

Vickrey-Clarke-Groves (VCG) Auctions

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

September 13, 2023

Abstract

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

Contents

1	Introduction	3
1.1	Rationale for developing set theory as replacing one bidder in a second price auction	4
1.2	Bridging	4
1.3	Main theorems	4
1.4	Scala code extraction	5
2	Additional material that we would have expected in Set.thy	5
2.1	Equality	5
2.2	Trivial sets	6
2.3	The image of a set under a function	6
2.4	Big Union	6
2.5	Miscellaneous	7

*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	8
4	Locus where a function or a list (of linord type) attains its maximum value	16
5	Additional operators on relations, going beyond Relations.thy, and properties of these operators	18
5.1	Evaluating a relation as a function	19
5.2	Restriction	19
5.3	Relation outside some set	20
5.4	Flipping pairs of relations	20
5.5	Evaluation as a function	21
5.6	Paste	21
6	Additional properties of relations, and operators on relations, as they have been defined by Relations.thy	22
6.1	Right-Uniqueness	22
6.2	Converse	25
6.3	Injectivity	26
7	Toolbox of various definitions and theorems about sets, relations and lists	26
7.1	Facts and notations about relations, sets and functions.	26
7.2	Ordered relations	30
7.3	Indicator function in set-theoretical form.	41
7.4	Lists	43
7.5	Computing all the permutations of a list	44
7.6	A more computable version of <i>toFunction</i>	46
7.7	Cardinalities of sets.	47
7.8	Some easy properties on real numbers	50
8	Definitions about those Combinatorial Auctions which are strict (i.e., which assign all the available goods)	50
8.1	Types	51
8.2	VCG mechanism	51
9	Sets of injections, partitions, allocations expressed as suitable subsets of the corresponding universes	52
9.1	Preliminary lemmas	53
9.2	Definitions of various subsets of <i>UNIV</i>	53
9.3	Results about the sets defined in the previous section	53
9.4	Bridging theorem for injections	63
9.5	Computable injections	67

10 Termination theorem for uniform tie-breaking	69
10.1 Uniform tie breaking: definitions	69
10.2 Termination theorem for the uniform tie-breaking scheme . .	71
10.3 Results on summed bid vectors	72
10.4 From Pseudo-allocations to allocations	79
11 VCG auction: definitions and theorems	83
11.1 Definition of a VCG auction scheme, through the pair $(vcga,$ $vcgp)$	83
11.2 Computable versions of the VCG formalization	90
12 VCG auction: Scala code extraction	94

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 $\llbracket ?A \subseteq ?B; ?B \subseteq ?A \rrbracket \implies ?A = ?B$ and $(\bigwedge x. x \in ?A \implies x \in ?B) \implies ?A \subseteq ?B$ to a single step

```
lemma equalitySubsetI:  $(\bigwedge x. x \in A \implies x \in B) \implies (\bigwedge x. x \in B \implies x \in A) \implies A = B$   
  <proof>
```

2.2 Trivial sets

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

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

The empty set is trivial.

lemma *trivial-empty: trivial* $\{\}$
<proof>

A singleton set is trivial.

lemma *trivial-singleton: trivial* $\{x\}$
<proof>

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

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

<proof>

Any subset of a trivial set is trivial.

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

<proof>

There are no two different elements in a trivial set.

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

<proof>

2.3 The image of a set under a function

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

lemma *image-Collect-mem: $\{ f x \mid x . x \in S \} = f ' S$*
<proof>

2.4 Big Union

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

lemma *Union-member: $(\exists S \in F . x \in S) \longleftrightarrow x \in \bigcup F$*
<proof>

2.5 Miscellaneous

lemma *trivial-subset-non-empty*: **assumes** *trivial* $t \cap X \neq \{\}$
shows $t \subseteq X$
<proof>

lemma *trivial-implies-finite*: **assumes** *trivial* X
shows *finite* X
<proof>

lemma *lm01*: **assumes** *trivial* $(A \times B)$
shows $(\text{finite } (A \times B) \ \& \ \text{card } A * (\text{card } B) \leq 1)$
<proof>

lemma *lm02*: **assumes** *finite* X
shows *trivial* $X = (\text{card } X \leq 1)$
<proof>

lemma *lm03*: **shows** *trivial* $\{x\}$
<proof>

lemma *lm04*: **assumes** *trivial* $X \ \{x\} \subseteq X$
shows $\{x\} = X$
<proof>

lemma *lm05*: **assumes** \neg *trivial* X *trivial* T
shows $X - T \neq \{\}$
<proof>

lemma *lm06*: **assumes** $(\text{finite } (A \times B) \ \& \ \text{card } A * (\text{card } B) \leq 1)$
shows *trivial* $(A \times B)$
<proof>

lemma *lm07*: *trivial* $(A \times B) = (\text{finite } (A \times B) \ \& \ \text{card } A * (\text{card } B) \leq 1)$
<proof>

lemma *trivial-empty-or-singleton*: *trivial* $X = (X = \{\} \vee X = \{\text{the-elem } X\})$
<proof>

lemma *trivial-cartesian*: **assumes** *trivial* X *trivial* Y
shows *trivial* $(X \times Y)$
<proof>

lemma *trivial-same*: *trivial* $X = (\forall x1 \in X. \forall x2 \in X. x1 = x2)$
<proof>

lemma *lm08*: **assumes** $(\text{Pow } X \subseteq \{\{\}, X\})$
shows *trivial* X
<proof>

lemma *lm09*: **assumes** *trivial X*
shows $(Pow\ X \subseteq \{\{\}, X\})$
 $\langle proof \rangle$

lemma *lm10*: *trivial X = (Pow X \subseteq { {}, X })*
 $\langle proof \rangle$

lemma *lm11*: $(\{x\} \times UNIV) \cap P = \{x\} \times (P \text{ “ } \{x\})$
 $\langle proof \rangle$

lemma *lm12*: $(x, y) \in P = (y \in P \text{ “ } \{x\})$
 $\langle proof \rangle$

lemma *lm13*: **assumes** *inj-on f A inj-on f B*
shows $inj\text{-on}\ f\ (A \cup B) = (f \text{ “ } (A - B) \cap (f \text{ “ } (B - A)) = \{\})$
 $\langle proof \rangle$

lemma *injection-union*: **assumes** *inj-on f A inj-on f B (f ‘ A) \cap (f ‘ B) = {}*
shows *inj-on f (A \cup B)*
 $\langle proof \rangle$

lemma *lm14*: $(Pow\ X = \{X\}) = (X = \{\})$
 $\langle proof \rangle$

end

3 Partitions of sets

theory *Partitions*

imports

SetUtils

begin

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

P is a family of non-overlapping sets.

definition *is-non-overlapping*

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

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

lemma *subset-is-non-overlapping:*

assumes *subset: $P \subseteq Q$ and
non-overlapping: is-non-overlapping Q*
shows *is-non-overlapping P*

<proof>

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

lemma *remove-from-eq-class-preserves-disjoint:*

fixes *elem::'a
and $X::'a$ set
and $P::'a$ set set*
assumes *non-overlapping: is-non-overlapping P
and eq-class: $X \in P$
and elem: $elem \in X$*
shows *$X - \{elem\} \notin P$*

<proof>

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

lemma *non-overlapping-extension1:*

fixes *$P::'a$ set set
and $X::'a$ set*
assumes *partition: is-non-overlapping P
and disjoint: $X \cap \bigcup P = \{\}$
and non-empty: $X \neq \{\}$*
shows *is-non-overlapping (insert $X P$)*

<proof>

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

lemma *disj-eq-classes:*

fixes *$P::'a$ set set
and $X::'a$ set*
assumes *is-non-overlapping P
and $X \in P$*
shows *$X \cap \bigcup (P - \{X\}) = \{\}$*

<proof>

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

lemma *no-empty-in-non-overlapping:*

assumes *is-non-overlapping p*
shows *$\{\} \notin p$*

<proof>

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

definition *is-partition-of (infix partitions 75)*
where *is-partition-of* $P A = (\bigcup P = A \wedge \textit{is-non-overlapping } P)$

No partition of a non-empty set is empty.

lemma *non-empty-imp-non-empty-partition:*

assumes $A \neq \{\}$
and $P \textit{ partitions } A$
shows $P \neq \{\}$
 $\langle \textit{proof} \rangle$

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:* $P \textit{ partitions } A$
obtains X **where** $x \in X$ **and** $X \in P$
 $\langle \textit{proof} \rangle$

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:* $P \textit{ partitions } A$
shows $\exists S \in P - \{B\} . x \in S$

$\langle \textit{proof} \rangle$

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:* $P \textit{ partitions } A$
shows $\exists! X \in P . x \in X$
 $\langle \textit{proof} \rangle$

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

lemma *set-partitions-itself:*

assumes $A \neq \{\}$
shows $\{A\} \textit{ partitions } A$ $\langle \textit{proof} \rangle$

The empty set is a partition of the empty set.

lemma *emptyset-part-emptyset1:*

shows $\{\} \textit{ partitions } \{\}$
 $\langle \textit{proof} \rangle$

Any partition of the empty set is empty.

lemma *emptyset-part-emptyset2:*

assumes P partitions $\{\}$
shows $P = \{\}$
 <proof>

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

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

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

lemma *emptyset-part-emptyset3*:
shows *all-partitions* $\{\} = \{\{\}\}$
 <proof>

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

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

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

lemma *non-overlapping-extension2*:
fixes $new_el::'a$
and $P::'a \text{ set set}$
and $X::'a \text{ set}$
assumes *non-overlapping: is-non-overlapping* P
and *class-element*: $X \in P$
and $new: new_el \notin \bigcup P$
shows *is-non-overlapping* (*insert-into-member* $new_el P X$)
 <proof>

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 $'a \text{ set set set}$, 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 $'a \text{ set set list}$, but we need $'a \text{ set list list}$. This is because it is hard to impossible to convert a set to a list, whereas it is easy to convert a *list* to a *set*.

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

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 $set (insert-into-member-list\ new-el\ Sets\ S) = insert-into-member\ new-el\ (set\ Sets)\ S$
 $\langle proof \rangle$

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

lemma *insert-into-member-partition1*:
fixes *elem*::'a
and *P*::'a set set
and *set*::'a set
shows $\bigcup (insert-into-member\ elem\ P\ set) = \bigcup (insert\ (set\ \cup\ \{elem\})\ (P - \{set\}))$
 $\langle proof \rangle$

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

definition *coarser-partitions-with* ::'a \Rightarrow 'a set set \Rightarrow 'a set set set
where *coarser-partitions-with new-el P* =
insert
— Let P be a partition of a set Set ,
— and suppose $new-el \notin Set$, i.e. $\{new-el\} \notin P$,
— then the following constructs a partition of $Set \cup \{new-el\}$ 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 $new-el \notin Set$,
— then the following constructs
— the set of those partitions of $Set \cup \{new-el\}$ 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 \Rightarrow 'a set list \Rightarrow '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 $Set \cup \{new-el\}$ obtained by
- inserting a new class $\{new-el\}$ and leaving all previous classes unchanged.
- $(\{new-el\} \# P)$
- $\#$
- Let P be a partition of a set Set ,
- and suppose $new-el \notin Set$,
- then the following constructs
- the set of those partitions of $Set \cup \{new-el\}$ 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 $set (map set (coarser-partitions-with-list new-el P)) =$
 $coarser-partitions-with new-el (set P)$
 $\langle proof \rangle$

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
 $\langle proof \rangle$

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

lemma *coarser-partitions-covers*:
fixes $elem::'a$
and $P::'a set set$
and $Q::'a set set$
assumes $Q \in coarser-partitions-with elem P$
shows $\bigcup Q = insert elem (\bigcup P)$
 $\langle proof \rangle$

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

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

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

lemma *partition-without-covers*:
fixes *elem::'a*
and *P::'a set set*
shows $\bigcup (partition-without\ elem\ P) = (\bigcup P) - \{elem\}$
⟨*proof*⟩

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

lemma *super-class*:
assumes $X \in partition-without\ elem\ P$
obtains *Z* **where** $Z \in P$ **and** $X = Z - \{elem\}$
⟨*proof*⟩

The class of sets obtained by removing an element from a non-overlapping class is another non-overlapping class.

lemma *non-overlapping-without-is-non-overlapping*:
fixes *elem::'a*
and *P::'a set set*
assumes *is-non-overlapping P*
shows *is-non-overlapping (partition-without elem P)* (**is** *is-non-overlapping ?Q*)
⟨*proof*⟩

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 ∈ ∪ P*
shows $P \in coarser-partitions-with\ elem\ (partition-without\ elem\ P)$
(**is** $P \in coarser-partitions-with\ elem\ ?Q$)
⟨*proof*⟩

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

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

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

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

$concat (map (coarser-partitions-with-list elem) Ps)$

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

lemma $all-coarser-partitions-with-list-equivalence$:

fixes $elem::'a$
and $Ps::'a\ set\ list\ list$
assumes $distinct: \forall P \in set\ Ps . distinct\ P$
shows $set (map\ set (all-coarser-partitions-with-list\ elem\ Ps)) = all-coarser-partitions-with\ elem (set (map\ set\ Ps))$
(is $?list-expr = ?set-expr$
 $\langle proof \rangle$)

all partitions of a set (given as list) in form of a set

fun $all-partitions-set :: 'a\ list \Rightarrow 'a\ set\ set\ set$

where

$all-partitions-set\ [] = \{\{\}\}$ |

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

all partitions of a set (given as list) in form of a list

fun $all-partitions-list :: 'a\ list \Rightarrow 'a\ set\ list\ list$

where

$all-partitions-list\ [] = [[]]$ |

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

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

lemma $coarser-partitions-with-list-distinct$:

fixes ps

assumes $ps-coarser: ps \in set (coarser-partitions-with-list\ x\ Q)$

and $distinct: distinct\ Q$

and $partition: is-non-overlapping (set\ Q)$

and $new: \{x\} \notin set\ Q$

shows $distinct\ ps$

$\langle proof \rangle$

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

lemma $all-partitions-equivalence'$:

fixes $xs::'a\ list$

shows $distinct\ xs \Longrightarrow$

$((set (map\ set (all-partitions-list\ xs)) =$

$all-partitions (set\ xs)) \wedge (\forall ps \in set (all-partitions-list\ xs) . distinct\ ps))$

$\langle proof \rangle$

The classical definition $all-partitions$ and the algorithmic (constructive) definition $all-partitions-list$ are equivalent. This is a front-end theorem derived from $distinct\ ?xs \Longrightarrow set (map\ set (all-partitions-list\ ?xs)) = all-partitions$

$(set\ ?xs) \wedge (\forall ps \in set\ (all-partitions-list\ ?xs). distinct\ ps)$; it does not make the auxiliary statement about partitions being distinct lists.

theorem *all-partitions-paper-equiv-alg*:

fixes $xs :: 'a\ list$

shows $distinct\ xs \implies set\ (map\ set\ (all-partitions-list\ xs)) = all-partitions\ (set\ xs)$

<proof>

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

definition *all-partitions-alg* $:: 'a :: linorder\ set \Rightarrow 'a\ set\ list\ list$

where $all-partitions-alg\ X = all-partitions-list\ (sorted-list-of-set\ X)$

end

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

theory *Argmax*

imports *Main*

begin

Structural induction is used in proofs on lists.

lemma *structInduct*: **assumes** $P\ []$ **and** $\forall x\ xs. P\ (xs) \longrightarrow P\ (x\#\ xs)$
shows $P\ l$

<proof>

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

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

lemma *argmaxLemma*: $argmax\ f\ A = \{ x \in A . f\ x = Max\ (f\ 'A) \}$

<proof>

lemma *maxLemma*:

assumes $x \in X\ finite\ X$

shows $Max\ (f\ 'X) \geq f\ x$

(is $?L \geq ?R$) *<proof>*

lemma *lm01*:

$argmax\ f\ A = A \cap f\ -'\{Max\ (f\ 'A)\}$

<proof>

lemma *lm02*:

assumes $y \in f\ 'A$

shows $A \cap f^{-1}\{y\} \neq \{\}$
 ⟨proof⟩

lemma *argmaxEquivalence*:

assumes $\forall x \in X. f\ x = g\ x$

shows $\text{argmax}\ f\ X = \text{argmax}\ g\ X$

⟨proof⟩

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

corollary *argmax-non-empty-iff*: **assumes** $\text{finite}\ X\ X \neq \{\}$

shows $\text{argmax}\ f\ X \neq \{\}$

⟨proof⟩

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* :: $('a \Rightarrow \text{bool}) \Rightarrow 'a\ \text{list} \Rightarrow \text{nat}\ \text{list}$

where $\text{filterpositions}\ P\ l = \text{map}\ \text{snd}\ (\text{filter}\ (P\ o\ \text{fst})\ (\text{zip}\ l\ (\text{upt}\ 0\ (\text{size}\ l))))$

definition *filterpositions2*

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

definition *maxpositions*

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

lemma *lm03*: $\text{maxpositions}\ l = [n. n \leftarrow [0..<\text{size}\ l], !n \geq \text{Max}\ (\text{set}\ l)]$

⟨proof⟩

definition *argmaxList*

where $\text{argmaxList}\ f\ l = \text{map}\ (\text{nth}\ l)\ (\text{maxpositions}\ (\text{map}\ f\ l))$

lemma *lm04*: $[n . n <- l, P\ n] = [n . n <- l, n \in \text{set}\ l, P\ n]$

⟨proof⟩

lemma *lm05*: $[n . n <- [0..<m], P\ n] = [n . n <- [0..<m], n \in \text{set}\ [0..<m], P\ n]$

⟨proof⟩

lemma *lm06*: **fixes** $f\ m\ P$
shows $(\text{map } f\ [n . n <- [0..<m], P\ n]) = [f\ n . n <- [0..<m], P\ n]$
 $\langle\text{proof}\rangle$

lemma *map-commutes-a*: $[f\ n . n <- [], Q\ (f\ n)] = [x <- (\text{map } f\ []). Q\ x]$
 $\langle\text{proof}\rangle$

lemma *map-commutes-b*: $\forall\ x\ xs. ([f\ n . n <- xs, Q\ (f\ n)] = [x <- (\text{map } f\ xs). Q\ x]) \longrightarrow$
 $[f\ n . n <- (x\#\!xs), Q\ (f\ n)] = [x <- (\text{map } f\ (x\#\!xs)). Q\ x]$
 $\langle\text{proof}\rangle$

lemma *map-commutes*: **fixes** $f::'a \Rightarrow 'b$ **fixes** $Q::'b \Rightarrow \text{bool}$ **fixes** $xs::'a\ \text{list}$
shows $[f\ n . n <- xs, Q\ (f\ n)] = [x <- (\text{map } f\ xs). Q\ x]$
 $\langle\text{proof}\rangle$

lemma *lm07*: **fixes** $f\ l$
shows $\text{maxpositions } (\text{map } f\ l) =$
 $[n . n <- [0..<\text{size } l], f\ (l!\!n) \geq \text{Max } (f\ (\text{set } l))]$
 $(\text{is } \text{maxpositions } (?f\ l) = -)$
 $\langle\text{proof}\rangle$

lemma *lm08*: **fixes** $f\ l$
shows $\text{argmaxList } f\ l =$
 $[l!\!n . n <- [0..<\text{size } l], f\ (l!\!n) \geq \text{Max } (f\ (\text{set } l))]$
 $\langle\text{proof}\rangle$

