Vickrey-Clarke-Groves (VCG) Auctions

M. B. Caminati\textsuperscript{*} M. Kerber\textsuperscript{*} C. Lange\textsuperscript{†} C. Rowat\textsuperscript{‡}

September 13, 2023

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

1 Introduction 3
1.1 Rationale for developing set theory as replacing one bidder in a second price auction 4
1.2 Bridging 4
1.3 Main theorems 4
1.4 Scala code extraction 5

2 Additional material that we would have expected in Set.thy 5
2.1 Equality 5
2.2 Trivial sets 6
2.3 The image of a set under a function 6
2.4 Big Union 6
2.5 Miscellaneous 7

\textsuperscript{*}School of Computer Science, University of Birmingham, UK
\textsuperscript{†}Fraunhofer IAIS and University of Bonn, Germany, and School of Computer Science, University of Birmingham, UK
\textsuperscript{‡}Department of Economics, University of Birmingham, UK
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Partitions of sets</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Locus where a function or a list (of linord type) attains its maximum value</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Additional operators on relations, going beyond Relations.thy, and properties of these operators</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Evaluating a relation as a function</td>
<td>28</td>
</tr>
<tr>
<td>5.2</td>
<td>Restriction</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Relation outside some set</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>Flipping pairs of relations</td>
<td>30</td>
</tr>
<tr>
<td>5.5</td>
<td>Evaluation as a function</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>Paste</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Additional properties of relations, and operators on relations, as they have been defined by Relations.thy</td>
<td>32</td>
</tr>
<tr>
<td>6.1</td>
<td>Right-Uniqueness</td>
<td>32</td>
</tr>
<tr>
<td>6.2</td>
<td>Converse</td>
<td>36</td>
</tr>
<tr>
<td>6.3</td>
<td>Injectivity</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>Toolbox of various definitions and theorems about sets, relations and lists</td>
<td>37</td>
</tr>
<tr>
<td>7.1</td>
<td>Facts and notations about relations, sets and functions.</td>
<td>37</td>
</tr>
<tr>
<td>7.2</td>
<td>Ordered relations</td>
<td>41</td>
</tr>
<tr>
<td>7.3</td>
<td>Indicator function in set-theoretical form.</td>
<td>53</td>
</tr>
<tr>
<td>7.4</td>
<td>Lists</td>
<td>55</td>
</tr>
<tr>
<td>7.5</td>
<td>Computing all the permutations of a list</td>
<td>56</td>
</tr>
<tr>
<td>7.6</td>
<td>A more computable version of toFunction.</td>
<td>58</td>
</tr>
<tr>
<td>7.7</td>
<td>Cardinalities of sets</td>
<td>60</td>
</tr>
<tr>
<td>7.8</td>
<td>Some easy properties on real numbers</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>Definitions about those Combinatorial Auctions which are strict (i.e., which assign all the available goods)</td>
<td>63</td>
</tr>
<tr>
<td>8.1</td>
<td>Types</td>
<td>63</td>
</tr>
<tr>
<td>8.2</td>
<td>VCG mechanism</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>Sets of injections, partitions, allocations expressed as suitable subsets of the corresponding universes</td>
<td>65</td>
</tr>
<tr>
<td>9.1</td>
<td>Preliminary lemmas</td>
<td>65</td>
</tr>
<tr>
<td>9.2</td>
<td>Definitions of various subsets of UNIV.</td>
<td>66</td>
</tr>
<tr>
<td>9.3</td>
<td>Results about the sets defined in the previous section</td>
<td>66</td>
</tr>
<tr>
<td>9.4</td>
<td>Bridging theorem for injections</td>
<td>84</td>
</tr>
<tr>
<td>9.5</td>
<td>Computable injections</td>
<td>90</td>
</tr>
</tbody>
</table>
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 \([?A \subseteq ?B; ?B \subseteq ?A] \implies \ ?A = ?B\) and \(\forall x. x \in ?A \implies x \in ?B\) \implies \ ?A \subseteq ?B\) to a single step

lemma equalitySubsetI: \(\forall x. x \in A \implies x \in B\) \implies \(\forall x. x \in B \implies x \in A\)

\implies A = B

by blast
2.2 Trivial sets

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

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

The empty set is trivial.

**lemma** trivial-empty: trivial $\{\}$

*unfolding* trivial-def *by* (rule empty-subsetI)

A singleton set is trivial.

**lemma** trivial-singleton: trivial $\{x\}$

*unfolding* trivial-def *by* simp

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

**lemma** singleton-sub-trivial-uniq:

- **fixes** $x \ X$
- **assumes** $\{x\} \subseteq X$ and trivial $X$
- **shows** $x = \text{the-elem } X$

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

Any subset of a trivial set is trivial.

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

*shows* trivial $X$

*using* assms *unfolding* trivial-def

*by* (metis (full-types) subset-empty subset-insertI2 subset-singletonD)

There are no two different elements in a trivial set.

**lemma** trivial-imp-no-distinct:

- **assumes** triv: trivial $X$ and $x: x \in X$ and $y: y \in X$
- **shows** $x = y$

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

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: $\{ f x \mid x \in S \} = f \cdot S$

*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: ($\exists \ S \in F . \ x \in S$) $\iff$ $x \in \bigcup F$

*by* blast
2.5 Miscellaneous

**lemma** trivial-subset-non-empty: **assumes** trivial \( t \cap X \neq \{\} \)
**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** \( (\text{finite } (A \times B) \& \text{ card } A \ast \text{ card } B \leq 1) \)
**using** trivial-def assms One-nat-def 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** \( \text{trivial } X = (\text{card } X \leq 1) \)
**using** assms One-nat-def card.empty card-insert-if card-mono card-seteq empty-iff
  \[ \text{empty-subsetI finite.cases finite.emptyI finite-insert insert-mono} \]
  \[ \text{trivial-def trivial-singleton} \]
**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** \( \neg \text{ trivial } X \) trivial \( T \)
**shows** \( X - T \neq \{\} \)
**using** assms by (metis Diff-iff empty-iff subsetI trivial-subset)

**lemma** lm06: **assumes** \( \text{finite } (A \times B) \& \text{ card } A \ast \text{ card } B \leq 1 \)
**shows** \( \text{trivial } (A \times B) \)
**unfolding** trivial-def **using** trivial-def assms by (metis card-cartesian-product lm02)

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

**lemma** trivial-empty-or-singleton: trivial \( X = (\{\} \lor X = \{\text{the-elem } X\}) \)
**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
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.

\[ P \text{ is a family of non-overlapping sets.} \]

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

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

**lemma** `subset-is-non-overlapping`:  
assumes \( P \subseteq Q \) and \( \text{non-overlapping} : Q \)  
shows \( \text{is-non-overlapping} \ P \)  

**proof** –  
{  
fix \( X, Y \) assume \( X \in P \land Y \in P \)  
then have \( X \in Q \land Y \in Q \) using `subset` by fast  
then have \( X \cap Y \neq \{\} \iff X = Y \) 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 $X P$ \& $Z \in$ insert $X P$
  have $Y \cap Z \neq \{\} \iff Y = Z$
  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
  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: $Y \in P - \{X\}$ \& $x \in Y$ by blast
  have $x \in X \cap Y \land Y \in P$
    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 \land 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 \setminus B
  and part: P partitions A
  shows \exists S \in P \setminus \{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 \langle x \in X \rangle \langle X \in P \rangle 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 \land x \in X
  by (rule elem-in-partition) blast
moreover {
A non-empty set “is” a partition of itself.

**Lemma set-partitions-itself:**

**Assumes** \( A \neq \emptyset \)

**Shows** \( \{ A \} \) partitions \( A \) unfolding is-partition-of-def is-non-overlapping-def

**Proof**

\[
\begin{align*}
\text{fix } X \text{ Y assume } Y \in P \land x \in Y \\
\text{then have } Y = X \\
\text{using part in-set} \\
\text{unfolding is-partition-of-def is-non-overlapping-def} \\
\text{by (metis disjoint-iff-not-equal)}
\end{align*}
\]

ultimately show ?thesis by (rule ex1I)

qed

The empty set is a partition of the empty set.

**Lemma emptyset-part-emptyset1:**

**Shows** \( \emptyset \) partitions \( \emptyset \) 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 \( \emptyset \)

**Shows** \( P = \emptyset \)

**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**

\[
\text{all-partitions } A = \{ P : P \text{ partitions } A \}
\]

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 \( \emptyset = \{ \{ \} \} \)
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 ⇒ 'a set ⇒ 'a set set
  where insert-into-member new-el Sets S = insert (S ∪ {new-el}) (Sets − {S})

Using insert-into-member to insert a fresh element, which is not a member
of the set S being partitioned, into a non-overlapping family of sets yields
another non-overlapping family.

lemma non-overlapping-extension2:
  fixes new-el::'a
  and P::'a set set
  and X::'a set
  assumes non-overlapping: is-non-overlapping P
  and class-element: X ∈ P
  and new: new-el /∈ ∪ 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 ∩ ∪ (P − {X}) = {}
    using non-overlapping class-element by (rule disj-eq-classes)
  from * have non-empty: ?Y ≠ {} by blast
  from * have disjoint: ?Y ∩ ∪ (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 ⇒ 'a set list ⇒ 'a set ⇒ 'a set list
  where insert-into-member-list new-el Sets S = (S ∪ {new-el}) # (remove1 S Sets)

insert-into-member-list and insert-into-member are equivalent (as in returning
the same set).
lemma insert-into-member-list-equivalence:
  fixes new-el:'a
  and Sets:'a set list
  and S:'a set
  assumes distinct Sets
  shows set (insert-into-member-list new-el Sets S) = insert-into-member new-el (set Sets) S
  unfolding insert-into-member-list-def insert-into-member-def using assms by simp

an alternative characterization of the set partitioned by a partition obtained
by inserting an element into an equivalence class of a given partition (if P
is a partition)

lemma insert-into-member-partition1:
  fixes elem:'a
  and P:'a set set
  and set:'a set
  shows ⋃ (insert-into-member elem P set) = ⋃ (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
  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 $\text{Set}$,  
— and suppose $\text{new-el} \not\in \text{Set}$, i.e. $\{\text{new-el}\} \not\in \text{set } P$,  
— then the following constructs a partition of $\text{Set} \cup \{\text{new-el}\}$ obtained by  
— inserting a new class $\{\text{new-el}\}$ and leaving all previous classes unchanged.  
$(\{\text{new-el}\} \# P) \#

\text{— Let } P \text{ be a partition of a set } \text{Set},  
\text{— and suppose } \text{new-el} \not\in \text{Set},  
\text{— then the following constructs}  
\text{— the set of those partitions of } \text{Set} \cup \{\text{new-el}\} \text{ obtained by}  
\text{— inserting } \text{new-el} \text{ into one class of } P \text{ at a time.}  
(map ((\text{insert-into-member-list } \text{new-el } P)) P)$

\text{coarser-partitions-with-list} \text{ and } \text{coarser-partitions-with} \text{ are equivalent.}

\text{lemma coarser-partitions-with-list-equivalence:}

\text{assumes distinct } P
\text{ shows set } (map \text{ set } (\text{coarser-partitions-with-list } \text{new-el } P)) = \text{coarser-partitions-with new-el } (\text{set } P)

\text{proof —}
\text{ have set } (map \text{ set } (\text{coarser-partitions-with-list } \text{new-el } P)) = \text{set } (map ((\{\text{new-el}\} \# P) \# (map ((\text{insert-into-member-list } \text{new-el } P)) P))

\text{ unfolding coarser-partitions-with-list-def ..}
\text{ also have } \ldots = \text{insert } (\text{insert } \{\text{new-el}\} \text{ set } P) ((\text{set } \circ (insert-into-member-list \text{new-el } P)) \setminus \text{set } P)

\text{by simp}
\text{ also have } \ldots = \text{insert } (\text{insert } \{\text{new-el}\} \text{ set } P) ((\text{insert-into-member new-el } \text{set } \text{set } P)) \setminus \text{set } P)

\text{using assms insert-into-member-list-equivalence by (metis comp-apply)}
\text{ finally show } \text{thesis unfolding coarser-partitions-with-def .}
\text{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.

\text{lemma non-overlapping-extension3:}

\text{fixes elem:'a}
\text{ and } P:'a set set
\text{ and } Q:'a set set
\text{ assumes P-non-overlapping: is-non-overlapping } P
\text{ and new-elem: elem } \notin \bigcup P
\text{ and Q-coarser: } Q \in \text{coarser-partitions-with elem } P
\text{ shows is-non-overlapping } Q

\text{proof —}
\text{ let } ?q = \text{insert } \{\text{elem}\} P
\text{ have Q-coarser-unfolded: } Q \in \text{insert } ?q (\text{insert-into-member elem } P \setminus P)

\text{ using Q-coarser}
\text{ unfolding coarser-partitions-with-def}

\text{ by fast}
\text{ show } \text{thesis}
\text{ proof (cases } Q = ?q)$
case True
  then show ?thesis 
    using P-non-overlapping new-elem non-overlapping-extension1 
    by fastforce 
next
case False 
  then have Q ∈ (insert-into-member elem P) ∪ P using Q-coarser-unfolded 
    by fastforce 
  then show ?thesis using non-overlapping-extension2 P-non-overlapping new-elem 
    by fast 
qed 
qed

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

lemma coarser-partitions-covers:
  fixes elem::'a 
  and P::'a set set 
  and Q::'a set set 
  assumes Q ∈ coarser-partitions-with elem P 
  shows ⋃ Q = insert elem (⋃ P) 
proof --
  let ?S = ⋃ P 
  have Q-cases: Q ∈ (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 ⋃ (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-monotone Un-insert-right insert-Diff-single) 
    with eq-class-in-P have ⋃ (insert (eq-class ∪ {elem}) (P − {eq-class})) = ??S 
      by blast 
    then have ⋃ (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 ∪ {elem} are child nodes of one partition of S.
definition partition-without :: 'a ⇒ 'a set set ⇒ 'a set set
where partition-without elem P = (λX . X − {elem}) ' P − {{}}

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

lemma partition-without-covers:
fixes elem::'a
and P::'a set set
shows ∪ (partition-without elem P) = (∪ P) − {elem}

proof −
have ∪ (partition-without elem P) = ∪ ((λx . x − {elem}) ' P − {{}})
unfolding partition-without-def by fast
also have . . . = ∪ 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 ∈ partition-without elem P
obtains Z where Z ∈ P and X = Z − {elem}

proof −
from assms have X ∈ (λX . X − {elem}) ' P − {{}} unfolding partition-without-def .
then obtain Z where Z-in-P: Z ∈ 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 ∀ X1 ∈ ?Q. ∀ X2 ∈ ?Q. X1 ∩ X2 ≠ {} ←→ X1 = X2
proof
fix X1 assume X1-in-Q: X1 ∈ ?Q
then obtain Z1 where Z1-in-P: Z1 ∈ P and Z1-sup: X1 = Z1 − {elem}
by (rule super-class)
have X1-non-empty: X1 ≠ {} using X1-in-Q partition-without-def by fast
show ∀ X2 ∈ ?Q. X1 ∩ X2 ≠ {} ←→ X1 = X2
proof
fix X2 assume X2 ∈ ?Q
then obtain Z2 where Z2-in-P: Z2 ∈ P and Z2-sup: X2 = Z2 − {elem}
by (rule super-class)

\[ X_1 \cap X_2 \neq \emptyset \rightarrow X_1 = X_2 \]

proof

assume \( X_1 \cap X_2 \neq \emptyset \)

then have \( Z_1 \cap Z_2 \neq \emptyset \) using \( Z_1\text{-sup} Z_2\text{-sup} \) by fast

then have \( Z_1 = Z_2 \) using \( Z_1\text{-in-P} Z_2\text{-in-P} \) assms unfolding is-non-overlapping-def

by fast

then show \( X_1 = X_2 \) using \( Z_1\text{-sup} Z_2\text{-sup} \) by fast

qed

moreover have \( X_1 = X_2 \) using \( Z_1\text{-sup} Z_2\text{-sup} \) by fast

ultimately show \( (X_1 \cap X_2 \neq \emptyset) \leftrightarrow X_1 = X_2 \) by blast

qed

coarser-partitions-with elem is the “inverse” of partition-without elem.

lemma coarser-partitions-inv-without:

fixes elem::'a

and \( P::'a \text{ set set} \)

assumes non-overlapping: is-non-overlapping \( P \)

and elem: elem \( \in \bigcup P \)

shows \( P \in \text{coarser-partitions-with elem (partition-without elem \( P \))} \)

(is \( P \in \text{coarser-partitions-with elem } Q \))

proof

let \( ?\text{remove-elem} = \lambda X \cdot X - \{ \text{elem} \} \)

obtain \( Y \)

where elem-eq-class: elem \( \in Y \) and elem-eq-class': \( Y \in P \) using elem ..

let \( ?\text{elem-neq-classes} = P - \{ Y \} \)

have P-wrt-elem: \( P = ?\text{elem-neq-classes} \cup \{ Y \} \) using elem-eq-class' by blast

let \( ?\text{elem-eq} = Y - \{ \text{elem} \} \)

have Y-elem-eq: \( ?\text{remove-elem} \ ' \{ Y \} = \{ ?\text{elem-eq} \} \) by fast

have elem-neq-classes-part: is-non-overlapping \( ?\text{elem-neq-classes} \)

using subset-is-non-overlapping non-overlapping

by blast

have elem-eq-wrt-P: \( ?\text{elem-eq} \in \text{?\text{remove-elem} } ' \ P \) using elem-eq-class' by blast

\{ fix \( W \) assume \( W\text{-eq-class}: W \in ?\text{elem-neq-classes} \)

then have \( \text{elem} \notin W \)

using elem-eq-class elem-eq-class' non-overlapping is-non-overlapping-def

by fast

then have \( ?\text{remove-elem} W = W \) by simp \}

then have elem-neq-classes-id: \( ?\text{remove-elem} \ ' \ ?\text{elem-neq-classes} = ?\text{elem-neq-classes} \)

by fastforce

have Q-unfolded: \( ?Q = ?\text{remove-elem} \ ' \ P - \{\{\}\} \)
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 $\notin$ P
using elem-eq-class elem-eq-class' non-overlapping Diff-Int-distrib2 Diff-iff empty-Diff
insert-iff
unfolding is-non-overlapping-def by metis
then have ?elem-eq $\notin$ P
using non-overlapping no-empty-in-non-overlapping
by metis
then have elem-neq-classes: ?elem-neq-classes − {?elem-eq} = ?elem-neq-classes
by fastforce

show ?thesis
proof cases
assume ?elem-eq $\notin$ ?Q
then have ?elem-eq $\in$ {{}}
using elem-eq-wrt-P Q-unfolded
by (metis DiffI)
then have Y-singleton: Y = {elem} using elem-eq-class by fast
then have ?Q = ?elem-neq-classes − {{}}
using Q-wrt-elem
by force
then have ?Q = ?elem-neq-classes
using no-empty-in-non-overlapping elem-neq-classes-part
by blast
then have insert {elem} ?Q = P
using Y-singleton elem-eq-class'
by fast
then show ?thesis unfolding coarser-partitions-with-def by auto
next
assume True: $\neg$ ?elem-eq $\notin$ ?Q
hence Y': ?elem-neq-classes ∪ {?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
Given a set \( P \) of partitions, this is intended to compute the set of all coarser partitions (given an extension element) of all partitions in \( P \).

**Definition** \( \text{all-coarser-partitions-with} :: \ a \Rightarrow \ (\ a \Rightarrow \ \text{set} \ \text{set} \ \text{list}) \Rightarrow \ (\ a \Rightarrow \ \text{set} \ \text{list} \ \text{list}) \)

where \( \text{all-coarser-partitions-with} \ \text{elem} \ \text{Ps} = \bigcup \) \( \text{(coarser-partitions-with elem ' Ps)} \)

the list variant of \( \text{all-coarser-partitions-with} \)

**Definition** \( \text{all-coarser-partitions-with-list} :: \ a \Rightarrow \ (\ a \Rightarrow \ \text{set} \ \text{list} \ \text{list}) \Rightarrow \ (\ a \Rightarrow \ \text{set} \ \text{list} \ \text{list}) \)

where \( \text{all-coarser-partitions-with-list} \ \text{elem} \ \text{Ps} = \) \( \text{concat} \) \( \text{(map (coarser-partitions-with-list elem) Ps)} \)

\( \text{all-coarser-partitions-with-list} \) and \( \text{all-coarser-partitions-with} \) are equivalent.

**Lemma** \( \text{all-coarser-partitions-with-list-equivalence} : \)

\f[ \text{fixes elem::'a} \quad \text{and Ps::'a set list list} \]

\f[ \text{assumes distinct: } \forall \ P \in \text{set Ps} . \text{ distinct P} \]

\f[ \text{shows set (map set (all-coarser-partitions-with-list elem Ps)) = all-coarser-partitions-with elem (set (map set Ps))} \]

\f[ \text{(is ?list-expr = ?set-expr)} \]

**Proof**

\f[ \text{have ?list-expr = set (map set (concat (map (coarser-partitions-with-list elem) Ps))} ]

\f[ \text{unfolding all-coarser-partitions-with-list-def ..} \]

\f[ \text{also have } \ldots = \text{set ' (} \bigcup \ x \in \text{(coarser-partitions-with-list elem) ' (set Ps)} . \text{ set x}) \text{ by simp} \]

\f[ \text{also have } \ldots = \text{set ' (} \bigcup \ x \in \text{ (coarser-partitions-with-list elem P) | P . P \in \text{ set Ps} \} . \text{ set x}) \]

\f[ \text{by (simp add: image-Collect-mem)} \]

\f[ \text{also have } \ldots = \bigcup \ { \text{set (map set (coarser-partitions-with-list elem P))} | P . P \in \text{ set Ps} \} \quad \text{by auto} \]

\f[ \text{also have } \ldots = \bigcup \ { \text{coarser-partitions-with elem (set P)} | P . P \in \text{ set Ps} \}

\f[ \text{using distinct coarser-partitions-with-list-equivalence by fast} \]

\f[ \text{also have } \ldots = \bigcup \ { \text{coarser-partitions-with elem ' (set (set Ps))} \text{ by (simp add: image-Collect-mem)}} \]

\f[ \text{also have } \ldots = \ ?set-expr unfolding all-coarser-partitions-with-def ..} \]

\f[ \text{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
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 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 \textit{all-partitions} and the algorithmic (constructive) definition \textit{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
let ?P-without-x = partition-without x P
have P-partitions-exc-x: ∪ ?P-without-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)

22
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
using all-coarser-partitions-with-def by fast
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 \( \bigcup P = \text{set } xs \cup \{x\} \)
using Q-covers P-in-Y Y-coarser’ coarser-partitions-covers by fast
then have \( \bigcup P = \text{set } (x \# xs) \)
using x-notin-xs P-wrt-Q Q-covers
by (metis List.set-simps(2) insert-is-Un sup-commute)
then have P partitions (set (x \# xs))
using P-partition unfolding is-partition-of-def by blast
then show P \in\ all-partitions (set (x \# xs)) unfolding all-partitions-def ..
qed
moreover have \( \forall ps \in \text{set } (\text{all-partitions-list } (x \# xs)) \) . distinct ps
proof
fix ps::’a set list assume ps-part: ps \in (\text{all-partitions-list } (x \# xs))

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

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

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

from Cons.prems have \( x \notin \text{set } xs \) by simp
then have new: \( \{x\} \notin \text{set } qs \)
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 \( \text{case} .. \)

qed

The classical definition \textit{all-partitions} and the algorithmic (constructive) definition \textit{all-partitions-list} are equivalent. This is a front-end theorem derived from \( \text{distinct ?xs} \implies \text{set (map set (all-partitions-list ?xs))} = \text{all-partitions (set ?xs)} \land (\forall ps \in \text{set (all-partitions-list ?xs)}. \text{distinct ps}); \) it does not make the auxiliary statement about partitions being distinct lists.

\textbf{theorem} all-partitions-paper-equiv-alg:
  \textbf{fixes} \( \text{xs} :: 'a \text{ list} \)
  \textbf{shows} \( \text{distinct \( \text{xs} \implies \text{set (map set (all-partitions-list \text{xs}))} = \text{all-partitions (set \text{xs})} \)) \)
  \textbf{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 \textit{all-partitions-list}

\textbf{definition} all-partitions-alg :: 'a::linorder set \Rightarrow 'a set list list
  \textbf{where} \( \text{all-partitions-alg} \text{ X} = \text{all-partitions-list (sorted-list-of-set X)} \)

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.

