Vickrey-Clarke-Groves (VCG) Auctions

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

March 17, 2025

Abstract

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

Contents

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

1 Introduction

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

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

The following files are part of the Auction Theory Toolbox (ATT) [1] developed in the ForMaRE project [2]. The theories `CombinatorialAuction.thy`, `StrictCombinatorialAuction.thy` and `UniformTieBreaking.thy` contain the relevant definitions and theorems; `CombinatorialAuctionExamples.thy`

and `CombinatorialAuctionCodeExtraction.thy` present simple helper definitions to run them on given examples and to export them to the Scala language, respectively; `FirstPrice.thy` shows how easy it is to adapt the definitions to the first price combinatorial auction. The remaining theories contain more general mathematical definitions and theorems.

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

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

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

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

1.2 Bridging

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

1.3 Main theorems

The main theorems about VCG auctions are:

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

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

onlyGoodsAreAllocated: only the actually available goods are allocated.

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

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

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

1.4 Scala code extraction

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

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

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

theory *SetUtils*
imports
Main

begin

2.1 Equality

An inference (introduction) rule that combines $[\![?A \subseteq ?B;\ ?B \subseteq ?A]\!] \Longrightarrow ?A = ?B$ and $(\bigwedge x.\ x \in ?A \Longrightarrow x \in ?B) \Longrightarrow ?A \subseteq ?B$ to a single step

lemma *equalitySubsetI*: $(\bigwedge x\ .\ x \in A \Longrightarrow x \in B) \Longrightarrow (\bigwedge x\ .\ x \in B \Longrightarrow x \in A) \Longrightarrow A = B$
⟨*proof*⟩

2.2 Trivial sets

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

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

The empty set is trivial.

lemma *trivial-empty*: $trivial\ \{\}$
⟨*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 $x = the\text{-}elem\ X$

⟨*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* $t \cap X \neq \{\}$
shows $t \subseteq X$
⟨*proof*⟩

lemma *trivial-implies-finite*: **assumes** *trivial X*
shows *finite X*
⟨*proof*⟩

lemma *lm01*: **assumes** *trivial* $(A \times B)$
shows (*finite* $(A \times B)$ & *card A* $*$ (*card B*) $\leq$ *1*)
⟨*proof*⟩

lemma *lm02*: **assumes** *finite X*
shows *trivial X*=(*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** ¬ *trivial X trivial T*
shows $X - T \neq \{\}$
⟨*proof*⟩

lemma *lm06*: **assumes** (*finite* $(A \times B)$ & *card A* $*$ (*card B*) $\leq$ *1*)
shows *trivial* $(A \times B)$
⟨*proof*⟩

lemma *lm07*: *trivial* $(A \times B)$ = (*finite* $(A \times B)$ & *card A* $*$ (*card B*) $\leq$ *1*)
⟨*proof*⟩

lemma *trivial-empty-or-singleton*: *trivial X* = $(X = \{\} \vee X = \{$*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** (*Pow X* $\subseteq \{\{\},X\}$)
shows *trivial X*
⟨*proof*⟩

lemma *lm09*: **assumes** *trivial X*
shows (*Pow X* ⊆ {{},*X*})
⟨*proof*⟩

lemma *lm10*: *trivial X* = (*Pow X* ⊆ {{},*X*})
⟨*proof*⟩

lemma *lm11*: ({*x*} × *UNIV*) ∩ *P* = {*x*} × (*P* `` {*x*})
⟨*proof*⟩

lemma *lm12*: (*x*,*y*) ∈ *P* = (*y* ∈ *P*``{*x*})
⟨*proof*⟩

lemma *lm13*: **assumes** *inj-on f A inj-on f B*
shows *inj-on f* (*A* ∪ *B*) = (*f*`(*A*−*B*) ∩ (*f*`(*B*−*A*)) = {})
⟨*proof*⟩

lemma *injection-union*: **assumes** *inj-on f A inj-on f B* (*f*`*A*) ∩ (*f*`*B*) = {}
shows *inj-on f* (*A* ∪ *B*)
⟨*proof*⟩

lemma *lm14*: (*Pow X* = {*X*}) = (*X*={})
⟨*proof*⟩

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* = (∀ *X*∈*P* . ∀ *Y*∈ *P* . (*X* ∩ *Y* ≠ {} ⟷ *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* ∧ *is-non-overlapping P*)

No partition of a non-empty set is empty.

lemma *non-empty-imp-non-empty-partition*:
  assumes $A \neq \{\}$
    and *P partitions A*
  shows $P \neq \{\}$
  ⟨*proof*⟩

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

lemma *elem-in-partition*:
  assumes *in-set*: $x \in A$
    and *part*: *P partitions A*
  obtains *X* **where** $x \in X$ **and** $X \in P$
  ⟨*proof*⟩

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

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

⟨*proof*⟩

Every element of a partitioned set ends up in exactly one set.

lemma *elem-in-uniq-set*:
  assumes *in-set*: $x \in A$
    and *part*: *P partitions A*
  shows $\exists!\ X \in P\ .\ x \in X$
⟨*proof*⟩

A non-empty set "is" a partition of itself.

lemma *set-partitions-itself*:
  assumes $A \neq \{\}$
  shows $\{A\}$ *partitions A* ⟨*proof*⟩

The empty set is a partition of the empty set.

lemma *emptyset-part-emptyset1*:
  shows $\{\}$ *partitions* $\{\}$
  ⟨*proof*⟩

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

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

lemma *non-overlapping-extension2*:
fixes *new-el*::*'a*
and *P*::*'a set set*
and *X*::*'a set*
assumes *non-overlapping*: *is-non-overlapping P*
and *class-element*: *X* ∈ *P*
and *new*: *new-el* ∉ ⋃ *P*
shows *is-non-overlapping* (*insert-into-member new-el P X*)
⟨*proof*⟩

inserts an element into a specified set inside the given list of sets – the list variant of *insert-into-member*

The rationale for this variant and for everything that depends on it is: While it is possible to computationally enumerate "all partitions of a set" as an *'a set set set*, we need a list representation to apply further computational functions to partitions. Because of the way we construct partitions (using functions such as *all-coarser-partitions-with* below) it is not sufficient to simply use *'a set set list*, but we need *'a set list list*. This is because it is hard to impossible to convert a set to a list, whereas it is easy to convert a *list* to a *set*.

definition *insert-into-member-list* :: *'a* ⇒ *'a set list* ⇒ *'a set* ⇒ *'a set list*
where *insert-into-member-list new-el Sets S* = (*S* ∪ {*new-el*}) # (*remove1 S Sets*)

insert-into-member-list and *insert-into-member* are equivalent (as in returning the same set).

lemma *insert-into-member-list-equivalence*:
fixes *new-el*::*'a*
and *Sets*::*'a set list*
and *S*::*'a set*
assumes *distinct Sets*
shows *set* (*insert-into-member-list new-el Sets S*) = *insert-into-member new-el* (*set Sets*) *S*
⟨*proof*⟩

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

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

⟨*proof*⟩

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

definition *coarser-partitions-with* ::*'a* ⇒ *'a set set* ⇒ *'a set set set*
where *coarser-partitions-with new-el P* =
insert
— Let P be a partition of a set *Set*,
— and suppose *new-el* $\notin$ *Set*, i.e. $\{new\text{-}el\} \notin P$,
— then the following constructs a partition of *Set* ∪ $\{new\text{-}el\}$ obtained by
— inserting a new class $\{new\text{-}el\}$ and leaving all previous classes unchanged.
(*insert* $\{new\text{-}el\}$ *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* ∪ $\{new\text{-}el\}$ obtained by
— inserting *new-el* into one class of P at a time.
((*insert-into-member new-el P*) ‘ *P*)

the list variant of *coarser-partitions-with*

definition *coarser-partitions-with-list* ::*'a* ⇒ *'a set list* ⇒ *'a set list list*
where *coarser-partitions-with-list new-el P* =
— Let P be a partition of a set *Set*,
— and suppose *new-el* $\notin$ *Set*, i.e. $\{new\text{-}el\} \notin$ *set P*,

— then the following constructs a partition of $Set \cup \{new\text{-}el\}$ obtained by
— inserting a new class $\{new\text{-}el\}$ and leaving all previous classes unchanged.
$(\{new\text{-}el\} \# P)$
#
— Let P be a partition of a set Set,
— and suppose $new\text{-}el \notin Set$,
— then the following constructs
— the set of those partitions of $Set \cup \{new\text{-}el\}$ obtained by
— inserting $new\text{-}el$ into one class of P at a time.
$(map\ ((insert\text{-}into\text{-}member\text{-}list\ new\text{-}el\ P))\ P)$

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

lemma *coarser-partitions-with-list-equivalence*:
assumes *distinct P*
shows *set* (*map set* (*coarser-partitions-with-list new-el P*)) =
coarser-partitions-with new-el (*set P*)
⟨*proof*⟩

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

lemma *non-overlapping-extension3*:
fixes *elem*::'*a*
and *P*::'*a set set*
and *Q*::'*a set set*
assumes *P-non-overlapping*: *is-non-overlapping P*
and *new-elem*: *elem* ∉ ⋃ *P*
and *Q-coarser*: *Q* ∈ *coarser-partitions-with elem P*
shows *is-non-overlapping Q*
⟨*proof*⟩

Let 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* ∈ *coarser-partitions-with elem P*
shows ⋃ *Q* = *insert elem* (⋃ *P*)
⟨*proof*⟩

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* = (λ*X* . *X* − {*elem*}) ‘ *P* − {{}}

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

lemma *partition-without-covers*:
  fixes *elem*::*'a*
    and *P*::*'a set set*
  shows ⋃ (*partition-without elem P*) = (⋃ *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* ∈ *partition-without elem P*
  obtains *Z* **where** *Z* ∈ *P* **and** *X* = *Z* − {*elem*}
⟨*proof*⟩

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

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

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* ∈ *coarser-partitions-with elem* (*partition-without elem P*)
    (**is** *P* ∈ *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* = ⋃ (*coarser-partitions-with elem* ‘ *Ps*)

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

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

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

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

lemma *all-coarser-partitions-with-list-equivalence*:
fixes *elem*::'*a*
and *Ps*::'*a set list list*
assumes *distinct*: ∀ *P* ∈ *set Ps* . *distinct P*
shows *set* (*map set* (*all-coarser-partitions-with-list elem Ps*)) = *all-coarser-partitions-with elem* (*set* (*map set Ps*))
(**is** *?list-expr* = *?set-expr*)
⟨*proof*⟩

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

fun *all-partitions-set* :: '*a list* ⇒ '*a set set set*
where
all-partitions-set [] = {{}} |
all-partitions-set (*e* # *X*) = *all-coarser-partitions-with e* (*all-partitions-set X*)

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

fun *all-partitions-list* :: '*a list* ⇒ '*a set list list*
where
all-partitions-list [] = [[]] |
all-partitions-list (*e* # *X*) = *all-coarser-partitions-with-list e* (*all-partitions-list X*)

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

lemma *coarser-partitions-with-list-distinct*:
fixes *ps*
assumes *ps-coarser*: *ps* ∈ *set* (*coarser-partitions-with-list x Q*)
and *distinct*: *distinct Q*
and *partition*: *is-non-overlapping* (*set Q*)
and *new*: {*x*} ∉ *set Q*
shows *distinct ps*
⟨*proof*⟩

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

lemma *all-partitions-equivalence'*:
fixes *xs*::'*a list*
shows *distinct xs* ⟹
((*set* (*map set* (*all-partitions-list xs*)) =
all-partitions (*set xs*)) ∧ (∀ *ps* ∈ *set* (*all-partitions-list xs*) . *distinct ps*))
⟨*proof*⟩

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* ⟹ *set* (*map set* (*all-partitions-list ?xs*)) = *all-partitions*

(*set ?xs*) ∧ (∀ *ps*∈*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* ⟹ *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* ⇒ '*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** ∀ *x xs*. *P* (*xs*) ⟶ *P* (*x*#*xs*)
shows *P l*
⟨*proof*⟩

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

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

lemma *argmaxLemma*: *argmax f A* = { *x* ∈ *A* . *f x* = *Max* (*f* ‘ *A*) }
⟨*proof*⟩

lemma *maxLemma*:
assumes *x* ∈ *X finite X*
shows *Max* (*f*‘*X*) >= *f x*
(**is** *?L* >= *?R*) ⟨*proof*⟩

lemma *lm01*:
argmax f A = *A* ∩ *f* −‘ {*Max* (*f* ‘ *A*)}
⟨*proof*⟩

lemma *lm02*:
assumes *y* ∈ *f*‘*A*

shows $A \cap f -` \{y\} \neq \{\}$
⟨*proof*⟩

lemma *argmaxEquivalence*:
assumes $\forall x{\in}X.\ f\ x = g\ x$
shows *argmax f X* = *argmax g X*
⟨*proof*⟩

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

corollary *argmax-non-empty-iff*: **assumes** *finite X X* ≠ {}
shows *argmax f X* ≠{}
⟨*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* => *bool*) => '*a list* => *nat list*
where *filterpositions P l* = *map snd* (*filter* (*P o fst*) (*zip l* (*upt 0* (*size l*))))

definition *filterpositions2*
where *filterpositions2 P l* = [*n*. *n* ← [*0*..<*size l*], *P* (*l*!*n*)]

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

lemma *lm03*: *maxpositions l* = [*n*. *n*←[*0*..<*size l*], *l*!*n* ≥ *Max*(*set l*)]
⟨*proof*⟩

