Vickrey-Clarke-Groves (VCG) Auctions

M. B. Caminati∗ M. Kerber∗ C. Lange† C. Rowat‡

October 11, 2017

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

∗School of Computer Science, University of Birmingham, UK
†Fraunhofer IAIS and University of Bonn, Germany, and School of Computer Science, University of Birmingham, UK
‡Department of Economics, University of Birmingham, UK
3 Partitions of sets

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

4.1 Evaluating a relation as a function

4.2 Restriction

4.3 Relation outside some set

4.4 Flipping pairs of relations

4.5 Evaluation as a function

4.6 Paste

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

5.1 Right-Uniqueness

5.2 Converse

5.3 Injectivity

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

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

7.1 Facts and notations about relations, sets and functions.

7.2 Ordered relations

7.3 Indicator function in set-theoretical form.

7.4 Lists

7.5 Computing all the permutations of a list

7.6 A more computable version of toFunction.

7.7 Cardinalities of sets.

7.8 Some easy properties on real numbers

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

8.1 Types

8.2 VCG mechanism

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

9.1 Preliminary lemmas

9.2 Definitions of various subsets of UNIV.

9.3 Results about the sets defined in the previous section

9.4 Bridging theorem for injections

9.5 Computable injections
10 Termination theorem for uniform tie-breaking

10.1 Uniform tie breaking: definitions

10.2 Termination theorem for the uniform tie-breaking scheme

10.3 Results on summed bid vectors

10.4 From Pseudo-allocations to allocations

11 VCG auction: definitions and theorems

11.1 Definition of a VCG auction scheme, through the pair \((v\text{g}a, v\text{g}p)\)

11.2 Computable versions of the VCG formalization

12 VCG auction: Scala code extraction

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 ⊆ {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} ⊆ X and trivial X
shows x = 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 ⊆ 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 ∈ X and y: y ∈ 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 | x ∈ S \} = f ' S
by auto

2.4 Big Union

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

lemma Union-member: (∃ S ∈ F . x ∈ S) ⟷ x ∈ \bigcup F
by blast
2.5 Miscellaneous

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

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

lemma lm01: assumes trivial (A \times B)
shows \ (finite (A\times B) \& card A \ast (card B) \leq 1)
using trivial-def assms One-nat-def cartesian-product card-empty card-insert-disjoint empty-if finite.emptyI le0 trivial-implies-finite order-refl subset-singletonD 
by (metis (no-types))

lemma lm02: assumes finite X 
shows trivial X = (card X \leq 1)
using assms One-nat-def card-empty card-insert-if card-mono card-seteq empty-if 
empty-subsetI finite.cases finite.emptyI finite-insert insert-mono 
trivial-def trivial-singleton 
by (metis (no-types))

lemma lm03: shows trivial \{x\}
by (metis order-refl the-elem-eq trivial-def)

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

lemma lm05: assumes \sim trivial X trivial T 
shows X = T \neq {} 
using assms by (metis Diff_iff empty_iff subsetI trivial-subset)

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

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

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

lemma trivial-cartesian: assumes trivial X trivial Y 
shows trivial (X \times Y) 
using assms lm07 One-nat-def Sigma-empty1 Sigma-empty2 card-empty
lemmas trivial-def trivial-empty

by (metis (full-types))

lemma trivial-same: trivial X = (\forall x1 \in X. \forall x2 \in X. x1 = x2)

using trivial-def trivial-imp-no-distinct ex-in-conv insertCI subsetI subset-singletonD

trivial-singleton

by (metis (no-types, hide-lams))

lemma lm08: assumes (Pow X \subseteq \{\},X))

shows trivial X

unfolding trivial-same using assms by auto

lemma lm09: assumes trivial X

shows (Pow X \subseteq \{\},X))

using assms trivial-same by fast

lemma lm10: trivial X = (Pow X \subseteq \{\},X))

using lm08 lm09 by metis

lemma lm11: \{x\} \times UNIV) \cap P = \{x\} \times (P ^{\prime}\{x\})

by fast

lemma lm12: (x,y) \in P = (y \in P ^{\prime}\{x\})

by simp

lemma lm13: assumes inj-on f A inj-on f B

shows inj-on f (A \cup B) = (f^\prime(A-B) \cap (f^\prime(B-A)) = \{\})

using assms inj-on-Un by (metis)

lemma injection-union: assumes inj-on f A inj-on f B (f^\prime(A) \cap (f^\prime(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 \texttt{all-partitions-list} to algorithmically compute this set (then represented as a list). This function is suitable for code generation. We prove the equivalence of the two definition in order to ensure that the generated code correctly implements the original textbook-style definition. For further background on the overall approach, see Caminati, Kerber, Lange, Rowat: Proving soundness of combinatorial Vickrey auctions and generating verified executable code, 2013.

\( P \) is a family of non-overlapping sets.

\textbf{definition} is-non-overlapping

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

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

\textbf{lemma} subset-is-non-overlapping:

\begin{align*}
\text{assumes} & \quad \text{subset: } P \subseteq Q \text{ and} \\
\text{shows} & \quad \text{is-non-overlapping } P \text{ using } \text{subset by fast} \\
\end{align*}

\textbf{proof} –

\{ 
\begin{align*}
\text{fix } X \ Y \text{ assume } X \in P \land Y \in P \\
\text{then have } X \in Q \land Y \in Q \text{ using subset by fast} \\
\text{then have } X \cap Y \neq \{} \iff X = Y \text{ using non-overlapping unfolding} \\
\text{is-non-overlapping-def by force} \\
\end{align*}
\}

\text{then show } \?\text{thesis unfolding is-non-overlapping-def by force} \\
\text{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.

\textbf{lemma} remove-from-eq-class-preserves-disjoint:

\begin{align*}
\text{fixes} & \quad \text{elem::'a} \\
\text{and } X::'a \text{ set} \\
\text{and } P::'a \text{ set set} \\
\text{assumes} & \quad \text{non-overlapping: is-non-overlapping } P \\
\text{and } eq-class: X \in P \\
\text{and } elem: \text{ elem } \in X \\
\text{shows} & \quad X - \{\text{elem}\} \notin P \\
\text{using} & \quad \text{assms } \text{Int-Diff is-non-overlapping-def Diff-disjoint Diff-eq-empty-iff} \\
\text{Int-absorb2 insert-Diff-if insert-not-empty by } & \text{(metis)} \\
\end{align*}

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

\textbf{lemma} non-overlapping-extension1:

\begin{align*}
\text{fixes} & \quad P::'a \text{ set set} \\
\text{and } X::'a \text{ set} \\
\end{align*}
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 $X P \land Z \in$ insert $X P$
  have $Y \cap Z \neq \{\} \leftrightarrow Y = Z$
  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: $Y \in P - \{X\} \land x \in Y$ 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 \ (\text{infix partitions 75})

where is-partition-of \( P \ A = (\bigcup \ P = A \land \text{is-non-overlapping} \ P) \)

No partition of a non-empty set is empty.

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

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

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

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

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

proof
from \( \text{assms} \) 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 \( \text{thesis} \) 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 \( X \ where \ \ast: X \in P \land x \in X \)
by (rule elem-in-partition) blast
moreover {
fix $Y$ assume $Y \subseteq P \land x \in Y$
then have $Y = X$
  using part in-set *
unfolding is-partition-of-def is-non-overlapping-def
by (metis disjoint-iff-not-equal)
}
ultimately show ?thesis by (rule ex1I)
qed

A non-empty set “is” a partition of itself.

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

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

The empty set is a partition of the empty set.

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

Any partition of the empty set is empty.

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

Classical set-theoretical definition of “all partitions of a set $A$”

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

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

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

inserts an element new_el into a specified set S inside a given family of sets

definition insert-into-member :: 'a set set ⇒ 'a set set
  where insert-into-member new-el Sets S = (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 set set ⇒ 'a set 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 Set,
and suppose new-el $\notin$ Set, i.e. \{new-el\} $\notin$ set P, then the following constructs a partition of 'Set $\cup \{\text{new-el}\}' obtained by inserting a new class \{new-el\} and leaving all previous classes unchanged. *)

\[
\{\text{new-el}\} \# P
\]

(*) Let P be a partition of a set Set, and suppose new-el $\notin$ Set, then the following constructs the set of those partitions of 'Set $\cup \{\text{new-el}\}' obtained by inserting new-el into one class of P at a time. *)

(map ((insert-into-member-list new-el P)) P)

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

**lemma** coarser-partitions-with-list-equivalence:

assumes distinct P

shows set (map set (coarser-partitions-with-list new-el P)) = coarser-partitions-with new-el (set P)

**proof** –

have set (map set (coarser-partitions-with-list new-el P)) = set (map ((\{new-el\} # P) # (map ((insert-into-member-list new-el P)) P))) unfolding coarser-partitions-with-list-def ..

also have ... = insert (insert \{new-el\} (set P)) ((set o (insert-into-member-list new-el P)) ' set P)

by simp

also have ... = insert (insert \{new-el\} (set P)) ((insert-into-member new-el \{new-el\} (set P)) ' set P)

using assms insert-into-member-list-equivalence by (metis comp-apply)


qed

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

**lemma** non-overlapping-extension3:

fixes elem: 'a
and P::'a set set
and Q::'a set set

assumes P-non-overlapping: is-non-overlapping P

and new-elem: elem $\notin$ $\bigcup$ P

and Q-coarser: Q $\in$ coarser-partitions-with elem P

shows is-non-overlapping Q

**proof** –

let ?q = insert \{elem\} P

have Q-coarser-unfolded: Q $\in$ insert ?q (insert-into-member elem P ' P)

using Q-coarser

unfolding coarser-partitions-with-def

by fast

show ?thesis

**proof** (cases Q = ?q)

case True
then show ?thesis 
  using P-non-overlapping new-elem non-overlapping-extension1 
  by fastforce
next
  case False
  then have Q ∈ (insert-into-member elem P) ⋈ P using Q-coarser-unfolded by fastforce
  then show ?thesis using non-overlapping-extension2 P-non-overlapping new-elem 
  by fast
  qed
qed

Let P be a partition of a set S, and elem an element (which may or may not be in S already). Then, any member of coarser-partitions-with elem P is a set of sets whose union is S ∪ {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-right insert-Diff-single) 
      with eq-class-in-P have ⋃ insert (eq-class ∪ {elem}) (P − {eq-class}) = ?S 
        ⋃ {elem} 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:

```plaintext
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:

```plaintext
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:

```plaintext
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)
        have X1 ∩ X2 ≠ {} → X1 = X2
          proof
```

17
assume $X_1 \cap X_2 \neq \{\}$
then have $Z_1 \cap Z_2 \neq \{\}$ using $Z_1$-sup $Z_2$-sup by fast
then have $Z_1 = Z_2$ using $Z_1$-in-$P$ $Z_2$-in-$P$ assms unfolding is-non-overlapping-def
by fast
then show $X_1 = X_2$ using $Z_1$-sup $Z_2$-sup by fast
qed
moreover have $X_1 = X_2$ using $Z_1$-in-$P$ $Z_2$-in-$P$ assms unfolding is-non-overlapping-def
by fast
then show $X_1 = X_2$ using $Z_1$-sup $Z_2$-sup by fast
qed
ultimately show $(X_1 \cap X_2 \neq \{\}) \iff X_1 = X_2$ 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 set set
  assumes non-overlapping: is-non-overlapping P
  and elem: elem $\in \bigcup P$
  shows $P \in$ coarser-partitions-with elem (partition-without elem P)
  (is $P \in$ coarser-partitions-with elem ?Q)
proof
let $?remove-elem = \lambda X . X - \{\text{elem}\}$
obtain Y
  where elem-eq-class: elem $\in Y$ and elem-eq-class': Y $\in P$ using elem ..
let $?elem-neq-classes = P - \{Y\}$
have P-wrt-elem: $P = ?elem-neq-classes \cup \{Y\}$ using elem-eq-class' by blast
let $?elem-eq = Y - \{\text{elem}\}$
have Y-elem-eq: $?remove-elem \ Y = \{?elem-eq\}$ by fast
have elem-neq-classes-part: is-non-overlapping ?elem-neq-classes
  using subset-is-non-overlapping non-overlapping
  by blast
have elem-eq-wrt-P: $?elem-eq \in ?remove-elem \ P$ using elem-eq-class' by blast

18
also have \( \ldots = \text{?remove-elem} \ (\text{?elem-neq-classes} \cup \{Y\}) - \{\{}\} \) using 
P-wrt-elem by presburger
also have \( \ldots = \text{?elem-neq-classes} \cup \{\text{?elem-eq} \} - \{\{}\} \)
  using Y-elem-eq elem-neq-classes-id image-Un by metis
finally have Q-wrt-elem: \( ?Q = \text{?elem-neq-classes} \cup \{\text{?elem-eq} \} - \{\{}\} \).

have \( ?\text{elem-eq} = \{\} \lor ?\text{elem-eq} \notin P \)
  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 \( ?\text{elem-eq} \notin P \)
  using non-overlapping no-empty-in-non-overlapping by metis
then have elem-neq-classes: \( ?\text{elem-neq-classes} - \{?\text{elem-eq}\} = ?\text{elem-neq-classes} \)
by fastforce

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

19
by force
then have \{ Y \} \cup ?elem-neq-classes \in insert-into-member elem \?Q ' \?Q using True by blast
then have \{ Y \} \cup ?elem-neq-classes \in coarser-partitions-with elem \?Q unfolding coarser-partitions-with-def by simp
then show ?thesis using P-wrt-elem by simp
qed
$qed$
$qed$

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

definition all-coarser-partitions-with :: 'a \Rightarrow 'a set set set \Rightarrow 'a set set set
where all-coarser-partitions-with elem \( Ps \) = \( \bigcup \) (coarser-partitions-with elem \( 'Ps \))

the list variant of all-coarser-partitions-with

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

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

lemma all-coarser-partitions-with-list-equivalence:
fixes elem::'a
and \( Ps ::'a \) set list list
assumes \( \) distinct: \( \forall \ P \in set Ps \). distinct \( P \)
shows set (map set (all-coarser-partitions-with-list elem \( Ps \))) = all-coarser-partitions-with elem (set (map set \( Ps \)))
(is \( ?\text{list-expr} = ?\text{set-expr} \))
proof
have \( ?\text{list-expr} = set (map set (concat (map (coarser-partitions-with-list elem \( Ps \)) \( Ps \)))) \)
unfolding all-coarser-partitions-with-list-def ..
also have \( \ldots = set \ ' (\bigcup \ x \in (\text{coarser-partitions-with-list elem} \ ' (set \( Ps \)) \cdot \text{set} \ x) \) by simp
also have \( \ldots = set \ ' (\bigcup \ x \in \{ \text{coarser-partitions-with-list elem \( P \mid P \cdot P \in set \( Ps \)} \cdot \text{set} \ x) \)
by (simp add: image-Collect-mem)
also have \( \ldots = \bigcup \ \{ \text{set (map set (coarser-partitions-with-list elem \( P \))) \mid P \cdot P \in set \( Ps \} \cdot \text{set} \ x) \)
by auto
also have \( \ldots = \bigcup \ \{ \text{coarser-partitions-with elem \( (set \ P) \mid P \cdot P \in set \( Ps \)} \}
using distinct coarser-partitions-with-list-equivalence by fast
also have \( \ldots = \bigcup \ (\text{coarser-partitions-with elem} \ ' (set \ ' (set \( Ps \)))) \) by (simp add: image-Collect-mem)
also have \( \ldots = \bigcup \ (\text{coarser-partitions-with elem} \ ' (set (map set \( Ps \))) \) by simp
also have \( \ldots = ?\text{set-expr unfolding all-coarser-partitions-with-def} .. \)
finally show ?thesis .
qed
all partitions of a set (given as list) in form of a set

20
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 x by auto
    from ps-insert have ps = (X ∪ \{x\}) # (removeAll 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: \forall x ∈ set Q. \forall 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
The classical definition \textit{all-partitions} and the algorithmic (constructive) definition \textit{all-partitions-list} are equivalent.

\textbf{lemma} \textit{all-partitions-equivalence}:

\textbf{fixes} \textit{xs}::\texttt{'}a list
\textbf{shows} \textit{distinct} \textit{xs} \iff
\textit{(set} (\textit{map} \textit{set} \textit{(all-partitions-list} \textit{xs})) = 
\textit{all-partitions} \textit{(set} \textit{xs}) \land (\forall \textit{ps} \in \textit{set} \textit{(all-partitions-list} \textit{xs}) . \textit{distinct} \textit{ps}))

\textbf{proof} \textbf{(induct} \textit{xs})
\textbf{case} \textit{Nil}
\textbf{have} \textit{set} (\textit{map} \textit{set} \textit{(all-partitions-list} \textit{[]})) = \textit{all-partitions} \textit{(set} \textit{[])}
\textbf{unfolding} \textit{List.set-simps(1) emptyset-part-emptyset3} \textbf{by} \textbf{simp}

\textbf{moreover have} \textbf{\forall} \textit{ps} \in \textit{set} \textit{(all-partitions-list} \textit{[]}). \textit{distinct} \textit{ps} \textbf{by} \textbf{fastforce}

\textbf{ultimately show} \textbf{?case} \textbf{..}
\textbf{next}
\textbf{case} \textbf{(Cons} \textit{x} \textit{xs})
\textbf{from} \textbf{Cons.prems} \textbf{Cons.hyps}
\textbf{have} \textbf{hyp-equiv}:
\textit{set} (\textit{map} \textit{set} \textit{(all-partitions-list} \textit{xs})) = \textit{all-partitions} \textit{(set} \textit{xs})
\textbf{by} \textbf{simp}

\textbf{from} \textbf{Cons.prems} \textbf{Cons.hyps}
\textbf{have} \textbf{hyp-distinct}:
\textbf{\forall} \textit{ps} \in \textit{set} \textit{(all-partitions-list} \textit{xs}). \textit{distinct} \textit{ps} \textbf{by} \textbf{simp}

\textbf{have} \textbf{distinct-xs}:
\textbf{distinct} \textit{xs} \textbf{using} \textbf{Cons.prems} \textbf{by} \textbf{simp}
\textbf{have} \textbf{x-notin-xs}:
\textbf{x} \notin \textit{set} \textit{xs} \textbf{using} \textbf{Cons.prems} \textbf{by} \textbf{simp}

\textbf{have} \textbf{set} (\textit{map} \textit{set} \textit{(all-partitions-list} \textit{(x} \texttt{#} \textit{xs}))) = \textit{all-partitions} \textit{(set} \textit{(x} \texttt{#} \textit{xs}))
\textbf{proof (rule} \textbf{equalitySubsetI})
\textbf{fix} \textit{P}::\texttt{'}a set
\textbf{let} \textit{?P-without-x} = \textit{partition-without} \textit{x} \textit{P}
\textbf{have} \textbf{P-partitions-exc-x}:
\textbf{\bigcup} \textit{?P-without-x} = \textbf{\bigcup} \textit{?P-without-x} = \textbf{\bigcup}\textit{P} \textbf{by} \textbf{simp}

\textbf{assume} \textit{P} \in \textit{all-partitions} \textit{(set} \textit{(x} \texttt{#} \textit{xs}))
\textbf{then have} \textbf{is-partition-of}:
\textbf{P} \textbf{partitions} \textbf{(set} \textit{(x} \texttt{#} \textit{xs})) \textbf{unfolding} \textbf{all-partitions-def}
\textbf{..}
\textbf{then have} \textbf{is-non-overlapping}: \textbf{is-non-overlapping} \textbf{P} \textbf{unfolding} \textbf{is-partition-of-def}
\textbf{by} \textbf{simp}
\textbf{from} \textbf{is-partition-of} \textbf{have} \textbf{P-covers}:
\textbf{\bigcup} \textbf{P} = \textbf{\bigcup} \textit{P} \textbf{by} \textbf{simp}

\textbf{have} \textbf{?P-without-x partitions} \textbf{(set} \textit{xs})
\textbf{unfolding} \textbf{is-partition-of-def}
\textbf{using} \textbf{is-non-overlapping} \textbf{non-overlapping-without-is-non-overlapping partition-without-covers}
\textbf{P-covers x-notin-xs}
by (metis Diff-insert-absorb List.set-simps(2))
with hyp-equiv have p-list: \( ?P \text{-without-}x \in \text{set} (\text{map \ set} (\text{all-partitions-list \ }xs)) \)
unfolding all-partitions-def by fast
have \( P \in \text{coarser-partitions-with} \ x \ ?P \text{-without-}x \)
  using coarser-partitions-insert-without is-non-overlapping P-covers
  by (metis List.set-simps(2) insertI1)
then have \( P \in \bigcup (\text{coarser-partitions-with} \ x \ \text{set} (\text{map \ all-partitions-list \ }xs)) \)
  using p-list by blast
then have \( P \in \text{all-coarser-partitions-with} x \ (\text{set} (\text{map \ all-partitions-list} \ xs)) \)
unfolding all-coarser-partitions-with-def by fast
then obtain \( Y \) where P-in-Y: \( P \in Y \)
  and Y-coarser: \( Y \in \text{coarser-partitions-with} \ x \ (\text{all-partitions} (\text{set} \ xs)) \).
from P-in-Y Y-coarser have P-wrt-Q: \( P \in \text{coarser-partitions-with} x \ Q \)
  by fast
then have \( Q \in \text{all-partitions} (\text{set} \ xs) \) using Q-part-xs by simp
then have Q partitions (set xs) unfolding all-partitions-def ..
then have is-non-overlapping Q and Q-covers: \( \bigcup Q = \text{set} \ xs \)
  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 \text{-}xs by fast
have \( \bigcup P = \text{set} \ \text{xs} \cup \{x\} \)
using $Q$-covers $P$-in-$Y$ coarser' coarser-partitions-covers by fast
then have $\bigcup P = \text{set } (x \neq xs)$
  using $x$-notin-xs $P$-wrt-$Q$ $Q$-covers
  by (metis List.set-simps(2) insert-is-Un sup-commute)
then have $P$ partitions $(\text{set } (x \neq xs))$
  using $P$-partition unfolding is-partition-of-def by blast
then show $P \in \text{all-partitions } (\text{set } (x \neq xs))$ unfolding all-partitions-def .. qed
moreover have $\forall ps \in \text{set } (\text{all-partitions-list } (x \neq xs)) \cdot \text{distinct } ps$
proof
  fix ps::'a set list assume ps-part: ps $\in \text{set } (\text{all-partitions-list } (x \neq xs))$
  have \text{set } (\text{all-partitions-list } (x \neq xs)) = \text{set } (\text{all-coarser-partitions-with-list } x \ (\text{all-partitions-list } xs))
    by simp
  also have $\ldots = \text{set } (\text{concat } (\text{map } (\text{coarser-partitions-with-list } x) \ (\text{all-partitions-list } xs)))$
    unfolding all-coarser-partitions-with-list-def ..
  also have $\ldots = \bigcup ((\text{set } \circ (\text{coarser-partitions-with-list } x)) \ \circ \ (\text{set } (\text{all-partitions-list } xs)))$
    by simp
  finally have all-parts-unfolded: $\text{set } (\text{all-partitions-list } (x \neq xs)) = \bigcup ((\text{set } \circ (\text{coarser-partitions-with-list } x)) \ \circ \ (\text{set } (\text{all-partitions-list } xs)))$.
with ps-part obtain qs
  where qs: qs $\in \text{set } (\text{all-partitions-list } xs)$
  and ps-coarser: ps $\in \text{set } (\text{coarser-partitions-with-list } x \ qs)$
  using UnionE comp-def imageE by auto
from qs have set qs $\in \text{set } (\text{map } \text{set } (\text{all-partitions-list } (xs)))$ by simp
with distinct-xs hyp-equiv have qs-hyp: set qs $\in \text{all-partitions } (\text{set } xs)$ 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 \textit{distinct} ?xs \RedRightarrow set (map set (all-partitions-list ?xs)) = all-partitions (set ?xs) \land (\forall ps \in set (all-partitions-list ?xs). \textit{distinct} ps); it does not make the auxiliary statement about partitions being distinct lists.

\textbf{theorem} \textit{all-partitions-paper-equiv-alg}:
\begin{itemize}
  \item \textbf{fixes} \textit{xs}::'a list
  \item \textbf{shows} \textit{distinct} \textit{xs} \RedRightarrow set (map set (all-partitions-list \textit{xs})) = all-partitions (set \textit{xs})
  \item \textbf{using} \textit{all-partitions-equivalence}' by blast
\end{itemize}

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}
\begin{itemize}
  \item \textbf{definition} \textit{all-partitions-alg} :: 'a::linorder set \Rightarrow 'a set list list
  \item \textbf{where} \textit{all-partitions-alg} \textit{X} = all-partitions-list (sorted-list-of-set \textit{X})
\end{itemize}

\section{Additional operators on relations, going beyond Relations.thy, and properties of these operators}

\textbf{theory} \textit{RelationOperators}
\begin{itemize}
  \item \textbf{imports} \textit{SetUtils} \textit{HOL-Library.Code-Target-Nat}
\end{itemize}

\begin{itemize}
\item \textbf{4.1 Evaluating a relation as a function}
\item If an input has a unique image element under a given relation, return that element; otherwise return a fallback value.
\item \textbf{fun} \textit{eval-rel-or} :: ('a \times 'b) set \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b
\item \textbf{where} \textit{eval-rel-or} \textit{R} \textit{a} \textit{z} = (let \textit{im} = \textit{R} "" {\textit{a}} in if card \textit{im} = 1 then the-elem \textit{im} else \textit{z})
\end{itemize}

right-uniqueness of a relation: the image of a \textit{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 \textit{\textlambda} functions.

\textbf{definition} \textit{runiq} :: ('a \times 'b) set \Rightarrow bool
\textbf{where} \textit{runiq} \textit{R} = (\forall X. \textit{trivial} X \Rightarrow \textit{trivial} (\textit{R} "" X))

\item \textbf{4.2 Restriction}
\item 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

4.3 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 $\text{Domain } R = \text{insert } x A$
and $x \notin A$
shows $\text{Domain } (R \text{ outside } \{x\}) = A$
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: $P = (P \text{ outside } X) \cup (P || X)$
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: $\text{Range } (R \text{ outside } X) \subseteq R'' (\text{Domain } R - X)$
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 $\text{Range } R \subseteq Y$
shows $\text{Range } (R \text{ outside } X) \subseteq Y$
using assms outside-union-restrict by (metis Range-mono inf-sup-ord (3) subset-trans)

4.4 Flipping pairs of relations

flipping a pair: exchanging first and second component

definition flip where $\text{flip } \text{tup} = (\text{snd } \text{tup}, \text{fst } \text{tup})$

Flipped pairs can be found in the converse relation.