\textbf{lemma} structInduct: \textbf{assumes} \( P [] \) \textbf{and} \( \forall x \text{ xs}. \ P (\text{xs}) \implies P (x \text{"} xs) \)
  \textbf{shows} \( P l \)
  \textbf{using} \text{assms list-nonempty-induct by (metis)}

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

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

\textbf{lemma} argmaxLemma: argmax \( \text{f A} = \{ x \in A . \text{f x} = \text{Max (f ' A)} \} \)
  \textbf{by} simp

\textbf{lemma} maxLemma:
  \textbf{assumes} \( x \in X \text{ finite X} \)
  \textbf{shows} \( \text{Max (f'X)} \geq f x \)
  \text{is} \( ?L \geq ?R \) \textbf{using} \text{assms}
  \textbf{by} \ (\text{metis (opaque-lifting, no-types) Max.coboundedI finite-imageI image-eqI})
lemma \textit{lm01}:
\[ \text{argmax } f \ A = A \cap \{ \text{Max } (f \ A) \} \]
\textbf{by force}

lemma \textit{lm02}:
\begin{itemize}
  \item assumes \( y \in f' A \)
  \item shows \( A \cap f' \{y\} \neq \{\} \)
\end{itemize}
\textbf{using assms by blast}

lemma \textit{argmaxEquivalence}:
\begin{itemize}
  \item assumes \( \forall x \in X. f x = g x \)
  \item shows \( \text{argmax } f X = \text{argmax } g X \)
\end{itemize}
\textbf{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.

\textbf{corollary} \textit{argmax-non-empty-iff}:
\begin{itemize}
  \item assumes \textit{finite} \( X \neq \{\} \)
  \item shows \( \text{argmax } f X \neq \{\} \)
\end{itemize}
\textbf{using assms Max-in finite-imageI image-is-empty \textit{lm01} \textit{lm02} by (metis (no-types))}

The previous definition of \textit{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 \textit{filterpositions} and \textit{filterpositions2}.

\textbf{definition} \textit{filterpositions} :: \(\forall \ a \Rightarrow \text{bool} \Rightarrow \text{a list} \Rightarrow \text{nat list} \)
\textbf{where} \textit{filterpositions} \( P \ l = \text{map snd } (\text{filter } (P \circ \text{fst}) (\text{zip } l (\text{upt } 0 \text{ size } l))) \)

\textbf{definition} \textit{filterpositions2}
\textbf{where} \textit{filterpositions2} \( P \ l = [n. n \leftarrow [0..<\text{size } l], P (!!n)] \)

\textbf{definition} \textit{maxpositions}
\textbf{where} \textit{maxpositions} \( l = \text{filterpositions2 } (\%x . x \geq \text{Max } (\text{set } l)) \ l \)

\textbf{lemma} \textit{lm03}: \textit{maxpositions} \( l = [n. n \leftarrow [0..<\text{size } l], !!n \geq \text{Max}(\text{set } l)] \)
\textbf{unfolding} \textit{maxpositions-def filterpositions2-def by fastforce}

\textbf{definition} \textit{argmaxList}
\textbf{where} \textit{argmaxList} \( f \ l = \text{map } (\text{nth } l) (\text{maxpositions } (\text{map } f \ l)) \)
lemma lm04: \([n . n \leftarrow l, P n] = [n . n \leftarrow l, n \in \text{set } l, P n]\)
proof -

have \(\text{map } (\lambda uu. \text{if } P uu \text{ then } [uu] \text{ else } []) \ l = \)
\(\text{map } (\lambda uu. \text{if } uu \in \text{set } l \text{ then } \text{if } P uu \text{ then } [uu] \text{ else } []) \ l\) by simp
thus \(\text{concat } (\text{map } (\lambda n. \text{if } P n \text{ then } [n] \text{ else } []) \ l) = \)
\(\text{concat } (\text{map } (\lambda n. \text{if } n \in \text{set } l \text{ then } \text{if } P n \text{ then } [n] \text{ else } []) \ l)\) by presburger
qed

lemma lm05: \([n . n \leftarrow [0..<m], P n] = [n . n \leftarrow [0..<m], n \in \text{set } [0..<m], P n]\)
using lm04 by fast

lemma lm06: \(\text{fixes } f \text{ m } P\)
shows \(\text{map } (\text{map } f \ [n . n \leftarrow [0..<m], P n]) = [f n . n \leftarrow [0..<m], P n]\)
by (induct m) auto

lemma map-commutes-a: \([f n . n \leftarrow [], Q (f n)] = [x \leftarrow (\text{map } f []). Q x]\)
by simp

lemma map-commutes-b: \(\forall \ x \ xs. ([f n . n \leftarrow xs, \ Q (f n)] = [x \leftarrow (\text{map } f xs). Q x])\)
\([f n . n \leftarrow (x#xs), Q (f n)] = [x \leftarrow (\text{map } f (x#xs)). Q x])\)
by simp

lemma map-commutes: \(\text{fixes } f::'a \Rightarrow 'b \text{ fixes } Q::'b \Rightarrow \text{bool} \text{ fixes } xs::'a \text{ list}\)
shows \([f n . n \leftarrow xs, Q (f n)] = [x \leftarrow (\text{map } f xs). Q x]\)
using map-commutes-a map-commutes-b structInduct by fast

lemma lm07: \(\text{fixes } f \ l\)
shows \(\text{maxpositions } (\text{map } f \ l) = \)
\([n . n \leftarrow [0..<\text{size } l], f (\#n) \geq \text{Max } (f(\text{set } l))]\)
\(\text{(is } \text{maxpositions } (\#f) = \_))\)
proof -

have \(\text{maxpositions } \#f = \)
\([n . n \leftarrow [0..<\text{size } \#f], n \in \text{set}[0..<\text{size } \#f], \#f!n \geq \text{Max } (\text{set } \#f)]\)
using lm04 unfolding filterpositions2-def maxpositions-def .
also have ... =
\([n . n \leftarrow [0..<\text{size } l], (\#f!n \geq \text{Max } (\text{set } \#f))]\) by simp
also have ... =
\([n . n \leftarrow [0..<\text{size } l], (\#f!n \geq \text{Max } (\text{set } \#f))]\)
using nth-map by (metis (poly-guards-query, opaque-lifting)) also have ... =
\[ n \cdot n \leftarrow [0..<\text{size } l], (\forall n \in \set{0..<\text{size } l}, f (\forall n) \geq \text{Max} (\set{f l})) \]

using \textit{atLeastLessThan-iff le0 set-up} by \textit{metis(no-types)}

also have \( \ldots = \)
\[ n \cdot n \leftarrow [0..<\text{size } l], f (\forall n) \geq \text{Max} (\set{f l}) \]

using \textit{lm05} by \textit{presburger}

finally show \( \vdash \)thesis by \textit{auto} 

\textbf{qed}

\textbf{lemma} \textit{lm08}: fixes \( f l \)
shows \textit{argmaxList} \( f l = \)
\[ [ \forall n \cdot n \leftarrow [0..<\text{size } l], f (\forall n) \geq \text{Max} (f'(\set{l})) ] \]

unfolding \textit{lm07} \textit{argmaxList-def} by \textit{metis \textit{lm06}}

The theorem expresses that \textit{argmaxList} is the list of arguments greater equal the Max of the list.

\textbf{theorem} \textit{argmaxadequacy}: fixes \textit{f}::\textit{a => (}'b::linorder\textit{)} fixes \textit{l::a list}
shows \textit{argmaxList} \( f l = [ x \leftarrow l. f x \geq \text{Max} (f'(\set{l})) ] \)
\[ (\text{is } \vdash \) lh=-) \]

\textbf{proof} –
\[ \text{let } \vdash y::{'(}b::\text{linorder}\text{)} . y \geq \text{Max} (f'(\set{l})) \]
\[ \text{let } \vdash \text{m}h=[\text{n}th l n \cdot n \leftarrow [0..<\text{size } l], \vdash (f (\text{n}th l n))] \]
\[ \text{let } \vdash rh=[ x \leftarrow (\text{map} (\text{n}th l) [0..<\text{size } l]). \vdash (f x)] \]
\[ \text{have } \vdash lh = \vdash m\text{h using } \text{lm08 by fast} \]
\[ \text{also have } \ldots = \vdash rh \text{ using map-commutes by fast} \]
\[ \text{also have } \ldots = [ x \leftarrow l. \vdash (f x)] \text{ using map-nth by metis} \]
\[ \text{finally show } \vdash \text{thesis by force} \]

\textbf{qed}

\textbf{5} \hspace{1em} \textbf{Additional operators on relations, going beyond Relations.thy, and properties of these operators}

\textbf{theory} \textit{RelationOperators}

\textbf{imports}
SetUtils
HOL-Library.Code-Target-Nat

\textbf{begin}

\textbf{5.1} \hspace{1em} \textbf{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.

\textbf{fun} \textit{eval-rel-or} :: (\textit{a} \times \textit{b}) set \Rightarrow \textit{a} \Rightarrow \textit{b} \Rightarrow \textit{b}

\textbf{where} \textit{eval-rel-or} \textit{R} \textit{a} \textit{z} = (let \textit{im} = \textit{R} "\{\textit{a}\}" \textit{in} if \textit{card} \textit{im} = 1 then \textit{the-elem} \textit{im} else \textit{z})
right-uniqueness of a relation: the image of a trivial set (i.e. an empty or singleton set) under the relation is trivial again. This is the set-theoretical way of characterizing functions, as opposed to \(\Lambda\) functions.

**definition** runiq :: ('a × 'b) set ⇒ bool
  where runiq R = (∀ X. trivial X → 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 × 'b) set ⇒ 'a set ⇒ ('a × 'b) set (infix || 75)
  where R || X = (X × Range R) ∩ R

extensional characterization of the pairs within a restricted relation

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

alternative statement of ?R || ?X = \{ (x, y) | x y . x ∈ ?X ∧ (x, y) ∈ ?R \}
without explicitly naming the pair’s components

**lemma** restrict-ext': R || X = \{ p . fst p ∈ X ∧ p ∈ R \}
  proof –
  have R || X = \{ (x, y) | x y . x ∈ X ∧ (x, y) ∈ R \} by (rule restrict-ext)
  also have . . . = \{ p . fst p ∈ X ∧ p ∈ R \} by force
  finally show ?thesis .
  qed

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

**lemma** restrict-empty: P || {} = {}
  unfolding restrict-def by simp

A restriction is a subrelation of the original relation.

**lemma** restriction-is-subrel: P || X ⊆ P
  using restrict-def by blast

Restricting a relation only has an effect within its domain.

**lemma** restriction-within-domain: P || X = P || (X ∩ (Domain P))
  unfolding restrict-def by fast

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

**lemma** restrict-to-singleton: P || \{ x \} = \{ x \} × (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 × 'b) set ⇒ 'a set ⇒ ('a × '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 /∈ 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) sub-set-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 ⊆ Q
  shows P ++ Q = P ∪ 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 ∪ 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 ∪ 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 = {1} ∨ 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 cardinalityOneTheElemIden
ty 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 ⊆ 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 ←→ (∀ 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 ←→ (∀ x . \(R '' \{x\}\) ⊆ \(\{R ,,, x\}\))
by (metis eval-rel.simps runiq-alt trivial-def)

lemma rightUniquePair:
assumes runiq f
assumes \((x,y)\)∈ 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 ←→ (∀ x y y'. (x, y) ∈ R ∧ (x, y') ∈ R −→ y = y')
unfolding runiq-alt trivial-same by blast

lemma rightUniqueFunctionAfterInverse:
assumes runiq f
shows \((f''(f''^−1''Y))\) ⊆ 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 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 ∈ 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 ←→ (∀ a ∈ Domain R . 3! b . (a, b) ∈ R)
  using runiq-basic by (metis 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 ∈ Domain R
             and  runiq: runiq R
  shows  (x, R ,, x) ∈ 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 ∈ Domain R
  shows  R ,, a ∈ 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: \( x \in \text{Domain} \ R \)
- **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** \( \text{Domain} \ R \cap X \cap Y = \{\} \)
- **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 \( (P^{-1}) \) \( y \notin \text{Range} \ P \)
- **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 \( ?P^{-1} \subseteq P^{-1} \cup \{(y,x)\} \) by blast
  - moreover have ... = \( P^{-1} \cup \{(y,x)\} \) by fast
  - moreover have \( \text{Domain} \ (P^{-1}) \cap \text{Domain} \ \{(y,x)\} = \{\} \) 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 \( ?\text{thesis} \) 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 \( X \to Y = \{ R : \text{Domain} \ R = X \land \text{Range} \ R \subseteq Y \land \text{runiq} \ R \land \text{runiq} \ (R^{-1}) \} \)

The following definition is a constructive (computational) characterization of the set of all injections \( X \to Y \), 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 \( Y - \text{Range} \ R \) ] . R ← injections-alg xs Y ]

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

end

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

theory MiscTools

imports
  HOL-Library.Discrete
  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 "<<")

\(<\<" \) 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 \( f \) \( \text{pair} \) \( = \) \( f ++ \{(\text{fst \ pair}, \text{snd \ pair})\} \)

37
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 ln001:
(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 oth-
ernwise not needed.

definition graph
where graph X f = {(x, f x) | x. x ∈ X}

lemma ln002:
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 \( R \subseteq \text{graph} \ (\text{Domain} \ R \ (\text{toFunction} \ R) \))
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 \( R = \text{graph} \ (\text{Domain} \ R \ (\text{toFunction} \ R) \))
using assms lm002 lm003 by fast

lemma domainOfGraph:
runiq (\text{graph} \ X \ f) \& Domain (\text{graph} \ X \ f) = X
unfolding graph-def
using rightUniqueRestrictedGraph by fast

abbreviation eval-rel2 \((R::(\equiv a \times (\equiv b \ \text{set})) \ \text{set})) \ (x::\equiv a) = \bigcup (R''\{(x\})
notation eval-rel2 \((\text{infix},,,,75))

lemma imageEquivalence:
assumes runiq \((f::((\equiv a \times (\equiv b \ \text{set})) \ \text{set})) \ x \in \text{Domain} \ f\)
shows \(f,.,x = f,.,x\)
using assms Image-runiq-eq-eval cSup-singleton by metis

lemma lm005:
Graph \( f = \text{graph} \ \text{UNIV} \ f \)
unfolding Graph-def graph-def by simp

lemma graphIntersection:
\(\text{graph} \ (X \cap Y) \ f \subseteq (\text{graph} \ X \ f) \mid Y\)
unfolding graph-def
using Int-iff mem-Collect-eq RelationOperators. restrict-ext subrelI by auto

definition runiqs
where \(\text{runiqs} = \{f. \ \text{runiq} \ f\}\)

lemma outsideOutside:
\((P \ \text{outside} \ X) \ \text{outside} \ Y) = P \ \text{outside} \ (X \cup Y)\)
unfolding Outside-def by blast

corollary lm006:
\((P \ \text{outside} \ X) \ \text{outside} \ X) = P \ \text{outside} \ X\)
using outsideOutside by force

lemma lm007:
assumes \((X \cap \text{Domain} \ P) \subseteq \text{Domain} \ Q)\)
shows $P ** Q = (P \text{ outside } X) ** Q$

unfolding paste-def Outside-def using assms by blast

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

corollary outsideUnion:
$R = (R \text{ outside } \{x\}) \cup (\{x\} \times (R '' \{x\}))$
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:
$R = (R \text{ outside } \{x\}) ** (\{x\} \times (R '' \{x\}))$
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:
$(P || (X \cap Y)) \subseteq (P||X)$ & $P \text{ outside } (X \cup Y) \subseteq P \text{ outside } X$
by (metis doubleRestriction le-sup-iff outsideOutside outside-union-restrict subset-refl)

lemma lm016:
$P || X \subseteq (P|| (X \cup Y))$ & $P \text{ outside } X \subseteq P \text{ outside } (X \cap Y)$
using lm015 distrib-sup-le sup-idem le-inf-iff subset-antisym sup-commute
by (metis sup-ge1)

lemma lm017:
$P''(X \cap \text{Domain } P) = P''X$
by blast

lemma cardinalityOneSubset:
assumes card $X = 1$ and $X \subseteq Y$
shows $\text{Union } X \in Y$
using assms cardinalityOneTheElemIdentity by (metis cSup-singleton insert-subset)

lemma cardinalityOneTheElem:
assumes card $X = 1$ and $X \subseteq Y$
shows the-elem $X \in Y$
using assms by (metis (full-types) insert-subset cardinalityOneTheElemIdentity)

lemma lm018:
$(R \text{ outside } X_1) \text{ outside } X_2 = (R \text{ outside } X_2) \text{ outside } X_1$
by (metis outsideOutside sup-commute)

7.2 Ordered relations

lemma lm019:
assumes card $X \geq 1$ and $x, y \in X$
shows $y > \text{Max } X$
using assms by (metis (poly-guards-query) Max-in One-nat-def card-eq-0-iff lessI not-le)

lemma lm020:
assumes finite $X$ and $mx \in X$
shows $f(x) < f(mx)$
using assms not-less by fastforce

lemma lm021:
assumes finite $X$ and $mx \in X$
shows $f(x) \in \{mx\}$
using assms mk-disjoint-insert by force

lemma lm022:
assumes finite $X$ and $mx \in X$
shows $f(X) = \{mx\}$
using assms lm021 by (metis argmax-non-empty-iff equals0D subset-singletonD)

corollary argmaxProperty:
$(\text{finite } X \wedge mx \in X \wedge (\forall aa \in X \setminus \{mx\}. f(aa) < f(mx)) \rightarrow \text{argmax } f(X) = \{mx\}$
using lm022 by metis
corollary lm023:
assembles finite $X \in X \forall x \in X. x \neq mx \rightarrow f x < f mx$
shows $\text{argmax}\ f X = \{mx\}$
using asssms lm022 by (metis Diff-iff insertI1)

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

lemma lm025:
assembles inj-on $f\ X$
shows inj-on $\text{image} f (\text{Pow} X)$
using asssms inj-on-image-eq-iff inj-onI PowD by (metis (mono-tags, lifting))

lemma injectionPowerset:
assembles inj-on $f\ Y\ X \subseteq Y$
shows inj-on $\text{image} f (\text{Pow} X)$
using asssms lm025 by (metis subset-inj-on)

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

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

lemma finestPartUnion:
$X = \bigcup (\text{finestpart} X)$
using finestPart by auto

lemma lm026:
$\text{Union} \circ \text{finestpart} = id$
using finest-part-def finestPartUnion by fastforce

lemma lm027:
inj-on $\text{Union} (\text{finestpart} ' UNIV)$
using lm026 by (metis inj-on-id inj-on-imageI)

lemma nonEqualitySetOfSets:
assembles $X \neq Y$
shows $\{\{x\}|x \in X\} \neq \{\{x\}|x \in Y\}$
using asssms 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 finestpart x) \land (x \in X))\} = \bigcup (\text{finestpart} \cdot X)
by auto

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

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

lemma setOfPairs:
\{(\text{fst pair}, \{ y\})\mid y \in \text{snd pair}\} =
\{(\text{fst pair}) \times \{ \{ y \}\mid y \in \text{snd pair}\}
by fastforce

lemma lm029:
x \in X = (\{ x \} \in \text{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 \text{Pow} X
using assms by blast

lemma lm031:
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

43
shows \(\sum f A = \sum f (A \cap B) + \sum f (A - B)\)

using \textit{assms} by (metis DiffD2 Int-iff Un-Diff-Int Un-commute finite-Un sum.union-inter-neutral)

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

lemma \textit{lm033:}
  assumes \((\text{Domain } P \subseteq \text{Domain } Q)\)
  shows \((P \leftrightarrow Q) = Q\)
  unfolding \textit{paste-def Outside-def} using \textit{assms} by fast

lemma \textit{lm034:}
  assumes \((P \leftrightarrow Q = Q)\)
  shows \((\text{Domain } P \subseteq \text{Domain } Q)\)
  using \textit{assms} \textit{paste-def Outside-def} by blast

lemma \textit{lm035:}
  \((\text{Domain } P \subseteq \text{Domain } Q) = (P \leftrightarrow Q = Q)\)
  using \textit{lm033 lm034} by \textit{metis}

lemma \((P || \text{(Domain } Q)) \leftrightarrow Q = Q\)
  by (metis \textit{Int-lower2 restrictedDomain lm035})

lemma \textit{lm036:}
  \(P || X = P \text{ outside } (\text{Domain } P - X)\)
  using \textit{Outside-def restrict-def} by \textit{fastforce}

lemma \textit{lm037:}
  \((P \text{ outside } X) \subseteq P || ((\text{Domain } P) - X)\)
  using \textit{lm036 lm016} by (metis \textit{Int-commute restrictedDomain outside-reduces-domain})

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

lemma \textit{lm039:}
  \((P \text{ outside } X) \cap (Q || X) = \{\}\)
  using \textit{lm038} by \textit{fast}

lemma \textit{lm040:}
  \((P \text{ outside } (X \cup Y)) \cap (Q || X) = \{\}\) \& \((P \text{ outside } X) \cap (Q || (X \cap Z)) = \{\}\)
using Outside-def restrict-def lm039 lm015 by fast

lemma lm041:
\[ P \text{ outside } X = P \ || \ ((\text{Domain } P) - X) \]
using Outside-def restrict-def lm037 by fast

lemma lm042:
\[ R^{-1}(X - Y) = (R|X)^{-1}(X - Y) \]
using restrict-def by blast

lemma lm043:
assumes \[ \bigcup_{XX} X \subseteq XX \ x \in XX \ x \neq {} \]
shows \[ x \cap X \neq {} \]
using assms by blast

lemma setVsList:
assumes \[ \forall l \in \text{set } L1. \text{ set } L2 = f2 \ (\text{set } l) \ N \]
shows \[ \text{set } \ [\text{set } \ (\text{g2 l N}). l \leftarrow (\text{g1 G})] = \{f2 P N | P. P \in \text{set } (\text{map set } L1)\} \]
using assms by auto

lemma lm045:
\[ (\forall l \in \text{set } (g1 G). \text{ set } \ (g2 l N) = f2 \ (\text{set } l) \ N) \rightarrow \]
\[ \{f2 P N | P. P \in \text{set } (\text{map set } (g1 G))\} = \text{set } \ (\text{set } \ (g2 l N)). l \leftarrow \ g1 G \]
by auto

lemma lm046:
assumes \[ X \cap Y = {} \]
shows \[ R^{\prime\prime} X = (R \text{ outside } Y)^{\prime\prime} X \]
using assms Outside-def Image-def by blast

lemma lm047:
assumes \[ (\text{Range } P) \cap (\text{Range } Q) = {} \]
\[ \text{ runiq } (P^\sim-1) \text{ runiq } (Q^\sim-1) \]
shows \[ \text{ runiq } ((P \cup Q)^\sim-1) \]
using assms by (metis Domain-converse converse-Un disj-Un-runiq)

lemma lm048:
assumes \[ (\text{Range } P) \cap (\text{Range } Q) = {} \]
\[ \text{ runiq } (P^\sim-1) \text{ runiq } (Q^\sim-1) \]
shows \[ \text{ runiq } ((P +* Q)^\sim-1) \]
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 \iff a \in \text{Domain } R
using assms card-Suc-eq One-nat-def
by (metis Image-within-domain Suc-neq-Zero assms rightUniqueSetCardinality)

lemma lm050:
inj (\lambda a. ((\text{fst a}, \text{fst (snd a)}), \text{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 \neq \{}
shows Max (f\':A) \in f\':A
using assms by (metis Max-in finite-imageI image-is-empty)

lemma lm053:
argmax f A \subseteq f -\{\text{Max (f \': A)}\}
by force

lemma lm054:
argmax f A = A \cap \{ x . f x = \text{Max (f \': A)} \}
by auto

lemma lm055:
(x \in \text{argmax} f X) = (x \in X \& f x = \text{Max (f \': X)})
using argmax.simps mem-Collect-eq by (metis (mono-tags, lifting))

lemma rangeEmpty:
Range -\{} = \{\{}\}
by auto

lemma finitePairSecondRange:
(\forall \text{pair} \in R. \text{finite (snd pair)}) = (\forall y \in \text{Range } R. \text{finite } y)
by fastforce

lemma lm056:
\text{fst }\text{flip pair} = \text{snd }\text{flip pair} \& \text{snd pair} = \text{fst (flip pair)}
by force

lemma lm057:
\text{fst pair} = \text{snd (flip pair)} \& \text{snd pair} = \text{fst (flip pair)}
unfolding flip-def by simp

lemma flip-flip2:
  flip \circ flip = id
  using flip-flip by fastforce

lemma lm058:
  fst = (snd\circ flip)
  using lm057 by fastforce

lemma lm059:
  snd = (fst\circ flip)
  using lm057 by fastforce

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

lemma lm062:
  inj-on fst P = inj-on snd (P^\sim 1)
  using lm060 flip-conv by (metis converse-converse inj-on-imageI lm059)

lemma sumPairsInverse:
  assumes runiq (P^\sim 1)
  shows sum (f \circ snd) P = sum f (Range P)
  using assms lm062 converse-converse rightUniqueInjectiveOnFirst rightUniqueInjectiveOnFirst
    sum.reindex snd-eq-Range
  by metis