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

lemma *lm04*: [*n* . *n* <− *l*, *P n*] = [*n* . *n* <− *l*, *n* ∈ *set l*, *P n*]
⟨*proof*⟩

lemma *lm05*: [*n* . *n* <− [*0*..<*m*], *P n*] = [*n* . *n* <− [*0*..<*m*], *n* ∈ *set* [*0*..<*m*], *P n*]
⟨*proof*⟩

lemma *lm06*: **fixes** *f m P*
shows (*map f* [*n* . *n* <− [*0*..<*m*], *P n*]) = [*f n* . *n* <− [*0*..<*m*], *P n*]
⟨*proof*⟩

lemma *map-commutes-a*: [*f n* . *n* <− [], *Q* (*f n*)] = [*x* <− (*map f* []). *Q x*]
⟨*proof*⟩

lemma *map-commutes-b*: ∀ *x xs*. ([*f n* . *n* <− *xs*, *Q* (*f n*)] = [*x* <− (*map f xs*). *Q x*] ⟶
[*f n* . *n* <− (*x*#*xs*), *Q* (*f n*)] = [*x* <− (*map f* (*x*#*xs*)). *Q x*])
⟨*proof*⟩

lemma *map-commutes*: **fixes** *f*::*'a* => *'b* **fixes** *Q*::*'b* => *bool* **fixes** *xs*::*'a list*
shows [*f n* . *n* <− *xs*, *Q* (*f n*)] = [*x* <− (*map f xs*). *Q x*]
⟨*proof*⟩

lemma *lm07*: **fixes** *f l*
shows *maxpositions* (*map f l*) =
[*n* . *n* <− [*0*..<*size l*], *f* (*l*!*n*) ≥ *Max* (*f*‘(*set l*))]
(**is** *maxpositions* (*?fl*) = -)
⟨*proof*⟩

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

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

theorem *argmaxadequacy*: **fixes** *f*::*'a* => (*'b*::*linorder*) **fixes** *l*::*'a list*
shows *argmaxList f l* = [*x* <− *l*. *f x* ≥ *Max* (*f*‘(*set l*))]
(**is** *?lh*=-)
⟨*proof*⟩

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* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a \Rightarrow 'b \Rightarrow 'b$
where *eval-rel-or R a z* = (*let im* = *R* `` {*a*} *in if card im* = *1 then the-elem im else z*)

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

definition *runiq* :: $('a \times 'b)$ *set* $\Rightarrow$ *bool*
where *runiq R* = ($\forall$ *X* . *trivial X* $\longrightarrow$ *trivial* (*R* `` *X*))

5.2 Restriction

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

definition *restrict* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $('a \times 'b)$ *set* (**infix** $\langle||\rangle$ *75*)
where *R* || *X* = (*X* $\times$ *Range R*) $\cap$ *R*

extensional characterization of the pairs within a restricted relation

lemma *restrict-ext*: *R* || *X* = {(*x*, *y*) | *x y* . *x* $\in$ *X* $\wedge$ (*x*, *y*) $\in$ *R*}
⟨*proof*⟩

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

lemma *restrict-ext'*: *R* || *X* = {*p* . *fst p* $\in$ *X* $\wedge$ *p* $\in$ *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* $\subseteq$ *P*
⟨*proof*⟩

Restricting a relation only has an effect within its domain.

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

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

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

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

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

Considering a relation outside a singleton set $\{x\}$ reduces its domain by x.

corollary *Domain-outside-singleton*:
  assumes *Domain* $R =$ *insert* x A
    and $x \notin A$
  shows *Domain* $(R$ *outside* $\{x\}) = A$
  ⟨*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*: $P = (P$ *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*: *Range* $(R$ *outside* $X) \subseteq R$ `` (*Domain* $R - X)$
  ⟨*proof*⟩

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

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

5.4 Flipping pairs of relations

flipping a pair: exchanging first and second component

definition *flip* **where** *flip tup* = (*snd tup*, *fst tup*)

Flipped pairs can be found in the converse relation.

lemma *flip-in-conv*:
  assumes *tup* $\in R$
  shows *flip tup* $\in R^{-1}$
  ⟨*proof*⟩

Flipping a pair twice doesn't change it.

lemma *flip-flip*: *flip* (*flip tup*) = *tup*
⟨*proof*⟩

Flipping all pairs in a relation yields the converse relation.

lemma *flip-conv*: *flip* ‘ $R = R^{-1}$
⟨*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* :: $('a \times 'b)$ *set* ⇒ $'a$ ⇒ $'b$ (**infix** ‹,,› 75)
where R ,, a = *the-elem* (R ‘‘ {a})

5.6 Paste

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

definition *paste* (**infix** ‹+∗› 75)
where P +∗ Q = (P *outside Domain* Q) ∪ Q

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

lemma *paste-subrel*:
assumes P || *Domain* $Q \subseteq Q$
shows P +∗ $Q = P \cup Q$
⟨*proof*⟩

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

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

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

lemma *paste-outside-restrict*: P = (P *outside* X) +∗ (P || X)
⟨*proof*⟩

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

lemma *paste-Domain*: *Domain*(P +∗ Q)=*Domain* P∪*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*: $Range\ (P +* Q) \subseteq Range\ P \cup Range\ Q$
⟨*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*: $card\ P = card\ (P^{\frown}{-}1)$
⟨*proof*⟩

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

lemma *lm02*: $trivial\ X = (X{=}\{\} \vee card\ X{=}1)$
⟨*proof*⟩

lemma *lm03*: $trivial\ P = trivial\ (P^{\frown}{-}1)$
⟨*proof*⟩

lemma *restrictedRange*: $Range\ (P||X) = P`` X$
⟨*proof*⟩

lemma *doubleRestriction*: $((P\ ||\ X)\ ||\ Y) = (P\ ||\ (X \cap Y))$
⟨*proof*⟩

lemma *restrictedDomain*: $Domain\ (R||X) = Domain\ R \cap X$
⟨*proof*⟩

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

lemma *subrel-runiq*:

assumes *runiq Q P* $\subseteq$ *Q*
shows *runiq P*
⟨*proof*⟩

lemma *rightUniqueInjectiveOnFirstImplication*:
assumes *runiq P*
shows *inj-on fst P*
⟨*proof*⟩

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

lemma *runiq-alt*: *runiq R* $\longleftrightarrow$ (∀ *x* . *trivial* (*R* `` {*x*}))
⟨*proof*⟩

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

lemma *runiq-wrt-eval-rel*: *runiq R* = (∀ *x* . *R* `` {*x*} $\subseteq$ {*R* ,, *x*})
⟨*proof*⟩

lemma *rightUniquePair*:
assumes *runiq f*
assumes (*x*,*y*)∈*f*
shows *y*=*f*,,*x*
⟨*proof*⟩

lemma *runiq-basic*: *runiq R* $\longleftrightarrow$ (∀ *x y y'* . (*x*, *y*) ∈ *R* ∧ (*x*, *y'*) ∈ *R* $\longrightarrow$ *y* = *y'*)

⟨*proof*⟩

lemma *rightUniqueFunctionAfterInverse*:
assumes *runiq f*
shows *f*``(*f*^−1``*Y*) $\subseteq$ *Y*
⟨*proof*⟩

lemma *lm04*:
assumes *runiq f y1* ∈ *Range f*
shows (*f*^−1 `` {*y1*} ∩ *f*^−1 `` {*y2*} ≠ {}) = (*f*^−1``{*y1*}=*f*^−1``{*y2*})
⟨*proof*⟩

lemma *converse-Image*:
assumes *runiq*: *runiq R*
and *runiq-conv*: *runiq* (*R*^−1)
shows (*R*^−1) `` *R* `` *X* $\subseteq$ *X*
⟨*proof*⟩

lemma *lm05*:
assumes *inj-on fst P*
shows *runiq P*
⟨*proof*⟩

lemma *rightUniqueInjectiveOnFirst*: (*runiq P*) = (*inj-on fst P*)
⟨*proof*⟩

lemma *disj-Un-runiq*:
assumes *runiq P runiq Q* (*Domain P*) ∩ (*Domain Q*) = {}
shows *runiq* (*P* ∪ *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*)| *x*. *P x*}
⟨*proof*⟩

lemma *rightUniqueSetCardinality*:
assumes *x* ∈ *Domain R runiq R*
shows *card* (*R*‘‘{*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* ∈ *Domain R runiq R*
shows *R* ‘‘ {*x*} = {*R* ,, *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*:

runiq R $\longleftrightarrow$ ($\forall$ *a* $\in$ *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 ,, x on some argument x in its domain, occurs in R's range. Note that we need runiq R in order to get a definite value for R ,, x

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

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

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

lemma *runiq-converse-paste-singleton*:
assumes *runiq* (*P*^−1) *y*∉(*Range P*)
shows *runiq* ((*P* +∗ {(*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* $X\ Y = \{R\ .\ Domain\ R = X \wedge Range\ R \subseteq Y \wedge runiq\ R \wedge runiq\ (R^{-1})\}$

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

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

lemma *Image-within-domain'*:
fixes x R
shows ($x \in Domain\ R$) = (R `` {x} ≠ {})
⟨*proof*⟩

end

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

theory *MiscTools*

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

begin

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

7.1 Facts and notations about relations, sets and functions.

notation *paste* (**infix** ‹+<› 75)

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

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

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

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

Turns a HOL function into a set-theoretical function

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

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

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

lemma
  toFunction = *eval-rel*
  ⟨*proof*⟩

lemma *lm001*:
  (($P \cup Q$) || X) = ((P || X) ∪ (Q||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 || (*Domain* P))
  notation *update* (**infix** ‹+⌢› 75)

definition *runiqer* :: ('a × 'b) *set* => ('a × 'b) *set*
  where *runiqer* R = { (x, *THE* y. $y \in R$ `` {x})| 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 || X is not computable. Duplicates the eponymous definition found in *Function-Order*, which is otherwise not needed.

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

lemma *lm002*:

assumes *runiq R*
shows $R \supseteq$ *graph* (*Domain R*) (*toFunction R*)
⟨*proof*⟩

lemma *lm003*:
assumes *runiq R*
shows $R \subseteq$ *graph* (*Domain R*) (*toFunction R*)
⟨*proof*⟩

lemma *lm004*:
assumes *runiq R*
shows $R =$ *graph* (*Domain R*) (*toFunction R*)
⟨*proof*⟩

lemma *domainOfGraph*:
runiq(*graph X f*) & *Domain*(*graph X f*)=*X*
⟨*proof*⟩

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

lemma *imageEquivalence*:
assumes *runiq* (*f*::(('*a* × ('*b set*)) *set*)) $x \in$ *Domain f*
shows *f*,,*x* = *f*,,,*x*
⟨*proof*⟩

lemma *lm005*:
Graph f=*graph UNIV f*
⟨*proof*⟩

lemma *graphIntersection*:
graph ($X \cap Y$) *f* $\subseteq$ ((*graph X f*) || *Y*)
⟨*proof*⟩

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

lemma *outsideOutside*:
((*P outside X*) *outside Y*) = *P outside* ($X \cup Y$)
⟨*proof*⟩

corollary *lm006*:
((*P outside X*) *outside X*) = *P outside X*
⟨*proof*⟩

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

shows $P +\!\!* Q = (P \ outside \ X) +\!\!* Q$
⟨*proof*⟩

corollary *lm008*:
$P +\!\!* Q = (P \ outside \ (Domain \ Q)) +\!\!* Q$
⟨*proof*⟩

corollary *outsideUnion*:
$R = (R \ outside \ \{x\}) \cup (\{x\} \times (R \ `` \ \{x\}))$
⟨*proof*⟩

lemma *lm009*:
$P = P \cup \{x\} \times P``\{x\}$
⟨*proof*⟩

corollary *lm010*:
$R = (R \ outside \ \{x\}) +\!\!* (\{x\} \times (R \ `` \ \{x\}))$
⟨*proof*⟩

lemma *lm011*:
$R \subseteq R +\!\!* (\{x\} \times (R``\{x\}))$
⟨*proof*⟩

lemma *lm012*:
$R \supseteq R +\!\!* (\{x\} \times (R``\{x\}))$
⟨*proof*⟩

lemma *lm013*:
$R = R +\!\!* (\{x\} \times (R``\{x\}))$
⟨*proof*⟩

lemma *rightUniqueTrivialCartes*:
assumes *trivial* Y
shows *runiq* $(X \times Y)$
⟨*proof*⟩

lemma *lm014*:
runiq $((X \times \{x\}) +\!\!* (Y \times \{y\}))$
⟨*proof*⟩

lemma *lm015*:
$(P \ || \ (X \cap Y)) \subseteq (P||X) \quad \& \quad P \ outside \ (X \cup Y) \subseteq P \ outside \ X$
⟨*proof*⟩

lemma *lm016*:
$P \ || \ X \subseteq (P||(X \cup Y)) \quad \& \quad P \ outside \ X \subseteq P \ outside \ (X \cap Y)$
⟨*proof*⟩

lemma *lm017*:
$P`\!`(X \cap Domain\ P) = P`\!`X$
⟨*proof*⟩

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

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

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

7.2 Ordered relations

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

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

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

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

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

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

⟨*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*}) ‘ *X*

lemma *finestPart*:
finestpart X = {{*x*}|*x* . *x*∈*X*}
⟨*proof*⟩

lemma *finestPartUnion*:
X=$\bigcup$ (*finestpart X*)
⟨*proof*⟩

lemma *lm026*:
Union ∘ *finestpart* = *id*
⟨*proof*⟩

lemma *lm027*:
inj-on Union (*finestpart* ‘ *UNIV*)
⟨*proof*⟩

lemma *nonEqualitySetOfSets*:
assumes *X* ≠ *Y*
shows {{*x*}| *x*. *x* ∈ *X*} ≠ {{*x*}| *x*. *x* ∈ *Y*}
⟨*proof*⟩

corollary *lm028*:
inj-on finestpart UNIV
⟨*proof*⟩

lemma *unionFinestPart*:
{*Y* | *Y*. ∃ *x*.((*Y* ∈ *finestpart x*) ∧ (*x* ∈ *X*))} = $\bigcup$(*finestpart*‘*X*)

