Vickrey-Clarke-Groves (VCG) Auctions

M. B. Caminati[*] M. Kerber[*] C. Lange[†] C. Rowat[‡]

March 17, 2025

Abstract

A VCG auction (named after their inventors Vickrey, Clarke, and Groves) is a generalization of the single-good, second price Vickrey auction to the case of a combinatorial auction (multiple goods, from which any participant can bid on each possible combination). We formalize in this entry VCG auctions, including tie-breaking and prove that the functions for the allocation and the price determination are well-defined. Furthermore we show that the allocation function allocates goods only to participants, only goods in the auction are allocated, and no good is allocated twice. We also show that the price function is non-negative. These properties also hold for the automatically extracted Scala code.

Contents

[*]School of Computer Science, University of Birmingham, UK
[†]Fraunhofer IAIS and University of Bonn, Germany, and School of Computer Science, University of Birmingham, UK
[‡]Department of Economics, University of Birmingham, UK

1 Introduction

An auction mechanism is mathematically represented through a pair of functions (a, p): the first describes how some given goods at stake are allocated among the bidders (also called participants or agents), while the second specifies how much each bidder pays following this allocation. Each possible output of this pair of functions is referred to as an outcome of the auction. Both functions take the same argument, which is another function, commonly called a bid vector b; it describes how much each bidder values the possible outcomes of the auction. This valuation is usually expressed through money. In this setting, some common questions are the study of the quantitative and qualitative properties of a given auction mechanism (e.g., whether it maximizes some relevant quantity, such as revenue, or whether it is efficient, that is, whether it allocates the item to the bidder who values it most), and the study of the algorithms running it (in particular, their correctness).

A VCG auction (named after their inventors Vickrey, Clarke, and Groves) is a generalization of the single-good, second price Vickrey auction to the case of a combinatorial auction (multiple goods, from which any participant can bid on each possible combination). We formalize in this entry VCG auctions, including tie-breaking and prove that the functions a and p are well-defined. Furthermore we show that the allocation function a allocates goods only to participants, only goods in the auction are allocated, and no good is allocated twice. Furthermore we show that the price function p is non-negative. These properties also hold for the automatically extracted Scala code. For further details on the formalization, see [4]. For background information on VCG auctions, see [5].

The following files are part of the Auction Theory Toolbox (ATT) [1] developed in the ForMaRE project [2]. The theories `CombinatorialAuction.thy`, `StrictCombinatorialAuction.thy` and `UniformTieBreaking.thy` contain the relevant definitions and theorems; `CombinatorialAuctionExamples.thy` and `CombinatorialAuctionCodeExtraction.thy` present simple helper definitions to run them on given examples and to export them to the Scala language, respectively; `FirstPrice.thy` shows how easy it is to adapt the definitions to the first price combinatorial auction. The remaining theories contain more general mathematical definitions and theorems.

1.1 Rationale for developing set theory as replacing one bidder in a second price auction

Throughout the whole ATT, there is a duality in the way mathematical notions are modeled: either through objects typical of lambda calculus and HOL (lambda-abstracted functions and lists, for example) or through objects typical of set theory (for example, relations, intersection, union, set difference, Cartesian product).

This is possible because inside HOL, it is possible to model a simply-typed set theory which, although quite restrained if compared to, e.g., ZFC, is powerful enough for many standard mathematical purposes.

ATT freely adopts one approach, the other, or a mixture thereof, depending on technical and expressive convenience. A technical discussion of this topic can be found in [3].

1.2 Bridging

One of the differences between the approaches of functional definitions on the one hand and classical (often set-theoretical) definitions on the other hand is that, commonly (although not always), the first approach is better suited to produce Isabelle/HOL definitions which are computable (typically, inductive definitions); while the definitions from the second approach are often more general (e.g., encompassing infinite sets), closer to pen-and-paper mathematics, but also not computable. This means that many theorems are proved with respect to definitions of the second type, while in the end we want them to apply to definitions of the first type, because we want our theorems to hold for the code we will be actually running. Hence, bridging theorems are needed, showing that, for the limited portions of objects for which we state both kinds of definitions, they are the same.

1.3 Main theorems

The main theorems about VCG auctions are:

the definiteness theorem: our definitions grant that there is exactly one solution; this is ensured by `vcgaDefiniteness`.

PairwiseDisjointAllocations: no good is allocated to more than one participant.

onlyGoodsAreAllocated: only the actually available goods are allocated.

the adequacy theorem: the solution provided by our algorithm is indeed the one prescribed by standard pen-and-paper definition.

NonnegPrices: no participant ends up paying a negative price (e.g., no participant receives money at the end of the auction).

Bridging theorems: as discussed above, such theorems permit to apply the theorems in this list to the executable code Isabelle generates.

1.4 Scala code extraction

Isabelle permits to generate, from our definition of VCG, Scala code to run any VCG auction. Use `CombinatorialAuctionCodeExtraction.thy` for this. This code is in the form of Scala functions which can be evaluated on any input (e.g., a bidvector) to return the resulting allocation and prices.

To deploy such functions use the provided Scala wrapper (taking care of the output and including sample inputs). In order to do so, you can evaluate inside Isabelle/JEdit the file `CombinatorialAuctionCodeExtraction.thy` (position the cursor on its last line and wait for Isabelle/JEdit to end all its processing). This will result in the file `/dev/shm/VCG-withoutWrapper.scala`, which can be automatically appended to the wrapper by running the shell script at the end of `CombinatorialAuctionCodeExtraction.thy`. For details of how to run the Scala code see http://www.cs.bham.ac.uk/research/projects/formare/vcg.php.

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

theory *SetUtils*
imports
Main

begin

2.1 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*

2.2 Trivial sets

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

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

The empty set is trivial.

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

A singleton set is trivial.

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

If 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 $x = the\text{-}elem\ X$

  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*)

2.3 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*

2.4 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*

2.5 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* = (∀ *x1* ∈ *X*. ∀ *x2* ∈ *X*. *x1* = *x2*)
using *trivial-def trivial-imp-no-distinct ex-in-conv insertCI subsetI subset-singletonD*
trivial-singleton
by (*metis* (*no-types*, *opaque-lifting*))

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

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

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

lemma *lm11*: ({*x*} × *UNIV*) ∩ *P* = {*x*} × (*P* `` {*x*})
by *fast*

lemma *lm12*: (*x*,*y*) ∈ *P* = (*y* ∈ *P*``{*x*})
by *simp*

lemma *lm13*: **assumes** *inj-on f A inj-on f B*
shows *inj-on f* (*A* ∪ *B*) = (*f*`(*A*−*B*) ∩ (*f*`(*B*−*A*)) = {})
using *assms inj-on-Un* **by** (*metis*)

lemma *injection-union*: **assumes** *inj-on f A inj-on f B* (*f*`*A*) ∩ (*f*`*B*) = {}
shows *inj-on f* (*A* ∪ *B*)
using *assms lm13* **by** *fast*

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

end

3 Partitions of sets

theory *Partitions*
imports
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 $Y \cap Z \neq \{\}$
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 $Y \cap Z \neq \{\}$
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 $X \in P$
shows $X \cap \bigcup (P - \{X\}) = \{\}$
proof –
{
fix x::$'a$
assume *x-in-two-eq-classes*: $x \in X \cap \bigcup (P - \{X\})$
then obtain Y **where** *other-eq-class*: $Y \in P - \{X\} \wedge x \in Y$ **by** *blast*
have $x \in X \cap Y \wedge Y \in P$
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 $x \in \{\}$ **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 $\{\} \notin p$

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* **unfolding** *is-partition-of-def is-non-overlapping-def*
by *fastforce*

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_el into a specified set S inside a given family of sets

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

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* = $(S \cup \{new\text{-}el\})$ # (*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* ∪ {*elem*}) (*P* − {*set*}))

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

Assuming that *P* is a partition of a set *S*, and *new-el* ∉ *S*, the function defined below yields all possible partitions of *S* ∪ {*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 non_overlapping.

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 $Q \in$ (*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* ∨ *Q* = *insert* {*elem*} *P*
using *assms* **unfolding** *coarser-partitions-with-def* **by** *fast*
{
fix *eq-class* **assume** *eq-class-in-P*: *eq-class* ∈ *P*
have $\bigcup$ (*insert* (*eq-class* ∪ {*elem*}) (*P* − {*eq-class*})) = *?S* ∪ (*eq-class* ∪ {*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* ∪ {*elem*}) (*P* − {*eq-class*})) = *?S* ∪ {*elem*} **by** *blast*
then have $\bigcup$ (*insert-into-member elem P eq-class*) = *?S* ∪ {*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* :: $'a \Rightarrow 'a\ set\ set \Rightarrow 'a\ set\ set$
  where *partition-without elem P* = $(\lambda X\ .\ X - \{elem\})$ ‘ $P - \{\{\}\}$

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*) = $\bigcup$ (($\lambda x\ .\ x - \{elem\}$) ‘ $P - \{\{\}\}$)
    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* ∪ {*Y*}) − {{}} **using** *P-wrt-elem*
by *presburger*
also have ... = *?elem-neq-classes* ∪ {*?elem-eq*} − {{}}
using *Y-elem-eq elem-neq-classes-id image-Un* **by** *metis*
finally have *Q-wrt-elem*: *?Q* = *?elem-neq-classes* ∪ {*?elem-eq*} − {{}} **.**

have *?elem-eq* = {} ∨ *?elem-eq* ∉ *P*
using *elem-eq-class elem-eq-class′ non-overlapping Diff-Int-distrib2 Diff-iff empty-Diff insert-iff*
unfolding *is-non-overlapping-def* **by** *metis*
then have *?elem-eq* ∉ *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* ∉ *?Q*
then have *?elem-eq* ∈ {{}}
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*: ¬ *?elem-eq* ∉ *?Q*
hence *Y′*: *?elem-neq-classes* ∪ {*?elem-eq*} − {{}} = *?elem-neq-classes* ∪ {*?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*} ∪ *?elem-neq-classes*) *?elem-eq* = *insert* (*?elem-eq* ∪ {*elem*}) (({*?elem-eq*} ∪ *?elem-neq-classes*) − {*?elem-eq*})
unfolding *insert-into-member-def* **..**
also have ... = ({} ∪ *?elem-neq-classes*) ∪ {*?elem-eq* ∪ {*elem*}} **using** *elem-neq-classes* **by** *force*
also have ... = *?elem-neq-classes* ∪ {*Y*} **using** *elem-eq-class* **by** *blast*
finally have *insert-into-member elem* ({*?elem-eq*} ∪ *?elem-neq-classes*) *?elem-eq*

= *?elem-neq-classes* ∪ {*Y*} **.**
then have *?elem-neq-classes* ∪ {*Y*} = *insert-into-member elem ?Q ?elem-eq*
using *Q-wrt-elem Y′ partition-without-def*
by *force*
then have {*Y*} ∪ *?elem-neq-classes* ∈ *insert-into-member elem ?Q ‘ ?Q* **using** *True* **by** *blast*
then have {*Y*} ∪ *?elem-neq-classes* ∈ *coarser-partitions-with elem ?Q* **unfolding** *coarser-partitions-with-def* **by** *simp*
then show *?thesis* **using** *P-wrt-elem* **by** *simp*
qed
qed

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

definition *all-coarser-partitions-with* :: *′a ⇒ ′a set set set ⇒ ′a set set set*
where *all-coarser-partitions-with elem Ps* = ⋃ (*coarser-partitions-with elem ‘ Ps*)

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

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

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

lemma *all-coarser-partitions-with-list-equivalence*:
fixes *elem*::*′a*
and *Ps*::*′a set list list*
assumes *distinct*: ∀ *P* ∈ *set Ps* . *distinct P*
shows *set* (*map set* (*all-coarser-partitions-with-list elem Ps*)) = *all-coarser-partitions-with elem* (*set* (*map set Ps*))
(**is** *?list-expr* = *?set-expr*)
proof −
have *?list-expr* = *set* (*map set* (*concat* (*map* (*coarser-partitions-with-list elem*) *Ps*)))
unfolding *all-coarser-partitions-with-list-def* **..**
also have . . . = *set ‘* (⋃ *x* ∈ (*coarser-partitions-with-list elem*) *‘* (*set Ps*) . *set x*) **by** *simp*
also have . . . = *set ‘* (⋃ *x* ∈ { *coarser-partitions-with-list elem P* | *P* . *P* ∈ *set Ps* } . *set x*)
by (*simp add*: *image-Collect-mem*)
also have . . . = ⋃ { *set* (*map set* (*coarser-partitions-with-list elem P*)) | *P* . *P* ∈ *set Ps* } **by** *auto*
also have . . . = ⋃ { *coarser-partitions-with elem* (*set P*) | *P* . *P* ∈ *set Ps* }
using *distinct coarser-partitions-with-list-equivalence* **by** *fast*
also have . . . = ⋃ (*coarser-partitions-with elem ‘* (*set ‘* (*set Ps*))) **by** (*simp add*: *image-Collect-mem*)
also have . . . = ⋃ (*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* ⇒ *'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* ⇒ *'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* ∈ *set* (*coarser-partitions-with-list x Q*)
      and *distinct*: *distinct Q*
      and *partition*: *is-non-overlapping* (*set Q*)
      and *new*: {*x*} ∉ *set Q*
  shows *distinct ps*
proof −
  have *set* (*coarser-partitions-with-list x Q*) = *insert* ({*x*} # *Q*) (*set* (*map* (*insert-into-member-list x Q*) *Q*))
    unfolding *coarser-partitions-with-list-def* **by** *simp*
  with *ps-coarser* **have** *ps* ∈ *insert* ({*x*} # *Q*) (*set* (*map* ((*insert-into-member-list x Q*)) *Q*)) **by** *blast*
  then show *?thesis*
  proof
    assume *ps* = {*x*} # *Q*
    with *distinct* **and** *new* **show** *?thesis* **by** *simp*
  next
    assume *ps* ∈ *set* (*map* (*insert-into-member-list x Q*) *Q*)
    then obtain *X* **where** *X-in-Q*: *X* ∈ *set Q* **and** *ps-insert*: *ps* = *insert-into-member-list x Q X* **by** *auto*
      from *ps-insert* **have** *ps* = (*X* ∪ {*x*}) # (*remove1 X Q*) **unfolding** *insert-into-member-list-def* **.**
      also have ... = (*X* ∪ {*x*}) # (*removeAll X Q*) **using** *distinct* **by** (*metis distinct-remove1-removeAll*)
    finally have *ps-list*: *ps* = (*X* ∪ {*x*}) # (*removeAll X Q*) **.**

    have *distinct-tl*: *X* ∪ {*x*} ∉ *set* (*removeAll X Q*)
    proof
      from *partition* **have** *partition'*: ∀ *x*∈*set Q*. ∀ *y*∈*set Q*. (*x* ∩ *y* ≠ {}) = (*x* = *y*) **unfolding** *is-non-overlapping-def* **.**
      assume *X* ∪ {*x*} ∈ *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* ⟹
((*set* (*map set* (*all-partitions-list xs*)) =
all-partitions (*set xs*)) ∧ (∀ *ps* ∈ *set* (*all-partitions-list xs*) . *distinct ps*))
proof (*induct xs*)
case *Nil*
have *set* (*map set* (*all-partitions-list* [])) = *all-partitions* (*set* [])
unfolding *List.set-simps*(*1*) *emptyset-part-emptyset3* **by** *simp*

moreover have ∀ *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* ∈ *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* ∈ *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* ∈ *all-coarser-partitions-with x* (*all-partitions* (*set xs*)) **by** *fast*
then have *P* ∈ ⋃ (*coarser-partitions-with x* ‘ (*all-partitions* (*set xs*)))
unfolding *all-coarser-partitions-with-def* **.**
then obtain *Y*
where *P-in-Y*: *P* ∈ *Y*
and *Y-coarser*: *Y* ∈ *coarser-partitions-with x* ‘ (*all-partitions* (*set xs*)) **..**
from *Y-coarser* **obtain** *Q*
where *Q-part-xs*: *Q* ∈ *all-partitions* (*set xs*)
and *Y-coarser'*: *Y* = *coarser-partitions-with x Q* **..**
from *P-in-Y Y-coarser'* **have** *P-wrt-Q*: *P* ∈ *coarser-partitions-with x Q* **by** *fast*
then have *Q* ∈ *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*: ⋃ *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* ∈ *all-partitions* (*set* (*x # xs*)) **unfolding** *all-partitions-def* **..**
qed
moreover have ∀ *ps* ∈ *set* (*all-partitions-list* (*x # xs*)) . *distinct ps*
proof
fix *ps*::′*a set list* **assume** *ps-part*: *ps* ∈ *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* ∘ (*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* ∘ (*coarser-partitions-with-list x*)) ‘ (*set* (*all-partitions-list xs*))) **.**

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

from *qs* **have** *set qs* ∈ *set* (*map set* (*all-partitions-list* (*xs*))) **by** *simp*
with *distinct-xs hyp-equiv* **have** *qs-hyp*: *set qs* ∈ *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* ∉ *set xs* **by** *simp*
then have *new*: {*x*} ∉ *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* $\Rightarrow$ *'a set list list*
where *all-partitions-alg X* = *all-partitions-list* (*sorted-list-of-set X*)

end

4 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* $\Rightarrow$ *'b::linorder*) $\Rightarrow$ *'a set* $\Rightarrow$ *'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*) }
by *simp*

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

lemma *lm01*:
argmax f A = *A* ∩ *f* −' {*Max* (*f* ' *A*)}
by *force*

lemma *lm02*:
assumes *y* ∈ *f*'*A*
shows *A* ∩ *f* −' {*y*} ≠ {}
using *assms* **by** *blast*

lemma *argmaxEquivalence*:
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*))

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

corollary *argmax-non-empty-iff*: **assumes** *finite X X* ≠ {}
shows *argmax f X* ≠{}
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*)]
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*])
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*, *opaque-lifting*)) **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
```

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