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

lemma lm063:
  assumes inj-on g X
  shows sum f (g'X) = sum (f \circ g) X
  using assms by (metis sum.reindex)

lemma functionOnFirstEqualsSecond:
  assumes runiq R z \in 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 ∩ 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 ∈ Domain (f||X)
shows (f||X)`{x} = f``{x}
using assms doubleRestriction restrictedRange Int-empty-right Int-iif
Int-insert-right-if1 restrictedDomain
by metis

lemma lm069:
assumes x ∈ X ∩ Domain f runiq (f||X)
shows (f||X)`x = f``x
using assms doubleRestriction restrictedRange Int-empty-right Int-iif Int-insert-right-if1
eval-rel.simps
by metis

lemma lm070:
assumes runiq (f||X)
shows sum (toFunction (f||X)) (X ∩ Domain f) = sum (toFunction f) (X ∩ Domain f)
using assms sum.cong lm069 toFunction-def by metis

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

corollary sumRestrictedOnFunction:
assumes runiq (f||X)
shows sum (toFunction (f||X)) (X ∩ 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 o 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 o finestpart = finite
using finiteFinestpart by fastforce

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

corollary lm073:
assumes x ∈ 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 raniq a
shows \{ (x, \{y\}) | x, y ∈ \bigcup (a"\{x\}) & x ∈ Domain a \} = 
\{ (x, \{y\}) | x, y ∈ a\times & x ∈ Domain a \}
using assms Image-runiq-eq-eval
by (metis (lifting, no-types) cSup-singleton)

lemma lm076:
assumes raniq 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 fast

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 
  shows \( X \subseteq Y \)
  using assms by (metis Int-lower1 Int-lower2 card-seteq order-refl)

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

lemma lm085:
  assumes \( P \) xx 
  shows \( \{(x, f x) | x. P x\}, \text{xx} = f \text{xx} \)
proof - 
  let \(?F\) = \(\{(x, f x) | x. P x\}\) 
  let \(?X\) = \(?F\)\{"xx\} 
  have \(?X\) = \{f xx\} using Image-def assms by blast 
  thus \(\text{thesis} \) by fastforce
qed

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

lemma lm086:
  \( \text{Graph } f, xx = f xx \)
using UNIV-I graphEqImage lm005 by (metis (no-types))

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

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

lemma lm089:
  \( \text{Range}(f \text{ outside } X) \supseteq (\text{Range } f) - (f''X) \)
using Outside-def by blast

lemma lm090:
  assumes raniq \( P \)
  shows \( (P^{-1}' \cap ((\text{Range } P) - Y)) \cap ((P^{-1})''Y) = {} \)
using assms rightUniqueFunctionAfterInverse by blast
lemma lm091:
  assumes runiq (P⁻¹)
  shows (P''((Domain P) − X)) ∩ (P''X) = {} 
  using assms rightUniqueFunctionAfterInverse by fast

lemma lm092:
  assumes runiq f runiq (f⁻¹)
  shows Range(f outside X) ⊆ (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⁻¹)
  shows Range(f outside X) = (Range f)−(f''X)
  using assms lm089 lm092 by (metis order-class.order.antisym)

lemma unionIntersectionEmpty:
  (∀ x∈X. ∀ y∈Y. x∩y = {}) = ((∪X)∩(∪Y)={}) 
  by blast

lemma setEqualityAsDifference:
  {x}−{y} = {} = (x = y) 
  by auto

lemma lm093:
  assumes R ≠ {} Domain R ∩ X ≠ {} 
  shows R''X ≠ {} 
  using assms by blast

lemma lm095:
  R ⊆ (Domain R) × (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 (⋃XX) 
  shows ∀ X ∈ XX. finite X 
  using assms by (metis Union-upper finite-subset)

lemma lm096:
  assumes runiq f X ⊆ (f⁻¹)''Y 
  shows f''X ⊆ Y 
  using assms rightUniqueFunctionAfterInverse by (metis Image-mono order-refl subset-trans)

lemma lm097:
assumes $y \in f^\prime \{x\}$ 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
$Chi$ $X$ $Y$ == $(Y \times \{0::\text{nat}\}) + \ast (X \times \{1\})$
notation $Chi$ (infix $\langle\mid\rangle$ 80)

abbreviation
$chii$ $X$ $Y$ == toFunction ($X$ $\langle\mid\rangle$ $Y$)
notation $chii$ (infix $\langle\mid\rangle$ 80)

abbreviation
$chi$ $X$ == indicator $X$

lemma lm098:
runiq ($X \langle\mid\rangle$ $Y$)
by (rule lm014)

lemma lm099:
assumes $x \in X$
shows $1 \in (X \langle\mid\rangle$ $Y$) $\langle\mid\rangle$ $\{x\}$
using assms toFunction-def paste-def Outside-def runiq-def lm014 by blast

lemma lm100:
assumes $x \in Y - X$
shows $0 \in (X \langle\mid\rangle$ $Y$) $\langle\mid\rangle$ $\{x\}$
using assms toFunction-def paste-def Outside-def runiq-def lm014 by blast

lemma lm101:
assumes $x \in X \cup Y$
shows ($X \langle\mid\rangle$ $Y$), $x = chi$ $X$ $x$ (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 ($X \langle\mid\rangle$ $Y$) $x = chi$ $X$ $x$
using assms toFunction-def lm101 by metis

corollary lm103:
sum ($X \langle\mid\rangle$ $Y$) ($X \cup Y$) = sum (chi $X$) ($X \cup Y$)
using lm102 sum.cong by metis

53
corollary lm104:
assumes $\forall x \in X. \ f \ x = g \ x$
shows $\text{sum } f \ X = \text{sum } g \ X$
using assms by (metis (poly-guards-query) sum.cong)

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

corollary lm106:
assumes $Z \subseteq X \cup Y$
shows $\text{sum } (X < | Y) \ Z = \text{sum } (\chi X) \ Z$
proof
have $\forall x \in Z. (X < | Y) \ x = (\chi X) \ x$ using assms lm102 in-mono by metis
thus ?thesis using lm104 by blast
qed

corollary lm107:
$\text{sum } (\chi X) \ (Z - X) = 0$
by simp

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

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

corollary lm110:
assumes $Z \subseteq X \cup Y$ finite $Z$
shows $\text{sum } (X < | Y) \ Z = \text{sum } (X < | Y) \ (Z \cap X)$
using assms lm078 lm108 comm-monoid-add-class.add-0 by metis

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

corollary lm112:
assumes $Z \subseteq X \cup Y$ finite $Z$
shows $\text{sum } (X < | Y) \ Z = \text{card } (Z \cap X)$
using assms lm111 by (metis lm106 sum-indicator-eq-card)
corollary subsetCardinality:
  assumes $Z \subseteq X \cup Y$ finite $Z$
  shows $(\sum (X <| Y) X) - (\sum (X <| Y) Z) = \text{card } X - \text{card } (Z \cap X)$
  using assms lm112 by (metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite)

corollary differenceSumVsCardinality:
  assumes $Z \subseteq X \cup Y$ finite $Z$
  shows $(\int (\sum (X <| Y) X) - (\sum (X <| Y) Z)) = (\int (\text{card } X) - (\int (\text{card } (Z \cap X)))$
  using assms lm112 by (metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite)

lemma lm113:
  $(\int (n::\text{nat}) = \text{real } n)$
  by simp

corollary differenceSumVsCardinalityReal:
  assumes $Z \subseteq X \cup Y$ finite $Z$
  shows $(\text{real } (\sum (X <| Y) X) - (\sum (X <| Y) Z)) = (\text{real } (\text{card } X) - (\text{real } (\text{card } (Z \cap X)))$
  using assms lm112 by (metis Int-absorb2 Un-upper1 card.infinite equalityE sum.infinite)

7.4 Lists

lemma lm114:
  assumes $\exists n \in \{0..<\text{size } l\}. \ P \ (l!n)$
  shows $[n. n \leftarrow \{0..<\text{size } l\}, \ P \ (l!n)] \neq []$
  using assms by auto

lemma lm115:
  assumes $ll \in \text{set } (l::\text{a list})$
  shows $\exists n \in (\text{nth } l) - ' (\text{set } l). \ ll=\text{l!n}$
  using assms(1) by (metis in-set-conv-nth vimageI2)

lemma lm116:
  assumes $ll \in \text{set } (l::\text{a list})$
  shows $\exists n. \ ll=\text{l!n} \& \ n < \text{size } l \& \ n \geq 0$
  using assms in-set-conv-nth by (metis le0)

lemma lm117:
  assumes $P - ' \{\text{True}\} \cap \text{set } l \neq \{\}$
  shows $\exists n \in \{0..<\text{size } l\}. \ P \ (l!n)$

55
using assms \textit{lm116} by fastforce

\textbf{lemma nonEmptyListFiltered:}
\begin{itemize}
  \item \textbf{assumes} $P \not\vdash \{\text{True}\} \cap \text{set } l \neq \{\}
  \item \textbf{shows} $[n. n \leftarrow [0..<\text{size } l], P (\text{l!} n)] \neq []$
\end{itemize}
\textbf{using} assms \textit{filterpositions2-def lm117 lm114} \textbf{by} \textit{metis}

\textbf{lemma \textit{lm118:}}
\begin{itemize}
  \item $(\text{nth } l \cdot \text{ set } ([n. n \leftarrow [0..<\text{size } l], \forall x. x \in X \Rightarrow (\text{l!} n)]) \subseteq X \cap \text{set } l$
\end{itemize}
\textbf{by} \textit{force}

\textbf{corollary \textit{lm119:}}
\begin{itemize}
  \item $(\text{nth } l \cdot \text{ set } (\text{filterpositions2 } \forall x. x \in X \Rightarrow (\text{l!} n))) \subseteq X \cap \text{set } l$
\end{itemize}
\textbf{unfolding} \textit{filterpositions2-def} \textbf{using} \textit{lm118} \textbf{by} \textit{fast}

\textbf{lemma \textit{lm120:}}
\begin{itemize}
  \item $(n \in \{0..<N\}) = ((n::\text{nat}) < N)$
\end{itemize}
\textbf{using} \textit{atLeastLessThan-def atLeast0LessThan lessThan-iff} \textbf{by} \textit{metis}

\textbf{lemma \textit{lm121:}}
\begin{itemize}
  \item \textbf{assumes} $X \subseteq \{0..<\text{size list}\}$
  \item \textbf{shows} $(\text{nth } l) X \subseteq \text{set } l$
\end{itemize}
\textbf{using} assms \textit{atLeastLessThan-def atLeast0LessThan lessThan-iff} \textbf{by} \textit{auto}

\textbf{lemma \textit{lm122:}}
\begin{itemize}
  \item $\text{set } ([n. n \leftarrow [0..<\text{size } l], P (\text{l!} n)]) \subseteq \{0..<\text{size } l\}$
\end{itemize}
\textbf{by} \textit{force}

\textbf{lemma \textit{lm123:}}
\begin{itemize}
  \item $\text{set } (\text{filterpositions2 } \text{ pre } l) \subseteq \{0..<\text{size list}\}$
\end{itemize}
\textbf{using} \textit{filterpositions2-def lm122} \textbf{by} \textit{metis}

\textbf{7.5 Computing all the permutations of a list}

\textbf{abbreviation}\hspace{1em} \textit{rotateLeft} \hspace{0.5em}== \hspace{0.5em} \textit{rotate}

\textbf{abbreviation}
\begin{itemize}
  \item $\text{rotateRight } n \ l \hspace{0.5em}== \hspace{0.5em} \text{rotateLeft } (\text{size } l - (n \ \text{mod } (\text{size } l))) \ l$
\end{itemize}

\textbf{abbreviation}
\begin{itemize}
  \item $\text{insertAt } x \ l \ n \hspace{0.5em}== \hspace{0.5em} \text{rotateRight } n \ (x \# (\text{rotateLeft } n \ l))$
\end{itemize}
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:
  ∀. set (perm2 [] n) = set []
  by simp

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

corollary permutationInvariance:
  ∀. set (perm2 (l::'a list) n) = set l
proof (induct l)
  let ?P = %l::('a list). (∀. set (perm2 l n) = set l)
show \( \forall P \) using \( \text{lm127} \) by force
fix \( x \) fix \( l \)
assume \( \forall P \, l \)
show \( \forall P \ (x \# l) \) by force
qed

corollary \( \text{takeAllPermutation} \):
set (\text{perm2} \, (\text{takeAll} \, (\forall x. \in X)) \, \, l) \, n) \subseteq X \cap \text{set} \, l
using \( \text{listIntersectionWithSet permutationInvariance} \) by \( \text{metis} \)

abbreviation \( \text{subList} \, l \, xl \) \( \equiv \) \( \text{map} \, (\text{nth} \, l) \, (\text{takeAll} \, (\forall x. \, x \leq \text{size} \, l)) \, xl \)

7.6 A more computable version of \( \text{toFunction} \).

abbreviation \( \text{toFunctionWithFallback} \, R \, \text{fallback} \) \( \equiv \) \( (\forall x. \, (R'' \, \{x\} = \{R, x\}) \, \text{then} \, (R, x) \, \text{else} \, \text{fallback}) \)
notation \( \text{toFunctionWithFallback} \) (infix \( \text{Else} \, 75 \))

abbreviation \( \text{sum'} \) where
\( \text{sum'} \, R \, X \equiv \text{sum} \, (R \, \text{Else} \, 0) \, X \)

lemma \( \text{lm129} \):
assumes \( \text{runiq} \, f \, x \in \text{Domain} \, f \)
shows \( (f \, \text{Else} \, 0) \, x = (\text{toFunction} \, f) \, x \)
using \( \text{assms} \) by (\( \text{metis} \, \text{Image-runiq-eq-eval} \, \text{toFunction-def} \))

lemma \( \text{lm130} \):
assumes \( \text{runiq} \, f \)
shows \( \text{sum} \, (f \, \text{Else} \, 0) \, (X \cap (\text{Domain} \, f)) = \text{sum} \, (\text{toFunction} \, f) \, (X \cap (\text{Domain} \, f)) \)
using \( \text{assms} \, \text{sum.cong} \, \text{lm129} \) by fastforce

lemma \( \text{lm131} \):
assumes \( Y \subseteq f^{-1}\{0\} \)
shows \( \text{sum} \, f \, Y = 0 \)
using \( \text{assms} \) by (\( \text{metis} \, \text{rev-subsetD} \, \text{sum.neutral} \, \text{vimage-singleton-eq} \))

lemma \( \text{lm132} \):
assumes \( Y \subseteq f^{-1}\{0\} \) finite \( X \)
shows \( \text{sum} \, f \, X = \text{sum} \, f \, (X - Y) \)
using \( \text{Int-lower2} \, \text{add.comm-neutral} \, \text{assms(1)} \, \text{assms(2)} \, \text{lm078} \, \text{lm131} \, \text{order-trans} \)
by (\( \text{metis} \, \text{(no-types)} \))

lemma \( \text{lm133} \):
\( -(\text{Domain} \, f) \subseteq (f \, \text{Else} \, 0) - \{0\} \)
corollary lm134:
  assumes finite X
  shows \( \sum (f \text{ Else } 0) X = \sum (f \text{ Else } 0) (X \cap \text{Domain } f) \)
proof –
  have \( X \cap \text{Domain } f = X - (-\text{Domain } f) \) by simp
  thus \(?thesis using assms lm133 lm132 by fastforce\)
qed

corollary lm135:
  assumes finite X
  shows \( \sum (f \text{ Else } 0) (X \cap \text{Domain } f) = \sum (f \text{ Else } 0) X \)
(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 \( \sum (f \text{ Else } 0) X = \sum (\text{toFunction } f) (X \cap \text{Domain } f) \)
(is \(?L=?R\))
proof –
  have \(?R = \sum (f \text{ Else } 0) (X \cap \text{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:
  \( \sum (f \text{ Else } 0) X = \sum' f X \)
by fast

corollary lm138:
  assumes finite X runiq f
  shows \( \sum (\text{toFunction } f) (X \cap \text{Domain } f) = \sum' f X \)
using assms lm137 lm136 by fastforce

lemma lm139:
  \( \text{argmax} (\sum' b) = (\text{argmax} \circ \sum') b \)
by simp

lemma domainConstant:
  Domain \((Y \times \{0::\text{nat}\}) = Y \) & Domain \((X \times \{1\}) = X \)
by blast

lemma domainCharacteristicFunction:
  Domain \((X <|| Y) = X \cup Y \)
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 ln140:
assumes runiq R runiq (R^\(-1\))
shows \( (R''A) \cap (R''B) = R''(A\cap B) \)
using assms by (metis image-cong)

lemma intersectionEmptyRelationIntersectionEmpty:
assumes runiq (R^\(-1\)) runiq R X1 \( \cap \) X2 = {}
shows \( (R''X1) \cap (R''X2) = {} \)
using assms by (metis image-cong)

lemma lm141:
assumes runiq f trivial Y
shows trivial \( (f''(f^\(-1''Y)) \)
using assms by (metis image-cong)

lemma lm142:
assumes trivial X
shows card \( (Pow X)\subset \{1,2\} \)
using trivial-empty-or-singleton card-Pow-empty assms trivial-implies-finite
by (metis image-cong)

lemma lm143:
assumes card \( (Pow A) = 1 \)
shows A = {}
using assms by (metis image-cong)

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

corollary ln145:
\( (finite A) = (\card (Pow A) \neq 0) \)
using ln144 by metis

lemma lm146:
assumes card \( (Pow A) \neq 0 \)
shows \( A=\text{Discrete.log} \ (\card (Pow A)) \)
using assms log-exp card-Pow by (metis image-cong)
lemma log-2 [simp]:
Discrete.log 2 = 1
using log-exp [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. (w::'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

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 \( \text{alpha}\ N\ G\ b\ n \equiv \text{Max} \left( (\text{sum} \ b) \cdot (\text{allAllocations} \ (N - \{n\}) \ G) \right) \)

abbreviation \( \text{alphaAlg}\ N\ G\ b\ n \equiv \text{Max} \left( (\text{proceeds} \ b) \cdot (\text{set} \ (\text{allAllocationsAlg}) \ (N - \{n\}) \ (G:-\ list)) \right) \)

abbreviation \( \text{remainingValueRel}\ N\ G\ t\ b\ n \equiv \text{sum} \ b \ (\text{winningAllocationRel} \ N \ G \ t \ b \ \text{is n}) \)

abbreviation \( \text{remainingValueAlg}\ N\ G\ t\ b\ n \equiv \text{proceeds} \ b \ (\text{winningAllocationAlg} \ N \ G \ t \ b \ \text{is n}) \)

abbreviation \( \text{paymentsRel}\ N\ G\ t \equiv (\text{alpha}\ N\ G) - (\text{remainingValueRel} \ N \ G \ t) \)

definition \( \text{paymentsAlg}\ N\ G\ t \equiv (\text{alphaAlg}\ N\ G) - (\text{remainingValueAlg} \ N \ G \ t) \)

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 \( \text{lm001} \):
  assumes \( \text{Y} \in \text{set} \ (\text{all-partitions-alg} \ X) \)
  shows \( \text{distinct} \ \text{Y} \)
  using \( \text{assms distinct-sorted-list-of-set all-partitions-alg-def all-partitions-equivalence'} \)
  by \( \text{metis} \)

lemma \( \text{lm002} \):
  assumes \( \text{finite} \ G \)
  shows \( \text{all-partitions} \ G \ = \ \text{set} \ (\text{set} \ (\text{all-partitions-alg} \ G)) \)
9.2 Definitions of various subsets of UNIV.

abbreviation \( \text{isChoice } R == \forall x. R''\{x\} \subseteq x \)
abbreviation \( \text{partitionsUniverse} == \{X. \text{is-non-overlapping } X\} \)

**Lemma** partitionsUniverse \( \subseteq \text{Pow } \text{UNIV} \)

by simp

abbreviation \( \text{partitionValuedUniverse} == \bigcup P \in \text{partitionsUniverse}. \text{Pow } (\text{UNIV} \times P) \)

**Lemma** partitionValuedUniverse \( \subseteq \text{Pow } (\text{UNIV} \times (\text{Pow } \text{UNIV})) \)

by simp

abbreviation \( \text{injectionsUniverse} == \{R. (\text{runiq } R) \& (\text{runiq } (R\sim-1))\} \)

abbreviation \( \text{allocationsUniverse} == \text{injectionsUniverse} \cap \text{partitionValuedUniverse} \)

abbreviation \( \text{totalRels } X Y == \{R. \text{Domain } R = X \& \text{Range } R \subseteq Y\} \)

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 \( \subseteq \text{injectionsUniverse} \)

using injections-def by fast

**Lemma** lm007: injections X Y \( \subseteq \text{injectionsUniverse} \)

using injections-def by blast

**Lemma** lm008: injections X Y = totalRels X Y \( \cap \text{injectionsUniverse} \)

using injections-def by (simp add: Collect-conj-eq Int-assoc)

**Lemma** allocationInverseRangeDomainProperty:

assumes \( a \in \text{allAllocations } N G \)

shows \( a\sim-1 \in \text{injections } (\text{Range } a) \)
(Range a) partitions G &
Domain a ⊆ N

unfolding injections-def using assms all-partitions-def injections-def by fast-force

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 &
raniq (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 ←→
raniq a & runiq (a⁻¹) & (Domain a) partitions G & Range a ⊆ N

67
using allocationRightUniqueRangeDomain lm010 by blast

lemma lm011:
possible-allocations-rel' G N \subseteq injectionsUniverse
using injections-def by force

lemma lm012:
possible-allocations-rel G N \subseteq \{ a. (Range a) \subseteq N \& (Domain a) \in 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=\bigcup \{ injections Y B \mid Y \in 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 \subseteq
injectionsUniverse \cap \{ a. Range a \subseteq N \& Domain a \in all-partitions G \}
using lm012 lm011 injections-def by fastforce

lemma lm017:
possible-allocations-rel G N \supseteq
injectionsUniverse \cap \{ a. Domain a \in all-partitions G \& Range a \subseteq N \}
using injections-def by auto

lemma lm018:
possible-allocations-rel G N =
injectionsUniverse \cap \{ a. Domain a \in all-partitions G \& Range a \subseteq N \}
using lm016 lm017 by blast

lemma lm019:
converse ' injectionsUniverse = injectionsUniverse
by auto
lemma lm020:

\[
\text{converse}'(A \cap B) = (\text{converse}'A) \cap (\text{converse}'B)
\]

by force

lemma allocationInjectionsUniverseProperty:

\[
\text{allAllocations } N G =
\]

\[
\text{injectionsUniverse } \cap \{ \text{a. Domain } a \subseteq N \ & \ \text{Range } a \in \text{all-partitions } G \}
\]

proof –

let \(?A=\text{possible-allocations-rel } G N\)

let \(?c=\text{converse}\)

let \(?I=\text{injectionsUniverse}\)

let \(?P=\text{all-partitions } G\)

let \(?d=\text{Domain}\)

let \(?r=\text{Range}\)

have \(?c'?A = (\text{?c}'I) \cap (\text{?c}'\{ \text{a. } ?r \ a \subseteq N \ & \ ?d \ a \in ?P \})\) using lm018 by fastforce

moreover have \(...) using \text{lm018} by fastforce

moreover have \(...) using \text{lm019} by metis

ultimately show \text{thesis} by presburger

qed

lemma lm021:

\[
\text{allAllocations } N G \subseteq \text{injectionsUniverse}
\]

using allocationInjectionsUniverseProperty by fast

lemma lm022:

\[
\text{allAllocations } N G \subseteq \text{partitionValuedUniverse}
\]

using allocationInverseRangeDomainProperty is-partition-of-def is-non-overlapping-def

by auto blast

corollary allAllocationsUniverse:

\[
\text{allAllocations } N G \subseteq \text{allocationsUniverse}
\]

using lm021 lm022 by (metis (lifting, mono-tags) inf.bounded-iff)

corollary possibleAllocationsRelCharacterization:

\[
\text{a } \in \text{allAllocations } N G =
\]

\[
(\text{a } \in \text{injectionsUniverse } \& \ \text{Domain } a \subseteq N \ & \ \text{Range } a \in \text{all-partitions } G)
\]

using allocationInjectionsUniverseProperty Int-Collect Int-iff by (metis (lifting))

corollary lm023:

assumes \(a \in \text{allAllocations } N1 G\)

shows \(a \in \text{allAllocations } (N1 \cup N2) G\)

proof –

have Domain \(a \subseteq N1 \cup N2\) using assms(1) possibleAllocationsRelCharacterization

by (metis le-supI1)
moreover have \( a \in \text{injectionsUniverse} \& \text{Range } a \in \text{all-partitions } G \)

using assms possibleAllocationsRelCharacterization by blast

ultimately show \(?\text{thesis using possibleAllocationsRelCharacterization by blast}\)

qed

corollary \text{lm024}: allAllocations \( N1 \ G \subseteq \text{allAllocations } (N1 \cup N2) \ G \)

using \text{lm023 by (metis subsetI)}

lemma \text{lm025}:

assumes \((\bigcup P1) \cap (\bigcup P2) = \{\})