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

Flipping a pair twice doesn’t change it.

lemma flip-flip: $\text{flip } (\text{flip } \text{tup}) = \text{tup}$
by (metis flip-def fst-conv snd-conv surjective-pairing)

Flipping all pairs in a relation yields the converse relation.

lemma flip-conv: $\text{flip } ^{\prime} R = R^{-1}$
proof –
  have $\text{flip } ^{\prime} R = \{ \text{flip } \text{tup } | \text{tup } . \text{tup } \in R \}$ by (metis image-Collect-mem)
  also have $\ldots = \{ \text{tup } . \text{tup } \in R^{-1} \}$ using flip-in-conv by (metis converse-converse flip-flip)
  also have $\ldots = R^{-1}$ by simp
  finally show thesis .
qed
4.5 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.

```plaintext
fun eval-rel :: (′a × 'b) set ⇒ 'a ⇒ 'b  (infix ,, 75)
  where R ,, a = the-elm (R "" {a})
```

4.6 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.

```plaintext
definition paste (infix +* 75)
  where $P +* Q = (P \text{ outside Domain } Q) \cup Q$
```

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

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

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

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

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

```plaintext
lemma paste-outside-restrict: $P = (P \text{ outside } X) +* (P || X)$
proof −
  have $\text{Domain } (P \text{ outside } X) \cap \text{Domain } (P || X) = {}$
    unfolding Outside-def restrict-def by fast
  moreover have $P = P \text{ outside } X \cup P || X$ by (rule outside-union-restrict)
  ultimately show thesis using paste-disj-domains by metis
qed
```

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

```plaintext
lemma paste-Domain: $\text{Domain}(P +* Q) = \text{Domain } P \cup \text{Domain } Q$ unfolding paste-def Outside-def by blast
```