⟨*proof*⟩

lemma *rangeSetOfPairs*:
Range {(*fst pair*, *Y*)| *Y pair*. *Y* ∈ *finestpart* (*snd pair*) & *pair* ∈ *X*} =
{*Y*. ∃ *x*. ((*Y* ∈ *finestpart x*) ∧ (*x* ∈ *Range X*))}
⟨*proof*⟩

lemma *setOfPairsEquality*:
{(*fst pair*, {*y*})| *y pair*. *y* ∈ *snd pair* & *pair* ∈ *X*} =
{(*fst pair*, *Y*)| *Y pair*. *Y* ∈ *finestpart* (*snd pair*) & *pair* ∈ *X*}
⟨*proof*⟩

lemma *setOfPairs*:
{(*fst pair*, {*y*})| *y*. *y* ∈ *snd pair*} =
{*fst pair*} × {{*y*}| *y*. *y* ∈ *snd pair*}
⟨*proof*⟩

lemma *lm029*:
x ∈ *X* = ({*x*} ∈ *finestpart X*)
⟨*proof*⟩

lemma *pairDifference*:
{(*x*,*X*)}−{(*x*, *Y*)} = {*x*}×({*X*}−{*Y*})
⟨*proof*⟩

lemma *lm030*:
assumes ⋃ *P* = *X*
shows *P* ⊆ *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* ∩ *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 ⊆ Domain Q*)
shows (*P +∗ Q*) = *Q*
⟨*proof*⟩

lemma *lm034*:
assumes (*P +∗ Q=Q*)
shows (*Domain P ⊆ Domain Q*)
⟨*proof*⟩

lemma *lm035*:
(*Domain P ⊆ 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*) ⊆ *P* || ((*Domain P*)−*X*)
⟨*proof*⟩

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

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

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

lemma *lm041*:
P outside X = *P* || ((*Domain P*) − *X*)
⟨*proof*⟩

lemma *lm042*:
R‘‘(*X*−*Y*) = (*R||X*)*‘‘*(*X*−*Y*)
⟨*proof*⟩

lemma *lm043*:
assumes $\bigcup\ XX \subseteq X\ x \in XX\ x \neq \{\}$
shows $x \cap X \neq \{\}$
⟨*proof*⟩

lemma *lm044*:
assumes $\forall\, l \in set\ L1.\ set\ L2 = f2\ (set\ l)\ N$
shows $set\ [set\ L2.\ l <-\ L1]\ =\ \{f2\ P\ N|\ P.\ P \in set\ (map\ set\ L1)\}$
⟨*proof*⟩

lemma *setVsList*:
assumes $\forall\, l \in set\ (g1\ G).\ set\ (g2\ l\ N) = f2\ (set\ l)\ N$
shows $set\ [set\ (g2\ l\ N).\ l <-\ (g1\ G)]\ =\ \{f2\ P\ N|\ P.\ P \in set\ (map\ set\ (g1\ G))\}$
⟨*proof*⟩

lemma *lm045*:
$(\forall\, l \in set\ (g1\ G).\ set\ (g2\ l\ N) = f2\ (set\ l)\ N) \longrightarrow$
$\{f2\ P\ N|\ P.\ P \in set\ (map\ set\ (g1\ G))\} = set\ [set\ (g2\ l\ N).\ l <-\ g1\ G]$
⟨*proof*⟩

lemma *lm046*:
assumes $X \cap Y\ =\ \{\}$
shows $R``X = (R\ outside\ Y)``X$
⟨*proof*⟩

lemma *lm047*:
assumes $(Range\ P) \cap (Range\ Q) = \{\}\ runiq\ (P^{-1})\ runiq\ (Q^{-1})$
shows $runiq\ ((P \cup Q)^{-1})$
⟨*proof*⟩

lemma *lm048*:
assumes $(Range\ P) \cap (Range\ Q) = \{\}\ runiq\ (P^{-1})\ runiq\ (Q^{-1})$
shows $runiq\ ((P +\!\!* Q)^{-1})$
⟨*proof*⟩

lemma *lm049*:
assumes *runiq R*
shows $card\ (R\ ``\ \{a\}) = 1 \longleftrightarrow a \in Domain\ R$
⟨*proof*⟩

lemma *lm050*:
$inj\ (\lambda a.\ ((fst\ a, fst\ (snd\ a)), snd\ (snd\ a)))$
⟨*proof*⟩

lemma *lm051*:
assumes *finite X x > Max X*
shows $x \notin X$
⟨*proof*⟩

lemma *lm052*:
assumes *finite A* $A \neq \{\}$
shows *Max* (*f*‘*A*) ∈ *f*‘*A*
⟨*proof*⟩

lemma *lm053*:
argmax f A ⊆ *f* −‘ {*Max* (*f* ‘ *A*)}
⟨*proof*⟩

lemma *lm054*:
argmax f A = *A* ∩ { *x* . *f x* = *Max* (*f* ‘ *A*) }
⟨*proof*⟩

lemma *lm055*:
(*x* ∈ *argmax f X*) = (*x* ∈ *X* & *f x* = *Max* (*f* ‘ *X*))
⟨*proof*⟩

lemma *rangeEmpty*:
Range −‘ {{}} = {{}}
⟨*proof*⟩

lemma *finitePairSecondRange*:
(∀ *pair* ∈ *R*. *finite* (*snd pair*)) = (∀ *y* ∈ *Range R*. *finite y*)
⟨*proof*⟩

lemma *lm056*:
fst ‘ *P* = *snd* ‘ (*P*^−*1*)
⟨*proof*⟩

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

lemma *flip-flip2*:
flip ∘ *flip* = *id*
⟨*proof*⟩

lemma *lm058*:
fst = (*snd*∘*flip*)
⟨*proof*⟩

lemma *lm059*:

snd = (*fst*∘*flip*)
⟨*proof*⟩

lemma *lm060*:
inj-on fst P = *inj-on* (*snd*∘*flip*) *P*
⟨*proof*⟩

lemma *lm062*:
inj-on fst P = *inj-on snd* (*P*^−1)
⟨*proof*⟩

lemma *sumPairsInverse*:
assumes *runiq* (*P*^−1)
shows *sum* (*f* ∘ *snd*) *P* = *sum f* (*Range P*)
⟨*proof*⟩

lemma *notEmptyFinestpart*:
assumes *X* ≠ {}
shows *finestpart X* ≠ {}
⟨*proof*⟩

lemma *lm063*:
assumes *inj-on g X*
shows *sum f* (*g*‘*X*) = *sum* (*f* ∘ *g*) *X*
⟨*proof*⟩

lemma *functionOnFirstEqualsSecond*:
assumes *runiq R z* ∈ *R*
shows *R*,,(*fst z*) = *snd z*
⟨*proof*⟩

lemma *lm064*:
assumes *runiq R*
shows *sum* (*toFunction R*) (*Domain R*) = *sum snd R*
⟨*proof*⟩

corollary *lm065*:
assumes *runiq* (*f*||*X*)
shows *sum* (*toFunction* (*f*||*X*)) (*X* ∩ *Domain f*) = *sum snd* (*f*||*X*)
⟨*proof*⟩

lemma *lm066*:
Range (*R outside X*) = *R*‘‘((*Domain R*) − *X*)
⟨*proof*⟩

lemma *lm067*:
(*R*||*X*) ‘‘ *X* = *R*‘‘*X*
⟨*proof*⟩

lemma *lm068*:
assumes $x \in Domain\ (f||X)$
shows $(f||X)``\{x\} = f``\{x\}$
⟨*proof*⟩

lemma *lm069*:
assumes $x \in X \cap Domain\ f\ runiq\ (f||X)$
shows $(f||X),,x = f,,x$
⟨*proof*⟩

lemma *lm070*:
assumes $runiq\ (f||X)$
shows $sum\ (toFunction\ (f||X))\ (X \cap Domain\ f) = sum\ (toFunction\ f)\ (X \cap Domain\ f)$
⟨*proof*⟩

corollary *sumRestrictedToDomainInvariant*:
assumes $runiq\ (f||X)$
shows $sum\ (toFunction\ f)\ (X \cap Domain\ f) = sum\ snd\ (f||X)$
⟨*proof*⟩

corollary *sumRestrictedOnFunction*:
assumes $runiq\ (f||X)$
shows $sum\ (toFunction\ (f||X))\ (X \cap Domain\ f) = sum\ snd\ (f||X)$
⟨*proof*⟩

lemma *cardFinestpart*:
$card\ (finestpart\ X) = card\ X$
⟨*proof*⟩

corollary *lm071*:
$finestpart\ \{\} = \{\} \quad \& \quad card \circ finestpart = card$
⟨*proof*⟩

lemma *finiteFinestpart*:
$finite\ (finestpart\ X) = finite\ X$
⟨*proof*⟩

lemma *lm072*:
$finite \circ finestpart = finite$
⟨*proof*⟩

lemma *finestpartSubset*:
assumes $X \subseteq Y$
shows $finestpart\ X \subseteq finestpart\ Y$
⟨*proof*⟩

corollary *lm073*:
assumes $x \in X$

shows *finestpart x* $\subseteq$ *finestpart* ($\bigcup$ *X*)
⟨*proof*⟩

lemma *lm074*:
$\bigcup$ (*finestpart* ‘ *XX*) $\subseteq$ *finestpart* ($\bigcup$ *XX*)
⟨*proof*⟩

lemma *lm075*:
$\bigcup$ (*finestpart* ‘ *XX*) $\supseteq$ *finestpart* ($\bigcup$ *XX*)
(**is** *?L* $\supseteq$ *?R*)
⟨*proof*⟩

corollary *commuteUnionFinestpart*:
$\bigcup$ (*finestpart* ‘ *XX*) = *finestpart* ($\bigcup$ *XX*)
⟨*proof*⟩

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

lemma *lm076*:
assumes *runiq P*
shows *card* (*Domain P*) = *card P*
⟨*proof*⟩

lemma *finiteDomainImpliesFinite*:
assumes *runiq f*
shows *finite* (*Domain f*) = *finite f*
⟨*proof*⟩

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

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

corollary *lm078*:
assumes *finite X*
shows *sum f X* = *sum f* (*X*−*Y*) + (*sum f* (*X* $\cap$ *Y*))
⟨*proof*⟩

lemma *lm079*:
(*P* +∗ *Q*)‘‘(*Domain Q*∩*X*) = *Q*‘‘(*Domain Q*∩*X*)
⟨*proof*⟩

corollary *lm080*:
$(P +* Q)``(X \cap (Domain\ Q)) = Q``X$
⟨*proof*⟩

corollary *lm081*:
assumes $X \cap (Domain\ Q) = \{\}$
shows $(P +* Q)\ ``\ X = (P\ outside\ (Domain\ Q))``\ X$
⟨*proof*⟩

lemma *lm082*:
assumes $X \cap Y = \{\}$
shows $(P\ outside\ Y)``X = P``X$
⟨*proof*⟩

corollary *lm083*:
assumes $X \cap (Domain\ Q) = \{\}$
shows $(P +* Q)``X = P``X$
⟨*proof*⟩

lemma *lm084*:
assumes *finite X finite Y* $card(X \cap Y) = card\ X$
shows $X \subseteq Y$
⟨*proof*⟩

lemma *cardinalityIntersectionEquality*:
assumes *finite X finite Y* $card\ X = card\ Y$
shows $(card\ (X \cap Y) = card\ X) = (X = Y)$
⟨*proof*⟩

lemma *lm085*:
assumes $P\ xx$
shows $\{(x, f\ x) |\ x.\ P\ x\},,xx = f\ xx$
⟨*proof*⟩

lemma *graphEqImage*:
assumes $x \in X$
shows $graph\ X\ f,,x = f\ x$
⟨*proof*⟩

lemma *lm086*:
$Graph\ f,,x = f\ x$
⟨*proof*⟩

lemma *lm087*:
$toFunction\ (Graph\ f) = f$ (**is** *?L=-*)
⟨*proof*⟩

lemma *lm088*:

R *outside* $X \subseteq R$
⟨*proof*⟩

lemma *lm089*:
Range(f *outside* X) $\supseteq$ (*Range* f)$-$(f‘‘X)
⟨*proof*⟩

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

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

lemma *lm092*:
assumes *runiq* f *runiq* (f^-1)
shows *Range*(f *outside* X) $\subseteq$ (*Range* f)$-$(f‘‘X)
⟨*proof*⟩

lemma *rangeOutside*:
assumes *runiq* f *runiq* (f^-1)
shows *Range*(f *outside* X) $=$ (*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 $R \neq \{\}$ *Domain* $R \cap X \neq \{\}$
shows R‘‘$X \neq \{\}$
⟨*proof*⟩

lemma *lm095*:
$R \subseteq$ (*Domain* R) $\times$ (*Range* R)
⟨*proof*⟩

lemma *finiteRelationCharacterization*:
(*finite* (*Domain* Q) & *finite* (*Range* Q)) = *finite* Q
⟨*proof*⟩

lemma *familyUnionFiniteEverySetFinite*:
assumes *finite* ($\bigcup$ *XX*)
shows $\forall X \in XX.$ *finite X*
⟨*proof*⟩

lemma *lm096*:
assumes *runiq f X* $\subseteq$ (*f*$^{-}$*−1*)‘‘*Y*
shows *f*‘‘*X* $\subseteq$ *Y*
⟨*proof*⟩

lemma *lm097*:
assumes *y* $\in$ *f*‘‘{*x*} *runiq f*
shows *f*,,*x* = *y*
⟨*proof*⟩

