injections-alg* xs Y) = *injections* (*set* xs) Y
  finite Y

shows *set* (*injections-alg* (*x#xs*) *Y*) = *injections* ({*x*} ∪ *set xs*) *Y*
(**is** *?l=?r*)
proof −
have *?l* = (⋃ *f*∈*injections* (*set xs*) *Y*. {*f* +∗ {(*x*,*y*)} | *y* . *y* ∈ *Y*−*Range f*})
using *assms*(*2*,*3*) *lm83d* **by** *auto*
moreover have ... = *?r* **using** *assms*(*1*) **by** (*rule lm78b*)
ultimately show *?thesis* **by** *presburger*
qed

lemma *listInduct*: **assumes** *P* [] ∀ *xs x*. *P xs* ⟶ *P* (*x#xs*)
shows ∀ *x*. *P x*
using *assms* **by** (*metis structInduct*)

lemma *injectionsFromEmptyAreEmpty*: *set* (*injections-alg* [] *Z*) = {{}}
by *simp*

theorem *injections-equiv*: **assumes** *finite Y* **and** *distinct X*
shows *set* (*injections-alg X Y*) = *injections* (*set X*) *Y*
proof −
let *?P*=λ *l*. (*distinct l* ⟶ (*set* (*injections-alg l Y*)=*injections* (*set l*) *Y*))
have *?P* [] **using** *injectionsFromEmptyAreEmpty list.set*(*1*) *lm44b* **by** (*metis*)
moreover have ∀ *x xs*. *?P xs* ⟶ *?P* (*x#xs*) **using** *assms*(*1*) *lm86b* **by** (*metis distinct.simps*(*2*) *list.simps*(*15*))
ultimately have *?P X* **by** (*rule structInduct*)
then show *?thesis* **using** *assms*(*2*) **by** *blast*
qed

lemma *lm67*: **assumes** *l* ∈ *set* (*all-partitions-list G*) *distinct G*
shows *distinct l*
using *assms all-partitions-list-def* **by** (*metis all-partitions-equivalence′*)

lemma *lm03*: **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** (*metis card-ge-0-finite*)

lemma *lm05*: **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*]
proof −
let *?g1*=*all-partitions-list* **let** *?f2*=*injections* **let** *?g2*=*injections-alg*
have ∀ *l* ∈ *set* (*?g1 G*). *set* (*?g2 l N*) = *?f2* (*set l*) *N* **using** *assms lm03* **by** *blast*
then have *set* [*set* (*?g2 l N*). *l* <− *?g1 G*] = {*?f2 P N*| *P*. *P* ∈ *set* (*map set* (*?g1 G*))} **apply** (*rule setVsList*) **done**
moreover have ... = {*?f2 P N*| *P*. *P* ∈ *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* ∈ *all-partitions* (*set G*)} =
Union (*set* [*set* (*injections-alg l N*) . *l* ← *all-partitions-list G*])
(**is** *Union ?L* = *Union ?R*)
proof −
have *?L* = *?R* **using** *assms* **by** (*rule lm05*) **thus** *?thesis* **by** *presburger*
qed

corollary *lm70b*: **assumes** *card N > 0 distinct G*
shows *possibleAllocationsRel N* (*set G*) =
set(*possibleAllocationsAlg N G*)
proof −
let *?LL*=⋃ {*injections P N*| *P*. *P* ∈ *all-partitions* (*set G*)}
let *?RR*=⋃ (*set* [*set* (*injections-alg l N*) . *l* ← *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

end

40 Implementation of integer numbers by target-language integers

theory *Code-Target-Int*
imports *Main*
begin

code-datatype *int-of-integer*

declare [[*code drop*: *integer-of-int*]]

context
includes *integer.lifting*
begin

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

lemma [*code*]:
Int.Pos = *int-of-integer* ∘ *integer-of-num*

by *transfer* (*simp add*: *fun-eq-iff*)

lemma [*code*]:
Int.Neg = *int-of-integer* ∘ *uminus* ∘ *integer-of-num*
by *transfer* (*simp add*: *fun-eq-iff*)

lemma [*code-abbrev*]:
int-of-integer (*numeral k*) = *Int.Pos k*
by *transfer simp*

lemma [*code-abbrev*]:
int-of-integer (− *numeral k*) = *Int.Neg k*
by *transfer simp*

lemma [*code*, *symmetric*, *code-post*]:
0 = *int-of-integer 0*
by *transfer simp*

lemma [*code*, *symmetric*, *code-post*]:
1 = *int-of-integer 1*
by *transfer simp*

lemma [*code*]:
k + *l* = *int-of-integer* (*of-int k* + *of-int l*)
by *transfer simp*

lemma [*code*]:
− *k* = *int-of-integer* (− *of-int k*)
by *transfer simp*

lemma [*code*]:
k − *l* = *int-of-integer* (*of-int k* − *of-int l*)
by *transfer simp*

lemma [*code*]:
Int.dup k = *int-of-integer* (*Code-Numeral.dup* (*of-int k*))
by *transfer simp*

declare [[*code drop*: *Int.sub*]]

lemma [*code*]:
k ∗ *l* = *int-of-integer* (*of-int k* ∗ *of-int l*)
by *simp*

lemma [*code*]:
Divides.divmod-abs k l = *map-prod int-of-integer int-of-integer*
(*Code-Numeral.divmod-abs* (*of-int k*) (*of-int l*))
by (*simp add*: *prod-eq-iff*)

lemma [*code*]:
  $k\ div\ l = int\text{-}of\text{-}integer\ (of\text{-}int\ k\ div\ of\text{-}int\ l)$
  by *simp*

lemma [*code*]:
  $k\ mod\ l = int\text{-}of\text{-}integer\ (of\text{-}int\ k\ mod\ of\text{-}int\ l)$
  by *simp*

lemma [*code*]:
  $HOL.equal\ k\ l = HOL.equal\ (of\text{-}int\ k :: integer)\ (of\text{-}int\ l)$
  by *transfer* (*simp add*: *equal*)

lemma [*code*]:
  $k \leq l \longleftrightarrow (of\text{-}int\ k :: integer) \leq of\text{-}int\ l$
  by *transfer rule*

lemma [*code*]:
  $k < l \longleftrightarrow (of\text{-}int\ k :: integer) < of\text{-}int\ l$
  by *transfer rule*
end

lemma (**in** *ring-1*) *of-int-code-if*:
  $of\text{-}int\ k = (if\ k = 0\ then\ 0$
    $else\ if\ k < 0\ then\ -\ of\text{-}int\ (-\ k)$
    $else\ let$
      $(l, j) = divmod\text{-}int\ k\ 2;$
      $l' = 2 * of\text{-}int\ l$
    $in\ if\ j = 0\ then\ l'\ else\ l' + 1)$
proof −
  from *mod-div-equality* **have** $*$: $of\text{-}int\ k = of\text{-}int\ (k\ div\ 2 * 2 + k\ mod\ 2)$ **by** *simp*
  show *?thesis*
    by (*simp add*: *Let-def divmod-int-mod-div of-int-add* [*symmetric*])
      (*simp add*: $*$ *mult.commute*)
qed