Pasting two relations yields a subrelation of their union.

```plaintext
lemma paste-sub-Un: $P +* Q \subseteq P \cup Q$
  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: \( \text{Range} \ (P \circ Q) \subseteq \text{Range} \ P \cup \text{Range} \ Q \)

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

end

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

**theory** RelationProperties

**imports**

RelationOperators

HOL.Conditionally-Complete-Lattices

begin

5.1 Right-Uniqueness

**lemma** injflip: \( \text{inj-on} \ \text{flip} \ A \)

**by** (metis flip-flip inj-on-def)

**lemma** lm01: \( \text{card} \ P = \text{card} \ (P^\rightarrow 1) \)

**using** card-image flip-conv injflip by metis

**lemma** cardinalityOneTheElemIdentity: \( (\text{card} \ X = 1) = (X = \{\text{the-elem} \ X\}) \)

**by** (metis One-nat-def card-Suc-eq card-empty empty-iff the-elem-eq)

**lemma** lm02: \( \text{trivial} \ X = (X = \{} \lor \text{card} \ X = 1) \)

**using** cardinalityOneTheElemIdentity order-refl subset-singletonD trivial-def trivial-empty

**by** (metis (no-types))

**lemma** lm03: \( \text{trivial} \ P = \text{trivial} \ (P^\rightarrow 1) \)

**using** trivial-def subset-singletonD subset-refl subset-insertI cardinalityOneTheElemIdentity converse-inject

**by** metis

**lemma** restrictedRange: \( \text{Range} \ (P \llcorner X) = P^\llcorner X \)

**unfolding** restrict-def **by** blast

**lemma** doubleRestriction: \( ((P \llcorner X) \llcorner Y) = (P \llcorner (X \cap Y)) \)

**unfolding** restrict-def **by** fast

**lemma** restrictedDomain: \( \text{Domain} \ (R \llcorner X) = \text{Domain} \ R \cap X \)

**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
using Image-empty trivial-empty-or-singleton the-elem-eq
by (metis (no-types))

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

lemma runiq-wrt-eval-rel: runiq R = (∀ 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^c *(f^−1”的 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”的 {y1} ∩ f^−1”的 {y2} ≠ {y}) = (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 \((R^{-1})'' R'' X \subseteq X\)
using assms by (metis converse-converse rightUniqueFunctionAfterInverse)

**lemma** \texttt{lm05}:\nassumes \(\text{inj-on } \text{fst } P\)
shows \(\text{runiq } P\)
unfolding \(\text{runiq-basic}\)
using assms \(\text{fst-conv inj-on-def old.prod.inject}\)
by (metis (no-types))

**lemma** \texttt{rightUniqueInjectiveOnFirst}: \((\text{runiq } P) = (\text{inj-on } \text{fst } P)\)
using rightUniqueInjectiveOnFirstImplication \texttt{lm05} by blast

**lemma** \texttt{disj-Un-runiq}:\nassumes \(\text{runiq } P \text{ runiq } Q \ (\text{Domain } P) \cap (\text{Domain } Q) = \{\}\)
shows \(\text{runiq } (P \cup Q)\)
using assms rightUniqueInjectiveOnFirst \(\text{fst-eq-Domain injection-union}\) by metis

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

**corollary** \texttt{runiq-paste2}:\nassumes \(\text{runiq } Q \text{ runiq } P\)
shows \(\text{runiq } (P ++ Q)\)
using assms \texttt{runiq-paste1} \(\text{subrel-runiq Diff-subset Outside-def}\)
by (metis)

**lemma** \texttt{rightUniqueRestrictedGraph}: \(\text{runiq } \{(x,f \ x) | x. P \ x\}\)
unfolding \(\text{runiq-basic}\) by fast

**lemma** \texttt{rightUniqueSetCardinality}:\nassumes \(x \in \text{Domain } R \text{ runiq } R\)
shows \(\text{card } (R''\{x\})=1\)
using assms \texttt{lm02} DomainE Image-singleton-iff empty-iff
by (metis \texttt{runiq-alt})

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

**lemma** \texttt{Image-runiq-eq-eval}:\nassumes \(x \in \text{Domain } R \text{ runiq } R\)
shows \(R''\{x\} = \{R \ x, x\}\)
using assms \texttt{rightUniqueSetCardinality}
by (metis \texttt{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 . ∃! b . (a, b) ∈ R)

**Using** runiq-basic **By** (metis Domain.I 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)

### 5.2 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 ∈ Domain R

**Shows** R⁻¹ " " (x) = {x}

**Proof** –
have \( \{ x \} \subseteq R^{-1} \vdash R \vdash \{ x \} \) using domain by fast
have trivial \( (R \vdash \{ x \}) \) using runiq domain by (metis runiq-def trivial-singleton)
then have trivial \( (R^{-1} \vdash R \vdash \{ x \}) \)
  using assms runiq-def by blast
then show \(?thesis
  using sup by (metis singleton-sub-trivial-uniq subset-antisym trivial-def)
qed

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

**lemma** disj-Domain-imp-disj-Image:
assumes Domain R ∩ X ∩ Y = {}
assumes runiq R
and runiq \((R^{-1})\)
shows \((R \vdash X) \cap (R \vdash Y) = {}\)
using assms unfolding runiq-basic by blast

**lemma** runiq-converse-paste-singleton:
assumes runiq \((P^{-1})\) \(y \notin (\text{Range } P)\)
shows runiq \(((P +\star \{(x,y)\})^{-1})\)
(is \(?u (\?P^{-1})\))

**proof**
  have \((\?P) \subseteq P \cup \{(x,y)\}\) using assms by (metis paste-sub-Un)
  then have \(?P^{-1} \subseteq P^{-1} \cup \{(x,y)\}^{-1}\) by blast
  moreover have \(\ldots = P^{-1} \cup \{(y,x)\}\) by fast
  moreover have 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 \(?thesis by (metis subrel-runiq)
qed

### 5.3 Injectivity

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

**definition** injections :: \(\text{ar } \text{set } \Rightarrow \text{br } \text{set } \Rightarrow (\text{ar } \times \text{br })\) \text{set set}
  where injections X Y = \{ R : Domain R = X \land \text{Range } R \subseteq Y \land \text{runiq } R \land runiq \((R^{-1})\)\}

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

**fun** injections-alg
  where injections-alg [] Y = [[]]
injections-alg \((x \# xs)\) Y = concat [ | R +\star \{(x,y)\} \: y \leftarrow \text{sorted-list-of-set } (Y \setminus \text{Range } R) ]
lemma Image-within-domain':
  fixes x R
  shows (x ∈ Domain R) = (R "\{x\} \neq \{\})
  by blast
end

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

theory Argmax
imports Main
begin

Structural induction is used in proofs on lists.

lemma structInduct: assumes P [] and ∀ x xs. P (xs) −→ P (x#xs)
  shows P l
  using assms list-nonempty-induct by (metis)

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

fun argmax :: ('a ⇒ 'b::linorder) ⇒ 'a set ⇒ 'a set
  where argmax f A = { x ∈ A . f x = Max (f ' A) }

lemma argmaxLemma: argmax f A = { x ∈ A . f x = Max (f ' A) }
  by simp

lemma maxLemma:
  assumes x ∈ X finite X
  shows Max (f'X) >= f x
  (is ?L >= ?R) using assms
  by (metis (hide-lams, no-types) Max.coboundedI finite-imageI image-eqI)

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

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

lemma argmaxEquivalence:
  assumes ∀x∈X. f x = g x
shows $\text{argmax } f \ X = \text{argmax } g \ X$
using assms argmaxLemma Collect-cong image-cong
by (metis(no-types,lifting))

The arg max of a function over a non-empty set is non-empty.
corollary argmax-non-empty-iff: assumes finite $X \ X \neq \{}$
shows $\text{argmax } f \ X \neq \{}$
using assms Max-in finite-imageI image-is-empty lm01
by (metis(no-types))

The previous definition of argmax operates on sets. In the following we
define a corresponding notion on lists. To this end, we start with defining a
filter predicate and are looking for the elements of a list satisfying a given
predicate; but, rather than returning them directly, we return the (sorted)
list of their indices. This is done, in different ways, by filterpositions and
filterpositions2.
definition filterpositions :: ('a => bool) => 'a list => nat list
where filterpositions P l = map snd (filter (P o fst) (zip l (upt 0 (size l))))
definition filterpositions2 where
filterpositions2 P l = [n. n <- [0..<size l], P l!n]
definition maxpositions where
maxpositions l = filterpositions2 (%x. x >= Max (set l)) l
lemma lm03: maxpositions l = [n. n<-[0..<size l], l!n >= Max(set l)]
unfolding maxpositions-def filterpositions2-def by fastforce
definition argmaxList where argmaxList f l = map (nth l) (maxpositions (map f l))
lemma lm04: [n. n <= l, P n] = [n. n <= l, n \in set l, P n]
proof –

have map (\uu. if P uu then [uu] else []) l =
map (\uu. if uu \in set l then if P uu then [uu] else else [:]) l by simp
thus concat (map (\n. if P n then [n] else else []) l) =
concat (map (\n. if n \in set l then if P n then [n] else else []) l) by presburger
qed
lemma lm05: [n. n <= [0..<m], P n] = [n. n <= [0..<m], n \in set [0..<m], P n]

35
Lemma \textit{lm06}: fixes \(f \ m \ P\)

shows \((\mathit{map} \ f \ \{n . n \leftarrow [0..<m], P \ n\}) = \{ f \ n . n \leftarrow [0..<m], P \ n\}\)

by (induct \(m\)) auto

Lemma \textit{map-commutes-a}: \([f \ n . n \leftarrow [], Q \ (f \ n)] = [x \leftarrow (\mathit{map} \ f \ [])]. \ Q \ x\]

by simp

Lemma \textit{map-commutes-b}: \(\forall x \ xs. ([f \ n . n \leftarrow xs, Q \ (f \ n)] = [x \leftarrow (\mathit{map} \ f \ xs)]. \ Q \ x\]

by simp

Lemma \textit{map-commutes}: fixes \(f::'a \Rightarrow 'b \ fixes \ Q::'b \Rightarrow bool \ fixes \ xs::'a \ list\)

shows \([f \ n . n \leftarrow xs, Q \ (f \ n)] = [x \leftarrow (\mathit{map} \ f \ xs)]. \ Q \ x\]

using \textit{map-commutes-a} \textit{map-commutes-b} \textit{structInduct} by fast

Lemma \textit{lm07}: fixes \(f \ l\)

shows \(\mathit{maxpositions} \ (\mathit{map} \ f \ l) =\)

\[\{n . n \leftarrow [0..<\mathit{size} \ l], f \ (\texttt{!} \ n) \geq \textbf{Max} \ (f\texttt{'} \ (\texttt{set} \ l))\}\]

(is \(\mathit{maxpositions} \ (\texttt{'} \ ?f) = \texttt{-}\))

proof –

have \(\mathit{maxpositions} \ ?fl =\)

\([n . n \leftarrow [0..<\texttt{size} \ ?fl], n \in \texttt{set}[0..<\texttt{size} \ ?fl], ?fl\texttt{'} \ n \geq \textbf{Max} \ (\texttt{set} \ ?fl)]\)

using \textit{lm04} unfolding \texttt{filterpositions2-def} \texttt{maxpositions-def}. 

also have \(\ldots \ =\)

\([n . n \leftarrow [0..<\texttt{size} \ l], (n<\texttt{size} \ l), (\texttt{'} ?fl\texttt{'} \ n \geq \textbf{Max} \ (\texttt{set} ?fl))]\) by simp

also have \(\ldots \ =\)

\([n . n \leftarrow [0..<\texttt{size} \ l], (n<\texttt{size} \ l) \land (f \ (\texttt{!} \ n) \geq \textbf{Max} \ (\texttt{set} \ ?fl))]\)

using \texttt{nth-map} by (metis \{poly-guards-query, hide-lams\}) also have \(\ldots \ =\)

\([n . n \leftarrow [0..<\texttt{size} \ l], (n \in \texttt{set}[0..<\texttt{size} \ l]),(f \ (\texttt{!} \ n) \geq \textbf{Max} \ (\texttt{set} ?fl))]\)

using \texttt{atLeastLessThan_iff \ le0 \ set-upt} by (metis \{no-types\})

also have \(\ldots \ =\)

\([n . n \leftarrow [0..<\texttt{size} \ l], f \ (\texttt{!} \ n) \geq \textbf{Max} \ (\texttt{set} ?fl)]\) using \textit{lm05} by presburger

finally show \(\texttt{thesis}\) by auto

qed

Lemma \textit{lm08}: fixes \(f \ l\)

shows \(\texttt{argmaxList} \ f \ l =\)

\[\{ \texttt{!} \ n . n \leftarrow [0..<\texttt{size} \ l], f \ (\texttt{!} \ n) \geq \textbf{Max} \ (f\texttt{'} \ (\texttt{set} \ l))\}\]

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

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

**Theorem** argmaxadequacy: fixes $f :: 'a => ('b::linorder) 
fixes $l :: 'a list 
shows argmaxList $f $l = [ x <- $l. f x >= Max (f'(set $l))] 
(is $?lh=.)

**Proof** -
let $?P=% y:('b::linorder) . y >= Max (f'(set $l)) 
let $?mh=[nth $l n . n <- [0..<size $l]. $?P (f (nth $l n))] 
let $?rh=[ x <- (map (nth $l) [0..<size $l]). $?P (f x)] 
have $?lh = $?mh using lm08 by fast 
also have ... = $?rh using map-commutes by fast 
also have ...=[x <- $l. $?P (f x)] using map-nth by metis 
finally show $?thesis by force

**qed**

**end**

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

**theory** MiscTools

**imports**

**begin**

**7.1 Facts and notations about relations, sets and functions.**

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

+< abbreviation permits to shorten the notation for altering a function $f$ in
a single point by giving a pair $(a, b)$ so that the new function has value $b$
with argument $a$.

**abbreviation** singlepaste
  where singlepaste $f$ pair == $f$ ++ [(fst pair, snd pair)]

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

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

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

Turns a HOL function into a set-theoretical function
definition
Graph \( f = \{(x, f x) \mid x . \text{True}\}\)

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

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

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

lemma lm001:
\(((P \cup Q) \parallel X) = ((P \parallel X) \cup (Q\parallel X))\)
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 \( \text{fun-upd}\)

definition update
where \( update P Q = P +^* (Q \parallel (\text{Domain } P))\)
notation update (infix +^ 75)

definition runiqer :: \((\alpha \times \beta) \set\rightarrow (\alpha \times \beta) \set\)
where \(runiqer R = \{(x, \text{THE } y . y \in R ''\{x\}\}) \mid x . x \in \text{Domain } R\}\)

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

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

lemma lm002:
assumes runiq \( R\)
shows \( R \supseteq graph (\text{Domain } R) (\text{toFunction } R)\)
unfolding graph-def toFunction-def
using assms graph-def toFunction-def eval-runiq-rel by fastforce

lemma lm003:
assumes runiq \( R\)
shows \( R \subseteq graph (\text{Domain } R) (\text{toFunction } R)\)
unfolding graph-def toFunction-def
using assms eval-runiq-rel runiq-basic Domain.DominI 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 ($R :: (a \times ('b set)) set$) $x :: a$ == $\bigcup (R'' \{x\})$
notation eval-rel2 (infix ,, 75)

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

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

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

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

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

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

lemma lm007:
assumes $(X \cap \text{Domain } P) \subseteq \text{Domain } Q$
shows $P +* Q = (\text{outside } X) +* Q$
unfolding paste-def Outside-def using assms by blast

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

corollary outsideUnion:
$R = (\text{outside } \{x\}) \cup (\{x\} \times (R'' \{x\}))$
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:
\[ R = (R \ outside \ \{ x \}) +* (\{ x \} \times (R'' \ \{ x \})) \]
by (metis paste-outside-restrict restrict-to-singleton)

lemma lm011:
\[ R \subseteq R +* (\{ x \} \times (R'' \ \{ x \})) \]
using lm010 lnm008 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 × Y)
using assms runiq-def Image-subset lm013 trivial-subset lm011 by (metis (no-types))

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

lemma lm015:
\[ (P \ || \ (X \cap Y)) \subseteq (P | X) \quad \text{&} \quad P \ outside \ (X \cup Y) \subseteq P \ outside \ X \]
using Outside-def restrict-def Sigma-Un-distrib1 Un-upper1 inf-mono Diff-mono subset-refl
by (metis (lifting) Sigma-mono inf-le1)

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

lemma lm017:
\[ P'' (X \cap \ Domain \ P) = P'' X \]
by blast

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

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

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

7.2 Ordered relations

lemma lm019:
assumes card X ≥ 1 ∀x ∈ X. y > x
shows y > Max X using asms by (metis (poly-guards-query) Max-in One-nat-def card-eq-0-iff lessI not-le)

lemma lm020:
assumes finite X mx ∈ X f x < f mx
shows x /∈ argmax f X using asms not-less by fastforce

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

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

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

corollary lm023:
assumes finite X mx ∈ X ∀x ∈ X. x ≠ mx −→ f x < f mx
shows argmax f X = {mx} using asms lm022 by (metis Diff-iff insertI1)

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

lemma ln025:
  assumes inj-on f X
  shows inj-on (image f) (Pow X)
  using assms inj-on-image-eq-iff inj-onI PowD by (metis (mono-tags, lifting))

lemma injectionPowerset:
  assumes inj-on f Y X ⊆ Y
  shows inj-on (image f) (Pow X)
  using assms ln025 by (metis subset-inj-on)

definition finestpart
  where finestpart X = (%x. {x}) ∘ X

lemma finestPart:
  finestpart X = {x|x . x∈ X}
  unfolding finestpart-def by blast

lemma finestPartUnion:
  X = (finestpart'X)
  using finestPart by auto

lemma ln026:
  Union ∘ finestpart = id
  using finestpart-def finestPartUnion by fastforce

lemma ln027:
  inj-on Union (finestpart ∘ UNIV)
  using ln026 by (metis inj-on-id inj-on-imageI)

lemma nonEqualitySetOfSets:
  assumes X ≠ Y
  shows {{x} | x . x∈ X} ≠ {{x} | x . x∈ Y}
  using assms by auto

corollary ln028:
  inj-on finestpart UNIV
  using nonEqualitySetOfSets finestPart by (metis (lifting, no-types) injI)

lemma unionFinestPart:
  { Y | Y . EX x.(Y ∈ finestpart x) & (x ∈ X))} = (∪(finestpart'X)
  by auto
lemma rangeSetOfPairs:
  \( \text{Range}\{ (\text{fst pair}, Y) | Y \text{ pair. } Y \in \text{finestpart\ (snd pair)} \land \text{pair } \in \text{X} \}\) =
  \{ Y. \exists x. ( (Y \in \text{finestpart\ x}) \land (x \in \text{Range\ x}) ) \}
by auto

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

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

lemma lm029:
  \( x \in \text{X} = (\{ x \} \in \text{finestpart\ X}) \)
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 (\text{sorted-list-of-set\ X}) = X
using assms by simp

lemma lm032:
  assumes finite A
  shows sum f A = sum f (A \cap B) + sum f (A - B)
using assms by (metis DiffD2 Int-iff Un-Diff-Int Un-commute finite-Un sum.union-inter-neutral)

corollary sumOutside:
  assumes finite g
  shows sum f g = sum f (g \text{ outside\ X}) + (sum f (g||X))
unfolding Outside-def restrict-def using assms add.commute inf-commute lm032
by (metis)
lemma \texttt{lm033}:
assumes \((\text{Domain } P \subseteq \text{Domain } Q)\)
shows \((P \leftrightarrow Q) = Q\)
unfolding paste-def Outside-def using assms by fast

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

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

lemma \texttt{lm036}:
\((P \mid\mid \text{Domain } Q) \leftrightarrow Q = Q\)
by (metis Int-lower2 restrictedDomain \texttt{lm035})

lemma \texttt{lm037}:
\((P \mid\mid X) = P \text{ outside } (\text{Domain } P - X)\)
using Outside-def restrict-def by fastforce

lemma \texttt{lm038}:
\((P \mid\mid X) \subseteq (P \mid\mid ((\text{Domain } P) - X))\)
using \texttt{lm036} \texttt{lm016} by (metis Int-commute restrictedDomain outside-reduces-domain)

lemma \texttt{lm039}:
\((P \mid\mid X) \cap (Q \mid\mid X) = \{\}\)
using \texttt{lm038} by fast

lemma \texttt{lm040}:
\((P \mid\mid (X \cup Y)) \cap (Q \mid\mid X) = \{\} \land (P \mid\mid X) \cap (Q \mid\mid (X \cap Z)) = \{\}\)
using Outside-def restrict-def \texttt{lm039} \texttt{lm015} by fast

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

lemma \texttt{lm042}:
\(R^{\leftrightarrow}(X - Y) = (R \mid\mid X)^{\leftrightarrow}(X - Y)\)
using restrict-def by blast

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

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

lemma setVsList:
  assumes $\forall l \in \text{set } (g1 G), \text{set } (g2 l N) = f2 \ (\text{set } l) N$
  shows $\text{set } [\text{set } (g2 l N), l \leftarrow (g1 G)] = \{f2 \ P \ N| \ P. \ P \in \text{set } (\text{map set } (g1 G))\}$
  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 $R''X = (R \text{ outside } Y)''X$
  using assms Outside-def Image-def by blast

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

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

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

45
lemma lm050:
  inj (λa. ((fst a, fst (snd a)), snd (snd a)))
by (auto intro: injI)

lemma lm051:
  assumes finite X x > Max X
  shows x ∉ X
  using assms Max.coboundedI by (metis leD)

lemma lm052:
  assumes finite A A ≠ {} 
  shows Max (f'A) ∈ f'A
  using assms by (metis Max-in finite-imageI image-is-empty)

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

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

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

lemma rangeEmpty:
  Range − ' {{}} = {{}}
  by auto

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

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

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

lemma flip-flip2:
  flip ◦ flip = id
  using flip-flip by fastforce

lemma lm058:
  fst = (snd ◦ flip)
using `lm057` by fastforce

lemma `lm059`:
\[ \text{snd} = (\text{fst} \circ \text{flip}) \]
using `lm057` by fastforce

lemma `lm060`:
\[ \text{inj-on} \text{ fst} \ P = \text{inj-on} (\text{snd} \circ \text{flip}) \ P \]
using `lm058` by metis

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

lemma `sumPairsInverse`:
 assumes `runiq (P^\sim - 1)`
 shows \( \sum (f \circ \text{snd}) P = \sum f (\text{Range} P) \)
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 \( R,,,(\text{fst} z) = \text{snd} z \)
using assms by (metis rightUniquePair surjective-pairing)

lemma `lm064`:
 assumes `runiq R`
 shows \( \sum (\text{toFunction} R) (\text{Domain} R) = \sum \text{snd} R \)
using assms toFunction-def sum.reindex-cong functionOnFirstEqualsSecond rightUniqueInjectiveOnFirst
by (metis (no-types) `fst-eq-Domain`)

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

lemma `lm066`:

47
Range (R outside X) = R''((Domain R) \setminus X)
by (metis Diff-idemp ImageE Range.intro 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-iff
Int-insert-right-if1 restrictedDomain
by metis

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

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

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

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

lemma cardFinestpart:
card (finestpart X) = card X
using finestpart-def by (metis (lifting) card-image inj-on-inverseI the-elem-eq)

corollary lm071:
finestpart {} = {} \& card 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. y ∈ ∪ (a"\{ x \}) & x ∈ Domain a \} =
\{ (x, \{ y \}) | x y. y ∈ a.,x & x ∈ Domain a \}
using assms Image-runiq-eq-eval
by (metis (lifting, no-types) cSup-singleton)

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

lemma finiteDomainImpliesFinite:
assumes raniq 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 \( f = \lambda x, y. \) let \( g = \text{curry}(f) \) x let \( h = f \)
have inj-on \( f \) \( Y \) by (metis (no-types) Pair-inject inj-onI)
moreover have \( \{x\} \times Y = f' Y \) by (fast)
moreover have \( \forall y. y \in Y \rightarrow g y = h (f y) \) by simp
ultimately show \( \text{thesis} \) using sum.reindex-cong by (metis)
qed

lemma ln077:
  sum (\( \lambda y. f (x, y) \)) \( Y \) = sum f (\( \{x\} \times Y \))
using sumCurry Sigma-cong curry-def sum.cong by fastforce

corollary ln078:
  assumes finite \( X \)
  shows \( \text{sum } f X = \text{sum } f (X - Y) + (\text{sum } f (X \cap Y)) \)
using assms Diff-iff IntD2 Un-Diff-Int finite-Un inf-commute sum.union-inter-neutral
by (metis)

lemma ln079:
  \( (P +\ast Q)''(\text{Domain } Q \cap X) = Q''(\text{Domain } Q \cap X) \)
unfolding paste-def Outside-def Image-def Domain-def by blast

corollary ln080:
  \( (P +\ast Q)''(X \cap (\text{Domain } Q)) = Q''X \)
using Int-commute ln079 by (metis ln017)

corollary ln081:
  assumes \( X \cap (\text{Domain } Q) = \{\} \)
  shows \( (P +\ast Q)''X = (P \text{ outside } (\text{Domain } Q))''X \)
using assms paste-def by fast

lemma ln082:
  assumes \( X \cap Y = \{\} \)
  shows \( (P \text{ outside } Y)''X = P''X \)
using assms Outside-def by fast

corollary ln083:
  assumes \( X \cap (\text{Domain } Q) = \{\} \)
  shows \( (P +\ast Q)''X = P''X \)
using assms ln081 ln082 by (metis)

lemma ln084:
  assumes finite \( X \) finite \( Y \) \( \text{card}(X \cap Y) = \text{card } X \)
  shows \( X \subseteq Y \)
using assms by (metis Int-lower1 Int-lower2 card-seteq order-refl)

corollary cardinalityIntersectionEquality:
  assumes finite \( X \) finite \( Y \) \( \text{card } X = \text{card } Y \)
shows \((\text{card } (X \cap Y) = \text{card } X) = (X = Y)\)

using assms lm084 by (metis card-seteq le-iff-inf order-refl)

lemma lm085:
assumes \(P \, xx\)
shows \(\{ (x, f \, x) \mid x. \, P \, x \}, xx \) = \( f \, xx \)

proof –
let \(?F=\{(x, f \, x) \mid x. \, P \, x \}\) let \(?X=\, ?F''\{xx\}\)
have \(?X=\{f xx\}\) using Image-def assms by blast thus \(?thesis\) by fastforce

qed

lemma graphEqImage:
assumes \(x \in X\)
shows \(\text{graph } X \, f, \, x = f \, x\)

unfolding graph-def using assms lm085 by (metis (mono-tags) Gr-def)

lemma lm086:
Graph \(f, x = f \, x\)

using UNIV-I graphEqImage lm005 by (metis (no-types))

lemma lm087:
\(\text{toFunction } (\text{Graph } f) = f\) (is \(?L\=-\))

proof –
\{fix \(x\) have \(?L \, x= f \, x\) unfolding toFunction-def lm086 by metis\}

thus \(?thesis\) by blast

qed

lemma lm088:
\(R \, \text{outside } X \subseteq R\)

by (metis outside-union-restrict subset-Un-eq sup-left-idem)

lemma lm089:
\(\text{Range}(f \, \text{outside } X) \supseteq (\text{Range } f) - (f'\cdot X)\)

using Outside-def by blast

lemma lm090:
assumes \(runiq \, P\)
shows \((P^{-1}\cdot((\text{Range } P) - Y)) \cap ((P^{-1}\cdot Y) = \{\}\)

using assms rightUniqueFunctionAfterInverse by blast

lemma lm091:
assumes \(runiq \, \, (P^{-1})\)
shows \((P^{-1}\cdot((\text{Domain } P) - X)) \cap (P^{-1}\cdot X) = \{\}\)

using assms rightUniqueFunctionAfterInverse by fast

lemma lm092:
assumes \(runiq \, f \, runiq \, (f'\cdot 1)\)
shows \(\text{Range}(f \, \text{outside } X) \subseteq (\text{Range } f) - (f'\cdot X)\)

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 lm094:
  R''{}={}
  by (metis Image-empty)

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 ∈ f''{x} runiq f
  shows f.,x = y 
  using assms by (metis Image-singleton-iff rightUniquePair)
7.3 Indicator function in set-theoretical form.

abbreviation
Outside' \( X f \equiv f \) outside \( X \)

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

abbreviation
\( \text{chii} \ X \ Y \equiv \text{toFunction} \ (X <|| Y) \)
notation \( \text{chii} \ (\text{infix} \ <| \ 80) \)

abbreviation
\( \text{chi} \ X \equiv \text{indicator} \ X \)

lemma \( \text{lm098} \):
\( \text{runiq} \ (X <|| Y') \)
by \( \text{(rule lm014)} \)

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

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

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

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

corollary \( \text{lm103} \):
\( \text{sum} \ (X <|| Y) \ (X \cup Y) = \text{sum} \ (\text{chi} \ X) \ (X \cup Y) \)
using \( \text{lm102 sum.cong} \) by \( \text{metis} \)

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

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

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

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

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

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

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

corollary lm112:
assumes $Z \subseteq X \cup Y$ finite $Z$
shows $\sum (X <\mid Y) \; Z = \operatorname{card} (Z \cap X)$
using assms lm111 by (metis lm106 sum-indicator-eq-card)

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

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

corollary differenceSumVsCardinalityReal:
assumes $Z \subseteq X \cup Y$ finite $Z$
shows $\text{real} \left( \sum (X <| Y) X \right) - \text{real} (\sum (X <| Y) Z) = 
\text{real} (\text{card} X) - \text{real} (\text{card} (Z \cap X))$
using asms 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 $\exists n. n \leftarrow [0..<\text{size } l], P \ (l!n) \neq []$
using asms by auto

lemma lm115:
assumes $ll \in \text{set} \ (l::'a list)$
shows $\exists n \in (\text{nth } l) -\ ' \ (\text{set } l). ll=!!n$
using assms(l1) by (metis in-set-conv-nth vimageI2)

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

lemma lm117:
assumes $P \ -\ ' \ \{\text{True}\} \cap \text{set } l \neq \{}$
shows $\exists n \in \{0..<\text{size } l\}. P \ (l!n)$
using assms lm116 by fastforce

lemma nonEmptyListFiltered:
assumes $P \ -\ ' \ \{\text{True}\} \cap \text{set } l \neq \{}$
shows \( n, n \leftarrow [0..\text{size } l], P (\forall n) \neq [] \)
using assms filterpositions2-def lm117 lm114 by metis

lemma lm118:
\((\text{nth } l) \mapsto \text{set} ([n, n \leftarrow [0..\text{size } l], (\%x. x \in X) (\forall n)]) \subseteq X \cap \text{set } l\)
by force

corollary lm119:
\((\text{nth } l) \mapsto \text{set} (\text{filterpositions2 } (\%x. (x \in X)) l) \subseteq X \cap \text{set } l\)
unfolding filterpositions2-def using lm118 by fast

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

lemma lm121:
assumes \( X \subseteq \{0..\text{size list}\} \)
shows \((\text{nth } \text{list}) : X \subseteq \text{set list}\)
using assms atLeastLessThan-def atLeast0LessThan lessThan-iff by auto

lemma lm122:
\( \text{set} ([n, n \leftarrow [0..\text{size } l], P (\forall n)]) \subseteq \{0..\text{size } l\} \)
by force

lemma lm123:
\( \text{set} (\text{filterpositions2 pre list}) \subseteq \{0..\text{size list}\} \)
using filterpositions2-def lm122 by metis

7.5 Computing all the permutations of a list

abbreviation rotateLeft == rotate

abbreviation rotateRight n l == rotateLeft (size l - (n mod (size l))) l

abbreviation insertAt x l n == rotateRight n (x#(rotateLeft n l))

fun perm2 where
perm2 [] = (\%n. []) |
perm2 (x#l) = (\%n. insertAt x ((perm2 l) (n div (1 + size l))))

56
\[(n \mod (1 + \text{size } l))\]

**abbreviation**

\[\text{takeAll } P \text{ list} == \text{map} \ (\text{nth} \ \text{list}) \ (\text{filterpositions2} \ P \ \text{list})\]

**lemma** \(\text{permutationNotEmpty}:\)

**assumes** \(l \neq []\)

**shows** \(\text{perm2} \ l \ n \neq []\)

**using** \(\text{assms} \ \text{perm2} \text{.simps}(2) \ \text{rotate-is-nil-conv} \ \text{by} \ (\text{metis} \ \text{neq-nil-conv})\)

**lemma** \(\text{lm124}:\)

\[\text{set} \ (\text{takeAll} \ P \ \text{list}) = ((\text{nth} \ \text{list}) \ \text{'} \ \text{set} \ (\text{filterpositions2} \ P \ \text{list}))\]

**by** \(\text{simp}\)

**corollary** \(\text{listIntersectionWithSet}:\)

\[\text{set} \ (\text{takeAll} \ (%x. (x \in X)) \ l) \subseteq (X \cap \text{set} \ l)\]

**using** \(\text{lm119} \ \text{lm124} \ \text{by} \ \text{metis}\)

**corollary** \(\text{lm125}:\)

\[\text{set} \ (\text{takeAll} \ P \ \text{list}) \subseteq \text{set \ list}\]

**using** \(\text{lm123} \ \text{lm124} \ \text{lm121} \ \text{by} \ \text{metis}\)

**lemma** \(\text{takeAllSubset}:\)

\[\text{set} \ (\text{takeAll} \ (%x. x \in P) \ \text{list}) \subseteq P\]

**by** \(\text{metis} \ \text{Int-subset-iff} \ \text{listIntersectionWithSet}\)

**lemma** \(\text{lm126}:\)

\[\text{set} \ (\text{insertAt} \ x \ l \ n) = \{x\} \cup \text{set} \ l\]

**by** \(\text{simp}\)

**lemma** \(\text{lm127}:\)

\[\forall n. \ \text{set} \ (\text{perm2} \ [] \ n) = \text{set} []\]

**by** \(\text{simp}\)

**lemma** \(\text{lm128}:\)

**assumes** \(\forall n. \ \text{set} \ (\text{perm2} \ l \ n) = \text{set} \ l\)

**shows** \(\text{set} \ (\text{perm2} \ (x\#l) \ n) = \{x\} \cup \text{set} \ l\)

**using** \(\text{assms} \ \text{lm126} \ \text{by} \ \text{force}\)

**corollary** \(\text{permutationInvariance}:\)

\[\forall n. \ \text{set} \ (\text{perm2} \ ((l::'a \ \text{list}) \ n)) = \text{set} \ l\]

**proof** \(\text{(induct } l)\)

**let** \(?P = %l::('a \ \text{list}). \ \forall n. \ \text{set} \ (\text{perm2} \ l \ n) = \text{set} \ l\)

**show** \(?P \ [] \ \text{using} \ \text{lm127} \ \text{by} \ \text{force}\)

**fix** \(x \ \text{fix } l\)

**assume** \(?P \ l \ \text{then}\)

**show** \(?P \ (x\#l) \ \text{by} \ \text{force}\)

\text{qed}
corollary takeAllPermutation:
set (perm2 (takeAll (%x.(x∈X)) l) n) ⊆ X ∩ set l
using listIntersectionWithSet permutationInvariance by metis

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

7.6 A more computable version of toFunction.

abbreviation toFunctionWithFallback R fallback ==
(% x. if (R``{x} = {R,x}) then (R,x) else fallback)

notation
toFunctionWithFallback (infix Else 75)

abbreviation
sum' R X == sum (R Else 0) X

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

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

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

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

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

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

corollary lm135:
  assumes finite $X$
  shows $\text{sum } (f \text{ Else } 0) (X \cap \text{Domain } f) = \text{sum } (f \text{ Else } 0) X$
(is $?L=?R$)
proof
  have $?R=?L$ using assms by (rule lm134)
thus $\text{thesis }$ by simp
qed

corollary lm136:
  assumes finite $X$ runiq $f$
  shows $\text{sum } (f \text{ Else } 0) X = \text{sum } (\text{toFunction } f) (X \cap \text{Domain } f)$
(is $?L=?R$)
proof
  have $?R = \text{sum } (f \text{ Else } 0) (X \cap \text{Domain } f)$ using assms(2) lm130 by fastforce
moreover have $\ldots = ?L$ using assms(1) by (rule lm135)
ultimately show $\text{thesis }$ by presburger
qed

lemma lm137:
  $\text{sum } (f \text{ Else } 0) X = \text{sum'} f X$
by fast

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

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

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

lemma domainCharacteristicFunction:
  $\text{Domain } (X <\| Y) = X \cup Y$
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 lm140:**
assumes runiq R runiq (R⁻¹)
shows (R``A) ∩ (R``B) = R``(A∩B)
using assms rightUniqueInjectiveOnFirst converse-Image by force

**Lemma intersectionEmptyRelationIntersectionEmpty:**
assumes runiq (R⁻¹) runiq R X1 ∩ X2 = {}
shows (R``X1) ∩ (R``X2) = {}
using assms by (metis disj-Domain-imp-disj-Image inf-assoc inf-bot-right)

**Lemma lm141:**
assumes runiq f trivial Y
shows trivial (f`` (f⁻¹ `` Y))
using assms by (metis rightUniqueFunctionAfterInverse trivial-subset)

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

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

**Lemma lm144:**
(¬ (finite A)) = (card (Pow A) = 0)
by auto

**Corollary lm145:**
(finite A) = (card (Pow A) ≠ 0)
using lm144 by metis

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

**Lemma lm147:**
assumes card (Pow A) = 2
shows card A = 1
using assms lm146
by (metis log-exp power-one-right zero-not-eq-two)
lemma \textit{lm148}:
\begin{itemize}
\item assumes \(\text{card } (\text{Pow } X) = 1 \lor \text{card } (\text{Pow } X) = 2\)
\item shows \text{trivial } X
\item using \text{assms trivial-empty-or-singleton lm143 lm147 cardinalityOneTheElementIdentity} by \text{metis}
\end{itemize}

lemma \textit{lm149}:
\begin{itemize}
\item trivial \(A = (\text{card } (\text{Pow } A) \in \{1, 2\})\)
\item using \text{lm148 lm142} by \text{blast}
\end{itemize}

lemma \textit{lm150}:
\begin{itemize}
\item assumes \(R \subseteq f \text{ runiq } f \text{ Domain } f = \text{ Domain } R\)
\item shows \text{runiq } R
\item using \text{assms \text{Domain-iff contra-subsetD runiq-wrt-ex1 subrelI}} by \text{(metis \text{(full-types, hide-lams}}))
\end{itemize}

lemma \textit{lm151}:
\begin{itemize}
\item assumes \(f \subseteq g \text{ runiq } g \text{ Domain } f = \text{ Domain } g\)
\item shows \(g \subseteq f\)
\item using \text{assms Domain-iff contra-subsetD runiq-wrt-ex1 subrelI}
\item by \text{(metis \text{(full-types, hide-lams}}))
\end{itemize}

lemma \textit{lm152}:
\begin{itemize}
\item assumes \(R \subseteq f \text{ runiq } f \text{ Domain } f \subseteq \text{ Domain } R\)
\item shows \(f = R\)
\item using \text{assms \text{lm151}} by \text{(metis \text{Domain-mono dual-order.antisym}})
\end{itemize}

lemma \textit{lm153}:
\begin{itemize}
\item \(\text{graph } X \ f = (\text{Graph } f) \parallel X\)
\item using \text{inf-top.left-neutral lm005 domainOfGraph restrictedDomain lm152 graphIntersection restriction-is-subrel subrel-runiq subset-iff}
\item by \text{(metis \text{(erased, lifting}}))
\end{itemize}

lemma \textit{lm154}:
\begin{itemize}
\item \(\text{graph } (X \cap Y) \ f = (\text{graph } X \ f) \parallel Y\)
\item using \text{doubleRestriction lm153} by \text{metis}
\end{itemize}

lemma \textit{restrictionVsIntersection}:
\begin{itemize}
\item \(\{(x, f x) | x. x \in X2\} \parallel X1 = \{(x, f x) | x. x \in X2 \cap X1\}\)
\item using \text{graph-def lm154} by \text{metis}
\end{itemize}

lemma \textit{lm155}:
\begin{itemize}
\item assumes \(\text{runiq } f \ X \subseteq \text{Domain } f\)
\item shows \(\text{graph } X \ (\text{toFunction } f) = (f \parallel X)\)
\item proof –
\item \text{have } \\bigwedge v w. (v::'a set) \subseteq w \rightarrow w \cap v = v \text{ by (simp add: Int-commute inf.absorb1}}
\item \text{thus } \text{graph } X \ (\text{toFunction } f) = f \parallel X \text{ by (metis \text{assms(1) assms(2) doubleRestriction lm004 lm153}})
\item qed
\end{itemize}
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) o flip)

lemma lm158:
finite X = (X ∈ range set)
by (metis List.finite-set finite-list image-iff rangeI)

lemma lm159:
finte = (%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 bids = ((participant × goods) × price) set

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

abbreviation is-partition-of' $PA == \left( \bigcup P = A \land \text{is-non-overlapping } P \right)$

abbreviation all-partitions' $A == \{ P : \text{is-partition-of' } PA \}$

abbreviation possible-allocations-rel' $GN == \text{Union}\{ \text{injections } YN \mid Y . Y \in \text{all-partitions' } G \}$

abbreviation allAllocations where
  allAllocations $NG == \text{converse' } (\text{possible-allocations-rel' } GN)$

the algorithmic version of possible-allocations-rel

fun possible-allocations-alg :: goods $\Rightarrow$ participant set $\Rightarrow$ allocation-rel list
  where possible-allocations-alg $GN ==$
    \text{concat } [ \text{injections-alg } YN . Y \leftarrow \text{all-partitions-alg } G ]

abbreviation allAllocationsAlg $NG ==$
  map converse (concat [(injections-alg I N) . I \leftarrow \text{all-partitions-list } G])

8.2 VCG mechanism

abbreviation winningAllocationsRel $NG b ==$
  \text{argmax} (\text{sum } b) (\text{allAllocations } NG)

abbreviation winningAllocationRel $NG t b == t (\text{winningAllocationsRel } NG b)$

abbreviation winningAllocationsAlg $NG b == \text{argmaxList} (\text{proceeds } b) (\text{allAllocationsAlg } NG)$

definition winningAllocationAlg $NG t b == t (\text{winningAllocationsAlg } NG b)$

calculating payments

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

abbreviation alpha $NG b n == \text{Max} ((\text{sum } b)'(\text{allAllocations } (N-\{n\}) G))$

abbreviation alphaAlg $NG b n == \text{Max} ((\text{proceeds } b)'(\text{set } (\text{allAllocationsAlg } (N-\{n\}) (G::- list))))$

abbreviation remainingValueRel $NG t b n == \text{sum } b ((\text{winningAllocationRel } NG t b) -- n)$
abbreviation \textit{remainingValueAlg} \(N \ G \ t \ b \ n\) \(\equiv\) \textit{proceeds} \(b\) \((\textit{winningAllocationAlg} \ N \ G \ t \ b) \ -- n\)

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

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

begin

9.1 Preliminary lemmas

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

\textbf{lemma} \textit{lm002}:
\begin{itemize}
\item \textbf{assumes} \(\text{finite} \ G\)
\item \textbf{shows} \(\text{all-partitions} \ G = \ \text{set} \ \{ \text{set} \ (\text{all-partitions-alg} \ G) \}\)
\item \textbf{using} \(\text{assms} \ \text{sortingSameSet} \ \text{all-partitions-alg-def} \ \text{all-partitions-paper-equiv-alg}
\ \text{distinct-sorted-list-of-set} \ \text{image-set}\)
\item \textbf{by} \textit{metis}
\end{itemize}

9.2 Definitions of various subsets of \textit{UNIV}.

abbreviation \textit{isChoice} \(R \equiv \forall x. R^{\uparrow} \{x\} \subseteq x\)
abbreviation \textit{partitionsUniverse} \(\equiv\) \(\{X. \ \text{is-non-overlapping} \ X\}\)

\textbf{lemma} \textit{partitionsUniverse} \(\subseteq \text{Pow} \ \text{UNIV}\)
\begin{itemize}
\item \textbf{by} \textit{simp}
\end{itemize}
9.3 Results about the sets defined in the previous section

**Lemma lm003:**
- **Assumes** \( \forall \ x1 \in X. (x1 \neq \{\} \land (\forall \ x2 \in X - \{x1\}. x1 \cap x2 = \{\})) \)
- **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 raniq-wrt-eval-rel subset-trans)

**Lemma lm006:** injections X Y \(\subseteq\) injectionsUniverse

**Using** injections-def by fast

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

**Using** injections-def by blast

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

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

**Lemma allocationInverseRangeDomainProperty:**
- **Assumes** \( a \in \text{allAllocations} \ N \ G \)
- **Shows** \( a^{-1} \in \text{injections} (\text{Range} \ a) \ N \ & \ (\text{Range} \ a) \ \text{partitions} \ G \ & \ Domain \ a \subseteq N \)

**Unfolding** injections-def using assms all-partitions-def injections-def by fastforce

**Lemma lm009:**
- **Assumes** is-non-overlapping XX YY \(\subseteq\) XX
- **Shows** (XX \(\cap\) YY) \(\subseteq\) (XX \(\cup\) (XX \(\cup\) YY))

**Proof**
- let \(?xx=XX \cap YY\) let \(?x=\cup XX\) let \(?y=\cup YY\)
- let \(?x=\cup X \cap YY\)

66
have \( \forall \ y \in YY, \forall \ x \in \exists xx, \ y \cap x = \{ \} \) using assms is-non-overlapping-def
by \( \text{metis Diff-iff set-rev-mp} \)
then have \( \bigcup \ ?xx \subseteq \ ?x \) using assms by blast
then have \( \bigcup \ ?xx = \ ?x \) by blast
moreover have is-non-overlapping \( ?xx \) using subset-is-non-overlapping
by \( \text{metis Diff-subset assms(1)} \)
ultimately
show \( \?thesis \) using is-partition-of-def by blast
qed

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

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

lemma allocationProperty:
\( a \in \text{possible-allocations-rel } G \ N \longleftrightarrow \)
\( \text{runiq } a \ & \ & \text{runiq } (a^{-1}) \ & \ & \text{(Domain } a) \ \text{partitions } G \ & \ & \text{Range } a \subseteq N \)
using allocationRightUniqueRangeDomain lm010 by blast

lemma lm011:
\( \text{possible-allocations-rel’ } G \ N \subseteq \text{injectionsUniverse} \)
using injections-def by force

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

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

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

lemma lm019:
converse’ \text{injectionsUniverse} = \text{injectionsUniverse}
by auto

lemma lm020:
converse’(A \cap B) = (converse’A) \cap (converse’B)
by force

lemma allocationInjectionsUniverseProperty:
allAllocations N G =
  \text{injectionsUniverse} \cap \{ a. \text{Domain} a \subseteq N \& \text{Range} a \in \text{all-partitions} G \}
proof –
let ?A=possible-allocations-rel G N
let ?c=converse
let \( \mathcal{I} = \text{injectionsUniverse} \)
let \( \mathcal{P} = \text{all-partitions G} \)
let \( d = \text{Domain} \)
let \( r = \text{Range} \)

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

moreover have \( \ldots = (\forall c'? \mathcal{I}) \cap \{ a.a. \ d a \subseteq N \ & \ r a \in \mathcal{P} \} \) by fastforce

moreover have \( \ldots = \mathcal{I} \cap \{ a.a. \ d a \subseteq N \ & \ r a \in \mathcal{P} \} \) using \( \text{lm019} \) by metis

ultimately show \( \mathcal{thesis} \) by presburger

qed

lemma \( \text{lm021} \):
\begin{align*}
\text{allAllocations } N \ G & \subseteq \text{injectionsUniverse} \\
\text{using } \text{allocationInjectionsUniverseProperty} \text{ by fast}
\end{align*}

lemma \( \text{lm022} \):
\begin{align*}
\text{allAllocations } N \ G & \subseteq \text{partitionValuedUniverse} \\
\text{using } \text{allocationInverseRangeDomainProperty} \text{ is-partition-of-def is-non-overlapping-def} \text{ by auto blast}
\end{align*}

corollary \( \text{allAllocationsUniverse} \):
\begin{align*}
\text{allAllocations } N \ G & \subseteq \text{allocationsUniverse} \\
\text{using } \text{lm021} \ \text{lm022} \text{ by (metis (lifting, mono-tags) inf.bounded-iff)}
\end{align*}

corollary \( \text{possibleAllocationsRelCharacterization} \):
\begin{align*}
a & \in \text{allAllocations } N \ G = \\
(a \in \text{injectionsUniverse} \ & \ \text{Domain} \ a \subseteq N \ & \ \text{Range} \ a \in \text{all-partitions G}) \\
\text{using } \text{allocationInjectionsUniverseProperty} \ Int-Collect \ Int-iff \text{ by (metis (lifting))}
\end{align*}

corollary \( \text{lm023} \):
assumes \( a \in \text{allAllocations } N1 \ G \)
sows \( a \in \text{allAllocations } (N1 \cup N2) \ G \)
proof
have \( \text{Domain} \ a \subseteq N1 \cup N2 \) using \( \text{assms} \) \( \text{possibleAllocationsRelCharacterization} \)
by (metis \( \text{le-supI1} \))

moreover have \( a \in \text{injectionsUniverse} \ & \ \text{Range} \ a \in \text{all-partitions G} \)
using \( \text{assms possibleAllocationsRelCharacterization} \) by blast

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

qed

corollary \( \text{lm024} \):
\begin{align*}
\text{allAllocations } N1 \ G & \subseteq \text{allAllocations } (N1 \cup N2) \ G \\
\text{using } \text{lm023} \text{ by (metis subsetI)}
\end{align*}

lemma \( \text{lm025} \):
assumes \( (\bigcup P_1) \cap (\bigcup P_2) = \{\} \)
  is-non-overlapping \( P_1 \) is-non-overlapping \( P_2 \)
  \( X \in P_1 \cup P_2 \) \( Y \in P_1 \cup P_2 \) \( X \cap Y \neq \{\} \)
shows \((X = Y)\)
unfolding is-non-overlapping-def using assms is-non-overlapping-def by fast

lemma \textit{lm026}:\nassumes \( (\bigcup P_1) \cap (\bigcup P_2) = \{\} \)
  is-non-overlapping \( P_1 \)
  is-non-overlapping \( P_2 \)
  \( X \in P_1 \cup P_2 \)
  \( Y \in P_1 \cup P_2 \)
  \((X = Y)\)
shows \(X \cap Y \neq \{\} \)
unfolding is-non-overlapping-def using assms is-non-overlapping-def by fast

lemma \textit{lm027}:\nassumes \( (\bigcup P_1) \cap (\bigcup P_2) = \{\} \)
  is-non-overlapping \( P_1 \)
  is-non-overlapping \( P_2 \)
shows is-non-overlapping \((P_1 \cup P_2)\)
unfolding is-non-overlapping-def using assms \textit{lm025} \textit{lm026} by metis

lemma \textit{lm028}:\nRange \( Q \cup (\text{Range } (P \text{ outside } \text{Domain } Q))) = \text{Range } (P +* Q)\)
by \(\text{(simp add: paste-def Range-Un-eq Un-commute)}\)

lemma \textit{lm029}:\nassumes \( a_1 \in \text{injectionsUniverse} \)
  \( a_2 \in \text{injectionsUniverse} \)
  \((\text{Range } a_1) \cap (\text{Range } a_2) = \{\} \)
  \((\text{Domain } a_1) \cap (\text{Domain } a_2) = \{\} \)
shows \( a_1 \cup a_2 \in \text{injectionsUniverse} \)
using assms disj-Un-runiq
by \(\text{(metis (no-types) Domain-converse converse-Un mem-Collect-eq)}\)

lemma nonOverlapping:\nassumes \( R \in \text{partitionValuedUniverse} \)
shows is-non-overlapping \((\text{Range } R)\)
proof  
  obtain \( P \) where  
  \( \emptyset: P \in \text{partitionsUniverse} \& R \subseteq \text{UNIV} \times P \) using assms by blast  
  have \( \text{Range } R \subseteq P \) using \( \emptyset \) by fast  
  then show \( \text{thesis} \) using \( \emptyset \) mem-Collect-eq subset-is-non-overlapping by \(\text{(metis)}\)
qed

lemma allocationUnion:  
assumes \( a_1 \in \text{allocationsUniverse} \)
  \( a_2 \in \text{allocationsUniverse} \)
\[
(\bigcup \{\text{Range } a1\}) \cap (\bigcup \{\text{Range } a2\}) = \{\}
\]
\[
(Domain a1) \cap (\text{Domain } a2) = \{\}
\]
shows \( a1 \cup a2 \in \text{allocationsUniverse} \)

proof –

let \(?a\) = \(a1 \cup a2\)
let \(?b1\) = \(a1^\sim - 1\)
let \(?b2\) = \(a2^\sim - 1\)
let \(?r\) = \(\text{Range}\)
let \(?d\) = \(\text{Domain}\)

let \(\mathcal{I}\) = \(\text{injectionsUniverse}\)
let \(\mathcal{P}\) = \(\text{partitionsUniverse}\)
let \(\mathcal{PV}\) = \(\text{partitionValuedUniverse}\)
let \(?u\) = \(\text{runiq}\)
let \(\mathcal{P}\) = \(\text{is-non-overlapping}\)

have \(?p\) (\(?r\) \(a1\)) & \(?p\) (\(?r\) \(a2\)) using \(\text{assms nonOverlapping}\) by blast then
moreover have \(?p\) (\(?r\) \(a1\) \cup \(?r\) \(a2\)) using \(\text{assms}\) by (metis \(\text{lm027}\))
then moreover have (\(?r\) \(a1\) \cup \(?r\) \(a2\)) \(\in\) \(\mathcal{P}\) by \(\text{simp}\)
moreover have \(?r\) \(?a\) = (\(?r\) \(a1\) \cup \(?r\) \(a2\)) using \(\text{assms}\) by \(\text{fast}\)
ultimately moreover have \(?p\) (\(?r\) \(?a\)) using \(\text{lm027}\) \(\text{assms}\) by \(\text{fastforce}\)
then moreover have \(?a\) \(\in\) \(\mathcal{PV}\) using \(\text{assms}\) by \(\text{fast}\)
moreover have \(?r\) \(a1\) \(\cap\) (\(?r\) \(a2\)) \(\subseteq\) \(\text{Pow}\) (\(\bigcup\) \(\{\text{Range } a1\}) \cap (\bigcup \{\text{Range } a2\})\) by \(\text{auto}\)
ultimately moreover have \(\{\} \notin (\text{\(?r\) a1}) \& \{\} \notin (\text{\(?r\) a2})\) using \(\text{is-non-overlapping-def}\) by (metis \(\text{Int-empty-left}\))
ultimately moreover have \(\text{\(?r\) a1} \cap (\text{\(?r\) a2}) = \{\}\) using \(\text{assms nonOverlapping}\) \(\text{is-non-overlapping-def}\) by \(\text{auto}\)
ultimately moreover have \(?a\) \(\in\) \(\mathcal{I}\) using \(\text{lm029}\) \(\text{assms}\) by \(\text{fastforce}\)
ultimately show \(?\text{thesis}\) by \(\text{blast}\)

qed

lemma \(\text{lm030}\):
assumes \(a \in \text{injectionsUniverse}\)
shows \(a - b \in \text{injectionsUniverse}\)
using \(\text{assms}\)
by (metis (lifting) \(\text{Diff-subset}\) \(\text{converse-mono}\) \(\text{mem-Collect-eq}\) \(\text{subrel-runiq}\))

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

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

71
corollary lm033:
\begin{align*}
\text{allAllocations } N & \text{ } G = \\
\text{injectionsUniverse } \cap (\text{Range } \rightarrow (\text{all-partitions } G)) & \cap (\text{Domain } \rightarrow (\text{Pow } N)) \\
\text{using } \text{lm032 } \text{Int-assoc by (metis)}
\end{align*}

lemma lm034:
\begin{align*}
\text{assumes } a & \in \text{allAllocations } N \text{ } G \\
\text{shows } (a^{-1} & \in \text{injections } (\text{Range } a) \text{ } N \& \\
& \text{Range } a \in \text{all-partitions } G) \\
\text{using } \text{assms} & \\
\text{by } (\text{metis } (\text{mono-tags, hide-lams) possssibleAllocationsRelCharacterization} \\
& \text{allocationInverseRangeDomainProperty})
\end{align*}

lemma lm035:
\begin{align*}
\text{assumes } a^{-1} & \in \text{injections } (\text{Range } a) \text{ } N \text{ } \text{Range } a \in \text{all-partitions } G \\
\text{shows } a & \in \text{allAllocations } N \text{ } G \\
\text{using } \text{assms } \text{image-iff by fastforce}
\end{align*}

lemma allocationReverseInjective:
\begin{align*}
a & \in \text{allAllocations } N \text{ } G = \\
(a^{-1} & \in \text{injections } (\text{Range } a) \text{ } N \& \text{ } \text{Range } a \in \text{all-partitions } G) \\
\text{using } \text{lm034 } \text{lm035 by metis}
\end{align*}

lemma lm036:
\begin{align*}
\text{assumes } a & \in \text{allAllocations } N \text{ } G \\
\text{shows } a & \in \text{injections } (\text{Domain } a) \text{ } (\text{Range } a) \& \\
& \text{Range } a \in \text{all-partitions } G \& \\
& \text{Domain } a \subseteq N \\
\text{using } \text{assms } \text{mem-Collect-eq injections-def possssibleAllocationsRelCharacteriza-} \\
& \text{tion order-reffl} \\