7.3 Indicator function in set-theoretical form.

abbreviation
Outside′ X f == *f outside X*

abbreviation
Chi X Y == (*Y* × {*0*::*nat*}) +∗ (*X* × {*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*) ‘‘ {*x*}
⟨*proof*⟩

lemma *lm100*:
assumes *x* $\in$ *Y*−*X*
shows *0* $\in$ (*X* <|| *Y*) ‘‘ {*x*}
⟨*proof*⟩

lemma *lm101*:
assumes *x* $\in$ *X* $\cup$ *Y*
shows (*X* <|| *Y*),,*x* = *chi X x* (**is** *?L*=*?R*)

⟨*proof*⟩

lemma *lm102*:
assumes $x \in X \cup Y$
shows $(X <\mid Y)\ x = chi\ X\ x$
⟨*proof*⟩

corollary *lm103*:
$sum\ (X <\mid Y)\ (X \cup Y) = sum\ (chi\ X)\ (X \cup Y)$
⟨*proof*⟩

corollary *lm104*:
assumes $\forall\, x \in X.\ f\ x = g\ x$
shows $sum\ f\ X = sum\ g\ X$
⟨*proof*⟩

corollary *lm105*:
assumes $\forall\, x \in X.\ f\ x = g\ x\ Y \subseteq X$
shows $sum\ f\ Y = sum\ g\ Y$
⟨*proof*⟩

corollary *lm106*:
assumes $Z \subseteq X \cup Y$
shows $sum\ (X <\mid Y)\ Z = sum\ (chi\ X)\ Z$
⟨*proof*⟩

corollary *lm107*:
$sum\ (chi\ X)\ (Z - X) = 0$
⟨*proof*⟩

corollary *lm108*:
assumes $Z \subseteq X \cup Y$
shows $sum\ (X <\mid Y)\ (Z - X) = 0$
⟨*proof*⟩

corollary *lm109*:
assumes *finite Z*
shows $sum\ (X <\mid Y)\ Z \quad = \quad sum\ (X <\mid Y)\ (Z - X) \quad + \quad (sum\ (X <\mid Y)\ (Z \cap X))$
⟨*proof*⟩

corollary *lm110*:
assumes $Z \subseteq X \cup Y$ *finite Z*
shows $sum\ (X <\mid Y)\ Z = sum\ (X <\mid Y)\ (Z \cap X)$
⟨*proof*⟩

corollary *lm111*:
assumes *finite Z*
shows $sum\ (chi\ X)\ Z = card\ (X \cap Z)$

⟨*proof*⟩

corollary *lm112*:
assumes $Z \subseteq X \cup Y$ *finite* Z
shows *sum* $(X <| Y)\ Z = card\ (Z \cap X)$
⟨*proof*⟩

corollary *subsetCardinality*:
assumes $Z \subseteq X \cup Y$ *finite* Z
shows $(sum\ (X <| Y)\ X) - (sum\ (X <| Y)\ Z) = card\ X - card\ (Z \cap X)$
⟨*proof*⟩

corollary *differenceSumVsCardinality*:
assumes $Z \subseteq X \cup Y$ *finite* Z
shows $int\ (sum\ (X <| Y)\ X) - int\ (sum\ (X <| Y)\ Z) = int\ (card\ X) - int\ (card\ (Z \cap X))$
⟨*proof*⟩

lemma *lm113*:
$int\ (n::nat) = real\ n$
⟨*proof*⟩

corollary *differenceSumVsCardinalityReal*:
assumes $Z \subseteq X \cup Y$ *finite* Z
shows $real\ (sum\ (X <| Y)\ X) - real\ (sum\ (X <| Y)\ Z) =$
$real\ (card\ X) - real\ (card\ (Z \cap X))$
⟨*proof*⟩

7.4 Lists

lemma *lm114*:
assumes $\exists\ n \in \{0..<size\ l\}.\ P\ (l!n)$
shows $[n.\ n \leftarrow [0..<size\ l],\ P\ (l!n)] \neq []$
⟨*proof*⟩

lemma *lm115*:
assumes $ll \in set\ (l::'a\ list)$
shows $\exists\ n \in (nth\ l) -` (set\ l).\ ll = l!n$
⟨*proof*⟩

lemma *lm116*:
assumes $ll \in set\ (l::'a\ list)$
shows $\exists\ n.\ ll = l!n\ \&\ n < size\ l\ \&\ n >= 0$
⟨*proof*⟩

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

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

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

corollary *lm119*:
$(nth\ l)` set\ (filterpositions2\ (\%x.(x{\in}X))\ l) \subseteq X \cap\ \ set\ l$
⟨*proof*⟩

lemma *lm120*:
$(n{\in}\{0..<N\}) = ((n{::}nat) < N)$
⟨*proof*⟩

lemma *lm121*:
assumes $X \subseteq \{0..<size\ list\}$
shows $(nth\ list)`X \subseteq set\ list$
⟨*proof*⟩

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

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

7.5 Computing all the permutations of a list

abbreviation
$rotateLeft == rotate$
abbreviation
$rotateRight\ n\ l == rotateLeft\ (size\ l - (n\ mod\ (size\ l)))\ l$

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

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

abbreviation
takeAll P list == map (nth list) (filterpositions2 P list)

lemma *permutationNotEmpty*:
assumes $l \neq []$
shows *perm2 l n* $\neq []$
⟨*proof*⟩

lemma *lm124*:
set (takeAll P list) = ((nth list) ' set (filterpositions2 P list))
⟨*proof*⟩

corollary *listIntersectionWithSet*:
set (takeAll (%x.(x∈X)) l) $\subseteq$ *(X* $\cap$ *set l)*
⟨*proof*⟩

corollary *lm125*:
set (takeAll P list) $\subseteq$ *set list*
⟨*proof*⟩

lemma *takeAllSubset*:
set (takeAll (%x. x∈ P) list) $\subseteq$ *P*
⟨*proof*⟩

lemma *lm126*:
set (insertAt x l n) $= \{x\} \cup$ *set l*
⟨*proof*⟩

lemma *lm127*:
$\forall$ *n. set (perm2 [] n) = set []*
⟨*proof*⟩

lemma *lm128*:
assumes $\forall$ *n. (set (perm2 l n) = set l)*
shows *set (perm2 (x#l) n)* $= \{x\} \cup$ *set l*
⟨*proof*⟩

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

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

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

7.6 A more computable version of *toFunction*.

abbreviation *toFunctionWithFallback R fallback* ==
(% *x*. *if* $(R``\{x\} = \{R,,x\})$ *then* $(R,,x)$ *else fallback*)
notation
toFunctionWithFallback (**infix** ‹*Else*› 75)

abbreviation *sum′* **where**
$sum'\ R\ X == sum\ (R\ Else\ 0)\ X$

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

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

⟨*proof*⟩

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

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

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

corollary *lm134*:

assumes *finite X*
shows *sum (f Else 0) X = sum (f Else 0) (X∩Domain f)*
⟨*proof*⟩

corollary *lm135*:
assumes *finite X*
shows *sum (f Else 0) (X∩Domain f) = sum (f Else 0) X*
(**is** *?L=?R*)
⟨*proof*⟩

corollary *lm136*:
assumes *finite X runiq f*
shows *sum (f Else 0) X = sum (toFunction f) (X∩Domain f)*
(**is** *?L=?R*)
⟨*proof*⟩

lemma *lm137*:
sum (f Else 0) X = sum' f X
⟨*proof*⟩

corollary *lm138*:
assumes *finite X runiq f*
shows *sum (toFunction f) (X∩Domain f) = sum' f X*
⟨*proof*⟩

lemma *lm139*:
argmax (sum' b) = (argmax ∘ sum') b
⟨*proof*⟩

lemma *domainConstant*:
Domain (Y × {0::nat}) = Y & Domain (X × {1}) = X
⟨*proof*⟩

lemma *domainCharacteristicFunction*:
Domain (X <|| Y) = X ∪ Y
⟨*proof*⟩

lemma *functionEquivalenceOnSets*:
assumes *∀ x ∈ X. f x = g x*
shows *f‘X = g‘X*
⟨*proof*⟩

7.7 Cardinalities of sets.

lemma *lm140*:
assumes *runiq R runiq (R^-1)*
shows *(R‘‘A) ∩ (R‘‘B) = R‘‘(A∩B)*
⟨*proof*⟩

lemma *intersectionEmptyRelationIntersectionEmpty*:
assumes *runiq* (R^{-1}) *runiq R X1* $\cap$ *X2* $= \{\}$
shows $(R``X1) \cap (R``X2) = \{\}$
⟨*proof*⟩

lemma *lm141*:
assumes *runiq f trivial Y*
shows *trivial* $(f `` (f^{-1} `` Y))$
⟨*proof*⟩

lemma *lm142*:
assumes *trivial X*
shows *card* $(Pow\ X) \in \{1,2\}$
⟨*proof*⟩

lemma *lm143*:
assumes *card* $(Pow\ A) = 1$
shows $A = \{\}$
⟨*proof*⟩

lemma *lm144*:
$(\neg\ (finite\ A)) = (card\ (Pow\ A) = 0)$
⟨*proof*⟩

corollary *lm145*:
$(finite\ A) = (card\ (Pow\ A) \neq 0)$
⟨*proof*⟩

lemma *lm146*:
assumes *card* $(Pow\ A) \neq 0$
shows *card A*=*floor-log* $(card\ (Pow\ A))$
⟨*proof*⟩

lemma *log-2* [*simp*]:
floor-log 2 = 1
⟨*proof*⟩

lemma *lm147*:
assumes *card* $(Pow\ A) = 2$
shows *card A = 1*
⟨*proof*⟩

lemma *lm148*:
assumes *card* $(Pow\ X) = 1 \vee$ *card* $(Pow\ X) = 2$
shows *trivial X*
⟨*proof*⟩

lemma *lm149*:

trivial A = (card (Pow A) ∈ {1,2})
⟨*proof*⟩

lemma *lm150*:
assumes *R ⊆ f runiq f Domain f = Domain R*
shows *runiq R*
⟨*proof*⟩

lemma *lm151*:
assumes *f ⊆ g runiq g Domain f = Domain g*
shows *g ⊆ f*
⟨*proof*⟩

lemma *lm152*:
assumes *R ⊆ f runiq f Domain f ⊆ Domain R*
shows *f = R*
⟨*proof*⟩

lemma *lm153*:
graph X f = (Graph f) || X
⟨*proof*⟩

lemma *lm154*:
graph (X ∩ Y) f = (graph X f) || Y
⟨*proof*⟩

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

lemma *lm155*:
assumes *runiq f X ⊆ Domain f*
shows *graph X (toFunction f) = (f||X)*
⟨*proof*⟩

lemma *lm156*:
(Graph f) `` X = f ` X
⟨*proof*⟩

lemma *lm157*:
assumes *X ⊆ Domain f runiq f*
shows *f``X = (eval-rel f)`X*
⟨*proof*⟩

lemma *cardOneImageCardOne*:
assumes *card A = 1*
shows *card (f`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) ∘ 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* = ((*participant* × *goods*) × *price*) *set*
type-synonym *bids* = *participant* ⇒ *goods* ⇒ *price*
type-synonym *allocation-rel* = (*goods* × *participant*) *set*
type-synonym *allocation* = (*participant* × *goods*) *set*
type-synonym *payments* = *participant* ⇒ *price*
type-synonym *bidvector* = (*participant* × *goods*) ⇒ *price*
abbreviation *bidvector* (*b*::*bids*) == *case-prod b*
abbreviation *proceeds* (*b*::*bidvector*) (*allo*::*allocation*) == *sum b allo*
abbreviation *winnersOfAllo* (*a*::*allocation*) == *Domain a*
abbreviation *allocatedGoods* (*allo*::*allocation*) == ⋃ (*Range allo*)

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

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

abbreviation *possible-allocations-rel' G N* == *Union*{*injections Y N* | *Y* . *Y* ∈ *all-partitions' G*}
abbreviation *allAllocations* **where**
allAllocations N G == *converse* ‘ (*possible-allocations-rel G N*)

algorithmic version of *possible-allocations-rel*

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

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

8.2 VCG mechanism

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

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

abbreviation *winningAllocationsAlg N G b == argmaxList (proceeds b) (allAllocationsAlg N G)*

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

payments

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

abbreviation *alpha N G b n == Max ((sum b)'(allAllocations (N−{n}) G))*

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

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

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

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

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

end

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

theory *Universes*

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

begin

9.1 Preliminary lemmas

lemma *lm001*:
assumes $Y \in$ *set* (*all-partitions-alg* X)
shows *distinct* Y
⟨*proof*⟩

lemma *lm002*:
assumes *finite* G
shows *all-partitions* G = *set* ‘ (*set* (*all-partitions-alg* G))
⟨*proof*⟩

9.2 Definitions of various subsets of *UNIV*.

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

abbreviation *partitionValuedUniverse* $== \bigcup P \in$ *partitionsUniverse*. *Pow* (*UNIV* $\times$ P)
lemma *partitionValuedUniverse* $\subseteq$ *Pow* (*UNIV* $\times$ (*Pow UNIV*))
⟨*proof*⟩