\text{is-non-overlapping } P1 \text{ is-non-overlapping } P2

\text{X} \in P1 \cup P2 \text{ Y} \in P1 \cup P2 \text{ X} \cap \text{ Y} \neq \{\}

shows \((\text{X} = \text{Y})\)

unfolding \text{is-non-overlapping-def} using assms is-non-overlapping-def by fast

lemma \text{lm026}:

assumes \((\bigcup P1) \cap (\bigcup P2) = \{\})

\text{is-non-overlapping } P1

\text{is-non-overlapping } P2

\text{X} \in P1 \cup P2

\text{Y} \in P1 \cup P2

(\text{X} = \text{Y})

shows \(\text{X} \cap \text{Y} \neq \{\})

unfolding \text{is-non-overlapping-def} using assms is-non-overlapping-def by fast

lemma \text{lm027}:

assumes \((\bigcup P1) \cap (\bigcup P2) = \{\})

\text{is-non-overlapping } P1

\text{is-non-overlapping } P2

shows \text{is-non-overlapping } (P1 \cup P2)

unfolding \text{is-non-overlapping-def} using assms \text{lm025 \ lm026 bymetis}

lemma \text{lm028}:

Range \( Q \cup (\text{Range } (P \text{ outside } (\text{Domain } Q))) = \text{Range } (P ++ Q) \)

by (simp add: paste-def Range-Un-eq Un-commute)

lemma \text{lm029}:

assumes \(a1 \in \text{injectionsUniverse}\)

\(a2 \in \text{injectionsUniverse}\)

\((\text{Range } a1) \cap (\text{Range } a2) = \{\})

\((\text{Domain } a1) \cap (\text{Domain } a2) = \{\})

shows \(a1 \cup a2 \in \text{injectionsUniverse}\)

using assms disj-Un-runiq

by (metis (no-types) Domain-converse converse-Un mem-Collect-eq)

lemma \text{nonOverlapping}:

assumes \(R \in \text{partitionValuedUniverse}\)
shows \( is\text{-}non\text{-}overlapping \) (Range \( R \))

proof –

obtain \( P \) where
\[
0: P \in \text{partitionsUniverse} \& R \subseteq \text{UNIV} \times P \text{ using } \text{assms by } \text{blast}
\]

have Range \( R \subseteq P \) using \( 0 \) by fast

then show \( \text{thesis using } 0 \text{ mem-Collect-eq subset-is-non-overlapping by (metis)} \)

qed

lemma allocationUnion:

assumes \( a1 \in \text{allocationsUniverse} \)
\( a2 \in \text{allocationsUniverse} \)

\[
(\bigcup (\text{Range } a1)) \cap (\bigcup (\text{Range } a2)) = \{\}
\]

\[
(\text{Domain } a1) \cap (\text{Domain } a2) = \{\}
\]

shows \( a1 \cup a2 \in \text{allocationsUniverse} \)

proof –

let \( ?a=a1 \cup a2 \)

let \( ?b1=a1^{-1} \)

let \( ?b2=a2^{-1} \)

let \( ?r=\text{Range} \)

let \( ?d=\text{Domain} \)

let \( ?I=\text{injectionsUniverse} \)

let \( ?P=\text{partitionsUniverse} \)

let \( ?PV=\text{partitionValuedUniverse} \)

let \( ?u=\text{runiq} \)

let \( ?b=\text{?a}^{-1} \)

let \( ?p=\text{is-non-overlapping} \)

have \( ?p (\?r a1) & ?p (\?r a2) \text{ using } \text{assms nonOverlapping by } \text{blast} \) then

moreover have \( ?p (\?r a1 \cup \?r a2) \text{ using } \text{assms by (metis lm027)} \)

then moreover have \( (?r a1 \cup ?r a2) \in ?P \) by simp

moreover have \( \?r \?a = (\?r a1 \cup \?r a2) \text{ using } \text{assms by fast} \)

ultimately moreover have \( ?p (\?r \?a) \text{ using lm027 assms by fastforce} \)

then moreover have \( ?a \in ?PV \text{ using } \text{assms by fast} \)

moreover have \( \?r a1 \cap (?r a2) \subseteq \text{Pow} (\bigcup (\?r a1) \cap (\bigcup (\?r a2))) \text{ by auto} \)

ultimately moreover have \( \{\} \notin (?r a1) \& \{\} \notin (?r a2) \)

using is-non-overlapping-def by (metis Int-empty-leq)

ultimately moreover have \( \?r a1 \cap (?r a2) = \{\} \)

using assms nonOverlapping is-non-overlapping-def by auto

ultimately moreover have \( \?a \in ?I \text{ using lm029 assms by fastforce} \)

ultimately show \( \text{thesis by blast} \)

qed

lemma lm030:

assumes \( a \in \text{injectionsUniverse} \)

shows \( a - b \in \text{injectionsUniverse} \)

using \( \text{assms} \)

by (metis (lifting) Diff-subset converse-mono mem-Collect-eq subrel-runiq)
lemma lm031:
\{ a. Domain a \subseteq N \ & \ Range a \in all-partitions G \} = 
(Domain \setminus \{\text{Pow} N\}) \cap (Range \setminus \{\text{all-partitions} G\})
by fastforce

lemma lm032:
allAllocations N G =
injectionsUniverse \cap ((Range \setminus \{\text{all-partitions} G\}) \cap (Domain \setminus \{\text{Pow} N\}))
using allocationInjectionsUniverseProperty lm031 by (metis (no-types) Int-commute)

corollary lm033:
allAllocations N G =
injectionsUniverse \cap (Range \setminus \{\text{all-partitions} G\}) \cap (Domain \setminus \{\text{Pow} N\})
using lm032 Int-assoc by (metis)

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

lemma lm035:
assumes a^{-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^{-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 possibleAllocationsRelCharacterization
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 possssibleAllocationsRelCharacterization 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 allocationInjectionsUniverseed Property
    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:
\[(x, X) \in \text{injectionsUniverse}\]

unfolding \text{runiq-basic} using \text{runiq-singleton-rel} by \text{blast}

**Lemma** allocationUniverseProperty:
\[(\{x, X\} - \{(x, \{\}\}\}) \in \text{allocationsUniverse}\]
using \text{lm042} singlePairInInjectionsUniverse \text{lm030} Int-iff by (\text{metis} (\text{no-types}))

**Lemma** \text{lm043}:
\text{assumes} \text{is-non-overlapping PP is-non-overlapping (Union PP)}
\text{shows} \text{is-non-overlapping (Union \ ' PP)}
\text{proof} –
\text{let} \ ?p=\text{is-non-overlapping}
\text{let} \ ?U=\text{Union}
\text{let} \ ?P2=\ ?U PP
\text{let} \ ?P1=\ ?U \ ' PP
\text{have} 0: \forall X \in ?P1. \forall Y \in ?P1. (X \cap Y = \{\} \rightarrow X \neq Y)
using \text{assms is-non-overlapping-def} \text{Int-absorb Int-empty-left UnionI Union-disjoint}
\text{ex-in-conv} \text{imageE}
\text{by} (\text{metis} (\text{opaque-lifting, no-types}))
\{
\text{fix} X Y
\text{assume}
1: X \in ?P1 & Y \in ?P1 & X \neq Y
\text{then obtain} XX YY
\text{where}
2: X = ?U XX & Y = ?U YY & XX \in PP & YY \in PP by \text{blast}
\text{then have} XX \subseteq \text{Union PP} & YY \subseteq \text{Union PP} & XX \cap YY = \{\}
using 1 \text{ is-non-overlapping-def \text{assms}(1) \text{Sup-upper} by \text{metis}}
\text{then moreover have} \forall x \in XX. \forall y \in YY. x \cap y = \{\} \text{ using \text{assms}(2) \text{is-non-overlapping-def}}
\text{by} (\text{metis \text{IntI empty-iff subsetCE}})
\text{ultimately have} X \cap Y = \{\} \text{ using \text{assms} 0 1 2 \text{ is-non-overlapping-def} by \text{auto}}
\}
\text{then show} \ ?\text{thesis using} 0 \text{ is-non-overlapping-def by \text{metis}}
\text{qed}

**Lemma** \text{lm044}:
\text{assumes} a \in \text{allocationsUniverse}
\text{shows} \ (a - ((X \cup \{i\}) \times (\text{Range a}))) \cup \{(\{i\}, \bigcup (a''(X \cup \{i\})) - \{(i, \{\}\})\}) \in \text{allocationsUniverse} \&
\quad \bigcup (\text{Range (a - ((X \cup \{i\}) \times (\text{Range a}))) \cup \{(\{i\}, \bigcup (a''(X \cup \{i\})) - \{(i, \{\}\}))\}} = \bigcup (\text{Range a})
\text{proof} –

74
let \(?d=\text{Domain}\)
let \(?r=\text{Range}\)
let \(?U=\text{Union}\)
let \(?p=\text{is-non-overlapping}\)
let \(?P=\text{partitionsUniverse}\)
let \(?u=\text{runiq}\)
let \(\Xi=\text{X} \cup \{i\}\)
let \(?b=\Xi \times (?r \ a)\)
let \(?a1=a - ?b\)
let \(?Yi=a''\Xi\)
let \(?Y=\text{U} ?Yi\)
let \(?A2=\{(i, ?Y)\}\)
let \(?a3=\{(i,\{\})\}\)
let \(?a2=\{\ a2 \ - \ a3 \}\)
let \(?aa1=\text{a outside} \ ?Xi\)
let \(?c=\ a1 \cup \ a2\)
let \(?t1=c \in \text{allocationsUniverse}\)

have
1: \(?U(\ ?r(\ a1 \cup a2))=\ ?U(\ ?r \ a1) \cup (\ ?U(\ ?r \ a2))\) by (metis \ Range-Un-eq \ Union-Un-distrib)

have
2: \(?U(\ ?r \ a) \subseteq \ ?U(\ ?r \ a1) \cup (\ ?U(\ ?r \ a''\Xi)) \ & \ ?U(\ ?r \ a1) \cup (\ ?U(\ ?r \ a2) \subseteq \ ?U(\ ?r \ a)\)
by blast

then have
4: \(?p (\ ?r \ a1) \ & \ ?p ?Yi\) using \ subset-is-non-overlapping\ by \ metis

have \(\ ?a1 \in \text{allocationsUniverse} \ & \ ?a2 \in \text{allocationsUniverse}\)

using \ allocationUniverseProperty \ assms(1) \ reducedAllocation\ by \ fastforce

then have \(\ (a=a1 \ | \ \ a2=\{\})\rightarrow \ ?t1\)

using \ Un-empty-left\ by \ (metis \ lifting, \ no-types) \ Un-absorb2 \ empty-subsetI)

moreover have \(\ (a1=\{} \ | \ a2=\{}\rightarrow \ ?U (\ ?r \ a)=\ ?U (\ ?r \ a1) \cup (\ ?U (\ ?r \ a2)\) \)
by \ fast

ultimately have
5: \(\ (a1=\{} \ | \ a2=\{})\rightarrow \ ?thesis\ using \ 1\ by \ simp\)

{ assume
6: \(?a1\neq\{} \ & \ ?a2\neq\{}\)

then have \(?r \ a2 \geq \{\ Y\}\)

using \ Diff-cancel \ Range-insert \ empty-subsetI \ insert-Diff-single \ insert-iff \ insert-subset\)

by \ (metis \ (opaque-lifting, \ no-types))

then have
7: \(?U (\ ?r \ a)=\ ?U (\ ?r \ a1) \cup (\ ?U (\ ?r \ a2)\)\ using \ 2\ by \ blast\)

have \(?r \ a1 \neq \{} \ & \ ?r \ a2 \neq \{}\) using \ 6\ by \ auto

moreover have \(?r \ a1 \subseteq a''(\ ?d \ a1)\) using \ assms\ by \ blast

moreover have \(?Yi \cap (\ a''(\ ?d \ a - \ ?Xi))=\{}\)

using \ assms \ 3, \ 6\ Diff-disjoint \ intersectionEmptyRelationIntersectionEmpty
by metis
ultimately moreover have \( ?r \ ?a1 \cap ?Yi = \{ \} \) & \( ?Yi \neq \{ \} \) by blast
ultimately moreover have \( ?p \{ ?r \ ?a1, \ ?Yi \} \) unfolding is-non-overlapping-def

using IntI Int-commute empty-iff insert-iff subsetI subset-empty by metis
moreover have \( ?U \{ ?r \ ?a1, \ ?Yi \} \subseteq ?r \ a \) by auto
then moreover have \( ?p \ ( ?U \{ ?r \ ?a1, \ ?Yi \}) \) by (metis 3 Outside-def subset-is-non-overlapping)
ultimately moreover have \( ?p \{ ?U\{ ?r ?a1, \ ?Yi \}\} \) using bm043 by fast
moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \neq y \)
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \)
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \) using IntI empty-iff by metis
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \)
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \) using IntI empty-iff by metis
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \)
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \)
ultimately moreover have \( \forall x \in ?r \ ?a1. \forall y \in ?Yi. \ x \cap y = \{ \} \) using unionIntersectionEmpty

proof –

have \( \forall v0. \forall v1. \ v0 \in \text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a) \rightarrow (\forall v1. \ v1 \in \ a^{\vdash} \ (X \cup \{i\})) \rightarrow v0 \cap v1 = \{\} \)
by (metis (no-types) \( \forall x \in \text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a). \forall y \in a^{\vdash} \ (X \cup \{i\}). \ x \cap y = \{\} \))
thus \( \bigcup (\text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a)) \cap \bigcup (\forall \ a^{\vdash} \ (X \cup \{i\})) = \{\} \) by blast
qed

then have \( ?U \ ( ?r \ ?a1) \cap \ ?Y = \{ \} \) using unionIntersectionEmpty

by fastforce
ultimately have \( ?a1 \in \text{allocationsUniverse} \) using assms(1) reducedAllocation
by blast
moreover have \( ?a2 \in \text{allocationsUniverse} \) using allocationUniverseProperty
by fastforce
ultimately have \( ?a1 \in \text{allocationsUniverse} \) & \( ?a2 \in \text{allocationsUniverse} \) & \( \bigcup (\text{Range} \ ?a1) \cap \bigcup (\text{Range} \ ?a2) = \{\} \) & Domain \( ?a1 \cap \text{Domain} \ a \)

\( \ ?a2 = \{\} \)
by blast
then have \( \?I \) using allocationUnion by auto
then have \( \?thesis \) using 1 7 by simp
qed

corollary \( \text{allocationsUniverseOutsideUn}\):
assumes \( a \in \text{allocationsUniverse} \)
shows \( \ (a \ outside \ (X \cup \{i\})) \cup \ \{i\} \times (\bigcup (a^{\vdash}(X \cup \{i\})) - \{\{\}) \}) \in \text{allocation-} \)

\( \bigcup ((a \ outside \ (X \cup \{i\})) \cup \ \{i\} \times (\bigcup (a^{\vdash}(X \cup \{i\})) - \{\{\}) \}) = \)
\( \bigcup (\text{Range} \ a) \)
proof –

have \( a = ((X \cup \{i\}) \times \text{Range} \ a) = a \ outside \ (X \cup \{i\}) \) using Outside-def by metis
moreover have \( a = ((X \cup \{i\}) \times \text{Range} \ a) = a \ outside \ (X \cup \{i\}) \) using Outside-def by metis

76
\{(i,\{}\}\} \in \textit{allocationsUniverse} \\
\textbf{using} \textit{assms lm044 by fastforce} \\
moreover have \(\bigcup (\text{Range } ((a \setminus ((X\cup\{i\})\times (\text{Range } a)) \cup \{(i, \bigcup (a^\prime(X \cup \{i\})))) - \{(i,\{}\})\})\) = \(\bigcup (\text{Range } a)\) \\
\textbf{using} \textit{assms lm044 by (metis \textit{(no-types)})} \\
ultimately have \\
\(\{(i, \bigcup (a^\prime(X \cup \{i\})))) - \{(i,\{}\}\})\) = \(\bigcup (\text{Range } a)\) \\
\textbf{by simp} \\
moreover have \\
\(\{(i, \bigcup (a^\prime(X \cup \{i\})))) - \{(i,\{}\}\}) = \{i\} \times \{\bigcup (a^\prime(X\cup\{i\}))\} - \{(i,\{}\}\})\) \\
\textbf{by fast} \\
ultimately show \(?\text{thesis by auto}\) \\
\textit{qed} \\

\textbf{lemma} \textit{lm045:} \\
\textit{assumes} \(\text{Domain } a \cap X \neq {}\) \(a \in \textit{allocationsUniverse}\) \\
\textit{shows} \(\bigcup (a^\prime X) \neq {}\) \\
\textit{proof} – \\
let \(?p = \text{is-non-overlapping}\) \\
let \(?r = \text{Range}\) \\
have \(?p (\forall r \ a \text{ using assms Int-iff nonOverlapping by auto}\) \\
moreover have \(a^\prime X \subseteq ?r \ a \text{ by fast}\) \\
ultimately have \(?p (a^\prime X) \text{ using assms subset-is-non-overlapping by blast}\) \\
moreover have \(a^\prime X \neq {}\) \text{ using assms by fast}\) \\
ultimately show \(?\text{thesis by \textit{(metis Union-member all-not-in-conv no-empty-in-non-overlapping)}}\) \\
\textit{qed} \\

\textbf{corollary} \textit{lm046:} \\
\textit{assumes} \(\text{Domain } a \cap X \neq {}\) \(a \in \textit{allocationsUniverse}\) \\
\textit{shows} \(\{\bigcup (a^\prime(X\cup\{i\}))\} - \{\{}\} = \{\bigcup (a^\prime(X\cup\{i\}))\}\) \\
\textit{using} \textit{assms lm045 by fast}\n
\textbf{corollary} \textit{lm047:} \\
\textit{assumes} \(a \in \textit{allocationsUniverse}\) \\
\(\text{(Domain } a) \cap X \neq {}\) \\
\textit{shows} \\
\(\{\text{a outside } (X\cup\{i\}) \cup (\{i\} \times \{\bigcup (a^\prime(X\cup\{i\}))\})\} \in \textit{allocationsUniverse}\) \\
\textit{&} \\
\(\bigcup (\text{Range } ((\text{a outside } (X\cup\{i\}) \cup (\{i\} \times \{\bigcup (a^\prime(X\cup\{i\}))\}))\}) = \bigcup (\text{Range } a)\) \\
\textit{proof} – \\
let \(?t1 = \{\text{a outside } (X\cup\{i\}) \cup (\{i\} \times \{\bigcup (a^\prime(X\cup\{i\}))\} - \{\{}\})\} \in \textit{allocation-}
\( s_{\text{Universe}} \)

let \( ?t2 = \bigcup (\text{Range}((a \text{ outside } (X \cup \{i\}))) \cup \{i\} \times (\bigcup (a^{-1}(X \cup \{i\}))) - \{\{\})\)) = \bigcup (\text{Range} a) \)

have \( \emptyset \colon a \in \text{allocationsUniverse using assms(1) by fast} \)
then have \( ?t1 \& ?t2 \) using allocationsUniverseOutsideUnion
proof –
have \( a \in \text{allocationsUniverse} \longrightarrow \)
\( \quad a \text{ outside } (X \cup \{i\}) \cup \{i\} \times (\bigcup (a^{-1}(X \cup \{i\}))) - \{\{\})\) \( \in \text{allocationsUniverse} \)
using allocationsUniverseOutsideUnion by fastforce
hence \( a \text{ outside } (X \cup \{i\}) \cup \{i\} \times (\bigcup (a^{-1}(X \cup \{i\}))) - \{\{\})\) \( \in \text{allocationsUniverse} \)
by (metis 0)
thus \( a \text{ outside } (X \cup \{i\}) \cup \{i\} \times (\bigcup (a^{-1}(X \cup \{i\}))) - \{\{\})\) \( \in \bigcup (\text{Range} (a \text{ outside } (X \cup \{i\}) \cup \{i\} \times (\bigcup (a^{-1}(X \cup \{i\})))) \)
using \( \emptyset \) by (metis (no-types) allocationsUniverseOutsideUnion)
qed
moreover have \( \{\bigcup (a^{-1}(X \cup \{i\}))) - \{\{\}\} = \{} \{ \bigcup (a^{-1}(X \cup \{i\}))) \} \) using lm045 assms by fast
ultimately show ?thesis by auto
qed

abbreviation
bidMonotonicity b i ==
(\( \forall \; t \; t'. \; (\text{trivial } t \; \& \; \text{trivial } t' \; \& \; \text{Union } t \subseteq \text{Union } t') \longrightarrow \)
\( \quad \sum b (\{i\} \times t) \leq \sum b (\{i\} \times t') \) )

lemma lm048:
assumes bidMonotonicity b i runiq a
shows \( \sum b (\{i\} \times (a \text{ outside } X)^{-1}(\{i\})) \leq \sum b (\{i\} \times \bigcup (a^{-1}(X \cup \{i\}))) \) 

proof –
let \( ?u = \text{runiq} \)
let \( ?I = \{i\} \)
let \( ?R = a \text{ outside } X \)
let \( ?U = \text{Union} \)
let \( ?Xi = X \cup ?I \)
let \( ?t1 = ?R^{-1} ?I \)
let \( ?t2 = \{?U \circ ?R^{-1} ?I\} \)
have \( ?U (\{R^{-1} ?I\} \subseteq \{U (\{R^{-1}(X \cup \{I\})) \) by blast
moreover have \( \subseteq \) \( ?U (\{R^{-1}(X \cup \{I\})) \) using Outside-def by blast
ultimately have \( ?U (\{R^{-1} ?I\} \subseteq \{U (\{R^{-1}(X \cup \{I\})) \) by auto
then have \( \emptyset \colon ?U ?t1 \subseteq \{U ?t2 \) by auto
have \( ?u \; a \) using assms by fast

78
moreover have \(?R \subseteq a\) using Outside-def by blast ultimately
have \(?u \in R\) using subrel-runiq by metis
then have trivial \(?t1\) by (metis runiq-alt)
moreover have trivial \(?t2\) by (metis trivial-singleton)
ultimately show \(?\text{thesis}\) using assms 0 by blast
qed

lemma \textbf{lm049}:
assumes \(XX \in \text{partitionValuedUniverse}\)
shows \(\{} \notin \text{Range XX}\)
using assms mem-Collect-eq no-empty-in-non-overlapping by auto

corollary \textbf{emptyNotInRange}:
assumes \(a \in \text{allAllocations N G}\)
shows \(\{} \notin \text{Range a}\)
using assms \textbf{lm049} allAllocationsUniverse by auto blast

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

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

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

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

lemma \textbf{lm053}:
assumes \(a \in \text{allAllocations N G}\)
\(\, i \in N - X\)
\(\, \text{Domain a} \cap X \neq \{\}\\)
shows \(a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times \bigcup (a''(X \cup \{i\}))) \in \text{allAllocations } (N - X) (\bigcup (\text{Range a}))\)

\textbf{proof} –
let \(?R = a \text{ outside } X\)
let \(?I = \{i\}\)
let \(?U = \text{Union}\)
let ?u = runiq
let ?r = Range
let ?d = Domain
let ?aa = a outside (X ∪ {i}) ∪ ({i} × {?U(a"(X∪{i}))})).