```
theory RelationOperators
imports
  SetUtils
  HOL−Library.Code-Target-Nat

begin
```

5.1 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 × '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* $\Rightarrow$ *bool*
  where *runiq* $R = (\forall\ X\ .\ \mathit{trivial}\ X \longrightarrow \mathit{trivial}\ (R\ ``\ X))$

5.2 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 $R \mid\mid X = (X \times \mathit{Range}\ R) \cap R$

extensional characterization of the pairs within a restricted relation

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

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

lemma *restrict-ext'*: $R \mid\mid X = \{p\ .\ \mathit{fst}\ p \in X \wedge p \in R\}$
proof –
  have $R \mid\mid X = \{(x, y) \mid x\ y\ .\ x \in X \wedge (x, y) \in R\}$ **by** (*rule restrict-ext*)
  also have $\ldots = \{\ p\ .\ \mathit{fst}\ p \in X \wedge p \in R\ \}$ **by** *force*
  finally show *?thesis* **.**
qed

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

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

A restriction is a subrelation of the original relation.

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

Restricting a relation only has an effect within its domain.

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

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

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

5.3 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* × *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*)

5.4 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* ∈ R^{-1} } **using** *flip-in-conv* **by** (*metis converse-converse flip-flip*)
also have ... = R^{-1} **by** *simp*
finally show *?thesis* **.**
qed

5.5 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* × '*b*) *set* ⇒ '*a* ⇒ '*b* (**infix** ‹,,› *75*)
where R ,, a = *the-elem* (R ‘‘ {a})

5.6 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) ∪ 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 ∩ *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) ∩ *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

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

theory *RelationProperties*
imports
RelationOperators

begin

6.1 Right-Uniqueness

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

lemma *lm01*: *card P* = *card* (*P*^−*1*)
using *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* (*P*^−*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*: $((P\ ||\ X)\ ||\ Y) = (P\ ||\ (X \cap Y))$
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* **by** (*metis Image-empty2 trivial-empty-or-singleton trivial-singleton*)

an alternative definition of right-uniqueness in terms of (,,)

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 $f``(f^{-1}``Y) \subseteq Y$

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

lemma lm04:
  assumes runiq f y1 ∈ Range f
  shows (f^-1 `` {y1} ∩ f^-1 `` {y2} ≠ {}) = (f^-1``{y1}=f^-1``{y2})
  using assms rightUniqueFunctionAfterInverse by fast

lemma converse-Image:
  assumes runiq: runiq R
      and runiq-conv: runiq (R^-1)
  shows (R^-1) `` R `` X ⊆ 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) ∩ (Domain Q) = {}
  shows runiq (P ∪ 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 ∈ 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*)

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

6.3 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* . *Domain R* = *X* ∧ *Range R* ⊆ *Y* ∧ *runiq R* ∧ *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*
 where *injections-alg* [] *Y* = [{}] |
  injections-alg (*x* # *xs*) *Y* = *concat* [[*R* +∗ {(*x*,*y*)} . *y* ← *sorted-list-of-set* (*Y* − *Range R*)]
  . *R* ← *injections-alg xs Y*]

lemma *Image-within-domain'*:
 fixes *x R*
 shows (*x* ∈ *Domain R*) = (*R* ‘‘ {*x*} ≠ {})
 by *blast*

end

7 Toolbox of various definitions and theorems about sets, relations and lists

theory *MiscTools*

imports
HOL−Library.Discrete-Functions
HOL−Library.Code-Target-Nat
HOL−Library.Indicator-Function
Argmax
RelationProperties

begin

lemmas *restrict-def* = *RelationOperators.restrict-def*

7.1 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* ≡ *f outside* {*x*}

Turns a HOL function into a set-theoretical function

definition
Graph f = {(*x*, *f x*) | *x* . *True*}

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

definition
toFunction R = (λ *x* . (*R* ,, *x*))

lemma
toFunction = *eval-rel*
using *toFunction-def* **by** *blast*

lemma *lm001*:
((*P* ∪ *Q*) || *X*) = ((*P* || *X*) ∪ (*Q*||*X*))
unfolding *restrict-def* **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* || (*Domain P*))
notation *update* (**infix** ‹+⁀› *75*)

definition *runiqer* :: ('*a* × '*b*) *set* => ('*a* × '*b*) *set*
where *runiqer R* = { (*x*, *THE y*. *y* ∈ *R* `` {*x*})| *x*. *x* ∈ *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* || *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*) | *x*. *x* ∈ *X*}

lemma *lm002*:
assumes *runiq R*
shows *R* ⊇ *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* ⊆ *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*) == ⋃ (*R*``{*x*})
notation *eval-rel2* (**infix** ‹,,,› *75*)

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

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

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

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

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

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

lemma *lm007*:
assumes (*X* ∩ *Domain P*) ⊆ *Domain Q*

shows $P +* Q = (P \textit{ outside } X) +* Q$
unfolding *paste-def Outside-def* **using** *assms* **by** *blast*

corollary *lm008*:
$P +* Q = (P \textit{ outside } (\textit{Domain } Q)) +* Q$
using *lm007* **by** *fast*

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

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

corollary *lm010*:
$R = (R \textit{ outside } \{x\}) +* (\{x\} \times (R `` \{x\}))$
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 *rightUniqueTrivialCartes trivial-singleton runiq-paste2* **by** *metis*

lemma *lm015*:
$(P \,||\, (X \cap Y)) \subseteq (P||X) \quad \& \quad P \textit{ outside } (X \cup Y) \subseteq P \textit{ outside } X$
by (*metis doubleRestriction le-sup-iff outsideOutside outside-union-restrict subset-refl*)

lemma *lm016*:
$P \,||\, X \subseteq (P||(X \cup Y)) \quad \& \quad P \textit{ outside } X \subseteq P \textit{ outside } (X \cap Y)$

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*)

7.2 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 *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*∈*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 *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*:
$\{Y \mid Y.\ \exists x.((Y \in \textit{finestpart } x) \wedge (x \in X))\} = \bigcup(\textit{finestpart`X})$
by *auto*

lemma *rangeSetOfPairs*:
$\textit{Range } \{(\textit{fst pair}, Y) \mid Y \textit{ pair}.\ Y \in \textit{finestpart } (\textit{snd pair}) \ \&\ \textit{pair} \in X\} =$
$\{Y.\ \exists x.\ ((Y \in \textit{finestpart } x) \wedge (x \in \textit{Range } X))\}$
by *auto*

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

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

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

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

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

lemma *lm031*:
$\textit{argmax } f\ \{x\} = \{x\}$
by *auto*

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

lemma *lm032*:
assumes *finite A*

shows *sum f A* = *sum f* (*A* ∩ *B*) + *sum f* (*A* − *B*)
using *assms* **by** (*metis DiffD2 Int-iff Un-Diff-Int Un-commute finite-Un sum.union-inter-neutral*)

corollary *sumOutside*:
assumes *finite g*
shows *sum f g* = *sum f* (*g outside X*) + (*sum f* (*g*||*X*))
unfolding *Outside-def restrict-def* **using** *assms add.commute inf-commute lm032*
by (*metis*)

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

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

lemma *lm035*:
(*Domain P* ⊆ *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*) ⊆ *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 *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* ⊆ *X x* ∈ *XX x* ≠ {}
shows *x* ∩ *X* ≠ {}
using *assms* **by** *blast*

lemma *lm044*:
assumes ∀ *l* ∈ *set L1*. *set L2* = *f2* (*set l*) *N*
shows *set* [*set L2*. *l* <− *L1*] = {*f2 P N*| *P*. *P* ∈ *set* (*map set L1*)}
using *assms* **by** *auto*

lemma *setVsList*:
assumes ∀ *l* ∈ *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* ∈ *set* (*map set* (*g1 G*))}
using *assms* **by** *auto*

lemma *lm045*:
(∀ *l* ∈ *set* (*g1 G*). *set* (*g2 l N*) = *f2* (*set l*) *N*) −−>
{*f2 P N*| *P*. *P* ∈ *set* (*map set* (*g1 G*))} = *set* [*set* (*g2 l N*). *l* <− *g1 G*]
by *auto*

lemma *lm046*:
assumes *X* ∩ *Y* = {}
shows *R*‘‘*X* = (*R outside Y*)‘‘*X*
using *assms Outside-def Image-def* **by** *blast*

lemma *lm047*:
assumes (*Range P*) ∩ (*Range Q*) = {} *runiq* (*P*^−*1*) *runiq* (*Q*^−*1*)
shows *runiq* ((*P* ∪ *Q*)^−*1*)
using *assms* **by** (*metis Domain-converse converse-Un disj-Un-runiq*)

lemma *lm048*:
assumes (*Range P*) ∩ (*Range Q*) = {} *runiq* (*P*^−*1*) *runiq* (*Q*^−*1*)
shows *runiq* ((*P* +∗ *Q*)^−*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 (λ*a*. ((*fst a*, *fst* (*snd a*)), *snd* (*snd a*)))
  by (*auto intro*: *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* ≠ {}
  shows *Max* (*f*'*A*) ∈ *f*'*A*
  using *assms* **by** (*metis Max-in finite-imageI image-is-empty*)

lemma *lm053*:
  argmax f A ⊆ *f* −' {*Max* (*f* ' *A*)}
  by *force*

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

lemma *lm055*:
  (*x* ∈ *argmax f X*) = (*x* ∈ *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* ' ($P^{\frown}{-1}$)
  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*^−*1*)
using *lm060 flip-conv* **by** (*metis converse-converse inj-on-imageI lm059*)

lemma *sumPairsInverse*:
assumes *runiq* (*P*^−*1*)
shows *sum* (*f* ∘ *snd*) *P* = *sum f* (*Range P*)
using *assms lm062 converse-converse rightUniqueInjectiveOnFirst rightUniqueInjectiveOnFirst*
sum.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 *sum f* (*g*'*X*) = *sum* (*f* ∘ *g*) *X*
using *assms* **by** (*metis sum.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 *sum* (*toFunction R*) (*Domain R*) = *sum snd R*
using *assms toFunction-def sum.reindex-cong functionOnFirstEqualsSecond*

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

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

lemma *lm066*:
Range (*R outside X*) = $R``((Domain\ R) - X)$
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 *sum* (*toFunction* $(f||X)$) $(X \cap$ *Domain* $f)$ = *sum* (*toFunction* f) $(X \cap$ *Domain* $f)$
using *assms sum.cong lm069 toFunction-def* **by** *metis*

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

corollary *sumRestrictedOnFunction*:
assumes *runiq* $(f||X)$
shows *sum* (*toFunction* $(f||X)$) $(X \cap$ *Domain* $f)$ = *sum 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*:
sum ((*curry f*) *x*) *Y* = *sum f* ({*x*} × *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*} × *Y* = *?f* ‘ *Y* **by** *fast*
moreover have ∀ *y*. *y* ∈ *Y* ⟶ *?g y* = *?h* (*?f y*) **by** *simp*
ultimately show *?thesis* **using** *sum.reindex-cong* **by** *metis*
qed

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

corollary *lm078*:
assumes *finite X*
shows *sum f X* = *sum f* (*X*−*Y*) + (*sum f* (*X* ∩ *Y*))
using *assms Diff-iff IntD2 Un-Diff-Int finite-Un inf-commute sum.union-inter-neutral*

by *metis*

lemma *lm079*:
(*P* +∗ *Q*)‘‘(*Domain Q*∩*X*) = *Q*‘‘(*Domain Q*∩*X*)
unfolding *paste-def Outside-def Image-def Domain-def* **by** *blast*

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

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

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

corollary *lm083*:
assumes *X*∩ (*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 *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 *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*)

7.3 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*:
sum (*X* <| *Y*) $(X \cup Y)$ = *sum* (*chi X*) $(X \cup Y)$
using *lm102 sum.cong* **by** *metis*

corollary *lm104*:
assumes ∀ *x*∈*X*. *f x* = *g x*
shows *sum f X* = *sum g X*
using *assms* **by** (*metis* (*poly-guards-query*) *sum.cong*)

corollary *lm105*:
assumes ∀ *x*∈*X*. *f x* = *g x Y*⊆*X*
shows *sum f Y* = *sum g Y*
using *assms lm104* **by** (*metis contra-subsetD*)

corollary *lm106*:
assumes *Z* ⊆ *X* ∪ *Y*
shows *sum* (*X* <| *Y*) *Z* = *sum* (*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*:
sum (*chi X*) (*Z* − *X*) = *0*
by *simp*

corollary *lm108*:
assumes *Z* ⊆ *X* ∪ *Y*
shows *sum* (*X* <| *Y*) (*Z* − *X*) = *0*
using *assms lm107 lm106 Diff-iff in-mono subsetI* **by** *metis*

corollary *lm109*:
assumes *finite Z*
shows *sum* (*X* <| *Y*) *Z* = *sum* (*X* <| *Y*) (*Z* − *X*) + (*sum* (*X* <| *Y*) (*Z* ∩ *X*))
using *lm078 assms* **by** *blast*

corollary *lm110*:
assumes *Z* ⊆ *X* ∪ *Y finite Z*
shows *sum* (*X* <| *Y*) *Z* = *sum* (*X* <| *Y*) (*Z* ∩ *X*)
using *assms lm078 lm108 comm-monoid-add-class.add-0* **by** *metis*

corollary *lm111*:
assumes *finite Z*
shows *sum* (*chi X*) *Z* = *card* (*X* ∩ *Z*)
using *assms sum-indicator-eq-card* **by** (*metis Int-commute*)

corollary *lm112*:
assumes *Z* ⊆ *X* ∪ *Y finite Z*
shows *sum* (*X* <| *Y*) *Z* = *card* (*Z* ∩ *X*)
using *assms lm111* **by** (*metis lm106 sum-indicator-eq-card*)

corollary *subsetCardinality*:
assumes *Z* ⊆ *X* ∪ *Y finite Z*
shows (*sum* (*X* <| *Y*) *X*) − (*sum* (*X* <| *Y*) *Z*) = *card X* − *card* (*Z* ∩ *X*)
using *assms lm112* **by** (*metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite*)

corollary *differenceSumVsCardinality*:
assumes *Z* ⊆ *X* ∪ *Y finite Z*
shows *int* (*sum* (*X* <| *Y*) *X*) − *int* (*sum* (*X* <| *Y*) *Z*) = *int* (*card X*) − *int* (*card* (*Z* ∩ *X*))
using *assms lm112* **by** (*metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite*)

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

corollary *differenceSumVsCardinalityReal*:
assumes *Z*⊆*X*∪*Y finite Z*
shows *real* (*sum* (*X* <| *Y*) *X*) − *real* (*sum* (*X* <| *Y*) *Z*) =
real (*card X*) − *real* (*card* (*Z* ∩ *X*))
using *assms lm112* **by** (*metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite*)

7.4 Lists

lemma *lm114*:
assumes ∃ *n* ∈ {*0*..<*size l*}. *P* (*l*!*n*)
shows [*n*. *n* ← [*0*..<*size l*], *P* (*l*!*n*)] ≠ []
using *assms* **by** *auto*

lemma *lm115*:
assumes *ll* ∈ *set* (*l*::′*a list*)
shows ∃ *n*∈ (*nth l*) −‘ (*set l*). *ll*=*l*!*n*
using *assms*(*1*) **by** (*metis in-set-conv-nth vimageI2*)

lemma *lm116*:
assumes *ll* ∈ *set* (*l*::′*a list*)
shows ∃ *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*

7.5 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* ≠ []
shows *perm2 l n* ≠ []
using *assms 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 *takeAllSubset*:
set (*takeAll* (%*x*. *x*∈ *P*) *list*) ⊆ *P*
by (*metis Int-subset-iff listIntersectionWithSet*)

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 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*

abbreviation *subList l xl* == *map* (*nth l*) (*takeAll* (%*x*. *x* ≤ *size l*) *xl*)

7.6 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 *sum′* **where**
sum′ R X == *sum* (*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 *sum* (*f Else 0*) (*X*∩(*Domain f*)) = *sum* (*toFunction f*) (*X*∩(*Domain f*))

using *assms sum.cong lm129* **by** *fastforce*

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

lemma *lm132*:
assumes *Y* ⊆ *f*−‘{*0*} *finite X*
shows *sum f X* = *sum f* (*X*−*Y*)
using *Int-lower2 add.comm-neutral assms*(*1*) *assms*(*2*) *lm078 lm131 order-trans*
by (*metis* (*no-types*))

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

by *fastforce*

corollary *lm134*:
assumes *finite X*
shows *sum* (*f Else 0*) *X* = *sum* (*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 *sum* (*f Else 0*) (*X*∩*Domain f*) = *sum* (*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 *sum* (*f Else 0*) *X* = *sum* (*toFunction f*) (*X*∩*Domain f*)
(**is** *?L*=*?R*)
proof −
have *?R* = *sum* (*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*:
sum (*f Else 0*) *X* = *sum′ f X*
by *fast*

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

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

lemma *domainConstant*:
Domain (*Y* × {*0*::*nat*}) = *Y* & *Domain* (*X* × {*1*}) = *X*
by *blast*

lemma *domainCharacteristicFunction*:
Domain (*X* <|| *Y*) = *X* ∪ *Y*
using *domainConstant paste-Domain sup-commute* **by** *metis*

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

7.7 Cardinalities of sets.

lemma *lm140*:
assumes *runiq R runiq* ($R\hat{}-1$)
shows $(R``A) \cap (R``B) = R``(A \cap B)$
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 `` ($f\hat{}-1$ `` 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=floor-log* (*card* (*Pow A*))
using *assms floor-log-power card-Pow* **by** (*metis card.infinite finite-Pow-iff*)

lemma *log-2* [*simp*]:
floor-log 2 = 1
using *floor-log-power* [*of 1*] **by** *simp*

lemma *lm147*:
assumes *card* (*Pow A*) = *2*
shows *card A* = *1*
using *assms lm146* [*of A*] **by** *simp*

lemma *lm148*:
assumes *card* (*Pow X*) = *1* ∨ *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*) ∈ {*1*,*2*})
using *lm148 lm142* **by** *blast*

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

lemma *lm151*:
assumes *f* ⊆ *g runiq g Domain f* = *Domain g*
shows *g* ⊆ *f*
using *assms Domain-iff contra-subsetD runiq-wrt-ex1 subrelI*
by (*metis* (*full-types*,*opaque-lifting*))

lemma *lm152*:
assumes *R* ⊆ *f runiq f Domain f* ⊆ *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* ∩ *Y*) *f* = (*graph X f*) || *Y*
using *doubleRestriction lm153* **by** *metis*

lemma *restrictionVsIntersection*:
{(*x*, *f x*)| *x*. *x* ∈ *X2*} || *X1* = {(*x*, *f x*)| *x*. *x* ∈ *X2* ∩ *X1*}

using *graph-def lm154* **by** *metis*

lemma *lm155*:
assumes *runiq f X* ⊆ *Domain f*
shows *graph X* (*toFunction f*) = (*f*||*X*)
proof −
have ⋀*v w*. (*v*::′*a set*) ⊆ *w* ⟶ *w* ∩ *v* = *v* **by** (*simp add*: *Int-commute inf.absorb1*)
thus *graph X* (*toFunction f*) = *f* || *X* **by** (*metis assms*(*1*) *assms*(*2*) *doubleRestriction lm004 lm153*)
qed

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

lemma *lm157*:
assumes *X* ⊆ *Domain f runiq f*
shows *f*``*X* = (*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* ≠ {} **using** *assms* **by** *fastforce*
moreover have *card* (*f*`*A*) ≤ *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* ((*case-prod f*) ∘ *flip*)

lemma *lm158*:
finite X = (*X* ∈ *range set*)

by (*metis List.finite-set finite-list image-iff rangeI*)

lemma *lm159*:
finite = (%*X*. *X*∈*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*)

7.8 Some easy properties on real numbers

lemma *lm161*:
fixes *a*::*real*
fixes *b c*
shows *a*∗*b* − *a*∗*c*=*a*∗(*b*−*c*)
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 *lm161* **by** (*metis mult.commute*)

end

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

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

begin

8.1 Types

type-synonym *index* = *integer*
type-synonym *participant* = *index*
type-synonym *good* = *integer*
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*) == *case-prod b*
abbreviation *proceeds* (*b*::*bidvector*) (*allo*::*allocation*) == *sum b allo*
abbreviation *winnersOfAllo* (*a*::*allocation*) == *Domain a*
abbreviation *allocatedGoods* (*allo*::*allocation*) == ⋃ (*Range allo*)

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

abbreviation *is-partition-of′ P A* == (⋃ *P* = *A* ∧ *is-non-overlapping P*)
abbreviation *all-partitions′ A* == {*P* . *is-partition-of′ P A*}

abbreviation *possible-allocations-rel′ G N* == *Union*{*injections Y N* | *Y* . *Y* ∈ *all-partitions′ G*}
abbreviation *allAllocations* **where**
allAllocations 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 *allAllocationsAlg N G* ==
map converse (*concat* [(*injections-alg l N*) . *l* ← *all-partitions-list G*])

8.2 VCG mechanism

abbreviation *winningAllocationsRel N G b* ==
argmax (*sum b*) (*allAllocations N G*)

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

abbreviation *winningAllocationsAlg N G b* == *argmaxList* (*proceeds b*) (*allAllocationsAlg 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* ((*sum b*)‘(*allAllocations* (*N*−{*n*}) *G*))

abbreviation *alphaAlg N G b n* == *Max* ((*proceeds b*)‘(*set* (*allAllocationsAlg* (*N*−{*n*}) (*G*::- *list*))))

abbreviation *remainingValueRel N G t b n* == *sum 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

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

theory *Universes*

imports
HOL−Library.Code-Target-Nat
StrictCombinatorialAuction
HOL−Library.Indicator-Function

begin

9.1 Preliminary lemmas

lemma *lm001*:
  assumes *Y* ∈ *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 *lm002*:
  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*

9.2 Definitions of various subsets of *UNIV*.

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

abbreviation *partitionValuedUniverse* == $\bigcup$ $P \in$ *partitionsUniverse*. *Pow* (*UNIV* $\times$ P)
lemma *partitionValuedUniverse* $\subseteq$ *Pow* (*UNIV* $\times$ (*Pow UNIV*))
by *simp*

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

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

9.3 Results about the sets defined in the previous section

lemma *lm003*:
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 *lm004*:
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 *lm006*: *injections X Y* $\subseteq$ *injectionsUniverse*
using *injections-def* **by** *fast*

lemma *lm007*: *injections X Y* $\subseteq$ *injectionsUniverse*
using *injections-def* **by** *blast*

lemma *lm008*: *injections X Y* = *totalRels X Y* $\cap$ *injectionsUniverse*
using *injections-def* **by** (*simp add*: *Collect-conj-eq Int-assoc*)

lemma *allocationInverseRangeDomainProperty*:
assumes $a \in$ *allAllocations N G*
shows $a\hat{}{-1} \in$ *injections* (*Range a*) *N* &

(*Range a*) *partitions G* &
Domain a ⊆ *N*
unfolding *injections-def* **using** *assms all-partitions-def injections-def* **by** *fastforce*

lemma *lm009*:
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 rev-subsetD*)
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 *allocationRightUniqueRangeDomain*:
assumes *a* ∈ *possible-allocations-rel G N*
shows *runiq a* &
runiq (a^{-1}) &
(*Domain a*) *partitions G* &
Range a ⊆ *N*
proof −
obtain *Y* **where**
0: *a* ∈ *injections Y N* & *Y* ∈ *all-partitions G* **using** *assms* **by** *auto*
show *?thesis* **using** *0 injections-def all-partitions-def mem-Collect-eq* **by** *fastforce*
qed

lemma *lm010*:
assumes *runiq a runiq* (a^{-1}) (*Domain a*) *partitions G Range a* ⊆ *N*
shows *a* ∈ *possible-allocations-rel G N*
proof −
have *a* ∈ *injections* (*Domain a*) *N* **unfolding** *injections-def*
using *assms*(*1*) *assms*(*2*) *assms*(*4*) **by** *blast*
moreover have *Domain a* ∈ *all-partitions G* **using** *assms*(*3*) *all-partitions-def* **by** *fast*
ultimately show *?thesis* **using** *assms*(*1*) **by** *auto*
qed

lemma *allocationProperty*:
a ∈ *possible-allocations-rel G N* ⟷
runiq a & *runiq* (a^{-1}) & (*Domain a*) *partitions G* & *Range a* ⊆ *N*

using *allocationRightUniqueRangeDomain lm010* **by** *blast*

lemma *lm011*:
possible-allocations-rel′ G N ⊆ *injectionsUniverse*
using *injections-def* **by** *force*

lemma *lm012*:
possible-allocations-rel G N ⊆ {*a*. (*Range a*) ⊆ *N* & (*Domain a*) ∈ *all-partitions G*}
using *injections-def* **by** *fastforce*

lemma *lm013*:
injections X Y = *injections X Y*
using *injections-def* **by** *metis*

lemma *lm014*:
all-partitions X = *all-partitions′ X*
using *all-partitions-def is-partition-of-def* **by** *auto*

lemma *lm015*:
possible-allocations-rel′ A B = *possible-allocations-rel A B*
(**is** *?A*=*?B*)
proof −
have *?B*=⋃ { *injections Y B* | *Y* . *Y* ∈ *all-partitions A* }
by *auto*
moreover have ... = *?A* **using** *lm014* **by** *metis*
ultimately show *?thesis* **by** *presburger*
qed

lemma *lm016*:
possible-allocations-rel G N ⊆
injectionsUniverse ∩ {*a*. *Range a* ⊆ *N* & *Domain a* ∈ *all-partitions G*}
using *lm012 lm011 injections-def* **by** *fastforce*

lemma *lm017*:
possible-allocations-rel G N ⊇
injectionsUniverse ∩ {*a*. *Domain a* ∈ *all-partitions G* & *Range a* ⊆ *N*}
using *injections-def* **by** *auto*

lemma *lm018*:
possible-allocations-rel G N =
injectionsUniverse ∩ {*a*. *Domain a* ∈ *all-partitions G* & *Range a* ⊆ *N*}
using *lm016 lm017* **by** *blast*

lemma *lm019*:
converse ‘ *injectionsUniverse* = *injectionsUniverse*
by *auto*

lemma *lm020*:
converse‘(*A* ∩ *B*) = (*converse*‘*A*) ∩ (*converse*‘*B*)
by *force*

lemma *allocationInjectionsUnivervseProperty*:
allAllocations N G =
injectionsUniverse ∩ {*a. Domain a* ⊆ *N* & *Range a* ∈ *all-partitions G*}
proof −
let *?A=possible-allocations-rel G N*
let *?c=converse*
let *?I=injectionsUniverse*
let *?P=all-partitions G*
let *?d=Domain*
let *?r=Range*
have *?c*‘*?A* = (*?c*‘*?I*) ∩ *?c*‘ ({*a. ?r a* ⊆ *N* & *?d a* ∈ *?P*}) **using** *lm018* **by** *fastforce*
moreover have ... = (*?c*‘*?I*) ∩ {*aa. ?d aa* ⊆ *N* & *?r aa* ∈ *?P*} **by** *fastforce*
moreover have ... = *?I* ∩ {*aa. ?d aa* ⊆ *N* & *?r aa* ∈ *?P*} **using** *lm019* **by** *metis*
ultimately show *?thesis* **by** *presburger*
qed

lemma *lm021*:
allAllocations N G ⊆ *injectionsUniverse*
using *allocationInjectionsUnivervseProperty* **by** *fast*

lemma *lm022*:
allAllocations N G ⊆ *partitionValuedUniverse*
using *allocationInverseRangeDomainProperty is-partition-of-def is-non-overlapping-def*

by *auto blast*

corollary *allAllocationsUniverse*:
allAllocations N G ⊆ *allocationsUniverse*
using *lm021 lm022* **by** (*metis* (*lifting*, *mono-tags*) *inf.bounded-iff*)

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

corollary *lm023*:
assumes *a* ∈ *allAllocations N1 G*
shows *a* ∈ *allAllocations* (*N1* ∪ *N2*) *G*
proof −
have *Domain a* ⊆ *N1* ∪ *N2* **using** *assms*(*1*) *posssibleAllocationsRelCharacterization*
by (*metis le-supI1*)

moreover have *a* ∈ *injectionsUniverse* & *Range a* ∈ *all-partitions G*
using *assms posssibleAllocationsRelCharacterization* **by** *blast*
ultimately show *?thesis* **using** *posssibleAllocationsRelCharacterization* **by** *blast*

qed

corollary *lm024*:
allAllocations N1 G ⊆ *allAllocations* (*N1* ∪ *N2*) *G*
using *lm023* **by** (*metis subsetI*)

lemma *lm025*:
assumes (⋃ *P1*) ∩ (⋃ *P2*) = {}
is-non-overlapping P1 is-non-overlapping P2
X ∈ *P1* ∪ *P2 Y* ∈ *P1* ∪ *P2 X* ∩ *Y* ≠ {}
shows (*X* = *Y*)
unfolding *is-non-overlapping-def* **using** *assms is-non-overlapping-def* **by** *fast*

lemma *lm026*:
assumes (⋃ *P1*) ∩ (⋃ *P2*) = {}
is-non-overlapping P1
is-non-overlapping P2
X ∈ *P1* ∪ *P2*
Y ∈ *P1* ∪ *P2*
(*X* = *Y*)
shows *X* ∩ *Y* ≠ {}
unfolding *is-non-overlapping-def* **using** *assms is-non-overlapping-def* **by** *fast*

lemma *lm027*:
assumes (⋃ *P1*) ∩ (⋃ *P2*) = {}
is-non-overlapping P1
is-non-overlapping P2
shows *is-non-overlapping* (*P1* ∪ *P2*)
unfolding *is-non-overlapping-def* **using** *assms lm025 lm026* **by** *metis*

lemma *lm028*:
Range Q ∪ (*Range* (*P outside* (*Domain Q*))) = *Range* (*P* +* *Q*)
by (*simp add*: *paste-def Range-Un-eq Un-commute*)

lemma *lm029*:
assumes *a1* ∈ *injectionsUniverse*
a2 ∈ *injectionsUniverse*
(*Range a1*) ∩ (*Range a2*)={}
(*Domain a1*) ∩ (*Domain a2*) = {}
shows *a1* ∪ *a2* ∈ *injectionsUniverse*
using *assms disj-Un-runiq*
by (*metis* (*no-types*) *Domain-converse converse-Un mem-Collect-eq*)

lemma *nonOverlapping*:
assumes *R* ∈ *partitionValuedUniverse*

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

lemma *allocationUnion*:
assumes *a1* ∈ *allocationsUniverse*
a2 ∈ *allocationsUniverse*
(⋃ (*Range a1*)) ∩ (⋃ (*Range a2*)) = {}
(*Domain a1*) ∩ (*Domain a2*) = {}
shows *a1* ∪ *a2* ∈ *allocationsUniverse*
proof −
let *?a*=*a1* ∪ *a2*
let *?b1*=*a1*⌢−*1*
let *?b2*=*a2*⌢−*1*
let *?r*=*Range*
let *?d*=*Domain*
let *?I*=*injectionsUniverse*
let *?P*=*partitionsUniverse*
let *?PV*=*partitionValuedUniverse*
let *?u*=*runiq*
let *?b*=*?a*⌢−*1*
let *?p*=*is-non-overlapping*
have *?p* (*?r a1*) & *?p* (*?r a2*) **using** *assms nonOverlapping* **by** *blast* **then**
moreover have *?p* (*?r a1* ∪ *?r a2*) **using** *assms* **by** (*metis lm027*)
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** *lm027 assms* **by** *fastforce*
then moreover have *?a* ∈ *?PV* **using** *assms* **by** *fast*
moreover have *?r a1* ∩ (*?r a2*) ⊆ *Pow* (⋃ (*?r a1*) ∩ (⋃ (*?r a2*))) **by** *auto*
ultimately moreover have {} ∉ (*?r a1*) & {} ∉ (*?r a2*)
using *is-non-overlapping-def* **by** (*metis Int-empty-left*)
ultimately moreover have *?r a1* ∩ (*?r a2*) = {}
using *assms nonOverlapping is-non-overlapping-def* **by** *auto*
ultimately moreover have *?a* ∈ *?I* **using** *lm029 assms* **by** *fastforce*
ultimately show *?thesis* **by** *blast*
qed

lemma *lm030*:
assumes *a* ∈ *injectionsUniverse*
shows *a* − *b* ∈ *injectionsUniverse*
using *assms*
by (*metis* (*lifting*) *Diff-subset converse-mono mem-Collect-eq subrel-runiq*)

lemma *lm031*:
$\{a.\ Domain\ a \subseteq N\quad \&\quad Range\ a \in all\text{-}partitions\ G\} =$
$(Domain -`(Pow\ N))\ \cap\ (Range -`\ (all\text{-}partitions\ G))$
by *fastforce*

lemma *lm032*:
allAllocations N G $=$
injectionsUniverse $\cap$ ((*Range* $-$‘ (*all-partitions G*)) $\cap$ (*Domain* $-$‘(*Pow N*)))
using *allocationInjectionsUnivervseProperty lm031* **by** (*metis* (*no-types*) *Int-commute*)

corollary *lm033*:
allAllocations N G $=$
injectionsUniverse $\cap$ (*Range* $-$‘ (*all-partitions G*)) $\cap$ (*Domain* $-$‘(*Pow N*))
using *lm032 Int-assoc* **by** (*metis*)

lemma *lm034*:
assumes $a \in$ *allAllocations N G*
shows ($a\widehat{\ }{-1} \in$ *injections* (*Range a*) *N* &
Range a $\in$ *all-partitions G*)
using *assms*
by (*metis* (*mono-tags*, *opaque-lifting*) *posssibleAllocationsRelCharacterization*
allocationInverseRangeDomainProperty)

lemma *lm035*:
assumes $a\widehat{\ }{-1} \in$ *injections* (*Range a*) *N Range a* $\in$ *all-partitions G*
shows $a \in$ *allAllocations N G*
using *assms image-iff* **by** *fastforce*

lemma *allocationReverseInjective*:
$a \in$ *allAllocations N G* $=$
($a\widehat{\ }{-1} \in$ *injections* (*Range a*) *N* & *Range a* $\in$ *all-partitions G*)
using *lm034 lm035* **by** *metis*

lemma *lm036*:
assumes $a \in$ *allAllocations 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 posssibleAllocationsRelCharacterization order-refl*
by (*metis* (*mono-tags*, *lifting*))

lemma *lm037*:
assumes $a \in$ *injections* (*Domain a*) (*Range a*)
Range a $\in$ *all-partitions G*
Domain a $\subseteq N$
shows $a \in$ *allAllocations N G*
using *assms mem-Collect-eq posssibleAllocationsRelCharacterization injections-def*

by (*metis* (*erased*, *lifting*))

lemma *characterizationallAllocations*:
a ∈ *allAllocations N G* = (*a* ∈ *injections* (*Domain a*) (*Range a*) &
Range a ∈ *all-partitions G* &
Domain a ⊆ *N*)
using *lm036 lm037* **by** *metis*

lemma *lm038*:
assumes *a* ∈ *partitionValuedUniverse*
shows *a* − *b* ∈ *partitionValuedUniverse*
using *assms subset-is-non-overlapping* **by** *fast*

lemma *reducedAllocation*:
assumes *a* ∈ *allocationsUniverse*
shows *a* − *b* ∈ *allocationsUniverse*
using *assms lm030 lm038* **by** *auto*

lemma *lm039*:
assumes *a* ∈ *injectionsUniverse*
shows *a* ∈ *injections* (*Domain a*) (*Range a*)
using *assms injections-def mem-Collect-eq order-refl* **by** *blast*

lemma *lm040*:
assumes *a* ∈ *allocationsUniverse*
shows *a* ∈ *allAllocations* (*Domain a*) (⋃ (*Range a*))
proof −
let *?r=Range*
let *?p=is-non-overlapping*
let *?P=all-partitions*
have *?p* (*?r a*) **using** *assms nonOverlapping 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 allocationInjectionsUnivervseProperty mem-Collect-eq rev-subsetD*
by (*metis* (*lifting*, *no-types*))
qed

lemma *lm041*:
({*X*} ∈ *partitionsUniverse*) = (*X* ≠ {})
using *is-non-overlapping-def* **by** *fastforce*

lemma *lm042*:
{(*x*, *X*)} − {(*x*, {})} ∈ *partitionValuedUniverse*
using *lm041* **by** *auto*

lemma *singlePairInInjectionsUniverse*:

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

lemma *allocationUniverseProperty*:
{(x,X)} − {(x,{})} ∈ *allocationsUniverse*
using *lm042 singlePairInInjectionsUniverse lm030 Int-iff* **by** (*metis* (*no-types*))

lemma *lm043*:
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* (*opaque-lifting*, *no-types*))
{
fix X Y
assume
1: X ∈ ?P1 & Y∈?P1 & X ≠ Y
then obtain XX YY
where
2: X = ?U XX & Y=?U YY & XX∈PP & YY∈PP **by** *blast*
then have XX ⊆ *Union PP* & YY ⊆ *Union PP* & XX ∩ YY = {}
using *1 2 is-non-overlapping-def assms*(*1*) *Sup-upper* **by** *metis*
then moreover have ∀ x∈XX. ∀ y∈YY. x ∩ y = {} **using** *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 *lm044*:
assumes a ∈ *allocationsUniverse*
shows (a − ((X∪{i})×(*Range* a))) ∪
({(i, ⋃ (a‘‘(X ∪ {i})))} − {(i,{})}) ∈ *allocationsUniverse* &
⋃ (*Range* ((a − ((X∪{i})×(*Range* a))) ∪ ({(i, ⋃ (a‘‘(X ∪ {i})))} − {(i,{})}))) =
⋃(*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* ∪ {*i*}
let *?b=?Xi* × (*?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* ∪ *?a2*
let *?t1=?c* ∈ *allocationsUniverse*
have
1: *?U*(*?r*(*?a1*∪*?a2*))=*?U*(*?r ?a1*) ∪ (*?U*(*?r ?a2*)) **by** (*metis Range-Un-eq Union-Un-distrib*)

have
2: *?U*(*?r a*) ⊆ *?U*(*?r ?a1*) ∪ *?U*(*a``?Xi*) & *?U*(*?r ?a1*) ∪ *?U*(*?r ?a2*) ⊆ *?U*(*?r a*) **by** *blast*
have
3: *?u a* & *?u* (*a*^−*1*) & *?p* (*?r a*) & *?r ?a1* ⊆ *?r a* & *?Yi* ⊆ *?r a*
using *assms Int-iff nonOverlapping mem-Collect-eq* **by** *auto*
then have
4: *?p* (*?r ?a1*) & *?p ?Yi* **using** *subset-is-non-overlapping* **by** *metis*
have *?a1* ∈ *allocationsUniverse* & *?a2* ∈ *allocationsUniverse*
using *allocationUniverseProperty assms*(*1*) *reducedAllocation* **by** *fastforce*
then have (*?a1* = {} ∨ *?a2* = {})⟶ *?t1*
using *Un-empty-left* **by** (*metis* (*lifting*, *no-types*) *Un-absorb2 empty-subsetI*)
moreover have (*?a1* = {} ∨ *?a2* = {})⟶ *?U* (*?r a*) = *?U* (*?r ?a1*) ∪ *?U* (*?r ?a2*) **by** *fast*
ultimately have
5: (*?a1* = {} ∨ *?a2* = {})⟶ *?thesis* **using** *1* **by** *simp*
{
assume
6: *?a1*≠{} & *?a2*≠{}
then have *?r ?a2*⊇{*?Y*}
using *Diff-cancel Range-insert empty-subsetI insert-Diff-single insert-iff insert-subset*
by (*metis* (*opaque-lifting*, *no-types*))
then have
7: *?U* (*?r a*) = *?U* (*?r ?a1*) ∪ *?U* (*?r ?a2*) **using** *2* **by** *blast*
have *?r ?a1* ≠ {} & *?r ?a2* ≠ {} **using** *6* **by** *auto*
moreover have *?r ?a1* ⊆ *a``*(*?d ?a1*) **using** *assms* **by** *blast*
moreover have *?Yi* ∩ (*a``*(*?d a* − *?Xi*)) = {}
using *assms 3 6 Diff-disjoint intersectionEmptyRelationIntersectionEmpty*

by *metis*
ultimately moreover have *?r ?a1* ∩ *?Yi* = {} & *?Yi* ≠ {} **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*} ⊆ *?r a* **by** *auto*
then moreover have *?p* (*?U* {*?r ?a1*, *?Yi*}) **by** (*metis 3 Outside-def subset-is-non-overlapping*)
ultimately moreover have *?p* (*?U*'{(*?r ?a1*), *?Yi*}) **using** *lm043* **by** *fast*
moreover have ... = {*?U* (*?r ?a1*), *?Y*} **by** *force*
ultimately moreover have ∀ *x* ∈ *?r ?a1*. ∀ *y*∈*?Yi*. *x* ≠ *y*
using *IntI empty-iff* **by** *metis*
ultimately moreover have ∀ *x* ∈ *?r ?a1*. ∀ *y*∈*?Yi*. *x* ∩ *y* = {}
using *3 4 6 is-non-overlapping-def* **by** (*metis rev-subsetD*)
ultimately have *?U* (*?r ?a1*) ∩ *?Y* = {} **using** *unionIntersectionEmpty*
proof −
have ∀ *v0*. *v0* ∈ *Range* (*a* − (*X* ∪ {*i*}) × *Range a*) ⟶ (∀ *v1*. *v1* ∈ *a* ‘‘ (*X* ∪ {*i*}) ⟶ *v0* ∩ *v1* = {})
by (*metis* (*no-types*) ‹∀ *x*∈*Range* (*a* − (*X* ∪ {*i*}) × *Range a*). ∀ *y*∈*a* ‘‘ (*X* ∪ {*i*}). *x* ∩ *y* = {}›)
thus ⋃(*Range* (*a* − (*X* ∪ {*i*}) × *Range a*)) ∩ ⋃(*a* ‘‘ (*X* ∪ {*i*})) = {} **by** *blast*
qed
then have
?U (*?r ?a1*) ∩ (*?U* (*?r ?a2*)) = {} **by** *blast*
moreover have *?d ?a1* ∩ (*?d ?a2*) = {} **by** *blast*
moreover have *?a1* ∈ *allocationsUniverse* **using** *assms*(*1*) *reducedAllocation* **by** *blast*
moreover have *?a2* ∈ *allocationsUniverse* **using** *allocationUniverseProperty* **by** *fastforce*
ultimately have *?a1* ∈ *allocationsUniverse* & *?a2* ∈ *allocationsUniverse* &
⋃(*Range ?a1*) ∩ ⋃(*Range ?a2*) = {} & *Domain ?a1* ∩ *Domain ?a2* = {}
by *blast*
then have *?t1* **using** *allocationUnion* **by** *auto*
then have *?thesis* **using** *1 7* **by** *simp*
}
then show *?thesis* **using** *5* **by** *linarith*
qed

corollary *allocationsUniverseOutsideUnion*:
assumes *a* ∈ *allocationsUniverse*
shows (*a outside* (*X*∪{*i*})) ∪ ({*i*}×({⋃(*a*‘‘(*X*∪{*i*}))}−{{}})) ∈ *allocationsUniverse* &
⋃(*Range*((*a outside* (*X*∪{*i*})) ∪ ({*i*}×({⋃(*a*‘‘(*X*∪{*i*}))}−{{}})))) = ⋃(*Range a*)
proof −
have *a* − ((*X*∪{*i*})×(*Range a*)) = *a outside* (*X* ∪ {*i*}) **using** *Outside-def* **by** *metis*
moreover have (*a* − ((*X*∪{*i*})×(*Range a*))) ∪ ({(*i*, ⋃ (*a*‘‘(*X* ∪ {*i*})))} −

$\{(i,\{\})\}) \in$
allocationsUniverse
using *assms lm044* **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 lm044* **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 *simp*
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 *lm045*:
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 nonOverlapping* **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 *lm046*:
assumes *Domain a* $\cap X \neq \{\}$ $a \in$ *allocationsUniverse*
shows $\{\bigcup (a``(X \cup \{i\}))\} - \{\{\}\} = \{\bigcup (a``(X \cup \{i\}))\}$
using *assms lm045* **by** *fast*

corollary *lm047*:
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* $(X \cup \{i\})) \cup (\{i\} \times (\{\bigcup (a``(X \cup \{i\}))\} - \{\{\}\})) \in$ *allocation-*

sUniverse
let *?t2* = $\bigcup$(*Range*((*a outside* (*X*∪{*i*})) ∪ ({*i*}×({$\bigcup$(*a*‘‘(*X*∪{*i*}))}−{{}})))) = $\bigcup$(*Range a*)
have
0: *a* ∈ *allocationsUniverse* **using** *assms*(*1*) **by** *fast*
then have *?t1* & *?t2* **using** *allocationsUniverseOutsideUnion*
proof −
have *a* ∈ *allocationsUniverse* ⟶
a outside (*X* ∪ {*i*}) ∪ {*i*} × ({$\bigcup$(*a* ‘‘ (*X* ∪ {*i*}))} − {{}}) ∈ *allocationsUniverse*
using *allocationsUniverseOutsideUnion* **by** *fastforce*
hence *a outside* (*X* ∪ {*i*}) ∪ {*i*} × ({$\bigcup$(*a* ‘‘ (*X* ∪ {*i*}))} − {{}}) ∈ *allocationsUniverse*
by (*metis 0*)
thus *a outside* (*X* ∪ {*i*}) ∪ {*i*} × ({$\bigcup$(*a* ‘‘ (*X* ∪ {*i*}))} − {{}}) ∈
allocationsUniverse ∧ $\bigcup$(*Range* (*a outside* (*X* ∪ {*i*}) ∪ {*i*} × ({$\bigcup$(*a* ‘‘ (*X* ∪ {*i*}))} − {{}})))
= $\bigcup$(*Range a*)
using *0* **by** (*metis* (*no-types*) *allocationsUniverseOutsideUnion*)
qed
moreover have
{$\bigcup$(*a*‘‘(*X*∪{*i*}))}−{{}} = {$\bigcup$(*a*‘‘(*X*∪{*i*}))} **using** *lm045 assms* **by** *fast*
ultimately show *?thesis* **by** *auto*
qed

abbreviation
bidMonotonicity b i ==
(∀ *t t*′. (*trivial t* & *trivial t*′ & *Union t* ⊆ *Union t*′) ⟶
sum b ({*i*}×*t*) ≤ *sum b* ({*i*}×*t*′))

lemma *lm048*:
assumes *bidMonotonicity b i runiq a*
shows *sum b* ({*i*}×((*a outside X*)‘‘{*i*})) ≤ *sum b* ({*i*}×{$\bigcup$(*a*‘‘(*X*∪{*i*}))})
proof −
let *?u* = *runiq*
let *?I* = {*i*}
let *?R* = *a outside X*
let *?U* = *Union*
let *?Xi* = *X* ∪*?I*
let *?t1* = *?R*‘‘*?I*
let *?t2* = {*?U* (*a*‘‘*?Xi*)}
have *?U* (*?R*‘‘*?I*) ⊆ *?U* (*?R*‘‘(*X*∪*?I*)) **by** *blast*
moreover have ... ⊆ *?U* (*a*‘‘(*X*∪*?I*)) **using** *Outside-def* **by** *blast*
ultimately have *?U* (*?R*‘‘*?I*) ⊆ *?U* (*a*‘‘(*X*∪*?I*)) **by** *auto*
then have
0: *?U ?t1* ⊆ *?U ?t2* **by** *auto*
have *?u a* **using** *assms* **by** *fast*

moreover have *?R* ⊆ *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 *lm049*:
assumes $XX \in partitionValuedUniverse$
shows $\{\} \notin Range\ XX$
using *assms mem-Collect-eq no-empty-in-non-overlapping* **by** *auto*

corollary *emptyNotInRange*:
assumes $a \in allAllocations\ N\ G$
shows $\{\} \notin Range\ a$
using *assms lm049 allAllocationsUniverse* **by** *auto blast*

lemma *lm050*:
assumes $a \in allAllocations\ N\ G$
shows $Range\ a \subseteq Pow\ G$
using *assms allocationInverseRangeDomainProperty is-partition-of-def* **by** (*metis subset-Pow-Union*)

corollary *lm051*:
assumes $a \in allAllocations\ N\ G$
shows $Domain\ a \subseteq N\ \&\ Range\ a \subseteq Pow\ G - \{\{\}\}$
using *assms lm050 insert-Diff-single emptyNotInRange subset-insert allocationInverseRangeDomainProperty* **by** *metis*

corollary *allocationPowerset*:
assumes $a \in allAllocations\ N\ G$
shows $a \subseteq N \times (Pow\ G - \{\{\}\})$
using *assms lm051* **by** *blast*

corollary *lm052*:
$allAllocations\ N\ G \subseteq Pow\ (N \times (Pow\ G - \{\{\}\}))$
using *allocationPowerset* **by** *blast*

lemma *lm053*:
assumes $a \in allAllocations\ N\ G$
$i \in N - X$
$Domain\ a \cap X \neq \{\}$
shows $a\ outside\ (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a``(X \cup \{i\}))\}) \in allAllocations\ (N - X)\ (\bigcup\ (Range\ a))$
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*) *allAllocationsUniverse rev-subsetD* **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 lm047*) **done**
then have $?aa \in$ *allAllocations* $(?d\ ?aa)\ (?U\ (?r\ a))$
using *lm040* **by** (*metis* (*lifting*, *mono-tags*))
then have $?aa \in$ *allAllocations* $(?d\ ?aa \cup (?d\ a - X \cup \{i\}))\ (?U\ (?r\ a))$
by (*metis lm023*)
moreover have $?d\ a - X \cup \{i\} = ?d\ ?aa \cup (?d\ a - X \cup \{i\})$ **using** *Outside-def* **by** *auto*
ultimately have $?aa \in$ *allAllocations* $(?d\ a - X \cup \{i\})\ (?U\ (?r\ a))$ **by** *simp*
then have $?aa \in$ *allAllocations* $(?d\ a \cup \{i\} - X)\ (?U\ (?r\ a))$ **using** *2* **by** *simp*
moreover have $?d\ a \subseteq N$ **using** *assms*(*1*) *posssibleAllocationsRelCharacterization* **by** *metis*
then moreover have $(?d\ a \cup \{i\} - X) \cup (N - X) = N - X$ **using** *assms* **by** *fast*
ultimately have $?aa \in$ *allAllocations* $(N - X)\ (?U\ (?r\ a))$ **using** *lm024*
by (*metis* (*lifting*, *no-types*) *in-mono*)
then show *?thesis* **by** *fast*
qed

lemma *lm054*:
assumes *bidMonotonicity* (*b*::- => *real*) *i*
$a \in$ *allocationsUniverse*
Domain $a \cap X \neq \{\}$
finite a
shows *sum b* (*a outside X*) $\leq$
sum 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 ⊆ a & ?R−−i ⊆ a using Outside-def using lm088 by auto
  ultimately have finite (?R −− i) & ?u (?R−−i) & ?u ?R
    using finite-subset subrel-runiq by (metis assms(4))
  then moreover have trivial ({i}×(?R``{i})) using runiq-def
    by (metis trivial-cartesian trivial-singleton)
  moreover have ∀ X. (?R −− i) ∩ ({i}×X)={} using outside-reduces-domain
by force
  ultimately have
  1: finite (?R−−i) & finite ({i}×(?R``{i})) & (?R −− i) ∩ ({i}×(?R``{i}))={}
&
     finite ({i} × {?U(a``(X∪{i}))}) & (?R −− i) ∩ ({i} × {?U(a``(X∪{i}))})={}

    using Outside-def trivial-implies-finite by fast
  have ?R = (?R −− i) ∪ ({i}×?R``{i}) by (metis outsideUnion)
  then have sum b ?R = sum b (?R −− i) + sum b ({i}×(?R``{i}))
    using 1 sum.union-disjoint by (metis (lifting) sum.union-disjoint)
  moreover have sum b ({i}×(?R``{i})) ≤ sum b ({i}×{?U(a``(X∪{i}))})
    using lm048 assms(1) 0 by metis
 ultimately have sum b ?R ≤ sum b (?R −− i) + sum b ({i}×{?U(a``(X∪{i}))})
by linarith
  moreover have ... = sum b (?R −− i ∪ ({i} × {?U(a``(X∪{i}))}))
  using 1 sum.union-disjoint by auto
  moreover have ... = sum b ?aa by (metis outsideOutside)
  ultimately show ?thesis by simp
qed

lemma elementOfPartitionOfFiniteSetIsFinite:
  assumes finite X XX ∈ all-partitions X
  shows finite XX
  using all-partitions-def is-partition-of-def
  by (metis assms(1) assms(2) finite-UnionD mem-Collect-eq)

lemma lm055:
  assumes finite N finite G a ∈ allAllocations N G
  shows finite a
  using assms finiteRelationCharacterization rev-finite-subset
  by (metis characterizationallAllocations elementOfPartitionOfFiniteSetIsFinite)

lemma allAllocationsFinite:
  assumes finite N finite G
  shows finite (allAllocations N G)
proof −
  have finite (Pow(N×(Pow G−{{}}))) using assms finite-Pow-iff by blast
  then show ?thesis using lm052 rev-finite-subset by (metis(no-types))
qed

corollary lm056:
  assumes bidMonotonicity (b::- => real) i
          a ∈ allAllocations N G
```