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

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

9.3 Results about the sets defined in the previous section

lemma *lm003*:
assumes $\forall$ *x1* $\in X.$ (*x1* $\neq \{\}$ & ($\forall$ *x2* $\in X - \{$*x1*$\}.$ *x1* $\cap$ *x2* = $\{\}$))
shows *is-non-overlapping* X
⟨*proof*⟩

lemma *lm004*:
assumes $\forall x \in X.\ f\ x \in x$
shows *isChoice* (*graph* X f)
⟨*proof*⟩

lemma *lm006*: *injections* X $Y \subseteq$ *injectionsUniverse*
⟨*proof*⟩

lemma *lm007*: *injections* X $Y \subseteq$ *injectionsUniverse*
⟨*proof*⟩

lemma *lm008*: *injections X Y = totalRels X Y ∩ injectionsUniverse*
⟨*proof*⟩

lemma *allocationInverseRangeDomainProperty*:
assumes *a ∈ allAllocations N G*
shows *a^−1 ∈ injections (Range a) N &*
(Range a) partitions G &
Domain a ⊆ N
⟨*proof*⟩

lemma *lm009*:
assumes *is-non-overlapping XX YY ⊆ XX*
shows *(XX − YY) partitions (⋃ XX − ⋃ YY)*
⟨*proof*⟩

lemma *allocationRightUniqueRangeDomain*:
assumes *a ∈ possible-allocations-rel G N*
shows *runiq a &*
runiq (a^−1) &
(Domain a) partitions G &
Range a ⊆ N
⟨*proof*⟩

lemma *lm010*:
assumes *runiq a runiq (a^−1) (Domain a) partitions G Range a ⊆ N*
shows *a ∈ possible-allocations-rel G N*
⟨*proof*⟩

lemma *allocationProperty*:
a ∈ possible-allocations-rel G N ⟷
runiq a & runiq (a^−1) & (Domain a) partitions G & Range a ⊆ N
⟨*proof*⟩

lemma *lm011*:
possible-allocations-rel′ G N ⊆ injectionsUniverse
⟨*proof*⟩

lemma *lm012*:
possible-allocations-rel G N ⊆ {a. (Range a) ⊆ N & (Domain a) ∈ all-partitions G}
⟨*proof*⟩

lemma *lm013*:
injections X Y = injections X Y
⟨*proof*⟩

lemma *lm014*:

all-partitions X = all-partitions′ X
⟨*proof*⟩

lemma *lm015*:
possible-allocations-rel′ A B = possible-allocations-rel A B
(**is** *?A=?B*)
⟨*proof*⟩

lemma *lm016*:
possible-allocations-rel G N ⊆
injectionsUniverse ∩ {a. Range a ⊆ N & Domain a ∈ all-partitions G}
⟨*proof*⟩

lemma *lm017*:
possible-allocations-rel G N ⊇
injectionsUniverse ∩ {a. Domain a ∈ all-partitions G & Range a ⊆ N}
⟨*proof*⟩

lemma *lm018*:
possible-allocations-rel G N =
injectionsUniverse ∩ {a. Domain a ∈ all-partitions G & Range a ⊆ N}
⟨*proof*⟩

lemma *lm019*:
converse ‘ injectionsUniverse = injectionsUniverse
⟨*proof*⟩

lemma *lm020*:
converse‘(A ∩ B) = (converse‘A) ∩ (converse‘B)
⟨*proof*⟩

lemma *allocationInjectionsUnivervseProperty*:
allAllocations N G =
injectionsUniverse ∩ {a. Domain a ⊆ N & Range a ∈ all-partitions G}
⟨*proof*⟩

lemma *lm021*:
allAllocations N G ⊆ injectionsUniverse
⟨*proof*⟩

lemma *lm022*:
allAllocations N G ⊆ partitionValuedUniverse
⟨*proof*⟩

corollary *allAllocationsUniverse*:
allAllocations N G ⊆ allocationsUniverse
⟨*proof*⟩

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

corollary *lm023*:
assumes $a \in allAllocations\ N1\ G$
shows $a \in allAllocations\ (N1 \cup N2)\ G$
⟨*proof*⟩

corollary *lm024*:
$allAllocations\ N1\ G \subseteq allAllocations\ (N1 \cup N2)\ G$
⟨*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 $is\text{-}non\text{-}overlapping\ (P1 \cup P2)$
⟨*proof*⟩

lemma *lm028*:
$Range\ Q \cup (Range\ (P\ outside\ (Domain\ Q))) = Range\ (P +\!\!*\ Q)$
⟨*proof*⟩

lemma *lm029*:
assumes $a1 \in injectionsUniverse$
$a2 \in injectionsUniverse$
$(Range\ a1) \cap (Range\ a2) = \{\}$
$(Domain\ a1) \cap (Domain\ a2) = \{\}$
shows $a1 \cup a2 \in injectionsUniverse$

⟨*proof*⟩

lemma *nonOverlapping*:
assumes $R \in partitionValuedUniverse$
shows *is-non-overlapping* (*Range R*)
⟨*proof*⟩

lemma *allocationUnion*:
assumes $a1 \in allocationsUniverse$
$a2 \in allocationsUniverse$
$(\bigcup\ (Range\ a1)) \cap (\bigcup\ (Range\ a2)) = \{\}$
$(Domain\ a1) \cap (Domain\ a2) = \{\}$
shows $a1 \cup a2 \in allocationsUniverse$
⟨*proof*⟩

lemma *lm030*:
assumes $a \in injectionsUniverse$
shows $a - b \in injectionsUniverse$
⟨*proof*⟩

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

lemma *lm032*:
$allAllocations\ N\ G =$
$injectionsUniverse \cap ((Range -` (all\text{-}partitions\ G)) \cap (Domain -`(Pow\ N)))$
⟨*proof*⟩

corollary *lm033*:
$allAllocations\ N\ G =$
$injectionsUniverse \cap (Range -` (all\text{-}partitions\ G)) \cap (Domain -`(Pow\ N))$
⟨*proof*⟩

lemma *lm034*:
assumes $a \in allAllocations\ N\ G$
shows $(a^{-1} \in injections\ (Range\ a)\ N\ \&$
$Range\ a \in all\text{-}partitions\ G)$
⟨*proof*⟩

lemma *lm035*:
assumes $a^{-1} \in injections\ (Range\ a)\ N\ Range\ a \in all\text{-}partitions\ G$
shows $a \in allAllocations\ N\ G$
⟨*proof*⟩

lemma *allocationReverseInjective*:
$a \in allAllocations\ N\ G =$
$(a^{\frown}{-}1 \in injections\ (Range\ a)\ N\ \&\ Range\ a \in all\text{-}partitions\ G)$
⟨*proof*⟩

lemma *lm036*:
assumes $a \in allAllocations\ N\ G$
shows $a \in injections\ (Domain\ a)\ (Range\ a)\ \&$
$Range\ a \in all\text{-}partitions\ G\ \&$
$Domain\ a \subseteq N$
⟨*proof*⟩

lemma *lm037*:
assumes $a \in injections\ (Domain\ a)\ (Range\ a)$
$Range\ a \in all\text{-}partitions\ G$
$Domain\ a \subseteq N$
shows $a \in allAllocations\ N\ G$
⟨*proof*⟩

lemma *characterizationallAllocations*:
$a \in allAllocations\ N\ G = (a \in injections\ (Domain\ a)\ (Range\ a)\ \&$
$Range\ a \in all\text{-}partitions\ G\ \&$
$Domain\ a \subseteq N)$
⟨*proof*⟩

lemma *lm038*:
assumes $a \in partitionValuedUniverse$
shows $a - b \in partitionValuedUniverse$
⟨*proof*⟩

lemma *reducedAllocation*:
assumes $a \in allocationsUniverse$
shows $a - b \in allocationsUniverse$
⟨*proof*⟩

lemma *lm039*:
assumes $a \in injectionsUniverse$
shows $a \in injections\ (Domain\ a)\ (Range\ a)$
⟨*proof*⟩

lemma *lm040*:
assumes $a \in allocationsUniverse$
shows $a \in allAllocations\ (Domain\ a)\ (\bigcup\ (Range\ a))$
⟨*proof*⟩

lemma *lm041*:
$(\{X\} \in partitionsUniverse) = (X \neq \{\})$
⟨*proof*⟩

lemma *lm042*:
$\{(x,\ X)\} - \{(x,\ \{\})\} \in partitionValuedUniverse$
⟨*proof*⟩

lemma *singlePairInInjectionsUniverse*:
$\{(x,\ X)\} \in injectionsUniverse$
⟨*proof*⟩

lemma *allocationUniverseProperty*:
$\{(x,X)\} - \{(x,\{\})\} \in allocationsUniverse$
⟨*proof*⟩

lemma *lm043*:
assumes *is-non-overlapping PP is-non-overlapping* (*Union PP*)
shows *is-non-overlapping* (*Union ‘ PP*)
⟨*proof*⟩

lemma *lm044*:
assumes $a \in allocationsUniverse$
shows $(a - ((X \cup \{i\}) \times (Range\ a))) \cup$
$(\{(i, \bigcup\ (a\text{`}\text{`}(X \cup \{i\})))\} - \{(i,\{\})\}) \in allocationsUniverse\ \&$
$\bigcup\ (Range\ ((a - ((X \cup \{i\}) \times (Range\ a))) \cup (\{(i, \bigcup\ (a\text{`}\text{`}(X \cup \{i\})))\} - \{(i,\{\})\}))) =$
$\bigcup (Range\ a)$
⟨*proof*⟩

corollary *allocationsUniverseOutsideUnion*:
assumes $a \in allocationsUniverse$
shows $(a\ outside\ (X \cup \{i\})) \cup (\{i\} \times (\{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\} - \{\{\}\})) \in allocationsUniverse\ \&$
$\bigcup (Range((a\ outside\ (X \cup \{i\})) \cup (\{i\} \times (\{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\} - \{\{\}\})))) = \bigcup (Range\ a)$
⟨*proof*⟩

lemma *lm045*:
assumes $Domain\ a \cap X \neq \{\}\ a \in allocationsUniverse$
shows $\bigcup (a\text{`}\text{`}X) \neq \{\}$
⟨*proof*⟩

corollary *lm046*:
assumes $Domain\ a \cap X \neq \{\}\ a \in allocationsUniverse$
shows $\{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\} - \{\{\}\} = \{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\}$
⟨*proof*⟩

corollary *lm047*:
assumes $a \in allocationsUniverse$
$(Domain\ a) \cap X \neq \{\}$
shows $(a\ outside\ (X \cup \{i\})) \cup (\{i\} \times \{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\}) \in allocationsUniverse\ \&$

$\bigcup (Range((a\ outside\ (X \cup \{i\})) \cup (\{i\} \times \{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\}))) =$
$\bigcup (Range\ a)$
⟨*proof*⟩

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

lemma *lm048*:
assumes $bidMonotonicity\ b\ i\ runiq\ a$
shows $sum\ b\ (\{i\} \times ((a\ outside\ X)\text{`}\text{`}\{i\})) \leq sum\ b\ (\{i\} \times \{\bigcup (a\text{`}\text{`}(X \cup \{i\}))\})$
⟨*proof*⟩

lemma *lm049*:
assumes $XX \in partitionValuedUniverse$
shows $\{\} \notin Range\ XX$
⟨*proof*⟩

corollary *emptyNotInRange*:
assumes $a \in allAllocations\ N\ G$
shows $\{\} \notin Range\ a$
⟨*proof*⟩

lemma *lm050*:
assumes $a \in allAllocations\ N\ G$
shows $Range\ a \subseteq Pow\ G$
⟨*proof*⟩

corollary *lm051*:
assumes $a \in allAllocations\ N\ G$
shows $Domain\ a \subseteq N\ \&\ Range\ a \subseteq Pow\ G - \{\{\}\}$
⟨*proof*⟩

corollary *allocationPowerset*:
assumes $a \in allAllocations\ N\ G$
shows $a \subseteq N \times (Pow\ G - \{\{\}\})$
⟨*proof*⟩

corollary *lm052*:
$allAllocations\ N\ G \subseteq Pow\ (N \times (Pow\ G - \{\{\}\}))$
⟨*proof*⟩

lemma *lm053*:
assumes $a \in allAllocations\ N\ G$
$i \in N-X$
$Domain\ a \cap X \neq \{\}$
shows $a\ outside\ (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a`(X \cup \{i\}))\}) \in$
$allAllocations\ (N-X)\ (\bigcup\ (Range\ a))$
⟨*proof*⟩

lemma *lm054*:
assumes $bidMonotonicity\ (b::- => real)\ i$
$a \in allocationsUniverse$
$Domain\ a \cap X \neq \{\}$
$finite\ a$
shows $sum\ b\ (a\ outside\ X) \leq$
$sum\ b\ (a\ outside\ (X \cup \{i\}) \cup (\{i\} \times \{\bigcup (a`(X \cup \{i\}))\}))$
⟨*proof*⟩

lemma *elementOfPartitionOfFiniteSetIsFinite*:
assumes $finite\ X\ XX \in all\text{-}partitions\ X$
shows $finite\ XX$
⟨*proof*⟩

lemma *lm055*:
assumes $finite\ N\ finite\ G\ a \in allAllocations\ N\ G$
shows $finite\ a$
⟨*proof*⟩