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

lemma [*code*]:
  $nat = nat\text{-}of\text{-}integer \circ of\text{-}int$
  including *integer.lifting* **by** *transfer* (*simp add*: *fun-eq-iff*)

code-identifier
  code-module *Code-Target-Int* $\rightharpoonup$
    (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

end

41 VCG auction: definitions and theorems

theory *CombinatorialAuction*

imports

UniformTieBreaking
~~/src/HOL/Library/Code-Target-Nat
~~/src/HOL/Library/Code-Target-Int

~~/src/HOL/Library/Code-Numeral

begin

42 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 *seller* == (*0*::*integer*)
abbreviation *allAllocations N G* == *possibleAllocationsRel N G*
abbreviation *allAllocations' N Ω* == *injectionsUniverse* ∩
{*a*. *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions Ω*}
abbreviation *allAllocations'' N G* == *allocationsUniverse*∩{*a*. *Domain a* ⊆ *N*
& ⋃ *Range a* = *G*}
lemma *lm28*: *allAllocations N G*=*allAllocations' N G* &
allAllocations N G=*allAllocations'' N G* **using** *lm19 nn24* **by** *metis*
lemma *lm28b*:
(*a* ∈ *allAllocations'' N G*) = (*a* ∈ *allocationsUniverse*& *Domain a* ⊆ *N* & ⋃
Range a = *G*) **by** *force*
abbreviation *soldAllocations N Ω* ==
(*Outside'* {*seller*}) ‘ (*allAllocations* (*N* ∪ {*seller*}) Ω)
abbreviation *soldAllocations' N Ω* ==
(*Outside'* {*seller*}) ‘ (*allAllocations'* (*N* ∪ {*seller*}) Ω)
abbreviation *soldAllocations'' N Ω* ==
(*Outside'* {*seller*}) ‘ (*allAllocations''* (*N* ∪ {*seller*}) Ω)
lemma *lm28c*:
soldAllocations N G = *soldAllocations' N G* & *soldAllocations' N G* = *soldAllocations'' N G*
using *assms lm28* **by** *metis*
corollary *lm28d*: *soldAllocations* = *soldAllocations'* & *soldAllocations'* = *soldAllocations''*
& *soldAllocations* = *soldAllocations''* **using** *lm28c* **by** *metis*
lemma *lm32*: *soldAllocations* (*N*−{*seller*}) *G* ⊆ *soldAllocations N G* **using** *Outside-def*
by *simp*

lemma *lm34*: (*a* ∈ *allocationsUniverse*) = (*a* ∈ *allAllocations''* (*Domain a*) (⋃
Range a))

by *blast*
lemma *lm35*: **assumes** $N1 \subseteq N2$ **shows** *allAllocations''* $N1$ $G \subseteq$ *allAllocations''* $N2$ G
using *assms* **by** *auto*
lemma *lm36*: **assumes** $a \in$ *allAllocations''* N G **shows** *Domain* $(a$ $--$ $x) \subseteq N-\{x\}$
using *assms Outside-def* **by** *fastforce*
lemma *lm37*: **assumes** $a \in$ *soldAllocations* N G **shows** $a \in$ *allocationsUniverse*
proof $-$
obtain *aa* **where** $a=aa$ $--$ *seller* & $aa \in$ *allAllocations* $(N \cup \{seller\})$ G
using *assms* **by** *blast*
then have $a \subseteq aa$ & $aa \in$ *allocationsUniverse* **unfolding** *Outside-def* **using** *nn24b* **by** *blast*
then show *?thesis* **using** *lm35b* **by** *blast*
qed
lemma *lm38*: **assumes** $a \in$ *soldAllocations* N G **shows** $a \in$ *allAllocations''* (*Domain* a) ($\bigcup$ *Range* a)
proof $-$ **show** *?thesis* **using** *assms lm37* **by** *blast* **qed**
lemma **assumes** $N1 \subseteq N2$ **shows** *soldAllocations''* $N1$ $G \subseteq$ *soldAllocations''* $N2$ G
using *assms lm35 lm36 nn24c lm28b lm28 lm34 lm38 Outside-def* **by** *blast*

lemma *lll59b*: *runiq* $(X \times \{y\})$ **using** *rightUniqueTrivialCartes* **by** (*metis trivial-singleton*)
lemma *lm37b*: $\{x\} \times \{y\} \in$ *injectionsUniverse* **using** *Universes.lm37* **by** *fastforce*
lemma *lm40b*: **assumes** $a \in$ *soldAllocations''* N G **shows** $\bigcup$ *Range* $a \subseteq G$ **using** *assms Outside-def* **by** *blast*
lemma *lm41*: $a \in$ *soldAllocations''* N G =
(*EX aa. aa* $--$ (*seller*)=a & $aa \in$*allAllocations''* $(N \cup \{seller\})$ 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* $a \subseteq N-\{seller\}$ $\bigcup$ *Range* $a \subseteq G$ **shows**
$a \in$ *soldAllocations''* N G **using** *assms lm41*
proof $-$
let *?i=seller* **let** *?Y*=$\{G-\bigcup$ *Range* $a\}-\{\{\}\}$ **let** *?b*=$\{$*?i*$\} \times$ *?Y* **let** *?aa*=$a \cup$*?b*
let *?aa'*=a $+*$ *?b*
have
1: $a \in$ *allocationsUniverse* **using** *assms*(*1*) **by** *fast*
have *?b* $\subseteq \{($*?i*$,G-\bigcup$ *Range* $a)\} - \{($*?i*$, \{\})\}$ **by** *fastforce* **then have**
2: *?b* $\in$ *allocationsUniverse* **using** *allocationUniverseProperty lm35b* **by** (*metis*(*no-types*))
have
3: $\bigcup$ *Range* $a \cap \bigcup$ (*Range ?b*) = $\{\}$ **by** *blast* **have**
4: *Domain* $a \cap$ *Domain ?b* $=\{\}$ **using** *assms* **by** *fast*
have *?aa* $\in$ *allocationsUniverse* **using** *1 2 3 4* **by** (*rule lm23*)
then have *?aa* $\in$ *allAllocations''* (*Domain ?aa*)
($\bigcup$ *Range ?aa*) **unfolding** *lm34* **by** *metis* **then have**
?aa $\in$ *allAllocations''* $(N \cup \{$*?i*$\})$ ($\bigcup$ *Range ?aa*) **using** *lm35 assms paste-def* **by**