have
1: a ∈ allocationsUniverse using assms(1) allAllocationsUniverse rev-subsetD
by blast
have i /∈ X using assms by fast
then have
2: ?d a − X ∪ {i} = ?d a ∪ {i} − X by fast
have a ∈ allocationsUniverse using 1 by fast
moreover have ?d a ∩ X ≠ {} using assms by fast
ultimately have ?aa ∈ allocationsUniverse & ?U (?r ?aa) = ?U (?r a) apply
(rule lm007) done
then have ?aa ∈ allAllocations (?d ?aa) (?U (?r a))
using lm007 by (metis (lifting, mono-tags))
then have ?aa ∈ allAllocations (?d ?aa ∪ (?d a − X ∪ {i})) (?U (?r a))
by (metis lm007)
moreover have ?d a − X ∪ {i} = ?d a ∪ (?d a − X ∪ {i}) using Outside-def
by auto
ultimately have ?aa ∈ allAllocations (?d a − X ∪ {i}) (?U (?r a)) by simp
then have ?aa ∈ allAllocations (?d a ∪ {i} − X) (?U (?r a)) using 2 by simp
moreover have ?d a ⊆ N using assms(1) possisbleAllocationsRelCharacteriza-
tion by metis
then moreover have (?d a ∪ {i} − X) ∪ (N − X) = N − X using assms by
fast
ultimately have ?aa ∈ 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 ∈ allocationsUniverse
Domain a ∩ X ≠ {} 
finte a
shows sum b (a outside X) ≤ 
sum b (a outside (X ∪ {i}) ∪ ({i} × {?U(a"(X∪{i}))})))

proof –
let ?R = a outside X
let ?I = {i}
let ?U = Union
let ?u = runiq
let ?r = Range
let ?d = Domain
let ?aa = a outside (X ∪ {i}) ∪ ({i} × {?U(a"(X∪{i}))})))
have a ∈ injectionsUniverse using assms by fast
then have
0: ?u a by simp
moreover have \( ?R \subseteq a \& \ ?u \subseteq a \) using Outside-def using \lm088 by auto
ultimately have finite \((?R -- i) \& \ ?u \) & \( ?u \?R\)
  using finite-subset subrel-runiq by (metis assms(4))
then moreover have trivial \((\{i\} \times (?R" \{i\}))\) using runiq-def
  by (metis trivial-cartesian trivial-singleton)
musre have \( \forall X. (\{i\} \times X) = \{\}\) using outside-reduces-domain
  by force
ultimately have \( I : \text{finite } (?R -- i) \& \text{finite } \(\{i\} \times (?R" \{i\})\) \)&
  \( (?R -- i) \cap \(\{i\} \times (?R" \{i\})\) = \{\}\)
  using Outside-def trivial-implies-finite by fast
have \(?R = (?R -- i) \cup (\{i\} \times (?R" \{i\}))\) by (metis outsideUnion)
then have \(?R = \text{sum } b \?R = \text{sum } b \text{ (?R -- i)} + \text{sum } b \(\{i\} \times (?R" \{i\})\)\)
  using I sum.union-disjoint by (metis (lifting) sum.union-disjoint)
moreover have \( ?R = \text{sum } b \(\{i\} \times (?R" \{i\})\) \leq \text{sum } b \(\{i\} \times \{U(a"(X\cup\{i\}))\}\)\)
  using \lm048 \(\text{assms(1)} \) 0 by metis
ultimately have \( ?R \leq \text{sum } b \text{ (?R -- i)} + \text{sum } b \(\{i\} \times \{U(a"(X\cup\{i\}))\}\)\)
  by linarith
moreover have \( \cdots = \text{sum } b \text{ (?R -- i)} \cup (\{i\} \times \{U(a"(X\cup\{i\}))\})\)
  using I sum.union-disjoint by auto
moreover have \( \cdots = \text{sum } b \ ?aa \) by (metis outsideOutside)
ultimately show \(?thesis\) by simp
qed

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

lemma \lm055:
  assumes \( \text{finite } N \text{ finite } G \ a \in \text{allAllocations } N G\)
  shows \( \text{finite } a\)
  using assms finiteRelationCharacterization rev-finite-subset
  by (metis characterizationallAllocations elementOfPartitionOfFiniteSetIsFinite)

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

corollary \lm056:
  assumes \( \text{bidMonotonicity } (b : => \text{ real }) \ i \ a \in \text{allAllocations } N G\)

\( i \in N \setminus X \)
\( \text{Domain } a \cap X \neq \{\} \)
\( \text{finite } N \)
\( \text{finite } G \)
shows \( \text{Max} ((\text{sum } b)\{\text{allAllocations } (N \setminus X) G\}) \geq \text{sum } b (a \text{ outside } X) \)

proof –

let \( ?aa = a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a''(X \cup \{i\}))\}) \)
have bidMonotonicity \((b::=\Rightarrow \text{real}) \ i\) using assms(1) by fast
moreover have \( a \in \text{allocationsUniverse} \) using assms(2) allAllocationsUniverse by blast
moreover have \( \text{finite } a \) using assms lm055 by metis
ultimately have \( 0 : \text{sum } b ?aa \leq \text{sum } b ?aa \) by \( \text{thesis} \)
then show \( ?\text{thesis} \) using \( \theta \) by linarith
qed

lemma cardinalityPreservation:
assumes \( \forall X \in XX. \text{finite } X \text{ is-non-overlapping } XX \)
shows \( \text{card} (\bigcup XX) = \text{sum } \text{card } XX \)
by \( \text{metis} \) assms \( \text{is-non-overlapping-def} \) card-Union-disjoint disjointI

corollary cardSumCommute:
assumes \( XX \text{ partitions } X \) \( \text{finite } X \) \( \text{finite } XX \)
shows \( \text{card} (\bigcup XX) = \text{sum } \text{card } XX \)
using assms cardinalityPreservation by \( \text{metis} \) is-partition-of-def familyUnion-FiniteEverySetFinite

lemma sumUnionDisjoint1:
assumes \( \forall A \in C. \text{finite } A \forall A \in C. \forall B \in C. A \neq B \rightarrow A \text{ Int } B = \{\} \)
shows \( \text{sum } f (\text{Union } C) = \text{sum } (\text{sum } f) C \)
using assms sum.Union-disjoint by fastforce

corollary sumUnionDisjoint2:
assumes \( \forall x \in X. \text{finite } x \text{ is-non-overlapping } X \)
shows \( \text{sum } f (\bigcup X) = \text{sum } (\text{sum } f) X \)
using assms sum.UnionDisjoint1 is-non-overlapping-def by fast

82
corollary sumUnionDisjoint3:
assumes $\forall x \in X. \text{finite } x X \text{ partitions } XX$
shows $\text{sum } f XX = \text{sum } (\text{sum } f) X$
using assms by (metis is-partition-of-def sumUnionDisjoint2)

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

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

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

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

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

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

corollary allAllocationsIntersectionSubset:
allocactions $N G \subseteq \text{allocationsUniverse } \&$
allocation $N G \subseteq \{a. (\text{Domain a}) \subseteq N \& \cup (\text{Range a}) = G\}$
(is $?L \subseteq ?R1 \cap ?R2$)
proof -
have $?L \subseteq ?R1 \& ?L \subseteq ?R2$ by (rule lm061) thus ?thesis by auto
qed

corollary allAllocationsIntersection:
allocation $N G =$
$(\text{allocationsUniverse } \cap \{a. (\text{Domain a}) \subseteq N \& \cup (\text{Range a}) = G\})$
(is $?L = ?R$)
proof 

have \( \mathcal{L} \subseteq \mathcal{R} \) using allAllocationsIntersectionSubset by metis 
moreover have \( \mathcal{R} \subseteq \mathcal{L} \) using lm058 by fast 
ultimately show \(?thesis\) by force 
qed 

**corollary** allAllocationsIntersectionSetEquals: 
\( a \in allAllocations \ N \ G = \) 
\((a \in allocationsUniverse \ & \ (\text{Domain } a) \subseteq N \ & \bigcup (\text{Range } a) = G) \) 
using allAllocationsIntersection Int-Collect by (metis (mono-tags, lifting)) 

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

9.4 Bridging theorem for injections 

**lemma** lm062: 
\( \text{totalRels } \emptyset \ Y = \{ \emptyset \} \) 
by fast 

**lemma** lm063: 
\( \emptyset \in \text{injectionsUniverse} \) 
by (metis CollectI converse-empty runiq-emptyrel) 

**lemma** lm064: 
\( \text{injectionsUniverse} \cap (\text{totalRels } \emptyset \ Y) = \{ \emptyset \} \) 
using lm062 lm063 by fast 

**lemma** lm065: 
assumes runiq \( f \ x \notin \text{Domain } f \) 
shows \( \{ f \cup \{(x, y)\} \mid y \cdot y \in A \} \subseteq \text{runiqs} \) 
unfolding paste-def runiqs-def using assms runiq-basic by blast 

**lemma** lm066: 
\( \text{converse } (\text{converse } \cdot X) = X \) 
by auto 

**lemma** lm067: 
\( \text{runiq } (f^{-1}) = (f \in \text{converse}'\text{runiqs}) \) 
unfolding runiqs-def by auto 

**lemma** lm068: 
assumes runiq \( (f^{-1}) A \cap \text{Range } f = \{ \} \) 
shows \( \text{converse } \cdot \{ f \cup \{(x, y)\} \mid y \cdot y \in A \} \subseteq \text{runiqs} \) 
using assms lm065 by fast 

**lemma** lm069:
assumes \( f \in \text{converse}'\text{runiq} \cap \text{Range} f = \{\} \)
shows \( \{f \cup \{(x, y)\} \mid y \in A\} \subseteq \text{converse}'\text{runiq} \)
(is \(?l \subseteq ?r\)
proof –
  have \text{runiq} (\sim^{-1} f) using assms(1) \text{ lm067} by blast
  then have \text{converse}' ?l \subseteq \text{runiq} using assms(2) by (rule \text{ lm068})
  then have \(?r \supseteq \text{converse}'\{\text{converse}?r\}) by auto
  moreover have \text{converse}'\{\text{converse}?l\} = ?l by (rule \text{ lm066})
  ultimately show \(?thesis by simp
qed

lemma \text{ lm070}:
\{ R \cup \{(x, y)\} \mid y \in A \} \subseteq \text{totalRels} (\{ x \} \cup \text{Domain} R) (A \cup \text{Range} R)
  by force

lemma \text{ lm071}:
\text{injectionsUniverse} = \text{runiq} \cap \text{converse}'\text{runiq}
  unfolding \text{runiq-def} by auto

lemma \text{ lm072}:
assumes \( f \in \text{injectionsUniverse} x \notin \text{Domain} f A \cap \text{Range} f = \{\} \)
shows \( \{f \cup \{(x, y)\} \mid y \in A\} \subseteq \text{injectionsUniverse} \)
(is \(?l \subseteq ?r\)
proof –
  have \( f \in \text{converse}'\text{runiq} \) using assms(1) \text{ lm071} by blast
  then have \(?l \subseteq \text{converse}'\text{runiq} \) using assms(2) by (rule \text{ lm069})
  moreover have \(?l \subseteq \text{runiq} \) using assms(1,2) \text{ lm065} by force
  ultimately show \(?thesis using \text{ lm071} by blast
qed

lemma \text{ lm073}:
\text{injections} \( X \ Y = \text{totalRels} \ X \ Y \cap \text{injectionsUniverse} \)
  using \text{ lm008} by \text{metis}

lemma \text{ lm074}:
assumes \( f \in \text{injectionsUniverse} \)
shows \( f \text{ outside} A \in \text{injectionsUniverse} \)
using assms by (metis (no-types) \text{Outside-def} \text{ lm030})

lemma \text{ lm075}:
assumes \( R \in \text{totalRels} A \ B \)
shows \( R \text{ outside} C \in \text{totalRels} (A - C) \ B \)
  unfolding \text{Outside-def} using assms by blast

lemma \text{ lm076}:
assumes \( g \in \text{injections} A \ B \)
shows \( g \text{ outside} C \in \text{injections} (A - C) \ B \)
using assms \text{Outside-def} \text{Range-outside-sub} \text{ lm030} \text{ mem-Collect-eq} \text{ outside-reduces-domain}
  unfolding \text{injections-def}

85
by fastforce

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

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

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

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

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

lemma lm082:
assumes $g \in \text{injections } X Y \ x \in \text{Domain } g$
shows $g \in \{ g' - x \cup \{ (x,y) \}|g. y \in Y - (\text{Range}(g' - x)) \}$
proof -
let $?g = g' - x$
have $g \in \text{injectionsUniverse}$ using assms(1) lm008 by fast
then moreover have $g',x \in g''\{x\}$
using assms(2) by (metis Image-runiq-eq-eval insert11 mem-Collect-eq)
ultimately have $g',x \in Y - \text{Range } ?g$ using lm081 assms(1) unfolding injections-def by fast
moreover have $g = ?g \cup \{ (x, g',x) \}$
using assms bm080 mem-Collect-eq unfolding injections-def by (metis (lifting))
ultimately show $?thesis$ by blast
qed

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

corollary lm084:
assumes $x \notin X \ g \in \text{injections } (\{ x \} \cup X) Y$
(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 $(\bigcup f \in \text{injections} \ X \ Y. \ \{ f \cup \{(x, y)\} \mid y . \ y \in Y - (\text{Range} \ f) \}) = \text{injections} \ (\{x\} \cup X) \ Y$
(is $\ ?r=\text{injections} \ ?X -$)
proof
have $0: \ ?r = (\bigcup f \in \text{injections} \ X \ Y. \ \{ f \cup \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \})$
(by $\text{ blast}$)
have $\ ?r' \subseteq \text{injections} \ ?X \ Y$ using $\text{assms}$ by (rule $\text{lm086}$) moreover have ...
= $\text{injections} \ ?X \ Y$
unfolding $\text{lm005}$
by $\text{ simp}$ ultimately have $\ ?r \subseteq \text{injections} \ ?X \ Y$ using $0$ by $\text{simp}$
moreover have $\text{injections} \ ?X \ Y \subseteq \ ?r$ using $\text{assms}$ by (rule $\text{lm085}$)
ultimately show $\ ?\text{thesis}$ by $\text{ blast}$
qed

lemma $\text{lm087}$:
assumes $\forall x. \ (P \ x \to (f \ x = g \ x))$
shows $\text{Union} \ \{ f \ | \ P \ x \} = \text{Union} \ \{ g \ x \mid x. \ P \ x \}$
using $\text{assms}$ by $\text{ blast}$

lemma $\text{lm088}$:
assumes $x \notin \text{Domain} \ R$
shows $R \ddagger \{(x,y)\} = R \cup \{(x,y)\}$
using $\text{assms}$ by (metis (erased, lifting) $\text{Domain-empty} \ \text{Domain-insert} \ \text{Int-insert-right-if0} \ \text{disjoint-iff-not-equal} \ \text{ex-in-conv} \ \text{paste-disj-domains})$

lemma $\text{lm089}$:
assumes $x \notin X$
shows $(\bigcup f \in \text{injections} \ X \ Y. \ \{ f \ddagger \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \}) = (\bigcup f \in \text{injections} \ X \ Y. \ \{ f \cup \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \})$
(is $\ ?l = \ ?r$)
proof
have $0: \ \forall f \in \text{injections} \ X \ Y. \ x \notin \text{Domain} \ f$ unfolding $\text{injections-def}$ using $\text{assms}$
by $\text{ fast}$
then have
1: $?l=\text{Union} \ \{ f \ddagger \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \}| f . \ f \in \text{injections} \ X \ Y \ & \ x \notin \text{Domain} \ f \}$
(by $\text{ blast}$)
(is $\ ?l' = ?l$) using $\text{assms}$ by $\text{ auto}$
moreover have
2: $?r=\text{Union} \ \{ f \cup \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \}| f . \ f \in \text{injections} \ X \ Y \ & \ x \notin \text{Domain} \ f \}$
(is $\ ?r' = ?r$) using $\text{ assms}$ $0$ by $\text{ auto}$

have $\forall f. \ f \in \text{injections} \ X \ Y \ & \ x \notin \text{Domain} \ f \to$
$\{ f \ddagger \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \} = \{ f \cup \{(x, y)\} \mid y . \ y \in Y - \text{Range} \ f \}$
using \texttt{lm088 by force}

then have \(?l'=?r'\) by (rule \texttt{lm087})

then show \(?l=?r\) using \texttt{1 2 by presburger}

\texttt{qed}

corollary \texttt{lm090}:

assumes \(x \notin X\)

shows \((\bigcup f \in \text{injections } X \ Y. \{f + \star \{(x, y)\} \mid y \cdot y \in Y - \text{Range } f\}) = \text{injections } ((\{x\} \cup X) \ Y)\)

(is \(?l=?r\))

\texttt{proof} –

have \(?l=(\bigcup f \in \text{injections } X \ Y. \{f \cup \{(x, y)\} \mid y \cdot y \in Y - \text{Range } f\})\) using \texttt{assms by (rule lm089)}

moreover have \(...=?r\) using \texttt{assms by (rule injectionsUnionCommute)}

ultimately show \(?\text{thesis by simp}\)

\texttt{qed}

\texttt{lemma \texttt{lm091}:}

\texttt{set \{f \cup \{(x,y)\}. y \leftarrow (filter \{(y. y \notin (\text{Range } f)\}) Y\} =}
\{f \cup \{(x,y)\}. y. y \in (\text{set } Y) - (\text{Range } f)\}

by \texttt{auto}

\texttt{lemma \texttt{lm092}:}

assumes \(\forall x \in \text{set } L. \text{set } (F x) = G x\)

shows \(\text{set } (\text{concat } [F x. x \leftarrow L]) = (\bigcup x \in \text{set } L. G x)\)

using \texttt{assms by force}

\texttt{lemma \texttt{lm093}:}

\texttt{set } (\text{concat } [f \cup \{(x,y)\}. y \leftarrow (\text{filter \{(y. y \notin (\text{Range } f)\}) Y\}]. f \leftarrow F]) =
(\bigcup f \in \text{set } F. \{f \cup \{(x,y)\} \mid y \cdot y \in (\text{set } Y) - (\text{Range } f)\})

by \texttt{auto}

\texttt{lemma \texttt{lm094}:}

assumes \(\text{finite } Y\)

shows \(\text{set } (f + \star \{(x,y)\}. y \leftarrow \text{sorted-list-of-set} (Y - (\text{Range } f))\} =\)
\{ f + \star \{(x,y)\}. y \cdot y \in Y - (\text{Range } f)\}

using \texttt{assms by auto}

\texttt{lemma \texttt{lm095}:}

assumes \(\text{finite } Y\)

shows \(\text{set } (\text{concat } [f + \star \{(x,y)\}. y \leftarrow \text{sorted-list-of-set}(Y - (\text{Range } f)]. f \leftarrow F]) =\)
(\bigcup f \in \text{set } F.\{f + \star \{(x,y)\} \mid y \cdot y \in Y - (\text{Range } f)\})

using \texttt{assms \texttt{lm094 \texttt{lm092 by auto}}}

89
9.5 Computable injections

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

corollary ln096:
  set (injectionsAlg (x # xs) Y) =
    (⋃ f ∈ set (injectionsAlg xs Y) f ∪ {(x,y)} | y . y ∈ (set Y) − (Range f))
  using ln093 by auto

corollary ln097:
  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 ln096 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 ln098:
  injections {} Y = {}
  by (simp add: ln008 ln062 runiq-emptyrel)

lemma ln099:
  injections {} Y = {}
  unfolding injections-def by (metis ln098 injections-def)

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

lemma ln100:
  assumes x /∈ set xs set (injectionsAlg xs Y) = injections (set xs) (set Y)
  shows set (injectionsAlg (x # xs) Y) = injections (set X) (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 ln097)
    moreover have ... = ?r using assms(1) by (rule injectionsUnionCommute)
    ultimately show ?thesis by simp
  qed
lemma lm101:
  assumes \( x \notin \text{set} \; \text{xs} \)
  \[
  \text{set} \; (\text{injections-alg} \; \text{xs} \; Y) = \text{injections} \; (\{x\} \cup \text{set} \; \text{xs}) \; Y
  \]
  finite \( Y \)
  shows \( \text{set} \; (\text{injections-alg} \; (x \# \text{xs}) \; Y) = \text{injections} \; (\{x\} \cup \text{set} \; \text{xs}) \; Y \)
(is \(?l = ?r\))
proof
  have \(?l = (\bigcup f \in \text{injections} \; (\text{set} \; \text{xs}) \; Y. \; \{f \; \circ \; \{(x,y)\} \mid y \in \text{Y} - \text{Range} \; f\})\)
    using assms(2, 3) lm095 by auto
  moreover have \(\ldots = ?r\) using assms(1) by (rule lm090)
  ultimately show \(?thesis\) by simp
qed

lemma listInduct:
  assumes \( P [] \\forall \; \text{xs} \; x. \; P \; \text{xs} \rightarrow P \; (x \# \text{xs})\)
  shows \( \\forall \; x. \; P \; x\)
using assms by (metis structInduct)

lemma injectionsFromEmptyAreEmpty:
  set (injections-alg [] \; Z) = \{\}\nby simp

theorem injections-equiv:
  assumes finite \( Y \) and distinct \( X \)
  shows \( \text{set} \; (\text{injections-alg} \; X \; Y) = \text{injections} \; (\text{set} \; X) \; Y \)
proof
  let \(?P = \lambda \; l. \; \text{distinct} \; l \rightarrow (\text{set} \; (\text{injections-alg} \; l \; Y) = \text{injections} \; (\text{set} \; l) \; Y)\)
  have \(?P []\) using injectionsFromEmptyAreEmpty list(1) lm099 by metis
  moreover have \(\forall \; x \; \text{xs}. \; ?P \; \text{xs} \rightarrow ?P \; (x \# \text{xs})\)
    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 \( l \in \text{set} \; (\text{all-partitions-list} \; G) \) distinct \( G \)
  shows \( \text{distinct} \; l \)
using assms by (metis all-partitions-equivalence')

lemma bridgingInjection:
  assumes card \( N > 0 \) distinct \( G \)
  shows \( \forall \; l \in \text{set} \; (\text{all-partitions-list} \; G). \; \text{set} \; (\text{injections-alg} \; l \; N) = \text{injections} \; (\text{set} \; l) \; N \)
using lm102 injections-equiv assms by (metis card-ge-0-finite)

lemma lm103:
assumes \( \text{card} \ N > 0 \) distinct \( G \)
shows \( \{ \text{injections} \ P \ N \mid P \in \text{all-partitions} \ (\text{set} \ G) \} = \) \( \text{set} \ \{ \text{injections-alg} \ l \ N \mid l \in \text{all-partitions-list} \ G \} \)

proof
- let \(?g1 = \text{all-partitions-list} \)
  let \(?f2 = \text{injections} \)
  let \(?g2 = \text{injections-alg} \)
  have \( \forall l \in \text{set} \ (?g1 \ G) \). \( \text{set} \ (\text{injections-alg} \ l \ N) \) = \( ?f2 \ (\text{set} \ l) \) \( N \) using \( \text{assms} \ \text{bridgingInjection} \) by blast
  then have \( \text{set} \ (\text{injections-alg} \ l \ N) \) = \( ?f2 \ (\text{set} \ l) \) \( N \) using \( \text{assms} \ \text{bridgingInjection} \) by blast
  ultimately show \( ?\text{thesis} \) by presburger
qed

lemma \( \text{lm104} \): 
assumes \( \text{card} \ N > 0 \) distinct \( G \)
shows \( \bigcup \ \{ \text{injections} \ P \ N \mid P \in \text{all-partitions} \ (\text{set} \ G) \} = \) \( \bigcup \ \text{set} \ (\text{injections-alg} \ l \ N) \ . \ l \leftarrow \text{all-partitions-list} \ G \) (is \( \bigcup \ ?L = \bigcup \ ?R \))
proof
- have \( ?L = ?R \) using \( \text{assms} \) by (rule \( \text{lm104} \)) thus \( ?\text{thesis} \) by presburger
qed

corollary \text{allAllocationsBridgingLemma}:
assumes \( \text{card} \ N > 0 \) distinct \( G \)
shows \( \text{allAllocations} \ N \ (\text{set} \ G) = \) \( \text{set}(\text{allAllocationsAlg} \ N \ G) \)
proof
- let \( ?LL = \bigcup \ \{ \text{injections} \ P \ N \mid P \in \text{all-partitions} \ (\text{set} \ G) \} \)
  let \( ?RR = \bigcup \ (\text{set} \ (\text{injections-alg} \ l \ N) \ . \ l \leftarrow \text{all-partitions-list} \ G) \)
  have \( ?LL = ?RR \) using \( \text{assms} \) by (rule \( \text{lm104} \))
  then have \text{converse} \ ?LL = \text{converse} \ ?RR \) by simp
  thus \( ?\text{thesis} \) by force
qed

end

10 Termination theorem for uniform tie-breaking

theory \text{UniformTieBreaking}

imports
\text{StrictCombinatorialAuction}
\text{Universes}

92
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:
\(\{x\} \times (\{\}_X \setminus \{\}) \in \text{allocationsUniverse} \)
using allocationUniverseProperty pairDifference by metis