i ∈ *N*−*X*
Domain a ∩ *X* ≠ {}
finite N
finite G
shows *Max* ((*sum b*)‘(*allAllocations* (*N*−*X*) *G*)) ≥
sum 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*) *allAllocationsUniverse* **by** *blast*
moreover have *Domain a* ∩ *X* ≠ {} **using** *assms*(*4*) **by** *auto*
moreover have *finite a* **using** *assms lm055* **by** *metis*
ultimately have
0: *sum b* (*a outside X*) ≤ *sum b ?aa* **by** (*rule lm054*)
have *?aa* ∈ *allAllocations* (*N*−*X*) (⋃ (*Range a*)) **using** *assms lm053* **by** *metis*
moreover have ⋃ (*Range a*) = *G*
using *assms allocationInverseRangeDomainProperty is-partition-of-def* **by** *metis*
ultimately have *sum b ?aa* ∈ (*sum b*)‘(*allAllocations* (*N*−*X*) *G*) **by** (*metis imageI*)
moreover have *finite* ((*sum b*)‘(*allAllocations* (*N*−*X*) *G*))
using *assms allAllocationsFinite assms*(*5*,*6*) **by** (*metis finite-Diff finite-imageI*)
ultimately have *sum b ?aa* ≤ *Max* ((*sum b*)‘(*allAllocations* (*N*−*X*) *G*)) **by** *auto*
then show *?thesis* **using** *0* **by** *linarith*
qed