auto
moreover have *Range ?aa* = *Range a* ∪ *?Y* **by** *blast* **then moreover have**
$\bigcup$ *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′* ∈ *allAllocations″* (*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∈*allAllocations N* Ω=(*a*∈*injectionsUniverse* & *Domain a*⊆*N* & *Range a*∈*all-partitions* Ω)
by (*metis* (*full-types*) *lm19c*)

lemma *lm37n*: **assumes** *a* ∈ *soldAllocations″ N G* **shows** *Domain a* ⊆ *N*−{*seller*} & *a* ∈ *allocationsUniverse*
proof −
let *?i*=*seller* **obtain** *aa* **where**
0: *a*=*aa* −− *?i* & *aa* ∈ *allAllocations″* (*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* ∈ *soldAllocations 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* ∈ *soldAllocations″ N G* **shows**
a ∈ *allocationsUniverse* & *Domain a* ⊆ *N*−{*seller*} & $\bigcup$ *Range a* ⊆ *G*
proof −
have *a* ∈ *allocationsUniverse* **using** *assms lm37n* **by** *blast*
moreover have *Domain a* ⊆ *N*−{*seller*} **using** *assms lm37n* **by** *blast*
moreover have $\bigcup$ *Range a* ⊆ *G* **using** *assms lm40b* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

corollary *lm37d*:
(*a*∈*soldAllocations″ N G*)=(*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*seller*} & $\bigcup$ *Range a* ⊆ *G*)
using *lm37c lm37e* **by** (*metis* (*mono-tags*))

lemma *lm42*: (*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*seller*} & $\bigcup$ *Range a* ⊆ *G*) =
(*a*∈*allocationsUniverse*& *a*∈{*aa*. *Domain aa* ⊆ *N*−{*seller*} & $\bigcup$ *Range aa* ⊆ *G*})

by (*metis* (*lifting*, *no-types*) *mem-Collect-eq*)

corollary *lm37f*: $(a{\in}soldAllocations''\ N\ G)=$
$(a{\in}allocationsUniverse\&\ a{\in}\{aa.\ Domain\ aa \subseteq N{-}\{seller\}\ \&\ \bigcup\ Range\ aa \subseteq G\})$
(**is** *?L* = *?R*)
proof −
have $?L = (a{\in}allocationsUniverse\&\ Domain\ a \subseteq N{-}\{seller\}\ \&\ \bigcup\ Range\ a \subseteq G)$ **by** (*rule lm37d*)
moreover have ... = *?R* **by** (*rule lm42*) **ultimately show** *?thesis* **by** *presburger*
qed

corollary *lm37g*: $a{\in}soldAllocations''\ N\ G=$
$(a{\in}\ (allocationsUniverse \cap \{aa.\ Domain\ aa \subseteq N{-}\{seller\}\ \&\ \bigcup\ Range\ aa \subseteq G\}))$

using *lm37f* **by** (*metis* (*mono-tags*) *Int-iff*)

abbreviation $soldAllocations'''\ N\ G ==$
$allocationsUniverse{\cap}\{aa.\ Domain\ aa{\subseteq}N{-}\{seller\}\ \&\ \bigcup Range\ aa{\subseteq}G\}$

lemma *lm44*: **assumes** $a \in soldAllocations'''\ N\ G$ **shows** $a\ {-}{-}\ n \in soldAllocations'''\ (N{-}\{n\})\ G$
proof −
let *?bb=seller* **let** *?d=Domain* **let** *?r=Range* **let** $?X2=\{aa.\ ?d\ aa{\subseteq}N{-}\{?bb\}\ \&\ \bigcup\ ?r\ aa{\subseteq}G\}$
let $?X1=\{aa.\ ?d\ aa{\subseteq}N{-}\{n\}{-}\{?bb\}\ \&\ \bigcup\ ?r\ aa{\subseteq}G\}$
have $a{\in}?X2$ **using** *assms*(*1*) **by** *fast* **then have**
0: $?d\ a \subseteq N{-}\{?bb\}\ \&\ \bigcup\ ?r\ a \subseteq G$ **by** *blast* **then have** $?d\ (a{-}{-}n) \subseteq N{-}\{?bb\}{-}\{n\}$

using *outside-reduces-domain* **by** (*metis Diff-mono subset-refl*) **moreover have**

$... = N{-}\{n\}{-}\{?bb\}$ **by** *fastforce* **ultimately have**
$?d\ (a{-}{-}n) \subseteq N{-}\{n\}{-}\{?bb\}$ **by** *blast* **moreover have** $\bigcup\ ?r\ (a{-}{-}n) \subseteq G$
unfolding *Outside-def* **using** *0* **by** *blast* **ultimately have** $a\ {-}{-}\ n \in\ ?X1$ **by** *fast* **moreover have**
$a{-}{-}n \in allocationsUniverse$ **using** *assms*(*1*) *Int-iff lm35d* **by** (*metis*(*lifting*,*mono-tags*))

ultimately show *?thesis* **by** *blast*
qed

corollary *lm37h*: $soldAllocations''\ N\ G{=}soldAllocations'''\ N\ G$
(**is** *?L=?R*) **proof** − {**fix** *a* **have** $a \in\ ?L = (a \in\ ?R)$ **by** (*rule lm37g*)} **thus** *?thesis* **by** *blast* **qed**

lemma *lm28e*: $soldAllocations{=}soldAllocations'\ \&\ soldAllocations'{=}soldAllocations''$ &
$soldAllocations''{=}soldAllocations'''$ **using** *lm37h lm28d* **by** *metis*

corollary *lm44b*: **assumes** $a \in soldAllocations\ N\ G$ **shows** $a\ {-}{-}\ n \in soldAllo$-

cations $(N-\{n\})$ *G*

proof $-$
let *?A'=soldAllocations'''* **have** $a \in$ *?A' N G* **using** *assms lm28e* **by** *metis* **then have** a $--$ $n \in$ *?A'* $(N-\{n\})$ *G* **by** (*rule lm44*) **thus** *?thesis* **using** *lm28e* **by** *metis*
qed

corollary *lm37i*: **assumes** *G1* $\subseteq$ *G2* **shows** *soldAllocations''' N G1* $\subseteq$ *soldAllocations''' N G2*
using *assms* **by** *blast*