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

corollary lm004:
\(\text{allocationsUniverse} \neq \{} \)
using lm003 by fast

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

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

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

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

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

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

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

lemma lm013:
assumes $a \in \text{allAllocations} N G \text{ finite} G$
shows finite $(\text{Range} a)$
using assms elementOfPartitionOfFiniteSetIsFinite by (metis allocationReversalseInjective)

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

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

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

10.3 Results on summed bid vectors

lemma lm016:
$\text{summedBidVectorRel} \text{ bids} N G =$
$\{((\text{pair}, \text{sum} (\%g. \text{bids} (\text{fst} \text{ pair}, \ g)) (\text{finestpart} (\text{snd} \ text{ pair})))) | \text{pair. pair} \in N \times (\text{Pow} G - \{\{\}))\}$
by blast
corollary lm017:
\[
\{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \} \parallel a = \\
\{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \cap a \}
\]
by (metis restrictionVsIntersection)

corollary lm018:
\[
\text{summedBidVectorRel bids N G} \parallel a = \\
\{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \cap a \}
\]
(is ?L = ?R)
proof –
let ?l = summedBidVectorRel
let ?M = \{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \}
have ?l bids N G = ?M by (rule lm016)
then have ?L = (?M \parallel a) by presburger
moreover have ... = ?R by (rule lm017)
ultimately show ?thesis by simp
qed

lemma lm019:
\[
\text{summedBid bids} \cdot ((N \times (\text{Pow} G - \{\}) \}) \cap a) = \\
\{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \cap a \}
\]
by blast

corollary lm020:
\[
\text{summedBidVectorRel bids N G} \parallel a = \text{(summedBid bids) \cdot ((N \times (\text{Pow} G - \{\}) \}) \cap a) \\
\]
(is ?L= ?R)
proof –
let ?l=summedBidVectorRel
let ?p=summedBid
let ?M= \{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) \text{ (finestpart} (\text{snd pair}))) | \\
\text{pair. pair} \in (N \times (\text{Pow} G - \{\}) \}) \cap a \}
have ?L = ?M by (rule lm018)
moreover have ... = ?R using lm019 by blast
ultimately show ?thesis by simp
qed

lemma summedBidInjective:
\[
inj-on \text{ (summedBid bids) UNIV} \]
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

98
by (metis (lifting, no-types) inf-commute inf-sup-aci(1))

qed

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

qed

lemma lm029:
  assumes \{\} \notin \text{Range} \alpha \ \forall X \in \omega \ \alpha \ \text{finite X}
  \text{is-non-overlapping} \ (\omega \ \alpha)
  \text{shows} \text{card} \ (\text{pseudoAllocation} \alpha) = \text{sum} \ (\text{card} \circ \omega) \alpha
  \text{is} \ \Rightarrow \ \Rightarrow \ \text{?L} = \text{?R}
proof –
  \text{have} \ \Rightarrow \ \Rightarrow \ \text{?L} = \text{sum} \ \text{card} \ (\omega \ \alpha)
  \text{unfolding} \ \text{pseudoAllocation-def}
  \text{using assms by (simp add: cardinalityPreservation)}
  \text{moreover have} \ \cdots = \ ?R \text{ using assms(1) lm028 sum.reindex by blast}
  \text{ultimately show} \ \text{?thesis} \text{ by simp}

qed

lemma lm030:
  \text{card} \ (\omega \ \text{pair}) = \text{card} \ (\text{snd pair})
  \text{using} \text{cardFinestpart} \text{ card-cartesian-product-singleton by metis}

corollary lm031:
  \text{card} \circ \omega = \text{card} \circ \text{snd}
  \text{using} \text{lm030 by fastforce}

corollary lm032:
  \text{assumes} \{\} \notin \text{Range} \alpha \ \forall \ \text{pair} \in \alpha \ \text{finite} \ (\text{snd pair}) \ \text{finite a runiq a}
shows $\text{card} \ (\text{pseudoAllocation} \ a) = \sum (\text{card} \circ \text{snd}) \ a$

proof

let $?P = \text{pseudoAllocation}$
let $?c = \text{card}$

have $\forall \ \text{pair} \in a. \ \text{finite} \ \text{(omega pair)}$ using finiteFinestpart assms by blast

moreover have $\text{is-non-overlapping} \ \text{(omega } a)$ using assms lm027 by force

ultimately have $?c \ (\ ?P \ a) = \sum (\ ?c \circ \text{omega} \ a)$ using assms lm029 by force

moreover have $... = \sum (\ ?c \circ \text{snd}) \ a$ using lm031 by metis

ultimately show $?\text{thesis}$ by simp
qed

corollary lm033:

assumes runiq $(a^\sim -1)$ runiq a finite a $\{\} \notin \text{Range} \ a \ \forall \ \text{pair} \in a. \ \text{finite} \ \text{(snd pair)}$

shows $\text{card} \ (\text{pseudoAllocation} \ a) = \sum \text{card} \ \text{(Range} \ a)$

using assms sumPairsInverse lm032 by force

corollary lm034:

assumes $a \in \text{allAllocations} \ N \ G \ \text{finite} \ G$

shows $\text{card} \ (\text{pseudoAllocation} \ a) = \text{card} \ G$

proof

have $\{\} \notin \text{Range} \ a$ using assms by (metis emptyNotInRange)

moreover have $\forall \ \text{pair} \in a. \ \text{finite} \ \text{(snd pair)}$ using assms lm014 finitePairSecondRange by metis

moreover have $\text{finite} \ a$ using assms allocationFinite by blast

moreover have runiq a using assms

by (metis (lifting) Int-lower1 in-mono allocationInjectionsUniverseProperty mem-Collect-eq)

moreover have runiq $(a^\sim -1)$ using assms

by (metis (mono-tags) injections-def characterization allAllocations mem-Collect-eq)

ultimately have $\text{card} \ (\text{pseudoAllocation} \ a) = \sum \text{card} \ \text{(Range} \ a)$ using lm033 by fast

moreover have $... = \text{card} \ G$ using assms lm015 by metis

ultimately show $?\text{thesis}$ by simp
qed

corollary lm035:

assumes $\text{pseudoAllocation} \ aa \subseteq \text{pseudoAllocation} \ a \cup (N \times (\text{finestpart} \ G))$

finite $(\text{pseudoAllocation} \ aa)$

shows $\sum (\text{toFunction} \ (\text{bidMaximizedBy} \ a \ N \ G)) \ (\text{pseudoAllocation} \ a) -

(\sum (\text{toFunction} \ (\text{bidMaximizedBy} \ a \ N \ G)) \ (\text{pseudoAllocation} \ aa)) =

\text{card} \ (\text{pseudoAllocation} \ a) -

\text{card} \ (\text{pseudoAllocation} \ aa \cap (\text{pseudoAllocation} \ a))$

using assms subsetCardinality by blast

corollary lm036:

assumes $\text{pseudoAllocation} \ aa \subseteq \text{pseudoAllocation} \ a \cup (N \times (\text{finestpart} \ G))$

finite $(\text{pseudoAllocation} \ aa)$

shows $\int \ (\sum (\text{maxbid} \ a \ N \ G)) \ (\text{pseudoAllocation} \ a) -$
\[
\text{int} \left( \text{sum} \left( \text{maxbid} \ a \ N \ G \right) \right) = \text{int} \left( \text{card} \ (\text{pseudoAllocation} \ a) \right) - \text{int} \left( \text{card} \ (\text{pseudoAllocation} \ aa \cap (\text{pseudoAllocation} \ a)) \right)
\]

using differenceSumVsCardinality assms by blast

lemma lm037:
\[\text{pseudoAllocation} \ {} = {}\]

unfolding pseudoAllocation-def by simp

corollary lm038:
assumes \(a \in \text{allAllocations} \ N \ {}\)
shows \((\text{pseudoAllocation} \ a) = {}\)

unfolding pseudoAllocation-def using assms lm012 by blast

corollary lm039:
assumes \(a \in \text{allAllocations} \ N \ G \ \text{finite} \ G \ G \neq {}\)
shows \(\text{finite} \ (\text{pseudoAllocation} \ a)\)
proof –
\[\text{have card} \ (\text{pseudoAllocation} \ a) = \text{card} \ G \text{ using assms(1,2) lm034 by blast}\]
thus \(\text{finite} \ (\text{pseudoAllocation} \ a) \text{ using assms(2,3) by fastforce}\)
qed

corollary lm040:
assumes \(a \in \text{allAllocations} \ N \ G \ \text{finite} \ G\)
shows \(\text{finite} \ (\text{pseudoAllocation} \ a)\)
using assms finite.emptyI lm039 lm038 by (metis (no-types))

lemma lm041:
assumes \(a \in \text{allAllocations} \ N \ G \ aa \in \text{allAllocations} \ N \ G \ \text{finite} \ G\)
shows \(\text{card} \ (\text{pseudoAllocation} \ aa \cap (\text{pseudoAllocation} \ a)) = \text{card} \ (\text{pseudoAllocation} \ a)\)
(proof –
let \(?P=\text{pseudoAllocation}\)
let \(?c=\text{card}\)
let \(?A=?P \ a\)
let \(?AA=?P \ aa\)
\[\text{have } ?c \ ?A=\ ?c \ G \ & \ ?c \ ?AA=\ ?c \ G \text{ using assms lm034 by (metis (lifting, mono-tags))}\]
moreover have finite ?A & finite ?AA using assms lm040 by blast
ultimately show \(?\text{thesis using assms cardinalityIntersectionEquality by (metis(no-types,lifting))}\)
qed

lemma lm042:
\[\omega \text{ pair } = \{\text{fst } \text{pair}\} \times \{|\{y\}| \ y. \ y \in \text{snd } \text{pair}\}\]
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 ∈ 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 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 \( a = \{ (x, \{ y \}) | x, y \in \bigcup (a^*(x)) \land x \in \text{Domain} \ a \} \)
(is \( ?L = ?R \))
proof –
have \( ?L = \{ (\text{fst, pair}, \{ y \}) | y \in \text{snd, pair} \land \text{pair} \in \ a \} \) by (rule lm051)
moreover have \( ... = ?R \) by (rule lm052)
ultimately show \( ?\text{thesis by simp} \)
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:
\( N \times (\text{Pow} G - \{\{\}\}) = \text{Domain} (\text{summedBidVectorRel bids N G}) \)
by blast
corollary lm056:
assumes \( a \in \text{allAllocations N G} \)
shows \( a \subseteq \text{Domain} (\text{summedBidVectorRel bids N G}) \)
proof –
let \( ?p = \text{allAllocations} \)
let \( ?L = \text{summedBidVectorRel} \)
have \( a \subseteq (N \times (\text{Pow} G - \{\{\}\})) \) using asms allocationPowerset by (metis (no-types))
moreover have \( N \times (\text{Pow} G - \{\{\}\}) = \text{Domain} (?L bids N G) \) using summed-BidVectorCharacterization by blast
ultimately show \( ?\text{thesis by blast} \)
qed

corollary lm057:
\( \text{sum} (\text{summedBidVector bids N G}) (a \cap (\text{Domain} (\text{summedBidVectorRel bids N G}))) = \)
\( \text{sum snd} ((\text{summedBidVectorRel bids N G}) || a) \)
using sumRestrictedToDomainInvariant lm055 by fast
corollary lm058:
assumes \( a \in \text{allAllocations N G} \)
shows \( \text{sum} (\text{summedBidVector bids N G}) a = \text{sum snd} ((\text{summedBidVectorRel bids N G}) || a) \)
proof –
let \( ?L = \text{summedBidVector} \) let \( ?L = \text{summedBidVectorRel} \)
have \( a \subseteq \text{Domain} (?L \text{ bids N G}) \) using asms by (rule lm056)
then have \( a = a \cap \text{Domain} (?L \text{ bids N G}) \) by blast

103
then have \( \sum (\forall l \text{ bids } N \ G) \ a = \sum (\forall l \text{ bids } N \ G) \ (a \cap \text{Domain} \ (\forall L \text{ bids } N \ G)) \)
by presburger
thus \(?\text{thesis}\) using \(\text{ln057}\) by auto
qed

corollary \(\text{ln059}\):
assumes \(a \in \text{allAllocations } N \ G\)
shows \(\sum (\summedBidVector \text{ bids } N \ G) \ a = \sum \text{snd} \ ((\summedBid \text{ bids}) \cdot (\forall L \text{ bids } N \ G) \ (a \cap \text{Domain} \ (\forall L \text{ bids } N \ G)))\)
(is \(?X=\?)\)
proof –
let \(?p = \summedBid\)
let \(?l = \summedBidVector\)
let \(?L = \summedBidVectorRel\)
let \(?A = N \times (\text{Pow} \ G - \{\} ))\)
let \(?inner2 = (\forall p \text{ bids}) \ (\forall A \cap a)\)
let \(?inner1 = (\forall L \text{ bids } N \ G)\)
have \(?R = \sum \text{snd} \ ?inner1\) using assms \(\text{ln020}\) by (metis \(\text{no-types}\))
moreover have \(\sum (\forall l \text{ bids } N \ G) \ a = \sum \text{snd} \ ?inner1\) using assms by (rule \(\text{ln058}\))
ultimately show \(?\text{thesis}\) by simp
qed

corollary \(\text{ln060}\):
assumes \(a \in \text{allAllocations } N \ G\)
shows \(\sum (\summedBidVector \text{ bids } N \ G) \ a = \sum \text{snd} \ ((\summedBid \text{ bids}) \cdot a)\)
(is \(?L=?R\))
proof –
let \(?p=\summedBid\)
let \(?l=\summedBidVector\)
have \(?L = \sum \text{snd} \ ((\forall p \text{ bids}) \cdot (\forall X \cap a) )\) using assms by (rule \(\text{ln059}\))
moreover have ... = \(?R\) using assms \(\text{allocationPowerset Int-absorb1}\) by (metis \(\text{no-types}\))
ultimately show \(?\text{thesis}\) by simp
qed

corollary \(\text{ln061}\):
\(\sum \text{snd} \ ((\summedBid \text{ bids}) \cdot a) = \sum (\text{snd} \circ (\summedBid \text{ bids})) \ a\)
using \(\text{sum.reindex}\ \text{ln021}\) by blast

corollary \(\text{ln062}\):
assumes \(a \in \text{allAllocations } N \ G\)
shows \(\sum (\summedBidVector \text{ bids } N \ G) \ a = \sum (\text{snd} \circ (\summedBid \text{ bids})) \ a\)
(is \(?L=?R\))
proof –
let \(?p = \summedBid\)
let \(?l = \summedBidVector\)
have \( L = \sum \text{snd} (\{(p \ \text{bids})' \ a\}) \) using assms by (rule lm060)
moreover have \( \ldots = \sum R \) using assms lm061 by blast
ultimately show \( ?\text{thesis} \) by simp
qed

corollary lm063:
assumes \( a \in \text{allAllocations} \ N \ G \)
shows \( \sum (\text{summedBidVector} \ \text{bids} \ N \ G) \ a = \sum ((\sum \text{bids}) \circ \omega) \ a \)
(is \( ?L=?R \))
proof –
  let \( \text{inner1} = \text{snd} \circ (\text{summedBid} \ \text{bids}) \)
  let \( \text{inner2} = (\sum \text{bids}) \circ \omega \)
  let \( ?M=\sum ?\text{inner1} \ a \)
  have \( ?L=\sum \text{inner1} \ a \) using assms by (rule lm062)
moreover have \( \text{inner1} = \text{inner2} \) using lm023 assms by fastforce
ultimately show \( ?L=\sum R \) using assms by metis
qed

corollary lm064:
assumes \( a \in \text{allAllocations} \ N \ G \)
shows \( \sum (\text{summedBidVector} \ \text{bids} \ N \ G) \ a = \sum (\sum \text{bids}) \ (\omega' \ a) \)
proof –
  have \( \{} \notin \text{Range} \ a \) using assms by (metis emptyNotInRange)
  then have \( \text{inj-on} \ \omega \ a \) using lm028 by blast
  then have \( \sum (\sum \text{bids}) \ (\omega' \ a) = \sum ((\sum \text{bids}) \circ \omega) \ a \)
  by (rule \( \text{sum.reindex} \))
moreover have \( \sum (\text{summedBidVector} \ \text{bids} \ N \ G) \ a = \sum ((\sum \text{bids}) \circ \omega) \ a \)
using assms lm063 by (rule \( \text{Extraction.exE-realizer} \))
ultimately show \( \text{thesis} \) by presburger
qed

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

corollary lm066:
assumes \( \forall y \in (\text{Range} \ a) . \ \text{finite} \ y \)
shows \( \forall y \in (\omega' \ a) . \ \text{finite} \ y \)
using assms lm065 imageE finitePairSecondRange by fast

corollary lm067:
assumes \( a \in \text{allAllocations} \ N \ G \ \text{finite} \ G \)
shows \( \forall x \in (\omega' \ a) . \ \text{finite} \ x \)
using assms lm066 lm014 by (metis(no-types))

corollary lm068:
assumes \( a \in \text{allAllocations} \ N \ G \)
shows is-non-overlapping \((\omega^{a})\)

proof –

have runiq \(a\) by (metis (no-types) assms image-iff allocationRightUniqueRange-Domain)

moreover have \(\emptyset \notin \text{Range } a\) using assms by (metis emptyNotInRange)

ultimately show \(?\text{thesis}\) using \(\text{lm027}\) by blast

qed

lemma \(\text{lm069}\):  
assumes \(a \in \text{allAllocations } N\ G\) finite \(G\)
shows \(\text{sum (sum bids) (omega' a) = sum bids (} \bigcup \text{ (omega' a)})\)
using assms \(\text{sumUnionDisjoint2 lm068 lm067 }\) by (metis (lifting, mono-tags))

corollary \(\text{lm070}\):  
assumes \(a \in \text{allAllocations } N\ G\) finite \(G\)
shows \(\text{sum (summedBidVector bids } N\ G)\ a = \text{sum bids (pseudoAllocation } a)\)
(is \(?L = ?R\))
proof –

have \(?L = \text{sum (sum bids) (omega' a)}\) using assms \(\text{lm064}\) by blast
moreover have \(\text{... = sum bids (} \bigcup \text{ (omega' a)})\) using assms \(\text{lm069}\) by blast
ultimately show \(?\text{thesis}\) unfolding \(\text{pseudoAllocation-def}\) by presburger

qed

lemma \(\text{lm071}\):

\(\text{Domain (pseudoAllocation } a) \subseteq \text{Domain } a\)

unfolding \(\text{pseudoAllocation-def}\) by fastforce

corollary \(\text{lm072}\):

\(\text{assumes } a \in \text{allAllocations } N\ G\)

shows \(\text{Domain (pseudoAllocation } a) \subseteq N\ \& \ \text{Range (pseudoAllocation } a) = \text{finestpart } G\)

using assms \(\text{lm071 allocationInverseRangeDomainProperty lm050 is-partition-of-def}\)
subset-trans
by (metis (no-types))

corollary \(\text{lm073}\):

\(\text{assumes } a \in \text{allAllocations } N\ G\)

shows \(\text{pseudoAllocation } a \subseteq N \times \text{finestpart } G\)

proof –

let \(?p = \text{pseudoAllocation}\)
let \(?aa = ?p \ a\)
let \(?d = \text{Domain}\)
let \(?r = \text{Range}\)

have \(?d \ ?aa \subseteq N\) using assms \(\text{lm072}\) by (metis (lifting, mono-tags))

moreover have \(?r \ ?aa \subseteq \text{finestpart } G\) using assms \(\text{lm072}\) by (metis (lifting, mono-tags) equalityE)

ultimately have \(?d \ ?aa \times (?r \ ?aa) \subseteq N \times \text{finestpart } G\) by auto
then show \(?aa \subseteq N \times \text{finestpart } G\) by auto

qed
10.4 From Pseudo-allocations to allocations

abbreviation 
\textit{pseudoAllocationInv pseudo} ::= \{(x, \bigcup (\textit{pseudo} \ "\ {x})) | x, x \in \text{Domain pseudo}\}

lemma \textit{lm074}:
\begin{itemize}
\item assumes \textit{runiq a} \{\} \not\in \text{Range a}
\item shows \textit{a = pseudoAllocationInv (pseudoAllocation a)}
\end{itemize}

proof
\begin{itemize}
\item let \textit{?p} = \{(x, Y) | Y \in \text{finestpart (a, x) & x \in \text{Domain a}}\}
\item have \forall x \in \text{Domain a}. a.,x \neq \{\} by (metis \textit{assms eval-runiq-in-Range})
\item then have \forall x \in \text{Domain a}. \textit{finestpart (a, x) \neq \{\}} by (metis \text{notEmptyFinestpart})
\end{itemize}

then have \textit{Domain a} \subseteq \textit{Domain ?p} by \text{force}

moreover have \textit{Domain a} \supseteq \textit{Domain ?p} by \text{fast}

ultimately have \item 1: \textit{Domain a} = \textit{Domain ?p} by \text{fast}
\begin{itemize}
\item fix z assume z \in \textit{?a}
\item then obtain x where
\item x \in \text{Domain ?p} & z = (x, \bigcup (\textit{?p} \ "\ {x})) by \text{blast}
\item then have x \in \text{Domain a} & z = (x, \bigcup (\textit{?p} \ "\ {x})) by \text{fast}
\item then moreover have \textit{?p} \ "\ {x} = \textit{finestpart (a, x)} using \textit{assms by fastforce}
\item moreover have \bigcup (\textit{finestpart (a, x)})= a.,x by (metis \textit{finestPartUnion})
\item ultimately have z \in a by (metis \textit{assms(1) eval-runiq-rel})
\end{itemize}

then have \item 2: \textit{?a} \subseteq a by \text{fast}
\begin{itemize}
\item fix z assume 0: z \in a let \textit{?x = fst z} let \textit{?Y = a.,?x} let \textit{?YY = finestpart ?Y}
\item have z \in a & ?x \in \text{Domain a} using 0 by (metis \textit{fst-eq-Domain rev-image-eqI})
\end{itemize}

then have \item 3: z \in a & ?x \in \text{Domain ?p using 1 by presburger}
\begin{itemize}
\item then have \textit{?p} \ "\ \{?x\} = ?YY by \text{fastforce}
\item then have \bigcup (\textit{?p} \ "\ {?x}) = ?Y by (metis \textit{finestPartUnion})
\item moreover have z = (?x, ?Y) using \textit{assms by (metis 0 functionOnFirstElement-equalsSecond surjective-pairing)}
\end{itemize}

ultimately have z \in \textit{?a} using 3 by (metis \textit{(lifting, mono-tags) mem-Collect-eq})

then have a = ?a using 2 by \text{blast}

moreover have \textit{?p = pseudoAllocation a using \textit{lm048 assms by (metis (lifting, mono-tags))}}

ultimately show \textit{?thesis by auto}

qed

corollary \textit{lm075}:
\begin{itemize}
\item assumes a \in \textit{runiqs} \cap \textit{Pow (UNIV \times (UNIV - \{\{\}\\}}))
\end{itemize}
shows $(\text{pseudoAllocationInv} \circ \text{pseudoAllocation}) \ a = \text{id} \ a$

proof –

have $\text{runiq} \ a$ using runiqs-def assms by fast

moreover have $\{\} \notin \text{Range} \ a$ using assms by blast

ultimately show $?\text{thesis}$ using lm074 by fastforce

qed

lemma lm076:

inj-on $(\text{pseudoAllocationInv} \circ \text{pseudoAllocation}) \ (\text{runiqs} \cap \text{Pow} \ (\text{UNIV} \times (\text{UNIV} - \{\{\}\})))$

proof –

let $?\text{ne} = \text{Pow} \ (\text{UNIV} \times (\text{UNIV} - \{\{\}\}))$

let $?X = \text{runiqs} \cap ?\text{ne}$

let $?\text{f} = \text{pseudoAllocationInv} \circ \text{pseudoAllocation}$

have $\forall a1 \in ?X. \forall a2 \in ?X. ?\text{f} a1 = ?\text{f} a2 \longrightarrow \text{id} a1 = \text{id} a2$ using lm075 by blast

then have $\forall a1 \in ?X. \forall a2 \in ?X. ?\text{f} a1 = ?\text{f} a2 \longrightarrow a1 = a2$ by auto

thus $?\text{thesis}$ unfolding inj-on-def by blast

qed

corollary lm077:

inj-on $\text{pseudoAllocation}$ $(\text{runiqs} \cap \text{Pow} \ (\text{UNIV} \times (\text{UNIV} - \{\{\}\})))$

using lm076 inj-on-imageI2 by blast

lemma lm078:

injectionsUniverse $\subseteq$ runiqs

using runiqs-def Collect-conj-eq Int-lower1 by metis

lemma lm079:

partitionValuedUniverse $\subseteq$ Pow $(\text{UNIV} \times (\text{UNIV} - \{\{\}\}))$

using is-non-overlapping-def by force

corollary lm080:

allocationsUniverse $\subseteq$ runiqs $\cap$ Pow $(\text{UNIV} \times (\text{UNIV} - \{\{\}\}))$

using lm078 lm079 by auto

corollary lm081:

inj-on $\text{pseudoAllocation}$ allocationsUniverse

using lm077 lm080 subset-inj-on by blast

corollary lm082:

inj-on $\text{pseudoAllocation}$ $(\text{allAllocations} \ N \ G)$

proof –

have $\text{allAllocations} \ N \ G \subseteq \text{allocationsUniverse}$ by (metis no-types allAllocationsUniverse)

thus inj-on $\text{pseudoAllocation}$ $(\text{allAllocations} \ N \ G)$ 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 \neq \{} finite N distinct G set G \neq \{}
  shows winningAllocationsRel N (set G) bids \cap set (allAllocationsAlg N G) \neq \{}
  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 \subseteq 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 \in X) − \{'True\}'
  by blast

corollary lm086:
  assumes N \neq \{} finite N distinct G set G \neq \{}
  shows (%x. x \in winningAllocationsRel N (set G) bids) − \{'True\}' \cap set (allAllocationsAlg N G) \neq \{}
  using assms lm084 lm085 by metis