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

theorem *argmaxadequacy*: **fixes** $f::'a \Rightarrow ('b::\text{linorder})$ **fixes** $l::'a\ \text{list}$
shows $\text{argmaxList } f\ l = [x <- l. f\ x \geq \text{Max } (f\ (\text{set } l))]$
 $(\text{is } ?lh = -)$
 $\langle\text{proof}\rangle$

end

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

theory *RelationOperators*
imports
SetUtils
HOL-Library.Code-Target-Nat

begin

5.1 Evaluating a relation as a function

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

fun *eval-rel-or* :: ('a × 'b) set ⇒ 'a ⇒ 'b ⇒ 'b
where *eval-rel-or* R a z = (let im = R “ {a} in if card im = 1 then the-elem im else z)

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

definition *runiq* :: ('a × 'b) set ⇒ bool
where *runiq* R = (∀ X . trivial X → trivial (R “ X))

5.2 Restriction

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

definition *restrict* :: ('a × 'b) set ⇒ 'a set ⇒ ('a × 'b) set (**infix** || 75)
where R || X = (X × Range R) ∩ R

extensional characterization of the pairs within a restricted relation

lemma *restrict-ext*: R || X = {(x, y) | x y . x ∈ X ∧ (x, y) ∈ R}
⟨proof⟩

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⟩

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

lemma *restrict-empty*: P || {} = {}
⟨proof⟩

A restriction is a subrelation of the original relation.

lemma *restriction-is-subrel*: P || X ⊆ P
⟨proof⟩

Restricting a relation only has an effect within its domain.

lemma *restriction-within-domain*: P || X = P || (X ∩ (Domain P))
⟨proof⟩

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

lemma *restrict-to-singleton*: P || {x} = {x} × (P “ {x})
⟨proof⟩

5.3 Relation outside some set

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

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

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

lemma *outside-reduces-domain*: $\text{Domain } (P \text{ outside } X) = (\text{Domain } P) - X$
⟨proof⟩

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 $\text{Domain } (R \text{ outside } \{x\}) = A$
⟨proof⟩

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

lemma *outside-union-restrict*: $P = (P \text{ outside } X) \cup (P \parallel X)$
⟨proof⟩

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*: $\text{Range } (R \text{ outside } X) \subseteq R \text{ `` } (\text{Domain } R - X)$
⟨proof⟩

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

lemma *Range-outside-sub*:
assumes $\text{Range } R \subseteq Y$
shows $\text{Range } (R \text{ outside } X) \subseteq Y$
⟨proof⟩

5.4 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 $\text{flip } \text{tup} \in R^{-1}$
⟨proof⟩

Flipping a pair twice doesn't change it.

lemma *flip-flip*: $\text{flip} (\text{flip } \text{tup}) = \text{tup}$
 ⟨*proof*⟩

Flipping all pairs in a relation yields the converse relation.

lemma *flip-conv*: $\text{flip} \text{ ` } R = R^{-1}$
 ⟨*proof*⟩

5.5 Evaluation as a function

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

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

5.6 Paste

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

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

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.

lemma *paste-subrel*:
assumes $P \parallel \text{Domain } Q \subseteq Q$
shows $P +* Q = P \cup Q$
 ⟨*proof*⟩

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

lemma *paste-disj-domains*:
assumes $\text{Domain } P \cap \text{Domain } Q = \{\}$
shows $P +* Q = P \cup Q$
 ⟨*proof*⟩

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

lemma *paste-outside-restrict*: $P = (P \text{ outside } X) +* (P \parallel X)$
 ⟨*proof*⟩

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

lemma *paste-Domain*: $\text{Domain}(P +* Q) = \text{Domain } P \cup \text{Domain } Q$ ⟨*proof*⟩

Pasting two relations yields a subrelation of their union.

lemma *paste-sub-Un*: $P +* Q \subseteq P \cup Q$
<proof>

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

lemma *paste-Range*: $\text{Range } (P +* Q) \subseteq \text{Range } P \cup \text{Range } Q$
<proof>
end

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

theory *RelationProperties*
imports
 RelationOperators

begin

6.1 Right-Uniqueness

lemma *injflip*: *inj-on flip A*
<proof>

lemma *lm01*: $\text{card } P = \text{card } (P^{\wedge-1})$
<proof>

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

lemma *lm02*: *trivial X = (X = {}) \vee card X = 1*
<proof>

lemma *lm03*: *trivial P = trivial (P[^]-1)*
<proof>

lemma *restrictedRange*: $\text{Range } (P || X) = P^{\wedge} X$
<proof>

lemma *doubleRestriction*: $((P || X) || Y) = (P || (X \cap Y))$
<proof>

lemma *restrictedDomain*: $\text{Domain } (R || X) = \text{Domain } R \cap X$
<proof>

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

lemma *subrel-runiq*:

assumes *runiq* $Q \subseteq P \subseteq Q$
shows *runiq* P
 \langle *proof* \rangle

lemma *rightUniqueInjectiveOnFirstImplication*:

assumes *runiq* P
shows *inj-on fst* P
 \langle *proof* \rangle

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

lemma *runiq-alt*: $\text{runiq } R \longleftrightarrow (\forall x . \text{trivial } (R \text{ `` } \{x\}))$
 \langle *proof* \rangle

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

lemma *runiq-wrt-eval-rel*: $\text{runiq } R = (\forall x . R \text{ `` } \{x\} \subseteq \{R \text{ ,, } x\})$
 \langle *proof* \rangle

lemma *rightUniquePair*:

assumes *runiq* f
assumes $(x,y) \in f$
shows $y=f \text{ ,, } x$
 \langle *proof* \rangle

lemma *runiq-basic*: $\text{runiq } R \longleftrightarrow (\forall x y y' . (x, y) \in R \wedge (x, y') \in R \longrightarrow y = y')$
 \langle *proof* \rangle

lemma *rightUniqueFunctionAfterInverse*:

assumes *runiq* f
shows $f \text{ `` } (f \text{ } ^{-1} \text{ `` } Y) \subseteq Y$
 \langle *proof* \rangle

lemma *lm04*:

assumes *runiq* f $y1 \in \text{Range } f$
shows $(f \text{ } ^{-1} \text{ `` } \{y1\} \cap f \text{ } ^{-1} \text{ `` } \{y2\} \neq \{\}) = (f \text{ } ^{-1} \text{ `` } \{y1\} = f \text{ } ^{-1} \text{ `` } \{y2\})$
 \langle *proof* \rangle

lemma *converse-Image*:

assumes *runiq*: *runiq* R
and *runiq-conv*: *runiq* $(R \text{ } ^{-1})$
shows $(R \text{ } ^{-1}) \text{ `` } R \text{ `` } X \subseteq X$
 \langle *proof* \rangle

lemma *lm05*:

assumes *inj-on fst* P
shows *runiq* P
 \langle *proof* \rangle

lemma *rightUniqueInjectiveOnFirst*: $(\text{runiq } P) = (\text{inj-on fst } P)$
<proof>

lemma *disj-Un-runiq*:
assumes *runiq P runiq Q* $(\text{Domain } P) \cap (\text{Domain } Q) = \{\}$
shows *runiq (P \cup Q)*
<proof>

lemma *runiq-paste1*:
assumes *runiq Q runiq (P outside Domain Q)*
shows *runiq (P $+*$ Q)*
<proof>

corollary *runiq-paste2*:
assumes *runiq Q runiq P*
shows *runiq (P $+*$ Q)*
<proof>

lemma *rightUniqueRestrictedGraph*: *runiq* $\{(x, f x) \mid x. P x\}$
<proof>

lemma *rightUniqueSetCardinality*:
assumes $x \in \text{Domain } R$ *runiq R*
shows $\text{card } (R \text{ `` } \{x\}) = 1$
<proof>

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

lemma *Image-runiq-eq-eval*:
assumes $x \in \text{Domain } R$ *runiq R*
shows $R \text{ `` } \{x\} = \{R \text{ ,, } x\}$
<proof>

lemma *lm06*:
assumes *trivial f*
shows *runiq f*
<proof>

A singleton relation is right-unique.

corollary *runiq-singleton-rel*: *runiq* $\{(x, y)\}$
<proof>

The empty relation is right-unique

lemma *runiq-emptyrel*: *runiq* $\{\}$
<proof>

lemma *runiq-wrt-ex1*:

$\text{runiq } R \iff (\forall a \in \text{Domain } R . \exists! b . (a, b) \in R)$
 ⟨proof⟩

alternative characterization of the fact that, if a relation R is right-unique, its evaluation $R \text{ ,, } x$ on some argument x in its domain, occurs in R 's range. Note that we need $\text{runiq } R$ in order to get a definite value for $R \text{ ,, } x$

lemma *eval-runiq-rel:*

assumes *domain:* $x \in \text{Domain } R$
and *runiq:* $\text{runiq } R$
shows $(x, R \text{ ,, } x) \in R$
 ⟨proof⟩

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

lemma *eval-runiq-in-Range:*

assumes *runiq* R
and $a \in \text{Domain } R$
shows $R \text{ ,, } a \in \text{Range } R$
 ⟨proof⟩

6.2 Converse

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

lemma *converse-Image-singleton-Domain:*

assumes *runiq:* $\text{runiq } R$
and *runiq-conv:* $\text{runiq } (R^{-1})$
and *domain:* $x \in \text{Domain } R$
shows $R^{-1} \text{ `` } R \text{ `` } \{x\} = \{x\}$
 ⟨proof⟩

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

lemma *disj-Domain-imp-disj-Image:*

assumes $\text{Domain } R \cap X \cap Y = \{\}$
assumes *runiq* R
and *runiq* (R^{-1})
shows $(R \text{ `` } X) \cap (R \text{ `` } Y) = \{\}$
 ⟨proof⟩

lemma *runiq-converse-paste-singleton:*

assumes *runiq* $(P \hat{-} 1) \text{ } y \notin (\text{Range } P)$
shows *runiq* $((P \text{ +* } \{(x, y)\})^{-1})$
(is ?u (?P^{^-} 1))
 ⟨proof⟩