corollary *lm33*: **assumes** *G1* $\subseteq$ *G2* **shows** *soldAllocations'' N G1* $\subseteq$ *soldAllocations'' N G2*
using *assms lm37i lm37h*
proof $-$
have *soldAllocations'' N G1* = *soldAllocations''' N G1* **by** (*rule lm37h*)
moreover have ... $\subseteq$ *soldAllocations''' N G2* **using** *assms*(*1*) **by** (*rule lm37i*)
moreover have ... = *soldAllocations'' N G2* **using** *lm37h* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed

abbreviation *maximalAllocations'' N* Ω *b* == *argmax* (*setsum b*) (*soldAllocations N* Ω)

abbreviation *maximalStrictAllocations' N G b*==
argmax (*setsum b*) (*allAllocations* ($\{$*seller*$\}\cup N$) *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*

abbreviation *randomBids' N* Ω *b random*==*resolvingBid'* ($N\cup\{$*seller*$\}$) Ω *b random*

abbreviation *vcgas N G b r* == *Outside'* $\{$*seller*$\}$ `
((*argmax*$\circ$*setsum*) (*randomBids' N G b r*)
((*argmax*$\circ$*setsum*) *b* (*allAllocations* ($N\cup\{$*seller*$\}$) (*set G*))))
abbreviation *vcga N G b r* == *the-elem* (*vcgas N G b r*)
abbreviation *vcga' N G b r* == (*the-elem*
(*argmax* (*setsum* (*randomBids' N G b r*)) (*maximalStrictAllocations' N* (*set G*) *b*))) $--$ *seller*

lemma *lm001*: **assumes**
card ((*argmax*∘*setsum*) (*randomBids′ N G b r*) ((*argmax*∘*setsum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set G*))))=*1*
shows *vcga N G b r* =
(*the-elem*
((*argmax*∘*setsum*) (*randomBids′ N G b r*) ((*argmax*∘*setsum*) *b* (*allAllocations* ({*seller*}∪*N*) (*set G*))))) −− *seller*
using *assms cardOneTheElem* **by** *auto*

corollary *lm001b*: **assumes**
card ((*argmax*∘*setsum*) (*randomBids′ N G b r*) ((*argmax*∘*setsum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set G*))))=*1*
shows *vcga N G b r* = *vcga′ N G b r* (**is** *?l*=*?r*) **using** *assms lm001*
proof −
have *?l*=(*the-elem* ((*argmax*∘*setsum*) (*randomBids′ N G b r*)
((*argmax*∘*setsum*) *b* (*allAllocations* ({*seller*}∪*N*) (*set G*))))) −− *seller*
using *assms* **by** (*rule lm001*) **moreover have** ... = *?r* **by** *force* **ultimately show** *?thesis* **by** *blast*
qed

lemma *lm92c*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
card (
(*argmax*∘*setsum*) (*randomBids′ N G bids random*)
((*argmax*∘*setsum*) *bids* (*allAllocations* (*N*∪{*seller*}) (*set G*))))=*1* (**is** *card ?l*=-)
proof −
let *?N*=*N*∪{*seller*} **let** *?b′*=*randomBids′ N G bids random* **let** *?s*=*setsum* **let** *?a*=*argmax* **let** *?f*=*?a* ∘ *?s*
have *1*: *?N*≠{} **by** *auto* **have** *4*: *finite ?N* **using** *assms*(*3*) **by** *simp*
have *?a* (*?s ?b′*) (*?a* (*?s bids*) (*allAllocations ?N* (*set G*)))=
{*chosenAllocation′ ?N G bids random*} (**is** *?L*=*?R*)
using *1 assms*(*1*) *assms*(*2*) *4* **by** (*rule lm92*)
moreover have *?L*= *?f ?b′* (*?f bids* (*allAllocations ?N* (*set G*))) **by** *auto*
ultimately have *?l*={*chosenAllocation′ ?N G bids random*} **by** *presburger*
moreover have *card* ...=*1* **by** *simp* **ultimately show** *?thesis* **by** *presburger*
qed

lemma *lm002*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
vcga N G b r = *vcga′ N G b r* (**is** *?l*=*?r*) **using** *assms lm001b lm92c* **by** *blast*

theorem *vcgaDefiniteness*: **assumes** *distinct G set G* ≠ {} *finite N* **shows**
card (*vcgas N G b r*)=*1*
using *assms lm92c cardOneImageCardOne*
proof −
have *card* ((*argmax*∘*setsum*) (*randomBids′ N G b r*) ((*argmax*∘*setsum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set G*))))=*1*
(**is** *card ?X*=-) **using** *assms lm92c* **by** *blast*
moreover have (*Outside′*{*seller*}) ‘ *?X* = *vcgas N G b r* **by** *blast*
ultimately show *?thesis* **using** *cardOneImageCardOne* **by** *blast*
**qed*

theorem *vcgpDefiniteness*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows** $\exists!$ *y. vcgap N G b r n=y* **using** *assms vcgaDefiniteness* **by** *simp*

Here we are showing that our way of randomizing using randomBids' actually breaks the tie: we are left with a singleton after the tie-breaking step.

lemma *lm92b*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
card (*argmax* (*setsum* (*randomBids'* *N G b r*)) (*maximalStrictAllocations'* *N* (*set G*) *b*))=*1*
(**is** *card ?L*=-)

proof –
let *?n*={*seller*} **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 lm92*) **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*

corollary *lm58*: **assumes** *distinct G set G* $\neq$ {} *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 lm92b lm57*
proof –
have *card ?X*=*1* **using** *assms* **by** (*rule lm92b*) **moreover have** *?X* ⊆ *?R* **by** *auto*
ultimately show *?thesis* **using** *cardinalityOneTheElem* **by** *blast*
qed

corollary *lm58b*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
vcga' *N G b r* ∈ (*Outside'* {*seller*})‘(*maximalStrictAllocations'* *N* (*set G*) *b*)
using *assms lm58* **by** *blast*

lemma *lm62*: (*Outside'* {*seller*})‘(*maximalStrictAllocations'* *N G b*) ⊆ *soldAllocations N G*
using *Outside-def* **by** *force*

theorem *lm58d*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
vcga' *N G b r* ∈ *soldAllocations N* (*set G*) (**is** *?a* ∈ *?A*) **using** *assms lm58b lm62*
proof – **have** *?a* ∈ (*Outside'* {*seller*})‘(*maximalStrictAllocations'* *N* (*set G*) *b*)