\text{by } (\text{metis } (\text{mono-tags, lifting}))
\end{align*}

lemma lm037:
\begin{align*}
\text{assumes } a & \in \text{injections } (\text{Domain } a) \text{ } (\text{Range } a) \\
& \text{Range } a \in \text{all-partitions } G \\
& \text{Domain } a \subseteq N \\
\text{shows } a & \in \text{allAllocations } N \text{ } G \\
\text{using } \text{assms } \text{mem-Collect-eq possssibleAllocationsRelCharacterization injections-def} \\
\text{by } (\text{metis (erased, lifting)})
\end{align*}

lemma characterizationallAllocations:
\begin{align*}
a & \in \text{allAllocations } N \text{ } G = (a \in \text{injections } (\text{Domain } a) \text{ } (\text{Range } a) \& \\
& \text{Range } a \in \text{all-partitions } G \& \\
& \text{Domain } a \subseteq N) \\
\text{using } \text{lm036 } \text{lm037 by metis}
\end{align*}

lemma lm038:
\begin{align*}
\text{assumes } a & \in \text{partitionValuedUniverse}
\end{align*}
shows $a - b \in \text{partitionValuedUniverse}$
using assms subset-is-non-overlapping by fast

**lemma** reducedAllocation:
assumes $a \in \text{allocationsUniverse}$
shows $a - b \in \text{allocationsUniverse}$
using assms lm030 lm038 by auto

**lemma** lm039:
assumes $a \in \text{injectionsUniverse}$
shows $a \in \text{injections} (\text{Domain } a) (\text{Range } a)$
using assms injections-def mem-Collect-eq order-refl by blast

**lemma** lm040:
assumes $a \in \text{allocationsUniverse}$
shows $a \in \text{allAllocations} (\text{Domain } a) (\bigcup (\text{Range } a))$
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 $?r a \in ?P (\bigcup (\ ?r a))$ unfolding all-partitions-def
using is-partition-of-def mem-Collect-eq by (metis)
then show $?thesis$
using assms IntI Int-lower1 equalityE allocationInjectionsUniverseProperty
mem-Collect-eq set-rev-mp
by (metis (lifting, no-types))
qed

**lemma** lm041:
$(\{X\} \in \text{partitionsUniverse}) = (X \neq \{\})$
using is-non-overlapping-def by fastforce

**lemma** lm042:
$(x, X)) \rightarrow \{(x, \{\})\} \in \text{partitionValuedUniverse}$
using lm041 by auto

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

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

**lemma** lm043:
assumes is-non-overlapping PP is-non-overlapping (Union PP)
shows is-non-overlapping (Union · PP)
proof 
let ?p = is-non-overlapping
let ?U = Union
let ?P2 = ?U PP
let ?P1 = ?U \cdot PP
have 
0: \forall X \in ?P1. \forall Y \in ?P1. \ (X \cap Y = \{\} \rightarrow X \neq Y)
  using assms is-non-overlapping-def Int-absorb Int-empty-left UnionI Union-disjoint

ex-in-conv imageE
by (metis (hide-lams, no-types))

\{ 
fix X Y
assume
1: X \in ?P1 \& Y \in ?P1 \& X \neq Y
then obtain XX YY
  where
2: X = ?U XX \& Y = ?U YY \& XX \subseteq PP \& YY \subseteq PP by blast
then have XX \subseteq Union PP \& YY \subseteq Union PP \& XX \cap YY = \{\}
  using 1 2 is-non-overlapping-def assms(1) Sup-upper by metis
then moreover have \forall x \in XX. \forall y \in YY. \ x \cap y = \{\} using assms(2)
  is-non-overlapping-def
  by (metis IntI empty-iff subsetCE)
ultimately have X \cap Y = \{\} using assms 0 1 2 is-non-overlapping-def by auto
\}
then show ?thesis using 0 is-non-overlapping-def by metis
qed

lemma lm044: 
assumes a \in allocationsUniverse
shows (a - ((X \cup \{i\}) \times (Range a))) \cup 
  \{\{(i, \mathcal{U} (a''(X \cup \{i\})))\} - \{(i, \{\})\}\} \in allocationsUniverse \& 
  \cup (Range ((a - ((X \cup \{i\}) \times (Range a))) \cup \{(i, \mathcal{U} (a''(X \cup \{i\})))\} - 
  \{(i, \{\})\})) = 
  \cup (Range a)
proof 
let ?d = Domain
let ?r = Range
let ?U = Union
let ?p = is-non-overlapping
let ?P = partitionsUniverse
let ?u = runiq
let ?Xi = X \cup \{i\}
let ?b = ?Xi \times (?r a)
let ?al = a - ?b
let ?Yi = a'' ?Xi
let ?Y = ?U \?Yi
let \( ?a2 = \{(i, ?Y)\} \)
let \( ?a3 = \{(i, \{\})\} \)
let \( ?a2 = ?A2 - ?a3 \)
let \( ?aa1 = a \) outside \( ?Xi \)
let \( ?c = ?a1 \cup ?a2 \)
let \( ?t1 = ?c \in \) allocationsUniverse
have I: \( ?U(?r(?a1\cup ?a2)) = ?U(?r ?a1) \cup (\sim ?U(?r ?a2)) \) by (metis Range-Un-eq Union-Un-distrib)

have 2: \( ?U(?r a) \subseteq ?U(?r ?a1) \cup ?U(a''(?Xi)) \& ?U(?r ?a1) \cup ?U(?r ?a2) \subseteq ?U(?r a) \) by blast
have 3: \( ?u a \& ?u (a'' - 1) \& ?p (?r a) \& ?r ?a1 \subseteq ?r a \& ?Yi \subseteq ?r a \)
  using assms Int-iff nonOverlapping mem-Collect-eq by auto
then have 4: \( ?p (?r ?a1) \& ?r ?Yi \) using subset-is-non-overlapping by metis
have \( ?a1 \in \) allocationsUniverse \& \( ?a2 \in \) allocationsUniverse
  using allocationUniverseProperty assms(1) reducedAllocation by fastforce
then have \( (?a1 = \{} \& ?a2 = \{}\) \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 \( ?r ?a2 \supseteq \{Y\} \)
      using Diff-cancel Range-insert empty-subsetI insert-Diff-single insert-iff insert-subset
      by (metis (hide-lams, 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 bm04\3 by fast
    moreover have \( \ldots = \{?U (?r ?a1), ?Y\} \) by force
    ultimately moreover have \( \forall x \in ?r ?a1. \forall y \in ?Yi. x \neq y \)
    using IntI empty-iff by metis
ultimately moreover have $\forall \ x \in \exists r \ ?a1. \ \forall \ y\in \exists Yi. \ x \cap \ y = \{\}$
using 3 4 6 is-non-overlapping-def by (metis set-rev-mp)
ultimately have $\exists U \ ?(\exists r \ ?a1) \cap \ ?Y = \{\}$ using unionIntersectionEmpty
proof –
have $\forall v0. \ v0 \in \text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a) \longrightarrow (\forall v1. \ v1 \in a \ " \ (X \cup \{i\}) \longrightarrow v0 \cap v1 = \{\})$
by (metis (no-types) $\forall x\in\text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a). \ \forall y\in a \ " \ (X \cup \{i\}). \ \ x \cap \ y = \{\})$
thus $\bigcup \text{Range} \ (a - (X \cup \{i\}) \times \text{Range} \ a) \cap \bigcup (a \ " \ (X \cup \{i\})) = \{\}$ by blast
qed
then have
$\exists U \ ?(\exists r \ ?a1) \cap \ ?U \ (\exists r \ ?a2) = \{\}$ by blast
moreover have $\exists d \ ?a1 \cap \ ?(\exists d \ ?a2) = \{\}$ by blast
moreover have $\exists a1 \in \text{allocationsUniverse} \ \ \text{using} \ \ \text{assms}(I) \ \text{reducedAllocation}$
by blast
moreover have $\exists a2 \in \text{allocationsUniverse} \ \text{using} \ \text{allocationUniverseProperty}$
by fastforce
ultimately have $\exists a1 \in \text{allocationsUniverse} \ & \ \exists a2 \in \text{allocationsUniverse} \ \ \ \bigcup \text{Range} \ ?a1 \cap \bigcup \text{Range} \ ?a2 = \{\} \ \& \ \text{Domain} \ ?a1 \cap \text{Domain} \ ?a2 = \{\}$
by blast
then have $\exists U \ \text{using} \ \text{allocationUnion} \ \text{by} \ \text{auto}$
then have $\exists \text{thesis} \ \text{using} \ I \ 7 \ \text{by} \ \text{simp}$
} then show $\exists \text{thesis} \ \text{using} \ 5 \ \text{by} \ \text{linarith}$
qed

**corollary allocationsUniverseOutsideUnion:**
assumes $a \in \text{allocationsUniverse}$
shows $(a \ outside \ (X\cup\{i\})) \cup \{\{\}\} \times (\bigcup(a^{\ "}(X\cup\{i\}))-\{\{\}\}) \in \text{allocationUniverse} \ & \ \bigcup (\text{Range}((a \ outside \ (X\cup\{i\}))) \cup \{\{\}\} \times (\bigcup(a^{\ "}(X\cup\{i\}))-\{\{\}\}))) = \bigcup (\text{Range} \ a)$
proof –
have $a - ((X\cup\{i\})\times(\text{Range} \ a)) = a \ outside \ (X \cup \{i\}) \ \text{using} \ \text{Outside-def by} \ \text{metis}$
moreover have $(a - ((X\cup\{i\})\times(\text{Range} \ a))) \cup (\{\{\}\} \times (\bigcup(a^{\ "}(X \cup \{i\}))) - \{\{\}\})) \in \text{allocationsUniverse}$
using assms ln044 by fastforce
moreover have $\bigcup (\text{Range} \ ((a - ((X\cup\{i\})\times(\text{Range} \ a))) \cup (\{\{\}\} \times (\bigcup(a^{\ "}(X \cup \{i\}))) - \{\{\}\}))) = \bigcup (\text{Range} \ a)$
using assms ln044 by (metis (no-types))
ultimately have $(a \ outside \ (X\cup\{i\})) \cup (\{\{\}\} \times (\bigcup(a^{\ "}(X \cup \{i\}))) - \{\{\}\})) \in \text{allocationsUniverse} \ & \ \bigcup (\text{Range} \ (((a \ outside \ (X\cup\{i\})) \cup (\{\{\}\} \times (\bigcup(a^{\ "}(X \cup \{i\}))) - \{\{\}\}))) = \bigcup (\text{Range} \ a) \bigcup
by simp
moreover have \( \{i, \bigcup (a''(X \cup \{i\}))\} - \{(\emptyset)\} = \{i\} \times (\bigcup (a''(X \cup \{i\}))\) \)
by fast
ultimately show {?thesis by auto
qed

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

corollary \textit{lm046}:
\[\text{assumes } \text{Domain} \ a \cap X \neq \emptyset \quad a \in \text{allocationsUniverse}\]
\[\text{shows } \{\bigcup (a''(X \cup \{i\}))\} - \{(\emptyset)\} = \{\bigcup (a''(X \cup \{i\}))\}\]
using \text{assms lm045 by fast}

corollary \textit{lm047}:
\[\text{assumes } a \in \text{allocationsUniverse}\]
\[\text{(\text{Domain} \ a) \cap X \neq \emptyset}\]
\[\text{shows } (a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\bigcup (a''(X \cup \{i\}))) \in \text{allocationsUniverse}\]
&
\[\bigcup (\text{Range}((a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\bigcup (a''(X \cup \{i\}))))))) = \bigcup (\text{Range} a)\]
proof –
let \(?t1 = (a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\bigcup (a''(X \cup \{i\})))) - \{(\emptyset)\}) \in \text{allocationsUniverse}\]
let \(?t2 = \bigcup (\text{Range}((a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\bigcup (a''(X \cup \{i\})))) - \{(\emptyset)\})) = \bigcup (\text{Range} a)\]
have \(\emptyset: a \in \text{allocationsUniverse using assms(1) by fast}\)
then have \(?t1 & ?t2 using allocationsUniverseOutsideUnion\)
proof –
have a \in \text{allocationsUniverse} ---
a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times (\bigcup (a'' (X \cup \{i\})))) - \{(\emptyset)\}) \in \text{allocationsUniverse}\]
using \text{allocationsUniverseOutsideUnion by fastforce}
hence a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times (\bigcup (a'' (X \cup \{i\})))) - \{(\emptyset)\}) \in \text{allocationsUniverse}
by (metis 0)
thus \( a \) outside \((X \cup \{i\}) \cup \{i\} \times \{(\bigcup (a''(X \cup \{i\}))) - \{\}\}) \in \\text{allocationsUniverse} \land \bigcup \text{Range } (a \) outside \((X \cup \{i\}) \cup \{i\} \times \{(\bigcup (a''(X \cup \{i\}))) - \{\}\})) = \bigcup \text{Range } a\)
using \( 0 \) by (metis (no-types) allocationsUniverseOutsideUnion)

qed

moreover have
\( \{(\bigcup (a''(X \cup \{i\}))) - \{\}\} = \{(\bigcup (a''(X \cup \{i\})))\} \) using \( \text{ln045 assms by fast} \)
ultimately show \( \text{thesis by auto} \)

qed

abbreviation
\begin{align*}
\text{bidMonotonicity } b \ i & \equiv \\
& (\forall \ t \ t'. (\text{trivial } t \land \text{trivial } t' \land \text{Union } t \subseteq \text{Union } t') \longrightarrow \\
& \text{sum } b \ ( \{i\} \times t) \leq \text{sum } b \ ( \{i\} \times t')
\end{align*}

lemma \( \text{lm048:} \)
asumes \( \text{bidMonotonicity } b \ i \text{ runiq } a \)
shows \( \text{sum } b \ ( \{i\} \times ((a \text{ outside } X)''\{i\})) \leq \text{sum } b \ ( \{i\} \times (\bigcup (a''(X \cup \{i\})))\))

proof –
let \( ?u = \text{runiq} \)
let \( ?I = \{i\} \)
let \( ?R = a \text{ outside } X \)
let \( ?U = \text{Union} \)
let \( ?Xi = X \cup ?I \)
let \( ?t1 = ?R''?I \)
let \( ?t2 = \{?U \ (a''?Xi)\} \)

have \( ?U \ (\ ?R''?I) \subseteq ?U \ (\ ?R''(X \cup ?I)) \) by blast
moreover have \( ... \subseteq ?U \ (a''(X \cup ?I)) \) using \( \text{Outside-def by blast} \)
ultimately have \( ?U \ (\ ?R''?I) \subseteq ?U \ (a''(X \cup ?I)) \) by auto
then have
\( \emptyset: ?U \ ?t1 \subseteq ?U \ ?t2 \) by auto
have \( ?u a \) using \( \text{assms by fast} \)
moreover have \( ?R \subseteq a \) using \( \text{Outside-def by blast ultimately} \)
have \( ?u ?R \) using \( \text{subrel-runiq by metis} \)
then have \( \text{trivial } ?t1 \) by (metis \( \text{runiq-alt} \))
moreover have \( \text{trivial } ?t2 \) by (metis \( \text{trivial-singleton} \))
ultimately show \( \text{thesis using } \text{assms } 0 \) by blast
qed

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

corollary \( \text{emptyNotInRange:} \)

78
assumes $a \in \text{allAllocations} \ N \ G$
shows $\emptyset \notin \text{Range} \ a$
using assms lm049 allAllocationsUniverse by auto blast

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

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

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

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

proof
let $\mathcal{R} = a \text{ outside} \ X$
let $\mathcal{I} = \{i\}$
let $\mathcal{U} = \text{Union}$
let $u = \text{runiq}$
let $r = \text{Range}$
let $d = \text{Domain}$
let $?aa = a \text{ outside} \ (X \cup \{i\}) \cup (\{i\} \times \{?U (a'' (X \cup \{i\})))\})$

have $1: a \in \text{allocationsUniverse}$ using assms(1) allAllocationsUniverse set-rev-mp by blast

have $i \notin X$ using assms by fast
then have $2: \ ?d \ a - X \cup \{i\} = \ ?d \ a \cup \{i\} - X$ by fast
have $a \in \text{allocationsUniverse}$ using $1$ by fast
moreover have $\ ?d \ a \cap X \neq \{\}$ using assms by fast
ultimately have $\ ?aa \in \text{allocationsUniverse}$ & $\ ?U (\ ?r \ ?aa) = ?U (\ ?r \ a)$ apply
(rule lm047) done

then have \( ?aa \in \text{allAllocations} \ (\ ?d \ ?a) \) (\ ?r a \)
using lm047 by (metis (lifting, mono-tags))
then have \( ?aa \in \text{allAllocations} \ (\ ?d \ ?a \cup \ (\ ?d a - X \cup \ {i})) \) (\ ?r a \)
by (metis lm023)
moreover have \( ?d a - X \cup \ {i} = ?d a \cup (\ ?d a - X \cup \ {i}) \)
by auto
ultimately have \( ?aa \in \text{allAllocations} \ (\ ?d a \cup \ (\ ?d a - X \cup \ {i})) \) (\ ?r a \)
by simp
then have \( ?aa \in \text{allAllocations} \ (\ ?d a \cup \ (\ ?d a - X \cup \ {i})) \) (\ ?r a \)
by (metis lm024)
then show \( \text{thesis} \) by fast
qed

lemma lm054:
assumes bidMonotonicity (b::==> real) i
a \in \text{allocationsUniverse}
Domain a \cap X \neq \{\}
finite a
shows \( \sum b \ (a \ outside \ X) \leq \sum b \ (a \ outside \ (X \cup \ {i}) \cup (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) \)
proof --
let \( \mathcal{R} = a \ outside \ X \)
let \( \mathcal{I} = \{i\} \)
let \( \mathcal{U} = \text{Union} \)
let \( \mathcal{R}'' = \text{runiq} \)
let \( \mathcal{R} = \text{Range} \)
let \( \mathcal{d} = \text{Domain} \)
let \( ?aa = a \ outside \ (X \cup \ {i}) \cup (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) \)
have a \in \text{injectionsUniverse} using assms by fast
then have
0: \( ?u a \ a \ by \ simp \)
moreover have \( ?R \subseteq a \ & \ ?R--i \subseteq a \) using Outside-def using lm088 by auto
ultimately have finite \( (\ ?R -- i) \ & \ ?u (\ ?R--i) \ & \ ?u ?R \)
using finite-subset subrel-runiq by (metis assms(4))
then moreover have trivial \( (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) \) using runiq-def
by (metis trivial-cartesian trivial-singleton)
moreover have \( \forall X. \ (\ ?R -- i) \cap (\ {i} \times X) = \{\} \) using outside-reduces-domain
by force
ultimately have
1: finite \( (\ ?R--i) \ & \ finite (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) \ & \ (\ ?R -- i) \cap (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) = \{\} \)
&
finite \( (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) \ & \ (\ ?R -- i) \cap (\ {i} \times (\bigcup (a''(X\cup{\{i\}))))))) = \{\} \)
using Outside-def trivial-implies-finite by fast
have \( \mathcal{R} = (\mathcal{R} - i) \cup \{(i) \times \mathcal{R}''\{i\}\} \) by (metis outsideUnion)
then have \( \sum b \, \mathcal{R} = \sum b (\mathcal{R} - i) + \sum b \{(i) \times (\mathcal{R}''\{i\})\} \)
  using 1 sum.union-disjoint by (metis (lifting) sum.union-disjoint)
moreover have \( \sum b \{(i) \times (\mathcal{R}''\{i\})\} \leq \sum b \{(i) \times \{U (a''(X \cup \{i\}))\}\} \)
  using lm048 assms by blast
ultimately show \( \sum b \mathcal{R} \leq \sum b (\mathcal{R} - i) + \sum b \{(i) \times \{U (a''(X \cup \{i\}))\}\} \)
by linarith
moreover have ...
ultimately have \( \sum b \mathcal{R} = \sum b (\mathcal{R} - i) \cup \{(i) \times \{U (a''(X \cup \{i\}))\}\} \)
using 1 sum.union-disjoint by auto
moreover have ...
ultimately show \( ?thesis \) by simp
qed

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

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

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

corollary lm056:
  assumes bidMonotonicity (b::-\( \Rightarrow \) real) i
  a \( \in \) allAllocations N G
  i \( \in \) N-\(\sim\)X
  Domain a \( \cap \) X \( \neq \) \{\}
  finite N
  finite G
  shows Max ((\sum b)'(allAllocations (N-\(\sim\)X) G)) \( \geq \)
  \( \sum b (a \text{ outside X}) \)
proof –
  let \( ?aa = a \text{ outside } (X \cup \{i\}) \cup \{(i) \times \{U (a''(X \cup \{i\}))\}\} \)
  have bidMonotonicity (b::-\( \Rightarrow \) real) i using assms(1) by fast
  moreover have a \( \in \) allocationsUniverse using assms(2) allAllocationsUniverse
  by blast
  moreover have Domain a \( \cap \) X \( \neq \) \{\} using assms(4) by auto
  moreover have finite a using assms lm055 by metis

81
ultimately have
\( \theta: \sum b (a \text{ outside } X) \leq \sum b ?aa \) by (rule lm054)

have \( ?aa \in \text{allAllocations} (N - X) (\bigcup (\text{Range } a)) \) using assms lm053 by metis

moreover have \( \sum b ?aa \in (\sum b') (\text{allAllocations} (N - X) G) \) by (metis imageI)

ultimately have \( \sum b ?aa \in (\sum b) '\) (\text{allAllocations} (N - X) G) by (metis)

moreover have \( \bigcup (\text{Range } a) = G \) using assms allocationInverseRangeDomainProperty is-partition-of-def by fast

ultimately have \( \sum b ?aa \in (\sum b)'(\text{allAllocations} (N - X) G) \) by (metis)

then show \( ?\text{thesis} \) using \( \theta \) by linarith

qed

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

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

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 \( \sum f (\bigcup C) = \sum (\sum f) C \)
using assms sum,Union-disjoint by fastforce

corollary sumUnionDisjoint2:
assumes \( \forall x \in X. \text{finite } x \text{ is-non-overlapping } X \)
shows \( \sum f (\bigcup X) = \sum (\sum f) X \)
using assms sumUnionDisjoint1 is-non-overlapping-def by fast

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

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

lemma lm057:
assumes \( a \in \text{allocationsUniverse Domain} a \subseteq N (\bigcup \text{Range } a) = G \)

82
shows \ a \in\ allAllocations\ N\ G
using\ \text{assms}\ \text{possibleAllocationsRelCharacterization}\ \text{lm040} \ \text{by}\ \text{(metis (mono-tags, lifting))}

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

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

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

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

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

corollary\ \text{allAllocationsIntersection}:
\text{allAllocations}\ N\ G\ \text{=}\ \text{(allocationsUniverse}\ \cap\ \{a.\ (\text{Domain}\ a)\ \subseteq\ N\ &\ (\bigcup\ Range\ a) = G\})
(is \ ?L\ =\ \?R)
proof –
have \ ?L\ \subseteq\ \?R\ \text{using allAllocationsIntersectionSubSet by metis}
moreover have \ ?R\ \subseteq\ \?L\ \text{using lm058 by fast}
ultimately show \ ?thesis\ \text{by force}
qed

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

corollary\ \text{allocationsUniverseOutside}:
assumes\ \text{a}\ \in\ \text{allocationsUniverse}
shows a outside $X \in \text{allocationsUniverse}$
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 $\{ f \cup \{(x, y)| y \cdot y \in A\} \subseteq \text{runiqs} \}$
  unfolding paste-def runiqs-def using assms runiq-basic by blast

lemma lm066:
  converse ' (converse ' $X$) = $X$
  by auto

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

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

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

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

lemma \(\text{lm071}\):

\[\text{injectionsUniverse} = \text{runiq} \cap \text{converse'}\text{runiq}\]

unfolding \text{runiq-def} by auto

lemma \(\text{lm072}\):

assumes \(f \in \text{injectionsUniverse}\) \(x \notin \text{Domain}\) \(A \cap (\text{Range}\ f) = \{\}\)

shows \[\{f \cup \{(x, y)\} \mid y \in A\} \subseteq \text{injectionsUniverse}\]

(is \(\forall l \subseteq \forall r\))

proof –

have \(f \in \text{converse'}\text{runiq}\) using \text{assms} \((1)\) \(\text{lm071}\) by blast

then have \(\forall l \subseteq \text{converse'}\text{runiq}\) using \text{assms} \((3)\) by (rule \text{lm069})

moreover have \(\forall l \subseteq \text{runiq}\) using \text{assms} \((1, 2)\) \(\text{lm065}\) by force

ultimately show \(\forall \text{thesis}\) using \(\text{lm071}\) by blast

qed

lemma \(\text{lm073}\):

\[\text{injections} X Y = \text{totalRels} X Y \cap \text{injectionsUniverse}\]

using \(\text{lm008}\) by metis

lemma \(\text{lm074}\):

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

shows \(f\) outside \(A \in \text{injectionsUniverse}\)

using \text{assms} by (metis (no-types) \text{Outside-def} \text{lm030})

lemma \(\text{lm075}\):

assumes \(R \in \text{totalRels} A B\)

shows \(R\) outside \(C \in \text{totalRels} (A \setminus C) B\)

unfolding \text{Outside-def} using \text{assms} by blast

lemma \(\text{lm076}\):

assumes \(g \in \text{injections} A B\)

shows \(g\) outside \(C \in \text{injections} (A \setminus C) B\)

using \text{assms} \text{Outside-def} \text{Range-outside-sub} \text{lm030} \text{mem-Collect-eq} \text{outside-reduces-domain}

unfolding \text{injections-def}

by fastforce

lemma \(\text{lm077}\):

assumes \(g \in \text{injections} A B\)

shows \(g\) outside \(C \in \text{injections} (A \setminus C) B\)

using \text{assms} \text{lm076} by metis

lemma \(\text{lm078}\):

\[\{x\} \times \{y\} = \{(x, y)\}\]

by simp

lemma \(\text{lm079}\):

assumes \(x \in \text{Domain}\) \(f\) \(\text{runiq}\) \(f\)
shows \( \{x\} \times f''(x) = \{(x,f,x)\} \)

using assms lm078 Image-runiq-eq-eval by metis

**corollary** lm080:

- **assumes** \( x \in \text{Domain } f \text{ runiq } f \)
- **shows** \( f = (f -\rightarrow x) \cup \{(x,f,x)\} \)
- **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** \( g \in \{f -\rightarrow x \cup \{(x,y)\}|y \in Y - (\text{Range}(f -\rightarrow x))\} \)
- **proof** –
  - **let** \( ?f = g -\rightarrow x \)
  - **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 insertI1 mem-Collect-eq)
  - **ultimately have** \( g,x \in Y - \text{Range } ?f \)
    - using lm081 assms(1) unfolding injections-def
    - by fast
  - **moreover have** \( g = ?f \cup \{(x,g,x)\} \)
    - using assms lm080 mem-Collect-eq unfolding injections-def by (metis (lifting))

- **ultimately show** \( \theta \text{thesis by blast} \)

**qed**

**corollary** lm083:

- **assumes** \( x \notin X \ g \in \text{injections } \{\{x\} \cup X\} \ Y \)
- **shows** \( g -\rightarrow x \in \text{injections } X \ Y \)
- **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** \( g \in \text{injections } \{?X\} \ Y \)
- **shows** \( \exists f \in \text{injections } X \ Y. \ g \in \{f \cup \{(x,y)\}|y \in Y - (\text{Range } f)\} \)
- **proof** –
  - **let** \( ?f = g -\rightarrow x \)
  - **have** \( \theta: g\in \text{injections } \{?X\} \ Y \) using assms by metis
  - **have** \( \text{Domain } g = ?X \)
    - using assms(2) mem-Collect-eq unfolding injections-def by (metis (mono-tags, lifting))
  - **then have** \( 1: x \in \text{Domain } g \) by simp then have \( ?f \in \text{injections } X \ Y \) using assms lm083
    - **by** fast
  - **moreover have** \( g \in \{?f \cup \{(x,y)\}|y \in Y - \text{Range } ?f\} \)
    - using 0 1 by (rule lm082)
ultimately show \[\text{thesis}\] by blast

\[\text{qed}\]

corollary \text{lm085}:
assumes \(x \notin X\)
shows \(\text{injections } (\{x\} \cup X) Y \subseteq (\bigcup f \in \text{injections } X \rightarrow Y. \{f \cup \{(x, y)\} | y \in Y - (\text{Range } f)\})\)
using asms \text{lm084} by auto

lemma \text{lm086}:
assumes \(x \notin X\)
shows \(\{f \cup \{(x, y)\} | y \notin Y - \text{Range } f\} \subseteq \text{injections } (\{x\} \cup X) Y\)
using asms \text{lm072} \text{injections-def} \text{ lm073} \text{ lm070}

proof
\{ fix \(f\)
assume \(f \in \text{injections } X \rightarrow Y\)
then have
0: \(f \in \text{injectionsUniverse} \& x \notin \text{Domain } f \& \text{Domain } f = X \& \text{Range } f \subseteq Y\)
using asms unfolding \text{injections-def} by fast
then have \(f \in \text{injectionsUniverse}\) by fast
moreover have \(x \notin \text{Domain } f\) using 0 by fast
moreover have \(Y - \text{Range } f\) \cap \text{Range } f = \{\}\) by blast
ultimately have \(\{f \cup \{(x, y)\} | y \notin Y - \text{Range } f\} \subseteq \text{injectionsUniverse}\)
by (rule \text{lm072})
moreover have \(\{f \cup \{(x, y)\} | y \notin Y - \text{Range } f\} \subseteq \text{totalRels } (\{x\} \cup X) Y\)
using \text{lm070} 0 by force
ultimately have \(\{f \cup \{(x, y)\} | y \notin Y - \text{Range } f\} \subseteq \text{injectionsUniverse} \cap \text{totalRels } (\{x\} \cup X) Y\)
by auto
\}
thus \[\text{thesis}\] using \text{lm008} unfolding \text{injections-def} by blast
\[\text{qed}\]

corollary \text{injectionsUnionCommute}:
assumes \(x \notin X\)
shows \(\bigcup f \in \text{injections } X \rightarrow Y. \{f \cup \{(x, y)\} | y \notin Y - \text{Range } f\} = \text{injections } (\{x\} \cup X) Y\)
(is \(r=\text{injections } ?X -\))

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

lemma lm087:
  assumes ∀ x. (P x → (f x = g x))
  shows Union {f x| x. P x} = Union {g x | x. P x}
  using assms by blast

lemma lm088:
  assumes x /∈ Domain R
  shows R ++ {(x,y)} = R ∪ {(x,y)}
  using assms by (metis (erased, lifting) Domain-empty Domain-insert Int-insert-right-if0
disjoint-iff-not-equal ex-in-conv paste-disj-domains)

lemma lm089:
  assumes x /∈ X
  shows (⋃ f ∈ injections X ?Y. {f ++ {(x, y)} | y . y ∈ Y − Range f}) =