6.3 Injectivity

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

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

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)  $Y = \text{concat } [ [ R +* \{(x,y)\} . y \leftarrow \text{sorted-list-of-set}$   
( $Y - \text{Range } R$ ) ]  
        .  $R \leftarrow \text{injections-alg } xs Y$  ]
```

lemma *Image-within-domain'*:
fixes $x R$
shows $(x \in \text{Domain } R) = (R \text{ “ } \{x\} \neq \{\})$
<proof>

end

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

theory *MiscTools*

imports

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

begin

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

7.1 Facts and notations about relations, sets and functions.

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

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

abbreviation *singlepaste*

where $singlepaste\ f\ pair == f\ +* \{(fst\ pair,\ snd\ pair)\}$

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

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

abbreviation *singleoutside* (**infix** -- 75)

where $f\ --\ x \equiv f\ outside\ \{x\}$

Turns a HOL function into a set-theoretical function

definition

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

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

definition

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

lemma

$toFunction = eval-rel$

<proof>

lemma *lm001*:

$((P \cup Q) \parallel X) = ((P \parallel X) \cup (Q \parallel X))$

<proof>

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

definition *update*

where $update\ P\ Q = P\ +* (Q \parallel (Domain\ P))$

notation *update* (**infix** +^ 75)

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

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

graph is like *Graph*, but with a built-in restriction to a given set X . This makes it computable for finite X , whereas $Graph\ f \parallel X$ 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 $R \supseteq \text{graph } (\text{Domain } R) (\text{toFunction } R)$
 $\langle \text{proof} \rangle$

lemma *lm003*:
assumes *runiq* *R*
shows $R \subseteq \text{graph } (\text{Domain } R) (\text{toFunction } R)$
 $\langle \text{proof} \rangle$

lemma *lm004*:
assumes *runiq* *R*
shows $R = \text{graph } (\text{Domain } R) (\text{toFunction } R)$
 $\langle \text{proof} \rangle$

lemma *domainOfGraph*:
 $\text{runiq}(\text{graph } X f) \ \& \ \text{Domain}(\text{graph } X f) = X$
 $\langle \text{proof} \rangle$

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

lemma *imageEquivalence*:
assumes *runiq* ($f::('a \times ('b \text{ set})) \text{ set}$) $x \in \text{Domain } f$
shows $f \text{ ,, } x = f \text{ ,, , } x$
 $\langle \text{proof} \rangle$

lemma *lm005*:
 $\text{Graph } f = \text{graph } \text{UNIV } f$
 $\langle \text{proof} \rangle$

lemma *graphIntersection*:
 $\text{graph } (X \cap Y) f \subseteq ((\text{graph } X f) \parallel Y)$
 $\langle \text{proof} \rangle$

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

lemma *outsideOutside*:
 $((P \text{ outside } X) \text{ outside } Y) = P \text{ outside } (X \cup Y)$
 $\langle \text{proof} \rangle$

corollary *lm006*:
 $((P \text{ outside } X) \text{ outside } X) = P \text{ outside } X$
 $\langle \text{proof} \rangle$

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

shows $P +* Q = (P \text{ outside } X) +* Q$
<proof>

corollary *lm008:*

$P +* Q = (P \text{ outside } (\text{Domain } Q)) +* Q$
<proof>

corollary *outsideUnion:*

$R = (R \text{ outside } \{x\}) \cup (\{x\} \times (R \text{ “ } \{x\}))$
<proof>

lemma *lm009:*

$P = P \cup \{x\} \times P \text{ “ } \{x\}$
<proof>

corollary *lm010:*

$R = (R \text{ outside } \{x\}) +* (\{x\} \times (R \text{ “ } \{x\}))$
<proof>

lemma *lm011:*

$R \subseteq R +* (\{x\} \times (R \text{ “ } \{x\}))$
<proof>

lemma *lm012:*

$R \supseteq R +* (\{x\} \times (R \text{ “ } \{x\}))$
<proof>

lemma *lm013:*

$R = R +* (\{x\} \times (R \text{ “ } \{x\}))$
<proof>

lemma *rightUniqueTrivialCartes:*

assumes *trivial* Y

shows *runiq* $(X \times Y)$

<proof>

lemma *lm014:*

runiq $((X \times \{x\}) +* (Y \times \{y\}))$
<proof>

lemma *lm015:*

$(P \parallel (X \cap Y)) \subseteq (P \parallel X) \quad \& \quad P \text{ outside } (X \cup Y) \subseteq P \text{ outside } X$
<proof>

lemma *lm016:*

$P \parallel X \subseteq (P \parallel (X \cup Y)) \quad \& \quad P \text{ outside } X \subseteq P \text{ outside } (X \cap Y)$
<proof>

lemma *lm017*:
 $P''(X \cap \text{Domain } P) = P''X$
 ⟨proof⟩

lemma *cardinalityOneSubset*:
 assumes $\text{card } X=1$ and $X \subseteq Y$
 shows $\text{Union } X \in Y$
 ⟨proof⟩

lemma *cardinalityOneTheElem*:
 assumes $\text{card } X=1$ $X \subseteq Y$
 shows *the-elem* $X \in Y$
 ⟨proof⟩

lemma *lm018*:
 $(R \text{ outside } X1) \text{ outside } X2 = (R \text{ outside } X2) \text{ outside } X1$
 ⟨proof⟩

7.2 Ordered relations

lemma *lm019*:
 assumes $\text{card } X \geq 1 \ \forall x \in X. y > x$
 shows $y > \text{Max } X$
 ⟨proof⟩

lemma *lm020*:
 assumes *finite* $X \ mx \in X \ f \ x < f \ mx$
 shows $x \notin \text{argmax } f \ X$
 ⟨proof⟩

lemma *lm021*:
 assumes *finite* $X \ mx \in X \ \forall x \in X - \{mx\}. f \ x < f \ mx$
 shows $\text{argmax } f \ X \subseteq \{mx\}$
 ⟨proof⟩

lemma *lm022*:
 assumes *finite* $X \ mx \in X \ \forall x \in X - \{mx\}. f \ x < f \ mx$
 shows $\text{argmax } f \ X = \{mx\}$
 ⟨proof⟩

corollary *argmaxProperty*:
 $(\text{finite } X \ \& \ mx \in X \ \& \ (\forall aa \in X - \{mx\}. f \ aa < f \ mx)) \longrightarrow \text{argmax } f \ X = \{mx\}$
 ⟨proof⟩

corollary *lm023*:
 assumes *finite* $X \ mx \in X \ \forall x \in X. x \neq mx \longrightarrow f \ x < f \ mx$
 shows $\text{argmax } f \ X = \{mx\}$

<proof>

lemma *lm024*:

assumes $f \circ g = id$

shows *inj-on* g *UNIV* *<proof>*

lemma *lm025*:

assumes *inj-on* f X

shows *inj-on* (*image* f) (*Pow* X)

<proof>

lemma *injectionPowerset*:

assumes *inj-on* f Y $X \subseteq Y$

shows *inj-on* (*image* f) (*Pow* X)

<proof>

definition *finestpart*

where *finestpart* $X = (\%x. \{x\}) \text{ ` } X$

lemma *finestPart*:

finestpart $X = \{\{x\} \mid x \in X\}$

<proof>

lemma *finestPartUnion*:

$X = \bigcup (\textit{finestpart } X)$

<proof>

lemma *lm026*:

$\textit{Union} \circ \textit{finestpart} = id$

<proof>

lemma *lm027*:

inj-on *Union* (*finestpart* ` *UNIV*)

<proof>

lemma *nonEqualitySetOfSets*:

assumes $X \neq Y$

shows $\{\{x\} \mid x \in X\} \neq \{\{x\} \mid x \in Y\}$

<proof>

corollary *lm028*:

inj-on *finestpart* *UNIV*

<proof>

lemma *unionFinestPart*:

$\{Y \mid Y. \exists x. ((Y \in \textit{finestpart } x) \wedge (x \in X))\} = \bigcup (\textit{finestpart } X)$

<proof>

lemma *rangeSetOfPairs:*

$\text{Range } \{(fst\ pair, Y) \mid Y\ pair. Y \in finestpart\ (snd\ pair) \ \&\ pair \in X\} =$
 $\{Y. \exists x. ((Y \in finestpart\ x) \wedge (x \in Range\ X))\}$
<proof>

lemma *setOfPairsEquality:*

$\{(fst\ pair, \{y\}) \mid y\ pair. y \in snd\ pair \ \&\ pair \in X\} =$
 $\{(fst\ pair, Y) \mid Y\ pair. Y \in finestpart\ (snd\ pair) \ \&\ pair \in X\}$
<proof>

lemma *setOfPairs:*

$\{(fst\ pair, \{y\}) \mid y. y \in snd\ pair\} =$
 $\{fst\ pair\} \times \{\{y\} \mid y. y \in snd\ pair\}$
<proof>

lemma *lm029:*

$x \in X = (\{x\} \in finestpart\ X)$
<proof>

lemma *pairDifference:*

$\{(x, X)\} - \{(x, Y)\} = \{x\} \times (\{X\} - \{Y\})$
<proof>

lemma *lm030:*

assumes $\bigcup P = X$
shows $P \subseteq Pow\ X$
<proof>

lemma *lm031:*

$argmax\ f\ \{x\} = \{x\}$
<proof>

lemma *sortingSameSet:*

assumes *finite* X
shows $set\ (sorted\ list\ of\ set\ X) = X$
<proof>

lemma *lm032:*

assumes *finite* A
shows $sum\ f\ A = sum\ f\ (A \cap B) + sum\ f\ (A - B)$
<proof>

corollary *sumOutside:*

assumes *finite* g

shows $sum\ f\ g = sum\ f\ (g\ outside\ X) + (sum\ f\ (g||X))$
 ⟨proof⟩

lemma *lm033*:
assumes $(Domain\ P \subseteq Domain\ Q)$
shows $(P\ +*\ Q) = Q$
 ⟨proof⟩

lemma *lm034*:
assumes $(P\ +*\ Q = Q)$
shows $(Domain\ P \subseteq Domain\ Q)$
 ⟨proof⟩

lemma *lm035*:
 $(Domain\ P \subseteq Domain\ Q) = (P\ +*\ Q = Q)$
 ⟨proof⟩

lemma
 $(P|| (Domain\ Q))\ +*\ Q = Q$
 ⟨proof⟩

lemma *lm036*:
 $P||X = P\ outside\ (Domain\ P - X)$
 ⟨proof⟩

lemma *lm037*:
 $(P\ outside\ X) \subseteq P\ ||\ ((Domain\ P) - X)$
 ⟨proof⟩

lemma *lm038*:
 $Domain\ (P\ outside\ X) \cap Domain\ (Q\ ||\ X) = \{\}$
 ⟨proof⟩

lemma *lm039*:
 $(P\ outside\ X) \cap (Q\ ||\ X) = \{\}$
 ⟨proof⟩

lemma *lm040*:
 $(P\ outside\ (X \cup Y)) \cap (Q\ ||\ X) = \{\}$ & $(P\ outside\ X) \cap (Q\ ||\ (X \cap Z)) = \{\}$
 ⟨proof⟩

lemma *lm041*:
 $P\ outside\ X = P\ ||\ ((Domain\ P) - X)$
 ⟨proof⟩

lemma *lm042*:
 $R\ \text{``}(X - Y) = (R||X)\ \text{``}(X - Y)$
 ⟨proof⟩

lemma *lm043*:

assumes $\bigcup XX \subseteq X \ x \in XX \ x \neq \{\}$
shows $x \cap X \neq \{\}$
<proof>

lemma *lm044*:

assumes $\forall l \in \text{set } L1. \text{ set } L2 = f2 (\text{set } l) N$
shows $\text{set } [\text{set } L2. l \leftarrow L1] = \{f2 P N \mid P. P \in \text{set } (\text{map set } L1)\}$
<proof>

lemma *setVsList*:

assumes $\forall l \in \text{set } (g1 G). \text{ set } (g2 l N) = f2 (\text{set } l) N$
shows $\text{set } [\text{set } (g2 l N). l \leftarrow (g1 G)] = \{f2 P N \mid P. P \in \text{set } (\text{map set } (g1 G))\}$
<proof>

lemma *lm045*:

$(\forall l \in \text{set } (g1 G). \text{ set } (g2 l N) = f2 (\text{set } l) N) \longrightarrow$
 $\{f2 P N \mid P. P \in \text{set } (\text{map set } (g1 G))\} = \text{set } [\text{set } (g2 l N). l \leftarrow g1 G]$
<proof>

lemma *lm046*:

assumes $X \cap Y = \{\}$
shows $R \text{ `` } X = (R \text{ outside } Y) \text{ `` } X$
<proof>

lemma *lm047*:

assumes $(\text{Range } P) \cap (\text{Range } Q) = \{\}$ *runiq* $(P \hat{-} 1)$ *runiq* $(Q \hat{-} 1)$
shows *runiq* $((P \cup Q) \hat{-} 1)$
<proof>

lemma *lm048*:

assumes $(\text{Range } P) \cap (\text{Range } Q) = \{\}$ *runiq* $(P \hat{-} 1)$ *runiq* $(Q \hat{-} 1)$
shows *runiq* $((P +* Q) \hat{-} 1)$
<proof>

lemma *lm049*:

assumes *runiq* R
shows $\text{card } (R \text{ `` } \{a\}) = 1 \longleftrightarrow a \in \text{Domain } R$
<proof>

lemma *lm050*:

inj $(\lambda a. ((\text{fst } a, \text{fst } (\text{snd } a)), \text{snd } (\text{snd } a)))$
<proof>

lemma *lm051*:

assumes *finite* X $x > \text{Max } X$

shows $x \notin X$

<proof>

lemma *lm052*:

assumes *finite* A $A \neq \{\}$

shows $\text{Max } (f' A) \in f' A$

<proof>

lemma *lm053*:

$\text{argmax } f A \subseteq f^{-1} \{ \text{Max } (f' A) \}$

<proof>

lemma *lm054*:

$\text{argmax } f A = A \cap \{ x . f x = \text{Max } (f' A) \}$

<proof>

lemma *lm055*:

$(x \in \text{argmax } f X) = (x \in X \ \& \ f x = \text{Max } (f' X))$

<proof>

lemma *rangeEmpty*:

$\text{Range } -' \{\{\}\} = \{\{\}\}$

<proof>

lemma *finitePairSecondRange*:

$(\forall \text{ pair} \in R. \text{finite } (\text{snd } \text{pair})) = (\forall y \in \text{Range } R. \text{finite } y)$

<proof>

lemma *lm056*:

$\text{fst } ' P = \text{snd } ' (P \hat{-} 1)$

<proof>

lemma *lm057*:

$\text{fst } \text{pair} = \text{snd } (\text{flip } \text{pair}) \ \& \ \text{snd } \text{pair} = \text{fst } (\text{flip } \text{pair})$

<proof>

lemma *flip-flip2*:

$\text{flip} \circ \text{flip} = \text{id}$

<proof>

lemma *lm058*:

$\text{fst} = (\text{snd} \circ \text{flip})$

<proof>

lemma *lm059*:

$snd = (fst \circ flip)$
 $\langle proof \rangle$

lemma *lm060*:
 $inj\text{-}on\ fst\ P = inj\text{-}on\ (snd \circ flip)\ P$
 $\langle proof \rangle$

lemma *lm062*:
 $inj\text{-}on\ fst\ P = inj\text{-}on\ snd\ (P^{-1})$
 $\langle proof \rangle$

lemma *sumPairsInverse*:
assumes $runiq\ (P^{-1})$
shows $sum\ (f \circ snd)\ P = sum\ f\ (Range\ P)$
 $\langle proof \rangle$

lemma *notEmptyFinestpart*:
assumes $X \neq \{\}$
shows $finestpart\ X \neq \{\}$
 $\langle proof \rangle$

lemma *lm063*:
assumes $inj\text{-}on\ g\ X$
shows $sum\ f\ (g'X) = sum\ (f \circ g)\ X$
 $\langle proof \rangle$

lemma *functionOnFirstEqualsSecond*:
assumes $runiq\ R\ z \in R$
shows $R.,(fst\ z) = snd\ z$
 $\langle proof \rangle$

lemma *lm064*:
assumes $runiq\ R$
shows $sum\ (toFunction\ R)\ (Domain\ R) = sum\ snd\ R$
 $\langle proof \rangle$

corollary *lm065*:
assumes $runiq\ (f||X)$
shows $sum\ (toFunction\ (f||X))\ (X \cap Domain\ f) = sum\ snd\ (f||X)$
 $\langle proof \rangle$

lemma *lm066*:
 $Range\ (R\ outside\ X) = R^{-1}((Domain\ R) - X)$
 $\langle proof \rangle$

lemma *lm067*:
 $(R||X)^{-1}X = R^{-1}X$
 $\langle proof \rangle$

lemma *lm068*:

assumes $x \in \text{Domain } (f||X)$

shows $(f||X) \{x\} = f \{x\}$

<proof>

lemma *lm069*:

assumes $x \in X \cap \text{Domain } f$ *runiq* $(f||X)$

shows $(f||X) x = f x$

<proof>

lemma *lm070*:

assumes *runiq* $(f||X)$

shows $\text{sum } (\text{toFunction } (f||X)) (X \cap \text{Domain } f) = \text{sum } (\text{toFunction } f) (X \cap \text{Domain } f)$

<proof>

corollary *sumRestrictedToDomainInvariant*:

assumes *runiq* $(f||X)$

shows $\text{sum } (\text{toFunction } f) (X \cap \text{Domain } f) = \text{sum } \text{snd } (f||X)$

<proof>

corollary *sumRestrictedOnFunction*:

assumes *runiq* $(f||X)$

shows $\text{sum } (\text{toFunction } (f||X)) (X \cap \text{Domain } f) = \text{sum } \text{snd } (f||X)$

<proof>

lemma *cardFinestpart*:

$\text{card } (\text{finestpart } X) = \text{card } X$

<proof>

corollary *lm071*:

$\text{finestpart } \{\} = \{\}$ & $\text{card } \circ \text{finestpart} = \text{card}$

<proof>

lemma *finiteFinestpart*:

$\text{finite } (\text{finestpart } X) = \text{finite } X$

<proof>

lemma *lm072*:

$\text{finite } \circ \text{finestpart} = \text{finite}$

<proof>

lemma *finestpartSubset*:

assumes $X \subseteq Y$

shows $\text{finestpart } X \subseteq \text{finestpart } Y$

<proof>

corollary *lm073*:

assumes $x \in X$

shows $\text{finestpart } x \subseteq \text{finestpart } (\bigcup X)$
(proof)

lemma *lm074*:
 $\bigcup (\text{finestpart } ' XX) \subseteq \text{finestpart } (\bigcup XX)$
(proof)

lemma *lm075*:
 $\bigcup (\text{finestpart } ' XX) \supseteq \text{finestpart } (\bigcup XX)$
(is ?L \supseteq ?R)
(proof)

corollary *commuteUnionFinestpart*:
 $\bigcup (\text{finestpart } ' XX) = \text{finestpart } (\bigcup XX)$
(proof)

lemma *unionImage*:
assumes *runiq a*
shows $\{(x, \{y\}) \mid x \ y. y \in \bigcup (a \{x\}) \ \& \ x \in \text{Domain } a\} =$
 $\{(x, \{y\}) \mid x \ y. y \in a, x \ \& \ x \in \text{Domain } a\}$
(proof)

lemma *lm076*:
assumes *runiq P*
shows $\text{card } (\text{Domain } P) = \text{card } P$
(proof)

lemma *finiteDomainImpliesFinite*:
assumes *runiq f*
shows $\text{finite } (\text{Domain } f) = \text{finite } f$
(proof)

lemma *sumCurry*:
 $\text{sum } ((\text{curry } f) x) Y = \text{sum } f (\{x\} \times Y)$
(proof)

lemma *lm077*:
 $\text{sum } (\%y. f (x,y)) Y = \text{sum } f (\{x\} \times Y)$
(proof)

corollary *lm078*:
assumes *finite X*
shows $\text{sum } f X = \text{sum } f (X - Y) + (\text{sum } f (X \cap Y))$
(proof)

lemma *lm079*:
 $(P +* Q) \text{``} (\text{Domain } Q \cap X) = Q \text{``} (\text{Domain } Q \cap X)$
(proof)

corollary *lm080*:

$(P +* Q) \text{ ``}(X \cap (\text{Domain } Q)) = Q \text{ ``} X$
<proof>

corollary *lm081*:

assumes $X \cap (\text{Domain } Q) = \{\}$
shows $(P +* Q) \text{ ``} X = (P \text{ outside } (\text{Domain } Q)) \text{ ``} X$
<proof>

lemma *lm082*:

assumes $X \cap Y = \{\}$
shows $(P \text{ outside } Y) \text{ ``} X = P \text{ ``} X$
<proof>

corollary *lm083*:

assumes $X \cap (\text{Domain } Q) = \{\}$
shows $(P +* Q) \text{ ``} X = P \text{ ``} X$
<proof>

lemma *lm084*:

assumes *finite* X *finite* Y $\text{card}(X \cap Y) = \text{card } X$
shows $X \subseteq Y$
<proof>