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

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

lemma *sumUnionDisjoint1*:
assumes ∀ *A*∈*C*. *finite A* ∀ *A*∈*C*. ∀ *B*∈*C*. *A* ≠ *B* ⟶ *A Int B* = {}
shows *sum f* (*Union C*) = *sum* (*sum f*) *C*
using *assms sum.Union-disjoint* **by** *fastforce*

corollary *sumUnionDisjoint2*:
assumes ∀ *x*∈*X*. *finite x is-non-overlapping X*
shows *sum f* (⋃ *X*) = *sum* (*sum f*) *X*
using *assms sumUnionDisjoint1 is-non-overlapping-def* **by** *fast*

corollary *sumUnionDisjoint3*:
assumes $\forall$ *x*$\in$*X*. *finite x X partitions XX*
shows *sum f XX* = *sum* (*sum f*) *X*
using *assms* **by** (*metis is-partition-of-def sumUnionDisjoint2*)

corollary *sum-associativity*:
assumes *finite x X partitions x*
shows *sum f x* = *sum* (*sum f*) *X*
using *assms sumUnionDisjoint3*
by (*metis is-partition-of-def familyUnionFiniteEverySetFinite*)

lemma *lm057*:
assumes *a* $\in$ *allocationsUniverse Domain a* $\subseteq$ *N* $\bigcup$(*Range a*) = *G*
shows *a* $\in$ *allAllocations N G*
using *assms posssibleAllocationsRelCharacterization lm040* **by** (*metis* (*mono-tags*, *lifting*))

corollary *lm058*:
(*allocationsUniverse* $\cap$ {*a*. (*Domain a*) $\subseteq$ *N* & $\bigcup$(*Range a*) = *G*}) $\subseteq$
allAllocations N G
using *lm057* **by** *fastforce*

corollary *lm059*:
allAllocations N G $\subseteq$ {*a*. (*Domain a*) $\subseteq$ *N*}
using *allocationInverseRangeDomainProperty* **by** *blast*

corollary *lm060*:
allAllocations N G $\subseteq$ {*a*. $\bigcup$(*Range a*) = *G*}
using *is-partition-of-def allocationInverseRangeDomainProperty mem-Collect-eq subsetI*
by (*metis*(*mono-tags*))

corollary *lm061*:
allAllocations N G $\subseteq$ *allocationsUniverse* &
allAllocations N G $\subseteq$ {*a*. (*Domain a*) $\subseteq$ *N* & $\bigcup$(*Range a*) = *G*}
using *lm059 lm060 conj-subset-def allAllocationsUniverse* **by** (*metis* (*no-types*))

corollary *allAllocationsIntersectionSubset*:
allAllocations 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 lm061*) **thus** *?thesis* **by** *auto*
qed

corollary *allAllocationsIntersection*:
allAllocations N G =
(*allocationsUniverse* $\cap$ {*a*. (*Domain a*) $\subseteq$ *N* & $\bigcup$(*Range a*) = *G*})
(**is** *?L* = *?R*)

proof −
  have *?L* ⊆ *?R* **using** *allAllocationsIntersectionSubset* **by** *metis*
  moreover have *?R* ⊆ *?L* **using** *lm058* **by** *fast*
  ultimately show *?thesis* **by** *force*
qed

corollary *allAllocationsIntersectionSetEquals*:
  a ∈ *allAllocations N G* =
   (*a* ∈ *allocationsUniverse* & (*Domain a*) ⊆ *N* & ⋃(*Range a*) = *G*)
  using *allAllocationsIntersection Int-Collect* **by** (*metis* (*mono-tags*, *lifting*))

corollary *allocationsUniverseOutside*:
  assumes *a* ∈ *allocationsUniverse*
  shows *a outside X* ∈ *allocationsUniverse*
  using *assms Outside-def* **by** (*metis* (*lifting*, *mono-tags*) *reducedAllocation*)

9.4 Bridging theorem for injections

lemma *lm062*:
  totalRels {} *Y* = {{}}
  by *fast*

lemma *lm063*:
  {} ∈ *injectionsUniverse*
  by (*metis CollectI converse-empty runiq-emptyrel*)

lemma *lm064*:
  injectionsUniverse ∩ (*totalRels* {} *Y*) = {{}}
  using *lm062 lm063* **by** *fast*

lemma *lm065*:
  assumes *runiq f x*∉*Domain f*
  shows { *f* ∪ {(*x*, *y*)} | *y* . *y* ∈ *A* } ⊆ *runiqs*
  unfolding *paste-def runiqs-def* **using** *assms runiq-basic* **by** *blast*

lemma *lm066*:
  converse ‘ (*converse* ‘ *X*) = *X*
  by *auto*

lemma *lm067*:
  runiq (*f*^−*1*) = (*f* ∈ *converse*‘*runiqs*)
  unfolding *runiqs-def* **by** *auto*

lemma *lm068*:
  assumes *runiq* (*f*^−*1*) *A* ∩ *Range f* = {}
  shows *converse* ‘ { *f* ∪ {(*x*, *y*)} | *y* . *y* ∈ *A* } ⊆ *runiqs*
  using *assms lm065* **by** *fast*

lemma *lm069*:

assumes *f*∈*converse*‘*runiqs A* ∩ *Range f* = {}
shows {*f* ∪ {(*x*, *y*)}| *y*. *y* ∈ *A*} ⊆ *converse*‘*runiqs*
(**is** *?l* ⊆ *?r*)
proof −
have *runiq* (*f*^−*1*) **using** *assms*(*1*) *lm067* **by** *blast*
then have *converse* ‘ *?l* ⊆ *runiqs* **using** *assms*(*2*) **by** (*rule lm068*)
then have *?r* ⊇ *converse*‘(*converse*‘*?l*) **by** *auto*
moreover have *converse*‘(*converse*‘*?l*)=*?l* **by** (*rule lm066*)
ultimately show *?thesis* **by** *simp*
qed

lemma *lm070*:
{ *R* ∪ {(*x*, *y*)} | *y* . *y* ∈ *A* } ⊆ *totalRels* ({*x*} ∪ *Domain R*) (*A* ∪ *Range R*)
by *force*

lemma *lm071*:
injectionsUniverse = *runiqs* ∩ *converse*‘*runiqs*
unfolding *runiqs-def* **by** *auto*

lemma *lm072*:
assumes *f* ∈ *injectionsUniverse x* ∉ *Domain f A* ∩ (*Range f*) = {}
shows {*f* ∪ {(*x*, *y*)}| *y*. *y* ∈ *A*} ⊆ *injectionsUniverse*
(**is** *?l* ⊆ *?r*)
proof −
have *f* ∈ *converse*‘*runiqs* **using** *assms*(*1*) *lm071* **by** *blast*
then have *?l* ⊆ *converse*‘*runiqs* **using** *assms*(*3*) **by** (*rule lm069*)
moreover have *?l* ⊆ *runiqs* **using** *assms*(*1*,*2*) *lm065* **by** *force*
ultimately show *?thesis* **using** *lm071* **by** *blast*
qed

lemma *lm073*:
injections X Y = *totalRels X Y* ∩ *injectionsUniverse*
using *lm008* **by** *metis*

lemma *lm074*:
assumes *f* ∈ *injectionsUniverse*
shows *f outside A* ∈ *injectionsUniverse*
using *assms* **by** (*metis* (*no-types*) *Outside-def lm030*)

lemma *lm075*:
assumes *R* ∈ *totalRels A B*
shows *R outside C* ∈ *totalRels* (*A*−*C*) *B*
unfolding *Outside-def* **using** *assms* **by** *blast*

lemma *lm076*:
assumes *g* ∈ *injections A B*
shows *g outside C* ∈ *injections* (*A* − *C*) *B*
using *assms Outside-def Range-outside-sub lm030 mem-Collect-eq outside-reduces-domain*
unfolding *injections-def*

by *fastforce*

lemma *lm077*:
assumes $g \in$ *injections A B*
shows *g outside* $C \in$ *injections* $(A - C)$ *B*
using *assms lm076* **by** *metis*

lemma *lm078*:
$\{x\} \times \{y\} = \{(x,y)\}$
by *simp*

lemma *lm079*:
assumes $x \in$ *Domain f runiq f*
shows $\{x\} \times f``\{x\} = \{(x,f,,x)\}$
using *assms lm078 Image-runiq-eq-eval* **by** *metis*

corollary *lm080*:
assumes $x \in$ *Domain f runiq f*
shows $f = (f -- x) \cup \{(x,f,,x)\}$
using *assms lm079 outsideUnion* **by** *metis*

lemma *lm081*:
assumes $f \in$ *injectionsUniverse*
shows *Range*(*f outside A*) = *Range f* $-$ $f``A$
using *assms mem-Collect-eq rangeOutside* **by** (*metis*)

lemma *lm082*:
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 $g \in$*injectionsUniverse* **using** *assms*(*1*) *lm008* **by** *fast*
then moreover have $g,,x \in g``\{x\}$
using *assms*(*2*) **by** (*metis Image-runiq-eq-eval insertI1 mem-Collect-eq*)
ultimately have $g,,x \in Y - Range\ ?f$ **using** *lm081 assms*(*1*) **unfolding** *injections-def* **by** *fast*
moreover have $g = ?f \cup \{(x,\ g,,x)\}$
using *assms lm080 mem-Collect-eq* **unfolding** *injections-def* **by** (*metis* (*lifting*))

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

corollary *lm083*:
assumes $x \notin X$ $g \in$ *injections* $(\{x\} \cup X)$ *Y*
shows $g--x \in$ *injections X Y*
using *assms lm077* **by** (*metis Diff-insert-absorb insert-is-Un*)

corollary *lm084*:
assumes $x \notin X$ $g \in$ *injections* $(\{x\} \cup X)$ *Y*

(**is** *g* ∈ *injections* (*?X*) *Y*)
shows ∃ *f* ∈ *injections X Y*. *g* ∈ {*f* ∪ {(*x*,*y*)}|*y*. *y* ∈ *Y* − (*Range f*)}
proof −
let *?f* = *g*−−*x*
have
0: *g*∈*injections ?X Y* **using** *assms* **by** *metis*
have *Domain g*=*?X*
using *assms*(*2*) *mem-Collect-eq* **unfolding** *injections-def* **by** (*metis* (*mono-tags*, *lifting*))
then have
1: *x* ∈ *Domain g* **by** *simp* **then have** *?f* ∈ *injections X Y* **using** *assms lm083* **by** *fast*
moreover have *g*∈{*?f*∪{(*x*,*y*)}|*y*. *y*∈*Y*−*Range ?f*} **using** *0 1* **by** (*rule lm082*)
ultimately show *?thesis* **by** *blast*
qed

corollary *lm085*:
assumes *x* ∉ *X*
shows *injections* ({*x*} ∪ *X*) *Y* ⊆
(⋃ *f* ∈ *injections X Y*. {*f* ∪ {(*x*, *y*)} | *y* . *y* ∈ *Y* − (*Range f*)})
using *assms lm084* **by** *auto*

lemma *lm086*:
assumes *x* ∉ *X*
shows (⋃ *f*∈*injections X Y*. {*f* ∪ {(*x*, *y*)} | *y* . *y* ∈ *Y*−*Range f*}) ⊆
injections ({*x*} ∪ *X*) *Y*
using *assms lm072 injections-def lm073 lm070*
proof −
{ fix *f*
assume *f* ∈ *injections X Y*
then have
0: *f* ∈ *injectionsUniverse* & *x* ∉ *Domain f* & *Domain f* = *X* & *Range f* ⊆ *Y*
using *assms* **unfolding** *injections-def* **by** *fast*
then have *f* ∈ *injectionsUniverse* **by** *fast*
moreover have *x* ∉ *Domain f* **using** *0* **by** *fast*
moreover have
1: (*Y*−*Range f*) ∩ *Range f* = {} **by** *blast*
ultimately have {*f* ∪ {(*x*, *y*)} | *y* . *y* ∈ (*Y*−*Range f*)} ⊆ *injectionsUniverse* **by** (*rule lm072*)
moreover have {*f* ∪ {(*x*, *y*)} | *y* . *y* ∈ (*Y*−*Range f*)} ⊆ *totalRels* ({*x*} ∪ *X*) *Y*
using *lm070 0* **by** *force*
ultimately have {*f* ∪ {(*x*, *y*)} | *y* . *y* ∈ (*Y*−*Range f*)} ⊆
injectionsUniverse ∩ *totalRels* ({*x*}∪*X*) *Y*
by *auto*
}
thus *?thesis* **using** *lm008* **unfolding** *injections-def* **by** *blast*
qed

corollary *injectionsUnionCommute*:
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′*) **by** *blast*
have $?r' \subseteq injections\ ?X\ Y$ **using** *assms* **by** (*rule lm086*) **moreover have** ...
$= injections\ ?X\ Y$
unfolding *lm005*
by *simp* **ultimately have** $?r \subseteq injections\ ?X\ Y$ **using** *0* **by** *simp*
moreover have $injections\ ?X\ Y \subseteq ?r$ **using** *assms* **by** (*rule lm085*)
ultimately show *?thesis* **by** *blast*
qed

lemma *lm087*:
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* **by** *blast*

lemma *lm088*:
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 *lm089*:
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 $\forall\ f.\ f{\in}injections\ X\ Y\ \&\ x \notin Domain\ f \longrightarrow$
$\{f +{*}\ \{(x,\ y)\} \mid y\ .\ y \in Y{-}Range\ f\}{=}\{f \cup \{(x,\ y)\} \mid y\ .\ y \in Y{-}Range\ f\}$

using *lm088* **by** *force*
then have *?l'=?r'* **by** (*rule lm087*)
then show *?l = ?r* **using** *1 2* **by** *presburger*
qed

corollary *lm090*:
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 lm089*)
moreover have ... = *?r* **using** *assms* **by** (*rule injectionsUnionCommute*)
ultimately show *?thesis* **by** *simp*
qed

lemma *lm091*:
set [$f \cup \{(x,y)\}$. $y \leftarrow$ (*filter* (%y. $y \notin$ (*Range f*)) Y)] =
$\{f \cup \{(x,y)\}$ | y . $y \in$ (*set Y*) − (*Range f*)$\}$
by *auto*

lemma *lm092*:
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 *lm093*:
set (*concat* [[$f \cup \{(x,y)\}$. $y \leftarrow$ (*filter* (%y. $y \notin$ *Range f*) Y)]. $f \leftarrow F$]) =
($\bigcup$ $f \in$ *set F*. $\{f \cup \{(x,y)\}$ | y . $y \in$ (*set Y*) − (*Range f*)$\}$)
by *auto*

lemma *lm094*:
assumes *finite Y*
shows *set* [f +∗ $\{(x,y)\}$. $y \leftarrow$ *sorted-list-of-set* (Y − (*Range f*))] =
$\{$ f +∗ $\{(x,y)\}$ | y . $y \in$ Y − (*Range f*)$\}$
using *assms* **by** *auto*

lemma *lm095*:
assumes *finite Y*
shows *set* (*concat* [[f +∗ $\{(x,y)\}$. $y \leftarrow$ *sorted-list-of-set*(Y − (*Range f*))]. $f \leftarrow$ F]) =
($\bigcup$ $f \in$ *set F*.$\{f$ +∗ $\{(x,y)\}$ | y . $y \in Y$ − (*Range f*)$\}$)
using *assms lm094 lm092* **by** *auto*

9.5 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 *lm096*:
 set (*injectionsAlg* (*x* # *xs*) *Y*) =
  (⋃ *f* ∈ *set* (*injectionsAlg xs Y*). {*f* ∪ {(*x*,*y*)} | *y* . *y* ∈ (*set Y*) − (*Range f*)})
 using *lm093* **by** *auto*

corollary *lm097*:
 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 lm096* **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 *lm098*:
 injections {} *Y* = {{}}
 by (*simp add*: *lm008 lm062 runiq-emptyrel*)

lemma *lm099*:
 injections {} *Y* = {{}}
 unfolding *injections-def* **by** (*metis lm098 injections-def*)

lemma *injectionsFromEmptyIsEmpty*:
 injectionsAlg [] *Y* = [{}]
 by *simp*

lemma *lm100*:
 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 lm097*)
 moreover have ... = *?r* **using** *assms*(*1*) **by** (*rule injectionsUnionCommute*)
 ultimately show *?thesis* **by** *simp*
qed

```
lemma lm101:
  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) lm095 by auto
  moreover have ... = ?r using assms(1) by (rule lm090)
  ultimately show ?thesis by simp
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) lm099 by metis
  moreover have ∀ x xs. ?P xs ⟶ ?P (x#xs)
   using assms(1) lm101 by (metis distinct.simps(2) insert-is-Un list.simps(15))

  ultimately have ?P X by (rule structInduct)
  then show ?thesis using assms(2) by blast
qed

lemma lm102:
  assumes l ∈ set (all-partitions-list G) distinct G
  shows   distinct l
  using assms by (metis all-partitions-equivalence′)


lemma bridgingInjection:
  assumes card N > 0 distinct G
  shows ∀ l ∈ set (all-partitions-list G). set (injections-alg l N) =
        injections (set l) N
  using lm102 injections-equiv assms by (metis card-ge-0-finite)


lemma lm103:
```

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 bridgingInjection* **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 *lm104*:
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 lm103*) **thus** *?thesis* **by** *presburger*
qed

corollary *allAllocationsBridgingLemma*:
assumes *card N > 0 distinct G*
shows *allAllocations N* (*set G*) =
set(*allAllocationsAlg 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* **by** (*rule lm104*)
then have *converse* ‘ *?LL* = *converse* ‘ *?RR* **by** *simp*
thus *?thesis* **by** *force*
qed

end

10 Termination theorem for uniform tie-breaking

theory *UniformTieBreaking*

imports
StrictCombinatorialAuction
Universes

HOL−Library.Code-Target-Nat

begin

10.1 Uniform tie breaking: definitions

Let us repeat the general context. Each bidder has made their bids and the VCG algorithm up to now allocates goods to the higher bidders. If there are several high bidders tie breaking has to take place. To do tie breaking we generate out of a random number a second bid vector so that the same algorithm can be run again to determine a unique allocation.