using *assms* **by** (*rule lm58b*) **thus** *?thesis* **using** *lm62* **by** *fastforce* **qed**
corollary *lm58f*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
vcga N G b r $\in$ *soldAllocations N* (*set G*) (**is** -$\in$*?r*)
proof − **have** *vcga′ N G b r* $\in$ *?r* **using** *assms* **by** (*rule lm58d*) **then show**
?thesis **using** *assms lm002* **by** *blast* **qed**

corollary *lm59b*: **assumes** $\forall$ *X*. *X* $\in$ *Range a* $\longrightarrow$*b* (*seller*, *X*)=*0* *finite a* **shows**

setsum b a = *setsum b* (*a*−−*seller*)
proof −
let *?n*=*seller* **have** *finite* (*a*||{*?n*}) **using** *assms restrict-def* **by** (*metis finite-Int*)

moreover have $\forall$ *z* $\in$ *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** *setsumOutside assms*(*2*) **by** (*metis comm-monoid-add-class.add.right-neutral*)
qed

corollary *lm59c*: **assumes** $\forall$ *a*$\in$*A*. *finite a* & ($\forall$ *X*. *X*$\in$*Range a* $\longrightarrow$ *b* (*seller*, *X*)=*0*)
shows {*setsum b a*| *a*. *a*$\in$*A*}={*setsum b* (*a* −− *seller*)| *a*. *a*$\in$*A*} **using** *assms lm59b*
by (*metis* (*lifting*, *no-types*))
corollary *lm58c*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
EX a. ((*a* $\in$ (*maximalStrictAllocations′ N* (*set G*) *b*))
& (*vcga′ N G b r* = *a* −− *seller*)
& (*a* $\in$ *argmax* (*setsum b*) (*allAllocations* ({*seller*}$\cup$*N*) (*set G*)))
) (**is** *EX a*. - & - & *a* $\in$ *?X*)
using *assms lm58b argmax-def* **by** *fast*

lemma assumes *distinct G set G* $\neq$ {} *finite N* **shows**
$\forall$ *aa*$\in$*allAllocations* ({*seller*}$\cup$*N*) (*set G*). *finite aa*
using *assms* **by** (*metis List.finite-set UniformTieBreaking.lm44*)

lemma *lm61*: **assumes** *distinct G set G* $\neq$ {} *finite N*
$\forall$ *aa*$\in$*allAllocations* ({*seller*}$\cup$*N*) (*set G*). $\forall$ *X* $\in$ *Range aa*. *b* (*seller*, *X*)=*0*
(**is** $\forall$ *aa*$\in$*?X*. -) **shows** *setsum b* (*vcga′ N G b r*)=*Max*{*setsum b aa*| *aa*. *aa* $\in$ *soldAllocations N* (*set G*)}
proof −
let *?n*=*seller* **let** *?s*=*setsum* **let** *?a*=*vcga′ N G b r* **obtain** *a* **where**
0: *a* $\in$ *maximalStrictAllocations′ N* (*set G*) *b* & *?a*=*a*−−*?n* &
(*a*$\in$*argmax* (*setsum b*) (*allAllocations*({*seller*}$\cup$*N*)(*set G*)))(**is** - & *?a*=- & *a*$\in$*?Z*)
using *assms*(*1*,*2*,*3*) *lm58c* **by** *blast* **have**
1: $\forall$ *aa*$\in$*?X*. *finite aa* & ($\forall$ *X*. *X*$\in$*Range aa* $\longrightarrow$ *b* (*?n*, *X*)=*0*) **using** *assms*(*4*)
List.finite-set UniformTieBreaking.lm44 **by** *metis* **have**
2: *a* $\in$ *?X* **using** *0* **by** *auto* **have** *a* $\in$ *?Z* **using** *0* **by** *fast*
then have *a* $\in$ *?X*$\cap${*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *lm78* **by** *simp*
then have *a* $\in$ {*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *lm78* **by** *simp*
moreover have *?s b* ‘ *?X* = {*?s b aa*| *aa*. *aa*$\in$*?X*} **by** *blast*
ultimately have *?s b a* = *Max* {*?s b aa*| *aa*. *aa*$\in$*?X*} **by** *auto*

moreover have $\{?s\ b\ aa|\ aa.\ aa{\in}?X\} = \{?s\ b\ (aa{-}{-}?n)|\ aa.\ aa{\in}?X\}$ **using** *1* **by** (*rule lm59c*)
moreover have ... $= \{?s\ b\ aa|\ aa.\ aa \in Outside'\ \{?n\}`?X\}$ **by** *blast*
moreover have ... $= \{?s\ b\ aa|\ aa.\ aa \in soldAllocations\ N\ (set\ G)\}$ **by** *simp*
ultimately have $Max\ \{?s\ b\ aa|\ aa.\ aa \in soldAllocations\ N\ (set\ G)\} = ?s\ b\ a$ **by** *presburger*
moreover have ... $= ?s\ b\ (a{-}{-}?n)$ **using** *1 2 lm59b* **by** (*metis* (*lifting*, *no-types*))
ultimately show $?s\ b\ ?a{=}Max\{?s\ b\ aa|\ aa.\ aa \in soldAllocations\ N\ (set\ G)\}$ **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** $distinct\ G\ set\ G \neq \{\}\ finite\ N\ \forall\ X.\ b\ (seller,\ X){=}0$
shows $setsum\ b\ (vcga'\ N\ G\ b\ r){=}Max\{setsum\ b\ aa|\ aa.\ aa \in soldAllocations\ N\ (set\ G)\}$
using *assms lm61* **by** *blast*

corollary *lm58e*: **assumes** $distinct\ G\ set\ G \neq \{\}\ finite\ N$ **shows**
$vcga'\ N\ G\ b\ r \in allocationsUniverse\ \&\ \bigcup\ Range\ (vcga'\ N\ G\ b\ r) \subseteq set\ G$ **using** *assms lm58b*
proof −
let $?a{=}vcga'\ N\ G\ b\ r$ **let** $?n{=}seller$
obtain a **where**
0: $?a{=}a\ {-}{-}\ seller\ \&\ a \in maximalStrictAllocations'\ N\ (set\ G)\ b$
using *assms lm58b* **by** *blast*
then moreover have
1: $a \in allAllocations\ (\{?n\}{\cup}N)\ (set\ G)$ **by** *auto*
moreover have $maximalStrictAllocations'\ N\ (set\ G)\ b \subseteq allocationsUniverse$
by (*metis* (*lifting*, *mono-tags*) *UniformTieBreaking.lm03 Universes.lm50 subset-trans*)
ultimately moreover have $?a{=}a\ {-}{-}\ seller\ \&\ a \in allocationsUniverse$ **by** *blast*
then have $?a \in allocationsUniverse$ **using** *lm35d* **by** *auto*
moreover have $\bigcup\ Range\ a{=}\ set\ G$ **using** *nn24c 1* **by** *metis*
then moreover have $\bigcup\ Range\ ?a \subseteq set\ G$ **using** *Outside-def 0* **by** *fast*
ultimately show *?thesis* **using** *lm35d Outside-def* **by** *blast*
qed