lemma *cardinalityIntersectionEquality*:

assumes *finite* X *finite* Y $\text{card } X = \text{card } Y$
shows $(\text{card } (X \cap Y) = \text{card } X) \implies (X = Y)$
<proof>

lemma *lm085*:

assumes $P \ x x$
shows $\{(x, f \ x) \mid x. P \ x\}, x x = f \ x x$
<proof>

lemma *graphEqImage*:

assumes $x \in X$
shows $\text{graph } X \ f, x = f \ x$
<proof>

lemma *lm086*:

$\text{Graph } f, x = f \ x$
<proof>

lemma *lm087*:

toFunction $(\text{Graph } f) = f$ (**is** ?L=-)
<proof>

lemma *lm088*:

$R \text{ outside } X \subseteq R$
(proof)

lemma *lm089*:
 $\text{Range}(f \text{ outside } X) \supseteq (\text{Range } f) - (f''X)$
(proof)

lemma *lm090*:
assumes *runiq* P
shows $(P^{-1}''((\text{Range } P) - Y)) \cap ((P^{-1})''Y) = \{\}$
(proof)

lemma *lm091*:
assumes *runiq* (P^{-1})
shows $(P''((\text{Domain } P) - X)) \cap (P''X) = \{\}$
(proof)

lemma *lm092*:
assumes *runiq* f *runiq* (f^{-1})
shows $\text{Range}(f \text{ outside } X) \subseteq (\text{Range } f) - (f''X)$
(proof)

lemma *rangeOutside*:
assumes *runiq* f *runiq* (f^{-1})
shows $\text{Range}(f \text{ outside } X) = (\text{Range } f) - (f''X)$
(proof)

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

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

lemma *lm093*:
assumes $R \neq \{\}$ $\text{Domain } R \cap X \neq \{\}$
shows $R''X \neq \{\}$
(proof)

lemma *lm095*:
 $R \subseteq (\text{Domain } R) \times (\text{Range } R)$
(proof)

lemma *finiteRelationCharacterization*:
 $(\text{finite } (\text{Domain } Q) \ \& \ \text{finite } (\text{Range } Q)) = \text{finite } Q$
(proof)

lemma *familyUnionFiniteEverySetFinite:*

assumes *finite* $(\bigcup XX)$

shows $\forall X \in XX. \text{finite } X$

<proof>

lemma *lm096:*

assumes *runiq* $f X \subseteq (f^{-1})^{\ast} Y$

shows $f^{\ast} X \subseteq Y$

<proof>

lemma *lm097:*

assumes $y \in f^{\ast}\{x\}$ *runiq* f

shows $f^{-1}x = y$

<proof>

7.3 Indicator function in set-theoretical form.

abbreviation

Outside' X f == f outside X

abbreviation

*Chi X Y == (Y \times {0::nat}) $+$ * (X \times {1})*

notation *Chi* (**infix** $<||$ 80)

abbreviation

chii X Y == toFunction (X $<||$ Y)

notation *chii* (**infix** $<|$ 80)

abbreviation

chi X == indicator X

lemma *lm098:*

runiq (X $<||$ Y)

<proof>

lemma *lm099:*

assumes $x \in X$

shows $1 \in (X <|| Y) \text{ `` } \{x\}$

<proof>

lemma *lm100:*

assumes $x \in Y - X$

shows $0 \in (X <|| Y) \text{ `` } \{x\}$

<proof>

lemma *lm101:*

assumes $x \in X \cup Y$

shows $(X <|| Y),,x = \text{chi } X x$ (**is** $?L=?R$)

<proof>

lemma *lm102*:

assumes $x \in X \cup Y$

shows $(X <| Y) x = \text{chi } X x$

<proof>

corollary *lm103*:

$\text{sum } (X <| Y) (X \cup Y) = \text{sum } (\text{chi } X) (X \cup Y)$

<proof>

corollary *lm104*:

assumes $\forall x \in X. f x = g x$

shows $\text{sum } f X = \text{sum } g X$

<proof>

corollary *lm105*:

assumes $\forall x \in X. f x = g x \ Y \subseteq X$

shows $\text{sum } f Y = \text{sum } g Y$

<proof>

corollary *lm106*:

assumes $Z \subseteq X \cup Y$

shows $\text{sum } (X <| Y) Z = \text{sum } (\text{chi } X) Z$

<proof>

corollary *lm107*:

$\text{sum } (\text{chi } X) (Z - X) = 0$

<proof>

corollary *lm108*:

assumes $Z \subseteq X \cup Y$

shows $\text{sum } (X <| Y) (Z - X) = 0$

<proof>

corollary *lm109*:

assumes *finite* Z

shows $\text{sum } (X <| Y) Z = \text{sum } (X <| Y) (Z - X) + (\text{sum } (X <| Y) (Z \cap X))$

<proof>

corollary *lm110*:

assumes $Z \subseteq X \cup Y$ *finite* Z

shows $\text{sum } (X <| Y) Z = \text{sum } (X <| Y) (Z \cap X)$

<proof>

corollary *lm111*:

assumes *finite* Z

shows $\text{sum } (\text{chi } X) Z = \text{card } (X \cap Z)$

<proof>

corollary *lm112:*

assumes $Z \subseteq X \cup Y$ *finite* Z

shows $\text{sum } (X <| Y) Z = \text{card } (Z \cap X)$

<proof>

corollary *subsetCardinality:*

assumes $Z \subseteq X \cup Y$ *finite* Z

shows $(\text{sum } (X <| Y) X) - (\text{sum } (X <| Y) Z) = \text{card } X - \text{card } (Z \cap X)$

<proof>

corollary *differenceSumVsCardinality:*

assumes $Z \subseteq X \cup Y$ *finite* Z

shows $\text{int } (\text{sum } (X <| Y) X) - \text{int } (\text{sum } (X <| Y) Z) = \text{int } (\text{card } X) - \text{int } (\text{card } (Z \cap X))$

<proof>

lemma *lm113:*

$\text{int } (n::\text{nat}) = \text{real } n$

<proof>

corollary *differenceSumVsCardinalityReal:*

assumes $Z \subseteq X \cup Y$ *finite* Z

shows $\text{real } (\text{sum } (X <| Y) X) - \text{real } (\text{sum } (X <| Y) Z) = \text{real } (\text{card } X) - \text{real } (\text{card } (Z \cap X))$

<proof>

7.4 Lists

lemma *lm114:*

assumes $\exists n \in \{0..<\text{size } l\}. P (!n)$

shows $[n. n \leftarrow [0..<\text{size } l], P (!n)] \neq []$

<proof>

lemma *lm115:*

assumes $ll \in \text{set } (l::'a \text{ list})$

shows $\exists n \in (\text{nth } l) - '(\text{set } l). ll=!n$

<proof>

lemma *lm116:*

assumes $ll \in \text{set } (l::'a \text{ list})$

shows $\exists n. ll=!n \ \& \ n < \text{size } l \ \& \ n \geq 0$

<proof>

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

lemma *nonEmptyListFiltered*:
assumes $P - \{True\} \cap \text{set } l \neq \{\}$
shows $[n. n \leftarrow [0..<\text{size } l], P (l!n)] \neq []$
 $\langle \text{proof} \rangle$

lemma *lm118*:
 $(\text{nth } l) \text{ 'set } ([n. n \leftarrow [0..<\text{size } l], (\%x. x \in X) (l!n)]) \subseteq X \cap \text{set } l$
 $\langle \text{proof} \rangle$

corollary *lm119*:
 $(\text{nth } l) \text{ 'set } (\text{filterpositions2 } (\%x.(x \in X)) l) \subseteq X \cap \text{set } l$
 $\langle \text{proof} \rangle$

lemma *lm120*:
 $(n \in \{0..<N\}) = ((n::\text{nat}) < N)$
 $\langle \text{proof} \rangle$

lemma *lm121*:
assumes $X \subseteq \{0..<\text{size } \text{list}\}$
shows $(\text{nth } \text{list}) \text{ 'X } \subseteq \text{set } \text{list}$
 $\langle \text{proof} \rangle$

lemma *lm122*:
 $\text{set } ([n. n \leftarrow [0..<\text{size } l], P (l!n)]) \subseteq \{0..<\text{size } l\}$
 $\langle \text{proof} \rangle$

lemma *lm123*:
 $\text{set } (\text{filterpositions2 } \text{pre } \text{list}) \subseteq \{0..<\text{size } \text{list}\}$
 $\langle \text{proof} \rangle$

7.5 Computing all the permutations of a list

abbreviation

$\text{rotateLeft} == \text{rotate}$

abbreviation

$\text{rotateRight } n \ l == \text{rotateLeft } (\text{size } l - (n \bmod (\text{size } l))) \ l$

abbreviation

$$\text{insertAt } x \ l \ n == \text{rotateRight } n \ (x\#(\text{rotateLeft } n \ l))$$
fun perm2 where

$$\begin{aligned} \text{perm2 } [] &= (\%n. []) \mid \\ \text{perm2 } (x\#l) &= (\%n. \text{insertAt } x \ ((\text{perm2 } l) \ (n \ \text{div} \ (1+\text{size } l))) \\ &\quad (n \ \text{mod} \ (1+\text{size } l))) \end{aligned}$$
abbreviation

$$\text{takeAll } P \ \text{list} == \text{map } (\text{nth } \text{list}) \ (\text{filterpositions2 } P \ \text{list})$$
lemma permutationNotEmpty:

$$\begin{aligned} &\text{assumes } l \neq [] \\ &\text{shows } \text{perm2 } l \ n \neq [] \\ &\langle \text{proof} \rangle \end{aligned}$$
lemma lm124:

$$\begin{aligned} \text{set } (\text{takeAll } P \ \text{list}) &= ((\text{nth } \text{list}) \text{ ' set } (\text{filterpositions2 } P \ \text{list})) \\ &\langle \text{proof} \rangle \end{aligned}$$
corollary listIntersectionWithSet:

$$\begin{aligned} \text{set } (\text{takeAll } (\%x.(x \in X)) \ l) &\subseteq (X \cap \text{set } l) \\ &\langle \text{proof} \rangle \end{aligned}$$
corollary lm125:

$$\begin{aligned} \text{set } (\text{takeAll } P \ \text{list}) &\subseteq \text{set } \text{list} \\ &\langle \text{proof} \rangle \end{aligned}$$
lemma takeAllSubset:

$$\begin{aligned} \text{set } (\text{takeAll } (\%x. x \in P) \ \text{list}) &\subseteq P \\ &\langle \text{proof} \rangle \end{aligned}$$
lemma lm126:

$$\begin{aligned} \text{set } (\text{insertAt } x \ l \ n) &= \{x\} \cup \text{set } l \\ &\langle \text{proof} \rangle \end{aligned}$$
lemma lm127:

$$\begin{aligned} \forall n. \text{set } (\text{perm2 } [] \ n) &= \text{set } [] \\ &\langle \text{proof} \rangle \end{aligned}$$
lemma lm128:

$$\begin{aligned} &\text{assumes } \forall n. (\text{set } (\text{perm2 } l \ n) = \text{set } l) \\ &\text{shows } \text{set } (\text{perm2 } (x\#l) \ n) = \{x\} \cup \text{set } l \\ &\langle \text{proof} \rangle \end{aligned}$$

corollary *permutationInvariance*:
 $\forall n. \text{set } (\text{perm2 } (l::'a \text{ list}) n) = \text{set } l$
 ⟨proof⟩

corollary *takeAllPermutation*:
 $\text{set } (\text{perm2 } (\text{takeAll } (\%x.(x \in X)) l) n) \subseteq X \cap \text{set } l$
 ⟨proof⟩

abbreviation *subList* $l \ xl == \text{map } (\text{nth } l) (\text{takeAll } (\%x. x \leq \text{size } l) \ xl)$

7.6 A more computable version of *toFunction*.

abbreviation *toFunctionWithFallback* $R \ \text{fallback} ==$
 $(\% x. \text{if } (R \ \{x\} = \{R, x\}) \text{ then } (R, x) \text{ else } \text{fallback})$

notation
toFunctionWithFallback (**infix** *Else* 75)

abbreviation *sum'* **where**
 $\text{sum}' R X == \text{sum } (R \ \text{Else } 0) X$

lemma *lm129*:
assumes $\text{runiq } f \ x \in \text{Domain } f$
shows $(f \ \text{Else } 0) \ x = (\text{toFunction } f) \ x$
 ⟨proof⟩

lemma *lm130*:
assumes $\text{runiq } f$
shows $\text{sum } (f \ \text{Else } 0) (X \cap (\text{Domain } f)) = \text{sum } (\text{toFunction } f) (X \cap (\text{Domain } f))$
 ⟨proof⟩

lemma *lm131*:
assumes $Y \subseteq f - \{0\}$
shows $\text{sum } f \ Y = 0$
 ⟨proof⟩

lemma *lm132*:
assumes $Y \subseteq f - \{0\}$ *finite* X
shows $\text{sum } f \ X = \text{sum } f \ (X - Y)$
 ⟨proof⟩

lemma *lm133*:
 $-(\text{Domain } f) \subseteq (f \ \text{Else } 0) - \{0\}$
 ⟨proof⟩

corollary *lm134*:

assumes *finite X*
shows $\text{sum } (f \text{ Else } 0) X = \text{sum } (f \text{ Else } 0) (X \cap \text{Domain } f)$
 $\langle \text{proof} \rangle$

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)
 $\langle \text{proof} \rangle$

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)
 $\langle \text{proof} \rangle$

lemma *lm137:*

$\text{sum } (f \text{ Else } 0) X = \text{sum}' f X$
 $\langle \text{proof} \rangle$

corollary *lm138:*

assumes *finite X runiq f*
shows $\text{sum } (\text{toFunction } f) (X \cap \text{Domain } f) = \text{sum}' f X$
 $\langle \text{proof} \rangle$

lemma *lm139:*

$\text{argmax } (\text{sum}' b) = (\text{argmax} \circ \text{sum}') b$
 $\langle \text{proof} \rangle$

lemma *domainConstant:*

$\text{Domain } (Y \times \{0::\text{nat}\}) = Y \ \& \ \text{Domain } (X \times \{1\}) = X$
 $\langle \text{proof} \rangle$

lemma *domainCharacteristicFunction:*

$\text{Domain } (X <|| Y) = X \cup Y$
 $\langle \text{proof} \rangle$

lemma *functionEquivalenceOnSets:*

assumes $\forall x \in X. f x = g x$
shows $f'X = g'X$
 $\langle \text{proof} \rangle$

7.7 Cardinalities of sets.

lemma *lm140:*

assumes *runiq R runiq (R⁻¹)*
shows $(R' A) \cap (R' B) = R'(A \cap B)$
 $\langle \text{proof} \rangle$

lemma *intersectionEmptyRelationIntersectionEmpty*:

assumes *runiq* (R^{-1}) *runiq* R $X1 \cap X2 = \{\}$

shows $(R \circ X1) \cap (R \circ X2) = \{\}$

<proof>

lemma *lm141*:

assumes *runiq* f *trivial* Y

shows *trivial* $(f \circ (f^{-1} \circ Y))$

<proof>

lemma *lm142*:

assumes *trivial* X

shows $\text{card } (\text{Pow } X) \in \{1, 2\}$

<proof>

lemma *lm143*:

assumes $\text{card } (\text{Pow } A) = 1$

shows $A = \{\}$

<proof>

lemma *lm144*:

$(\neg (\text{finite } A)) = (\text{card } (\text{Pow } A) = 0)$

<proof>

corollary *lm145*:

$(\text{finite } A) = (\text{card } (\text{Pow } A) \neq 0)$

<proof>

lemma *lm146*:

assumes $\text{card } (\text{Pow } A) \neq 0$

shows $\text{card } A = \text{Discrete.log } (\text{card } (\text{Pow } A))$

<proof>

lemma *log-2 [simp]*:

$\text{Discrete.log } 2 = 1$

<proof>

lemma *lm147*:

assumes $\text{card } (\text{Pow } A) = 2$

shows $\text{card } A = 1$

<proof>

lemma *lm148*:

assumes $\text{card } (\text{Pow } X) = 1 \vee \text{card } (\text{Pow } X) = 2$

shows *trivial* X

<proof>

lemma *lm149*:

trivial $A = (\text{card } (\text{Pow } A) \in \{1,2\})$
(*proof*)

lemma *lm150*:
assumes $R \subseteq f \text{ runiq } f \text{ Domain } f = \text{Domain } R$
shows $\text{runiq } R$
(*proof*)

lemma *lm151*:
assumes $f \subseteq g \text{ runiq } g \text{ Domain } f = \text{Domain } g$
shows $g \subseteq f$
(*proof*)

lemma *lm152*:
assumes $R \subseteq f \text{ runiq } f \text{ Domain } f \subseteq \text{Domain } R$
shows $f = R$
(*proof*)

lemma *lm153*:
 $\text{graph } X f = (\text{Graph } f) \parallel X$
(*proof*)

lemma *lm154*:
 $\text{graph } (X \cap Y) f = (\text{graph } X f) \parallel Y$
(*proof*)

lemma *restrictionVsIntersection*:
 $\{(x, f x) \mid x. x \in X2\} \parallel X1 = \{(x, f x) \mid x. x \in X2 \cap X1\}$
(*proof*)

lemma *lm155*:
assumes $\text{runiq } f X \subseteq \text{Domain } f$
shows $\text{graph } X (\text{toFunction } f) = (f \parallel X)$
(*proof*)

lemma *lm156*:
 $(\text{Graph } f) \text{ `` } X = f \text{ ' } X$
(*proof*)

lemma *lm157*:
assumes $X \subseteq \text{Domain } f \text{ runiq } f$
shows $f \text{ `` } X = (\text{eval-rel } f) \text{ ' } X$
(*proof*)

lemma *cardOneImageCardOne*:
assumes $\text{card } A = 1$
shows $\text{card } (f \text{ ' } A) = 1$
(*proof*)

lemma *cardOneTheElem*:
assumes *card A = 1*
shows *the-elem (f'A) = f (the-elem A)*
<proof>

abbreviation
swap f == curry ((case-prod f) o flip)

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

lemma *lm159*:
finite = (%X. X ∈ range set)
<proof>

lemma *lm160*:
swap f = (%x. %y. f y x)
<proof>

7.8 Some easy properties on real numbers

lemma *lm161*:
fixes *a::real*
fixes *b c*
shows *a*b - a*c = a*(b-c)*
<proof>

lemma *lm162*:
fixes *a::real*
fixes *b c*
shows *a*b - c*b = (a-c)*b*
<proof>

end

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

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

begin

8.1 Types

type-synonym *index* = *integer*
type-synonym *participant* = *index*
type-synonym *good* = *integer*
type-synonym *goods* = *good set*
type-synonym *price* = *real*
type-synonym *bids3* = $((\text{participant} \times \text{goods}) \times \text{price}) \text{ set}$
type-synonym *bids* = $\text{participant} \Rightarrow \text{goods} \Rightarrow \text{price}$
type-synonym *allocation-rel* = $(\text{goods} \times \text{participant}) \text{ set}$
type-synonym *allocation* = $(\text{participant} \times \text{goods}) \text{ set}$
type-synonym *payments* = $\text{participant} \Rightarrow \text{price}$
type-synonym *bidvector* = $(\text{participant} \times \text{goods}) \Rightarrow \text{price}$
abbreviation *bidvector* (*b::bids*) == *case-prod b*
abbreviation *proceeds* (*b::bidvector*) (*allo::allocation*) == *sum b allo*
abbreviation *winnersOfAllo* (*a::allocation*) == *Domain a*
abbreviation *allocatedGoods* (*allo::allocation*) == $\bigcup (\text{Range } \text{allo})$

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

abbreviation *is-partition-of'* *P A* == $(\bigcup P = A \wedge \text{is-non-overlapping } P)$
abbreviation *all-partitions'* *A* == $\{P . \text{is-partition-of}' P A\}$

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

abbreviation *allAllocations where*
 allAllocations N G == *converse ' (possible-allocations-rel G N)*

algorithmic version of *possible-allocations-rel*

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

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

8.2 VCG mechanism

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

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

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

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

payments