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

abbreviation *omega pair* == {*fst pair*} × (*finestpart* (*snd pair*))

definition *pseudoAllocation allocation* == ⋃ (*omega* ‘ *allocation*)

abbreviation *bidMaximizedBy allocation N G* ==
pseudoAllocation allocation <|| ((*N* × (*finestpart G*)))

abbreviation *maxbid a N G* ==
toFunction (*bidMaximizedBy a N G*)

abbreviation *summedBid bids pair* ==
(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*)))

abbreviation *summedBidSecond bids pair* ==
sum (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))

abbreviation *summedBidVectorRel bids N G* == (*summedBid bids*) ‘ (*N* × (*Pow G* − {{}}))

abbreviation *summedBidVector bids N G* == *toFunction* (*summedBidVectorRel bids N G*)

abbreviation *tiebids allocation N G* == *summedBidVector* (*maxbid allocation N G*) *N G*

abbreviation *Tiebids allocation N G* == *summedBidVectorRel* (*real*∘*maxbid allocation N G*) *N G*

definition *randomEl list* (*random*::*integer*) = *list* ! ((*nat-of-integer random*) *mod* (*size list*))

value *nat-of-integer* (−*3*::*integer*) *mod 2*

lemma *randomElLemma*:
  assumes *set list* ≠ {}
  shows *randomEl list random* ∈ *set list*
  using *assms* **by** (*simp add*: *randomEl-def*)

abbreviation *chosenAllocation N G bids random* ==
  randomEl (*takeAll* (%*x*. *x*∈(*winningAllocationsRel N* (*set G*) *bids*))
        (*allAllocationsAlg N G*))
    random

abbreviation *resolvingBid N G bids random* ==
  tiebids (*chosenAllocation N G bids random*) *N* (*set G*)

10.2 Termination theorem for the uniform tie-breaking scheme

corollary *winningAllocationPossible*:
  winningAllocationsRel N G b ⊆ *allAllocations N G*
  using *injectionsFromEmptyAreEmpty mem-Collect-eq subsetI* **by** *auto*

lemma *subsetAllocation*:
  assumes *a* ∈ *allocationsUniverse c* ⊆ *a*
  shows *c* ∈ *allocationsUniverse*
proof −
  have *c*=*a*−(*a*−*c*) **using** *assms*(*2*) **by** *blast*
  thus *?thesis* **using** *assms*(*1*) *reducedAllocation* **by** (*metis* (*no-types*))
qed

lemma *lm001*:
  assumes *a* ∈ *allocationsUniverse*
  shows *a outside X* ∈ *allocationsUniverse*
  using *assms reducedAllocation Outside-def* **by** (*metis* (*no-types*))

corollary *lm002*:
  $\{x\}\times(\{X\}-\{\{\}\}) \in$ *allocationsUniverse*
  using *allocationUniverseProperty pairDifference* **by** *metis*

corollary *lm003*:
  $\{(x,\{y\})\} \in$ *allocationsUniverse*
proof −
  have $\bigwedge x1.\ \{\} - \{x1::'a \times 'b\ set\} = \{\}$ **by** *simp*
  thus $\{(x, \{y\})\} \in$ *allocationsUniverse*
  by (*metis* (*no-types*) *allocationUniverseProperty empty-iff insert-Diff-if insert-iff prod.inject*)
qed

corollary *lm004*:
  allocationsUniverse $\neq \{\}$
  using *lm003* **by** *fast*

corollary *lm005*:
  $\{\} \in$ *allocationsUniverse*
  using *subsetAllocation lm003* **by** (*metis* (*lifting*, *mono-tags*) *empty-subsetI*)

lemma *lm006*:
  assumes $G \neq \{\}$
  shows $\{G\} \in$ *all-partitions* G
  using *all-partitions-def is-partition-of-def is-non-overlapping-def assms* **by** *force*

lemma *lm007*:
  assumes $n \in N$
  shows $\{(G,n)\} \in$ *totalRels* $\{G\}$ N
  using *assms* **by** *force*

lemma *lm008*:
  assumes $n\in N$
  shows $\{(G,n)\} \in$ *injections* $\{G\}$ N
  using *assms injections-def singlePairInInjectionsUniverse* **by** *fastforce*

corollary *lm009*:
  assumes $G\neq\{\}$ $n\in N$
  shows $\{(G,n)\} \in$ *possible-allocations-rel* G N
proof −
  have $\{(G,n)\} \in$ *injections* $\{G\}$ N **using** *assms lm008* **by** *fast*
  moreover have $\{G\} \in$ *all-partitions* G **using** *assms lm006* **by** *metis*
  ultimately show *?thesis* **by** *auto*
qed

corollary *lm010*:
  assumes $N \neq \{\}$ $G\neq\{\}$
  shows *allAllocations* N $G \neq \{\}$

using *assms lm009*
by (*metis* (*opaque-lifting*, *no-types*) *equals0I image-insert insert-absorb insert-not-empty*)

corollary *lm011*:
assumes $N \neq \{\}$ *finite N* $G \neq \{\}$ *finite G*
shows *winningAllocationsRel N G bids* $\neq \{\}$ & *finite* (*winningAllocationsRel N G bids*)
using *assms lm010 allAllocationsFinite argmax-non-empty-iff*
by (*metis winningAllocationPossible rev-finite-subset*)

lemma *lm012*:
allAllocations N $\{\} \subseteq \{\{\}\}$
using *emptyset-part-emptyset3 rangeEmpty characterizationallAllocations mem-Collect-eq subsetI vimage-def* **by** *metis*

lemma *lm013*:
assumes $a \in$ *allAllocations N G finite G*
shows *finite* (*Range a*)
using *assms elementOfPartitionOfFiniteSetIsFinite* **by** (*metis allocationReverseInjective*)

corollary *allocationFinite*:
assumes $a \in$ *allAllocations N G finite G*
shows *finite a*
using *assms finite-converse Range-converse imageE allocationProperty finiteDomainImpliesFinite lm013*
by (*metis* (*erased*, *lifting*))

lemma *lm014*:
assumes $a \in$ *allAllocations N G finite G*
shows $\forall$ $y \in$ *Range a. finite y*
using *assms is-partition-of-def allocationInverseRangeDomainProperty*
by (*metis Union-upper rev-finite-subset*)

corollary *lm015*:
assumes $a \in$ *allAllocations N G finite G*
shows *card G* = *sum card* (*Range a*)
using *assms cardSumCommute lm013 allocationInverseRangeDomainProperty* **by** (*metis is-partition-of-def*)

10.3 Results on summed bid vectors

lemma *lm016*:
summedBidVectorRel bids N G =
{(*pair*, *sum* (%*g. bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*)))|
pair. pair $\in N \times$ (*Pow G*$-\{\{\}\}$)}
by *blast*

corollary *lm017*:
{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G*−{{}})) } || *a* =
{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G*−{{}})) ∩ *a*}
by (*metis restrictionVsIntersection*)

corollary *lm018*:
(*summedBidVectorRel bids N G*) || *a* =
{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G* − {{}})) ∩ *a*}
(**is** *?L* = *?R*)
proof −
let *?l* = *summedBidVectorRel*
let *?M* = {(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ *N* × (*Pow G*−{{}})}
have *?l bids N G* = *?M* **by** (*rule lm016*)
then have *?L* = (*?M* || *a*) **by** *presburger*
moreover have ... = *?R* **by** (*rule lm017*)
ultimately show *?thesis* **by** *simp*
qed

lemma *lm019*:
(*summedBid bids*) ' ((*N* × (*Pow G* − {{}})) ∩ *a*) =
{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G* − {{}})) ∩ *a*}
by *blast*

corollary *lm020*:
(*summedBidVectorRel bids N G*) || *a* = (*summedBid bids*) ' ((*N* × (*Pow G* −
{{}})) ∩ *a*)
(**is** *?L*=*?R*)
proof −
let *?l*=*summedBidVectorRel*
let *?p*=*summedBid*
let *?M*={(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G* − {{}})) ∩ *a*}
have *?L* = *?M* **by** (*rule lm018*)
moreover have ... = *?R* **using** *lm019* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

lemma *summedBidInjective*:
inj-on (*summedBid bids*) *UNIV*
using *fst-conv inj-on-inverseI* **by** (*metis* (*lifting*))

corollary *lm021*:
inj-on (*summedBid bids*) *X*
using *fst-conv inj-on-inverseI* **by** (*metis* (*lifting*))

lemma *lm022*:
sum snd (*summedBidVectorRel bids N G*) =
sum (*snd* ∘ (*summedBid bids*)) (*N* × (*Pow G* − {{}}))
using *lm021 sum.reindex* **by** *blast*

corollary *lm023*:
snd (*summedBid bids pair*) = *sum bids* (*omega pair*)
using *sumCurry* **by** *force*

corollary *lm024*:
snd ∘ *summedBid bids* = (*sum bids*) ∘ *omega*
using *lm023* **by** *fastforce*

lemma *lm025*:
assumes *finite* (*finestpart* (*snd pair*))
shows *card* (*omega pair*) = *card* (*finestpart* (*snd pair*))
using *assms* **by** *force*

corollary *lm026*:
assumes *finite* (*snd pair*)
shows *card* (*omega pair*) = *card* (*snd pair*)
using *assms cardFinestpart card-cartesian-product-singleton* **by** *metis*

lemma *lm027*:
assumes {} ∉ *Range f runiq f*
shows *is-non-overlapping* (*omega* ‘ *f*)
proof −
let *?X=omega* ‘ *f* **let** *?p=finestpart*
{ **fix** *y1 y2*
assume *y1* ∈ *?X* ∧ *y2* ∈ *?X*
then obtain *pair1 pair2* **where**
y1 = *omega pair1* & *y2* = *omega pair2* & *pair1* ∈ *f* & *pair2* ∈ *f* **by** *blast*
then moreover have *snd pair1* ≠ {} & *snd pair1* ≠ {}
using *assms* **by** (*metis rev-image-eqI snd-eq-Range*)
ultimately moreover have *fst pair1* = *fst pair2* ⟷ *pair1* = *pair2*
using *assms runiq-basic surjective-pairing* **by** *metis*
ultimately moreover have *y1* ∩ *y2* ≠ {} ⟶ *y1* = *y2* **using** *assms* **by** *fast*
ultimately have *y1* = *y2* ⟷ *y1* ∩ *y2* ≠ {}
using *assms notEmptyFinestpart* **by** (*metis Int-absorb Times-empty insert-not-empty*)
}
thus *?thesis* **using** *is-non-overlapping-def*

by (*metis* (*lifting*, *no-types*) *inf-commute inf-sup-aci*(*1*))
qed

lemma *lm028*:
assumes {} ∉ *Range X*
shows *inj-on omega X*
proof −
let *?p=finestpart*
{
fix *pair1 pair2*
assume *pair1* ∈ *X* & *pair2* ∈ *X*
then have *snd pair1* ≠ {} & *snd pair2* ≠ {}
using *assms* **by** (*metis Range.intros surjective-pairing*)
moreover assume *omega pair1* = *omega pair2*
then moreover have *?p* (*snd pair1*) = *?p* (*snd pair2*) **by** *blast*
then moreover have *snd pair1* = *snd pair2* **by** (*metis finestPart nonEqualitySetOfSets*)
ultimately moreover have {*fst pair1*} = {*fst pair2*} **using** *notEmptyFinestpart*
by (*metis fst-image-times*)
ultimately have *pair1* = *pair2* **by** (*metis prod-eqI singleton-inject*)
}
thus *?thesis* **by** (*metis* (*lifting*, *no-types*) *inj-onI*)
qed

lemma *lm029*:
assumes {} ∉ *Range a* ∀ *X* ∈ *omega* ' *a*. *finite X*
is-non-overlapping (*omega* ' *a*)
shows *card* (*pseudoAllocation a*) = *sum* (*card* ∘ *omega*) *a*
(**is** *?L* = *?R*)
proof −
have *?L* = *sum card* (*omega* ' *a*)
unfolding *pseudoAllocation-def*
using *assms* **by** (*simp add*: *cardinalityPreservation*)
moreover have ... = *?R* **using** *assms*(*1*) *lm028 sum.reindex* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

lemma *lm030*:
card (*omega pair*)= *card* (*snd pair*)
using *cardFinestpart card-cartesian-product-singleton* **by** *metis*

corollary *lm031*:
card ∘ *omega* = *card* ∘ *snd*
using *lm030* **by** *fastforce*

corollary *lm032*:
assumes {} ∉ *Range a* ∀ *pair* ∈ *a*. *finite* (*snd pair*) *finite a runiq a*

shows *card* (*pseudoAllocation a*) = *sum* (*card* ∘ *snd*) *a*
proof −
let *?P=pseudoAllocation*
let *?c=card*
have ∀ *pair* ∈ *a*. *finite* (*omega pair*) **using** *finiteFinestpart assms* **by** *blast*
moreover have *is-non-overlapping* (*omega ' a*) **using** *assms lm027* **by** *force*
ultimately have *?c* (*?P a*) = *sum* (*?c* ∘ *omega*) *a* **using** *assms lm029* **by** *force*
moreover have ... = *sum* (*?c* ∘ *snd*) *a* **using** *lm031* **by** *metis*
ultimately show *?thesis* **by** *simp*
qed

corollary *lm033*:
assumes *runiq* ($a^\frown-1$) *runiq a finite a* {} ∉ *Range a* ∀ *pair* ∈ *a*. *finite* (*snd pair*)
shows *card* (*pseudoAllocation a*) = *sum card* (*Range a*)
using *assms sumPairsInverse lm032* **by** *force*

corollary *lm034*:
assumes *a* ∈ *allAllocations N G finite G*
shows *card* (*pseudoAllocation a*) = *card G*
proof −
have {} ∉ *Range a* **using** *assms* **by** (*metis emptyNotInRange*)
moreover have ∀ *pair* ∈ *a*. *finite* (*snd pair*) **using** *assms lm014 finitePairSecondRange* **by** *metis*
moreover have *finite a* **using** *assms allocationFinite* **by** *blast*
moreover have *runiq a* **using** *assms*
by (*metis* (*lifting*) *Int-lower1 in-mono allocationInjectionsUnivervseProperty mem-Collect-eq*)
moreover have *runiq* ($a^\frown-1$) **using** *assms*
by (*metis* (*mono-tags*) *injections-def characterizationallAllocations mem-Collect-eq*)
ultimately have *card* (*pseudoAllocation a*) = *sum card* (*Range a*) **using** *lm033* **by** *fast*
moreover have ... = *card G* **using** *assms lm015* **by** *metis*
ultimately show *?thesis* **by** *simp*
qed

corollary *lm035*:
assumes *pseudoAllocation aa* ⊆ *pseudoAllocation a* ∪ (*N* × (*finestpart G*))
finite (*pseudoAllocation aa*)
shows *sum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation a*) −
(*sum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation aa*)) =
card (*pseudoAllocation a*) −
card (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
using *assms subsetCardinality* **by** *blast*

corollary *lm036*:
assumes *pseudoAllocation aa* ⊆ *pseudoAllocation a* ∪ (*N* × (*finestpart G*))
finite (*pseudoAllocation aa*)
shows *int* (*sum* (*maxbid a N G*) (*pseudoAllocation a*)) −

int (*sum* (*maxbid a N G*) (*pseudoAllocation aa*)) =
int (*card* (*pseudoAllocation a*)) −
int (*card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*)))
using *differenceSumVsCardinality assms* **by** *blast*

lemma *lm037*:
pseudoAllocation {} = {}
unfolding *pseudoAllocation-def* **by** *simp*

corollary *lm038*:
assumes *a* ∈ *allAllocations N* {}
shows (*pseudoAllocation a*) = {}
unfolding *pseudoAllocation-def* **using** *assms lm012* **by** *blast*

corollary *lm039*:
assumes *a* ∈ *allAllocations N G finite G G* ≠ {}
shows *finite* (*pseudoAllocation a*)
proof −
have *card* (*pseudoAllocation a*) = *card G* **using** *assms*(*1*,*2*) *lm034* **by** *blast*
thus *finite* (*pseudoAllocation a*) **using** *assms*(*2*,*3*) **by** *fastforce*
qed

corollary *lm040*:
assumes *a* ∈ *allAllocations N G finite G*
shows *finite* (*pseudoAllocation a*)
using *assms finite.emptyI lm039 lm038* **by** (*metis* (*no-types*))

lemma *lm041*:
assumes *a* ∈ *allAllocations N G aa* ∈ *allAllocations N G finite G*
shows (*card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*)) = *card* (*pseudoAllocation a*)) =
(*pseudoAllocation a* = *pseudoAllocation aa*)
proof −
let *?P=pseudoAllocation*
let *?c=card*
let *?A=?P a*
let *?AA=?P aa*
have *?c ?A=?c G* & *?c ?AA=?c G* **using** *assms lm034* **by** (*metis* (*lifting*, *mono-tags*))
moreover have *finite ?A* & *finite ?AA* **using** *assms lm040* **by** *blast*
ultimately show *?thesis* **using** *assms cardinalityIntersectionEquality* **by** (*metis*(*no-types*,*lifting*))
qed

lemma *lm042*:
omega pair = {*fst pair*} × {{*y*}| *y*. *y* ∈ *snd pair*}
using *finestpart-def finestPart* **by** *auto*

lemma *lm043*:
omega pair = {(*fst pair*, {*y*})| *y*. *y* ∈ *snd pair*}
using *lm042 setOfPairs* **by** *metis*

lemma *lm044*:
pseudoAllocation a = ⋃ {{(*fst pair*, {*y*})| *y*. *y* ∈ *snd pair*}| *pair*. *pair* ∈ *a*}
unfolding *pseudoAllocation-def* **using** *lm043* **by** *blast*

lemma *lm045*:
⋃ {{(*fst pair*, {*y*})| *y*. *y* ∈ *snd pair*}| *pair*. *pair* ∈ *a*} =
{(*fst pair*, {*y*})| *y pair*. *y* ∈ *snd pair* & *pair* ∈ *a*}
by *blast*

corollary *lm046*:
pseudoAllocation a = {(*fst pair*, *Y*)| *Y pair*. *Y* ∈ *finestpart* (*snd pair*) & *pair* ∈ *a*}
unfolding *pseudoAllocation-def* **using** *setOfPairsEquality* **by** *fastforce*

lemma *lm047*:
assumes *runiq a*
shows {(*fst pair*, *Y*)| *Y pair*. *Y* ∈ *finestpart* (*snd pair*) & *pair* ∈ *a*} =
{(*x*, *Y*)| *Y x*. *Y* ∈ *finestpart* (*a*,,*x*) & *x* ∈ *Domain a*}
(**is** *?L=?R*)
using *assms Domain.DomainI fst-conv functionOnFirstEqualsSecond runiq-wrt-ex1 surjective-pairing*
by (*metis*(*opaque-lifting*,*no-types*))

corollary *lm048*:
assumes *runiq a*
shows *pseudoAllocation a* = {(*x*, *Y*)| *Y x*. *Y* ∈ *finestpart* (*a*,,*x*) & *x* ∈ *Domain a*}
unfolding *pseudoAllocation-def* **using** *assms lm047 lm046* **by** *fastforce*

corollary *lm049*:
Range (*pseudoAllocation a*) = ⋃ (*finestpart* ‘ (*Range a*))
unfolding *pseudoAllocation-def*
using *lm046 rangeSetOfPairs unionFinestPart* **by** *fastforce*

corollary *lm050*:
Range (*pseudoAllocation a*) = *finestpart* (⋃ (*Range a*))
using *commuteUnionFinestpart lm049* **by** *metis*

lemma *lm051*:
pseudoAllocation a = {(*fst pair*, {*y*})| *y pair*. *y* ∈ *snd pair* & *pair* ∈ *a*}
using *lm044 lm045* **by** (*metis* (*no-types*))

lemma *lm052*:
{(*fst pair*, {*y*})| *y pair*. *y* ∈ *snd pair* & *pair* ∈ *a*} =
{(*x*, {*y*})| *x y*. *y* ∈ ⋃ (*a*‘‘{*x*}) & *x* ∈ *Domain a*}

by *auto*

lemma *lm053*:
pseudoAllocation a = {(*x*, {*y*})| *x y*. *y* ∈ ⋃ (*a*‘‘{*x*}) & *x* ∈ *Domain a*}
(**is** *?L*=*?R*)
proof −
have *?L*={(*fst pair*, {*y*})| *y pair*. *y* ∈ *snd pair* & *pair* ∈ *a*} **by** (*rule lm051*)
moreover have ... = *?R* **by** (*rule lm052*)
ultimately show *?thesis* **by** *simp*
qed

lemma *lm054*:
runiq (*summedBidVectorRel bids N G*)
using *graph-def image-Collect-mem domainOfGraph* **by** (*metis*(*no-types*))

corollary *lm055*:
runiq (*summedBidVectorRel bids N G* || *a*)
unfolding *restrict-def* **using** *lm054 subrel-runiq Int-commute* **by** *blast*

lemma *summedBidVectorCharacterization*:
N × (*Pow G* − {{}}) = *Domain* (*summedBidVectorRel bids N G*)
by *blast*

corollary *lm056*:
assumes *a* ∈ *allAllocations N G*
shows *a* ⊆ *Domain* (*summedBidVectorRel bids N G*)
proof −
let *?p*=*allAllocations*
let *?L*=*summedBidVectorRel*
have *a* ⊆ *N* × (*Pow G* − {{}}) **using** *assms allocationPowerset* **by** (*metis*(*no-types*))
moreover have *N* × (*Pow G* − {{}}) = *Domain* (*?L bids N G*) **using** *summed-BidVectorCharacterization* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

corollary *lm057*:
sum (*summedBidVector bids N G*) (*a* ∩ (*Domain* (*summedBidVectorRel bids N G*))) =
sum snd ((*summedBidVectorRel bids N G*) || *a*)
using *sumRestrictedToDomainInvariant lm055* **by** *fast*

corollary *lm058*:
assumes *a* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum snd* ((*summedBidVectorRel bids N G*) || *a*)
proof −
let *?l*=*summedBidVector* **let** *?L*=*summedBidVectorRel*
have *a* ⊆ *Domain* (*?L bids N G*) **using** *assms* **by** (*rule lm056*)
then have *a* = *a* ∩ *Domain* (*?L bids N G*) **by** *blast*

then have *sum* (*?l bids N G*) *a* = *sum* (*?l bids N G*) (*a* ∩ *Domain* (*?L bids N G*))
    by *presburger*
  thus *?thesis* **using** *lm057* **by** *auto*
qed

corollary *lm059*:
  assumes *a* ∈ *allAllocations N G*
  shows *sum* (*summedBidVector bids N G*) *a* =
      sum snd ((*summedBid bids*) ‘ ((*N* × (*Pow G* − {{}})) ∩ *a*))
    (**is** *?X*=*?R*)
proof −
  let *?p* = *summedBid*
  let *?L* = *summedBidVectorRel*
  let *?l* = *summedBidVector*
  let *?A* = *N* × (*Pow G* − {{}})
  let *?inner2* = (*?p bids*)‘(*?A* ∩ *a*)
  let *?inner1* = (*?L bids N G*)||*a*
  have *?R* = *sum snd ?inner1* **using** *assms lm020* **by** (*metis* (*no-types*))
  moreover have *sum* (*?l bids N G*) *a* = *sum snd ?inner1* **using** *assms* **by** (*rule lm058*)
  ultimately show *?thesis* **by** *simp*
qed

corollary *lm060*:
  assumes *a* ∈ *allAllocations N G*
  shows *sum* (*summedBidVector bids N G*) *a* = *sum snd* ((*summedBid bids*) ‘ *a*)
  (**is** *?L*=*?R*)
proof −
  let *?p*=*summedBid*
  let *?l*=*summedBidVector*
  have *?L* = *sum snd* ((*?p bids*)‘((*N* × (*Pow G* − {{}}))∩ *a*)) **using** *assms* **by** (*rule lm059*)
  moreover have ... = *?R* **using** *assms allocationPowerset Int-absorb1* **by** (*metis* (*no-types*))
  ultimately show *?thesis* **by** *simp*
qed

corollary *lm061*:
  sum snd ((*summedBid bids*) ‘ *a*) = *sum* (*snd* ∘ (*summedBid bids*)) *a*
  using *sum.reindex lm021* **by** *blast*

corollary *lm062*:
  assumes *a* ∈ *allAllocations N G*
  shows *sum* (*summedBidVector bids N G*) *a* = *sum* (*snd* ∘ (*summedBid bids*)) *a*
  (**is** *?L*=*?R*)
proof −
  let *?p* = *summedBid*
  let *?l* = *summedBidVector*

have *?L* = *sum snd* ((*?p bids*)' *a*) **using** *assms* **by** (*rule lm060*)
moreover have ... = *?R* **using** *assms lm061* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

corollary *lm063*:
assumes *a* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum* ((*sum bids*) ∘ *omega*) *a*
(**is** *?L=?R*)
proof −
let *?inner1* = *snd* ∘ (*summedBid bids*)
let *?inner2*=(*sum bids*) ∘ *omega*
let *?M=sum ?inner1 a*
have *?L* = *?M* **using** *assms* **by** (*rule lm062*)
moreover have *?inner1* = *?inner2* **using** *lm023 assms* **by** *fastforce*
ultimately show *?L* = *?R* **using** *assms* **by** *metis*
qed