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

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

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

corollary lm090:
assumes $N \neq \{\} \text{ finite } N \text{ distinct } G \text{ set } G \neq \{\} \\
shows \text{chosenAllocation } N \text{ bids random } \in \text{winningAllocationsRel } N \text{ (set } G) \text{ bids} \\
proof
have $\forall x_1 \ b \ x. \text{ set } x_1 = \{\}$ \\
  $\forall (\text{randomEl } x_1 \ b \ x:\{a \times \ b \text{ set}\} \ x) \in x$ \\
  $\forall \text{ set } x_1 \subseteq x \text{ by (metis (no-types) randomElLemma subsetCE)}$ \\
thus \text{winningAllocationRel } N \text{ (set } G) \\
  $((\in) (\text{randomEl } (\text{takeAll } (\lambda x. \text{winningAllocationRel } N \text{ (set } G) \ ((\in) x) \text{ bids}) \text{ (allAllocationsAlg } N \text{ } G) \text{ random}) \text{ bids})$ \\
  by (metis lm088 assms(1) assms(2) assms(3) assms(4) takeAllSubset set-empty) \\
qed

lemma lm091:
assumes finite $G$ $a \in \text{allAllocations } N \text{ } G \text{ aa } \in \text{allAllocations } N \text{ } G$ \\
shows $\text{real}(\text{sum}(\text{maxbid } a \text{ } N \text{ } G)(\text{pseudoAllocation } a)) - \text{sum}(\text{maxbid } a \text{ } N \text{ } G)(\text{pseudoAllocation } aa) = \text{real}(\text{card } G) - \text{card } (\text{pseudoAllocation } aa \cap (\text{pseudoAllocation } a))$ \\
proof
let $?p = \text{pseudoAllocation}$ \\
let $?f = \text{finestpart}$ \\
let $?m = \text{maxbid}$ \\
let $?B = \text{?m a N G}$ \\
have $?p aa \subseteq N \times (?f a \cup (N \times ?f G)) \text{ by auto}$ \\
moreover have finite $?p aa \text{ using assms lm034 lm040 by blast}$ \\
ultimately have $\text{real}(\text{sum } ?B (\text{?p a})) - \text{sum } ?B (\text{?p aa}) = \text{real}(\text{card } (?p a)) - \text{card}(\text{?p aa } \cap (\text{?p a}))$ \\
  using differenceSumVsCardinalityReal by fast \\
moreover have ... = real (\text{card } G) - \text{card } (?p aa \cap (\text{?p a}))$ \\
  using assms lm034 by (metis (lifting, mono-tags)) \\
ultimately show $\text{thesis by simp}$ \\
qed

lemma lm092:
$\text{summedBidVectorRel } \text{bids } N \text{ } G = \text{graph } (N \times (\text{Pow } G - \{\{\}\})) \text{ (summedBidSecond bids)}$ \\
unfolding graph-def using lm016 by blast

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

corollary lm094:
assumes pair $\in N \times (\text{Pow } G - \{\{\}\})$
shows summedBidVector bids N G pair = summedBidSecond bids pair
using assms lm093 lm092 by (metis (mono-tags))

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

lemma lm096:
assumes pair ∈ N × (Pow G - {}) shows summedBidVector (real o (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 o (bids:: - => nat)) N G pair =
  (real o summedBidVector bids N G) pair
proof -
  fix esk48 _ :: 'a × 'b set
  { assume esk48 ∈ N × (Pow G - {})
    hence summedBidVector (real o bids) N G esk48 = real (summedBidVector bids N G esk48) using lm096 by blast
    hence esk48 ∉ X ∨ summedBidVector (real o bids) N G esk48 = (real o summedBidVector bids N G) esk48 by simp }
  hence esk48 ∉ X ∨ summedBidVector (real o bids) N G esk48 = (real o summedBidVector bids N G) esk48 using assms by blast }
  thus ∀ pair ∈ X. summedBidVector (real o bids) N G pair = (real o summedBidVector bids N G) pair by blast
qed

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

corollary lm099:
assumes aa ∈ allAllocations N G

111
shows \( \text{sum} \{ \text{summedBidVector} \circ (\text{bids} :: \text{= nat}) \} \) \( \text{N G} \) \( \text{aa} = \) \( \text{real} \{ \text{sum} \{ \text{summedBidVector \ bidders N G} \} \} \) \( \text{aa} \)

using assms lm098 allocationPowerset by \((\text{metis(lifting,mono-tags)})\)

corollary lm100:
assumes finite \( \text{G a} \in \text{allAllocations N G} \) \( \text{aa} \in \text{allAllocations N G} \)
shows \( \text{real} \{ \text{sum} \{ \text{tiebids a N G} \} \} \) \( \text{a} = \) \( \text{real} \{ \text{card} \{ \text{pseudoAllocation aa} \} \cap \{ \text{pseudoAllocation a} \} \} \)
(is \( \text{L} = \text{R} \))

proof –
let \( \text{?l= summedBidVector} \)
let \( \text{?m=maxbid} \)
let \( \text{?p=pseudoAllocation} \)
let \( \text{?bb=?m a N G} \)
let \( \text{?b=real o (?m a N G)} \)

have \( \text{real} \{ \text{sum} \{ ?l \} \} \) \( \text{?b=real o ( ?m a N G)} \)
then have
\( \text{1: } \text{?R} = \text{real} \{ (?s ?bb (?p a)) \} \) \( \text{=} \) \( (?s ?bb (?p aa)) \) by simp

have \( \text{?a (} ?l ?b N G \text{) aa } = \text{?s (} ?l \) \text{ by simp) using assms lm070 by blast} \)

ultimately have
\( \text{2: } \text{?R } = \text{real} \{ (?s ?bb (?p a)) \} \) \( \text{=} \) \( (?s ?bb (?p aa)) \) by \( \text{metis(1)} \)

have \( \text{?s (} ?l ?b N G \text{) aa } = \text{?s (} ?l ?bb N G \text{) aa} \) by \( \text{simp(2)} \)

by \( \text{rule lm099} \)
ultimately have \( \text{?s (} ?l ?bb N G \text{) a } = \text{real} \{ (?s ?bb (?p a)) \} \) by \( \text{simp(2)} \)
thus \( \text{thesis using 2 by simp}\)
 qed

corollary lm101:
assumes finite \( \text{G a} \in \text{allAllocations N G} \) \( \text{aa} \in \text{allAllocations N G} \)

shows \( x \leq \text{card} \{ \text{G} \} \) \&
\( x \geq 0 \) \&
\( (x=0 \iff a = aa) \) \&
\( (aa \neq a \implies \text{sum} \{ \text{tiebids a N G} \} a < \text{sum} \{ \text{tiebids a N G} \} a) \)

proof –
let \( \text{?p=pseudoAllocation} \)

have \( \text{real} \{ \text{card} \{ \text{G} \} \} \geq \text{real} \{ \text{card} \{ \text{G} \} \} \) \( \text{=} \) \( \text{card} \{ \text{?p aa} \} \cap \{ \text{?p a} \} \) by \( \text{force} \)

moreover have \( \text{real} \{ \text{sum} \{ \text{tiebids a N G} \} \} \) \( \text{a} = \) \( \text{real} \{ \text{card} \{ \text{pseudoAllocation aa} \} \cap \{ \text{pseudoAllocation a} \} \} \)
using \( \text{assms lm100 by blast} \)

ultimately have

112
1: \( x = \text{real(card } G \text{)} - \text{card(pseudoAllocation } aa \cap \text{pseudoAllocation } a) \) using assms by force

then have
2: \( x \leq \text{real(card } G \text{)} \) using assms by linarith

have
3: \( \text{card } (\text{?p aa} \cap (?p a)) = \text{card } G \) using assms \( \text{lm034} \) by blast

moreover have \( \text{finite } (\text{?p aa}) \) \& \( \text{finite } (\text{?p a}) \) using assms \( \text{lm040} \) by blast

ultimately have \( \text{card } (\text{?p aa} \cap ?p a) \leq \text{card } G \) using \text{Int-lower2 card-mono} by fastforce

then have
4: \( x \geq 0 \) using assms \( \text{lm100 1} \) by linarith

have
5: \( \text{card } (\text{?p aa} \cap (\text{?p a})) = \text{card } G \) using assms \( \text{lm010} \) by blast

moreover have \( \text{finite } (\text{?p aa}) \) \& \( \text{finite } (\text{?p a}) \) using assms \( \text{lm040} \) by blast

ultimately have \( \text{card } (\text{?p aa} \cap ?p a) = \text{card } G \) using \( \text{Int-lower2 card-mono} \) by fastforce

qed

corollary \( \text{lm102} \):
assumes \( \text{finite } G \) \( a \in \text{allAllocations } N \) \( G \)
\( \text{aa} \in \text{allAllocations } N \) \( G \) \( \text{aa} \neq a \)
shows \( \text{sum (tiebids } a \ N \ G \) \( \text{aa} < \text{sum (tiebids } a \ N \ G \) \( a \)
using assms \( \text{lm101} \) by blast

lemma \( \text{lm103} \):
assumes \( \text{N} \neq \{\} \) \( \text{finite } N \) \( \text{distinct } G \) \( \text{set } G \neq \{\} \)
\( \text{aa} \in (\text{allAllocations } N \ (\text{set } G))\)\(\text{\{-chosenAllocation } N \ G \text{ bids random}\})
shows \( \text{sum (resolvingBid } N \ G \text{ bids random) } aa < \text{sum (resolvingBid } N \ G \text{ bids random) } (\text{chosenAllocation } N \ G \text{ bids random}) \)
proof

let \( ?a=\text{chosenAllocation } N \ G \text{ bids random} \)
let \( ?p=\text{allAllocations} \)
let \( ?G=\text{set } G \)

have \( ?a \in \text{winningAllocationsRel } N \ (\text{set } G) \) bids using assms \( \text{lm090} \) by blast

moreover have \( \text{winningAllocationsRel } N \ (\text{set } G) \subseteq ?p \ N \ ?G \) using assms \( \text{winningAllocationPossible} \) by metis

ultimately have \( ?a \in ?p N ?G \) using \( \text{lm090 assms winningAllocationPossible} \) \( \text{rev-subsetD} \) by blast

then show \( ?\text{thesis using assms lm102 by blast} \)
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 \(?p = \text{allAllocations}\)
- Let \(?G = \text{set } G\)
- Let \(?X = \text{argmax } \text{(sum bids) } ?p \ N \ ?G\)
- Let \(?a = \text{chosenAllocation } N \ G \) bids random
- Let \(?b = \text{resolvingBid } N \ G \) bids random
- Let \(?f = \text{sum } ?b\)
- Let \(?t = \text{terminatingAuctionRel}\

  \begin{align*}
  \text{have } \forall aa \in (\text{allAllocations } N \ ?G) - \{?a\}. \ ?f aa < ?f \ ?a \\
  & \quad \text{using assms lm103 by blast} \\
  \text{then have } \forall aa \in ?X - \{?a\}. \ ?f aa < ?f \ ?a \text{ using assms lm103 by auto} \\
  \text{moreover have } \text{finite } N \ \text{using assms by simp} \\
  \text{then have } \text{finite } \ ?p \ N \ ?G \ \text{using assms allAllocationsFinite by (metis List.finite-set)} \\
  \text{then have } \text{finite } \ ?X \ \text{using assms by (metis finite-subset winningAllocationPossible)} \\
  \text{moreover have } \ ?a \in ?X \ \text{using lm090 assms by blast} \\
  \text{ultimately have } \text{finite } \ ?X \ \& \ ?a \in ?X \ \& \ (\forall aa \in ?X - \{?a\}. \ ?f aa < ?f \ ?a) \ \text{by force} \\
  \text{moreover have } (\text{finite } ?X \ \& \ ?a \in ?X \ \& \ (\forall aa \in ?X - \{?a\}. \ ?f aa < ?f \ ?a)) \longrightarrow \\
  \text{argmax } ?f \ ?X = \{?a\} \\
  \quad \text{by (rule argmaxProperty)} \\
  \text{ultimately have } \{?a\} = \text{argmax } ?f \ ?X \ \text{using injectionsFromEmptyIsEmpty by presburger} \\
  \text{moreover have } ... = ?t \ N \ G \ \text{bids random by simp} \\
  \text{ultimately show } \text{thesis by simp} \\
\end{align*}

**QED**

The computable variant of Else is defined next as Elsee.

**Definition** toFunctionWithFallbackAlg \( R \) fallback ==

\[
\text{(% x. if } (x \in \text{Domain } R) \text{ then } (R,,x) \text{ else fallback)}
\]

**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 == \( (0::\text{integer}) \)

abbreviation allAllocations’ \( N \ \Omega == \) injectionsUniverse \( \cap \{ a. \text{Domain } a \subseteq N \ & \text{Range } a \in \text{all-partitions } \Omega \}\)

abbreviation allAllocations” \( N \ \Omega == \) allocationsUniverse \( \cap \{ a. \text{Domain } a \subseteq N \ & \bigcup (\text{Range } a) = \Omega \}\)

lemma allAllocationsEquivalence:
allAllocations \( N \ \Omega = \) allAllocations’ \( N \ \Omega \ & \) allAllocations \( N \ \Omega = \) allAllocations” \( N \ \Omega \)
using allocationInjectionsUniverseProperty allAllocationsIntersection by metis

lemma allAllocationsVarCharacterization:
\( (a \in \text{allAllocations” } N \ \Omega) = (a \in \text{allocationsUniverse}\& \text{Domain } a \subseteq N \ & \bigcup (\text{Range } a) = \Omega) \)
by force

abbreviation soldAllocations \( N \ \Omega == (\text{Outside’ } \{\text{seller}\}) \cdot (\text{allAllocations } (N \cup \{\text{seller}\}) \ \Omega) \)

abbreviation soldAllocations’ \( N \ \Omega == (\text{Outside’ } \{\text{seller}\}) \cdot (\text{allAllocations’ } (N \cup \{\text{seller}\}) \ \Omega) \)

abbreviation soldAllocations” \( N \ \Omega == (\text{Outside’ } \{\text{seller}\}) \cdot (\text{allAllocations” } (N \cup \{\text{seller}\}) \ \Omega) \)

abbreviation soldAllocations’’ \( N \ \Omega == \) allocationsUniverse \( \cap \{ a.a. \text{Domain } a \subseteq N - \{\text{seller}\} \ & \bigcup (\text{Range } a) \subseteq \Omega \}\)

lemma soldAllocationsEquivalence:
\[ \text{soldAllocations} N \Omega = \text{soldAllocations}' N \Omega \& \\text{soldAllocations}' N \Omega = \text{soldAllocations}'' N \Omega \]

using allAllocationsEquivalence by metis

corollary soldAllocationsEquivalenceVariant:
\[ \text{soldAllocations} = \text{soldAllocations}' \& \\text{soldAllocations}' = \text{soldAllocations}'' \& \\text{soldAllocations} = \text{soldAllocations}'' \]

using soldAllocationsEquivalence by metis

lemma allocationSellerMonotonicity:
\[ \text{soldAllocations} (N - \{\text{seller}\}) \Omega \subseteq \text{soldAllocations} N \Omega \]

using Outside-def by simp

lemma allocationsUniverseCharacterization:
\[ (a \in \text{allocationsUniverse}) = (a \in \text{allAllocations}'' (\text{Domain} a) (\bigcup (\text{Range} a))) \]

by blast

lemma allocationMonotonicity:
\[ \text{assumes } N1 \subseteq N2 \]
\[ \text{shows } \text{allAllocations}'' N1 \Omega \subseteq \text{allAllocations}'' N2 \Omega \]

using assms by auto

lemma allocationWithOneParticipant:
\[ \text{assumes } a \in \text{allAllocations}'' N \Omega \]
\[ \text{shows } \text{Domain} (a \-- x) \subseteq N - \{x\} \]

using assms Outside-def by fastforce

lemma soldAllocationIsAllocation:
\[ \text{assumes } a \in \text{soldAllocations} N \Omega \]
\[ \text{shows } a \in \text{allocationsUniverse} \]

proof –
obtain aa where a =aa \-- seller \& aa \in \text{allAllocations} (N \cup \{\text{seller}\}) \Omega 

using assms by blast
then have a \subseteq aa \& aa \in \text{allocationsUniverse} 

unfolding Outside-def using allAllocationsIntersectionSubset by blast
then show ?thesis using subsetAllocation by blast

qed

lemma soldAllocationIsAllocationVariant:
\[ \text{assumes } a \in \text{soldAllocations} N \Omega \]
\[ \text{shows } a \in \text{allAllocations}'' (\text{Domain} a) (\bigcup (\text{Range} a)) \]

proof –
show ?thesis using assms soldAllocationIsAllocation 

by auto blast+

qed

lemma onlyGoodsAreSold:
\[ \text{assumes } a \in \text{soldAllocations}'' N \Omega \]

116
shows $\bigcup \text{ (Range } a \big) \subseteq \Omega$
using assms Outside-def by blast

**lemma** soldAllocationIsRestricted:
a $\in$ soldAllocations $''$ $N \Omega =$
$(\exists a a. a a = (\text{seller}) = a \land a a \in \text{allAllocations}'' (N \cup \{\text{seller}\}) \Omega)$
by blast

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

**lemma** allocatedToBuyerMeansSold:
assumes a $\in$ allocationsUniverse Domain a $\subseteq$ $N - \{\text{seller}\}$ $\cup$ (Range a) $\subseteq$ $\Omega$
shows a $\in$ soldAllocations $''$ $N \Omega$
proof
let $?i = \text{seller}$
let $?Y = \{\Omega - \bigcup \text{ (Range a)}\} - \{\{\}\}$
let $?b = \{?i\} \times $?Y
let $?aa = a + ?b
let $?aa' = a + * ?b
have
1: a $\in$ allocationsUniverse using assms(1) by fast
have $?b \subseteq \{(?i, \Omega - \bigcup \text{ (Range a)})\} - \{(?i, \{\})\}$ by fastforce
then have
2: $?b \in$ allocationsUniverse
using allocationUniverseProperty subsetAllocation by (metis(no-types))
have
3: $\bigcup \text{ (Range } a) \cap \bigcup \text{ (Range } ?b) = \{\}$ by blast
have
4: Domain a $\cap$ Domain ?b = {} using assms by fast
have $?aa \in$ allocationsUniverse using 1 2 3 4 by (rule allocationUnion)
then have $?aa \in$ allAllocations $''$ (Domain ?aa) $\bigcup$ (Range $?aa)$
unfolding allocationsUniverseCharacterization by metis
then have $?aa \in$ allAllocations $''$ (N $\cup$ {?i}) $\bigcup$ (Range ?aa)
using allocationMonotonicity assms paste-def by auto
moreover have Range $?aa = Range a \cup \ ?Y$ by blast
then moreover have $\bigcup \text{ (Range } ?aa) = \Omega$
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 $?aa' \in$ allAllocations $''$ (N $\cup$ {?i}) $\Omega$ by simp
moreover have Domain $?b \subseteq \{?i\}$ by fast
have $?aa'' = ?i = a = - - ?i$ by (rule restrictionConservation)
moreover have ... $=$ a using Outside-def assms(2) by auto
ultimately show $?\text{thesis using soldAllocationIsRestricted}$ by auto
qed
lemma allocationCharacterization:
\[ a \in \text{allAllocations} N \Omega = \]
\[ (a \in \text{injectionsUniverse} \& \text{Domain } a \subseteq N \& \text{Range } a \in \text{all-partitions } \Omega) \]
by (metis (full-types) possibleAllocationsRelCharacterization)

lemma lm01:
assumes \( a \in \text{soldAllocations}'' N \Omega \)
shows \( \text{Domain } a \subseteq N - \{\text{seller}\} \& a \in \text{allocationsUniverse} \)
proof
- let \(?i = \text{seller}\)
  obtain \(aa\) where
  \(\emptyset: a = aa \longrightarrow ?i \& aa \in \text{allAllocations}'' (N \cup \{ ?i \}) \Omega\)
  using assms(1) soldAllocationIsRestricted by blast
then have \(\text{Domain } aa \subseteq N \cup \{ ?i \}\) using allocationCharacterization by blast
then have \(\text{Domain } a \subseteq N - \{ ?i \}\) using \(\emptyset\) Outside-def by blast
moreover have \(a \in \text{soldAllocations} N \Omega\) using assms soldAllocationsEquivalenceVariant by metis
then moreover have \(a \in \text{allocationsUniverse}\) using soldAllocationIsAllocation by blast
ultimately show \(?\text{thesis}\) by blast
qed

corollary lm02:
assumes \( a \in \text{soldAllocations}'' N \Omega \)
shows \(a \in \text{allocationsUniverse} \& \text{Domain } a \subseteq N - \{\text{seller}\} \& \bigcup (\text{Range } a) \subseteq \Omega\)
proof
- have \(a \in \text{allocationsUniverse}\) using assms lm01 [of a] by blast
moreover have \(\text{Domain } a \subseteq N - \{\text{seller}\}\) using assms lm01 by blast
moreover have \(\bigcup (\text{Range } a) \subseteq \Omega\) using assms soldOnlyGoodsAreSold by blast
ultimately show \(?\text{thesis}\) by blast
qed

corollary lm03:
\( (a \in \text{soldAllocations}'' N \Omega) = \)
\[ (a \in \text{allocationsUniverse} \& a \in \{aa. \text{Domain } aa \subseteq N - \{\text{seller}\} \& \bigcup (\text{Range } aa) \subseteq \Omega\}) \]
(is \(?L = ?R\))
proof
- have \( (a \in \text{soldAllocations}'' N \Omega) = \)
  \( (a \in \text{allocationsUniverse} \& \text{Domain } a \subseteq N - \{\text{seller}\} \& \bigcup (\text{Range } a) \subseteq \Omega) \)
  using lm02 allocatedToBuyerMeansSold by (metis (mono-tags))
then have \(?L = (a \in \text{allocationsUniverse} \& \text{Domain } a \subseteq N - \{\text{seller}\} \& \bigcup (\text{Range } a) \subseteq \Omega)\) by fast
moreover have \(\ldots = ?R\) using mem-Collect-eq by (metis (lifting, no-types))
ultimately show \(?\text{thesis}\) by auto
qed
corollary lm04:
\[ a \in \text{soldAllocations}'' N \Omega = \left( a \in (\text{allocationsUniverse} \cap \{aa.\ \text{Domain} aa \subseteq N-\{\text{seller}\} \& \bigcup (\text{Range} aa) \subseteq \Omega\}) \right) \]
using lm03 by (metis (mono-tags) Int-iff)

corollary soldAllocationVariantEquivalence:
\[ \text{soldAllocations}'' N \Omega = \text{soldAllocations}''' N \Omega \]
is ?L=?R

proof -
{ 
  fix a
  have \(a \in ?L = (a \in ?R)\) by (rule lm04)
}
thus \(?thesis by blast\)
qed

lemma lm05:

assumes \(a \in \text{soldAllocations}'''' N \Omega\)

shows \(a \leftarrow \text{soldAllocations}'''' (N-\{n\}) \Omega\)

proof -
let ?bb = seller
let ?d = Domain
let ?r = Range
let \(?X1 = \{aa.\ \text{?d} aa \subseteq N-\{n\}-\{?bb\} \& \bigcup (?r aa) \subseteq \Omega\}\)
let \(?X2 = \{aa.\ \text{?d} aa \subseteq N-\{?bb\} \& \bigcup (?r aa) \subseteq \Omega\}\)

have \(a \in ?X2\) using assms(1) by fast
then have \(\emptyset: \text{?d} a \subseteq N-\{?bb\} \& \bigcup (?r a) \subseteq \Omega\) by blast
then have \(\text{?d} (a\leftarrow n) \subseteq N-\{?bb\}-\{n\}\)
  using outside-reduces-domain by (metis Diff-mono subset-refl)
moreover have ... = \(N-\{n\}-\{?bb\}\) by fastforce
ultimately have \(\text{?d} (a\leftarrow n) \subseteq N-\{n\}-\{?bb\}\) by blast
moreover have \(\bigcup (?r (a\leftarrow n)) \subseteq \Omega\)
  unfolding Outside-def using \(\emptyset\) by blast
ultimately have \(a \leftarrow n \in ?X1\) by fast
moreover have \(a\leftarrow n \in \text{allocationsUniverse}\)
  using assms(1) Int-iff allocationsUniverseOutside by (metis(lifting,mono-tags))
ultimately show \(?thesis by blast\)
qed

lemma allAllocationsEquivalenceExtended:
\[ \text{soldAllocations} = \text{soldAllocations}' \&\]
\[ \text{soldAllocations}' = \text{soldAllocations}'' \&\]
\[ \text{soldAllocations}'' = \text{soldAllocations}''' \]
using soldAllocationVariantEquivalence soldAllocationsEquivalenceVariant by metis
corollary soldAllocationRestriction:
assumes $a \in \text{soldAllocations } N \Omega$
shows $a \leftarrow n \in \text{soldAllocations } (N - \{n\}) \Omega$
proof
let $?A' = \text{soldAllocations}''$
have $a \in ?A' \Omega : \text{assms allAllocationsEquivalenceExtended by metis}$
then have $a \leftarrow n \in ?A' (N - \{n\}) \Omega$ by (rule bm05)
thus $?thesis$ using allAllocationsEquivalenceExtended by metis
qed

corollary allocationGoodsMonotonicity:
assumes $\Omega_1 \subseteq \Omega_2$
shows $\text{soldAllocations}'' N \Omega_1 \subseteq \text{soldAllocations}'' N \Omega_2$
using assms by blast

corollary allocationGoodsMonotonicityVariant:
assumes $\Omega_1 \subseteq \Omega_2$
shows $\text{soldAllocations}'' N \Omega_1 \subseteq \text{soldAllocations}'' N \Omega_2$
proof
have $\text{soldAllocations}'' N \Omega_1 = \text{soldAllocations}'' N \Omega_1$ by (rule soldAllocationVariantEquivalence)
moreover have $\ldots \subseteq \text{soldAllocations}'' N \Omega_2$
using assms(1) by (rule allocationGoodsMonotonicity)
moreover have $\ldots = \text{soldAllocations}'' N \Omega_2$ using soldAllocationVariantEquivalence by metis
ultimately show $?thesis$ by auto
qed