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

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

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

abbreviation $\text{remainingValueRel } N \ G \ t \ b \ n == \text{sum } b \ ((\text{winningAllocationRel } N \ G \ t \ b) \ -- \ n)$

abbreviation $\text{remainingValueAlg } N \ G \ t \ b \ n == \text{proceeds } b \ ((\text{winningAllocationAlg } N \ G \ t \ b) \ -- \ n)$

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

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

end

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

theory *Universes*

imports

HOL-Library.Code-Target-Nat

StrictCombinatorialAuction

HOL-Library.Indicator-Function

begin

9.1 Preliminary lemmas

lemma *lm001*:

assumes $Y \in \text{set } (\text{all-partitions-alg } X)$

shows *distinct* Y

<proof>

lemma *lm002*:

assumes *finite* G

shows *all-partitions* $G = \text{set } '(\text{set } (\text{all-partitions-alg } G))$

<proof>

9.2 Definitions of various subsets of *UNIV*.

abbreviation *isChoice* $R == \forall x. R \{x\} \subseteq x$

abbreviation *partitionsUniverse* $== \{X. \text{is-non-overlapping } X\}$

lemma *partitionsUniverse* $\subseteq \text{Pow } \text{UNIV}$

<proof>

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

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

<proof>

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

abbreviation *allocationsUniverse* $== \text{injectionsUniverse} \cap \text{partitionValuedUniverse}$

abbreviation *totalRels* $X \ Y == \{R. \text{Domain } R = X \ \& \ \text{Range } R \subseteq Y\}$

9.3 Results about the sets defined in the previous section

lemma *lm003*:

assumes $\forall x1 \in X. (x1 \neq \{\}) \ \& \ (\forall x2 \in X - \{x1\}. x1 \cap x2 = \{\})$

shows *is-non-overlapping* X

<proof>

lemma *lm004*:

assumes $\forall x \in X. f \ x \in x$

shows *isChoice* $(\text{graph } X \ f)$

<proof>

lemma *lm006*: *injections* $X \ Y \subseteq \text{injectionsUniverse}$

<proof>

lemma *lm007*: *injections* $X \ Y \subseteq \text{injectionsUniverse}$

<proof>

lemma *lm008*: $\text{injections } X \ Y = \text{totalRels } X \ Y \cap \text{injectionsUniverse}$
<proof>

lemma *allocationInverseRangeDomainProperty*:
assumes $a \in \text{allAllocations } N \ G$
shows $a^{-1} \in \text{injections } (\text{Range } a) \ N$ &
 $(\text{Range } a) \text{ partitions } G$ &
 $\text{Domain } a \subseteq N$
<proof>

lemma *lm009*:
assumes *is-non-overlapping* $XX \ YY \subseteq XX$
shows $(XX - YY) \text{ partitions } (\bigcup XX - \bigcup YY)$
<proof>

lemma *allocationRightUniqueRangeDomain*:
assumes $a \in \text{possible-allocations-rel } G \ N$
shows *runiq* a &
runiq (a^{-1}) &
 $(\text{Domain } a) \text{ partitions } G$ &
 $\text{Range } a \subseteq N$
<proof>

lemma *lm010*:
assumes *runiq* a *runiq* (a^{-1}) $(\text{Domain } a) \text{ partitions } G$ $\text{Range } a \subseteq N$
shows $a \in \text{possible-allocations-rel } G \ N$
<proof>

lemma *allocationProperty*:
 $a \in \text{possible-allocations-rel } G \ N \longleftrightarrow$
runiq a & *runiq* (a^{-1}) & $(\text{Domain } a) \text{ partitions } G$ & $\text{Range } a \subseteq N$
<proof>

lemma *lm011*:
 $\text{possible-allocations-rel}' \ G \ N \subseteq \text{injectionsUniverse}$
<proof>

lemma *lm012*:
 $\text{possible-allocations-rel } G \ N \subseteq \{a. (\text{Range } a) \subseteq N \ \& \ (\text{Domain } a) \in \text{all-partitions } G\}$
<proof>

lemma *lm013*:
 $\text{injections } X \ Y = \text{injections } X \ Y$
<proof>

lemma *lm014*:

all-partitions $X = \text{all-partitions}' X$
(proof)

lemma *lm015*:
possible-allocations-rel' $A B = \text{possible-allocations-rel } A B$
(is ?A=?B)
(proof)

lemma *lm016*:
possible-allocations-rel $G N \subseteq$
injectionsUniverse $\cap \{a. \text{Range } a \subseteq N \ \& \ \text{Domain } a \in \text{all-partitions } G\}$
(proof)

lemma *lm017*:
possible-allocations-rel $G N \supseteq$
injectionsUniverse $\cap \{a. \text{Domain } a \in \text{all-partitions } G \ \& \ \text{Range } a \subseteq N\}$
(proof)

lemma *lm018*:
possible-allocations-rel $G N =$
injectionsUniverse $\cap \{a. \text{Domain } a \in \text{all-partitions } G \ \& \ \text{Range } a \subseteq N\}$
(proof)

lemma *lm019*:
converse' injectionsUniverse $= \text{injectionsUniverse}$
(proof)

lemma *lm020*:
converse'(A \cap B) $= (\text{converse}'A) \cap (\text{converse}'B)$
(proof)

lemma *allocationInjectionsUniveruseProperty*:
allAllocations $N G =$
injectionsUniverse $\cap \{a. \text{Domain } a \subseteq N \ \& \ \text{Range } a \in \text{all-partitions } G\}$
(proof)

lemma *lm021*:
allAllocations $N G \subseteq \text{injectionsUniverse}$
(proof)

lemma *lm022*:
allAllocations $N G \subseteq \text{partitionValuedUniverse}$
(proof)

corollary *allAllocationsUniverse*:
allAllocations $N G \subseteq \text{allocationsUniverse}$
(proof)

corollary *possibleAllocationsRelCharacterization*:

$a \in \text{allAllocations } N \ G =$
($a \in \text{injectionsUniverse} \ \& \ \text{Domain } a \subseteq N \ \& \ \text{Range } a \in \text{all-partitions } G$)
(*proof*)

corollary *lm023*:

assumes $a \in \text{allAllocations } N1 \ G$
shows $a \in \text{allAllocations } (N1 \cup N2) \ G$
(*proof*)

corollary *lm024*:

$\text{allAllocations } N1 \ G \subseteq \text{allAllocations } (N1 \cup N2) \ G$
(*proof*)

lemma *lm025*:

assumes $(\bigcup P1) \cap (\bigcup P2) = \{\}$
is-non-overlapping P1 is-non-overlapping P2
 $X \in P1 \cup P2 \ Y \in P1 \cup P2 \ X \cap Y \neq \{\}$
shows $(X = Y)$
(*proof*)

lemma *lm026*:

assumes $(\bigcup P1) \cap (\bigcup P2) = \{\}$
is-non-overlapping P1
is-non-overlapping P2
 $X \in P1 \cup P2$
 $Y \in P1 \cup P2$
 $(X = Y)$
shows $X \cap Y \neq \{\}$
(*proof*)

lemma *lm027*:

assumes $(\bigcup P1) \cap (\bigcup P2) = \{\}$
is-non-overlapping P1
is-non-overlapping P2
shows *is-non-overlapping* $(P1 \cup P2)$
(*proof*)

lemma *lm028*:

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

lemma *lm029*:

assumes $a1 \in \text{injectionsUniverse}$
 $a2 \in \text{injectionsUniverse}$
 $(\text{Range } a1) \cap (\text{Range } a2) = \{\}$
 $(\text{Domain } a1) \cap (\text{Domain } a2) = \{\}$
shows $a1 \cup a2 \in \text{injectionsUniverse}$

<proof>

lemma *nonOverlapping*:

assumes $R \in \text{partition ValuedUniverse}$

shows *is-non-overlapping* ($\text{Range } R$)

<proof>

lemma *allocationUnion*:

assumes $a1 \in \text{allocationsUniverse}$

$a2 \in \text{allocationsUniverse}$

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

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

shows $a1 \cup a2 \in \text{allocationsUniverse}$

<proof>

lemma *lm030*:

assumes $a \in \text{injectionsUniverse}$

shows $a - b \in \text{injectionsUniverse}$

<proof>

lemma *lm031*:

$\{a. \text{Domain } a \subseteq N \ \& \ \text{Range } a \in \text{all-partitions } G\} =$
 $(\text{Domain } -'(\text{Pow } N)) \cap (\text{Range } -'(\text{all-partitions } G))$

<proof>

lemma *lm032*:

$\text{allAllocations } N \ G =$

$\text{injectionsUniverse} \cap ((\text{Range } -'(\text{all-partitions } G)) \cap (\text{Domain } -'(\text{Pow } N)))$

<proof>

corollary *lm033*:

$\text{allAllocations } N \ G =$

$\text{injectionsUniverse} \cap (\text{Range } -'(\text{all-partitions } G)) \cap (\text{Domain } -'(\text{Pow } N))$

<proof>

lemma *lm034*:

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

shows $(a^{-1} \in \text{injections } (\text{Range } a) \ N \ \&$

$\text{Range } a \in \text{all-partitions } G)$

<proof>

lemma *lm035*:

assumes $a^{-1} \in \text{injections } (\text{Range } a) \ N \ \& \ \text{Range } a \in \text{all-partitions } G$

shows $a \in \text{allAllocations } N \ G$

<proof>

lemma *allocationReverseInjective*:

$a \in \text{allAllocations } N \ G =$

$(a^{-1} \in \text{injections } (\text{Range } a) \ N \ \& \ \text{Range } a \in \text{all-partitions } G)$

$\langle \text{proof} \rangle$

lemma *lm036*:

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

shows $a \in \text{injections } (\text{Domain } a) \ (\text{Range } a) \ \&$

$\text{Range } a \in \text{all-partitions } G \ \&$

$\text{Domain } a \subseteq N$

$\langle \text{proof} \rangle$

lemma *lm037*:

assumes $a \in \text{injections } (\text{Domain } a) \ (\text{Range } a)$

$\text{Range } a \in \text{all-partitions } G$

$\text{Domain } a \subseteq N$

shows $a \in \text{allAllocations } N \ G$

$\langle \text{proof} \rangle$

lemma *characterizationalAllocations*:

$a \in \text{allAllocations } N \ G = (a \in \text{injections } (\text{Domain } a) \ (\text{Range } a) \ \&$

$\text{Range } a \in \text{all-partitions } G \ \&$

$\text{Domain } a \subseteq N)$

$\langle \text{proof} \rangle$

lemma *lm038*:

assumes $a \in \text{partitionValuedUniverse}$

shows $a - b \in \text{partitionValuedUniverse}$

$\langle \text{proof} \rangle$

lemma *reducedAllocation*:

assumes $a \in \text{allocationsUniverse}$

shows $a - b \in \text{allocationsUniverse}$

$\langle \text{proof} \rangle$

lemma *lm039*:

assumes $a \in \text{injectionsUniverse}$

shows $a \in \text{injections } (\text{Domain } a) \ (\text{Range } a)$

$\langle \text{proof} \rangle$

lemma *lm040*:

assumes $a \in \text{allocationsUniverse}$

shows $a \in \text{allAllocations } (\text{Domain } a) \ (\bigcup (\text{Range } a))$

$\langle \text{proof} \rangle$

lemma *lm041*:

$(\{X\} \in \text{partitionsUniverse}) = (X \neq \{\})$

$\langle \text{proof} \rangle$

lemma *lm042*:

$\{(x, X)\} - \{(x, \{\})\} \in \text{partitionValuedUniverse}$
 $\langle \text{proof} \rangle$

lemma *singlePairInInjectionsUniverse*:

$\{(x, X)\} \in \text{injectionsUniverse}$
 $\langle \text{proof} \rangle$

lemma *allocationUniverseProperty*:

$\{(x, X)\} - \{(x, \{\})\} \in \text{allocationsUniverse}$
 $\langle \text{proof} \rangle$

lemma *lm043*:

assumes *is-non-overlapping PP is-non-overlapping (Union PP)*
shows *is-non-overlapping (Union ‘ PP)*
 $\langle \text{proof} \rangle$

lemma *lm044*:

assumes $a \in \text{allocationsUniverse}$
shows $(a - ((X \cup \{i\}) \times (\text{Range } a))) \cup$
 $(\{(i, \bigcup (a^{“(X \cup \{i\})})\}) - \{(i, \{\})\}\}) \in \text{allocationsUniverse} \ \&$
 $\bigcup (\text{Range } ((a - ((X \cup \{i\}) \times (\text{Range } a))) \cup (\{(i, \bigcup (a^{“(X \cup \{i\})})\}) -$
 $\{(i, \{\})\}\})) =$
 $\bigcup (\text{Range } a)$
 $\langle \text{proof} \rangle$

corollary *allocationsUniverseOutsideUnion*:

assumes $a \in \text{allocationsUniverse}$
shows $(a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\{\bigcup (a^{“(X \cup \{i\})}) - \{\}\})) \in \text{allocation-}$
 $sUniverse \ \&$
 $\bigcup (\text{Range } ((a \text{ outside } (X \cup \{i\})) \cup (\{i\} \times (\{\bigcup (a^{“(X \cup \{i\})}) - \{\}\}))) =$
 $\bigcup (\text{Range } a)$
 $\langle \text{proof} \rangle$

lemma *lm045*:

assumes $\text{Domain } a \cap X \neq \{\}$ $a \in \text{allocationsUniverse}$
shows $\bigcup (a^{“X}) \neq \{\}$
 $\langle \text{proof} \rangle$

corollary *lm046*:

assumes $\text{Domain } a \cap X \neq \{\}$ $a \in \text{allocationsUniverse}$
shows $\{\bigcup (a^{“(X \cup \{i\})}) - \{\}\} = \{\bigcup (a^{“(X \cup \{i\})})\}$
 $\langle \text{proof} \rangle$

corollary *lm047*:

assumes $a \in \text{allocationsUniverse}$

$(\text{Domain } a) \cap X \neq \{\}$

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>

abbreviation

bidMonotonicity $b \ i \ ==$

$(\forall \ t \ t'. (\text{trivial } t \ \& \ \text{trivial } t' \ \& \ \text{Union } t \subseteq \text{Union } t') \longrightarrow$

$\text{sum } b \ (\{i\} \times t) \leq \text{sum } b \ (\{i\} \times t')$

lemma *lm048*:

assumes *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>

lemma *lm049*:

assumes $XX \in \text{partitionValuedUniverse}$

shows $\{\} \notin \text{Range } XX$

<proof>

corollary *emptyNotInRange*:

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

shows $\{\} \notin \text{Range } a$

<proof>

lemma *lm050*:

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

shows $\text{Range } a \subseteq \text{Pow } G$

<proof>

corollary *lm051*:

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

shows $\text{Domain } a \subseteq N \ \& \ \text{Range } a \subseteq \text{Pow } G - \{\{\}\}$

<proof>

corollary *allocationPowerset*:

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

shows $a \subseteq N \times (\text{Pow } G - \{\{\}\})$

<proof>

corollary *lm052*:

$\text{allAllocations } N \ G \subseteq \text{Pow } (N \times (\text{Pow } G - \{\{\}\}))$

<proof>

lemma *lm053*:

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

$i \in N - X$

$\text{Domain } a \cap X \neq \{\}$

shows $a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a \text{ ``}(X \cup \{i\}))\}) \in$
 $\text{allAllocations } (N - X) \ (\bigcup (\text{Range } a))$

<proof>

lemma *lm054*:

assumes $\text{bidMonotonicity } (b \text{ :- } \Rightarrow \text{ real}) \ i$

$a \in \text{allocationsUniverse}$

$\text{Domain } a \cap X \neq \{\}$

$\text{finite } a$

shows $\text{sum } b \ (a \text{ outside } X) \leq$
 $\text{sum } b \ (a \text{ outside } (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a \text{ ``}(X \cup \{i\}))\}))$

<proof>

lemma *elementOfPartitionOfFiniteSetIsFinite*:

assumes $\text{finite } X \ \text{XX} \in \text{all-partitions } X$

shows $\text{finite } \text{XX}$

<proof>

lemma *lm055*:

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

shows $\text{finite } a$

<proof>

lemma *allAllocationsFinite*:

assumes $\text{finite } N \ \text{finite } G$

shows $\text{finite } (\text{allAllocations } N \ G)$

<proof>

corollary *lm056*:

assumes $\text{bidMonotonicity } (b \text{ :- } \Rightarrow \text{ real}) \ i$

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

$i \in N - X$

$\text{Domain } a \cap X \neq \{\}$

$\text{finite } N$

$\text{finite } G$

shows $\text{Max } ((\text{sum } b) \text{ ``} (\text{allAllocations } (N - X) \ G)) \geq$
 $\text{sum } b \ (a \text{ outside } X)$

<proof>

lemma *cardinalityPreservation*:

assumes $\forall X \in \text{XX}. \ \text{finite } X \ \text{is-non-overlapping } \text{XX}$

shows $\text{card } (\bigcup \text{XX}) = \text{sum card } \text{XX}$

<proof>

corollary *cardSumCommute*:

assumes XX partitions X finite X finite XX
shows $\text{card} (\bigcup XX) = \text{sum card } XX$
<proof>

lemma *sumUnionDisjoint1*:

assumes $\forall A \in C. \text{finite } A \ \forall A \in C. \forall B \in C. A \neq B \longrightarrow A \text{ Int } B = \{\}$
shows $\text{sum } f (\text{Union } C) = \text{sum} (\text{sum } f) C$
<proof>

corollary *sumUnionDisjoint2*:

assumes $\forall x \in X. \text{finite } x$ is-non-overlapping X
shows $\text{sum } f (\bigcup X) = \text{sum} (\text{sum } f) X$
<proof>

corollary *sumUnionDisjoint3*:

assumes $\forall x \in X. \text{finite } x$ X partitions XX
shows $\text{sum } f XX = \text{sum} (\text{sum } f) X$
<proof>

corollary *sum-associativity*:

assumes $\text{finite } x$ X partitions x
shows $\text{sum } f x = \text{sum} (\text{sum } f) X$
<proof>

lemma *lm057*:

assumes $a \in \text{allocationsUniverse}$ $\text{Domain } a \subseteq N \ \bigcup (\text{Range } a) = G$
shows $a \in \text{allAllocations } N G$
<proof>

corollary *lm058*:

$(\text{allocationsUniverse} \cap \{a. (\text{Domain } a) \subseteq N \ \& \ \bigcup (\text{Range } a) = G\}) \subseteq$
 $\text{allAllocations } N G$
<proof>

corollary *lm059*:

$\text{allAllocations } N G \subseteq \{a. (\text{Domain } a) \subseteq N\}$
<proof>

corollary *lm060*:

$\text{allAllocations } N G \subseteq \{a. \bigcup (\text{Range } a) = G\}$
<proof>

corollary *lm061*:

$\text{allAllocations } N G \subseteq \text{allocationsUniverse} \ \&$