    (⋃ f ∈ injections X ?Y. {f ∪ {(x, y)} | y . y ∈ Y − Range f})
  (is ?l = ?r)
  proof –
  have 0: ∀ f ∈ injections X ?Y. x /∈ Domain f unfolding injections-def using assms
  by fast
    then have
      1: ?l = Union {{f ++ {(x, y)} | y . y ∈ Y − Range f}| f . f ∈ injections X ?Y & x
                   /∈ Domain f}
      (is ¬ (?) using assms by auto
        moreover have
          2: ?r = Union {{f ∪ {(x, y)} | y . y ∈ Y − Range f}| f . f ∈ injections X ?Y & x
                         /∈ Domain f}
      (is ¬ (?) using assms 0 by auto
        have ∀ f. f ∈ injections X ?Y & x /∈ Domain f →
                            {f ++ {(x, y)} | y . y ∈ Y − Range f} =
                         {f ∪ {(x, y)} | y . y ∈ Y − Range f} |
        using lm088 by force
        then have ?l = ?r by (rule lm087)
        then show ?l = ?r using 1 2 by presburger
      qed
    qed
  corollary lm090:
    assumes x /∈ X
    shows (⋃ f ∈ injections X ?Y. {f ++ {(x, y)} | y . y ∈ Y − Range f}) =
      injections {(x) ∪ X} ?Y
    (is ?l = ?r)
    proof –
    have ?l = (⋃ f ∈ injections X ?Y. {f ∪ {(x, y)} | y . y ∈ Y − Range f}) using
assms by (rule lm089)
morhmore have ... = ?r using assms by (rule injectionsUnionCommute)
ultimately show ?thesis by simp
qed

lemma lm091:
set {f ∪ {(x,y)} . y ← (filter (%y. y ∉ (Range f)) Y } ] =
{f ∪ {(x,y)} | y . y ∈ (set Y) − (Range f)}
by auto

lemma lm092:
assumes ∀ x ∈ set L. set (F x) = G x
shows set (concat [ F x . x ≤ L]) = (∪ x∈set L. G x)
using assms by force

lemma lm093:
set (concat [ [ f ∪ {(x,y)} . y ← (filter (%y. y ∉ Range f) Y ). f ← F ] ] =
(∪ f ∈ set F. {f ∪ {(x,y)} | y . y ∈ (set Y) − (Range f)})
by auto

lemma lm094:
assumes finite Y
shows set [ f +* {(x,y)} . y ← sorted-list-of-set (Y − (Range f)) ] =
{ f +* {(x,y)} | y . y ∈ Y − (Range f)}/nusing assms by auto

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

9.5 Computable injections

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

corollary lm096:
set (injectionsAlg (x # xs) Y) =
(∪ f ∈ set (injectionsAlg xs Y). {f ∪ {(x,y)} | y . y ∈ (set Y) − (Range f)})
using lm093 by auto
corollary lm097:
assumes set (injectionsAlg xs Y) = injections (set xs) (set Y)
shows set (injectionsAlg (x ≠ xs) Y) = 
  ( ⋃ f ∈ injections (set xs) (set Y). {f ∪ {(x,y)} | y . y ∈ (set Y) – (Range f)})
using assms lm096 by auto

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

lemma lm098:
injections {} Y = {{}}
by (simp add: lm008 lm062 runiq-emptyrel)

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

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

lemma lm100:
assumes x /∈ set xs set (injectionsAlg xs Y) = injections (set xs) (set Y)
shows set (injectionsAlg (x # xs) Y) = injections ({x} ∪ set xs) (set Y)
(is ?l=?r)
proof –
  have ?l = ( ⋃ f ∈ injections (set xs) (set Y). {f ∪ {(x,y)} | y . y ∈ (set Y) – Range f})
  using assms(2) by (rule lm097)
  moreover have ... = ?r using assms(1) by (rule injectionsUnionCommute)
  ultimately show ?thesis by simp
qed

lemma lm101:
assumes x /∈ set xs
set (injections-alg xs Y) = injections (set xs) Y
finite Y
shows set (injections-alg (x#xs) Y) = injections ({x} ∪ set xs) Y
(is ?l=?r)
proof –
  have ?l = ( ⋃ f ∈ injections (set xs) Y. {f +* {(x,y)} | y . y ∈ Y – Range f})
  using assms(2,3) lm095 by auto
  moreover have ... = ?r using assms(1) by (rule lm090)
  ultimately show ?thesis by simp
qed
lemma listInduct:
assumes \( P \emptyset \forall xs x. P xs \rightarrow P (x\#xs) \)
shows \( \forall x. P x \)
using assms by (metis structInduct)

lemma injectionsFromEmptyAreEmpty:
set \( \text{injections-alg } \emptyset Z \) = \{\}\nby simp

theorem injections-eqv:
assumes finite \( Y \) and distinct \( X \)
shows set \( \text{injections-alg } X Y \) = \( \text{injections } (\text{set } X) Y \)
proof –
let \( \lambda l. \text{distinct } l \rightarrow (\text{set } \text{injections-alg } l Y) = \text{injections } (\text{set } l) Y \)
have \( \lambda l \) using injectionsFromEmptyAreEmpty list(1) lm099 by metis
moreover have \( \forall x xs. \lambda P xs \rightarrow \lambda P (x\#xs) \)
using assms(1) lm101 by (metis distinct.simps(2) insert-is-Un list.simps(15))
ultimately have \( \lambda P X \) by (rule structInduct)
then show \( \lambda \)thesis using assms(2) by blast
qed

lemma lm102:
assumes \( l \in \text{set } (\text{all-partitions-list } G) \) distinct \( G \)
shows distinct \( l \)
using assms by (metis all-partitions-equivalence)

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

lemma lm103:
assumes card \( N > 0 \) distinct \( G \)
shows \( \{\text{injections } P N \mid P. P \in \text{all-partitions } (\text{set } G)\}\) = \( \text{set } \{\text{injections-alg } l N \mid l \in \text{all-partitions-list } G\} \)
proof –
let \( ?g1 = \text{all-partitions-list} \)
let \( ?f2 = \text{injections} \)
let \( ?g2 = \text{injections-alg} \)
have \( \forall l \in \text{set } (?g1 G). \text{set } (?g2 l N) = ?f2 (\text{set } l) N \) using assms bridgingInjection by blast
then have set \( \{\text{set } (?g2 l N). l \leftarrow ?g1 G\} = \{?f2 P N \mid P. P \in \text{set } (\text{map set } (?g1 G))\} \)
apply (rule setVsList) done
moreover have \( \ldots = \{?f2 P N \mid P. P \in \text{all-partitions } (\text{set } G)\} \)
lemma \textit{lm104}:  
\begin{itemize}
  \item \textbf{assumes} $\text{card } N > 0$ distinct $G$
  \item \textbf{shows} $\bigcup \{ \text{injections } P N \mid P. P \in \text{all-partitions } (\text{set } G) \} = \bigcup \{ \text{set } (\text{injections-alg } l \ N) \mid l \leftarrow \text{all-partitions-list } G \}$
\end{itemize}

\textbf{(is $\bigcup \{ \text{injections } P N \mid P. P \in \text{all-partitions } (\text{set } G) \} = \bigcup \{ \text{set } (\text{injections-alg } l \ N) \mid l \leftarrow \text{all-partitions-list } G \}$)}

\textbf{proof} –

\begin{itemize}
  \item \textbf{have} $?L = ?R$ \textbf{using} \textit{assms} \textbf{by} (rule \textit{lm103}) \textbf{thus} $?\text{thesis}$ \textbf{by} \textit{presburger}
\end{itemize}

\textbf{qed}

corollary \textit{allAllocationsBridgingLemma}:
\begin{itemize}
  \item \textbf{assumes} $\text{card } N > 0$ distinct $G$
  \item \textbf{shows} $\text{allAllocations } N (\text{set } G) = \text{set}(\text{allAllocationsAlg } N G)$
\end{itemize}

\textbf{proof} –

\begin{itemize}
  \item \textbf{let} $?LL = \bigcup \{ \text{injections } P N \mid P. P \in \text{all-partitions } (\text{set } G) \}$
  \item \textbf{let} $?RR = \bigcup \{ \text{set } (\text{injections-alg } l \ N) \mid l \leftarrow \text{all-partitions-list } G \}$
  \item \textbf{have} $?LL = ?RR$ \textbf{using} \textit{assms} \textbf{by} (rule \textit{lm104})
  \item \textbf{then have} $\text{converse} ' ?LL = ?RR$ \textbf{by} \textit{simp}
  \item \textbf{thus} $?\text{thesis}$ \textbf{by} \textit{force}
\end{itemize}

\textbf{qed}

end

10 Termination theorem for uniform tie-breaking

theory \textit{UniformTieBreaking}

imports
\textit{StrictCombinatorialAuction}
\textit{Universes}
\textit{HOL-Library.Code-Target-Nat}

begin

10.1 Uniform tie breaking: definitions

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

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

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

**definition** pseudoAllocation allocation == \bigcup (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) \cdot (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 
  by (metis assms bot.not-eq-extremum ex-in-conv gr-implies-not0 in-set-conv-nth  
                   randomEl-def semiring-numeral-div-class.pos-mod-bound)

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)×{{X}}−{{}}} ∈ allocationsUniverse  
  using allocationUniverseProperty pairDifference by metis

corollary lm003:
  {(x,{y})} ∈ allocationsUniverse  
  proof −
    have ∃x1. {} − {x1::'a × 'b set} = {} by simp 
    thus {(x, {y})} ∈ allocationsUniverse 
    by (metis (no-types) allocationUniverseProperty empty-iff insert-Diff-if insert-iff)
corollary lm004:
allocationsUniverse ≠ {}
using lm003 by fast

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

lemma lm006:
assumes G ≠ {}
shows {G} ∈ all-partitions G
using all-partitions-def is-partition-of-def is-non-overlapping-def assms by force

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

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

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

corollary lm010:
assumes N ≠ {} G≠{}
shows allAllocations N G ≠ {}
using assms lm009
by (metis (hide-lams, no-types) equals0I image-insert insert-absorb insert-not-empty)

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

lemma lm012:
lemma lm013:
  assumes \( a \in \text{allAllocations} \ N \ G \ \text{finite} \ G \)
  shows \( \text{finite} \ (\text{Range} \ a) \)
  using assms \( \text{elementOfPartitionOfFiniteSetIsFinite} \) by (metis allocationReversalInj)

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

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

10.3 Results on summed bid vectors

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

corollary lm017:
  \( \{(\text{pair}, \text{sum} (\%g. \ \text{bids} (\text{fst pair}, g)) (\text{finestpart} (\text{snd pair}))) | \text{pair. pair} \in (N \times (\text{Pow} \ G - \{}{}\)) \} \setminus 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:
  \( (\text{summedBidVectorRel} \ \text{bids} \ N \ G) \setminus a = \)
  \( \{(\text{pair}, \text{sum} (\%g. \ \text{bids} (\text{fst pair}, g)) (\text{finestpart} (\text{snd pair}))) | \)
proof –
let \( l = \text{summedBidVectorRel} \)
let \( M = \{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) (\text{finestpart} (\text{snd pair}))) | \text{pair}, \text{pair} \in (N \times (\text{Pow G} - \{\{\}\})) \cap a\} \)
have \( l \text{ bids } N \text{ G} = M \) by (rule lmo16)
then have \( L = (M \| a) \) by presburger
moreover have \( ... = R \) by (rule lmo17)
ultimately show \( \text{thesis} \) by simp
qed

lemma lmo19:
\((\text{summedBid bids} ) \cdot ((N \times (\text{Pow G} - \{\{\}\})) \cap a) = \{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) (\text{finestpart} (\text{snd pair}))) | \text{pair}, \text{pair} \in (N \times (\text{Pow G} - \{\{\}\})) \cap a\}\) by blast

corollary lmo20:
\((\text{summedBidVectorRel bids N G} \| a = (\text{summedBid bids} ) \cdot ((N \times (\text{Pow G} - \{\{\}\})) \cap a)\) (is \?L=\?R)
proof –
let \( l=\text{summedBidVectorRel} \)
let \( p=\text{summedBid} \)
let \( M = \{(\text{pair}, \text{sum} (\% g. \text{bids} (\text{fst pair}, g)) (\text{finestpart} (\text{snd pair}))) | \text{pair}, \text{pair} \in (N \times (\text{Pow G} - \{\{\}\})) \cap a\} \)
have \( L = M \) by (rule lmo18)
moreover have \( ... = R \) using lmo19 by blast
ultimately show \( \text{thesis} \) by simp
qed

lemma summedBidInjective:
inj-on \((\text{summedBid bids}) \text{ UNIV} \)
using \( \text{fst-conv inj-on-inverseI} \) by (metis \( \text{lifting} \))

corollary lmo21:
inj-on \((\text{summedBid bids}) X \)
using \( \text{fst-conv inj-on-inverseI} \) by (metis \( \text{lifting} \))

lemma lmo22:
\( \text{sum snd} (\text{summedBidVectorRel bids N G}) = \text{sum} (\text{snd} \circ (\text{summedBid bids})) (N \times (\text{Pow G} - \{\{\}\})) \)
using lmo21 \( \text{sum.reindex} \) by blast
**corollary** \( \text{lm023} \):
\[
\text{snd} (\text{summedBid bids pair}) = \text{sum bids} (\omega \text{pair})
\]
\text{using} \ \text{sumCurry by force}

**corollary** \( \text{lm024} \):
\[
\text{snd} \circ \text{summedBid bids} = (\text{sum bids}) \circ \omega
\]
\text{using} \ \text{lm023 by fastforce}

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

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

**lemma** \( \text{lm027} \):
\text{assumes} \ \emptyset \not\in \text{Range } f \text{ runiq } f
\text{shows} \ \text{is-non-overlapping} (\omega ' f)
\text{proof –}
\begin{align*}
& \text{let } ?X=\omega ' f \text{ let } ?p=\text{finestpart} \\
& \{ \text{fix } y1 y2 \\
& \quad \text{assume } y1 \in ?X \land y2 \in ?X \\
& \quad \text{then obtain } \text{pair1 pair2 where} \\
& \quad \quad y1 = \omega \text{pair1} \land y2 = \omega \text{pair2} \land \text{pair1} \in f \land \text{pair2} \in f \text{ by blast} \\
& \quad \text{then moreover have } \text{snd pair1} \neq \{\} \land \text{snd pair1} \neq \{\} \\
& \quad \text{using assms by (metis rev-image-eqI snd-eq-Range)} \\
& \quad \text{ultimately moreover have } \text{fst pair1} = \text{fst pair2} \iff \text{pair1} = \text{pair2} \\
& \quad \text{using assms runiq-basic surjective-pairing by metis} \\
& \quad \text{ultimately moreover have } y1 \cap y2 \neq \{\} \rightarrow y1 = y2 \text{ using assms by fast} \\
& \quad \text{ultimately have } y1 = y2 \iff y1 \cap y2 \neq \{\} \\
& \quad \text{using assms notEmptyFinestpart by (metis Int-absorb Times-empty insert-not-empty)} \\
& \}\n\text{thus } ?\text{thesis using is-non-overlapping-def} \\
& \quad \text{by (metis (lifting, no-types) inf-commute inf-sup-aci(1))} \\
\text{qed}
\end{align*}

**lemma** \( \text{lm028} \):
\text{assumes} \ \emptyset \not\in \text{Range } X
\text{shows} \ \text{inj-on } \omega X
\text{proof –}
\begin{align*}
& \text{let } ?p=\text{finestpart} \\
& \{ \text{fix } \text{pair1 pair2} \\
& \quad \text{assume } \text{pair1} \in X \land \text{pair2} \in X \\
& \quad \text{then have } \text{snd pair1} \neq \{\} \land \text{snd pair2} \neq \{\}
\end{align*}
98
using assms by (metis Range.intros surjective-pairing)
moreover assume omega pair1 = omega pair2
then moreover have ?p (snd pair1) = ?p (snd pair2) by blast
then moreover have snd pair1 = snd pair2 by (metis finestPart nonEqualitySetOfSets)
ultimately moreover have {fst pair1} = {fst pair2} using notEmptyFinestpart
by (metis fst-image-times)
ultimately have pair1 = pair2 by (metis prod-eqI singleton-inject)
thus ?thesis by (metis (lifting, no-types) inj-onI)
qed

lemma lm029:
assumes \{\} \notin Range a finite (omega a) \forall X \in omega a. finite X
is-non-overlapping (omega a)
shows card (pseudoAllocation a) = sum (card o omega) a
(is ?L = ?R)
proof –
have ?L = sum card (omega a)
unfolding pseudoAllocation-def
using assms(2,3,4) by (rule cardinalityPreservation)
moreover have ... = ?R using assms(1) lm028 sum.reindex by blast
ultimately show ?thesis by simp
qed

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

corollary lm031:
card o omega = card o snd
using lm030 by fastforce

corollary lm032:
assumes \{\} \notin Range a \forall pair \in a. finite (snd pair) finite a runiq a
shows card (pseudoAllocation a) = sum (card o snd) a
proof –
let ?P=pseudoAllocation
let ?c=card
have \forall pair \in a. finite (omega pair) using finiteFinitestpart assms by blast
moreover have is-non-overlapping (omega a) using assms lm027 by force
ultimately have ?c (?P a) = sum (?c o omega) a using assms lm029 by force
moreover have ... = sum (?c o snd) a using lm031 by metis
ultimately show ?thesis by simp
qed

corollary lm033:
assumes runiq (a^−1) runiq a finite a \{\} \notin Range a \forall \text{ pair } \in a. \text{ finite (snd pair)}
shows card (pseudoAllocation a) = sum card (Range a)
using assms sumPairsInverse lm032 by force

corollary lm034:
assumes a \in allAllocations N G finite G
shows card (pseudoAllocation a) = card G
proof
\begin{itemize}
\item have \{\} \notin Range a using assms by (metis emptyNotInRange)
\item moreover have \forall \text{ pair } \in a. \text{ finite (snd pair)} using assms lm014 finitePairSecondRange by metis
\item moreover have \text{ finite a using assms by blast}
\item moreover have \text{ runiq a using assms by (metis (lifting) Int-lover1 in-mono injectionsUniversesProperty mem-Collect-eq)}
\item moreover have \text{ runiq (a^−1) using assms by (metis (mono-tags) injections-def characterization allAllocations mem-Collect-eq)}
\end{itemize}
ultimately have card (pseudoAllocation a) = sum card (Range a) using lm033 by fast
moreover have \ldots = card G using assms lm015 by metis
ultimately show \?thesis by simp
qed

corollary lm035:
assumes pseudoAllocation aa \subseteq pseudoAllocation a \cup (N \times (\text{finestpart G}))
finite (pseudoAllocation aa)
shows \text{ sum (toFunction (bidMaximizedBy a N G)) (pseudoAllocation a)} - (\text{ sum (toFunction (bidMaximizedBy a N G)) (pseudoAllocation aa)})
= card (pseudoAllocation a) - card (pseudoAllocation aa \cap (pseudoAllocation a))
using assms subsetCardinality by blast

corollary lm036:
assumes pseudoAllocation aa \subseteq pseudoAllocation a \cup (N \times (\text{finestpart G}))
finite (pseudoAllocation aa)
shows \text{ int (sum (maxbid a N G) (pseudoAllocation a))} - (\text{ int (sum (maxbid a N G) (pseudoAllocation aa))})
= \text{ int (card (pseudoAllocation a))} - \text{ int (card (pseudoAllocation aa \cap (pseudoAllocation a))})
using differenceSumVsCardinality assms by blast

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

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

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

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

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

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

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

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

lemma lm045:
  ∪ {{(fst pair, {y})| y. y ∈ snd pair}| pair. pair ∈ a} = 
   {{(fst pair, {y})| y pair. y ∈ snd pair & pair ∈ a} 
  by blast
corollary lm046:
\[ \text{pseudoAllocation } a = \{ (\text{fst pair}, Y) \mid Y \text{ pair. } Y \in \text{finestpart} (\text{snd pair}) \& \text{pair } \in \text{a} \} \]

unfolding pseudoAllocation-def using setOfPairsEquality by fastforce

lemma lm047:

assumes \( \text{runiq } a \)

shows \( \{ (\text{fst pair}, Y) \mid Y \text{ pair. } Y \in \text{finestpart} (\text{a},x) \& \text{x } \in \text{Domain } a \} \)

using assms Domain.\ DomainI \\text{fst-cone \ functionOnFirstEqualsSecond \ runiq-wrt-ex1 \ surjective-pairing}

by (metis (hide-lams,no-types))

corollary lm048:

assumes \( \text{runiq } a \)

shows \( \text{pseudoAllocation } a = \{ (x, Y) \mid Y x. Y \in \text{finestpart} (\text{a},,x) \& x \in \text{Domain } a \} \)

unfolding pseudoAllocation-def using assms lm047 lm046 by fastforce

corollary lm049:

Range (\text{pseudoAllocation } a) = \bigcup (\text{finestpart } \setminus (\text{Range } a))

unfolding pseudoAllocation-def

using lm046 rangeSetOfPairs unionFinestPart by fastforce

corollary lm050:

Range (\text{pseudoAllocation } a) = \text{finestpart} (\bigcup \text{Range } a)

using commuteUnionFinestpart lm049 by metis

lemma lm051:

\( \text{pseudoAllocation } a = \{ (\text{fst pair}, \{ y \}) \mid y \text{ pair. } y \in \text{snd pair} \& \text{pair } \in \text{a} \} \)

using lm044 lm045 by (metis (no-types))

lemma lm052:

\( \{ (\text{fst pair}, \{ y \}) \mid y \text{ pair. } y \in \text{snd pair} \& \text{pair } \in \text{a} \} = \)

\( \{ (x, \{ y \}) \mid x y. y \in \bigcup (\text{a}^{-1}\{x\}) \& x \in \text{Domain } a \} \)

by auto

lemma lm053:

\( \text{pseudoAllocation } a = \{ (x, \{ y \}) \mid x y. y \in \bigcup (\text{a}^{-1}\{x\}) \& x \in \text{Domain } a \} \)

(is \( \text{?L=?R} \))

proof

have \( \text{?L=}\{ (\text{fst pair}, \{ y \}) \mid y \text{ pair. } y \in \text{snd pair} \& \text{pair } \in \text{a} \} \) by (rule lm051)

moreover have \( \ldots \Rightarrow \text{?R} \) by (rule lm052)

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

qed

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

**corollary** lm055:
runiq \((\text{summedBidVectorRel bids } N \ G \ |\ | a)\)
unfolding restrict-def using lm054 subrel-runiq Int-commute 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 assms allocationPowerset by (metis(no-types))
moreover have \(N \times (\text{Pow } G - \{\{\}\}) = \text{Domain } (?L \text{ bids } N \ G)\) using summed-BidVectorCharacterization by blast
ultimately show ?thesis by blast
qed

**corollary** lm057:
sum \((\text{summedBidVector bids } N \ G)\) \((a \cap (\text{Domain } (\text{summedBidVectorRel bids } N \ G)))\) =
sum snd \(((\text{summedBidVectorRel bids } N \ G) \ |\ | a)\)
using sumRestrictedToDomainInvariant lm055 by fast

**corollary** lm058:
assumes \(a \in \text{allAllocations } N \ G\)
shows sum \((\text{summedBidVector bids } N \ G)\) \(a = \text{sum} \ (\text{snd } ((\text{summedBidVectorRel bids } N \ G) \ |\ | a))\)
proof –
let \(?l=\text{summedBidVector}\) let \(?L=\text{summedBidVectorRel}\)
have \(a \subseteq \text{Domain } (?L \text{ bids } N \ G)\) using assms by (rule lm056)
then have \(a = a \cap \text{Domain } (?L \text{ bids } N \ G)\) by blast
then have \(\text{sum } (?l \text{ bids } N \ G) \ a = \text{sum } (?l \text{ bids } N \ G) \ (a \cap \text{Domain } (?L \text{ bids } N \ G))\)
by presburger
thus ?thesis using lm057 by auto
qed

**corollary** lm059:
assumes \(a \in \text{allAllocations } N \ G\)
shows sum \((\text{summedBidVector bids } N \ G)\) \(a = \text{sum} \ (\text{snd } ((\text{summedBid bids}) \cdot ((N \times (\text{Pow } G - \{\{\}\})) \cap a))\)
(is \(?X=?R\))
proof –

103
let \( ?p = \text{summedBid} \)
let \( ?L = \text{summedBidVectorRel} \)
let \( ?l = \text{summedBidVector} \)
let \( ?A = N \times (\text{Pow} G - \{\{\}\}) \)
let \( ?\text{inner2} = (?p \text{ bids})' (\{\}\cap a) \)
let \( ?\text{inner1} = (?l \text{ bids} N G)\|a \)

have \( ?R = \text{sum snd} ?\text{inner1} \) using assms lm020 by (metis (no-types))
moreover have \( \text{sum} \ (?l \text{ bids} N G) \) a = \( \text{sum} \ \text{snd} \ ?\text{inner1} \) using assms by (rule lm058)
ultimately show \( \text{?thesis} \) by simp
qed

corollary lm060:
assumes \( a \in \text{allAllocations} N G \)
shows \( \text{sum} \ (\text{summedBidVector bids} N G) \) a = \( \text{sum} \ \text{snd} \ ((\text{summedBid bids}) \cdot a) \) (is \( ?L=?R \))
proof –
let \( ?p=\text{summedBid} \)
let \( ?l=\text{summedBidVector} \)
have \( ?L = \text{sum snd} ((?p \text{ bids})'(N \times (\text{Pow} G - \{\{\}\})))\cap a)) \) using assms by (rule lm059)
moreover have ... = \( ?R \) using assms allocationPowerset Int-absorb1 by (metis (no-types))
ultimately show \( \text{?thesis} \) by simp
qed

corollary lm061:
\( \text{sum snd} ((\text{summedBid bids}) \cdot a) = \text{sum} \ (\text{snd} \circ (\text{summedBid bids})) \) a
using sum.reindex lm021 by blast

corollary lm062:
assumes \( a \in \text{allAllocations} N G \)
shows \( \text{sum} \ (\text{summedBidVector bids} N G) \) a = \( \text{sum} \ (\text{snd} \circ (\text{summedBid bids})) \) a (is \( ?L=?R \))
proof –
let \( ?p = \text{summedBid} \)
let \( ?l = \text{summedBidVector} \)
have \( ?L = \text{sum snd} ((?p \text{ bids})' a) \) using assms by (rule lm060)
moreover have ... = \( ?R \) using assms lm061 by blast
ultimately show \( \text{?thesis} \) by simp
qed

corollary lm063:
assumes \( a \in \text{allAllocations} N G \)
shows \( \text{sum} \ (\text{summedBidVector bids} N G) \) a = \( \text{sum} \ ((\text{sum bids}) \circ \omega) \) a (is \( ?L=?R \))
proof –
let \( ?\text{inner1} = \text{snd} \circ (\text{summedBid bids}) \) 
let \( ?\text{inner2}=(\text{sum bids}) \circ \omega \)
let \(M = \text{sum} \ ?\text{inner1} \ a\)

have \(\vdash L = M\) using \text{assms} by (rule \text{lm062})

moreover have \(\vdash \ ?\text{inner1} = \ ?\text{inner2}\) using \text{lm023} \text{assms} by \text{fastforce}

ultimately show \(\vdash L = R\) using \text{assms} by \text{metis}

qed

corollary \text{lm064}:
assumes \(a \in \text{allAllocations \ N G}\)
shows \(\text{sum} (\text{summedBidVector \ bids \ N \ G}) \ a = \text{sum} (\text{sum \ bids}) (\omega' a)\)

proof
  have \(\{\} \not\in \text{Range \ a}\) using \text{assms} by (metis \text{emptyNotInRange})
  then have \(\text{inj-on \ omega \ a}\) using \text{lm028} by blast
  then have \(\text{sum} (\text{sum \ bids}) (\omega' a) = \text{sum} ((\text{sum \ bids}) \circ \omega) a\)
    by (rule \text{sum.reindex})
  moreover have \(\text{sum} (\text{summedBidVector \ bids \ N \ G}) \ a = \text{sum} ((\text{sum \ bids}) \circ \omega) a\)
    using \text{assms \ lm063} by (rule \text{Extraction.exE-realizer})
  ultimately show \(?\text{thesis}\) by \text{presburger}

qed

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

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

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

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

proof
  have \(\text{runiq \ a}\) by (metis \text{(no-types) \ assms \ image-iff \ allocationRightUniqueRange-Domain})
  moreover have \(\{\} \not\in \text{Range \ a}\) using \text{assms} by (metis \text{emptyNotInRange})
  ultimately show \(?\text{thesis}\) using \text{lm027} by blast

qed

lemma \text{lm069}:
assumes \(a \in \text{allAllocations \ N \ G \ finite \ G}\)
shows \(\text{sum} (\text{sum \ bids}) (\omega' a) = \text{sum \ bids} (\bigcup (\omega' a))\)
using \text{assms \ sumUnionDisjoint2 \ lm068 \ lm067 \ by \ (metis (lifting, mono-tags))}
corollary lm070:
assumes a ∈ allAllocations N G finite G
shows sum (summedBidVector bids N G) a = sum bids (pseudoAllocation a)
(is ?L = ?R)
proof −
have ?L = sum (sum bids) (omega 'a) using assms lm064 by blast
moreover have ... = sum bids (∪ (omega 'a)) using assms lm069 by blast
ultimately show ?thesis unfolding pseudoAllocation-def by presburger
qed

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

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

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

10.4 From Pseudo-allocations to allocations

abbreviation pseudoAllocationInv pseudo == {⟨x, ∪ (pseudo " {x})⟩ | x. x ∈ Domain pseudo}

lemma lm074:
assumes runiq a {} ∉ Range a
shows a = pseudoAllocationInv (pseudoAllocation a)
proof −
let ?p=⟨{x, Y} | Y x. Y ∈ finestpart (a.,x) & x ∈ Domain a⟩
let ?a=⟨{x, ∪ (?p " {x})} | x. x ∈ Domain ?p⟩
have \( \forall x \in \text{Domain } a. \ a.,x \neq \{\} \) by (metis assms eval-runiq-in-Range)
then have \( \forall x \in \text{Domain } a. \ \text{finestpart } (a.,x) \neq \{\} \) by (metis notEmptyFinestpart)

then have Domain \( a \subseteq \text{Domain } ?p \) by force
moreover have Domain \( a \supseteq \text{Domain } ?p \) by fast
ultimately have 1: Domain \( a = \text{Domain } ?p \) by fast

\{ fix \( z \) assume \( z \in ?a \) then obtain \( x \) where \( x \in \text{Domain } ?p \& \ z=(x, \bigcup (\text{finestpart } (a.,x))) \) by blast
then have \( x \in \text{Domain } a \& \ z=(x, \bigcup (\text{finestpart } (a.,x))) \) by fast
then moreover have \( \text{finestpart } (a.,x) = \text{finestpart } (a.,x) \) using assms by fastforce
moreover have \( \bigcup (\text{finestpart } (a.,x)) = a.,x \) by (metis finestPartUnion)
ultimately have \( z \in a \) by (metis assms(1) eval-runiq-rel)
\}
then have 2: \( ?a \subseteq a \) by fast

\{ fix \( z \) assume \( 0: \ z \in a \ let \ ?x=fst \ z \ let \ ?Y=a.,\ ?x \ let \ ?YY=\text{finestpart } ?Y \) have \( z \in a \& \ ?x \in \text{Domain } a \) using \( 0 \) by (metis fst-eq-Domain rev-image-eqI)
then have 3: \( z \in a \& \ ?x \in \text{Domain } ?p \) using \( 1 \) by presburger
then have \( \text{finestpart } (a.,x) = ?YY \) by fastforce
then have \( \bigcup (\text{finestpart } (a.,x)) = ?Y \) by (metis finestPartUnion)
moreover have \( z = (?x, ?Y) \) using assms by (metis \( 0 \) functionOnFirstEqualsSecond surjective-pairing)
ultimately have \( z \in ?a \) using \( 3 \) by (metis \( 0 \) lifting mono-tags mem-Collect-eq)
\}
then have \( a = ?a \) using \( 2 \) by blast
moreover have \( ?p = \text{pseudoAllocation } a \) using \( \text{lm048 assms by (metis lifting, mono-tags)} \)
ultimately show \( \text{thesis} \) by auto
qed

**corollary\, lm075:**
assumes \( a \in \text{runiqs } \cap \text{Pow } (\text{UNIV } \times (\text{UNIV } - \{\{\}\})) \)
shows \( (\text{pseudoAllocationInv } \circ \text{pseudoAllocation}) \ a = \text{id } a \)
proof –
have \( \text{runiq } a \) using \( \text{runiqs-def assms by fast} \)
moreover have \( \{\} \notin \text{Range } a \) using \( \text{assms by blast} \)
ultimately show \( \text{thesis using } \text{lm074 by fastforce} \)
qed

**lemma\, lm076:**
inj-on \( (\text{pseudoAllocationInv } \circ \text{pseudoAllocation}) \) \( \text{runiqs } \cap \text{Pow } (\text{UNIV } \times (\text{UNIV } - \{\{\}\})) \)
proof –
let \(?ne=\text{Pow}(\text{UNIV} \times (\text{UNIV} - \{\})))\)
let \(?X=\text{runiqs} \cap \?ne\)
let \(?f=\text{pseudoAllocationInv} \circ \text{pseudoAllocation}\)
have \(\forall a1 \in \?X. \forall a2 \in \?X. ?f a1 = ?f a2 \rightarrow \text{id} a1 = \text{id} a2\) using \text{lm075}\ by blast
then have \(\forall a1 \in \?X. \forall a2 \in \?X. ?f a1 = ?f a2 \rightarrow a1 = a2\) by auto
thus \(\text{thesis}\) unfolding \text{inj-on-def}\ by blast
qed

corollary \text{lm077}:
\(\text{inj-on pseudoAllocation} (\text{runiqs} \cap \text{Pow} (\text{UNIV} \times (\text{UNIV} - \{\}))))\)
using \text{lm076}\ \text{inj-on-image1} by blast

lemma \text{lm078}:
\(\text{injectionsUniverse} \subseteq \text{runiqs}\)
using \text{runiqs-def}\ \text{Collect-conj-eq}\ \text{Int-lower1}\ by \text{metis}\n
lemma \text{lm079}:
\(\text{partitionValuedUniverse} \subseteq \text{Pow} (\text{UNIV} \times (\text{UNIV} - \{\})))\)
using \text{is-non-overlapping-def}\ by \text{force}\n
corollary \text{lm080}:
\(\text{allocationsUniverse} \subseteq \text{runiqs} \cap \text{Pow} (\text{UNIV} \times (\text{UNIV} - \{\})))\)
using \text{lm078} \text{lm079}\ by \text{auto}\n
corollary \text{lm081}:
\(\text{inj-on pseudoAllocation} \text{allocationsUniverse}\)
using \text{lm077} \text{lm080}\ \text{subset-inj-on}\ by blast

lemma \text{lm083}:
assumes \(\text{card} \ N > 0\) distinct \(\text{G}\)
shows \(\text{winningAllocationsRel} \ N \ (\text{set} \ \text{G}) \ \text{bids} \subseteq \text{set} \ (\text{allAllocationsAlg} \ N \ \text{G})\)
using \text{assms}\ \text{winningAllocationPossible} \text{allAllocationsBridgingLemma}\ by \text{(metis(no-types))}\n
corollary \text{lm084}:
assumes \(\text{N} \neq \{\}\) finite \(\text{N}\) distinct \(\text{G}\) set \(\text{G} \neq \{\}\)
shows \(\text{winningAllocationsRel} \ N \ (\text{set} \ \text{G}) \ \text{bids} \cap \text{set} \ (\text{allAllocationsAlg} \ N \ \text{G}) \neq \{\}\)
proof –
let \(\?w=\text{winningAllocationsRel}\)
let \(\?a=\text{allAllocationsAlg}\)
let \(?G = set G\)

have \(\text{card } N > 0\) using assms by (metis card-gt-0-iff)
then have \(?w N \subseteq \text{set } (\?a N G)\) using lm083 by (metis assms(3))
then show \(?thesis\) using assms lm011 by (metis List.finite-set le-iff-inf)

qed

lemma lm085:
\[ X = \{x \in X\} - '\{True\}\]
by blast

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

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

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

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

corollary lm090:
assumes \(N \neq \{\}\ \text{finite } N \text{ distinct } G \text{ set } G \neq \{\}\\)
shows \(\text{chosenAllocation } N \ G \ \text{bids random } \in \text{winningAllocationsRel } N \ (\text{set } G)\)

proof –
have \(\bigwedge x_1 \ b-x x. \ \text{set } x_1 = {}\)
\(\forall (\text{randomEl } x_1 \ b-x::(\text{a } \times \text{'b set} ) \text{ set}) \in x\)
\(\forall \sim \text{ set } x_1 \subseteq x \text{ by (metis (no-types) randomElLemma subsetCE)}\)
thus \(\text{winningAllocationRel } N \ (\text{set } G)\)
\(\text{op } \in (\text{randomEl } (\text{takeAll } (\lambda x. \text{winningAllocationRel } N \ (\text{set } G) \text{ op } \in x) \text{ bids})) \text{ (allAllocationsAlg } N \ G)\text{ random}) \text{ bids}\)

109
by (metis lm088 assms(1) assms(2) assms(3) assms(4) takeAllSubset set-empty)
qed

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

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

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

corollary lm094:
assumes pair ∈ N × (Pow G−{{}})
shows summedBidVector bids N G pair = summedBidSecond bids pair
using assms lm093 lm092 by (metis(mono-tags))

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

lemma lm096:
assumes \( pair \in N \times (\text{Pow } G - \{\{\}\}) \)
shows \( \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( pair = \)
\( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( pair \)
using \( \text{assms lm094 lm095 by (metis(no-types))} \)

corollary \( \text{lm097} \): 
assumes \( X \subseteq N \times (\text{Pow } G - \{\{\}\}) \)
shows \( \forall \) \( pair \in X, \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( pair = \)
\( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( pair \)
proof – 
\{ fix \( esk48_0 \) :: 'a \times 'b set 
\{ assume \( esk48_0 \in N \times (\text{Pow } G - \{\{\}\}) \) 
\hence \( \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( esk48_0 = \) \( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( esk48_0 \) using \( \text{lm096 by blast} \) 
\hence \( esk48_0 \notin X \vee \) \( \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( esk48_0 = \) \( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( esk48_0 \) using \( \text{assms by blast} \) 
thus \( \forall \) \( pair \in X, \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( pair = \) \( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( pair \) by blast 
qed 

corollary \( \text{lm098} \): 
assumes \( aa \subseteq N \times (\text{Pow } G - \{\{\}\}) \)
shows \( \text{sum} ((\text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G)) aa = \)
\( (\text{real} \circ (\text{summedBidVector bidding } N G)) aa \)
(is \( \?L = \?R \))
proof – 
have \( \forall \) \( pair \in aa, \text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G \) \( pair = \)
\( (\text{real} \circ (\text{summedBidVector bidding } N G)) \) \( pair \)
using \( \text{assms by (rule lm097)} \)
then have \( \?L = \text{sum} ((\text{real} \circ (\text{summedBidVector bidding } N G)) aa) \) using \( \text{sum.cong} \)
by force 
then show \( \?L = \?R \) by simp 
qed 

corollary \( \text{lm099} \): 
assumes \( aa \in \text{allAllocations } N G \)
shows \( \text{sum} ((\text{summedBidVector } (\text{real} \circ (\text{bids} :: - \Rightarrow \text{nat})) N G)) aa = \)
\( (\text{real} \circ (\text{summedBidVector bidding } N G)) aa \)
using \( \text{assms lm098 allocationPowerset by (metis(lifting,mono-tags))} \)

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

\[ 111 \]
let \( ?l = \text{summedBidVector} \)
let \( ?m = \text{maxbid} \)
let \( ?s = \text{sum} \)
let \( ?p = \text{pseudoAllocation} \)
let \( ?bb = ?m \cap N \cap G \)
let \( ?b = \text{real} \circ (\text{maxbid}) \)

have real \((\text{maxbid})\) − \(\text{sum}(\text{pseudoAllocation} \cap \text{pseudoAllocation})\) by blast

then have 1: \( \text{real}(\text{card}(G)) − \text{sum}(\text{tiebids} \cap N \cap G) \) = \( \text{real}(\text{card}(G)) − \text{sum}(\text{tiebids} \cap N \cap G) \) by simp

moreover have \( \text{real}(\text{card}(G)) \) = \( \text{real}(\text{card}(G)) \) by force

moreover have \( \text{finite}(\text{pseudoAllocation} \cap \text{pseudoAllocation}) \) by fastforce

moreover have \( \text{finite}(\text{pseudoAllocation} \cap \text{pseudoAllocation}) \) by fastforce

ultimately have \( \text{card}(\text{pseudoAllocation} \cap \text{pseudoAllocation}) \) ≤ \( \text{card}(G) \) by Int-lower2 card-mono by fastforce

then have 1: \( x = \text{real}(\text{card}(G)) − \text{card}(\text{pseudoAllocation} \cap \text{pseudoAllocation}) \) by simp

by force

ultimately have

2: \( x ≤ \text{real}(\text{card}(G)) \) by linarith

have 3: \( \text{card}(\text{pseudoAllocation} \cap \text{pseudoAllocation}) \) ≤ \( \text{card}(G) \) by Int-lower2 card-mono by fastforce

then have

112
4: \( x \geq 0 \) using assms lm100 1 by linarith

have \( \text{card (} ?p \text{ aa } \cap (\?p a) \text{)} = \text{card G } \leftrightarrow (\?p \text{ aa } = ?p a) \)
  using 3 lm041 4 assms by (metis (lifting, mono-tags))

moreover have \( ?p \text{ aa } = ?p a \rightarrow a = \text{aa} \) using assms lm082 inj-on-def
  by (metis (lifting, mono-tags))

ultimately have \( \text{card (} ?p \text{ aa } \cap (\?p a) \text{)} = \text{card G } \leftrightarrow (a = \text{aa}) \) by blast

moreover have \( x = \text{real (} \text{card G} \text{)} - \text{card (} ?p \text{ aa } \cap (\?p a) \text{)} \) using assms lm100
  by blast

ultimately have \( 5: x = 0 \leftrightarrow (a = \text{aa}) \) by linarith

then have
  \( a a \neq a \rightarrow \text{sum (tiebids a N G) aa } < \text{real (} \text{sum (tiebids a N G) a} \text{)} \)
  using 1 4 assms by auto

thus \( ?\text{thesis using 2 4 5} \) unfolding of-nat-less-iff by force

qed

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

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

proof –
  let \( ?a=\text{chosenAllocation N G bids random} \)
  let \( ?p=\text{allAllocations} \)
  let \( ?G=\text{set G} \)
  have \( ?a \in \text{winningAllocationsRel N (set G) bids using assms lm090 by blast} \)
  moreover have \( \text{winningAllocationsRel N (set G) bids } \subseteq ?p N ?G \) using assms
    winningAllocationPossible by metis
  ultimately have \( ?a \in ?p N ?G \) using lm090 assms winningAllocationPossible
    set-rev-mp by blast
  then show \( ?\text{thesis using assms lm102 by blast} \)

qed

abbreviation terminatingAuctionRel N G bids random ==
  \( \text{argmax (} \text{sum (resolvingBid N G bids random)} \text{)} \)
  \( \text{(argmax (} \text{sum bids (} \text{allAllocations N (set G)}) \text{))} \)

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

theorem winningAllocationUniqueness:
assumes \( N \neq \{\} \) distinct \( G \) set \( G \neq \{\} \) finite \( N \)

gives \( \text{terminatingAuctionRel} \ N \ G \ (\text{bids} \ \text{random}) = \{\text{chosenAllocation} \ N \ G \ \text{bids random}\} \)

proof –
let \( ?p = \text{allAllocations} \)
let \( ?G = \text{set} \ G \)
let \( ?X = \text{argmax} \ (\text{sum bids}) \ (\ ?p \ N \ ?G) \)
let \( ?a = \text{chosenAllocation} \ N \ G \ \text{bids random} \)
let \( ?b = \text{resolvingBid} \ N \ G \ \text{bids random} \)
let \( ?f = \text{sum} \ ?b \)
let \( ?t = \text{terminatingAuctionRel} \)

have \( \forall \ a a \in \{\text{allAllocations} \ N \ ?G\} - \{?a\}. \ ?f \ a a < ?f \ ?a \)
using \( \text{assms \ lm103 \ by \ blast} \)
then have \( \forall \ a a \in \ ?X - \{?a\}. \ ?f \ a a < ?f \ ?a \) using \( \text{assms \ lm103 \ by \ auto} \)
moreover have \( \text{finite} \ ?N \ \text{using} \ \text{assms} \) by \( \text{simp} \)
then have \( \text{finite} \ ?p \ ?G \) using \( \text{assms \ allAllocationsFinite} \) by \( \text{metis \ List.finite-set} \)
then have \( \text{finite} \ ?X \) using \( \text{assms} \) by \( \text{metis \ finite-subset \ winningAllocationPossible} \)
moreover have \( \ ?a \in \ ?X \) using \( \text{lm90 \ assms \ by \ blast} \)
ultimately have \( \ ?a \in \ ?X \) using \( \text{lm90 \ assms \ by \ blast} \)
by \( \text{force} \)
moreover have \( \text{finite} \ ?X \ & \ ?a \in \ ?X \ & \ (\forall \ a a \in \ ?X - \{?a\}. \ ?f \ a a < \ ?f \ ?a) \)
by \( \text{rule \ argmaxProperty} \)
ultimately have \( \{?a\} = \text{argmax} \ ?f \ ?X \) using \( \text{injectionsFromEmptyIsEmpty} \) by \( \text{presburger} \)
moreover have \( \ldots = \ ?t \ N \ G \ \text{bids random} \) by \( \text{simp} \)
ultimately show \( \ ?t \ N \ G \ \text{bids random} \) by \( \text{simp} \)
qed

The computable variant of Else is defined next as Elsee.

definition \( \text{toFunctionWithFallbackAlg} \ R \ \text{fallback} == \)
\( (% \ x. \ if \ (x \in \text{Domain} \ R) \ then \ (R,x) \ else \text{fallback}) \)
notation \( \text{toFunctionWithFallbackAlg} \ (\text{infix \ Elsee \ 75}) \)

end

11 VCG auction: definitions and theorems

theory \( \text{CombinatorialAuction} \)

imports \( \text{UniformTieBreaking} \)

begin
Definition of a VCG auction scheme, through the pair $(vcga, vcgp)$

**abbreviation** participants $b == \text{Domain (Domain } b)\text{)}$

**abbreviation** goods $== \text{sorted-list-of-set } (\text{Union } (\text{Range } \text{Domain } b))$

**abbreviation** seller $== (0::\text{integer})$

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

**abbreviation** allAllocations’’ $N \Omega == \text{allocationsUniverse } \cap \{ a. \text{Domain } a \subseteq N \& \bigcup \text{Range } a = \Omega \}$

**lemma** allAllocationsEquivalence:

\[ \text{allAllocations } N \Omega = \text{allAllocations’ } N \Omega \& \text{allAllocations } N \Omega = \text{allAllocations’’ } N \Omega \]

**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’’’ $N \Omega == \text{allocationsUniverse } \cap \{ aa. \text{Domain } a\subseteq N- \{\text{seller}\} \& \bigcup \text{Range } aa \subseteq \Omega \}$

**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:
soldAllocations \((N - \{\text{seller}\}) \Omega \subseteq soldAllocations N \Omega\)
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:
assumes \(N_1 \subseteq N_2\)
shows \(\text{allAllocations'' } N_1 \Omega \subseteq \text{allAllocations'' } N_2 \Omega\)
using assms by auto

lemma allocationWithOneParticipant:
assumes \(a \in \text{allAllocations'' } N \Omega\)
shows \(\text{Domain } (a -\times - x) \subseteq N - \{x\}\)
using assms Outside-def by fastforce

lemma soldAllocationIsAllocation:
assumes \(a \in \text{soldAllocations } N \Omega\)
shows \(a \in \text{allocationsUniverse}\)
proof
obtain aa where \(a = aa -\times - \text{seller} \& \ aa \in \text{allAllocations } (N \cup \{\text{seller}\}) \Omega\)
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:
assumes \(a \in \text{soldAllocations } N \Omega\)
shows \(a \in \text{allAllocations'' } (\text{Domain } a) (\bigcup \text{Range } a)\)
proof
show ?thesis using assms soldAllocationIsAllocation
by auto blast+
qed

lemma onlyGoodsAreSold:
assumes \(a \in \text{soldAllocations'' } N \Omega\)
shows \(\bigcup \text{Range } a \subseteq \Omega\)
using assms Outside-def by blast

lemma soldAllocationIsRestricted:
\(a \in \text{soldAllocations'' } N \Omega = (\text{EX aa. aa -\times - \{seller\} = a \& aa \in \text{allAllocations'' } (N \cup \{seller\}) \Omega)\)
by blast

lemma restrictionConservation:
\[(R \,\! \times \, Y) \, \rightarrow \, x = R \,\! \times \, x\]

unfolding Outside-def paste-def by blast

lemma allocatedToBuyerMeansSold:
assumes \(a \in \text{allocationsUniverse} \, \rightarrow \, \text{Domain} \, a \subseteq N - \{\text{seller}\} \cup \text{Range} \, a \subseteq \Omega\)
shows \(a \in \text{soldAllocations}'' \, N \, \Omega\)

proof –
let \(\,?i = \text{seller} \)
let \(?Y = \{\,\Omega - \bigcup \text{Range} \, a\} - \{\{\}\}\)
let \(?b = \{\,?i\}\times ?Y\)
let \(?aa = a \cup ?b\)
let \(?aa' = a \,\! \times \, ?b\)

have
1: \(a \in \text{allocationsUniverse} \) using assms(1) by fast

have \(?b \subseteq \{\,?i,\Omega - \bigcup \text{Range} \, a\} - \{\{\}\}\) by fastforce
then have
2: \(?b \in \text{allocationsUniverse} \)
using allocationUniverseProperty subsetAllocation by (metis(no-types))

have
3: \(\bigcup \text{Range} \, a \cup \bigcup \text{Range} \, ?b = \{\}\) by blast

have
4: \(\text{Domain} \, a \cap \bigcup \text{Range} \, ?b = \{\}\) using assms by fast

have \(?aa \in \text{allocationsUniverse} \) using 1 2 3 4 by (rule allocationUnion)
then have \(?aa \in \text{allAllocations}'' \, (\text{Domain} \, ?aa) \cup \text{Range} \, ?aa\)
unfolding allocationsUniverseCharacterization by metis

then have \(?aa \in \text{allAllocations}'' \, (N \cup \{\,?i\}\) \cup \text{Range} \, ?aa\)
using allocationMonotonicity assms paste-def by auto

moreover have \(\text{Range} \, ?aa = \bigcup \text{Range} \, ?a \cup \bigcup \text{Range} \, ?b\) 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 \(?aa' \in \text{allAllocations}'' \, (N \cup \{\,?i\}\) \cup \text{Range} \, ?aa\) by simp
moreover have \(\text{Domain} \, ?b \subseteq \{\,?i\}\) by fast
have \(?aa'' = ?a = a -\,\! \times \, \bigcup \text{Range} \, ?aa\) by (rule restrictionConservation)
moreover have \(\bigcup \text{Range} \, ?aa = \bigcup \text{Range} \, ?a\) by auto
ultimately show \(?\text{thesis}\) using soldAllocationIsRestricted by auto
qed

lemma allocationCharacterization:
a \in \text{allAllocations} \, N \, \Omega \Rightarrow 
(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 \( \bar{i} = \text{seller} \)

obtain \( aa \) where
\( 0: a = aa -- \bar{i} \& aa \in \text{allAllocations''} \ (N \cup \{ \bar{i} \}) \ \Omega \)
using \( \text{assms(1)} \) \( \text{soldAllocationIsRestricted by blast} \)
then have \( \text{Domain } aa \subseteq N \cup \{ \bar{i} \} \) using \( \text{allocationCharacterization by blast} \)
then have \( \text{Domain } a \subseteq N - \{ \bar{i} \} \) using \( \text{0 Outside-def by blast} \)
moreover have \( a \in \text{soldAllocations } N \ \Omega \) using \( \text{assms soldAllocationsEquivalenceVariant by blast} \)
then moreover have \( a \in \text{allocationsUniverse} \) using \( \text{soldAllocationIsAllocation by blast} \)
ultimately show \( \ ?thesis \) by blast
qed

corollary \( \text{lm02:} \)

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 \( \text{assms lm01[of a] by blast} \)
moreover have \( \text{Domain } a \subseteq N - \{ \text{seller} \} \) using \( \text{assms lm01 by blast} \)
moreover have \( \bigcup \text{Range } a \subseteq \Omega \) using \( \text{onlyGoodsAreSold by blast} \)
ultimately show \( ?thesis \) by blast
qed

corollary \( \text{lm03:} \)

\( (a \in \text{soldAllocations'' } N \ \Omega) = \)
\( (a \in \text{allocationsUniverse} \& a \in \{ aa. \text{Domain } aa \subseteq N - \{ \text{seller} \} \& \bigcup \text{Range } aa \subseteq \Omega \}) \)
(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 \( \text{lm02 allocatedToBuyerMeansSold by (metis (mono-tags))} \)
then have \( ?L = (a \in \text{allocationsUniverse} \& \text{Domain } a \subseteq N - \{ \text{seller} \} \& \bigcup \text{Range } a \subseteq \Omega) \) by fast
moreover have \( ... = ?R \) using \( \text{mem-Collect-eq by (metis (lifting, no-types))} \)
ultimately show \( ?thesis \) by auto
qed

corollary \( \text{lm04:} \)

\( a \in \text{soldAllocations'' } N \ \Omega = \)
\( (a \in (\text{allocationsUniverse} \cap \{ aa. \text{Domain } aa \subseteq N - \{ \text{seller} \} \& \bigcup \text{Range } aa \subseteq \Omega \})) \)
using \( \text{lm03 by (metis (mono-tags) Int-iff)} \)

corollary \( \text{soldAllocationVariantEquivalence:} \)

\( \text{soldAllocations'' } N \ \Omega = \text{soldAllocations''' } N \ \Omega \)
(is \( ?L = ?R \))

proof –

118
\{ 
\begin{align*}
&\text{fix } a \\
&\text{have } a \in ?L = (a \in ?R) \text{ by } (\text{rule lm04}) \\
&\end{align*}
\}

thus \(?thesis\) by blast

\begin{proof}
\begin{align*}
\text{let } \ ?bb & = \text{seller} \\
\text{let } \ ?d & = \text{Domain} \\
\text{let } \ ?r & = \text{Range} \\
\text{let } \ ?X1 & = \{aa. \ ?d aa \subseteq N - \{\ ?bb\} \& \bigcup \ ?r aa \subseteq \Omega\} \\
\text{let } \ ?X2 & = \{aa. \ ?d aa \subseteq N - \{\ ?bb\} \& \bigcup \ ?r aa \subseteq \Omega\} \\
\text{have } a \in \ ?X2 \text{ using } \text{assms}(1) \text{ by fast} \\
\text{then have } 0: \ ?d a \subseteq N - \{\ ?bb\} \& \bigcup \ ?r a \subseteq \Omega \text{ by blast} \\
\text{then have } \ ?d (a -\rightarrow n) \subseteq N - \{\ ?bb\} - \{n\} \\
\text{using } \text{outside-reduces-domain} \text{ by } (\text{metis } \text{Diff-monotonic} \ \text{subset-refl}) \\
\text{moreover have } ... = N - \{n\} - \{\ ?bb\} \text{ by fastforce} \\
\text{ultimately have } \ ?d (a -\rightarrow n) \subseteq N - \{n\} - \{\ ?bb\} \text{ by blast} \\
\text{moreover have } \bigcup \ ?r (a -\rightarrow n) \subseteq \Omega \\
\text{unfolding } \text{Outside-def} \text{ using } \emptyset \text{ by blast} \\
\text{ultimately have } a -\rightarrow n \in ?X1 \text{ by fast} \\
\text{moreover have } a -\rightarrow n \in \text{allocationsUniverse} \\
\text{using } \text{assms}(1) \ \text{Int-iff allocationsUniverseOutside by (metis lifting, mono-tags))} \\
\end{align*}
\end{proof}

ultimately show \(?thesis\) by blast

\begin{proof}
\begin{align*}
\text{lemma } lm05: \\
\text{assumes } a \in \text{soldAllocations'''} N \ \Omega \\
\text{shows } a -\rightarrow n \in \text{soldAllocations'''} (N - \{n\}) \ \Omega \\
\text{proof --} \\
\text{let } \ ?bb & = \text{seller} \\
\text{let } \ ?d & = \text{Domain} \\
\text{let } \ ?r & = \text{Range} \\
\text{let } \ ?X1 & = \{aa. \ ?d aa \subseteq N - \{\ ?bb\} \& \bigcup \ ?r aa \subseteq \Omega\} \\
\text{let } \ ?X2 & = \{aa. \ ?d aa \subseteq N - \{\ ?bb\} \& \bigcup \ ?r aa \subseteq \Omega\} \\
\text{have } a \in \ ?X2 \text{ using } \text{assms}(1) \text{ by fast} \\
\text{then have } 0: \ ?d a \subseteq N - \{\ ?bb\} \& \bigcup \ ?r a \subseteq \Omega \text{ by blast} \\
\text{then have } \ ?d (a -\rightarrow n) \subseteq N - \{n\} - \{\ ?bb\} - \{n\} \\
\text{using } \text{outside-reduces-domain} \text{ by } (\text{metis } \text{Diff-monotonic} \ \text{subset-refl}) \\
\text{moreover have } ... = N - \{n\} - \{\ ?bb\} \text{ by fastforce} \\
\text{ultimately have } \ ?d (a -\rightarrow n) \subseteq N - \{n\} - \{\ ?bb\} \text{ by blast} \\
\text{moreover have } \bigcup \ ?r (a -\rightarrow n) \subseteq \Omega \\
\text{unfolding } \text{Outside-def} \text{ using } \emptyset \text{ by blast} \\
\text{ultimately have } a -\rightarrow n \in ?X1 \text{ by fast} \\
\text{moreover have } a -\rightarrow n \in \text{allocationsUniverse} \\
\text{using } \text{assms}(1) \ \text{Int-iff allocationsUniverseOutside by (metis lifting, mono-tags))} \\
\end{align*}
\end{proof}

ultimately show \(?thesis\) by blast

\end{proof}

\begin{proof}
\begin{align*}
\text{lemma } allAllocationsEquivalenceExtended: \\
\text{soldAllocations} = \text{soldAllocations'} & \& \\
\text{soldAllocations'} = \text{soldAllocations'''} & \& \\
\text{soldAllocations'''} = \text{soldAllocations''} \\
\text{using } \text{soldAllocationVariantEquivalence soldAllocationsEquivalenceVariant by metis} \\
\end{align*}
\end{proof}

\begin{proof}
\begin{align*}
\text{corollary } soldAllocationRestriction: \\
\text{assumes } a \in \text{soldAllocations} N \ \Omega \\
\text{shows } a -\rightarrow n \in \text{soldAllocations} (N - \{n\}) \ \Omega \\
\text{proof --} \\
\text{let } \ ?A' & = \text{soldAllocations'''} \\
\text{have } a \in ?A' N \ \Omega \text{ using } \text{assms } \text{allAllocationsEquivalenceExtended} \text{ by metis} \\
\text{then have } a -\rightarrow n \in ?A' (N - \{n\}) \ \Omega \text{ by } (\text{rule lm05}) \\
\text{thus } \ ?thesis \text{ using } \text{allAllocationsEquivalenceExtended by metis} \\
\end{align*}
\end{proof}
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)
  ultimately show \( \ldots = \text{soldAllocations}'''' N \Omega_2 \)
    using soldAllocationVariantEquivalence
    by metis
  ultimately show \( \text{thesis} \) by auto
qed

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

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

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

abbreviation vega \( N \Omega b r \) ==
  the-elem \( (\text{vegas} N \Omega b r) \)

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

lemma lm06:
  assumes \( \text{card} ((\text{argmax} \circ \text{sum}) \ (\text{randomBids} N \Omega b r)