lemma *allAllocationsFinite*:
assumes $finite\ N\ finite\ G$
shows $finite\ (allAllocations\ N\ G)$
⟨*proof*⟩

corollary *lm056*:
assumes $bidMonotonicity\ (b::- => real)\ i$
$a \in allAllocations\ N\ G$
$i \in N-X$
$Domain\ a \cap X \neq \{\}$
$finite\ N$
$finite\ G$
shows $Max\ ((sum\ b)`(allAllocations\ (N-X)\ G)) \geq$
$sum\ b\ (a\ outside\ X)$
⟨*proof*⟩

lemma *cardinalityPreservation*:
assumes $\forall X \in XX.\ finite\ X\ is\text{-}non\text{-}overlapping\ XX$
shows $card\ (\bigcup\ XX) = sum\ card\ XX$
⟨*proof*⟩

corollary *cardSumCommute*:
assumes *XX partitions X finite X finite XX*
shows *card* $(\bigcup\ XX) = sum\ card\ XX$
⟨*proof*⟩

lemma *sumUnionDisjoint1*:
assumes $\forall\, A{\in}C.\ finite\ A\ \forall\, A{\in}C.\ \forall\, B{\in}C.\ A \neq B \longrightarrow A\ Int\ B = \{\}$
shows $sum\ f\ (Union\ C) = sum\ (sum\ f)\ C$
⟨*proof*⟩

corollary *sumUnionDisjoint2*:
assumes $\forall\, x{\in}X.$ *finite x is-non-overlapping X*
shows $sum\ f\ (\bigcup\ X) = sum\ (sum\ f)\ X$
⟨*proof*⟩

corollary *sumUnionDisjoint3*:
assumes $\forall\, x{\in}X.$ *finite x X partitions XX*
shows $sum\ f\ XX = sum\ (sum\ f)\ X$
⟨*proof*⟩

corollary *sum-associativity*:
assumes *finite x X partitions x*
shows $sum\ f\ x = sum\ (sum\ f)\ X$
⟨*proof*⟩

lemma *lm057*:
assumes $a \in allocationsUniverse\ Domain\ a \subseteq N \bigcup (Range\ a) = G$
shows $a \in allAllocations\ N\ G$
⟨*proof*⟩

corollary *lm058*:
$(allocationsUniverse \cap \{a.\ (Domain\ a) \subseteq N\ \&\ \bigcup (Range\ a) = G\}) \subseteq$
$allAllocations\ N\ G$
⟨*proof*⟩

corollary *lm059*:
$allAllocations\ N\ G \subseteq \{a.\ (Domain\ a) \subseteq N\}$
⟨*proof*⟩

corollary *lm060*:
$allAllocations\ N\ G \subseteq \{a.\ \bigcup (Range\ a) = G\}$
⟨*proof*⟩

corollary *lm061*:
$allAllocations\ N\ G \subseteq allocationsUniverse\ \&$
$allAllocations\ N\ G \subseteq \{a.\ (Domain\ a) \subseteq N\ \&\ \bigcup (Range\ a) = G\}$
⟨*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 *runiq* $(f^{-}-1)$ $A \cap$ *Range* $f = \{\}$
shows *converse* ‘ $\{ f \cup \{(x, y)\} \mid y . y \in A \} \subseteq$ *runiqs*
⟨*proof*⟩

lemma *lm069*:
assumes $f \in$*converse*‘*runiqs* $A \cap$ *Range* $f = \{\}$
shows $\{f \cup \{(x, y)\} \mid y. y \in A\} \subseteq$ *converse*‘*runiqs*
(**is** *?l* $\subseteq$ *?r*)
⟨*proof*⟩

lemma *lm070*:
$\{ R \cup \{(x, y)\} \mid y . y \in A \} \subseteq$ *totalRels* $(\{x\} \cup$ *Domain* $R)$ $(A \cup$ *Range* $R)$
⟨*proof*⟩

lemma *lm071*:
injectionsUniverse $=$ *runiqs* $\cap$ *converse*‘*runiqs*
⟨*proof*⟩

lemma *lm072*:
assumes $f \in$ *injectionsUniverse* $x \notin$ *Domain* f $A \cap$ (*Range* f) $= \{\}$
shows $\{f \cup \{(x, y)\} \mid y. y \in A\} \subseteq$ *injectionsUniverse*
(**is** *?l* $\subseteq$ *?r*)
⟨*proof*⟩

lemma *lm073*:
injections X Y $=$ *totalRels* X $Y \cap$ *injectionsUniverse*
⟨*proof*⟩

lemma *lm074*:
assumes $f \in$ *injectionsUniverse*
shows f *outside* $A \in$ *injectionsUniverse*
⟨*proof*⟩

lemma *lm075*:
assumes $R \in$ *totalRels* A B
shows R *outside* $C \in$ *totalRels* $(A-C)$ B
⟨*proof*⟩

lemma *lm076*:
assumes $g \in$ *injections* A B
shows g *outside* $C \in$ *injections* $(A - C)$ B
⟨*proof*⟩

lemma *lm077*:
assumes $g \in$ *injections* A B
shows g *outside* $C \in$ *injections* $(A - C)$ B
⟨*proof*⟩

lemma *lm078*:
$\{x\}\times\{y\}=\{(x,y)\}$
⟨*proof*⟩

lemma *lm079*:
assumes $x \in Domain\ f\ runiq\ f$
shows $\{x\}\times f``\{x\} = \{(x,f,,x)\}$
⟨*proof*⟩

corollary *lm080*:
assumes $x \in Domain\ f\ runiq\ f$
shows $f = (f\ --\ x) \cup \{(x,f,,x)\}$
⟨*proof*⟩

lemma *lm081*:
assumes $f \in injectionsUniverse$
shows $Range(f\ outside\ A) = Range\ f - f``A$
⟨*proof*⟩

lemma *lm082*:
assumes $g \in injections\ X\ Y\ x \in Domain\ g$
shows $g \in \{g--x \cup \{(x,y)\} | y.\ y \in Y - (Range(g--x))\}$
⟨*proof*⟩

corollary *lm083*:
assumes $x \notin X\ g \in injections\ (\{x\} \cup X)\ Y$
shows $g--x \in injections\ X\ Y$
⟨*proof*⟩

corollary *lm084*:
assumes $x \notin X\ g \in injections\ (\{x\} \cup X)\ Y$
(**is** $g \in injections\ (?X)\ Y$)
shows $\exists\ f \in injections\ X\ Y.\ g \in \{f \cup \{(x,y)\} | y.\ y \in Y - (Range\ f)\}$
⟨*proof*⟩

corollary *lm085*:
assumes $x \notin X$
shows $injections\ (\{x\} \cup X)\ Y \subseteq$
$(\bigcup\ f \in injections\ X\ Y.\ \{f \cup \{(x,\ y)\} \mid y\ .\ y \in Y - (Range\ f)\})$
⟨*proof*⟩

lemma *lm086*:
assumes $x \notin X$
shows $(\bigcup\ f \in injections\ X\ Y.\ \{f \cup \{(x,\ y)\} \mid y\ .\ y \in Y - Range\ f\}) \subseteq$
$injections\ (\{x\} \cup X)\ Y$
⟨*proof*⟩

corollary *injectionsUnionCommute*:

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

lemma *lm087*:
assumes $\forall\ x.\ (P\ x \longrightarrow (f\ x = g\ x))$
shows $Union\ \{f\ x|x.\ P\ x\} = Union\ \{g\ x \mid x.\ P\ x\}$
⟨*proof*⟩

lemma *lm088*:
assumes $x \notin Domain\ R$
shows $R +\!\!* \{(x,y)\} = R \cup \{(x,y)\}$
⟨*proof*⟩

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

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

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

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

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

lemma *lm094*:
assumes *finite Y*
shows *set* [*f* +∗ {(*x*,*y*)} . *y* ← *sorted-list-of-set* (*Y* − (*Range f*))] =
{ *f* +∗ {(*x*,*y*)} | *y* . *y* ∈ *Y* − (*Range f*)}
⟨*proof*⟩

lemma *lm095*:
assumes *finite Y*
shows *set* (*concat* [[*f* +∗ {(*x*,*y*)}. *y* ← *sorted-list-of-set*(*Y* − (*Range f*))]. *f* ← *F*]) =
(⋃ *f* ∈ *set F*.{*f* +∗ {(*x*,*y*)} | *y* . *y* ∈ *Y* − (*Range f*)})
⟨*proof*⟩

9.5 Computable injections

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

corollary *lm096*:
set (*injectionsAlg* (*x* # *xs*) *Y*) =
(⋃ *f* ∈ *set* (*injectionsAlg xs Y*). {*f* ∪ {(*x*,*y*)} | *y* . *y* ∈ (*set Y*) − (*Range f*)})
⟨*proof*⟩

corollary *lm097*:
assumes *set* (*injectionsAlg xs Y*) = *injections* (*set xs*) (*set Y*)
shows *set* (*injectionsAlg* (*x* # *xs*) *Y*) =
(⋃ *f* ∈ *injections* (*set xs*) (*set Y*). {*f* ∪ {(*x*,*y*)} | *y* . *y* ∈ (*set Y*) − (*Range f*)})
⟨*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 $x \notin set\ xs$ $set\ (injectionsAlg\ xs\ Y) = injections\ (set\ xs)\ (set\ Y)$
shows $set\ (injectionsAlg\ (x\ \#\ xs)\ Y) = injections\ (\{x\} \cup set\ xs)\ (set\ Y)$
(**is** *?l=?r*)
⟨*proof*⟩

lemma *lm101*:
assumes $x \notin set\ xs$
$set\ (injections\text{-}alg\ xs\ Y) = injections\ (set\ xs)\ Y$
finite Y
shows $set\ (injections\text{-}alg\ (x\#xs)\ Y) = injections\ (\{x\} \cup set\ xs)\ Y$
(**is** *?l=?r*)
⟨*proof*⟩

lemma *listInduct*:
assumes $P\ [] \forall\ xs\ x.\ P\ xs \longrightarrow P\ (x\#xs)$
shows $\forall x.\ P\ x$
⟨*proof*⟩

lemma *injectionsFromEmptyAreEmpty*:
$set\ (injections\text{-}alg\ []\ Z) = \{\{\}\}$
⟨*proof*⟩

theorem *injections-equiv*:
assumes *finite Y* **and** *distinct X*
shows $set\ (injections\text{-}alg\ X\ Y) = injections\ (set\ X)\ Y$
⟨*proof*⟩

lemma *lm102*:
assumes $l \in set\ (all\text{-}partitions\text{-}list\ G)$ *distinct G*
shows *distinct l*
⟨*proof*⟩

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

lemma *lm103*:
assumes *card N > 0 distinct G*
shows $\{injections\ P\ N |\ P.\ P \in all\text{-}partitions\ (set\ G)\} =$
$set\ [set\ (injections\text{-}alg\ l\ N)\ .\ l \leftarrow all\text{-}partitions\text{-}list\ G]$
⟨*proof*⟩

lemma *lm104*:
assumes *card N > 0 distinct G*
shows *Union {injections P N| P. P ∈ all-partitions (set G)} =*
Union (set [set (injections-alg l N) . l ← all-partitions-list G])
(**is** *Union ?L = Union ?R*)
⟨*proof*⟩

corollary *allAllocationsBridgingLemma*:
assumes *card N > 0 distinct G*
shows *allAllocations N (set G) =*
set(*allAllocationsAlg N G*)
⟨*proof*⟩

end

10 Termination theorem for uniform tie-breaking

theory *UniformTieBreaking*

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

begin

10.1 Uniform tie breaking: definitions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *randomElLemma*:
assumes *set list ≠ {}*
shows *randomEl list random ∈ set list*
⟨*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 ⊆ *allAllocations N G*
⟨*proof*⟩

lemma *subsetAllocation*:
assumes *a* ∈ *allocationsUniverse c* ⊆ *a*
shows *c* ∈ *allocationsUniverse*
⟨*proof*⟩

lemma *lm001*:
assumes *a* ∈ *allocationsUniverse*
shows *a outside X* ∈ *allocationsUniverse*
⟨*proof*⟩

corollary *lm002*:
{*x*}×({*X*}−{{}}) ∈ *allocationsUniverse*
⟨*proof*⟩

corollary *lm003*:
{(*x*,{*y*})} ∈ *allocationsUniverse*
⟨*proof*⟩

corollary *lm004*:
allocationsUniverse ≠ {}
⟨*proof*⟩

corollary *lm005*:
{} ∈ *allocationsUniverse*
⟨*proof*⟩

lemma *lm006*:
assumes *G* ≠ {}
shows {*G*} ∈ *all-partitions G*
⟨*proof*⟩

lemma *lm007*:
assumes *n* ∈ *N*
shows {(*G*,*n*)} ∈ *totalRels* {*G*} *N*
⟨*proof*⟩

lemma *lm008*:

assumes $n \in N$
shows $\{(G,n)\} \in$ *injections* $\{G\}$ N
⟨*proof*⟩

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

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

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

lemma *lm012*:
allAllocations N $\{\} \subseteq \{\{\}\}$
⟨*proof*⟩

lemma *lm013*:
assumes $a \in$ *allAllocations* N G *finite* G
shows *finite* (*Range* a)
⟨*proof*⟩

corollary *allocationFinite*:
assumes $a \in$ *allAllocations* N G *finite* G
shows *finite* a
⟨*proof*⟩

lemma *lm014*:
assumes $a \in$ *allAllocations* N G *finite* G
shows $\forall\ y \in$ *Range* a. *finite* y
⟨*proof*⟩

corollary *lm015*:
assumes $a \in$ *allAllocations* N G *finite* G
shows *card* G = *sum card* (*Range* a)
⟨*proof*⟩