 $\text{allAllocations } N G \subseteq \{a. (\text{Domain } a) \subseteq N \ \& \ \bigcup (\text{Range } a) = G\}$
<proof>

corollary *allAllocationsIntersectionSubset:*

$allAllocations\ N\ G \subseteq$
 $allocationsUniverse \cap \{a. (Domain\ a) \subseteq N \ \& \ \bigcup (Range\ a) = G\}$
(is $?L \subseteq ?R1 \cap ?R2$)
(proof)

corollary *allAllocationsIntersection:*

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

corollary *allAllocationsIntersectionSetEquals:*

$a \in allAllocations\ N\ G =$
 $(a \in allocationsUniverse \ \& \ (Domain\ a) \subseteq N \ \& \ \bigcup (Range\ a) = G)$
(proof)

corollary *allocationsUniverseOutside:*

assumes $a \in allocationsUniverse$
shows $a\ outside\ X \in allocationsUniverse$
(proof)

9.4 Bridging theorem for injections

lemma *lm062:*

$totalRels\ \{\}\ Y = \{\{\}\}$
(proof)

lemma *lm063:*

$\{\} \in injectionsUniverse$
(proof)

lemma *lm064:*

$injectionsUniverse \cap (totalRels\ \{\}\ Y) = \{\{\}\}$
(proof)

lemma *lm065:*

assumes $runiq\ f\ x \notin Domain\ f$
shows $\{f \cup \{(x, y)\} \mid y. y \in A\} \subseteq runiqs$
(proof)

lemma *lm066:*

$converse\ ' (converse\ ' X) = X$
(proof)

lemma *lm067:*

$runiq\ (f^{-1}) = (f \in converse\ 'runiqs)$
(proof)

lemma *lm068*:

assumes $\text{runiq } (f^{-1}) A \cap \text{Range } f = \{\}$
shows $\text{converse } \{ f \cup \{(x, y)\} \mid y . y \in A \} \subseteq \text{runiqs}$
<proof>

lemma *lm069*:

assumes $f \in \text{converse}'\text{runiqs } A \cap \text{Range } f = \{\}$
shows $\{ f \cup \{(x, y)\} \mid y . y \in A \} \subseteq \text{converse}'\text{runiqs}$
(is ?l \subseteq ?r)
<proof>

lemma *lm070*:

$\{ R \cup \{(x, y)\} \mid y . y \in A \} \subseteq \text{totalRels } (\{x\} \cup \text{Domain } R) (A \cup \text{Range } R)$
<proof>

lemma *lm071*:

$\text{injectionsUniverse} = \text{runiqs} \cap \text{converse}'\text{runiqs}$
<proof>

lemma *lm072*:

assumes $f \in \text{injectionsUniverse } x \notin \text{Domain } f A \cap (\text{Range } f) = \{\}$
shows $\{ f \cup \{(x, y)\} \mid y . y \in A \} \subseteq \text{injectionsUniverse}$
(is ?l \subseteq ?r)
<proof>

lemma *lm073*:

$\text{injections } X Y = \text{totalRels } X Y \cap \text{injectionsUniverse}$
<proof>

lemma *lm074*:

assumes $f \in \text{injectionsUniverse}$
shows $f \text{ outside } A \in \text{injectionsUniverse}$
<proof>

lemma *lm075*:

assumes $R \in \text{totalRels } A B$
shows $R \text{ outside } C \in \text{totalRels } (A - C) B$
<proof>

lemma *lm076*:

assumes $g \in \text{injections } A B$
shows $g \text{ outside } C \in \text{injections } (A - C) B$
<proof>

lemma *lm077*:

assumes $g \in \text{injections } A B$
shows $g \text{ outside } C \in \text{injections } (A - C) B$
<proof>

lemma *lm078*:

$\{x\} \times \{y\} = \{(x, y)\}$
<proof>

lemma *lm079*:

assumes $x \in \text{Domain } f$ *runiq* f
shows $\{x\} \times f''\{x\} = \{(x, f, x)\}$
<proof>

corollary *lm080*:

assumes $x \in \text{Domain } f$ *runiq* f
shows $f = (f -- x) \cup \{(x, f, x)\}$
<proof>

lemma *lm081*:

assumes $f \in \text{injectionsUniverse}$
shows $\text{Range}(f \text{ outside } A) = \text{Range } f - f''A$
<proof>

lemma *lm082*:

assumes $g \in \text{injections } X Y$ $x \in \text{Domain } g$
shows $g \in \{g -- x \cup \{(x, y)\} \mid y. y \in Y - (\text{Range}(g -- x))\}$
<proof>

corollary *lm083*:

assumes $x \notin X$ $g \in \text{injections } (\{x\} \cup X) Y$
shows $g -- x \in \text{injections } X Y$
<proof>

corollary *lm084*:

assumes $x \notin X$ $g \in \text{injections } (\{x\} \cup X) Y$
(is $g \in \text{injections } (?X) Y$
shows $\exists f \in \text{injections } X Y. g \in \{f \cup \{(x, y)\} \mid y. y \in Y - (\text{Range } f)\}$
<proof>

corollary *lm085*:

assumes $x \notin X$
shows $\text{injections } (\{x\} \cup X) Y \subseteq$
 $(\bigcup f \in \text{injections } X Y. \{f \cup \{(x, y)\} \mid y. y \in Y - (\text{Range } f)\})$
<proof>

lemma *lm086*:

assumes $x \notin X$
shows $(\bigcup f \in \text{injections } X Y. \{f \cup \{(x, y)\} \mid y. y \in Y - \text{Range } f\}) \subseteq$
 $\text{injections } (\{x\} \cup X) Y$
<proof>

corollary *injectionsUnionCommute*:

assumes $x \notin X$
shows $(\bigcup f \in \text{injections } X \ Y. \{f \cup \{(x, y)\} \mid y . y \in Y - (\text{Range } f)\}) =$
 $\text{injections } (\{x\} \cup X) \ Y$
(is ?r=injections ?X -)
 <proof>

lemma lm087:
assumes $\forall x. (P \ x \longrightarrow (f \ x = g \ x))$
shows $\text{Union } \{f \ x \mid x. P \ x\} = \text{Union } \{g \ x \mid x. P \ x\}$
 <proof>

lemma lm088:
assumes $x \notin \text{Domain } R$
shows $R \ +* \ \{(x, y)\} = R \cup \{(x, y)\}$
 <proof>

lemma lm089:
assumes $x \notin X$
shows $(\bigcup f \in \text{injections } X \ Y. \{f \ +* \ \{(x, y)\} \mid y . y \in Y - \text{Range } f\}) =$
 $(\bigcup f \in \text{injections } X \ Y. \{f \cup \{(x, y)\} \mid y . y \in Y - \text{Range } f\})$
(is ?l = ?r)
 <proof>

corollary lm090:
assumes $x \notin X$
shows $(\bigcup f \in \text{injections } X \ Y. \{f \ +* \ \{(x, y)\} \mid y . y \in Y - \text{Range } f\}) =$
 $\text{injections } (\{x\} \cup X) \ Y$
(is ?l=?r)
 <proof>

lemma lm091:
 $\text{set } [f \cup \{(x, y)\} . y \leftarrow (\text{filter } (\%y. y \notin (\text{Range } f)) \ Y)] =$
 $\{f \cup \{(x, y)\} \mid y . y \in (\text{set } Y) - (\text{Range } f)\}$
 <proof>

lemma lm092:
assumes $\forall x \in \text{set } L. \text{set } (F \ x) = G \ x$
shows $\text{set } (\text{concat } [F \ x . x <- L]) = (\bigcup x \in \text{set } L. G \ x)$
 <proof>

lemma lm093:
 $\text{set } (\text{concat } [[f \cup \{(x, y)\} . y \leftarrow (\text{filter } (\%y. y \notin \text{Range } f) \ Y)]. f \leftarrow F]) =$
 $(\bigcup f \in \text{set } F. \{f \cup \{(x, y)\} \mid y . y \in (\text{set } Y) - (\text{Range } f)\})$
 <proof>

lemma *lm094*:

assumes *finite Y*

shows $set [f \text{+*} \{(x,y)\} . y \leftarrow sorted\text{-list-of-set} (Y - (Range f))] =$
 $\{ f \text{+*} \{(x,y)\} \mid y . y \in Y - (Range f) \}$
<proof>

lemma *lm095*:

assumes *finite Y*

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

9.5 Computable injections

fun *injectionsAlg*

where

injectionsAlg [] (*Y::'a list*) = [{}]
injectionsAlg (*x#xs*) *Y* =
 $concat [[R \cup \{(x,y)\} . y \leftarrow (filter (\%y. y \notin Range R) Y)]$
 $. R \leftarrow injectionsAlg xs Y]$

corollary *lm096*:

$set (injectionsAlg (x \# xs) Y) =$
 $(\bigcup f \in set (injectionsAlg xs Y). \{f \cup \{(x,y)\} \mid y . y \in (set Y) - (Range f)\})$
<proof>

corollary *lm097*:

assumes $set (injectionsAlg xs Y) = injections (set xs) (set Y)$

shows $set (injectionsAlg (x \# xs) Y) =$
 $(\bigcup f \in injections (set xs) (set Y). \{f \cup \{(x,y)\} \mid y . y \in (set Y) - (Range$
 $f)\})$
<proof>

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*:

$injections \{\} Y = \{\{\}\}$
<proof>

lemma *lm099*:

$injections \{\} Y = \{\{\}\}$
<proof>

lemma *injectionsFromEmptyIsEmpty*:

$injectionsAlg [] Y = [{}]$
<proof>

lemma *lm100*:

assumes $x \notin \text{set } xs$ $\text{set } (\text{injectionsAlg } xs \ Y) = \text{injections } (\text{set } xs) \ (\text{set } Y)$
shows $\text{set } (\text{injectionsAlg } (x \# \ xs) \ Y) = \text{injections } (\{x\} \cup \text{set } xs) \ (\text{set } Y)$
(is ?l=?r)
<proof>

lemma *lm101*:

assumes $x \notin \text{set } xs$
 $\text{set } (\text{injections-alg } xs \ Y) = \text{injections } (\text{set } xs) \ Y$
finite Y
shows $\text{set } (\text{injections-alg } (x\#xs) \ Y) = \text{injections } (\{x\} \cup \text{set } xs) \ Y$
(is ?l=?r)
<proof>

lemma *listInduct*:

assumes $P \ \square \ \forall \ xs \ x. \ P \ xs \ \longrightarrow \ P \ (x\#xs)$
shows $\forall \ x. \ P \ x$
<proof>

lemma *injectionsFromEmptyAreEmpty*:

$\text{set } (\text{injections-alg } \square \ Z) = \{\{\}\}$
<proof>

theorem *injections-equiv*:

assumes *finite Y and distinct X*
shows $\text{set } (\text{injections-alg } X \ Y) = \text{injections } (\text{set } X) \ Y$
<proof>

lemma *lm102*:

assumes $l \in \text{set } (\text{all-partitions-list } G)$ *distinct G*
shows *distinct l*
<proof>

lemma *bridgingInjection*:

assumes $\text{card } N > 0$ *distinct G*
shows $\forall \ l \in \text{set } (\text{all-partitions-list } G). \ \text{set } (\text{injections-alg } l \ N) =$
 $\text{injections } (\text{set } l) \ N$
<proof>

lemma *lm103*:

assumes $\text{card } N > 0$ *distinct G*
shows $\{\text{injections } P \ N \mid P. \ P \in \text{all-partitions } (\text{set } G)\} =$
 $\text{set } [\text{set } (\text{injections-alg } l \ N) . \ l \leftarrow \text{all-partitions-list } G]$
<proof>

lemma *lm104*:
assumes *card N > 0 distinct G*
shows $\text{Union } \{ \text{injections } P \ N \mid P. P \in \text{all-partitions } (\text{set } G) \} =$
 $\text{Union } (\text{set } [\text{set } (\text{injections-alg } l \ N) . l \leftarrow \text{all-partitions-list } G])$
(is $\text{Union } ?L = \text{Union } ?R$ **)**
 $\langle \text{proof} \rangle$

corollary *allAllocationsBridgingLemma*:
assumes *card N > 0 distinct G*
shows $\text{allAllocations } N \ (\text{set } G) =$
 $\text{set}(\text{allAllocationsAlg } N \ G)$
 $\langle \text{proof} \rangle$

end

10 Termination theorem for uniform tie-breaking

theory *UniformTieBreaking*

imports
StrictCombinatorialAuction
Universes
HOL-Library.Code-Target-Nat

begin

10.1 Uniform tie breaking: definitions

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

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

abbreviation $\text{omega pair} == \{ \text{fst pair} \} \times (\text{finestpart } (\text{snd pair}))$

definition $\text{pseudoAllocation allocation} == \bigcup (\text{omega } ' \text{ allocation})$

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

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

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

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

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

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

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

abbreviation *Tiebids allocation N G* == *summedBidVectorRel (realomaxbid 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*
<proof>

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 $N\ G\ b \subseteq allAllocations\ N\ G$
(*proof*)

lemma *subsetAllocation*:
assumes $a \in allocationsUniverse\ c \subseteq a$
shows $c \in allocationsUniverse$
(*proof*)

lemma *lm001*:
assumes $a \in allocationsUniverse$
shows $a\ outside\ X \in allocationsUniverse$
(*proof*)

corollary *lm002*:
 $\{x\} \times (\{X\} - \{\{\}\}) \in allocationsUniverse$
(*proof*)

corollary *lm003*:
 $\{(x, \{y\})\} \in allocationsUniverse$
(*proof*)

corollary *lm004*:
 $allocationsUniverse \neq \{\}$
(*proof*)

corollary *lm005*:
 $\{\} \in allocationsUniverse$
(*proof*)

lemma *lm006*:
assumes $G \neq \{\}$
shows $\{G\} \in all-partitions\ G$
(*proof*)

lemma *lm007*:
assumes $n \in N$
shows $\{(G, n)\} \in totalRels\ \{G\}\ N$
(*proof*)

lemma *lm008*:

assumes $n \in N$
shows $\{(G, n)\} \in \text{injections } \{G\} N$
 $\langle \text{proof} \rangle$

corollary *lm009*:
assumes $G \neq \{\}$ $n \in N$
shows $\{(G, n)\} \in \text{possible-allocations-rel } G N$
 $\langle \text{proof} \rangle$

corollary *lm010*:
assumes $N \neq \{\}$ $G \neq \{\}$
shows $\text{allAllocations } N G \neq \{\}$
 $\langle \text{proof} \rangle$

corollary *lm011*:
assumes $N \neq \{\}$ $\text{finite } N$ $G \neq \{\}$ $\text{finite } G$
shows $\text{winningAllocationsRel } N G \text{ bids} \neq \{\}$ & $\text{finite } (\text{winningAllocationsRel } N G \text{ bids})$
 $\langle \text{proof} \rangle$

lemma *lm012*:
 $\text{allAllocations } N \{\} \subseteq \{\{\}\}$
 $\langle \text{proof} \rangle$

lemma *lm013*:
assumes $a \in \text{allAllocations } N G \text{ finite } G$
shows $\text{finite } (\text{Range } a)$
 $\langle \text{proof} \rangle$

corollary *allocationFinite*:
assumes $a \in \text{allAllocations } N G \text{ finite } G$
shows $\text{finite } a$
 $\langle \text{proof} \rangle$

lemma *lm014*:
assumes $a \in \text{allAllocations } N G \text{ finite } G$
shows $\forall y \in \text{Range } a. \text{finite } y$
 $\langle \text{proof} \rangle$

corollary *lm015*:
assumes $a \in \text{allAllocations } N G \text{ finite } G$
shows $\text{card } G = \text{sum card } (\text{Range } a)$
 $\langle \text{proof} \rangle$

10.3 Results on summed bid vectors

lemma *lm016*:
 $\text{summedBidVectorRel bids } N G =$

$\{(pair, \text{sum } (\%g. \text{bids } (fst \text{ pair}, g)) (\text{finestpart } (snd \text{ pair}))) \mid$
 $pair. pair \in N \times (Pow \ G - \{\{\}\})\}$
 <proof>

corollary *lm017*:

$\{(pair, \text{sum } (\%g. \text{bids } (fst \text{ pair}, g)) (\text{finestpart } (snd \text{ pair}))) \mid$
 $pair. pair \in (N \times (Pow \ G - \{\{\}\})) \} \parallel a =$
 $\{(pair, \text{sum } (\%g. \text{bids } (fst \text{ pair}, g)) (\text{finestpart } (snd \text{ pair}))) \mid$
 $pair. pair \in (N \times (Pow \ G - \{\{\}\})) \cap a\}$
 <proof>

corollary *lm018*:

$(\text{summedBidVectorRel } bids \ N \ G) \parallel a =$
 $\{(pair, \text{sum } (\%g. \text{bids } (fst \text{ pair}, g)) (\text{finestpart } (snd \text{ pair}))) \mid$
 $pair. pair \in (N \times (Pow \ G - \{\{\}\})) \cap a\}$
 (is ?L = ?R)
 <proof>

lemma *lm019*:

$(\text{summedBid } bids) \text{ ' } ((N \times (Pow \ G - \{\{\}\})) \cap a) =$
 $\{(pair, \text{sum } (\%g. \text{bids } (fst \text{ pair}, g)) (\text{finestpart } (snd \text{ pair}))) \mid$
 $pair. pair \in (N \times (Pow \ G - \{\{\}\})) \cap a\}$
 <proof>

corollary *lm020*:

$(\text{summedBidVectorRel } bids \ N \ G) \parallel a = (\text{summedBid } bids) \text{ ' } ((N \times (Pow \ G -$
 $\{\{\}\})) \cap a)$
 (is ?L=?R)
 <proof>

lemma *summedBidInjective*:

$\text{inj-on } (\text{summedBid } bids) \ UNIV$
 <proof>

corollary *lm021*:

$\text{inj-on } (\text{summedBid } bids) \ X$
 <proof>

lemma *lm022*:

$\text{sum } snd \ (\text{summedBidVectorRel } bids \ N \ G) =$
 $\text{sum } (snd \circ (\text{summedBid } bids)) \ (N \times (Pow \ G - \{\{\}\}))$
 <proof>

corollary *lm023*:

$snd (summedBid\ bids\ pair) = sum\ bids (omega\ pair)$
<proof>

corollary *lm024*:

$snd \circ summedBid\ bids = (sum\ bids) \circ omega$
<proof>

lemma *lm025*:

assumes *finite (finestpart (snd pair))*
shows $card (omega\ pair) = card (finestpart (snd\ pair))$
<proof>

corollary *lm026*:

assumes *finite (snd pair)*
shows $card (omega\ pair) = card (snd\ pair)$
<proof>

lemma *lm027*:

assumes $\{\} \notin Range\ f\ runiq\ f$
shows *is-non-overlapping (omega ' f)*
<proof>

lemma *lm028*:

assumes $\{\} \notin Range\ X$
shows *inj-on omega X*
<proof>

lemma *lm029*:

assumes $\{\} \notin Range\ a \ \forall X \in omega\ ' a.\ finite\ X$
is-non-overlapping (omega ' a)
shows $card (pseudoAllocation\ a) = sum (card \circ omega)\ a$
(**is** ?L = ?R)
<proof>

lemma *lm030*:

$card (omega\ pair) = card (snd\ pair)$
<proof>

corollary *lm031*:

$card \circ omega = card \circ snd$
<proof>

corollary *lm032*:

assumes $\{\} \notin Range\ a \ \forall pair \in a.\ finite (snd\ pair)\ finite\ a\ runiq\ a$
shows $card (pseudoAllocation\ a) = sum (card \circ snd)\ a$
<proof>

corollary *lm033*:

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

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

<proof>

corollary *lm034*:

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

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

<proof>

corollary *lm035*:

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

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

<proof>

corollary *lm036*:

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

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