corollary *lm064*:
assumes *a* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum* (*sum bids*) (*omega*' *a*)
proof −
have {} ∉ *Range a* **using** *assms* **by** (*metis emptyNotInRange*)
then have *inj-on omega a* **using** *lm028* **by** *blast*
then have *sum* (*sum bids*) (*omega* ' *a*) = *sum* ((*sum bids*) ∘ *omega*) *a*
by (*rule sum.reindex*)
moreover have *sum* (*summedBidVector bids N G*) *a* = *sum* ((*sum bids*) ∘ *omega*)
a
using *assms lm063* **by** (*rule Extraction.exE-realizer*)
ultimately show *?thesis* **by** *presburger*
qed

lemma *lm065*:
assumes *finite* (*snd pair*)
shows *finite* (*omega pair*)
using *assms finite.emptyI finite.insertI finite-SigmaI finiteFinestpart* **by** (*metis*(*no-types*))

corollary *lm066*:
assumes ∀ *y*∈(*Range a*). *finite y*
shows ∀ *y*∈(*omega* ' *a*). *finite y*
using *assms lm065 imageE finitePairSecondRange* **by** *fast*

corollary *lm067*:
assumes *a* ∈ *allAllocations N G finite G*
shows ∀ *x*∈(*omega* ' *a*). *finite x*
using *assms lm066 lm014* **by** (*metis*(*no-types*))

corollary *lm068*:
assumes *a* ∈ *allAllocations N G*

shows *is-non-overlapping* (*omega* ' *a*)
proof −
have *runiq a* **by** (*metis* (*no-types*) *assms image-iff allocationRightUniqueRange-Domain*)
moreover have {} ∉ *Range a* **using** *assms* **by** (*metis emptyNotInRange*)
ultimately show *?thesis* **using** *lm027* **by** *blast*
qed

lemma *lm069*:
assumes *a* ∈ *allAllocations N G finite G*
shows *sum* (*sum bids*) (*omega*' *a*) = *sum bids* (⋃ (*omega* ' *a*))
using *assms sumUnionDisjoint2 lm068 lm067* **by** (*metis* (*lifting*, *mono-tags*))

corollary *lm070*:
assumes *a* ∈ *allAllocations N G finite G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum bids* (*pseudoAllocation a*)
(**is** *?L* = *?R*)
proof −
have *?L* = *sum* (*sum bids*) (*omega* '*a*) **using** *assms lm064* **by** *blast*
moreover have ... = *sum bids* (⋃ (*omega* ' *a*)) **using** *assms lm069* **by** *blast*
ultimately show *?thesis* **unfolding** *pseudoAllocation-def* **by** *presburger*
qed

lemma *lm071*:
Domain (*pseudoAllocation a*) ⊆ *Domain a*
unfolding *pseudoAllocation-def* **by** *fastforce*

corollary *lm072*:
assumes *a* ∈ *allAllocations N G*
shows *Domain* (*pseudoAllocation a*) ⊆ *N* & *Range* (*pseudoAllocation a*) = *finestpart G*
using *assms lm071 allocationInverseRangeDomainProperty lm050 is-partition-of-def subset-trans*
by (*metis*(*no-types*))

corollary *lm073*:
assumes *a* ∈ *allAllocations N G*
shows *pseudoAllocation a* ⊆ *N* × *finestpart G*
proof −
let *?p* = *pseudoAllocation*
let *?aa* = *?p a*
let *?d* = *Domain*
let *?r* = *Range*
have *?d ?aa* ⊆ *N* **using** *assms lm072* **by** (*metis* (*lifting*, *mono-tags*))
moreover have *?r ?aa* ⊆ *finestpart G* **using** *assms lm072* **by** (*metis* (*lifting*, *mono-tags*) *equalityE*)
ultimately have *?d ?aa* × (*?r ?aa*) ⊆ *N* × *finestpart G* **by** *auto*
then show *?aa* ⊆ *N* × *finestpart G* **by** *auto*
qed

10.4 From Pseudo-allocations to allocations

abbreviation *pseudoAllocationInv pseudo* == $\{(x, \bigcup\ (pseudo\ ``\ \{x\}))|\ x.\ x \in Domain\ pseudo\}$

lemma *lm074*:
assumes *runiq a* $\{\} \notin$ *Range a*
shows *a* = *pseudoAllocationInv* (*pseudoAllocation a*)
proof −
let $?p=\{(x,\ Y)|\ Y\ x.\ Y \in finestpart\ (a,,x)\ \&\ x \in Domain\ a\}$
let $?a=\{(x, \bigcup\ (?p\ ``\ \{x\}))|\ x.\ x \in Domain\ ?p\}$
have $\forall x \in Domain\ a.\ a,,x \neq \{\}$ **by** (*metis assms eval-runiq-in-Range*)
then have $\forall x \in Domain\ a.\ finestpart\ (a,,x) \neq \{\}$ **by** (*metis notEmptyFinestpart*)

then have *Domain a* $\subseteq$ *Domain ?p* **by** *force*
moreover have *Domain a* $\supseteq$ *Domain ?p* **by** *fast*
ultimately have
1: *Domain a* = *Domain ?p* **by** *fast*
{
fix *z* **assume** $z \in ?a$
then obtain *x* **where**
$x \in Domain\ ?p\ \&\ z=(x\ ,\ \bigcup\ (?p\ ``\ \{x\}))$ **by** *blast*
then have $x \in Domain\ a\ \&\ z=(x\ ,\ \bigcup\ (?p\ ``\ \{x\}))$ **by** *fast*
then moreover have $?p``\{x\} = finestpart\ (a,,x)$ **using** *assms* **by** *fastforce*
moreover have $\bigcup\ (finestpart\ (a,,x))= a,,x$ **by** (*metis finestPartUnion*)
ultimately have $z \in a$ **by** (*metis assms*(*1*) *eval-runiq-rel*)
}
then have
2: $?a \subseteq a$ **by** *fast*
{
fix *z* **assume** *0*: $z \in a$ **let** *?x=fst z* **let** *?Y=a,,?x* **let** *?YY=finestpart ?Y*
have $z \in a\ \&\ ?x \in Domain\ a$ **using** *0* **by** (*metis fst-eq-Domain rev-image-eqI*)

then have
3: $z \in a\ \&\ ?x \in Domain\ ?p$ **using** *1* **by** *presburger*
then have $?p\ ``\ \{?x\} = ?YY$ **by** *fastforce*
then have $\bigcup\ (?p\ ``\ \{?x\}) = ?Y$ **by** (*metis finestPartUnion*)
moreover have $z = (?x,\ ?Y)$ **using** *assms* **by** (*metis 0 functionOnFirstEqualsSecond*
surjective-pairing)
ultimately have $z \in ?a$ **using** *3* **by** (*metis* (*lifting*, *mono-tags*) *mem-Collect-eq*)
}
then have $a = ?a$ **using** *2* **by** *blast*
moreover have *?p* = *pseudoAllocation a* **using** *lm048 assms* **by** (*metis* (*lifting*, *mono-tags*))
ultimately show *?thesis* **by** *auto*
qed

corollary *lm075*:
assumes $a \in runiqs \cap Pow\ (UNIV \times (UNIV - \{\{\}\}))$

shows (*pseudoAllocationInv* ∘ *pseudoAllocation*) *a* = *id a*
proof −
have *runiq a* **using** *runiqs-def assms* **by** *fast*
moreover have {} ∉ *Range a* **using** *assms* **by** *blast*
ultimately show *?thesis* **using** *lm074* **by** *fastforce*
qed

lemma *lm076*:
inj-on (*pseudoAllocationInv* ∘ *pseudoAllocation*) (*runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}})))
proof −
let *?ne*=*Pow* (*UNIV* × (*UNIV* − {{}}))
let *?X*=*runiqs* ∩ *?ne*
let *?f*=*pseudoAllocationInv* ∘ *pseudoAllocation*
have ∀ *a1* ∈ *?X*. ∀ *a2* ∈ *?X*. *?f a1* = *?f a2* ⟶ *id a1* = *id a2* **using** *lm075* **by** *blast*
then have ∀ *a1* ∈ *?X*. ∀ *a2* ∈ *?X*. *?f a1* = *?f a2* ⟶ *a1* = *a2* **by** *auto*
thus *?thesis* **unfolding** *inj-on-def* **by** *blast*
qed

corollary *lm077*:
inj-on pseudoAllocation (*runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}})))
using *lm076 inj-on-imageI2* **by** *blast*

lemma *lm078*:
injectionsUniverse ⊆ *runiqs*
using *runiqs-def Collect-conj-eq Int-lower1* **by** *metis*

lemma *lm079*:
partitionValuedUniverse ⊆ *Pow* (*UNIV* × (*UNIV* − {{}}))
using *is-non-overlapping-def* **by** *force*

corollary *lm080*:
allocationsUniverse ⊆ *runiqs* ∩ *Pow* (*UNIV* × (*UNIV* − {{}}))
using *lm078 lm079* **by** *auto*

corollary *lm081*:
inj-on pseudoAllocation allocationsUniverse
using *lm077 lm080 subset-inj-on* **by** *blast*

corollary *lm082*:
inj-on pseudoAllocation (*allAllocations N G*)
proof −
have *allAllocations N G* ⊆ *allocationsUniverse*
by (*metis* (*no-types*) *allAllocationsUniverse*)
thus *inj-on pseudoAllocation* (*allAllocations N G*) **using** *lm081 subset-inj-on* **by** *blast*
qed

lemma *lm083*:
assumes *card N > 0 distinct G*
shows *winningAllocationsRel N (set G) bids* ⊆ *set (allAllocationsAlg N G)*
using *assms winningAllocationPossible allAllocationsBridgingLemma* **by** (*metis*(*no-types*))

corollary *lm084*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows *winningAllocationsRel N (set G) bids* ∩ *set (allAllocationsAlg N G)* ≠ {}
proof −
let *?w = winningAllocationsRel*
let *?a = allAllocationsAlg*
let *?G = set G*
have *card N > 0* **using** *assms* **by** (*metis card-gt-0-iff*)
then have *?w N ?G bids* ⊆ *set (?a N G)* **using** *lm083* **by** (*metis assms*(*3*))
then show *?thesis* **using** *assms lm011* **by** (*metis List.finite-set le-iff-inf*)
qed

lemma *lm085*:
X = (%x. x ∈ X) −'{True}
by *blast*

corollary *lm086*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows (*%x. x∈winningAllocationsRel N (set G) bids*)−'{*True*} ∩
set (allAllocationsAlg N G) ≠ {}
using *assms lm084 lm085* **by** *metis*

lemma *lm087*:
assumes *P* −' {*True*} ∩ *set l* ≠ {}
shows *takeAll P l* ≠ []
using *assms nonEmptyListFiltered filterpositions2-def* **by** (*metis Nil-is-map-conv*)

corollary *lm088*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows *takeAll* (*%x. x ∈ winningAllocationsRel N (set G) bids*) (*allAllocationsAlg N G*) ≠ []
using *assms lm087 lm086* **by** *metis*

corollary *lm089*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows *perm2* (*takeAll* (*%x. x ∈ winningAllocationsRel N (set G) bids*)
(*allAllocationsAlg N G*))
n ≠ []
using *assms permutationNotEmpty lm088* **by** *metis*

corollary *lm090*:

assumes $N \neq \{\}$ *finite N distinct G set* $G \neq \{\}$
shows *chosenAllocation N G bids random* $\in$ *winningAllocationsRel N* (*set G*) *bids*
proof −
have $\bigwedge x_1$ *b-x x. set* $x_1 = \{\}$
∨ (*randomEl* x_1 *b-x*::('*a* × '*b set*) *set*) ∈ *x*
∨ ¬ *set* $x_1 \subseteq x$ **by** (*metis* (*no-types*) *randomElLemma subsetCE*)
thus *winningAllocationRel N* (*set G*)
((∈) (*randomEl* (*takeAll* (λ*x*. *winningAllocationRel N* (*set G*) ((∈) *x*) *bids*)
(*allAllocationsAlg N G*)) *random*)) *bids*
by (*metis lm088 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *takeAllSubset set-empty*)
qed

lemma *lm091*:
assumes *finite G a* ∈ *allAllocations N G aa* ∈ *allAllocations N G*
shows *real*(*sum*(*maxbid a N G*)(*pseudoAllocation a*)) −
sum(*maxbid a N G*)(*pseudoAllocation aa*) =
real (*card G*) −
card (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
proof −
let *?p* = *pseudoAllocation*
let *?f* = *finestpart*
let *?m* = *maxbid*
let *?B* = *?m a N G*
have *?p aa* ⊆ *N* × *?f G* **using** *assms lm073* **by** (*metis* (*lifting*, *mono-tags*))
then have *?p aa* ⊆ *?p a* ∪ (*N* × *?f G*) **by** *auto*
moreover have *finite* (*?p aa*) **using** *assms lm034 lm040* **by** *blast*
ultimately have *real*(*sum ?B* (*?p a*)) − *sum ?B* (*?p aa*) =
real(*card* (*?p a*))−*card*(*?p aa* ∩ (*?p a*))
using *differenceSumVsCardinalityReal* **by** *fast*
moreover have ... = *real* (*card G*) − *card* (*?p aa* ∩ (*?p a*))
using *assms lm034* **by** (*metis* (*lifting*, *mono-tags*))
ultimately show *?thesis* **by** *simp*
qed

lemma *lm092*:
summedBidVectorRel bids N G = *graph* (*N* × (*Pow G*−{{}})) (*summedBidSecond bids*)
unfolding *graph-def* **using** *lm016* **by** *blast*

lemma *lm093*:
assumes *x*∈*X*
shows *toFunction* (*graph X f*) *x* = *f x*
using *assms* **by** (*metis graphEqImage toFunction-def*)

corollary *lm094*:
assumes *pair* ∈ *N* × (*Pow G*−{{}})

shows *summedBidVector bids N G pair* = *summedBidSecond bids pair*
using *assms lm093 lm092* **by** (*metis*(*mono-tags*))

lemma *lm095*:
summedBidSecond (*real* ∘ ((*bids*:: - => *nat*))) *pair* = *real* (*summedBidSecond bids pair*)
by *simp*

lemma *lm096*:
assumes *pair* ∈ *N* × (*Pow G*−{{}})
shows *summedBidVector* (*real*∘(*bids*:: - => *nat*)) *N G pair* =
real (*summedBidVector bids N G pair*)
using *assms lm094 lm095* **by** (*metis*(*no-types*))

corollary *lm097*:
assumes *X* ⊆ *N* × (*Pow G* − {{}})
shows ∀ *pair* ∈ *X*. *summedBidVector* (*real* ∘ (*bids*::-=>*nat*)) *N G pair* =
(*real* ∘ (*summedBidVector bids N G*)) *pair*
proof −
{ fix $esk48_0$ :: ′*a* × ′*b set*
{ assume $esk48_0$ ∈ *N* × (*Pow G* − {{}})
hence *summedBidVector* (*real* ∘ *bids*) *N G* $esk48_0$ = *real* (*summedBidVector bids N G* $esk48_0$) **using** *lm096* **by** *blast*
hence $esk48_0$ ∉ *X* ∨ *summedBidVector* (*real* ∘ *bids*) *N G* $esk48_0$ = (*real* ∘ *summedBidVector bids N G*) $esk48_0$ **by** *simp* **}**
hence $esk48_0$ ∉ *X* ∨ *summedBidVector* (*real* ∘ *bids*) *N G* $esk48_0$ = (*real* ∘ *summedBidVector bids N G*) $esk48_0$ **using** *assms* **by** *blast* **}**
thus ∀ *pair*∈*X*. *summedBidVector* (*real* ∘ *bids*) *N G pair* = (*real* ∘ *summedBidVector bids N G*) *pair* **by** *blast*
qed

corollary *lm098*:
assumes *aa* ⊆ *N* × (*Pow G*−{{}})
shows *sum* ((*summedBidVector* (*real* ∘ (*bids*::-=>*nat*)) *N G*)) *aa* =
real (*sum* ((*summedBidVector bids N G*)) *aa*)
(**is** *?L*=*?R*)
proof −
have ∀ *pair* ∈ *aa*. *summedBidVector* (*real* ∘ *bids*) *N G pair* =
(*real* ∘ (*summedBidVector bids N G*)) *pair*
using *assms* **by** (*rule lm097*)
then have *?L* = *sum* (*real*∘(*summedBidVector bids N G*)) *aa* **using** *sum.cong* **by** *force*
then show *?thesis* **by** *simp*
qed

corollary *lm099*:
assumes *aa* ∈ *allAllocations N G*

```
  shows sum ((summedBidVector (real ∘ (bids::-=>nat)) N G)) aa =
        real (sum ((summedBidVector bids N G)) aa)
  using assms lm098 allocationPowerset by (metis(lifting,mono-tags))

corollary lm100:
  assumes finite G a ∈ allAllocations N G aa ∈ allAllocations N G
  shows real (sum (tiebids a N G) a) − sum (tiebids a N G) aa =
        real (card G) − card (pseudoAllocation aa ∩ (pseudoAllocation a))
  (is ?L=?R)
proof −
  let ?l=summedBidVector
  let ?m=maxbid
  let ?s=sum
  let ?p=pseudoAllocation
  let ?bb=?m a N G
  let ?b=real ∘ (?m a N G)
  have real (?s ?bb (?p a)) − (?s ?bb (?p aa)) = ?R using assms lm091 by blast
  then have
  1: ?R = real (?s ?bb (?p a)) − (?s ?bb (?p aa)) by simp
  have ?s (?l ?b N G) aa = ?s ?b (?p aa) using assms lm070 by blast moreover
have
  ... = ?s ?bb (?p aa) by fastforce
  moreover have (?s (?l ?b N G) aa) = real (?s (?l ?bb N G) aa) using assms(3)
by (rule lm099)
  ultimately have
  2: ?R = real (?s ?bb (?p a)) − (?s (?l ?bb N G) aa) by (metis 1)
  have ?s (?l ?b N G) a=(?s ?b (?p a)) using assms lm070 by blast
  moreover have ... = ?s ?bb (?p a) by force
  moreover have ... = real (?s ?bb (?p a)) by fast
  moreover have ?s (?l ?b N G) a = real (?s (?l ?bb N G) a) using assms(2)
by (rule lm099)
  ultimately have ?s (?l ?bb N G) a = real (?s ?bb (?p a)) by simp
  thus ?thesis using 2 by simp
qed

corollary lm101:
  assumes finite G a ∈ allAllocations N G aa ∈ allAllocations N G
          x = real (sum (tiebids a N G) a) − sum (tiebids a N G) aa
  shows x <= card G &
        x ≥ 0 &
       (x=0 ⟷ a = aa) &
       (aa ≠ a ⟶ sum (tiebids a N G) aa < sum (tiebids a N G) a)
proof −
  let ?p = pseudoAllocation
  have real (card G) >= real (card G) − card (?p aa ∩ (?p a)) by force
  moreover have real (sum (tiebids a N G) a) − sum (tiebids a N G) aa =
                real (card G) − card (pseudoAllocation aa ∩ (pseudoAllocation a))
          using assms lm100 by blast
  ultimately have
```

1: *x=real*(*card G*)−*card*(*pseudoAllocation aa*∩(*pseudoAllocation a*)) **using** *assms* **by** *force*
then have
2: *x* ≤ *real* (*card G*) **using** *assms* **by** *linarith*
have
3: *card* (*?p aa*) = *card G* & *card* (*?p a*) = *card G* **using** *assms lm034* **by** *blast*
moreover have *finite* (*?p aa*) & *finite* (*?p a*) **using** *assms lm040* **by** *blast*
ultimately have *card* (*?p aa* ∩ *?p a*) ≤ *card G* **using** *Int-lower2 card-mono* **by** *fastforce*
then have
4: *x* ≥ *0* **using** *assms lm100 1* **by** *linarith*
have *card* (*?p aa* ∩ (*?p a*)) = *card G* ⟷ (*?p aa* = *?p a*)
using *3 lm041 4 assms* **by** (*metis* (*lifting*, *mono-tags*))
moreover have *?p aa* = *?p a* ⟶ *a* = *aa* **using** *assms lm082 inj-on-def*
by (*metis* (*lifting*, *mono-tags*))
ultimately have *card* (*?p aa* ∩ (*?p a*)) = *card G* ⟷ (*a=aa*) **by** *blast*
moreover have *x* = *real* (*card G*) − *card* (*?p aa* ∩ (*?p a*)) **using** *assms lm100* **by** *blast*
ultimately have
5: *x* = *0* ⟷ (*a=aa*) **by** *linarith*
then have
aa ≠ *a* ⟶ *sum* (*tiebids a N G*) *aa* < *real* (*sum* (*tiebids a N G*) *a*)
using *1 4 assms* **by** *auto*
thus *?thesis* **using** *2 4 5*
unfolding *of-nat-less-iff* **by** *force*
qed

corollary *lm102*:
assumes *finite G a* ∈ *allAllocations N G*
aa ∈ *allAllocations N G aa* ≠ *a*
shows *sum* (*tiebids a N G*) *aa* < *sum* (*tiebids a N G*) *a*
using *assms lm101* **by** *blast*

lemma *lm103*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
aa ∈ (*allAllocations N* (*set G*))−{*chosenAllocation N G bids random*}
shows *sum* (*resolvingBid N G bids random*) *aa* <
sum (*resolvingBid N G bids random*) (*chosenAllocation N G bids random*)
proof −
let *?a=chosenAllocation N G bids random*
let *?p=allAllocations*
let *?G=set G*
have *?a* ∈ *winningAllocationsRel N* (*set G*) *bids* **using** *assms lm090* **by** *blast*
moreover have *winningAllocationsRel N* (*set G*) *bids* ⊆ *?p N ?G* **using** *assms winningAllocationPossible* **by** *metis*
ultimately have *?a* ∈ *?p N ?G* **using** *lm090 assms winningAllocationPossible rev-subsetD* **by** *blast*
then show *?thesis* **using** *assms lm102* **by** *blast*

qed

abbreviation *terminatingAuctionRel N G bids random ==*
argmax (*sum* (*resolvingBid N G bids random*))
(*argmax* (*sum bids*) (*allAllocations N* (*set G*)))

Termination theorem: it assures that the number of winning allocations is exactly one

theorem *winningAllocationUniqueness*:
assumes $N \neq \{\}$ *distinct G set G* $\neq \{\}$ *finite N*
shows *terminatingAuctionRel N G* (*bids*) *random* = {*chosenAllocation N G bids random*}
proof −
let *?p* = *allAllocations*
let *?G* = *set G*
let *?X* = *argmax* (*sum bids*) (*?p N ?G*)
let *?a* = *chosenAllocation N G bids random*
let *?b* = *resolvingBid N G bids random*
let *?f* = *sum ?b*
let *?t=terminatingAuctionRel*
have ∀ *aa* ∈ (*allAllocations N ?G*)−{*?a*}. *?f aa* < *?f ?a*
using *assms lm103* **by** *blast*
then have ∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a* **using** *assms lm103* **by** *auto*
moreover have *finite N* **using** *assms* **by** *simp*
then have *finite* (*?p N ?G*) **using** *assms allAllocationsFinite* **by** (*metis List.finite-set*)
then have *finite ?X* **using** *assms* **by** (*metis finite-subset winningAllocationPossible*)
moreover have *?a* ∈ *?X* **using** *lm090 assms* **by** *blast*
ultimately have *finite ?X* & *?a* ∈ *?X* & (∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a*) **by** *force*
moreover have (*finite ?X* & *?a* ∈ *?X* & (∀ *aa* ∈ *?X*−{*?a*}. *?f aa* < *?f ?a*)) ⟶ *argmax ?f ?X* = {*?a*}
by (*rule argmaxProperty*)
ultimately have {*?a*} = *argmax ?f ?X* **using** *injectionsFromEmptyIsEmpty* **by** *presburger*
moreover have ... = *?t N G bids random* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