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

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

abbreviation vcgas $N \Omega b r ==$ 
Outside' $\{\text{seller}\}$ $\left(\text{argmax} \circ \text{sum} \right)$ (randomBids $N \Omega b r$
$\left(\text{argmax} \circ \text{sum} \right)b (\text{allAllocations } (N \cup \{\text{seller}\}) (\text{set } \Omega)))$

abbreviation vcga $N \Omega b r == \text{the-elem } (\text{vcgas } N \Omega b r)$

abbreviation vcga' $N \Omega b r ==$ 
(\text{the-elem } \text{argmax } (\text{sum } (\text{randomBids } N \Omega b r))$
(maximalStrictAllocations \ N \ (set \ \Omega) \ b))

\[\text{-- seller}\]

**Lemma lm06:**

**Assumes** card ((argmax\circ\text{sum}) (\text{randomBids} \ N \ \Omega \ b \ r)
((argmax\circ\text{sum}) \ b \ (\text{allAllocations} \ (N\cup\{\text{seller}\}) \ (set \ \Omega))))

= 1

**Shows** vega \ N \ \Omega \ b \ r = (\text{the-elem} \ ((argmax\circ\text{sum}) (\text{randomBids} \ N \ \Omega \ b \ r)
((argmax\circ\text{sum}) \ b \ (\text{allAllocations} \ \Omega)))))

\[\text{-- seller}\]

**Using** assms cardOneTheElem by auto

**Corollary lm07:**

**Assumes** card ((argmax\circ\text{sum}) (\text{randomBids} \ N \ \Omega \ b \ r)
((argmax\circ\text{sum}) \ b \ (\text{allAllocations} \ (N\cup\{\text{seller}\}) \ (set \ \Omega))))

= 1

**Shows** vega \ N \ \Omega \ b \ r = vega' \ N \ \Omega \ b \ r
(is \ ?l = ?r)

**Proof**

- **Have** \ ?l = (\text{the-elem} \ ((argmax\circ\text{sum}) (\text{randomBids} \ N \ \Omega \ b \ r)
((argmax\circ\text{sum}) \ b \ (\text{allAllocations} \ \Omega)))))

\[\text{-- seller}\]

**Using** assms by (rule lm06)

**Moreover have** ... = ?r by force

**Ultimately show** ?thesis by blast

**Qed**

**Lemma lm08:**

**Assumes** distinct \ \Omega \ set \ \Omega \ \neq \ {} \ finite \ N

**Shows** card ((argmax\circ\text{sum}) (\text{randomBids} \ N \ \Omega \ bids \ random)
((argmax\circ\text{sum}) \ bids \ (\text{allAllocations} \ (N\cup\{\text{seller}\}) \ (set \ \Omega))))) = 1

(is \ card \ ?l=\cdot)

**Proof**

- **Let** \ ?N = N\cup\{\text{seller}\}

- **Let** \ ?b' = \text{randomBids} \ N \ \Omega \ bids \ random

- **Let** \ ?s = \text{sum}

- **Let** \ ?a = \text{argmax}

- **Let** \ ?f = ?a \circ ?s

**Have**

1: \ ?N\neq\{} by auto

**Have**

2: finite \ ?N using assms(3) by simp

**Have** ?a (?s ?b') (?a (?s bids (\text{allAllocations} \ ?N \ (set \ \Omega)))) =
{\text{chosenAllocation} \ ?N \ \Omega \ bids \ random} \ (is \ ?L=\cdot?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 vegaEquivalence:
  assumes distinct Ω set Ω ≠ {} finite N
  shows vega N Ω b r = vega' N Ω b r
  using assms lm07 lm08 by blast

theorem vegaDefiniteness:
  assumes distinct Ω set Ω ≠ {} finite N
  shows card (vegas N Ω b r) = 1
  proof –
    have card ((argmax o sum) (randomBids N Ω b r)
    (argmax o sum) b (allAllocations (N∪{seller}) (set Ω))) =
    1 (is card ?X = -) using assms lm08 by blast
    moreover have (Outside'(seller)) › ?X = vega N Ω b r by blast
    ultimately show ?thesis using cardOneImageCardOne by blast
  qed

lemma vegaDefinitenessVariant:
  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 vegaDefinitenessVariant)

moreover have ?X ⊆ ?R by auto

ultimately show ?thesis using cardinalityOneTheElem by blast

qed

corollary winningAllocationIsMaximalWithoutSeller:

assumes distinct Ω set Ω ≠ {} finite N

shows vega' 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

corollary onlyGoodsAreAllocatedAuxiliary:

assumes distinct Ω set Ω ≠ {} finite N

shows vega' 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 vega N Ω b r ∈ soldAllocations N (set Ω)

(is ?r)

proof –

have vega' N Ω b r ∈ ?r using assms by (rule onlyGoodsAreAllocatedAuxiliary)

then show ?thesis using assms vegaEquivalence 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 vegaIsMaximalAux1:
  assumes distinct Ω set Ω ≠ {} finite N
  shows ∃a. ((a ∈ (maximalStrictAllocations N (set Ω)) b) ∧ (vega' N Ω b r = a -- seller) &
           (a ∈ argmax (sum b) (allAllocations ({seller}∪N) (set Ω))))
  using assms winningAllocationsIsMaximalWithoutSeller by fast

proof –
  let ?n = seller
  let ?s = sum
  let ?a = vega' 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∈Ω)
  using assms(1,2,3) vegaIsMaximalAux1 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:**

- **Axioms**
  - $\Omega$ distinct $\Omega$ is not equal to $\{\}$ finite $N$ $\forall X. \ b (\text{seller}, X) = 0$
- **Shows**
  - $\sum b (\text{vega}' N \Omega b r) = \text{Max} \{\sum b a | a. \ a \in \text{soldAllocations} N (\text{set } \Omega)\}$
- **Using**
  - assms vcgaIsMaximalAux2 by blast

**Corollary vcgaIsAllocationAllocatingGoodsOnly:**

- **Axioms**
  - $\Omega$ distinct $\Omega$ is not equal to $\{\}$ finite $N$
- **Shows**
  - $\text{vega}' N \Omega b r \in \text{allocationsUniverse} \& \bigcup (\text{Range} (\text{vega}' N \Omega b r)) \subseteq \text{set }\Omega$
- **Proof**
  - let $?a = \text{vega}' N \Omega b r$
  - let $?n = \text{seller}$
  - obtain $a$ where
    - $\theta: ?a = a \rightarrow \text{seller} \& a \in \text{maximalStrictAllocations} N (\text{set }\Omega) b$
    - using assms winningAllocationIsMaximalWithoutSeller by blast
  - then moreover have
    - $a \in \text{allAllocations} ((\forall n) \cup N) (\text{set }\Omega)$ by auto
    - moreover have
      - $\text{maximalStrictAllocations} N (\text{set }\Omega) b \subseteq \text{allocationsUniverse}$
        - by (metis (lifting, mono-tags) winningAllocationPossible)
        - $\text{allAllocationsUniverse subset-trans}$
    - ultimately moreover have
      - $?a = a \rightarrow \text{seller} \& a \in \text{allocationsUniverse}$ by blast
    - then have $?a \in \text{allocationsUniverse}$ using $\text{allocationsUniverseOutside}$ by auto
    - moreover have $\bigcup (\text{Range } a) = \text{set }\Omega$ using $\text{allAllocationsIntersectionSetEquals}$
      - $\text{allAllocationsUniverse subset-trans}$
    - then moreover have $\bigcup (\text{Range } ?a) \subseteq \text{set }\Omega$ using $\text{Outside-def 0}$ by fast
    - ultimately show $?\text{thesis}$ using $\text{allocationsUniverseOutside}$ $\text{Outside-def}$ by blast
  qed

**Abbreviation vcgp N Ω b r n ==**

- $\text{Max} (\sum b \cdot (\text{soldAllocations} (N\{-n\}) (\text{set }\Omega)))$
- $\text{−} (\sum b (\text{vega } N \Omega b r \rightarrow n))$

**Theorem vcgpDefiniteness:**

- **Axioms**
  - $\Omega$ distinct $\Omega$ is not equal to $\{\}$ finite $N$
- **Shows**
  - $\exists! y. \text{vcgp } N \Omega b r n = y$
- **Using**
  - assms vcgaDefiniteness by simp

**Lemma soldAllocationsFinite:**

- **Axioms**
  - finite $N$ finite $\Omega$
- **Shows**
  - finite $\text{soldAllocations} N \Omega$
- **Using**
  - assms allAllocationsFinite finite.emptyI finite.imageI

  by metis
The price paid by any participant is non-negative.

**Theorem NonnegPrices:**

- **Assumes** distinct $\Omega$ set $\Omega \neq \{\}$ finite $N$  
- **Shows** $\text{vegp} \ N \ \Omega \ b \ r \ n >= (0::\text{price})$

**Proof** –

- let $\ ?a = \text{vega} \ N \ \Omega \ b \ r$
- let $\ ?A = \text{soldAllocations}$

  - have $\ ?a \in \ ?A \ N \ (\text{set} \ \Omega)$ using assms by (rule onlyGoodsAreAllocated)
  - then have $\ ?a \ -- \ n \in \ ?A \ (N-\{n\}) \ (\text{set} \ \Omega)$ by (rule soldAllocationRestriction)

  - moreover have $\text{finite} \ (\ ?A \ (N-\{n\}) \ (\text{set} \ \Omega))$

  - ultimately have $\text{Max} \ (\ ?f(\ ?A \ (N-\{n\}) \ (\text{set} \ \Omega))) \geq \ ?f \ (\ ?a \ -- \ n)$

  - (is $\ ?L >= \ ?R)$ by (rule maxLemma)

  - then show $\ ?L - \ ?R \ >= \ 0$ by linarith

**Qed**

**Lemma allocationDisjointAuxiliary:**

- **Assumes** $a \in \text{allocationsUniverse} \ \text{and} \ n1 \in \ Domain \ a \ \text{and} \ n2 \in \ Domain \ a \ \text{and} \ n1 \neq \ n2$

  - **Shows** $a\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\·
Nothing outside the set of goods is allocated.

**theorem** OnlyGoodsAllocated:
**assumes** distinct \( \Omega \) set \( \Omega \neq \{\} \) finite \( N \) \( g \in (\text{vega} \ N \ \Omega \ b \ r)\),\( n \)
**shows** \( g \in \text{set} \ \Omega \)
**proof** –
let \( ?a = \text{vega}' \ N \ \Omega \ b \ r \)
**have** \( ?a \in \text{allocationsUniverse} \) **using** \( \text{assms}(1,2,3) \) \( \text{vegaIsAllocationAllocatingGoodsOnly} \) **by** blast
then have 1: \( \text{runiq} \ ?a \) **using** \( \text{assms}(1,2,3) \) **by** blast
have 2: \( n \in \text{Domain} \ ?a \) **using** \( \text{assms} \text{vegaEquivalence} \) **by** fast
with 1 2 have \( ?a,.n \in \text{Range} \ ?a \) **using** \( \text{imageEquivalence} \) **by** fastforce
then have \( g \in \bigcup (\text{Range} \ ?a) \) **using** \( \text{assms} \text{vegaEquivalence} \) **by** blast
moreover have \( \bigcup (\text{Range} \ ?a) \subseteq \text{set} \ \Omega \) **using** \( \text{assms}(1,2,3) \) \( \text{vegaIsAllocationAllocatingGoodsOnly} \) **by** fast
ultimately show \( \text{thesis} \) **by** blast
qed

### 11.2 Computable versions of the VCG formalization

**abbreviation** maximalStrictAllocationsAlg \( N \ \Omega \ b == \)
\( \text{argmax} (\text{sum} \ b) (\text{set} (\text{allAllocationsAlg} (\{\text{seller}\} \cup N) \ \Omega)) \)

**definition** chosenAllocationAlg \( N \ \Omega \ b (r::\text{integer}) == \)
\( (\text{randomEl} (\text{takeAll} (\% x. x \in (\text{argmax } \circ \text{sum}) \ b (\text{set} (\text{allAllocationsAlg} N \ \Omega))) \text{allAllocationsAlg} N \ \Omega)) \)
\( r \)

**definition** maxbidAlg \( a N \ \Omega == (\text{bidMaximizedBy} a N \ \Omega) \text{Else } 0 \)

**definition** summedBidVectorAlg bids \( N \ \Omega == (\text{summedBidVectorRel} \ \text{bids} N \ \Omega) \)
\text{Else } 0

**definition** tiebidsAlg \( a N \ \Omega == \text{summedBidVectorAlg} (\text{maxbidAlg} a N \ \Omega) N \ \Omega \)

**definition** resolvingBidAlg \( N \ \Omega \ \text{bids random} == \)
\( \text{tiebidsAlg} (\text{chosenAllocationAlg} N \ \Omega \ \text{bids random}) N \ (\text{set} \ \Omega) \)

**definition** randomBidsAlg \( N \ \Omega \ b \ \text{random} == \text{resolvingBidAlg} (N \cup \{\text{seller}\}) \ \Omega \ b \text{ random} \)
definition $\text{vcaAlg Without Losers } N \Omega b r ==$
  (the-elem \argmax (\text{sum (randomBidsAlg } N \Omega b r))
  (\text{maximalStrictAllocationsAlg } N \Omega b))

-- seller

abbreviation $\text{addLosers participant set allocation } == (\text{participant set } \times \{ \}) +*$
allocation

definition $\text{vcaAlg } N \Omega b r = \text{addLosers } N (\text{vcaAlg Without Losers } N \Omega b r)$

abbreviation $\text{soldAllocationsAlg } N \Omega ==$
  (Outside' \{seller\}) ' set (\text{allAllocationsAlg } (N \cup \{seller\}) \Omega)

definition $\text{vcgpAlg } N \Omega b r n (\text{winning Allocation}::\text{allocation}) =$
  Max (\text{sum b'} (\text{soldAllocationsAlg } (N - \{n\}) \Omega))
  - (\text{sum b} (\text{winning Allocation} -- n))

lemma functionCompletion:
  assumes $x \in \text{Domain } f$
  shows $\text{toFunction } f x = (f \text{ Elsee } 0) x$
  unfolding $\text{toFunctionWithFallbackAlg-def}$ by (metis assms toFunction-def)

lemma lm09:
  assumes $\text{fst pair } \in N \text{ snd pair } \in \text{Pow } \Omega - \{\}\}$
  shows $\text{sum (%g. (\text{toFunction (bidMaximizedBy a } N \Omega)) (\text{fst pair, g}))$
  (\text{finestpart (snd pair)}) =
  \text{sum (%g. ((\text{bidMaximizedBy a } N \Omega) \text{ Elsee } 0) (\text{fst pair, g}))$
  (\text{finestpart (snd pair)})

proof
  let $?f1 = %g.(\text{toFunction (bidMaximizedBy a } N \Omega))(\text{fst pair, g})$
  let $?f2 = %g.(\text{bidMaximizedBy a } N \Omega) \text{ Elsee } 0)(\text{fst pair, g})$

  fix $g$ assume $g \in \text{finestpart (snd pair)}$
  then have
    $0: g \in \text{finestpart } \Omega$ using assms finestpartSubset by (metis Diff-iff Pow-iff in-mono)
    have $?f1 g = ?f2 g$
  proof
    have $\bigwedge x_1 x_2. (x_1, g) \in x_2 \times \text{finestpart } \Omega \lor x_1 \not\in x_2$ by (metis $0$ mem-Sigma-iff)
    then have $\text{(pseudoAllocation a < | } (N \times \text{finestpart } \Omega)) (\text{fst pair, g})$
      $\text{maxbidAlg a } N \Omega (\text{fst pair, g})$
      unfolding $\text{toFunctionWithFallbackAlg-def}$ maxbidAlg-def
by (metis (no-types) domainCharacteristicFunction UnCI assms(1) toFunction-def)
truth thesis unfolding maxbidAlg-def by blast
qed
)

thus ?thesis using sum.cong by simp
qed

**corollary lml0:**
assumes pair ∈ N × (Pow Ω − {{}})
shows summedBid (toFunction (bidMaximizedBy a N Ω)) pair =
    summedBid ((bidMaximizedBy a N Ω) Else 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 lms09 by blast
qed

**corollary lml11:**
∀ pair ∈ N × (Pow Ω − {{}}).
     summedBid (toFunction (bidMaximizedBy a N Ω)) pair =
     summedBid ((bidMaximizedBy a N Ω) Else 0) pair
using lml0 by blast

**corollary lml12:**
(summedBid (toFunction (bidMaximizedBy a N Ω))) ‘ (N × (Pow Ω − {{}})) =
(summedBid ((bidMaximizedBy a N Ω) Else 0)) ‘ (N × (Pow Ω − {{}}))
proof –
    have ∀ z ∈ ?Z. ?f1 z = ?f2 z by (rule lml11)
    thus ?thesis by (rule functionEquivalenceOnSets)
qed

**corollary lml13:**
summedBidVectorRel (toFunction (bidMaximizedBy a N Ω)) N Ω =
    summedBidVectorRel ((bidMaximizedBy a N Ω) Else 0) N Ω
using lml12 by metis

**corollary maxbidEquivalence:**
summedBidVectorRel (maxbid a N Ω) N Ω =
    summedBidVectorRel (maxbidAlg a N Ω) N Ω
unfolding maxbidAlg-def using lml13 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 
    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'\text{=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=∑ (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 (set Ω) − {{}}))
by (metis PowI allocationPowerset subsetI)

corollary resolvingBidBridgingLemmaVariant1:
  assumes finite N distinct Ω a ∈ maximalStrictAllocations N (set Ω)
  shows sum (randomBids N Ω b r) a = sum (randomBidsAlg 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-eg-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 resolvingBidBridgingLemmaVariant1)
qed
corollary tiebreakingGivesSingleton:
assumes distinct \( \Omega \) set \( \Omega \neq \{\} \) finite \( N \)
shows card (argmax (sum (randomBidsAlg \( N \) \( \Omega \) \( b \) \( r \)))
(maximalStrictAllocations \( N \) (set \( \Omega \)) \( b \))) = 1
proof –
have \( \forall a \in \text{maximalStrictAllocations} \( N \) (set \( \Omega \)) \( b \).
sum (randomBids \( N \) \( \Omega \) \( b \) \( r \)) \( a \) = sum (randomBidsAlg \( N \) \( \Omega \) \( b \) \( r \)) \( a \)
using assms(3,1) resolvingBidBridgingLemmaVariant2 by blast
then have argmax (sum (randomBidsAlg \( N \) \( \Omega \) \( b \) \( r \))) (maximalStrictAllocations
\( N \) (set \( \Omega \)) \( b \) =
argmax (sum (randomBids \( N \) \( \Omega \) \( b \) \( r \))) (maximalStrictAllocations \( N \) (set
\( \Omega \)) \( b \)
using argmaxEquivalence by blast
moreover have card ... = 1 using assms by (rule vegaDefinitenessVariant)
ultimately show ?thesis by simp
qed

theorem maximalAllocationBridgingTheorem:
assumes finite \( N \) distinct \( \Omega \)
shows maximalStrictAllocations \( N \) (set \( \Omega \)) \( b \) = maximalStrictAllocationsAlg \( N \) \( \Omega \) \( b \)
proof –
let \( ?N = \{\text{seller}\} \cup N \)
have card \( ?N > 0 \) using assms(1)
by (metis (full-types) card-gt-0-iff finite-insert insert-is-Un insert-not-empty)
thus ?thesis using assms(2) allAllocationsBridgingLemma by metis
qed

theorem vegaAlgDefinedness:
assumes distinct \( \Omega \) set \( \Omega \neq \{\} \) finite \( N \)
shows card (argmax (sum (randomBidsAlg \( N \) \( \Omega \) \( b \) \( r \))) (maximalStrictAllocationsAlg
\( N \) \( \Omega \) \( b \))) = 1
proof –
have card (argmax (sum (randomBidsAlg \( N \) \( \Omega \) \( b \) \( r \))) (maximalStrictAllocations
\( N \) (set \( \Omega \)) \( b \)) = 1
using assms by (rule tiebreakingGivesSingleton)
moreover have maximalStrictAllocations \( N \) (set \( \Omega \)) \( b \) = maximalStrictAllocationsAlg \( N \) \( \Omega \) \( b \)
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, \text{sorted-list-of-set}(a\_x))} (\text{sorted-list-of-set} \circ \text{Domain}) a)}

abbreviation singleBidConverter x == ((fst x, set ((fst o snd) x)), (snd o snd) x)
definition Bid2funcBid b = set (\map \text{singleBidConverter} b) \text{Elsee} (0::\text{integer})

definition participantsSet b = \text{fst} \cdot (\text{set} b)

definition goodsList b = \text{sorted-list-of-set} (\text{Union} ((\text{set} \cdot \text{fst} \circ \text{snd}) \cdot (\text{set} b)))

definition payments b r n (a::\text{allocation}) =
\text{vcgpAlg} ((\text{participantsSet} b)) (\text{goodsList} b) (\text{Bid2funcBid} b) r n (a::\text{allocation})

export-code \text{vcgaAlg} payments allocationPrettyPrint in \text{Scala} module-name \text{VCG}

file 'VCG−withoutWrapper.scala'

end

References