<proof>

lemma *lm037*:

$\text{pseudoAllocation } \{\} = \{\}$

<proof>

corollary *lm038*:

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

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

<proof>

corollary *lm039*:

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

shows $\text{finite } (\text{pseudoAllocation } a)$

<proof>

corollary *lm040*:

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

shows $\text{finite } (\text{pseudoAllocation } a)$

<proof>

lemma *lm041*:

assumes $a \in \text{allAllocations } N \ G \ aa \in \text{allAllocations } N \ G \ \text{finite } G$
shows $(\text{card } (\text{pseudoAllocation } aa \cap (\text{pseudoAllocation } a)) = \text{card } (\text{pseudoAllocation } a)) =$
 $(\text{pseudoAllocation } a = \text{pseudoAllocation } aa)$
 $\langle \text{proof} \rangle$

lemma *lm042*:
 $\text{omega pair} = \{\text{fst pair}\} \times \{\{y\} \mid y. y \in \text{snd pair}\}$
 $\langle \text{proof} \rangle$

lemma *lm043*:
 $\text{omega pair} = \{(\text{fst pair}, \{y\}) \mid y. y \in \text{snd pair}\}$
 $\langle \text{proof} \rangle$

lemma *lm044*:
 $\text{pseudoAllocation } a = \bigcup \{ \{(\text{fst pair}, \{y\}) \mid y. y \in \text{snd pair}\} \mid \text{pair. pair} \in a \}$
 $\langle \text{proof} \rangle$

lemma *lm045*:
 $\bigcup \{ \{(\text{fst pair}, \{y\}) \mid y. y \in \text{snd pair}\} \mid \text{pair. pair} \in a \} =$
 $\{(\text{fst pair}, \{y\}) \mid y \text{ pair. } y \in \text{snd pair} \ \& \ \text{pair} \in a \}$
 $\langle \text{proof} \rangle$

corollary *lm046*:
 $\text{pseudoAllocation } a = \{(\text{fst pair}, Y) \mid Y \text{ pair. } Y \in \text{finestpart } (\text{snd pair}) \ \& \ \text{pair} \in a \}$
 $\langle \text{proof} \rangle$

lemma *lm047*:
assumes *runiq a*
shows $\{(\text{fst pair}, Y) \mid Y \text{ pair. } Y \in \text{finestpart } (\text{snd pair}) \ \& \ \text{pair} \in a \} =$
 $\{(x, Y) \mid Y \text{ pair. } Y \in \text{finestpart } (a, x) \ \& \ x \in \text{Domain } a \}$
 $(\text{is } ?L = ?R)$
 $\langle \text{proof} \rangle$

corollary *lm048*:
assumes *runiq a*
shows $\text{pseudoAllocation } a = \{(x, Y) \mid Y \text{ pair. } Y \in \text{finestpart } (a, x) \ \& \ x \in \text{Domain } a \}$
 $\langle \text{proof} \rangle$

corollary *lm049*:
 $\text{Range } (\text{pseudoAllocation } a) = \bigcup (\text{finestpart } ` (\text{Range } a))$
 $\langle \text{proof} \rangle$

corollary *lm050*:
 $\text{Range } (\text{pseudoAllocation } a) = \text{finestpart } (\bigcup (\text{Range } a))$

<proof>

lemma *lm051:*

pseudoAllocation a = {(fst pair, {y}) | y pair. y ∈ snd pair & pair ∈ a}
<proof>

lemma *lm052:*

{(fst pair, {y}) | y pair. y ∈ snd pair & pair ∈ a} =
{(x, {y}) | x y. y ∈ ⋃ (a“{x}) & x ∈ Domain a}
<proof>

lemma *lm053:*

pseudoAllocation a = {(x, {y}) | x y. y ∈ ⋃ (a“{x}) & x ∈ Domain a}
(is ?L=?R)
<proof>

lemma *lm054:*

runiq (summedBidVectorRel bids N G)
<proof>

corollary *lm055:*

runiq (summedBidVectorRel bids N G || a)
<proof>

lemma *summedBidVectorCharacterization:*

N × (Pow G - {{{}}}) = Domain (summedBidVectorRel bids N G)
<proof>

corollary *lm056:*

assumes *a ∈ allAllocations N G*
shows *a ⊆ Domain (summedBidVectorRel bids N G)*
<proof>

corollary *lm057:*

sum (summedBidVector bids N G) (a ∩ (Domain (summedBidVectorRel bids N G))) =
sum snd ((summedBidVectorRel bids N G) || a)
<proof>

corollary *lm058:*

assumes *a ∈ allAllocations N G*
shows *sum (summedBidVector bids N G) a = sum snd ((summedBidVectorRel bids N G) || a)*
<proof>

corollary *lm059:*

assumes *a ∈ allAllocations N G*
shows *sum (summedBidVector bids N G) a =*
sum snd ((summedBid bids) ‘ ((N × (Pow G - {{{}})) ∩ a))

(is ?X=?R)
<proof>

corollary *lm060*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{sum } (\text{summedBidVector } \text{bids } N \ G) \ a = \text{sum } \text{snd } ((\text{summedBid } \text{bids}) \ ' a)$
(is ?L=?R)
<proof>

corollary *lm061*:

$\text{sum } \text{snd } ((\text{summedBid } \text{bids}) \ ' a) = \text{sum } (\text{snd } \circ (\text{summedBid } \text{bids})) \ a$
<proof>

corollary *lm062*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{sum } (\text{summedBidVector } \text{bids } N \ G) \ a = \text{sum } (\text{snd } \circ (\text{summedBid } \text{bids})) \ a$
(is ?L=?R)
<proof>

corollary *lm063*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{sum } (\text{summedBidVector } \text{bids } N \ G) \ a = \text{sum } ((\text{sum } \text{bids}) \circ \text{omega}) \ a$
(is ?L=?R)
<proof>

corollary *lm064*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{sum } (\text{summedBidVector } \text{bids } N \ G) \ a = \text{sum } (\text{sum } \text{bids}) \ (\text{omega} \ ' a)$
<proof>

lemma *lm065*:

assumes *finite* (*snd pair*)
shows *finite* (*omega pair*)
<proof>

corollary *lm066*:

assumes $\forall y \in (\text{Range } a). \text{finite } y$
shows $\forall y \in (\text{omega} \ ' a). \text{finite } y$
<proof>

corollary *lm067*:

assumes $a \in \text{allAllocations } N \ G \ \text{finite } G$
shows $\forall x \in (\text{omega} \ ' a). \text{finite } x$
<proof>

corollary *lm068*:

assumes $a \in \text{allAllocations } N \ G$
shows *is-non-overlapping* (*omega ' a*)
<proof>

lemma *lm069*:

assumes $a \in \text{allAllocations } N \ G \ \text{finite } G$
shows $\text{sum } (\text{sum bids}) \ (\text{omega}' a) = \text{sum bids } (\bigcup (\text{omega}' a))$
<proof>

corollary *lm070*:

assumes $a \in \text{allAllocations } N \ G \ \text{finite } G$
shows $\text{sum } (\text{summedBidVector bids } N \ G) \ a = \text{sum bids } (\text{pseudoAllocation } a)$
(is ?L = ?R)
<proof>

lemma *lm071*:

$\text{Domain } (\text{pseudoAllocation } a) \subseteq \text{Domain } a$
<proof>

corollary *lm072*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{Domain } (\text{pseudoAllocation } a) \subseteq N \ \& \ \text{Range } (\text{pseudoAllocation } a) = \text{finestpart } G$
<proof>

corollary *lm073*:

assumes $a \in \text{allAllocations } N \ G$
shows $\text{pseudoAllocation } a \subseteq N \times \text{finestpart } G$
<proof>

10.4 From Pseudo-allocations to allocations

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

lemma *lm074*:

assumes $\text{runiq } a \ \{\} \notin \text{Range } a$
shows $a = \text{pseudoAllocationInv } (\text{pseudoAllocation } a)$
<proof>

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>

lemma *lm076*:

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

corollary *lm077*:

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

<proof>

lemma *lm078*:

injectionsUniverse \subseteq *runiqs*

<proof>

lemma *lm079*:

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

<proof>

corollary *lm080*:

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

<proof>

corollary *lm081*:

inj-on pseudoAllocation allocationsUniverse

<proof>

corollary *lm082*:

inj-on pseudoAllocation (allAllocations N G)

<proof>

lemma *lm083*:

assumes *card N* $>$ 0 *distinct G*

shows *winningAllocationsRel N (set G) bids* \subseteq *set (allAllocationsAlg N G)*

<proof>

corollary *lm084*:

assumes *N* \neq $\{\}$ *finite N distinct G set G* \neq $\{\}$

shows *winningAllocationsRel N (set G) bids* \cap *set (allAllocationsAlg N G)* \neq $\{\}$

<proof>

lemma *lm085*:

X = ($\%x. x \in X$) - $\{\text{True}\}$

<proof>

corollary *lm086*:

assumes *N* \neq $\{\}$ *finite N distinct G set G* \neq $\{\}$

shows ($\%x. x \in$ *winningAllocationsRel N (set G) bids*) - $\{\text{True}\} \cap$
set (allAllocationsAlg N G) \neq $\{\}$

<proof>

lemma *lm087*:

assumes *P* - $\{\text{True}\} \cap$ *set l* \neq $\{\}$

shows *takeAll P l* \neq \square

<proof>

corollary *lm088*:

assumes $N \neq \{\}$ *finite* N *distinct* G *set* $G \neq \{\}$
shows $\text{takeAll } (\%x. x \in \text{winningAllocationsRel } N \text{ (set } G) \text{ bids) (allAllocationsAlg } N \text{ } G) \neq []$
 $\langle \text{proof} \rangle$

corollary *lm089*:

assumes $N \neq \{\}$ *finite* N *distinct* G *set* $G \neq \{\}$
shows $\text{perm2 } (\text{takeAll } (\%x. x \in \text{winningAllocationsRel } N \text{ (set } G) \text{ bids) (allAllocationsAlg } N \text{ } G))$
 $n \neq []$
 $\langle \text{proof} \rangle$

corollary *lm090*:

assumes $N \neq \{\}$ *finite* N *distinct* G *set* $G \neq \{\}$
shows $\text{chosenAllocation } N \text{ } G \text{ bids random} \in \text{winningAllocationsRel } N \text{ (set } G) \text{ bids}$
 $\langle \text{proof} \rangle$

lemma *lm091*:

assumes *finite* G $a \in \text{allAllocations } N \text{ } G$ $aa \in \text{allAllocations } N \text{ } G$
shows $\text{real}(\text{sum}(\text{maxbid } a \text{ } N \text{ } G)(\text{pseudoAllocation } a)) -$
 $\text{sum}(\text{maxbid } a \text{ } N \text{ } G)(\text{pseudoAllocation } aa) =$
 $\text{real}(\text{card } G) -$
 $\text{card}(\text{pseudoAllocation } aa \cap (\text{pseudoAllocation } a))$
 $\langle \text{proof} \rangle$

lemma *lm092*:

$\text{summedBidVectorRel } \text{bids } N \text{ } G = \text{graph } (N \times (\text{Pow } G - \{\{\}\})) (\text{summedBidSecond } \text{bids})$
 $\langle \text{proof} \rangle$

lemma *lm093*:

assumes $x \in X$
shows $\text{toFunction } (\text{graph } X \text{ } f) \text{ } x = f \text{ } x$
 $\langle \text{proof} \rangle$

corollary *lm094*:

assumes $\text{pair} \in N \times (\text{Pow } G - \{\{\}\})$
shows $\text{summedBidVector } \text{bids } N \text{ } G \text{ pair} = \text{summedBidSecond } \text{bids } \text{pair}$
 $\langle \text{proof} \rangle$

lemma *lm095*:

$\text{summedBidSecond } (\text{real} \circ ((\text{bids}:: - \Rightarrow \text{nat}))) \text{ pair} = \text{real}(\text{summedBidSecond } \text{bids } \text{pair})$
 $\langle \text{proof} \rangle$

lemma *lm096*:

assumes $pair \in N \times (Pow\ G - \{\{\}\})$
shows $summedBidVector\ (real \circ (bids:: - => nat))\ N\ G\ pair =$
 $real\ (summedBidVector\ bids\ N\ G\ pair)$
<proof>

corollary *lm097*:

assumes $X \subseteq N \times (Pow\ G - \{\{\}\})$
shows $\forall pair \in X. summedBidVector\ (real \circ (bids:: - => nat))\ N\ G\ pair =$
 $(real \circ (summedBidVector\ bids\ N\ G))\ pair$
<proof>

corollary *lm098*:

assumes $aa \subseteq N \times (Pow\ G - \{\{\}\})$
shows $sum\ ((summedBidVector\ (real \circ (bids:: - => nat))\ N\ G))\ aa =$
 $real\ (sum\ ((summedBidVector\ bids\ N\ G))\ aa)$
(is ?L=?R)
<proof>

corollary *lm099*:

assumes $aa \in allAllocations\ N\ G$
shows $sum\ ((summedBidVector\ (real \circ (bids:: - => nat))\ N\ G))\ aa =$
 $real\ (sum\ ((summedBidVector\ bids\ N\ G))\ aa)$
<proof>

corollary *lm100*:

assumes $finite\ G\ a \in allAllocations\ N\ G\ aa \in allAllocations\ N\ G$
shows $real\ (sum\ (tiebids\ a\ N\ G)\ a) - sum\ (tiebids\ a\ N\ G)\ aa =$
 $real\ (card\ G) - card\ (pseudoAllocation\ aa \cap (pseudoAllocation\ a))$
(is ?L=?R)
<proof>

corollary *lm101*:

assumes $finite\ G\ a \in allAllocations\ N\ G\ aa \in allAllocations\ N\ G$
 $x = real\ (sum\ (tiebids\ a\ N\ G)\ a) - sum\ (tiebids\ a\ N\ G)\ aa$
shows $x \leq card\ G \ \&$
 $x \geq 0 \ \&$
 $(x=0 \iff a = aa) \ \&$
 $(aa \neq a \implies sum\ (tiebids\ a\ N\ G)\ aa < sum\ (tiebids\ a\ N\ G)\ a)$
<proof>

corollary *lm102*:

assumes $finite\ G\ a \in allAllocations\ N\ G$
 $aa \in allAllocations\ N\ G\ aa \neq a$
shows $sum\ (tiebids\ a\ N\ G)\ aa < sum\ (tiebids\ a\ N\ G)\ a$
<proof>

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

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

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

theorem *winningAllocationUniqueness*:
assumes $N \neq \{\}$ *distinct G set G $\neq \{\}$ finite N*
shows $\text{terminatingAuctionRel } N \text{ } G \text{ (bids) random} = \{\text{chosenAllocation } N \text{ } G \text{ bids random}\}$
<proof>

The computable variant of Else is defined next as Elsee.

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

end

11 VCG auction: definitions and theorems

theory *CombinatorialAuction*

imports
UniformTieBreaking

begin

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

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

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

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

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

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

lemma $allAllocationsEquivalence:$
 $allAllocations N \Omega = allAllocations' N \Omega \ \& \ allAllocations N \Omega = allAllocations''$
 $N \Omega$
 $\langle proof \rangle$

lemma $allAllocationsVarCharacterization:$
 $(a \in allAllocations'' N \Omega) = (a \in allocationsUniverse \ \& \ Domain a \subseteq N \ \&$
 $\ \bigcup (Range a) = \Omega)$
 $\langle proof \rangle$

abbreviation $soldAllocations N \Omega == (Outside' \{seller\}) \ ' (allAllocations (N \cup$
 $\{seller\}) \ \Omega)$

abbreviation $soldAllocations' N \Omega == (Outside' \{seller\}) \ ' (allAllocations' (N$
 $\cup \{seller\}) \ \Omega)$

abbreviation $soldAllocations'' N \Omega == (Outside' \{seller\}) \ ' (allAllocations'' (N$
 $\cup \{seller\}) \ \Omega)$

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

lemma $soldAllocationsEquivalence:$
 $soldAllocations N \Omega = soldAllocations' N \Omega \ \&$
 $soldAllocations' N \Omega = soldAllocations'' N \Omega$
 $\langle proof \rangle$

corollary $soldAllocationsEquivalenceVariant:$
 $soldAllocations = soldAllocations' \ \&$
 $soldAllocations' = soldAllocations'' \ \&$
 $soldAllocations = soldAllocations''$
 $\langle proof \rangle$

lemma $allocationSellerMonotonicity:$
 $soldAllocations (N - \{seller\}) \ \Omega \subseteq soldAllocations N \Omega$
 $\langle proof \rangle$

lemma $allocationsUniverseCharacterization:$
 $(a \in allocationsUniverse) = (a \in allAllocations'' (Domain a) (\bigcup (Range a)))$
 $\langle proof \rangle$

lemma $allocationMonotonicity:$
assumes $N1 \subseteq N2$

shows $allAllocations'' N1 \Omega \subseteq allAllocations'' N2 \Omega$
(proof)

lemma *allocationWithOneParticipant*:
assumes $a \in allAllocations'' N \Omega$
shows $Domain (a \dashv\vdash x) \subseteq N - \{x\}$
(proof)

lemma *soldAllocationIsAllocation*:
assumes $a \in soldAllocations N \Omega$
shows $a \in allocationsUniverse$
(proof)

lemma *soldAllocationIsAllocationVariant*:
assumes $a \in soldAllocations N \Omega$
shows $a \in allAllocations'' (Domain a) (\bigcup (Range a))$
(proof)

lemma *onlyGoodsAreSold*:
assumes $a \in soldAllocations'' N \Omega$
shows $\bigcup (Range a) \subseteq \Omega$
(proof)

lemma *soldAllocationIsRestricted*:
 $a \in soldAllocations'' N \Omega =$
 $(\exists aa. aa \dashv\vdash (seller) = a \wedge aa \in allAllocations'' (N \cup \{seller\}) \Omega)$
(proof)

lemma *restrictionConservation*:
 $(R +* (\{x\} \times Y)) \dashv\vdash x = R \dashv\vdash x$
(proof)

lemma *allocatedToBuyerMeansSold*:
assumes $a \in allocationsUniverse$ $Domain a \subseteq N - \{seller\} \bigcup (Range a) \subseteq \Omega$
shows $a \in soldAllocations'' N \Omega$
(proof)

lemma *allocationCharacterization*:
 $a \in allAllocations N \Omega =$
 $(a \in injectionsUniverse \ \& \ Domain a \subseteq N \ \& \ Range a \in all-partitions \ \Omega)$
(proof)

lemma *lm01*:
assumes $a \in soldAllocations'' N \Omega$
shows $Domain a \subseteq N - \{seller\} \ \& \ a \in allocationsUniverse$
(proof)

corollary *lm02*:

assumes $a \in \text{soldAllocations}'' N \Omega$

shows $a \in \text{allocationsUniverse} \ \& \ \text{Domain } a \subseteq N - \{\text{seller}\} \ \& \ \bigcup (\text{Range } a) \subseteq \Omega$
<proof>

corollary *lm03*:

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

$(a \in \text{allocationsUniverse} \ \& \ a \in \{aa. \text{Domain } aa \subseteq N - \{\text{seller}\} \ \& \ \bigcup (\text{Range } aa) \subseteq \Omega\})$