The computable variant of Else is defined next as Elsee.

definition *toFunctionWithFallbackAlg R fallback ==*
(% *x*. *if* (*x* ∈ *Domain R*) *then* (*R*,,*x*) *else fallback*)
notation *toFunctionWithFallbackAlg* (**infix** ‹*Elsee*› *75*)

end

11 VCG auction: definitions and theorems

theory *CombinatorialAuction*

imports
UniformTieBreaking

begin

11.1 Definition of a VCG auction scheme, through the pair (*vcga*, *vcgp*)

abbreviation *participants b* == *Domain* (*Domain b*)

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

abbreviation *seller* == (*0*::*integer*)

abbreviation *allAllocations′ N* Ω ==
injectionsUniverse ∩ {*a*. *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions* Ω}

abbreviation *allAllocations″ N* Ω == *allocationsUniverse* ∩ {*a*. *Domain a* ⊆ *N* & ⋃(*Range a*) = Ω}

lemma *allAllocationsEquivalence*:
allAllocations N Ω = *allAllocations′ N* Ω & *allAllocations N* Ω = *allAllocations″ N* Ω
using *allocationInjectionsUnivervseProperty allAllocationsIntersection* **by** *metis*

lemma *allAllocationsVarCharacterization*:
(*a* ∈ *allAllocations″ N* Ω) = (*a* ∈ *allocationsUniverse*& *Domain a* ⊆ *N* & ⋃(*Range a*) = Ω)
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)
abbreviation *soldAllocations‴ N* Ω ==
allocationsUniverse ∩ {*aa*. *Domain aa*⊆*N*−{*seller*} & ⋃(*Range aa*)⊆Ω}
lemma *soldAllocationsEquivalence*:

soldAllocations N Ω = *soldAllocations′ N* Ω &
soldAllocations′ N Ω = *soldAllocations′′ N* Ω
using *allAllocationsEquivalence* **by** *metis*

corollary *soldAllocationsEquivalenceVariant*:
soldAllocations = *soldAllocations′* &
soldAllocations′ = *soldAllocations′′* &
soldAllocations = *soldAllocations′′*
using *soldAllocationsEquivalence* **by** *metis*

lemma *allocationSellerMonotonicity*:
soldAllocations (*N*−{*seller*}) Ω ⊆ *soldAllocations N* Ω
using *Outside-def* **by** *simp*

lemma *allocationsUniverseCharacterization*:
(*a* ∈ *allocationsUniverse*) = (*a* ∈ *allAllocations′′* (*Domain a*) (⋃(*Range a*)))
by *blast*

lemma *allocationMonotonicity*:
assumes *N1* ⊆ *N2*
shows *allAllocations′′ N1* Ω ⊆ *allAllocations′′ N2* Ω
using *assms* **by** *auto*

lemma *allocationWithOneParticipant*:
assumes *a* ∈ *allAllocations′′ N* Ω
shows *Domain* (*a* −− *x*) ⊆ *N*−{*x*}
using *assms Outside-def* **by** *fastforce*

lemma *soldAllocationIsAllocation*:
assumes *a* ∈ *soldAllocations N* Ω
shows *a* ∈ *allocationsUniverse*
proof −
obtain *aa* **where** *a* =*aa* −− *seller* & *aa* ∈ *allAllocations* (*N*∪{*seller*}) Ω
using *assms* **by** *blast*
then have *a* ⊆ *aa* & *aa* ∈ *allocationsUniverse*
unfolding *Outside-def* **using** *allAllocationsIntersectionSubset* **by** *blast*
then show *?thesis* **using** *subsetAllocation* **by** *blast*
qed

lemma *soldAllocationIsAllocationVariant*:
assumes *a* ∈ *soldAllocations N* Ω
shows *a* ∈ *allAllocations′′* (*Domain a*) (⋃(*Range a*))
proof −
show *?thesis* **using** *assms soldAllocationIsAllocation*
by *auto blast+*
qed

lemma *onlyGoodsAreSold*:
assumes *a* ∈ *soldAllocations′′ N* Ω

```
  shows ⋃ (Range a) ⊆ Ω
  using assms Outside-def by blast

lemma soldAllocationIsRestricted:
  a ∈ soldAllocations'' N Ω =
   (∃ aa. aa −− (seller) = a ∧ aa ∈ allAllocations'' (N ∪ {seller}) Ω)
  by blast


lemma restrictionConservation:
  (R +* ({x}×Y)) −− x = R −− x
  unfolding Outside-def paste-def by blast

lemma allocatedToBuyerMeansSold:
  assumes a ∈ allocationsUniverse Domain a ⊆ N−{seller} ⋃ (Range a) ⊆ Ω
  shows a ∈ soldAllocations'' N Ω
proof −
  let ?i = seller
  let ?Y = {Ω−⋃ (Range a)}−{{}}
  let ?b = {?i}×?Y
  let ?aa = a∪?b
  let ?aa' = a +* ?b
  have
  1: a ∈ allocationsUniverse using assms(1) by fast
  have ?b ⊆ {(?i,Ω−⋃(Range a))} − {(?i, {})} by fastforce
  then have
  2: ?b ∈ allocationsUniverse
    using allocationUniverseProperty subsetAllocation by (metis(no-types))
  have
  3: ⋃ (Range a) ∩ ⋃ (Range ?b) = {} by blast
  have
  4: Domain a ∩ Domain ?b ={} using assms by fast
  have ?aa ∈ allocationsUniverse using 1 2 3 4 by (rule allocationUnion)
  then have ?aa ∈ allAllocations'' (Domain ?aa) (⋃ (Range ?aa))
    unfolding allocationsUniverseCharacterization by metis
  then have ?aa ∈ allAllocations'' (N∪{?i}) (⋃ (Range ?aa))
    using allocationMonotonicity assms paste-def by auto
  moreover have Range ?aa = Range a ∪ ?Y by blast
  then moreover have ⋃ (Range ?aa) = Ω
   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}) Ω by simp
  moreover have Domain ?b ⊆ {?i} by fast
  have ?aa' −− ?i = a −− ?i by (rule restrictionConservation)
  moreover have ... = a using Outside-def assms(2) by auto
  ultimately show ?thesis using soldAllocationIsRestricted by auto
qed
```

lemma *allocationCharacterization*:
a ∈ *allAllocations N* Ω =
(*a* ∈ *injectionsUniverse* & *Domain a* ⊆ *N* & *Range a* ∈ *all-partitions* Ω)
by (*metis* (*full-types*) *posssibleAllocationsRelCharacterization*)

lemma *lm01*:
assumes *a* ∈ *soldAllocations'' N* Ω
shows *Domain a* ⊆ *N*−{*seller*} & *a* ∈ *allocationsUniverse*
proof −
let *?i* = *seller*
obtain *aa* **where**
0: *a* = *aa* −− *?i* & *aa* ∈ *allAllocations''* (*N* ∪ {*?i*}) Ω
using *assms*(*1*) *soldAllocationIsRestricted* **by** *blast*
then have *Domain aa* ⊆ *N* ∪ {*?i*} **using** *allocationCharacterization* **by** *blast*
then have *Domain a* ⊆ *N* − {*?i*} **using** *0 Outside-def* **by** *blast*
moreover have *a* ∈ *soldAllocations N* Ω **using** *assms soldAllocationsEquivalenceVariant* **by** *metis*
then moreover have *a* ∈ *allocationsUniverse* **using** *soldAllocationIsAllocation* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

corollary *lm02*:
assumes *a* ∈ *soldAllocations'' N* Ω
shows *a* ∈ *allocationsUniverse* & *Domain a* ⊆ *N*−{*seller*} & ⋃ (*Range a*) ⊆ Ω
proof −
have *a* ∈ *allocationsUniverse* **using** *assms lm01* [*of a*] **by** *blast*
moreover have *Domain a* ⊆ *N*−{*seller*} **using** *assms lm01* **by** *blast*
moreover have ⋃ (*Range a*) ⊆ Ω **using** *assms onlyGoodsAreSold* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

corollary *lm03*:
(*a* ∈ *soldAllocations'' N* Ω) =
(*a* ∈ *allocationsUniverse* & *a* ∈ {*aa*. *Domain aa* ⊆ *N*−{*seller*} & ⋃ (*Range aa*) ⊆ Ω})
(**is** *?L* = *?R*)
proof −
have (*a*∈*soldAllocations'' N* Ω) =
(*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*seller*} & ⋃ (*Range a*) ⊆ Ω)
using *lm02 allocatedToBuyerMeansSold* **by** (*metis* (*mono-tags*))
then have *?L* = (*a*∈*allocationsUniverse*& *Domain a* ⊆ *N*−{*seller*} & ⋃ (*Range a*) ⊆ Ω) **by** *fast*
moreover have ... = *?R* **using** *mem-Collect-eq* **by** (*metis* (*lifting*, *no-types*))
ultimately show *?thesis* **by** *auto*
qed

corollary *lm04*:
a ∈ *soldAllocations''* *N* Ω =
(*a*∈ (*allocationsUniverse* ∩ {*aa*. *Domain aa* ⊆ *N*−{*seller*} & ⋃ (*Range aa*) ⊆ Ω}))
using *lm03* **by** (*metis* (*mono-tags*) *Int-iff*)

corollary *soldAllocationVariantEquivalence*:
soldAllocations'' *N* Ω = *soldAllocations'''* *N* Ω
(**is** *?L*=*?R*)
proof −
{
fix *a*
have *a* ∈ *?L* = (*a* ∈ *?R*) **by** (*rule lm04*)
}
thus *?thesis* **by** *blast*
qed

lemma *lm05*:
assumes *a* ∈ *soldAllocations'''* *N* Ω
shows *a* −− *n* ∈ *soldAllocations'''* (*N*−{*n*}) Ω
proof −
let *?bb* = *seller*
let *?d* = *Domain*
let *?r* = *Range*
let *?X1* = {*aa*. *?d aa* ⊆ *N*−{*n*}−{*?bb*} & ⋃(*?r aa*)⊆Ω}
let *?X2* = {*aa*. *?d aa* ⊆ *N*−{*?bb*} & ⋃(*?r aa*) ⊆ Ω}
have *a*∈*?X2* **using** *assms*(*1*) **by** *fast*
then have
0: *?d a* ⊆ *N*−{*?bb*} & ⋃(*?r a*) ⊆ Ω **by** *blast*
then have *?d* (*a*−−*n*) ⊆ *N*−{*?bb*}−{*n*}
using *outside-reduces-domain* **by** (*metis Diff-mono subset-refl*)
moreover have ... = *N*−{*n*}−{*?bb*} **by** *fastforce*
ultimately have *?d* (*a*−−*n*) ⊆ *N*−{*n*}−{*?bb*} **by** *blast*
moreover have ⋃ (*?r* (*a*−−*n*)) ⊆ Ω
unfolding *Outside-def* **using** *0* **by** *blast*
ultimately have *a* −− *n* ∈ *?X1* **by** *fast*
moreover have *a*−−*n* ∈ *allocationsUniverse*
using *assms*(*1*) *Int-iff allocationsUniverseOutside* **by** (*metis*(*lifting*,*mono-tags*))

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

lemma *allAllocationsEquivalenceExtended*:
soldAllocations = *soldAllocations'* &
soldAllocations' = *soldAllocations''* &
soldAllocations'' = *soldAllocations'''*
using *soldAllocationVariantEquivalence soldAllocationsEquivalenceVariant* **by** *metis*

corollary *soldAllocationRestriction*:
  assumes $a \in$ *soldAllocations* $N\ \Omega$
  shows $a\ {-}{-}\ n \in$ *soldAllocations* $(N-\{n\})\ \Omega$
proof −
  let *?A′* = *soldAllocations′′′*
  have $a \in$ *?A′* $N\ \Omega$ **using** *assms allAllocationsEquivalenceExtended* **by** *metis*
  then have $a\ {-}{-}\ n \in$ *?A′* $(N-\{n\})\ \Omega$ **by** (*rule lm05*)
  thus *?thesis* **using** *allAllocationsEquivalenceExtended* **by** *metis*
qed

corollary *allocationGoodsMonotonicity*:
  assumes $\Omega 1 \subseteq \Omega 2$
  shows *soldAllocations′′′* $N\ \Omega 1 \subseteq$ *soldAllocations′′′* $N\ \Omega 2$
  using *assms* **by** *blast*

corollary *allocationGoodsMonotonicityVariant*:
  assumes $\Omega 1 \subseteq \Omega 2$
  shows *soldAllocations′′* $N\ \Omega 1 \subseteq$ *soldAllocations′′* $N\ \Omega 2$
proof −
  have *soldAllocations′′* $N\ \Omega 1$ = *soldAllocations′′′* $N\ \Omega 1$
    by (*rule soldAllocationVariantEquivalence*)
  moreover have ... $\subseteq$ *soldAllocations′′′* $N\ \Omega 2$
    using *assms*(*1*) **by** (*rule allocationGoodsMonotonicity*)
  moreover have ... = *soldAllocations′′* $N\ \Omega 2$ **using** *soldAllocationVariantEquivalence* **by** *metis*
  ultimately show *?thesis* **by** *auto*
qed

abbreviation *maximalStrictAllocations* $N\ \Omega\ b$ == *argmax* (*sum* b) (*allAllocations* $(\{seller\} \cup N)\ \Omega$)

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

abbreviation *vcgas* $N\ \Omega\ b\ r$ ==
  Outside′ {*seller*} ‘((*argmax*∘*sum*) (*randomBids* $N\ \Omega\ b\ r$)
        ((*argmax*∘*sum*) b (*allAllocations* $(N \cup \{seller\})$ (*set* Ω))))

abbreviation *vcga* $N\ \Omega\ b\ r$ == *the-elem* (*vcgas* $N\ \Omega\ b\ r$)

abbreviation *vcga′* $N\ \Omega\ b\ r$ ==
  (*the-elem* (*argmax* (*sum* (*randomBids* $N\ \Omega\ b\ r$))

(*maximalStrictAllocations N* (*set* Ω) *b*)))
−− *seller*

lemma *lm06*:
assumes *card* ((*argmax*∘*sum*) (*randomBids N* Ω *b r*)
((*argmax*∘*sum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set* Ω))))
= *1*
shows *vcga N* Ω *b r* =
(*the-elem* ((*argmax*∘*sum*) (*randomBids N* Ω *b r*)
((*argmax*∘*sum*) *b* (*allAllocations* ({*seller*}∪*N*) (*set*
Ω)))))
−− *seller*
using *assms cardOneTheElem* **by** *auto*

corollary *lm07*:
assumes *card* ((*argmax*∘*sum*) (*randomBids N* Ω *b r*)
((*argmax*∘*sum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set* Ω))))
= *1*
shows *vcga N* Ω *b r* = *vcga′ N* Ω *b r*
(**is** *?l* = *?r*)
proof −
have *?l* = (*the-elem* ((*argmax*∘*sum*) (*randomBids N* Ω *b r*)
((*argmax*∘*sum*) *b* (*allAllocations* ({*seller*}∪*N*) (*set*
Ω)))))
−− *seller*
using *assms* **by** (*rule lm06*)
moreover have ... = *?r* **by** *force*
ultimately show *?thesis* **by** *blast*
qed

lemma *lm08*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* ((*argmax*∘*sum*) (*randomBids N* Ω *bids random*)
((*argmax*∘*sum*) *bids* (*allAllocations* (*N*∪{*seller*}) (*set*
Ω)))) = *1*
(**is** *card ?l*=-)
proof −
let *?N* = *N*∪{*seller*}
let *?b′* = *randomBids N* Ω *bids random*
let *?s* = *sum*
let *?a* = *argmax*
let *?f* = *?a* ∘ *?s*
have
1: *?N*≠{} **by** *auto*
have
2: *finite ?N* **using** *assms*(*3*) **by** *simp*
have *?a* (*?s ?b′*) (*?a* (*?s bids*) (*allAllocations ?N* (*set* Ω))) =
{*chosenAllocation ?N* Ω *bids random*} (**is** *?L*=*?R*)

using *1 assms*(*1*) *assms*(*2*) *2* **by** (*rule winningAllocationUniqueness*)
moreover have *?L= ?f ?b′* (*?f bids* (*allAllocations ?N* (*set* Ω))) **by** *auto*
ultimately have *?l* = {*chosenAllocation ?N* Ω *bids random*} **by** *simp*
moreover have *card ...=1* **by** *simp* **ultimately show** *?thesis* **by** *simp*
qed

lemma *vcgaEquivalence*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *vcga N* Ω *b r* = *vcga′ N* Ω *b r*
using *assms lm07 lm08* **by** *blast*

theorem *vcgaDefiniteness*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* (*vcgas N* Ω *b r*) = *1*
proof −
have *card* ((*argmax*∘*sum*) (*randomBids N* Ω *b r*)
((*argmax*∘*sum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set* Ω)))) =
1
(**is** *card ?X* = -) **using** *assms lm08* **by** *blast*
moreover have (*Outside′*{*seller*}) ‘ *?X* = *vcgas N* Ω *b r* **by** *blast*
ultimately show *?thesis* **using** *cardOneImageCardOne* **by** *blast*
qed

lemma *vcgaDefinitenessVariant*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* (*argmax* (*sum* (*randomBids N* Ω *b r*))
(*maximalStrictAllocations N* (*set* Ω) *b*)) =
1
(**is** *card ?L*=-)
proof −
let *?n* = {*seller*}
have
1: (*?n* ∪ *N*)≠{} **by** *simp*
have
2: *finite* (*?n*∪*N*) **using** *assms*(*3*) **by** *fast*
have *terminatingAuctionRel* (*?n*∪*N*) Ω *b r* = {*chosenAllocation* (*?n*∪*N*) Ω *b r*}

using *1 assms*(*1*) *assms*(*2*) *2* **by** (*rule winningAllocationUniqueness*)
moreover have *?L* = *terminatingAuctionRel* (*?n*∪*N*) Ω *b r* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

theorem *winningAllocationIsMaximal*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *the-elem* (*argmax* (*sum* (*randomBids N* Ω *b r*))
(*maximalStrictAllocations N* (*set* Ω) *b*)) ∈
(*maximalStrictAllocations N* (*set* Ω) *b*)

(**is** *the-elem ?X* ∈ *?R*)
proof −
  have *card ?X=1* **using** *assms* **by** (*rule vcgaDefinitenessVariant*)
  moreover have *?X* ⊆ *?R* **by** *auto*
  ultimately show *?thesis* **using** *cardinalityOneTheElem* **by** *blast*
qed

corollary *winningAllocationIsMaximalWithoutSeller*:
  assumes *distinct* Ω *set* Ω ≠ {} *finite N*
  shows *vcga′ N* Ω *b r* ∈ (*Outside′* {*seller*}) ‘(*maximalStrictAllocations N* (*set* Ω) *b*)
  using *assms winningAllocationIsMaximal* **by** *blast*

lemma *maximalAllactionWithoutSeller*:
  (*Outside′* {*seller*}) ‘(*maximalStrictAllocations N* Ω *b*) ⊆ *soldAllocations N* Ω
  using *Outside-def* **by** *force*

lemma *onlyGoodsAreAllocatedAuxiliary*:
  assumes *distinct* Ω *set* Ω ≠ {} *finite N*
  shows *vcga′ N* Ω *b r* ∈ *soldAllocations N* (*set* Ω)
  (**is** *?a* ∈ *?A*)
proof −
  have *?a* ∈ (*Outside′* {*seller*}) ‘(*maximalStrictAllocations N* (*set* Ω) *b*)
    using *assms* **by** (*rule winningAllocationIsMaximalWithoutSeller*)
  thus *?thesis* **using** *maximalAllactionWithoutSeller* **by** *fastforce*
qed

theorem *onlyGoodsAreAllocated*:
  assumes *distinct* Ω *set* Ω ≠ {} *finite N*
  shows *vcga N* Ω *b r* ∈ *soldAllocations N* (*set* Ω)
  (**is** -∈*?r*)
proof −
  have *vcga′ N* Ω *b r* ∈ *?r* **using** *assms* **by** (*rule onlyGoodsAreAllocatedAuxiliary*)

  then show *?thesis* **using** *assms vcgaEquivalence* **by** *blast*
qed

corollary *neutralSeller*:
  assumes ∀ *X*. *X* ∈ *Range a* ⟶*b* (*seller*, *X*)=*0* *finite a*
  shows *sum b a* = *sum b* (*a*−−*seller*)
proof −
  let *?n* = *seller*
  have *finite* (*a*||{*?n*}) **using** *assms restrict-def* **by** (*metis finite-Int*)
  moreover have ∀ *z* ∈ *a*||{*?n*}. *b z=0* **using** *assms restrict-def* **by** *fastforce*
  ultimately have *sum b* (*a*||{*?n*}) = *0* **using** *assms* **by** (*metis sum.neutral*)
  thus *?thesis* **using** *sumOutside assms*(*2*) **by** (*metis add.comm-neutral*)
qed

corollary *neutralSellerVariant*:
  assumes ∀ *a*∈*A*. *finite a* & (∀ *X*. *X*∈*Range a* ⟶ *b* (*seller*, *X*)=*0*)
  shows {*sum b a*| *a*. *a*∈*A*} = {*sum b* (*a* −− *seller*)| *a*. *a*∈*A*}
  using *assms neutralSeller* **by** (*metis* (*lifting*, *no-types*))

lemma *vcgaIsMaximalAux1*:
  assumes *distinct* Ω *set* Ω ≠ {} *finite N*
  shows ∃ *a*. ((*a* ∈ (*maximalStrictAllocations N* (*set* Ω) *b*)) ∧ (*vcga′ N* Ω *b r* = *a* −− *seller*) &
  (*a* ∈ *argmax* (*sum b*) (*allAllocations* ({*seller*}∪*N*) (*set* Ω))))
  using *assms winningAllocationIsMaximalWithoutSeller* **by** *fast*

lemma *vcgaIsMaximalAux2*:
  assumes *distinct* Ω *set* Ω ≠ {} *finite N*
  ∀ *a* ∈ *allAllocations* ({*seller*}∪*N*) (*set* Ω). ∀ *X* ∈ *Range a*. *b* (*seller*, *X*)=*0*
  (**is** ∀ *a*∈*?X*. -)
  shows *sum b* (*vcga′ N* Ω *b r*) = *Max*{*sum b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)}
proof −
  let *?n* = *seller*
  let *?s* = *sum*
  let *?a* = *vcga′ N* Ω *b r*
  obtain *a* **where**
  0: *a* ∈ *maximalStrictAllocations N* (*set* Ω) *b* &
  ?a = *a*−−*?n* &
  (*a* ∈ *argmax* (*sum b*) (*allAllocations*({*seller*}∪*N*)(*set* Ω)))
  (**is** - & *?a*=- & *a*∈*?Z*)
  using *assms*(*1*,*2*,*3*) *vcgaIsMaximalAux1* **by** *blast*
  have
  1: ∀ *a* ∈ *?X*. *finite a* & (∀ *X*. *X*∈*Range a* ⟶ *b* (*?n*, *X*)=*0*)
  using *assms*(*4*) *List.finite-set allocationFinite* **by** *metis*
  have
  2: *a* ∈ *?X* **using** *0* **by** *auto* **have** *a* ∈ *?Z* **using** *0* **by** *fast*
  then have *a* ∈ *?X*∩{*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *injectionsUnionCommute* **by** *simp*
  then have *a* ∈ {*x*. *?s b x* = *Max* (*?s b* ‘ *?X*)} **using** *injectionsUnionCommute* **by** *simp*
  moreover have *?s b* ‘ *?X* = {*?s b a*| *a*. *a*∈*?X*} **by** *blast*
  ultimately have *?s b a* = *Max* {*?s b a*| *a*. *a*∈*?X*} **by** *auto*
  moreover have {*?s b a*| *a*. *a*∈*?X*} = {*?s b* (*a*−−*?n*)| *a*. *a*∈*?X*}
  using *1* **by** (*rule neutralSellerVariant*)
  moreover have ... = {*?s b a*| *a*. *a* ∈ *Outside′* {*?n*}‘*?X*} **by** *blast*
  moreover have ... = {*?s b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)} **by** *simp*
  ultimately have *Max* {*?s b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)} = *?s b a* **by** *simp*
  moreover have ... = *?s b* (*a*−−*?n*) **using** *1 2 neutralSeller* **by** (*metis* (*lifting*, *no-types*))
  ultimately show *?s b ?a*=*Max*{*?s b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)} **using** *0* **by** *simp*
qed