lemma $vcga'\ N\ G\ b\ r = the\text{-}elem\ ((argmax \circ setsum)\ (randomBids'\ N\ G\ b\ r)$
$((argmax \circ setsum)\ b\ (allAllocations\ (\{seller\}{\cup}N)\ (set\ G))))\ {-}{-}\ seller$ **by** *simp*

abbreviation $vcgp\ N\ G\ b\ r\ n ==$
$Max\ (setsum\ b\ `\ (soldAllocations\ (N{-}\{n\})\ (set\ G))) - (setsum\ b\ (vcga\ N\ G\ b\ r\ {-}{-}\ n))$

lemma *lm63*: **assumes** $x \in X\ finite\ X$ **shows** $Max\ (f`X) >= f\ x$ (**is** $?L >= ?R$)
using *assms*
by (*metis* (*hide-lams*, *no-types*) *Max.coboundedI finite-imageI image-eqI*)

lemma *lm59*: **assumes** *finite N finite G* **shows** *finite* (*soldAllocations N G*) **using**

assms lm59
finite.emptyI finite.insertI finite-UnI finite-imageI **by** *metis*

The price paid by any participant is non-negative.

theorem *NonnegPrices*: **assumes** *distinct G set G* $\neq$ {} *finite N* **shows**
vcgp N G b r n >= (*0::price*)
proof –
let *?a=vcga N G b r* **let** *?A=soldAllocations* **let** *?f=setsum b*
have *?a* $\in$ *?A N* (*set G*) **using** *assms* **by** (*rule lm58f*)
then have *?a* −− *n* $\in$ *?A* (*N*−{*n*}) (*set G*) **by** (*rule lm44b*)
moreover have *finite* (*?A* (*N*−{*n*}) (*set G*)) **using** *assms*(*3*) *lm59 finite-set finite-Diff* **by** *blast*
ultimately have *Max* (*?f*‘(*?A* (*N*−{*n*}) (*set G*))) $\geq$ *?f* (*?a* −− *n*) (**is** *?L* >= *?R*) **by** (*rule lm63*)
then show *?L* − *?R* >=*0* **by** *linarith*
qed

lemma *lm19b*: *allAllocations N G = possibleAllocationsRel N G* **using** *assms* **by** (*metis lm19*)
abbreviation *strictAllocationsUniverse == allocationsUniverse*

abbreviation *Goods bids ==* $\bigcup$((*snd*$\circ$*fst*)‘*bids*)

corollary *lm45*: **assumes** *a* $\in$ *soldAllocations‴ N G* **shows** *Range a* $\in$ *partitionsUniverse*
using *assms* **by** (*metis* (*lifting*, *mono-tags*) *Int-iff lm22 mem-Collect-eq*)

corollary *lm45a*: **assumes** *a* $\in$ *soldAllocations N G* **shows** *Range a* $\in$ *partitionsUniverse*
proof − **have** *a* $\in$ *soldAllocations‴ 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′* {*seller*}) ‘ (*terminatingAuctionRel N G* (*bids*) *random*) =
{*chosenAllocation′ N G bids random* −− (*seller*)} (**is** *?L=?R*) **using** *assms lm92 Outside-def*
proof −
have *?R = Outside′* {*seller*} ‘ {*chosenAllocation′ N G bids random*} **using** *Outside-def*

by *blast*
moreover have ... = (*Outside′*{*seller*})‘(*terminatingAuctionRel N G bids random*) **using** *assms lm92*
by *blast*
ultimately show *?thesis* **by** *presburger*
qed

lemma *terminatingAuctionRel N G b r* =
((*argmax* (*setsum* (*resolvingBid′ N G b* (*r*)))) $\circ$ (*argmax* (*setsum b*)))

(*possibleAllocationsRel N* (*set G*)) **by** *force*
term (*Union* ∘ (*argmax* (*setsum* (*resolvingBid′ N G b* (*r*)))) ∘ (*argmax* (*setsum b*)))
(*possibleAllocationsRel N* (*set G*))

lemma *maximalStrictAllocations′ N G b=winningAllocationsRel* ({*seller*} ∪ *N*) *G b* **by** *fast*

lemma *lm64*: **assumes** *a* ∈ *allocationsUniverse*
n1 ∈ *Domain a n2* ∈ *Domain a*
n1 ≠ *n2*
shows *a*,,*n1* ∩ *a*,,*n2*={} **using** *assms is-non-overlapping-def lm22 mem-Collect-eq*

proof − **have** *Range a* ∈ *partitionsUniverse* **using** *assms lm22* **by** *blast*
moreover have *a* ∈ *injectionsUniverse* & *a* ∈ *partitionValuedUniverse* **using** *assms* **by** (*metis* (*lifting*, *no-types*) *IntD1 IntD2*)
ultimately moreover have *a*,,*n1* ∈ *Range a* **using** *assms*
by (*metis* (*mono-tags*) *eval-runiq-in-Range mem-Collect-eq*)
ultimately moreover have *a*,,*n1* ≠ *a*,,*n2* **using**
assms converse.intros eval-runiq-rel mem-Collect-eq runiq-basic **by** (*metis* (*lifting*, *no-types*))
ultimately show *?thesis* **using** *is-non-overlapping-def* **by** (*metis* (*lifting*, *no-types*) *assms*(*3*) *eval-runiq-in-Range mem-Collect-eq*)
qed

lemma *lm64c*: **assumes** *a* ∈ *allocationsUniverse*
n1 ∈ *Domain a n2* ∈ *Domain a*
n1 ≠ *n2*
shows *a*,,,*n1* ∩ *a*,,,*n2*={} **using** *assms lm64 imageEquivalence* **by** *fastforce*

No good is assigned twice.