(is ?L = ?R)

<proof>

corollary *lm04*:

$a \in \text{soldAllocations}'' N \Omega =$

$(a \in (\text{allocationsUniverse} \cap \{aa. \text{Domain } aa \subseteq N - \{\text{seller}\} \ \& \ \bigcup (\text{Range } aa) \subseteq \Omega\}))$

<proof>

corollary *soldAllocationVariantEquivalence*:

$\text{soldAllocations}'' N \Omega = \text{soldAllocations}''' N \Omega$

(is ?L=?R)

<proof>

lemma *lm05*:

assumes $a \in \text{soldAllocations}''' N \Omega$

shows $a \dashv\vdash n \in \text{soldAllocations}''' (N - \{n\}) \Omega$
<proof>

lemma *allAllocationsEquivalenceExtended*:

$\text{soldAllocations} = \text{soldAllocations}' \ \&$

$\text{soldAllocations}' = \text{soldAllocations}'' \ \&$

$\text{soldAllocations}'' = \text{soldAllocations}'''$

<proof>

corollary *soldAllocationRestriction*:

assumes $a \in \text{soldAllocations} N \Omega$

shows $a \dashv\vdash n \in \text{soldAllocations} (N - \{n\}) \Omega$
<proof>

corollary *allocationGoodsMonotonicity*:

assumes $\Omega 1 \subseteq \Omega 2$

shows $\text{soldAllocations}''' N \Omega 1 \subseteq \text{soldAllocations}''' N \Omega 2$

<proof>

corollary *allocationGoodsMonotonicityVariant*:

assumes $\Omega 1 \subseteq \Omega 2$

shows $\text{soldAllocations}'' N \Omega 1 \subseteq \text{soldAllocations}'' N \Omega 2$

<proof>

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

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

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

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

abbreviation *vcga'* $N \ \Omega \ b \ r ==$
 $(\text{the-elem} \ (\text{argmax} \ (\text{sum} \ (\text{randomBids} \ N \ \Omega \ b \ r)) \ (\text{maximalStrictAllocations} \ N \ (\text{set } \Omega) \ b)))$
-- 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 $\text{vcga} \ N \ \Omega \ b \ r =$
 $(\text{the-elem} \ ((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ b \ r) \ ((\text{argmax} \circ \text{sum}) \ b \ (\text{allAllocations} \ (\{\text{seller}\} \cup N) \ (\text{set } \Omega))))))$
-- seller
<proof>

corollary *lm07:*

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 $\text{vcga} \ N \ \Omega \ b \ r = \text{vcga}' \ N \ \Omega \ b \ r$
(is ?l = ?r)
<proof>

lemma *lm08:*

assumes *distinct* $\Omega \ \text{set } \Omega \neq \{\}$ *finite* N
shows $\text{card} \ ((\text{argmax} \circ \text{sum}) \ (\text{randomBids} \ N \ \Omega \ \text{bids } \text{random}) \ ((\text{argmax} \circ \text{sum}) \ \text{bids} \ (\text{allAllocations} \ (N \cup \{\text{seller}\}) \ (\text{set } \Omega))))$

$\Omega))))) = 1$
 (is card ?l=-)
 <proof>

lemma *vcgaEquivalence*:
 assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
 shows $vcga\ N\ \Omega\ b\ r = vcga'\ N\ \Omega\ b\ r$
 <proof>

theorem *vcgaDefiniteness*:
 assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
 shows $card\ (vcgas\ N\ \Omega\ b\ r) = 1$
 <proof>

lemma *vcgaDefinitenessVariant*:
 assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
 shows $card\ (argmax\ (sum\ (randomBids\ N\ \Omega\ b\ r))\ (maximalStrictAllocations\ N\ (set\ \Omega)\ b)) =$
 1
 (is card ?L=-)
 <proof>

theorem *winningAllocationIsMaximal*:
 assumes *distinct* Ω *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>

corollary *winningAllocationIsMaximalWithoutSeller*:
 assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
 shows $vcga'\ N\ \Omega\ b\ r \in (Outside'\ \{seller\})'(maximalStrictAllocations\ N\ (set\ \Omega)\ b)$
 <proof>

lemma *maximalAllactionWithoutSeller*:
 $(Outside'\ \{seller\})'(maximalStrictAllocations\ N\ \Omega\ b) \subseteq soldAllocations\ N\ \Omega$
 <proof>

lemma *onlyGoodsAreAllocatedAuxiliary*:
 assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
 shows $vcga'\ N\ \Omega\ b\ r \in soldAllocations\ N\ (set\ \Omega)$
 (is ?a \in ?A)
 <proof>

theorem *onlyGoodsAreAllocated*:

assumes $distinct\ \Omega\ set\ \Omega \neq \{\}$ $finite\ N$
shows $vcga\ N\ \Omega\ b\ r \in soldAllocations\ N\ (set\ \Omega)$
(is $- \in ?r$)
 $\langle proof \rangle$

corollary *neutralSeller*:

assumes $\forall X. X \in Range\ a \longrightarrow b\ (seller, X) = 0$ $finite\ a$
shows $sum\ b\ a = sum\ b\ (a -- seller)$
 $\langle proof \rangle$

corollary *neutralSellerVariant*:

assumes $\forall a \in A. finite\ a \ \&\ (\forall X. X \in Range\ a \longrightarrow b\ (seller, X) = 0)$
shows $\{sum\ b\ a \mid a. a \in A\} = \{sum\ b\ (a -- seller) \mid a. a \in A\}$
 $\langle proof \rangle$

lemma *vcgaIsMaximalAux1*:

assumes $distinct\ \Omega\ set\ \Omega \neq \{\}$ $finite\ N$
shows $\exists a. ((a \in (maximalStrictAllocations\ N\ (set\ \Omega)\ b)) \wedge (vcga'\ N\ \Omega\ b\ r = a -- seller) \ \&\ (a \in argmax\ (sum\ b)\ (allAllocations\ (\{seller\} \cup N)\ (set\ \Omega))))$
 $\langle proof \rangle$

lemma *vcgaIsMaximalAux2*:

assumes $distinct\ \Omega\ set\ \Omega \neq \{\}$ $finite\ N$
 $\forall a \in allAllocations\ (\{seller\} \cup N)\ (set\ \Omega). \forall X \in Range\ a. b\ (seller, X) = 0$
(is $\forall a \in ?X. -$)
shows $sum\ b\ (vcga'\ N\ \Omega\ b\ r) = Max\{sum\ b\ a \mid a. a \in soldAllocations\ N\ (set\ \Omega)\}$
 $\langle proof \rangle$

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

theorem *vcgaIsMaximal*:

assumes $distinct\ \Omega\ set\ \Omega \neq \{\}$ $finite\ N \ \forall X. b\ (seller, X) = 0$
shows $sum\ b\ (vcga'\ N\ \Omega\ b\ r) = Max\{sum\ b\ a \mid a. a \in soldAllocations\ N\ (set\ \Omega)\}$
 $\langle proof \rangle$

corollary *vcgaIsAllocationAllocatingGoodsOnly*:

assumes $distinct\ \Omega\ set\ \Omega \neq \{\}$ $finite\ N$
shows $vcga'\ N\ \Omega\ b\ r \in allocationsUniverse \ \&\ \bigcup (Range\ (vcga'\ N\ \Omega\ b\ r)) \subseteq set\ \Omega$
 $\langle proof \rangle$

abbreviation *vcgp* $N\ \Omega\ b\ r\ n ==$

$Max\ (sum\ b\ ' (soldAllocations\ (N - \{n\})\ (set\ \Omega)))$
 $- (sum\ b\ (vcga\ N\ \Omega\ b\ r -- n))$

theorem *vcgpDefiniteness*:
assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
shows $\exists! y. \text{vcgp } N \ \Omega \ b \ r \ n = y$
 $\langle \text{proof} \rangle$

lemma *soldAllocationsFinite*:
assumes *finite* N *finite* Ω
shows *finite* (*soldAllocations* $N \ \Omega$)
 $\langle \text{proof} \rangle$

The price paid by any participant is non-negative.

theorem *NonnegPrices*:
assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N
shows *vcgp* $N \ \Omega \ b \ r \ n \geq (0::\text{price})$
 $\langle \text{proof} \rangle$

lemma *allocationDisjointAuxiliary*:
assumes $a \in \text{allocationsUniverse}$ **and** $n1 \in \text{Domain } a$ **and** $n2 \in \text{Domain } a$ **and**
 $n1 \neq n2$
shows $a.,n1 \cap a.,n2 = \{\}$
 $\langle \text{proof} \rangle$

lemma *allocationDisjoint*:
assumes $a \in \text{allocationsUniverse}$ **and** $n1 \in \text{Domain } a$ **and** $n2 \in \text{Domain } a$ **and**
 $n1 \neq n2$
shows $a.,,n1 \cap a.,,n2 = \{\}$
 $\langle \text{proof} \rangle$

No good is assigned twice.

theorem *PairwiseDisjointAllocations*:
assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N $n1 \neq n2$
shows $(\text{vcga}' \ N \ \Omega \ b \ r),,n1 \cap (\text{vcga}' \ N \ \Omega \ b \ r),,n2 = \{\}$
 $\langle \text{proof} \rangle$

Nothing outside the set of goods is allocated.

theorem *OnlyGoodsAllocated*:
assumes *distinct* Ω *set* $\Omega \neq \{\}$ *finite* N $g \in (\text{vcga} \ N \ \Omega \ b \ r),,n$
shows $g \in \text{set } \Omega$
 $\langle \text{proof} \rangle$

11.2 Computable versions of the VCG formalization

abbreviation *maximalStrictAllocationsAlg* $N \ \Omega \ b ==$
 $\text{argmax } (\text{sum } b) (\text{set } (\text{allAllocationsAlg } (\{\text{seller}\} \cup N) \ \Omega))$

definition *chosenAllocationAlg* $N \ \Omega \ b \ (r::\text{integer}) ==$
 $(\text{randomEl } (\text{takeAll } (\%x. x \in (\text{argmax } \circ \text{sum}) \ b) (\text{set } (\text{allAllocationsAlg } N \ \Omega))))$
 $(\text{allAllocationsAlg } N \ \Omega)$

r)

definition *maxbidAlg* $a N \Omega == (\text{bidMaximizedBy } a N \Omega) \text{ Elsee } 0$

definition *summedBidVectorAlg* $\text{bids } N \Omega == (\text{summedBidVectorRel } \text{bids } N \Omega) \text{ Elsee } 0$

definition *tiebidsAlg* $a N \Omega == \text{summedBidVectorAlg } (\text{maxbidAlg } a N \Omega) N \Omega$

definition *resolvingBidAlg* $N \Omega \text{ bids } \text{random} ==$
tiebidsAlg (*chosenAllocationAlg* $N \Omega \text{ bids } \text{random}$) $N (\text{set } \Omega)$

definition *randomBidsAlg* $N \Omega b \text{ random} == \text{resolvingBidAlg } (N \cup \{\text{seller}\}) \Omega b \text{ random}$

definition *vcgaAlgWithoutLosers* $N \Omega b r ==$
(*the-elem* (*argmax* (*sum* (*randomBidsAlg* $N \Omega b r$))
(*maximalStrictAllocationsAlg* $N \Omega b$)))
-- *seller*

abbreviation *addLosers* *participantset* *allocation* == (*participantset* $\times \{\{\}\}$) $+*$
allocation

definition *vcgaAlg* $N \Omega b r = \text{addLosers } N (\text{vcgaAlgWithoutLosers } N \Omega b r)$

abbreviation *soldAllocationsAlg* $N \Omega ==$
(*Outside'* $\{\text{seller}\}$) ' *set* (*allAllocationsAlg* $(N \cup \{\text{seller}\}) \Omega$)

definition *vcgpAlg* $N \Omega b r n (\text{winningAllocation}::\text{allocation}) =$
Max (*sum* b ' (*soldAllocationsAlg* $(N - \{n\}) \Omega$)) -
(*sum* b (*winningAllocation* -- n))

lemma *functionCompletion*:
assumes $x \in \text{Domain } f$
shows *toFunction* $f x = (f \text{ Elsee } 0) x$
{*proof*}

lemma *lm09*:

assumes $fst\ pair \in N\ snd\ pair \in Pow\ \Omega - \{\{\}\}$
shows $sum\ (\%g.\ (toFunction\ (bidMaximizedBy\ a\ N\ \Omega))\ (fst\ pair,\ g))$
 $\quad (finestpart\ (snd\ pair)) =$
 $sum\ (\%g.\ ((bidMaximizedBy\ a\ N\ \Omega)\ Elsee\ 0)\ (fst\ pair,\ g))$
 $\quad (finestpart\ (snd\ pair))$
 <proof>

corollary *lm10*:

assumes $pair \in N \times (Pow\ \Omega - \{\{\}\})$
shows $summedBid\ (toFunction\ (bidMaximizedBy\ a\ N\ \Omega))\ pair =$
 $summedBid\ ((bidMaximizedBy\ a\ N\ \Omega)\ Elsee\ 0)\ pair$
 <proof>

corollary *lm11*:

$\forall\ pair \in N \times (Pow\ \Omega - \{\{\}\})$.
 $summedBid\ (toFunction\ (bidMaximizedBy\ a\ N\ \Omega))\ pair =$
 $summedBid\ ((bidMaximizedBy\ a\ N\ \Omega)\ Elsee\ 0)\ pair$
 <proof>

corollary *lm12*:

$(summedBid\ (toFunction\ (bidMaximizedBy\ a\ N\ \Omega)))\ ' (N \times (Pow\ \Omega - \{\{\}\})) =$
 $(summedBid\ ((bidMaximizedBy\ a\ N\ \Omega)\ Elsee\ 0))\ ' (N \times (Pow\ \Omega - \{\{\}\}))$
 $(is\ ?f1\ ' ?Z = ?f2\ ' ?Z)$
 <proof>

corollary *lm13*:

$summedBidVectorRel\ (toFunction\ (bidMaximizedBy\ a\ N\ \Omega))\ N\ \Omega =$
 $summedBidVectorRel\ ((bidMaximizedBy\ a\ N\ \Omega)\ Elsee\ 0)\ N\ \Omega$
 <proof>

corollary *maxbidEquivalence*:

$summedBidVectorRel\ (maxbid\ a\ N\ \Omega)\ N\ \Omega =$
 $summedBidVectorRel\ (maxbidAlg\ a\ N\ \Omega)\ N\ \Omega$
 <proof>

lemma *summedBidVectorEquivalence*:

assumes $x \in (N \times (Pow\ \Omega - \{\{\}\}))$
shows $summedBidVector\ (maxbid\ a\ N\ \Omega)\ N\ \Omega\ x = summedBidVectorAlg\ (maxbidAlg\ a\ N\ \Omega)\ N\ \Omega\ x$
 $(is\ ?f1\ ?g1\ N\ \Omega\ x = ?f2\ ?g2\ N\ \Omega\ x)$
 <proof>

corollary *chosenAllocationEquivalence*:

assumes $card\ N > 0$ **and** $distinct\ \Omega$
shows $chosenAllocation\ N\ \Omega\ b\ r = chosenAllocationAlg\ N\ \Omega\ b\ r$
 <proof>

corollary *tiebidsBridgingLemma*:

assumes $x \in (N \times (\text{Pow } \Omega - \{\{\}\}))$

shows $\text{tiebids } a \ N \ \Omega \ x = \text{tiebidsAlg } a \ N \ \Omega \ x$

(is ?L=-)

<proof>

definition $\text{tiebids}' = \text{tiebids}$

corollary *tiebidsBridgingLemma'*:

assumes $x \in (N \times (\text{Pow } \Omega - \{\{\}\}))$

shows $\text{tiebids}' \ a \ N \ \Omega \ x = \text{tiebidsAlg } a \ N \ \Omega \ x$

<proof>

abbreviation $\text{resolvingBid}' \ N \ G \ \text{bids random} ==$

$\text{tiebids}' \ (\text{chosenAllocation } N \ G \ \text{bids random}) \ N \ (\text{set } G)$

lemma *resolvingBidEquivalence*:

assumes $x \in (N \times (\text{Pow } (\text{set } \Omega) - \{\{\}\}))$ $\text{card } N > 0$ $\text{distinct } \Omega$

shows $\text{resolvingBid}' \ N \ \Omega \ b \ r \ x = \text{resolvingBidAlg } N \ \Omega \ b \ r \ x$

<proof>

lemma *sumResolvingBidEquivalence*:

assumes $\text{card } N > 0$ $\text{distinct } \Omega$ $a \subseteq (N \times (\text{Pow } (\text{set } \Omega) - \{\{\}\}))$

shows $\text{sum } (\text{resolvingBid}' \ N \ \Omega \ b \ r) \ a = \text{sum } (\text{resolvingBidAlg } N \ \Omega \ b \ r) \ a$

(is ?L=?R)

<proof>

lemma *resolvingBidBridgingLemma*:

assumes $\text{card } N > 0$ $\text{distinct } \Omega$ $a \subseteq (N \times (\text{Pow } (\text{set } \Omega) - \{\{\}\}))$

shows $\text{sum } (\text{resolvingBid } N \ \Omega \ b \ r) \ a = \text{sum } (\text{resolvingBidAlg } N \ \Omega \ b \ r) \ a$

(is ?L=?R)

<proof>

lemma *allAllocationsInPowerset*:

$\text{allAllocations } N \ \Omega \subseteq \text{Pow } (N \times (\text{Pow } \Omega - \{\{\}\}))$

<proof>

corollary *resolvingBidBridgingLemmaVariant1*:

assumes $\text{card } N > 0$ $\text{distinct } \Omega$ $a \in \text{allAllocations } N \ (\text{set } \Omega)$

shows $\text{sum } (\text{resolvingBid } N \ \Omega \ b \ r) \ a = \text{sum } (\text{resolvingBidAlg } N \ \Omega \ b \ r) \ a$

<proof>

corollary *resolvingBidBridgingLemmaVariant2*:

assumes $\text{finite } N$ $\text{distinct } \Omega$ $a \in \text{maximalStrictAllocations } N \ (\text{set } \Omega) \ b$

shows $\text{sum } (\text{randomBids } N \ \Omega \ b \ r) \ a = \text{sum } (\text{randomBidsAlg } N \ \Omega \ b \ r) \ a$

<proof>

corollary *tiebreakingGivesSingleton*:

assumes $\text{distinct } \Omega$ $\text{set } \Omega \neq \{\}$ $\text{finite } N$

shows $\text{card } (\text{argmax } (\text{sum } (\text{randomBidsAlg } N \ \Omega \ b \ r)))$

$(\text{maximalStrictAllocations } N \text{ (set } \Omega \text{) } b) =$

1

<proof>

theorem *maximalAllocationBridgingTheorem:*

assumes *finite N distinct Ω*

shows $\text{maximalStrictAllocations } N \text{ (set } \Omega \text{) } b = \text{maximalStrictAllocationsAlg } N$
 $\Omega \text{ } b$

<proof>

theorem *vcgaAlgDefinedness:*

assumes *distinct Ω set $\Omega \neq \{\}$ finite N*

shows $\text{card (argmax (sum (randomBidsAlg } N \text{ } \Omega \text{ } b \text{ } r)) (\text{maximalStrictAllocationsAlg}$
 $N \text{ } \Omega \text{ } b)) = 1$

<proof>

end

12 VCG auction: Scala code extraction

theory *CombinatorialAuctionCodeExtraction*

imports

CombinatorialAuction

HOL-Library.Code-Target-Nat

HOL-Library.Code-Target-Int

begin

definition *allocationPrettyPrint a =*

$\{ \text{map } (\%x. (x, \text{sorted-list-of-set}(a, x))) ((\text{sorted-list-of-set} \circ \text{Domain}) a) \}$

abbreviation *singleBidConverter x == ((fst x, set ((fst o snd) x)), (snd o snd)*
x)

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

export-code vcgaAlg payments allocationPrettyPrint in Scala module-name VCG
    file <VCG-withoutWrapper.scala>

end

```

References

- [1] Formare github webpage. <https://github.com/formare/auctions/tree/master/isabelle/Auction/Vcg>, 2015. Accessed: 2015-05-08.
- [2] Formare project webpage. <http://www.cs.bham.ac.uk/research/projects/formare/>, 2015. Accessed: 2015-05-08.
- [3] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Set theory or higher order logic to represent auction concepts in isabelle? In *Intelligent Computer Mathematics*, pages 236–251. Springer, 2014. <http://arxiv.org/abs/1406.0774>.
- [4] M. B. Caminati, M. Kerber, C. Lange, and C. Rowat. Sound auction specification and implementation. 16th ACM Conference on Economics and Computation, 2015. <https://doi.org/10.1145/2764468.2764511>, <http://www.cs.bham.ac.uk/~mmk/publications/ec2015.pdf>.
- [5] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial auctions*. MIT Press, 2006.