Adequacy theorem: The allocation satisfies the standard pen-and-paper specification of a VCG auction. See, for example, [5, § 1.2].

theorem *vcgaIsMaximal*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N* ∀ *X*. *b* (*seller*, *X*) = *0*
shows *sum b* (*vcga′ N* Ω *b r*) = *Max*{*sum b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)}
using *assms vcgaIsMaximalAux2* **by** *blast*

corollary *vcgaIsAllocationAllocatingGoodsOnly*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *vcga′ N* Ω *b r* ∈ *allocationsUniverse* & ⋃ (*Range* (*vcga′ N* Ω *b r*)) ⊆ *set* Ω
proof −
let *?a* = *vcga′ N* Ω *b r*
let *?n* = *seller*
obtain *a* **where**
0: *?a* = *a* −− *seller* & *a* ∈ *maximalStrictAllocations N* (*set* Ω) *b*
using *assms winningAllocationIsMaximalWithoutSeller* **by** *blast*
then moreover have
1: *a* ∈ *allAllocations* ({*?n*}∪*N*) (*set* Ω) **by** *auto*
moreover have *maximalStrictAllocations N* (*set* Ω) *b* ⊆ *allocationsUniverse*
by (*metis* (*lifting*, *mono-tags*) *winningAllocationPossible*
allAllocationsUniverse subset-trans)
ultimately moreover have *?a* = *a* −− *seller* & *a* ∈ *allocationsUniverse* **by** *blast*
then have *?a* ∈ *allocationsUniverse* **using** *allocationsUniverseOutside* **by** *auto*
moreover have ⋃ (*Range a*) = *set* Ω **using** *allAllocationsIntersectionSetEquals 1* **by** *metis*
then moreover have ⋃ (*Range ?a*) ⊆ *set* Ω **using** *Outside-def 0* **by** *fast*
ultimately show *?thesis* **using** *allocationsUniverseOutside Outside-def* **by** *blast*
qed

abbreviation *vcgp N* Ω *b r n* ==
Max (*sum b* ‘ (*soldAllocations* (*N*−{*n*}) (*set* Ω)))
− (*sum b* (*vcga N* Ω *b r* −− *n*))

theorem *vcgpDefiniteness*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows ∃! *y*. *vcgp N* Ω *b r n* = *y*
using *assms vcgaDefiniteness* **by** *simp*

lemma *soldAllocationsFinite*:
assumes *finite N finite* Ω
shows *finite* (*soldAllocations N* Ω)
using *assms allAllocationsFinite finite.emptyI finite.insertI finite-UnI finite-imageI*
by *metis*

The price paid by any participant is non-negative.

```
theorem NonnegPrices:
  assumes distinct Ω set Ω ≠ {} finite N
  shows vcgp N Ω b r n >= (0::price)
proof −
  let ?a = vcga N Ω b r
  let ?A = soldAllocations
  let ?f = sum b
  have ?a ∈ ?A N (set Ω) using assms by (rule onlyGoodsAreAllocated)
  then have ?a −− n ∈ ?A (N−{n}) (set Ω) by (rule soldAllocationRestriction)
  moreover have finite (?A (N−{n}) (set Ω))
    using assms(3) soldAllocationsFinite finite-set finite-Diff by blast
  ultimately have Max (?f‘(?A (N−{n}) (set Ω))) ≥ ?f (?a −− n)
  (is ?L >= ?R) by (rule maxLemma)
  then show ?L − ?R >=0 by linarith
qed

lemma allocationDisjointAuxiliary:
  assumes a ∈ allocationsUniverse and n1 ∈ Domain a and n2 ∈ Domain a and
n1 ≠ n2
  shows a,,n1 ∩ a,,n2 = {}
proof −
  have Range a ∈ partitionsUniverse using assms nonOverlapping 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 allocationDisjoint:
  assumes a ∈ allocationsUniverse and n1 ∈ Domain a and n2 ∈ Domain a and
n1 ≠ n2
  shows a,,,n1 ∩ a,,,n2 = {}
  using assms allocationDisjointAuxiliary imageEquivalence by fastforce
```

No good is assigned twice.

```
theorem PairwiseDisjointAllocations:
  assumes distinct Ω set Ω ≠ {} finite N  n1 ≠ n2
  shows (vcga′ N Ω b r),,,n1 ∩ (vcga′ N Ω b r),,,n2 = {}
proof −
  have vcga′ N Ω b r ∈ allocationsUniverse
    using vcgaIsAllocationAllocatingGoodsOnly assms by blast
  then show ?thesis using allocationDisjoint assms by fast
```

qed

Nothing outside the set of goods is allocated.

theorem *OnlyGoodsAllocated*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N g* ∈ (*vcga N* Ω *b r*),,,*n*
shows *g* ∈ *set* Ω
proof −
let *?a* = *vcga′ N* Ω *b r*
have *?a* ∈ *allocationsUniverse* **using** *assms*(*1*,*2*,*3*) *vcgaIsAllocationAllocatingGoodsOnly* **by** *blast*
then have *1*: *runiq ?a* **using** *assms*(*1*,*2*,*3*) **by** *blast*
have *2*: *n* ∈ *Domain ?a* **using** *assms vcgaEquivalence* **by** *fast*
with *1* **have** *?a*,,*n* ∈ *Range ?a* **using** *eval-runiq-in-Range* **by** *fast*
with *1 2* **have** *?a*,,,*n* ∈ *Range ?a* **using** *imageEquivalence* **by** *fastforce*
then have *g* ∈ ⋃ (*Range ?a*) **using** *assms vcgaEquivalence* **by** *blast*
moreover have ⋃ (*Range ?a*) ⊆ *set* Ω **using** *assms*(*1*,*2*,*3*) *vcgaIsAllocationAllocatingGoodsOnly* **by** *fast*
ultimately show *?thesis* **by** *blast*
qed

11.2 Computable versions of the VCG formalization

abbreviation *maximalStrictAllocationsAlg N* Ω *b* ==
argmax (*sum b*) (*set* (*allAllocationsAlg* ({*seller*}∪*N*) Ω))

definition *chosenAllocationAlg N* Ω *b* (*r*::*integer*) ==
(*randomEl* (*takeAll* (%*x*. *x*∈ (*argmax* ∘ *sum*) *b* (*set* (*allAllocationsAlg N* Ω)))
(*allAllocationsAlg N* Ω))
r)

definition *maxbidAlg a N* Ω == (*bidMaximizedBy a N* Ω) *Elsee 0*

definition *summedBidVectorAlg bids N* Ω == (*summedBidVectorRel bids N* Ω) *Elsee 0*

definition *tiebidsAlg a N* Ω == *summedBidVectorAlg* (*maxbidAlg a N* Ω) *N* Ω

definition *resolvingBidAlg N* Ω *bids random* ==
tiebidsAlg (*chosenAllocationAlg N* Ω *bids random*) *N* (*set* Ω)

definition *randomBidsAlg N* Ω *b random* == *resolvingBidAlg* (*N*∪{*seller*}) Ω *b random*

definition *vcgaAlgWithoutLosers N Ω b r* ==
(*the-elem* (*argmax* (*sum* (*randomBidsAlg N Ω b r*))
(*maximalStrictAllocationsAlg N Ω b*)))
−− *seller*

abbreviation *addLosers participantset allocation* == (*participantset* × {{}}) +∗ *allocation*

definition *vcgaAlg N Ω b r* = *addLosers N* (*vcgaAlgWithoutLosers N Ω b r*)

abbreviation *soldAllocationsAlg N Ω* ==
(*Outside′* {*seller*}) ' *set* (*allAllocationsAlg* (*N* ∪ {*seller*}) Ω)

definition *vcgpAlg N Ω b r n* (*winningAllocation*::*allocation*) =
Max (*sum b* ' (*soldAllocationsAlg* (*N*−{*n*}) Ω)) −
(*sum b* (*winningAllocation* −− *n*))

lemma *functionCompletion*:
assumes *x* ∈ *Domain f*
shows *toFunction f x* = (*f Elsee 0*) *x*
unfolding *toFunctionWithFallbackAlg-def* **by** (*metis assms toFunction-def*)

lemma *lm09*:
assumes *fst pair* ∈ *N snd pair* ∈ *Pow* Ω − {{}}
shows *sum* (%*g*. (*toFunction* (*bidMaximizedBy a N* Ω)) (*fst pair*, *g*))
(*finestpart* (*snd pair*)) =
sum (%*g*. ((*bidMaximizedBy a N* Ω) *Elsee 0*) (*fst pair*, *g*))
(*finestpart* (*snd pair*))
proof −
let *?f1* = %*g*.(*toFunction* (*bidMaximizedBy a N* Ω))(*fst pair*, *g*)
let *?f2* = %*g*.((*bidMaximizedBy a N* Ω) *Elsee 0*)(*fst pair*, *g*)
{
fix *g* **assume** *g* ∈ *finestpart* (*snd pair*)
then have
0: *g* ∈ *finestpart* Ω **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* Ω ∨ $x_1 \notin x_2$ **by** (*metis 0 mem-Sigma-iff*)

then have (*pseudoAllocation a* <| (*N* × *finestpart* Ω)) (*fst pair*, *g*) =
maxbidAlg a N Ω (*fst pair*, *g*)
unfolding *toFunctionWithFallbackAlg-def maxbidAlg-def*

by (*metis* (*no-types*) *domainCharacteristicFunction UnCI assms*(*1*) *toFunction-def*)
thus *?thesis* **unfolding** *maxbidAlg-def* **by** *blast*
qed
}
thus *?thesis* **using** *sum.cong* **by** *simp*
qed

corollary *lm10*:
assumes *pair* ∈ *N* × (*Pow* Ω − {{}})
shows *summedBid* (*toFunction* (*bidMaximizedBy a N* Ω)) *pair* =
summedBid ((*bidMaximizedBy a N* Ω) *Elsee 0*) *pair*
proof −
have *fst pair* ∈ *N* **using** *assms* **by** *force*
moreover have *snd pair* ∈ *Pow* Ω − {{}} **using** *assms*(*1*) **by** *force*
ultimately show *?thesis* **using** *lm09* **by** *blast*
qed

corollary *lm11*:
∀ *pair* ∈ *N* × (*Pow* Ω − {{}}).
summedBid (*toFunction* (*bidMaximizedBy a N* Ω)) *pair* =
summedBid ((*bidMaximizedBy a N* Ω) *Elsee 0*) *pair*
using *lm10* **by** *blast*

corollary *lm12*:
(*summedBid* (*toFunction* (*bidMaximizedBy a N* Ω))) ‘ (*N* × (*Pow* Ω − {{}}))=
(*summedBid* ((*bidMaximizedBy a N* Ω) *Elsee 0*)) ‘ (*N* × (*Pow* Ω − {{}}))
(**is** *?f1* ‘ *?Z* = *?f2* ‘ *?Z*)
proof −
have ∀ *z* ∈ *?Z*. *?f1 z* = *?f2 z* **by** (*rule lm11*)
thus *?thesis* **by** (*rule functionEquivalenceOnSets*)
qed

corollary *lm13*:
summedBidVectorRel (*toFunction* (*bidMaximizedBy a N* Ω)) *N* Ω =
summedBidVectorRel ((*bidMaximizedBy a N* Ω) *Elsee 0*) *N* Ω
using *lm12* **by** *metis*

corollary *maxbidEquivalence*:
summedBidVectorRel (*maxbid a N* Ω) *N* Ω =
summedBidVectorRel (*maxbidAlg a N* Ω) *N* Ω
unfolding *maxbidAlg-def* **using** *lm13* **by** *metis*

lemma *summedBidVectorEquivalence*:
assumes *x* ∈ (*N* × (*Pow* Ω − {{}}))
shows *summedBidVector* (*maxbid a N* Ω) *N* Ω *x* = *summedBidVectorAlg* (*maxbidAlg a N* Ω) *N* Ω *x*
(**is** *?f1 ?g1 N* Ω *x* = *?f2 ?g2 N* Ω *x*)

proof −
  let *?h1* = *maxbid a N Ω*
  let *?h2* = *maxbidAlg a N Ω*
  have *summedBidVectorRel ?h1 N Ω* = *summedBidVectorRel ?h2 N Ω*
    using *maxbidEquivalence* **by** *metis*
  moreover have *summedBidVectorAlg ?h2 N Ω* = (*summedBidVectorRel ?h2 N Ω*) *Elsee 0*
    unfolding *summedBidVectorAlg-def* **by** *fast*
  ultimately have *summedBidVectorAlg ?h2 N Ω*=*summedBidVectorRel ?h1 N Ω Elsee 0* **by** *simp*
  moreover have ... *x* = (*toFunction* (*summedBidVectorRel ?h1 N Ω*)) *x*
    using *assms functionCompletion summedBidVectorCharacterization* **by** (*metis* (*mono-tags*))
  ultimately have *summedBidVectorAlg ?h2 N Ω x* = (*toFunction* (*summedBidVectorRel ?h1 N Ω*)) *x*
    by (*metis* (*lifting*, *no-types*))
  thus *?thesis* **by** *simp*
qed

corollary *chosenAllocationEquivalence*:
  assumes *card N* > *0* **and** *distinct Ω*
  shows *chosenAllocation N Ω b r* = *chosenAllocationAlg N Ω b r*
  using *assms allAllocationsBridgingLemma*
  by (*metis* (*no-types*) *chosenAllocationAlg-def comp-apply*)

corollary *tiebidsBridgingLemma*:
  assumes *x* ∈ (*N* × (*Pow* Ω − {{}}))
  shows *tiebids a N Ω x* = *tiebidsAlg a N Ω x*
  (**is** *?L*=-)
proof −
  have *?L* = *summedBidVector* (*maxbid a N Ω*) *N Ω x* **by** *fast*
  moreover have ...= *summedBidVectorAlg* (*maxbidAlg a N Ω*) *N Ω x*
    using *assms* **by** (*rule summedBidVectorEquivalence*)
  ultimately show *?thesis* **unfolding** *tiebidsAlg-def* **by** *fast*
qed

definition *tiebids′*=*tiebids*
corollary *tiebidsBridgingLemma′*:
  assumes *x* ∈ (*N* × (*Pow* Ω − {{}}))
  shows *tiebids′ a N Ω x* = *tiebidsAlg a N Ω x*
using *assms tiebidsBridgingLemma tiebids′-def* **by** *metis*

abbreviation *resolvingBid′ N G bids random* ==
  tiebids′ (*chosenAllocation N G bids random*) *N* (*set G*)

lemma *resolvingBidEquivalence*:
  assumes *x* ∈ (*N* × (*Pow* (*set* Ω) − {{}})) *card N* > *0 distinct* Ω
  shows *resolvingBid′ N Ω b r x* = *resolvingBidAlg N Ω b r x*

using *assms chosenAllocationEquivalence tiebidsBridgingLemma′ resolvingBidAlg-def*
by *metis*

lemma *sumResolvingBidEquivalence*:
assumes *card N > 0 distinct* Ω *a* ⊆ (*N* × (*Pow* (*set* Ω) − {{}}))
shows *sum* (*resolvingBid′ N* Ω *b r*) *a* = *sum* (*resolvingBidAlg N* Ω *b r*) *a*
(**is** *?L=?R*)
proof −
have ∀ *x*∈*a*. *resolvingBid′ N* Ω *b r x* = *resolvingBidAlg N* Ω *b r x*
using *assms resolvingBidEquivalence* **by** *blast*
thus *?thesis* **using** *sum.cong* **by** *force*
qed

lemma *resolvingBidBridgingLemma*:
assumes *card N > 0 distinct* Ω *a* ⊆ (*N* × (*Pow* (*set* Ω) − {{}}))
shows *sum* (*resolvingBid N* Ω *b r*) *a* = *sum* (*resolvingBidAlg N* Ω *b r*) *a*
(**is** *?L=?R*)
proof −
have *?L=sum* (*resolvingBid′ N* Ω *b r*) *a* **unfolding** *tiebids′-def* **by** *fast*
moreover have ...=*?R*
using *assms* **by** (*rule sumResolvingBidEquivalence*)
ultimately show *?thesis* **by** *simp*
qed

lemma *allAllocationsInPowerset*:
allAllocations N Ω ⊆ *Pow* (*N* × (*Pow* Ω − {{}}))
by (*metis PowI allocationPowerset subsetI*)

corollary *resolvingBidBridgingLemmaVariant1*:
assumes *card N > 0 distinct* Ω *a* ∈ *allAllocations N* (*set* Ω)
shows *sum* (*resolvingBid N* Ω *b r*) *a* = *sum* (*resolvingBidAlg N* Ω *b r*) *a*
proof −
have *a* ⊆ *N* × (*Pow* (*set* Ω) − {{}}) **using** *assms*(*3*) *allAllocationsInPowerset*
by *blast*
thus *?thesis* **using** *assms*(*1*,*2*) *resolvingBidBridgingLemma* **by** *blast*
qed

corollary *resolvingBidBridgingLemmaVariant2*:
assumes *finite N distinct* Ω *a* ∈ *maximalStrictAllocations N* (*set* Ω) *b*
shows *sum* (*randomBids N* Ω *b r*) *a* = *sum* (*randomBidsAlg N* Ω *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* Ω **using** *assms*(*2*) **by** *simp*
moreover have *a* ∈ *allAllocations* (*N*∪{*seller*}) (*set* Ω) **using** *assms*(*3*) **by** *fastforce*
ultimately show *?thesis* **unfolding** *randomBidsAlg-def* **by** (*rule resolvingBid-BridgingLemmaVariant1*)
qed

corollary *tiebreakingGivesSingleton*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* (*argmax* (*sum* (*randomBidsAlg N* Ω *b r*))
(*maximalStrictAllocations N* (*set* Ω) *b*)) =
1
proof −
have ∀ *a* ∈ *maximalStrictAllocations N* (*set* Ω) *b*.
sum (*randomBids N* Ω *b r*) *a* = *sum* (*randomBidsAlg N* Ω *b r*) *a*
using *assms*(*3*,*1*) *resolvingBidBridgingLemmaVariant2* **by** *blast*
then have *argmax* (*sum* (*randomBidsAlg N* Ω *b r*)) (*maximalStrictAllocations N* (*set* Ω) *b*) =
argmax (*sum* (*randomBids N* Ω *b r*)) (*maximalStrictAllocations N* (*set* Ω) *b*)
using *argmaxEquivalence* **by** *blast*
moreover have *card* ... = *1* **using** *assms* **by** (*rule vcgaDefinitenessVariant*)
ultimately show *?thesis* **by** *simp*
qed

theorem *maximalAllocationBridgingTheorem*:
assumes *finite N distinct* Ω
shows *maximalStrictAllocations N* (*set* Ω) *b* = *maximalStrictAllocationsAlg N* Ω *b*
proof −
let *?N* = {*seller*} ∪ *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*) *allAllocationsBridgingLemma* **by** *metis*
qed

theorem *vcgaAlgDefinedness*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* (*argmax* (*sum* (*randomBidsAlg N* Ω *b r*)) (*maximalStrictAllocationsAlg N* Ω *b*)) = *1*
proof −
have *card* (*argmax* (*sum* (*randomBidsAlg N* Ω *b r*)) (*maximalStrictAllocations N* (*set* Ω) *b*)) = *1*
using *assms* **by** (*rule tiebreakingGivesSingleton*)
moreover have *maximalStrictAllocations N* (*set* Ω) *b* = *maximalStrictAllocationsAlg N* Ω *b*
using *assms*(*3*,*1*) **by** (*rule maximalAllocationBridgingTheorem*)
ultimately show *?thesis* **by** *metis*
qed

end

12 VCG auction: Scala code extraction

theory *CombinatorialAuctionCodeExtraction*

imports
CombinatorialAuction

HOL−Library.Code-Target-Nat
HOL−Library.Code-Target-Int

begin

definition *allocationPrettyPrint a* =
{*map* (%*x*. (*x*, *sorted-list-of-set*(*a*,,*x*))) ((*sorted-list-of-set* ∘ *Domain*) *a*)}

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

definition *participantsSet b* = *fst* ‘ (*set b*)

definition *goodsList b* = *sorted-list-of-set* (*Union* ((*set o fst o snd*) ‘(*set b*)))

definition *payments b r n* (*a*::*allocation*) =
vcgpAlg ((*participantsSet b*)) (*goodsList b*) (*Bid2funcBid b*) *r n* (*a*::*allocation*)

export-code *vcgaAlg payments allocationPrettyPrint* **in** *Scala* **module-name** *VCG*

file ‹*VCG−withoutWrapper.scala*›

end

References

[1] Formare github webpage. https://github.com/formare/auctions/tree/master/isabelle/Auction/Vcg, 2015. Accessed: 2015-05-08.

[2] Formare project webpage. http://www.cs.bham.ac.uk/research/projects/formare/, 2015. Accessed: 2015-05-08.

[3] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Set theory or higher order logic to represent auction concepts in isabelle? In *Intelligent Computer Mathematics*, pages 236–251. Springer, 2014. http://arxiv.org/abs/1406.0774.

[4] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Sound auction specification and implementation. 16th ACM Conference on Economics and Computation, 2015. https://doi.org/10.1145/2764468.2764511, http://www.cs.bham.ac.uk/~mmk/publications/ec2015.pdf.

[5] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial auctions.* MIT Press, 2006.