theorem *PairwiseDisjointAllocations*:
fixes *n1*::*participant* **fixes** *G*::*goods list* **fixes** *N*::*participant set*
assumes *distinct G set G* ≠ {} *finite N*

n1 ≠ *n2*
shows (*vcga′ N G b r*),,,*n1* ∩ (*vcga′ N G b r*),,,*n2*={}
proof −
have *vcga′ N G b r* ∈ *allocationsUniverse* **using** *lm58e assms* **by** *blast*
then show *?thesis* **using** *lm64c assms* **by** *fast*
qed

lemma assumes *R*,,,*x* ≠ {} **shows** *x* ∈ *Domain R* **using** *assms*
proof − **have** $\bigcup$ (*R*‘‘{*x*}) ≠ {} **using** *assms*(*1*) **by** *fast*
then have *R*‘‘{*x*} ≠ {} **by** *fast* **thus** *?thesis* **by** *blast* **qed**

lemma assumes *runiq f* **and** *x* ∈ *Domain f* **shows** (*f* ,, *x*) ∈ *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 eval-rel-def lm002* **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** *imageEquivalence* **by** *fastforce*
then have *g* $\in \bigcup$ *Range ?a* **using** *assms lm002* **by** *blast*
moreover have $\bigcup$ *Range ?a* $\subseteq$ *set G* **using** *assms*(*1*,*2*,*3*) *lm58e* **by** *fast*
ultimately show *?thesis* **by** *blast*
qed

definition *allStrictAllocations N G == possibleAllocationsAlg N G*
abbreviation *maximalStrictAllocations N G b==*
argmax (*setsum b*) (*set* (*allStrictAllocations* ({*seller*}$\cup$*N*) *G*))

definition *maximalStrictAllocations2 N G b=*
argmax (*setsum b*) (*set* (*allStrictAllocations* ({*seller*}$\cup$*N*) *G*))

definition *chosenAllocation N G b* (*r*::*integer*) ==
hd(*perm2* (*takeAll* (%*x*. *x*$\in$ (*argmax* $\circ$ *setsum*) *b* (*set* (*allStrictAllocations N G*)))
(*allStrictAllocations N G*)) (*nat-of-integer r*))

definition *chosenAllocationEff N G b* (*r*::*integer*) ==
(*takeAll* (%*x*. *x*$\in$ (*argmax* $\circ$ *setsum*) *b* (*set* (*allStrictAllocations N G*))) (*allStrictAllocations N G*) ! (*nat-of-integer r*))

definition *maxbid a N G ==* (*bidMaximizedBy a N G*) *Elsee 0*
definition *summedBidVector bids N G ==* (*summedBidVectorRel bids N G*) *Elsee 0*
definition *tiebids a N G == summedBidVector* (*maxbid a N G*) *N G*
definition *resolvingBid N G bids random == tiebids* (*chosenAllocation N G bids random*) *N* (*set G*)
definition *randomBids N* Ω *b random==resolvingBid* (*N*$\cup${*seller*}) Ω *b random*
definition *vcgaAlgWithoutLosers N G b r ==* (*the-elem*
(*argmax* (*setsum* (*randomBids N G b r*)) (*maximalStrictAllocations N G b*))) *-- seller*
abbreviation *addLosers participantset allo==*(*participantset* $\times$ {{}}) +* *allo*
definition *vcgaAlg N G b r = addLosers N* (*vcgaAlgWithoutLosers N G b r*)
abbreviation *allAllocationsComp N* Ω ==
(*Outside'* {*seller*}) ' *set* (*allStrictAllocations* (*N* $\cup$ {*seller*}) Ω)
definition *vcgpAlg N G b r n =*
Max (*setsum b* ' (*allAllocationsComp* (*N*−{*n*}) *G*)) − (*setsum b* (*vcgaAlgWithoutLosers*

N G b r −− *n*))

lemma *lm01*: **assumes** $x \in$ *Domain f* **shows** *toFunction f x* = (*f Elsee 0*) *x*
unfolding *toFunctionWithFallback2-def*
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** $x \in$ (*N* × (*Pow G* − {{}})) **shows** *summedBidVector' b N G x*=*summedBidVector b N G x* **unfolding** *summedBidVector-def*

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 finestpartSubset 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* ∈ *finestpart* (*snd pair*) **then have**
   0: *g* ∈ *finestpart G* **using** *assms finestpartSubset* **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*)

   then have (*pseudoAllocation a* <| (*N* × *finestpart G*)) (*fst pair*, *g*) = *maxbid a N G* (*fst pair*, *g*)
   unfolding *toFunctionWithFallback2-def maxbid-def*
   by (*metis* (*no-types*) *lm04 UnCI assms*(*1*) *toFunction-def*)

thus *?thesis* **unfolding** *maxbid-def* **by** *blast*
qed
}
thus *?thesis* **using** *setsum.cong* **by** *simp*
qed

corollary *lm07*: **assumes** *pair* $\in N \times (Pow\ G - \{\{\}\})$ **shows**
summedBid (*toFunction* (*bidMaximizedBy a N G*)) *pair* =
summedBid ((*bidMaximizedBy a N G*) *Elsee 0*) *pair* **using** *assms lm06*
proof –
have *fst pair* $\in N$ **using** *assms* **by** *force*
moreover have *snd pair* $\in Pow\ G - \{\{\}\}$ **using** *assms*(*1*) **by** *force*
ultimately show *?thesis* **using** *lm06* **by** *blast*
qed

lemma *lm08*: **assumes** $\forall x \in X.\ f\ x = g\ x$ **shows** *f`X=g`X* **using** *assms* **by** (*metis image-cong*)

corollary *lm09*: $\forall$ *pair* $\in N \times (Pow\ G - \{\{\}\})$.
summedBid (*toFunction* (*bidMaximizedBy a N G*)) *pair* =
summedBid ((*bidMaximizedBy a N G*) *Elsee 0*) *pair* **using** *lm07*
by *blast*

corollary *lm10*:
(*summedBid* (*toFunction* (*bidMaximizedBy a N G*))) ` $(N \times (Pow\ G - \{\{\}\}))$=
(*summedBid* ((*bidMaximizedBy a N G*) *Elsee 0*)) ` $(N \times (Pow\ G - \{\{\}\}))$ (**is** *?f1* ` *?Z* = *?f2* ` *?Z*)
proof –
have $\forall$ *z* $\in$ *?Z*. *?f1 z* = *?f2 z* **by** (*rule lm09*) **thus** *?thesis* **by** (*rule lm08*)
qed

corollary *lm11*: *summedBidVectorRel* (*toFunction* (*bidMaximizedBy a N G*)) *N G* =
summedBidVectorRel ((*bidMaximizedBy a N G*) *Elsee 0*) *N G* **using** *lm10* **by** *metis*

corollary *lm12*: *summedBidVectorRel* (*maxbid′ a N G*) *N G* = *summedBidVectorRel* (*maxbid a N G*) *N G*
unfolding *maxbid-def* **using** *lm11* **by** *metis*