10.3 Results on summed bid vectors

lemma *lm016*:
summedBidVectorRel bids N G =

{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*)))|
pair. *pair* ∈ *N* × (*Pow G*−{{}})}
⟨*proof*⟩

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

corollary *lm018*:
(*summedBidVectorRel bids N G*) || *a* =
{(*pair*, *sum* (%*g*. *bids* (*fst pair*, *g*)) (*finestpart* (*snd pair*))) |
pair. *pair* ∈ (*N* × (*Pow G* − {{}})) ∩ *a*}
(**is** *?L* = *?R*)
⟨*proof*⟩

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

corollary *lm020*:
(*summedBidVectorRel bids N G*) || *a* = (*summedBid bids*) ‘ ((*N* × (*Pow G* − {{}})) ∩ *a*)
(**is** *?L*=*?R*)
⟨*proof*⟩

lemma *summedBidInjective*:
inj-on (*summedBid bids*) *UNIV*
⟨*proof*⟩

corollary *lm021*:
inj-on (*summedBid bids*) *X*
⟨*proof*⟩

lemma *lm022*:
sum snd (*summedBidVectorRel bids N G*) =
sum (*snd* ∘ (*summedBid bids*)) (*N* × (*Pow G* − {{}}))
⟨*proof*⟩

corollary *lm023*:

snd (*summedBid bids pair*) = *sum bids* (*omega pair*)
⟨*proof*⟩

corollary *lm024*:
snd ∘ *summedBid bids* = (*sum bids*) ∘ *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 {} ∉ *Range f runiq f*
shows *is-non-overlapping* (*omega* ‘ *f*)
⟨*proof*⟩

lemma *lm028*:
assumes {} ∉ *Range X*
shows *inj-on omega X*
⟨*proof*⟩

lemma *lm029*:
assumes {} ∉ *Range a* ∀ *X* ∈ *omega* ‘ *a*. *finite X*
is-non-overlapping (*omega* ‘ *a*)
shows *card* (*pseudoAllocation a*) = *sum* (*card* ∘ *omega*) *a*
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *lm030*:
card (*omega pair*)= *card* (*snd pair*)
⟨*proof*⟩

corollary *lm031*:
card ∘ *omega* = *card* ∘ *snd*
⟨*proof*⟩

corollary *lm032*:
assumes {} ∉ *Range a* ∀ *pair* ∈ *a*. *finite* (*snd pair*) *finite a runiq a*
shows *card* (*pseudoAllocation a*) = *sum* (*card* ∘ *snd*) *a*
⟨*proof*⟩

corollary *lm033*:
assumes *runiq* (a^{-1}) *runiq a finite a* $\{\} \notin$ *Range a* $\forall$ *pair* $\in$ *a. finite* (*snd pair*)
shows *card* (*pseudoAllocation a*) = *sum card* (*Range a*)
⟨*proof*⟩

corollary *lm034*:
assumes *a* $\in$ *allAllocations N G finite G*
shows *card* (*pseudoAllocation a*) = *card G*
⟨*proof*⟩

corollary *lm035*:
assumes *pseudoAllocation aa* $\subseteq$ *pseudoAllocation a* $\cup$ (*N* $\times$ (*finestpart G*))
finite (*pseudoAllocation aa*)
shows *sum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation a*) −
(*sum* (*toFunction* (*bidMaximizedBy a N G*)) (*pseudoAllocation aa*)) =
card (*pseudoAllocation a*) −
card (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
⟨*proof*⟩

corollary *lm036*:
assumes *pseudoAllocation aa* $\subseteq$ *pseudoAllocation a* $\cup$ (*N* $\times$ (*finestpart G*))
finite (*pseudoAllocation aa*)
shows *int* (*sum* (*maxbid a N G*) (*pseudoAllocation a*)) −
int (*sum* (*maxbid a N G*) (*pseudoAllocation aa*)) =
int (*card* (*pseudoAllocation a*)) −
int (*card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*)))
⟨*proof*⟩

lemma *lm037*:
pseudoAllocation {} = {}
⟨*proof*⟩

corollary *lm038*:
assumes *a* $\in$ *allAllocations N* {}
shows (*pseudoAllocation a*) = {}
⟨*proof*⟩

corollary *lm039*:
assumes *a* $\in$ *allAllocations N G finite G G* $\neq$ {}
shows *finite* (*pseudoAllocation a*)
⟨*proof*⟩

corollary *lm040*:
assumes *a* $\in$ *allAllocations N G finite G*
shows *finite* (*pseudoAllocation a*)
⟨*proof*⟩

lemma *lm041*:

assumes $a \in allAllocations\ N\ G\ aa \in allAllocations\ N\ G\ finite\ G$
shows $(card\ (pseudoAllocation\ aa \cap (pseudoAllocation\ a)) = card\ (pseudoAllocation\ a)) =$
$(pseudoAllocation\ a = pseudoAllocation\ aa)$
⟨*proof*⟩

lemma *lm042*:
$omega\ pair = \{fst\ pair\} \times \{\{y\} |\ y.\ y \in snd\ pair\}$
⟨*proof*⟩

lemma *lm043*:
$omega\ pair = \{(fst\ pair, \{y\}) |\ y.\ y \in\ snd\ pair\}$
⟨*proof*⟩

lemma *lm044*:
$pseudoAllocation\ a = \bigcup \{\{(fst\ pair, \{y\}) |\ y.\ y \in snd\ pair\} |\ pair.\ pair \in a\}$
⟨*proof*⟩

lemma *lm045*:
$\bigcup \{\{(fst\ pair, \{y\}) |\ y.\ y \in snd\ pair\} |\ pair.\ pair \in a\} =$
$\{(fst\ pair, \{y\}) |\ y\ pair.\ y \in snd\ pair\ \&\ pair \in a\}$
⟨*proof*⟩

corollary *lm046*:
$pseudoAllocation\ a = \{(fst\ pair,\ Y) |\ Y\ pair.\ Y \in finestpart\ (snd\ pair)\ \&\ pair \in a\}$
⟨*proof*⟩

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

corollary *lm048*:
assumes *runiq a*
shows $pseudoAllocation\ a = \{(x,\ Y) |\ Y\ x.\ Y \in finestpart\ (a,,x)\ \&\ x \in Domain\ a\}$
⟨*proof*⟩

corollary *lm049*:
$Range\ (pseudoAllocation\ a) = \bigcup\ (finestpart\ `\ (Range\ a))$
⟨*proof*⟩

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

⟨*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* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum snd* ((*summedBid bids*) ‘ *a*)
(**is** *?L=?R*)
⟨*proof*⟩

corollary *lm061*:
sum snd ((*summedBid bids*) ‘ *a*) = *sum* (*snd* ∘ (*summedBid bids*)) *a*
⟨*proof*⟩

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

corollary *lm063*:
assumes *a* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum* ((*sum bids*) ∘ *omega*) *a*
(**is** *?L=?R*)
⟨*proof*⟩

corollary *lm064*:
assumes *a* ∈ *allAllocations N G*
shows *sum* (*summedBidVector bids N G*) *a* = *sum* (*sum bids*) (*omega‘ a*)
⟨*proof*⟩

lemma *lm065*:
assumes *finite* (*snd pair*)
shows *finite* (*omega pair*)
⟨*proof*⟩

corollary *lm066*:
assumes ∀ *y*∈(*Range a*). *finite y*
shows ∀ *y*∈(*omega* ‘ *a*). *finite y*
⟨*proof*⟩

corollary *lm067*:
assumes *a* ∈ *allAllocations N G finite G*
shows ∀ *x*∈(*omega* ‘ *a*). *finite x*
⟨*proof*⟩

corollary *lm068*:
assumes *a* ∈ *allAllocations N G*
shows *is-non-overlapping* (*omega* ‘ *a*)
⟨*proof*⟩

lemma *lm069*:
assumes $a \in allAllocations\ N\ G$ *finite* G
shows $sum\ (sum\ bids)\ (omega`\ a) = sum\ bids\ (\bigcup\ (omega\ `\ a))$
⟨*proof*⟩

corollary *lm070*:
assumes $a \in allAllocations\ N\ G$ *finite* G
shows $sum\ (summedBidVector\ bids\ N\ G)\ a = sum\ bids\ (pseudoAllocation\ a)$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *lm071*:
$Domain\ (pseudoAllocation\ a) \subseteq Domain\ a$
⟨*proof*⟩

corollary *lm072*:
assumes $a \in allAllocations\ N\ G$
shows $Domain\ (pseudoAllocation\ a) \subseteq N \quad \& \quad Range\ (pseudoAllocation\ a) = finestpart\ G$
⟨*proof*⟩

corollary *lm073*:
assumes $a \in allAllocations\ N\ G$
shows $pseudoAllocation\ a \subseteq N \times finestpart\ G$
⟨*proof*⟩

10.4 From Pseudo-allocations to allocations

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

lemma *lm074*:
assumes $runiq\ a\ \{\} \notin Range\ a$
shows $a = pseudoAllocationInv\ (pseudoAllocation\ a)$
⟨*proof*⟩

corollary *lm075*:
assumes $a \in runiqs \cap Pow\ (UNIV \times (UNIV - \{\{\}\}))$
shows $(pseudoAllocationInv \circ pseudoAllocation)\ a = id\ a$
⟨*proof*⟩

lemma *lm076*:
$inj\text{-}on\ (pseudoAllocationInv \circ pseudoAllocation)\ (runiqs \cap Pow\ (UNIV \times (UNIV - \{\{\}\})))$
⟨*proof*⟩

corollary *lm077*:
$inj\text{-}on\ pseudoAllocation\ (runiqs \cap Pow\ (UNIV \times (UNIV - \{\{\}\})))$

⟨*proof*⟩

lemma *lm078*:
injectionsUniverse ⊆ *runiqs*
⟨*proof*⟩

lemma *lm079*:
partitionValuedUniverse ⊆ *Pow* (*UNIV* × (*UNIV* − {{}}))
⟨*proof*⟩

corollary *lm080*:
allocationsUniverse ⊆ *runiqs* ∩ *Pow* (*UNIV* × (*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* ⊆ *set* (*allAllocationsAlg N G*)
⟨*proof*⟩

corollary *lm084*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows *winningAllocationsRel N* (*set G*) *bids* ∩ *set* (*allAllocationsAlg N G*) ≠ {}
⟨*proof*⟩

lemma *lm085*:
X = (%*x*. *x* ∈ *X*) −‘{*True*}
⟨*proof*⟩

corollary *lm086*:
assumes *N* ≠ {} *finite N distinct G set G* ≠ {}
shows (%*x*. *x*∈*winningAllocationsRel N* (*set G*) *bids*)−‘{*True*} ∩
set (*allAllocationsAlg N G*) ≠ {}
⟨*proof*⟩

lemma *lm087*:
assumes *P* −‘ {*True*} ∩ *set l* ≠ {}
shows *takeAll P l* ≠ []
⟨*proof*⟩

corollary *lm088*:

assumes $N \neq \{\}$ *finite N distinct G set* $G \neq \{\}$
shows *takeAll* (%*x*. $x \in$ *winningAllocationsRel N* (*set G*) *bids*) (*allAllocationsAlg N G*) $\neq$ []
⟨*proof*⟩

corollary *lm089*:
assumes $N \neq \{\}$ *finite N distinct G set* $G \neq \{\}$
shows *perm2* (*takeAll* (%*x*. $x \in$ *winningAllocationsRel N* (*set G*) *bids*)
(*allAllocationsAlg N G*))
n $\neq$ []
⟨*proof*⟩

corollary *lm090*:
assumes $N \neq \{\}$ *finite N distinct G set* $G \neq \{\}$
shows *chosenAllocation N G bids random* $\in$ *winningAllocationsRel N* (*set G*) *bids*
⟨*proof*⟩

lemma *lm091*:
assumes *finite G a* $\in$ *allAllocations N G aa* $\in$ *allAllocations N G*
shows *real*(*sum*(*maxbid a N G*)(*pseudoAllocation a*)) −
sum(*maxbid a N G*)(*pseudoAllocation aa*) =
real (*card G*) −
card (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
⟨*proof*⟩

lemma *lm092*:
summedBidVectorRel bids N G = *graph* (*N* × (*Pow G*−{{}})) (*summedBidSecond bids*)
⟨*proof*⟩

lemma *lm093*:
assumes *x*∈*X*
shows *toFunction* (*graph X f*) *x* = *f x*
⟨*proof*⟩

corollary *lm094*:
assumes *pair* $\in$ *N* × (*Pow G*−{{}})
shows *summedBidVector bids N G pair* = *summedBidSecond bids pair*
⟨*proof*⟩

lemma *lm095*:
summedBidSecond (*real* ∘ ((*bids*:: - => *nat*))) *pair* = *real* (*summedBidSecond bids pair*)
⟨*proof*⟩

lemma *lm096*:
assumes *pair* ∈ *N* × (*Pow G*−{{}})
shows *summedBidVector* (*real*∘(*bids*:: - => *nat*)) *N G pair* =
real (*summedBidVector bids N G pair*)
⟨*proof*⟩

corollary *lm097*:
assumes *X* ⊆ *N* × (*Pow G* − {{}})
shows ∀ *pair* ∈ *X*. *summedBidVector* (*real* ∘ (*bids*::-=>*nat*)) *N G pair* =
(*real* ∘ (*summedBidVector bids N G*)) *pair*
⟨*proof*⟩

corollary *lm098*:
assumes *aa* ⊆ *N* × (*Pow G*−{{}})
shows *sum* ((*summedBidVector* (*real* ∘ (*bids*::-=>*nat*)) *N G*)) *aa* =
real (*sum* ((*summedBidVector bids N G*)) *aa*)
(**is** *?L*=*?R*)
⟨*proof*⟩

corollary *lm099*:
assumes *aa* ∈ *allAllocations N G*
shows *sum* ((*summedBidVector* (*real* ∘ (*bids*::-=>*nat*)) *N G*)) *aa* =
real (*sum* ((*summedBidVector bids N G*)) *aa*)
⟨*proof*⟩

corollary *lm100*:
assumes *finite G a* ∈ *allAllocations N G aa* ∈ *allAllocations N G*
shows *real* (*sum* (*tiebids a N G*) *a*) − *sum* (*tiebids a N G*) *aa* =
real (*card G*) − *card* (*pseudoAllocation aa* ∩ (*pseudoAllocation a*))
(**is** *?L*=*?R*)
⟨*proof*⟩

corollary *lm101*:
assumes *finite G a* ∈ *allAllocations N G aa* ∈ *allAllocations N G*
x = *real* (*sum* (*tiebids a N G*) *a*) − *sum* (*tiebids a N G*) *aa*
shows *x* <= *card G* &
x ≥ *0* &
(*x*=*0* ⟷ *a* = *aa*) &
(*aa* ≠ *a* ⟶ *sum* (*tiebids a N G*) *aa* < *sum* (*tiebids a N G*) *a*)
⟨*proof*⟩

corollary *lm102*:
assumes *finite G a* ∈ *allAllocations N G*
aa ∈ *allAllocations N G aa* ≠ *a*
shows *sum* (*tiebids a N G*) *aa* < *sum* (*tiebids a N G*) *a*
⟨*proof*⟩