    (\text{argmax} \circ \text{sum}) b (\text{allAllocations} (N \cup \{\text{seller}\}) (\text{set} \Omega)))) \)
  \( = 1 \)
  shows \( \text{vega} N \Omega b r = \)

120
\[
(\text{the-elem} \ ((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ b \ r)) \\
((\text{argmax} \circ \text{sum}) \ b \ (\text{allAllocations} \ (\text{set} \ {\text{seller}} \cup N) \ (\text{set} \ \Omega))))
\]

— seller
using assms cardOneTheElem by auto

corollary lm07:
assumes card \((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ b \ r)\) \\
\((\text{argmax} \circ \text{sum}) \ b \ (\text{allAllocations} \ (N \cup \{\text{seller}\}) \ (\text{set} \ \Omega)))\)
= 1
shows \(\text{vca} \ N \ \Omega \ b \ r = \text{vca}' \ N \ \Omega \ b \ r\)
(is \(?l = ?r\))
proof –
have \(?l = (\text{the-elem} \ ((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ b \ r)) \\
((\text{argmax} \circ \text{sum}) \ b \ (\text{allAllocations} \ (\text{set} \ \Omega))))\)
— seller
using assms by (rule lm06)
moreover have ...
ultimately show ?thesis by blast
qed

lemma lm08:
assumes distinct \(\Omega \ \text{set} \ \Omega \neq \{\} \ \text{finite} \ N\)
shows card \((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ \text{bids random})\) \\
\((\text{argmax} \circ \text{sum}) \ \text{bids} \ (\text{allAllocations} \ (N \cup \{\text{seller}\}) \ (\text{set} \ \Omega)))\)
= 1
(is card \(?l = -\))
proof –
let \(?N = N \cup \{\text{seller}\}\)
let \(?b' = \text{randomBids} \ N \ \Omega \ \text{bids random}\)
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 \(?a (\?s \ ?b') (\?a (\?s \text{bids}) \ (\text{allAllocations} \ ?N \ (\text{set} \ \Omega))) = \text{chosenAllocation} \ ?N \ \Omega \ \text{bids random}\) (is \(?L = ?R\))
using 1 assms(1) assms(2) 2 by (rule winningAllocationUniqueness)
moreover have \(?L = ?f \ ?b' (\?f \text{bids} \ (\text{allAllocations} \ ?N \ (\text{set} \ \Omega)))\) by auto
ultimately have \(?l = \{\text{chosenAllocation} \ ?N \ \Omega \ \text{bids random}\}\) by simp
moreover have card ...
ultimately show ?thesis by simp
qed

lemma vcaEquivalence:
assumes distinct \(\Omega \ \text{set} \ \Omega \neq \{\} \ \text{finite} \ N\)
shows \(\text{vca} \ N \ \Omega \ b \ r = \text{vca}' \ N \ \Omega \ b \ r\)
using assms lm07 lm08 by blast

**Theorem vegaDefiniteness:**

**Assumes** distinct $\Omega$ set $\Omega \neq \{}$ finite $N$
**Shows** card (vegas $N$ $\Omega$ $b$ $r$) = 1

**Proof** –

have card ((argmax$\circ$sum) (randomBids $N$ $\Omega$ $b$ $r$)

((argmax$\circ$sum) $b$ (allAllocations ($N \cup \{\text{seller}\}$) (set $\Omega$)))) = 1

(is card ?X = -) using assms lm08 by blast

moreover have ($\text{Outside}\{\text{seller}\}$) ?X = vegas $N$ $\Omega$ $b$ $r$ by blast

ultimately show ?thesis using cardOneImageCardOne by blast

qed

**Lemma vegaDefinitenessVariant:**

**Assumes** distinct $\Omega$ set $\Omega \neq \{}$ finite $N$
**Shows** card (argmax (sum (randomBids $N$ $\Omega$ $b$ $r$))

(maximalStrictAllocations $N$ (set $\Omega$) $b$)) = 1

(is card ?L=-)

**Proof** –

let ?n = \{ seller \}

have 1: (?n $\cup$ $N$)$\neq\{}$ by simp

have 2: finite (?n$\cup$N) using assms(3) by fast

have terminatingAuctionRel (?n\cup N) $\Omega$ $b$ $r$ = \{ chosenAllocation (?n$\cup$N) $\Omega$ $b$ $r$ \}

using 1 assms(1) assms(2) 2 by (rule winningAllocationUniqueness)

moreover have ?L = terminatingAuctionRel (?n\cup N) $\Omega$ $b$ $r$ by auto

ultimately show ?thesis by auto

qed

**Theorem winningAllocationIsMaximal:**

**Assumes** distinct $\Omega$ set $\Omega \neq \{}$ finite $N$
**Shows** the-elem (argmax (sum (randomBids $N$ $\Omega$ $b$ $r$))

(maximalStrictAllocations $N$ (set $\Omega$) $b$)) $\in$

(maximalStrictAllocations $N$ (set $\Omega$) $b$)

(is the-elem ?X $\in$ ?R)

**Proof** –

have card ?X=1 using assms by (rule vegaDefinitenessVariant)

moreover have ?X $\subseteq$ ?R by auto

ultimately show ?thesis using cardinalityOneTheElem by blast

qed

**Corollary winningAllocationIsMaximalWithoutSeller:**

**Assumes** distinct $\Omega$ set $\Omega \neq \{}$ finite $N$
shows $vega' N \Omega b r \in (Outside' \{seller\}) \cdot (\text{maximalStrictAllocations } N (set \ \Omega)) b$
  using assms winningAllocationIsMaximal by blast

lemma maximalAllactionWithoutSeller:
  $(Outside' \{seller\}) \cdot (\text{maximalStrictAllocations } N \Omega b) \subseteq \text{soldAllocations } N \Omega$
  using Outside-def by force

lemma onlyGoodsAreAllocatedAuxiliary:
  assumes distinct $\Omega$ set $\Omega \neq \{\}$ finite N
  shows $vega' N \Omega b r \in \text{soldAllocations } N (set \ \Omega)$
  (is $?a \in ?A$)
  proof –
  have $?a \in (Outside' \{seller\}) \cdot (\text{maximalStrictAllocations } N (set \ \Omega)) b$
    using assms by (rule winningAllocationIsMaximalWithoutSeller)
  thus $?\text{thesis using maximalAllactionWithoutSeller by fastforce}$
qed

theorem onlyGoodsAreAllocated:
  assumes distinct $\Omega$ set $\Omega \neq \{\}$ finite N
  shows $vega N \Omega b r \in \text{soldAllocations } N (set \ \Omega)$
  (is $-r$)
  proof –
  have $vega' N \Omega b r \in ?r$ using assms by (rule onlyGoodsAreAllocatedAuxiliary)
  then show $?\text{thesis using assms vegaEquivalence by blast}$
qed

corollary neutralSeller:
  assumes $\forall X. X \in \text{Range } a \rightarrow b (\text{seller}, X)=0$ finite a
  shows $\{\text{sum } b a | a. a \in A\} = \{\text{sum } b (a -- seller) | a. a \in A\}$
  proof –
  let $?n = \text{seller}$
  have $\text{finite } (a |\{|?n\}|)$ using assms restrict-def by (metis finite-Int)
  moreover have $\forall z \in a |\{|?n\}|. b z=0$ using assms restrict-def by fastforce
  ultimately have $\text{sum } b (a |\{|?n\}|) = 0$ using assms by (metis sum.neutral)
  thus $?\text{thesis using sumOutside assms(2) by (metis add.comm-neutral)}$
qed

corollary neutralSellerVariant:
  assumes $\forall a \in A. \text{finite } a \& (\forall X. X \in \text{Range } a \rightarrow b (\text{seller}, X)=0)$
  shows $\{\text{sum } b a | a. a \in A\} = \{\text{sum } b (a -- seller) | a. a \in A\}$
  using assms neutralSeller by (metis (lifting, no-types))

lemma vegaIsMaximalAux1:
  assumes distinct $\Omega$ set $\Omega \neq \{\}$ finite N
  shows $\exists a. ((a \in (\text{maximalStrictAllocations } N (set \ \Omega)) b) \& (vega' N \Omega b r = a -- seller)) \&
\( a \in \text{argmax} \ (\text{sum} \ b) \ (\text{allAllocations} \ (\{\text{seller}\} \cup N) \ (\text{set} \ \Omega)) \)

using assms winningAllocationIsMaximalWithoutSeller by fast

lemma \text{vegaIsMaximalAux2}:
  assumes distinct \( \Omega \) set \( \Omega \neq \{\} \) finite \( N \)
  \( \forall a \in \text{allAllocations} \ (\{\text{seller}\} \cup N) \ (\text{set} \ \Omega) \). \( \forall X \in \text{Range} \ a. \ b \ (\text{seller}, \ X)=0 \)
  (is \ \forall a\in\Omega \ x. \ -)
  shows \( \text{sum} \ b \ (\text{vega}^\prime \ N \ \Omega \ b \ r) = \text{Max} \{\text{sum} \ b \ a\} \ a. \ a \in \text{soldAllocations} \ N \ (\text{set} \ \Omega) \)

proof –
  let \( ?n = \text{seller} \)
  let \( ?s = \text{sum} \)
  let \( ?a = \text{vega}^\prime \ N \ \Omega \ b \ r \)
  obtain \( a \) where
  \( \theta: a \in \text{maximalStrictAllocations} \ N \ (\text{set} \ \Omega) \ b \ & \)
  \( ?a = a-\ -?n \ & \)
  \( (a \in \text{argmax} \ (\text{sum} \ b) \ (\text{allAllocations} \ (\{\text{seller}\} \cup N) \ (\text{set} \ \Omega))) \)
  (is - & \ ?a=-& \ a\in\Omega) 
  using assms(1,2,3) \text{vegaIsMaximalAux1} by blast
  have \( 1: \forall a \in \ ?X. \ \text{finite} \ a \ & \ (\forall X. \ X\in\text{Range} \ a \ \longrightarrow \ b \ (\ ?n, \ X)=0) \)
  using assms(4) List.finite-set allocationFinite by metis
  then have \( a \in \ ?X \) using 0 by auto have \( a \in \ ?Z \) using 0 by fast
  then have \( a \in \ ?X\cap\{x. \ ?s \ b \ x = \text{Max} \ (\ ?s \ b \ ?X)\} \) using injectionsUnionCommutate by simp
  then have \( a \in \{x. \ ?s \ b \ x = \text{Max} \ (\ ?s \ b \ ?X)\} \) using injectionsUnionCommutate by simp
  moreover have \( ?s \ b \ ?X = \{?s \ b \ a\} \ a. \ a\in\ ?X\} \) by blast
  ultimately have \( ?s \ b \ a = \text{Max} \ \{?s \ b \ a\} \ a. \ a\in\ ?X\} \) by auto
  moreover have \( \{?s \ b \ a\} \ a. \ a\in\ ?X\} = \{?s \ b \ (a-\ -?n)\} \ a. \ a\in\ ?X\} \)
  using 1 by (rule neutralSellerVariant)
  moreover have \( \ldots = \{?s \ b \ a\} \ a. \ a\in\ \text{Outside}' \{?n\} \ ?X\} \) by blast
  moreover have \( \ldots = \{?s \ b \ a\} \ a. \ a\in\ \text{soldAllocations} \ N \ (\text{set} \ \Omega)\} \) by simp
  ultimately have \( \text{Max} \ \{?s \ b \ a\} \ a. \ a\in\ \text{soldAllocations} \ N \ (\text{set} \ \Omega)\} = ?s \ b \ a \) by simp
  moreover have \( \ldots = ?s \ b \ (a-\ -?n) \) using 1 2 neutralSeller by (metis (lifting, no-types))
  ultimately show \( ?s \ b \ ?a=\text{Max} \{?s \ b \ a\} \ a. \ a\in\ \text{soldAllocations} \ N \ (\text{set} \ \Omega)\} \) 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 \text{vegaIsMaximal}:
  assumes distinct \( \Omega \) set \( \Omega \neq \{\} \) finite \( N \ \forall X. \ b \ (\text{seller}, \ X)=0 \)
  shows \( \text{sum} \ b \ (\text{vega}^\prime \ N \ \Omega \ b \ r) = \text{Max} \{\text{sum} \ b \ a\} \ a. \ a \in \text{soldAllocations} \ N \ (\text{set} \ \Omega) \)
  using assms vegaIsMaximalAux2 by blast
corollary vcgaIsAllocationAllocatingGoodsOnly:
  assumes distinct Ω set Ω ≠ {} finite N
  shows vega’ N Ω b r ∈ allocationsUniverse & ∪ Range (vega’ N Ω b r) ⊆ set Ω
proof –
  let ?a = vega’ N Ω b r
  let ?n = seller
  obtain a where
  θ: ?a = a ∸ seller & a ∈ maximalStrictAllocations N (set Ω) b
    using assms winningAllocationIsMaximalWithoutSeller by blast
then moreover have
  I: a ∈ allAllocations ({{?n}∪N} (set Ω)) by auto
moreover have maximalStrictAllocations N (set Ω) b ⊆ allocationsUniverse
  by (metis (lifting, mono-tags) winningAllocationPossible
       allAllocationsUniverse subset-trans)
ultimately moreover have ?a = a ∸ seller & a ∈ allocationsUniverse by blast
then have ?a ∈ allocationsUniverse using allocationsUniverseOutside by auto
moreover have ∪ Range a = set Ω using allAllocationsIntersectionSetEquals 1
by metis
then moreover have ∪ Range ?a ⊆ set Ω using Outside-def 0 by fast
ultimately show ?thesis using allocationsUniverseOutside Outside-def by blast
qed

abbreviation vcgp N Ω b r n ==
  Max (sum b · (soldAllocations (N−{n}) (set Ω)))
  − (sum b (vega N Ω b r ∸ n))

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

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

The price paid by any participant is non-negative.

theorem NonnegPrices:
  assumes distinct Ω set Ω ≠ {} finite N
  shows vcgp N Ω b r n >= (0::price)
proof –
  let ?a = vega N Ω b r

125
let \( \mathcal{A} = \text{soldAllocations} \)
let \( f = \text{sum } b \)

have \( a \in \mathcal{A} \land n \in \mathcal{A} \setminus \{n\} \) using assms by (rule onlyGoodsAreAllocated)
then have \( a \setminus n \in \mathcal{A} \setminus \{n\} \) by (rule soldAllocationRestriction)

moreover have finite \( (\mathcal{A} \setminus \{n\}) \) using assms by (rule finite-set)

ultimately have \( \text{Max}(f' (\mathcal{A} \setminus \{n\})) \geq f(a \setminus n) \) by (rule maxLemma)
then show \( L - R > 0 \) by linarith

qed

lemma allocationDisjointAuxiliary:
assumes \( a \in \text{allocationsUniverse} \land \text{Domain } a \land n1 \in \text{Domain } a \land n2 \in \text{Domain } a \land n1 \neq n2 \)
shows \( a \setminus n1 \cap a \setminus n2 = \{\} \)

proof -

have \( \text{Range } a \in \text{partitionsUniverse} \) using assms nonOverlapping by blast
moreover have \( a \in \text{injectionsUniverse} \land a \in \text{partitionValuedUniverse} \)

ultimately moreover have \( a \setminus n1 \in \text{Range } a \)

ultimately moreover have \( a \setminus n1 \neq a \setminus n2 \)

ultimately show \( \text{thesis} \)

using is-non-overlapping-def

by (metis (lifting, no-types) eval-runiq-in-Range mem-Collect-eq)

qed

lemma allocationDisjoint:
assumes \( a \in \text{allocationsUniverse} \land \text{Domain } a \land n1 \in \text{Domain } a \land n2 \in \text{Domain } a \land n1 \neq n2 \)
shows \( a \setminus n1 \cap a \setminus n2 = \{\} \)

using assms allocationDisjointAuxiliary imageEquivalence by fastforce

No good is assigned twice.

theorem PairwiseDisjointAllocations:
assumes \( \text{distinct } \Omega \land \text{set } \Omega \neq \{\} \land \text{finite } N \land n1 \neq n2 \)
shows \( (\text{vrga } N \setminus \Omega \setminus b \setminus r) 
\cap (\text{vrga } N \setminus \Omega \setminus b \setminus r) , , n2 = \{\} \)

proof -

have \( \text{vrga } N \setminus \Omega \setminus b \setminus r \in \text{allocationsUniverse} \)
using vrgaIsAllocationAllocatingGoodsOnly assms by blast
then show \( \text{thesis} \) using allocationDisjoint assms by fast

qed

Nothing outside the set of goods is allocated.

theorem OnlyGoodsAllocated:
assumes \( \text{distinct } \Omega \land \text{set } \Omega \neq \{\} \land \text{finite } N \land g \in (\text{vrga } N \setminus \Omega \setminus b \setminus r) , , n \)
shows \( g \in \text{set } \Omega \)
proof

let \( a = \text{vega}' N \Omega b \)

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 have \( a . n \in \text{Range } a \) using \( \text{eval-runiq-in-Range} \) by fastforce

then have \( g \in \bigcup \text{Range } a \subseteq \Omega \) 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 \( \text{maximalStrictAllocationsAlg } N \Omega b == \) \( \text{argmax } (\text{sum } b) (\text{set } (\text{allAllocationsAlg } (\{\text{seller}\} \cup N) \Omega)) \)

definition \( \text{chosenAllocationAlg } N \Omega b \text{ (r::integer) == } \) \( \text{randomEl } (\text{takeAll } (\% x. x \in (\text{argmax } \circ \text{sum } b) (\text{set } (\text{allAllocationsAlg } N \Omega)))) \)

r

definition \( \text{maxbidAlg } a N \Omega == (\text{bidMaximizedBy } a N \Omega) \) Else \( 0 \)

definition \( \text{summedBidVectorAlg } \text{bids } N \Omega == (\text{summedBidVectorRel } \text{bids } N \Omega) \) Else \( 0 \)

definition \( \text{tiebidsAlg } a N \Omega == \text{summedBidVectorAlg } (\text{maxbidAlg } a N \Omega) N \Omega \)

definition \( \text{resolvingBidAlg } N \Omega \text{ bids random == } \) \( \text{tiebidsAlg } (\text{chosenAllocationAlg } N \Omega \text{ bids random}) N \text{ (set } \Omega) \)

definition \( \text{randomBidsAlg } N \Omega b \text{ random == } \text{resolvingBidAlg } (N \cup \{\text{seller}\}) \Omega b \) \( \text{random} \)

definition \( \text{vegaAlgWithoutLosers } N \Omega b \) \( r == \)

(\( \text{the-elem } (\text{argmax } (\text{sum } (\text{randomBidsAlg } N \Omega b r))) \))

(\( \text{maximalStrictAllocationsAlg } N \Omega b \)))

−− seller

127
abbreviation addLosers participantset allocation == (participantset × {{}}) +∗ allocation

definition vegaAlg N Ω b r = addLosers N (vegaAlgWithoutLosers N Ω b r)

abbreviation soldAllocationsAlg N Ω == (Outside’ {seller}) · set (allAllocationsAlg (N ∪ {seller}) Ω)

definition vcpAlg N Ω b r n (winningAllocation::allocation) = Max (sum b · (soldAllocationsAlg (N−{n}) Ω)) − (sum b (winningAllocation −− n))

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

lemma lm09:
assumes fst pair ∈ N snd pair ∈ Pow Ω − {{}}
shows sum (%g. (toFunction (bidMaximizedBy a N Ω)) (fst pair, g))
(finestpart (snd pair)) =
sum (%g. ((bidMaximizedBy a N Ω) Else 0) (fst pair, g))
(finestpart (snd pair))
proof −
let ?f1 = %g.(toFunction (bidMaximizedBy a N Ω))(fst pair, g)
let ?f2 = %g.((bidMaximizedBy a N Ω) Else 0)(fst pair, g)
{
  fix g assume g ∈ finestpart (snd pair)
  then have
    0: g ∈ finestpart Ω using assms finestpartSubset by (metis Diff-iff Pow-iff in-mono)
  have ?f1 g = ?f2 g
  proof −
  have [x1 x2. (x1, g) ∈ x2 × finestpart Ω ∨ x1 ∉ x2 by (metis 0 mem-Sigma-iff)
    then have (pseudoAllocation a <| (N × finestpart Ω)) (fst pair, g) =
      maxbidAlg a N Ω (fst pair, g)
    unfolding toFunctionWithFallbackAlg-def maxbidAlg-def
    by (metis (no-types) domainCharacteristicFunction UnCI assms(1) toFunction-def)
    thus ?thesis unfolding maxbidAlg-def by blast
  qed
  } thus ?thesis using sum.cong by simp
qed
corollary $\text{lm10}$:
  assumes $\text{pair} \in N \times (\text{Pow } \Omega - \{\{\}\})$
  shows $\text{summedBid} \left(\text{toFunction} \left(\text{bidMaximizedBy a N } \Omega\right)\right) \text{pair} = \text{summedBid} \left(\left(\text{bidMaximizedBy a N } \Omega\right) \text{Else } 0\right) \text{pair}$
proof
  have $\text{fst } \text{pair} \in N$ using assms by force
  moreover have $\text{snd } \text{pair} \in \text{Pow } \Omega - \{\{\}\}$ using assms(1) by force
  ultimately show $?\text{thesis using } \text{lm09 by blast}$
qed

corollary $\text{lm11}$:
  $\forall \text{pair} \in N \times (\text{Pow } \Omega - \{\{\}\})$.
  $\text{summedBid} \left(\text{toFunction} \left(\text{bidMaximizedBy a N } \Omega\right)\right) \text{pair} = \text{summedBid} \left(\left(\text{bidMaximizedBy a N } \Omega\right) \text{Else } 0\right) \text{pair}$
  using $\text{lm10 by blast}$

corollary $\text{lm12}$:
  $(\text{summedBid} \left(\text{toFunction} \left(\text{bidMaximizedBy a N } \Omega\right)\right)) \cdot (N \times (\text{Pow } \Omega - \{\{\}\})) = (\text{summedBid} \left(\left(\text{bidMaximizedBy a N } \Omega\right) \text{Else } 0\right)) \cdot (N \times (\text{Pow } \Omega - \{\{\}\}))$
  (is $?f1 \cdot ?Z = ?f2 \cdot ?Z$)
proof
  have $\forall \ z \in ?Z. \ ?f1 \ z = ?f2 \ z$ by (rule $\text{lm11}$)
  thus $?\text{thesis by (rule functionEquivalenceOnSets}$
qed

corollary $\text{lm13}$:
  $\text{summedBidVectorRel} \left(\text{toFunction} \left(\text{bidMaximizedBy a N } \Omega\right)\right) \ N \ \Omega = \text{summedBidVectorRel} \left(\left(\text{bidMaximizedBy a N } \Omega\right) \text{Else } 0\right) \ N \ \Omega$
  using $\text{lm12 by metis}$

corollary $\text{maxbidEquivalence}$:
  $\text{summedBidVectorRel} \left(\text{maxbid a N } \Omega\right) \ N \ \Omega = \text{summedBidVectorRel} \left(\text{maxbidAlg a N } \Omega\right) \ N \ \Omega$
  unfolding $\text{maxbidAlg-def using } \text{lm13 by metis}$

lemma $\text{summedBidVectorEquivalence}$:
  assumes $\text{x} \in (N \times (\text{Pow } \Omega - \{\{\}\}))$
  shows $\text{summedBidVector} \left(\text{maxbid a N } \Omega\right) \ N \ \Omega \ \text{x} = \text{summedBidVectorAlg} \left(\text{maxbidAlg a N } \Omega\right) \ N \ \Omega \ \text{x}$
  (is $?f1 \ ?g1 \ N \ \Omega \ \text{x} = ?f2 \ ?g2 \ N \ \Omega \ \text{x}$)
proof
  let $?h1 = \text{maxbid a N } \Omega$
  let $?h2 = \text{maxbidAlg a N } \Omega$
  have $\text{summedBidVectorRel } ?h1 \ N \ \Omega = \text{summedBidVectorRel } ?h2 \ N \ \Omega$
  using $\text{maxbidEquivalence by metis}$
  moreover have $\text{summedBidVectorAlg } ?h2 \ N \ \Omega = (\text{summedBidVectorRel } ?h2 \ N \ \Omega) \ \text{Else } 0$

129
unfolding summedBidVectorAlg-def by fast
ultimately have summedBidVectorAlg \h1 N \Omega=\summedBidVectorRel \h1 N \\
\Omega
Elsee 0 by simp
moreover have \ldots \ x = (toFunction (\summedBidVectorRel \h1 N \Omega)) \ x \\
using assms functionCompletion summedBidVectorCharacterization by (metis \\
(mono-tags))
ultimately have summedBidVectorAlg \h2 N \Omega = (toFunction (\summedBidVectorRel \\
\h1 N \Omega)) \ x
by (metis (lifting, no-types))
thus \thesis by simp
qed

corollary chosenAllocationEquivalence:
assumes card N > 0 and distinct \Omega
shows chosenAllocation N \Omega b r = chosenAllocationAlg N \Omega b r \\
using assms allAllocationsBridgingLemma by (metis (no-types) chosenAllocationAlg-def comp-apply)
corollary tiebidsBridgingLemma:
assumes x \in (N \times (\pow \Omega - \{\}))) \\
shows tiebids a N \Omega x = tiebidsAlg a N \Omega x \\
(is \ ?L=-)
proof --
have \ ?L = \summedBidVector (\maxbid a N \Omega) N \Omega x by fast
moreover have \ldots = \summedBidVectorAlg (\maxbidAlg a N \Omega) N \Omega x \\
using assms by (rule summedBidVectorEquivalence)
ultimately show \thesis unfolding tiebidsAlg-def by fast
qed
definition tiebids'=tiebids
corollary tiebidsBridgingLemma':
assumes x \in (N \times (\pow \Omega - \{\}))) \\
shows tiebids' a N \Omega x = tiebidsAlg a N \Omega 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 \in (N \times (\pow (set \Omega) - \{\}))) \ card N > 0 distinct \Omega \\
shows resolvingBid' N \Omega b r x = resolvingBidAlg N \Omega b r x \\
using assms chosenAllocationEquivalence tiebidsBridgingLemma' resolvingBidAlg-def \\
by metis

lemma sumResolvingBidEquivalence:
assumes card N > 0 distinct \Omega a \subseteq (N \times (\pow (set \Omega) - \{\}))) \\
shows sum (resolvingBid' N \Omega b r) a = sum (resolvingBidAlg N \Omega b r) a \\
(is \ ?L=?R)
proof –
  have ∀x∈a. resolvingBid’ N Ω b r x = resolvingBidAlg N Ω b r x
  using assms resolvingBidEquivalence by blast
  thus ?thesis using sum.cong by force
qed

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

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

corollary resolvingBidBridgingLemmaVariant1:
  assumes card N > 0 distinct Ω a ∈ allAllocations N (set Ω)
  shows sum (resolvingBid N Ω b r) a = sum (resolvingBidAlg N Ω b r) a
  by blast
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
  by simpforce
proof –
  have card (N∪{seller}) > 0 using assms(1) sup-eq-bot-iff insert-not-empty
    by (metis card-qt-0-iff finite.emptyI finite.insertI finite-UnI)
  moreover have distinct Ω using assms(2) by simp
  moreover have a ∈ allAllocations (N∪{seller}) (set Ω) using assms(3) by fastforce
  ultimately show ?thesis unfolding randomBidsAlg-def by (rule resolvingBid-
    BridgingLemmaVariant1)
qed

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 a ⊆ N × (Pow (set Ω) − {{}}) using assms(3) allAllocationsInPowerset
  by blast
  thus ?thesis using assms(1,2) resolvingBidBridgingLemma by blast
qed

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

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

lemma 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
  by simpforce
proof –
  have card (N∪{seller}) > 0 using assms(1) sup-eq-bot-iff insert-not-empty
    by (metis card-qt-0-iff finite.emptyI finite.insertI finite-UnI)
  moreover have distinct Ω using assms(2) by simp
  moreover have a ∈ allAllocations (N∪{seller}) (set Ω) using assms(3) by fastforce
  ultimately show ?thesis unfolding randomBidsAlg-def by (rule resolvingBid-
    BridgingLemmaVariant1)
qed

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 a ⊆ N × (Pow (set Ω) − {{}}) using assms(3) allAllocationsInPowerset
  by blast
  thus ?thesis using assms(1,2) resolvingBidBridgingLemma by blast
qed

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

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

lemma 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
  by simpforce
proof –
  have card (N∪{seller}) > 0 using assms(1) sup-eq-bot-iff insert-not-empty
    by (metis card-qt-0-iff finite.emptyI finite.insertI finite-UnI)
  moreover have distinct Ω using assms(2) by simp
  moreover have a ∈ allAllocations (N∪{seller}) (set Ω) using assms(3) by fastforce
  ultimately show ?thesis unfolding randomBidsAlg-def by (rule resolvingBid-
    BridgingLemmaVariant1)
qed

lemma tiebreakingGivesSingleton:
  assumes distinct Ω set Ω ≠ {} finite N
  shows card (argmax (sum (randomBidsAlg N Ω b r))
    (maximalStrictAllocations N (set Ω) b)) = 1
proof –
have \( \forall a \in \text{maximalStrictAllocations \ N (set \ \Omega) \ b} \).

\[ \text{sum (randomBids \ N \ \Omega \ b \ r) \ a = sum (randomBidsAlg \ N \ \Omega \ b \ r) \ a} \]

using assms(3,1) resolvingBidBridgingLemmaVariant2 by blast

then have \( \text{argmax (sum (randomBidsAlg \ N \ \Omega \ b \ r)) (maximalStrictAllocations \ N (set \ \Omega) \ b)} \) =

\[ \text{argmax (sum (randomBids \ N \ \Omega \ b \ r)) (maximalStrictAllocations \ N (set \ \Omega) \ b)} \]

using argmaxEquivalence by blast

moreover have \( \text{card ... = 1 using assms by (rule vcgaDefinitenessVariant)} \)

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

qed

theorem maximalAllocationBridgingTheorem:

assumes finite \( N \ \text{distinct} \ \Omega \)

shows \( \text{maximalStrictAllocations \ N (set \ \Omega) \ b = maximalStrictAllocationsAlg \ N \ \Omega \ b} \)

proof –

let \( ?N = \{\text{seller}\} \cup N \)

have \( \text{card \ ?N > 0 using assms(1)} \)

by (metis (full-types) card-gt-0-iff finite-insert insert-is-\text{Un} insert-not-empty)

thus \( ?\text{thesis using assms(2)} \ \text{allAllocationsBridgingLemma by metis} \)

qed

theorem vcgaAlgDefinedness:

assumes \( \text{distinct} \ \Omega \ \text{set} \ \Omega \neq \{\} \ \text{finite} \ N \)

shows \( \text{card (argmax (sum (randomBidsAlg \ N \ \Omega \ b \ r)) (maximalStrictAllocationsAlg \ N \ \Omega \ b)) = 1} \)

proof –

have \( \text{card (argmax (sum (randomBidsAlg \ N \ \Omega \ b \ r)) (maximalStrictAllocations \ N (set \ \Omega) \ b)) = 1} \)

using assms by (rule tiebreakingGivesSingleton)

moreover have \( \text{maximalStrictAllocations \ N (set \ \Omega) \ b = maximalStrictAllocationsAlg \ N \ \Omega \ b} \)

using assms(3,1) by (rule maximalAllocationBridgingTheorem)

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

qed

12 VCG auction: Scala code extraction

theory CombinatorialAuctionCodeExtraction

imports

CombinatorialAuction

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

132
begin

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

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

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

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

**export-code** vcgAlg payments allocationPrettyPrint in Scala module-name VCG  
file VCG—withoutWrapper.scala

end

References