lemma *lm13*: **assumes** $x \in (N \times (Pow\ G - \{\{\}\}))$ **shows**
summedBidVector′ (*maxbid′ a N G*) *N G x* = *summedBidVector* (*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 *summedBidVectorRel ?h1 N G* = *summedBidVectorRel ?h2 N G* **using** *lm12*

by *metis*
moreover have *summedBidVector ?h2 N G*=(*summedBidVectorRel ?h2 N G*) *Elsee 0*
unfolding *summedBidVector-def* **by** *fast*
ultimately have *summedBidVector ?h2 N G*=*summedBidVectorRel ?h1 N G Elsee 0* **by** *presburger*
moreover have ... x = (*toFunction* (*summedBidVectorRel ?h1 N G*)) x **using** *assms*
lm01 UniformTieBreaking.lm64 **by** (*metis* (*mono-tags*))
ultimately have *summedBidVector ?h2 N G x* = (*toFunction* (*summedBidVectorRel ?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* (*possibleAllocationsAlg N G*)
using *assms Universes.lm70b* **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*
unfolding *allStrictAllocations-def*
using *assms lm70c lm24b* **by** (*metis*(*no-types*))

corollary *lm70d*: **assumes** *card N* > *0 distinct G* **shows**
allAllocations N (*set G*) = *set* (*allStrictAllocations N G*)
unfolding *allStrictAllocations-def*
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*))
unfolding *allStrictAllocations-def*
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*
unfolding *chosenAllocation-def* **using** *assms lm70f allStrictAllocations-def* **by** (*metis*(*no-types*))
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* = *summedBidVector′* (*maxbid′ a N G*) *N G x* **by** *fast* **moreover have** ...=
summedBidVector (*maxbid a N G*) *N G x* **using** *assms* **by** (*rule lm13*) **ultimately show** *?thesis*
unfolding *tiebids-def* **by** *fast*
qed

lemma *lm14*: **assumes** *card N* > *0 distinct G a* ⊆ (*N* × (*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 ∀ *x* ∈ *a*. *tiebids′ ?c N* (*set G*) *x* = *tiebids ?c N* (*set G*) *x* **using** *assms*(*3*) *lm13b* **by** *blast*
ultimately have ∀ *x* ∈ *a*. *?r′ x* = *resolvingBid N G b r x* **unfolding** *resolvingBid-def* **by** *presburger*
thus *?thesis* **using** *setsum.cong* **by** *simp*
qed
lemma *lm15*: *allAllocations N G* ⊆ *Pow* (*N* × (*Pow G* − {{}})) **by** (*metis PowI lm40c subsetI*)
corollary *lm14b*: **assumes** *card N* > *0 distinct G a* ∈ *allAllocations N* (*set G*)
shows *setsum* (*resolvingBid′ N G b r*) *a* = *setsum* (*resolvingBid N G b r*) *a*
proof −
have *a* ⊆ *N* × (*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* ∈ *maximalStrictAllocations′ N* (*set G*) *b*
shows *setsum* (*randomBids′ N G b r*) *a* = *setsum* (*randomBids N G b r*) *a*
proof −
have *card* (*N*∪{*seller*})>*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* ∈ *allAllocations* (*N*∪{*seller*}) (*set G*) **using** *assms*(*3*) **by** *fastforce*
ultimately show *?thesis* **unfolding** *randomBids-def* **by** (*rule lm14b*)
qed

lemma *lm16*: **assumes** ∀ *x*∈*X*. *f x* = *g x* **shows** *argmax f X*=*argmax g X*
using *assms argmaxLemma Collect-cong image-cong*
by (*metis*(*no-types*,*lifting*))

corollary *lm92e*: **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 lm92b 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 lm92b*)
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=*{*seller*} $\cup$ *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 lm92e*)
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* (({*seller*}$\cup$*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* $\in$ *all-partitions G*}
by *auto*

lemma *allAllocations N* Ω = *Union*{{*a*^–*1*|*a*. *a*$\in$*injections Y N*}|*Y*. *Y*$\in$*all-partitions* Ω} **by** *auto*

term ($\sum i{\in}X.\ f\ i$)
term ($\bigcup i{\in}X.\ x\ i$)
abbreviation *endowment a n == a,,,n*
abbreviation *vcgEndowment N G b r n==endowment (vcga N G b r) n*

abbreviation *firstPriceP N Ω b r n ==*
b (n, winningAllocationAlg N Ω r b,, n)

lemma assumes ∀ *X. b (n, X)* ≥ *0* **shows**
firstPriceP N Ω b r n ≥ *0* **using** *assms* **by** *blast*

abbreviation *goods == sorted-list-of-set o Union o Range o Domain*

abbreviation *allocationPrettyPrint2 a == map (%x. (x, sorted-list-of-set(a,,x)))*
((sorted-list-of-set ∘ Domain) a)
definition *helper (list) == (((hd∘hd) list, set (list!1)), hd(list!2))*
definition *listBid2funcBid listBid = (helper‘(set listBid)) Elsee (0::integer)*

abbreviation *singleBidConverter x == ((fst x, set ((fst o snd) x)), (snd o snd) x)*
definition *Bid2funcBid b = set (map singleBidConverter b) Elsee (0::integer)*

abbreviation *participantsSet b == fst ‘ (set b)*
abbreviation *goodsList2 b == sorted-list-of-set (Union ((set o fst o snd) ‘(set b)))*

definition *allocation b r* = {*allocationPrettyPrint2*
(vcgaAlg ((participantsSet b)) (goodsList2 b) (Bid2funcBid b) r)
}

definition *payments b r = vcgpAlg ((participantsSet b)) (goodsList2 b) (Bid2funcBid b) r*
export-code *allocation payments chosenAllocationEff* **in** *Scala* **module-name** *VCG* **file** */dev/shm/VCG.scala*

abbreviation *b01 ==*

{
((*1*::*integer*,{*11*::*integer*, *12*, *13*}),*20*::*integer*),
((*1*,{*11*,*12*}),*18*),
((*2*,{*11*}),*10*),
((*2*,{*12*}),*15*),
((*2*,{*12*,*13*}),*18*),
((*3*,{*11*}),*2*),
((*3*,{*11*,*12*}),*12*),
((*3*,{*11*,*13*}),*17*),
((*3*,{*12*,*13*}),*18*),
((*3*,{*11*,*12*,*13*}),*19*),
((*4*,{*11*,*12*,*13*,*14*,*15*,*16*}),*19*)
}
value *participants b01*

end