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

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

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

theorem *winningAllocationUniqueness*:
assumes $N \neq \{\}$ *distinct G set* $G \neq \{\}$ *finite N*
shows *terminatingAuctionRel N G* (*bids*) *random* = {*chosenAllocation N G bids random*}
⟨*proof*⟩

The computable variant of Else is defined next as Elsee.

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

end

11 VCG auction: definitions and theorems

theory *CombinatorialAuction*

imports
UniformTieBreaking

begin

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

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

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

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

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

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

lemma *allAllocationsEquivalence*:
allAllocations N Ω = allAllocations′ N Ω & allAllocations N Ω = allAllocations″ N Ω
⟨*proof*⟩

lemma *allAllocationsVarCharacterization*:
(a ∈ allAllocations″ N Ω) = (a ∈ allocationsUniverse& Domain a ⊆ N & ⋃(Range a) = Ω)
⟨*proof*⟩

abbreviation *soldAllocations N Ω == (Outside′ {seller}) ‘ (allAllocations (N ∪ {seller}) Ω)*

abbreviation *soldAllocations′ N Ω == (Outside′ {seller}) ‘ (allAllocations′ (N ∪ {seller}) Ω)*
abbreviation *soldAllocations″ N Ω == (Outside′ {seller}) ‘ (allAllocations″ (N ∪ {seller}) Ω)*
abbreviation *soldAllocations‴ N Ω ==*
allocationsUniverse ∩ {aa. Domain aa⊆N−{seller} & ⋃(Range aa)⊆Ω}
lemma *soldAllocationsEquivalence*:
soldAllocations N Ω = soldAllocations′ N Ω &
soldAllocations′ N Ω = soldAllocations″ N Ω
⟨*proof*⟩

corollary *soldAllocationsEquivalenceVariant*:
soldAllocations = soldAllocations′ &
soldAllocations′ = soldAllocations″ &
soldAllocations = soldAllocations″
⟨*proof*⟩

lemma *allocationSellerMonotonicity*:
soldAllocations (N−{seller}) Ω ⊆ soldAllocations N Ω
⟨*proof*⟩

lemma *allocationsUniverseCharacterization*:
(a ∈ allocationsUniverse) = (a ∈ allAllocations″ (Domain a) (⋃(Range a)))
⟨*proof*⟩

lemma *allocationMonotonicity*:
assumes *N1 ⊆ N2*

shows *allAllocations'' N1* $\Omega \subseteq$ *allAllocations'' N2* Ω
⟨*proof*⟩

lemma *allocationWithOneParticipant*:
assumes $a \in$ *allAllocations'' N* Ω
shows *Domain* $(a -- x) \subseteq N - \{x\}$
⟨*proof*⟩

lemma *soldAllocationIsAllocation*:
assumes $a \in$ *soldAllocations N* Ω
shows $a \in$ *allocationsUniverse*
⟨*proof*⟩

lemma *soldAllocationIsAllocationVariant*:
assumes $a \in$ *soldAllocations N* Ω
shows $a \in$ *allAllocations''* (*Domain a*) $(\bigcup$(*Range a*))
⟨*proof*⟩

lemma *onlyGoodsAreSold*:
assumes $a \in$ *soldAllocations'' N* Ω
shows $\bigcup$ (*Range a*) $\subseteq \Omega$
⟨*proof*⟩

lemma *soldAllocationIsRestricted*:
$a \in$ *soldAllocations'' N* $\Omega =$
$(\exists\, aa.\ aa -- (seller) = a \ \wedge \ aa \in$ *allAllocations''* $(N \cup \{seller\})\ \Omega)$
⟨*proof*⟩

lemma *restrictionConservation*:
$(R +* (\{x\} \times Y)) -- x = R -- x$
⟨*proof*⟩

lemma *allocatedToBuyerMeansSold*:
assumes $a \in$ *allocationsUniverse Domain* $a \subseteq N - \{seller\} \bigcup$ (*Range a*) $\subseteq \Omega$
shows $a \in$ *soldAllocations'' N* Ω
⟨*proof*⟩

lemma *allocationCharacterization*:
$a \in$ *allAllocations N* Ω $=$
$(a \in$ *injectionsUniverse* & *Domain* $a \subseteq N$ & *Range* $a \in$ *all-partitions* $\Omega)$
⟨*proof*⟩

lemma *lm01*:
assumes $a \in$ *soldAllocations'' N* Ω
shows *Domain* $a \subseteq N - \{seller\}$ & $a \in$ *allocationsUniverse*
⟨*proof*⟩

corollary *lm02*:
assumes $a \in soldAllocations'' \ N \ \Omega$
shows $a \in allocationsUniverse \ \& \ Domain \ a \subseteq N - \{seller\} \ \& \ \bigcup \ (Range \ a) \subseteq \Omega$
⟨*proof*⟩

corollary *lm03*:
$(a \in soldAllocations'' \ N \ \Omega) =$
$(a \in allocationsUniverse \ \& \ a \in \{aa. \ Domain \ aa \subseteq N - \{seller\} \ \& \ \bigcup \ (Range \ aa) \subseteq \Omega\})$
(**is** *?L* = *?R*)
⟨*proof*⟩

corollary *lm04*:
$a \in soldAllocations'' \ N \ \Omega =$
$(a \in (allocationsUniverse \cap \{aa. \ Domain \ aa \subseteq N - \{seller\} \ \& \ \bigcup \ (Range \ aa) \subseteq \Omega\}))$
⟨*proof*⟩

corollary *soldAllocationVariantEquivalence*:
$soldAllocations'' \ N \ \Omega = soldAllocations''' \ N \ \Omega$
(**is** *?L*=*?R*)
⟨*proof*⟩

lemma *lm05*:
assumes $a \in soldAllocations''' \ N \ \Omega$
shows $a -- \ n \in soldAllocations''' \ (N - \{n\}) \ \Omega$
⟨*proof*⟩

lemma *allAllocationsEquivalenceExtended*:
$soldAllocations = \ soldAllocations' \ \&$
$soldAllocations' = soldAllocations'' \ \&$
$soldAllocations'' = soldAllocations'''$
⟨*proof*⟩

corollary *soldAllocationRestriction*:
assumes $a \in soldAllocations \ N \ \Omega$
shows $a -- \ n \in soldAllocations \ (N - \{n\}) \ \Omega$
⟨*proof*⟩

corollary *allocationGoodsMonotonicity*:
assumes $\Omega 1 \subseteq \Omega 2$
shows $soldAllocations''' \ N \ \Omega 1 \subseteq soldAllocations''' \ N \ \Omega 2$
⟨*proof*⟩

corollary *allocationGoodsMonotonicityVariant*:
assumes $\Omega 1 \subseteq \Omega 2$
shows $soldAllocations'' \ N \ \Omega 1 \subseteq soldAllocations'' \ N \ \Omega 2$

⟨*proof*⟩

abbreviation *maximalStrictAllocations N Ω b* == *argmax* (*sum b*) (*allAllocations* ({*seller*}∪*N*) *Ω*)

abbreviation *randomBids N Ω b random* == *resolvingBid* (*N*∪{*seller*}) *Ω b random*

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

abbreviation *vcga N Ω b r* == *the-elem* (*vcgas N Ω b r*)

abbreviation *vcga′ N Ω b r* ==
(*the-elem* (*argmax* (*sum* (*randomBids N Ω b r*))
(*maximalStrictAllocations N* (*set Ω*) *b*)))
−− *seller*

lemma *lm06*:
assumes *card* ((*argmax*∘*sum*) (*randomBids N Ω b r*)
((*argmax*∘*sum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set Ω*))))
= *1*
shows *vcga N Ω b r* =
(*the-elem* ((*argmax*∘*sum*) (*randomBids N Ω b r*)
((*argmax*∘*sum*) *b* (*allAllocations* ({*seller*}∪*N*) (*set Ω*)))))
−− *seller*
⟨*proof*⟩

corollary *lm07*:
assumes *card* ((*argmax*∘*sum*) (*randomBids N Ω b r*)
((*argmax*∘*sum*) *b* (*allAllocations* (*N*∪{*seller*}) (*set Ω*))))
= *1*
shows *vcga N Ω b r* = *vcga′ N Ω b r*
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma *lm08*:
assumes *distinct Ω set Ω* ≠ {} *finite N*
shows *card* ((*argmax*∘*sum*) (*randomBids N Ω bids random*)
((*argmax*∘*sum*) *bids* (*allAllocations* (*N*∪{*seller*}) (*set*

Ω)))) = 1
(**is** card ?l=-)
⟨proof⟩

lemma vcgaEquivalence:
assumes distinct Ω set Ω ≠ {} finite N
shows vcga N Ω b r = vcga′ N Ω b r
⟨proof⟩

theorem vcgaDefiniteness:
assumes distinct Ω set Ω ≠ {} finite N
shows card (vcgas N Ω b r) = 1
⟨proof⟩

lemma vcgaDefinitenessVariant:
assumes distinct Ω set Ω ≠ {} finite N
shows card (argmax (sum (randomBids N Ω b r))
(maximalStrictAllocations N (set Ω) b)) =
1
(**is** card ?L=-)
⟨proof⟩

theorem winningAllocationIsMaximal:
assumes distinct Ω set Ω ≠ {} finite N
shows the-elem (argmax (sum (randomBids N Ω b r))
(maximalStrictAllocations N (set Ω) b)) ∈
(maximalStrictAllocations N (set Ω) b)
(**is** the-elem ?X ∈ ?R)
⟨proof⟩

corollary winningAllocationIsMaximalWithoutSeller:
assumes distinct Ω set Ω ≠ {} finite N
shows vcga′ N Ω b r ∈ (Outside′ {seller})‘(maximalStrictAllocations N (set Ω)
b)
⟨proof⟩

lemma maximalAllactionWithoutSeller:
(Outside′ {seller})‘(maximalStrictAllocations N Ω b) ⊆ soldAllocations N Ω
⟨proof⟩

lemma onlyGoodsAreAllocatedAuxiliary:
assumes distinct Ω set Ω ≠ {} finite N
shows vcga′ N Ω b r ∈ soldAllocations N (set Ω)
(**is** ?a ∈ ?A)
⟨proof⟩

theorem onlyGoodsAreAllocated:

assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *vcga N* Ω *b r* ∈ *soldAllocations N* (*set* Ω)
(**is** -∈*?r*)
⟨*proof*⟩

corollary *neutralSeller*:
assumes ∀ *X*. *X* ∈ *Range a* ⟶*b* (*seller*, *X*)=*0 finite a*
shows *sum b a* = *sum b* (*a*−−*seller*)
⟨*proof*⟩

corollary *neutralSellerVariant*:
assumes ∀ *a*∈*A*. *finite a* & (∀ *X*. *X*∈*Range a* ⟶ *b* (*seller*, *X*)=*0*)
shows {*sum b a*| *a*. *a*∈*A*} = {*sum b* (*a* −− *seller*)| *a*. *a*∈*A*}
⟨*proof*⟩

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

lemma *vcgaIsMaximalAux2*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
∀ *a* ∈ *allAllocations* ({*seller*}∪*N*) (*set* Ω). ∀ *X* ∈ *Range a*. *b* (*seller*, *X*)=*0*
(**is** ∀ *a*∈*?X*. -)
shows *sum b* (*vcga*′ *N* Ω *b r*) = *Max*{*sum b a*| *a*. *a* ∈ *soldAllocations N* (*set* Ω)}
⟨*proof*⟩

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

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

corollary *vcgaIsAllocationAllocatingGoodsOnly*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *vcga*′ *N* Ω *b r* ∈ *allocationsUniverse* & ⋃ (*Range* (*vcga*′ *N* Ω *b r*)) ⊆ *set* Ω
⟨*proof*⟩

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

theorem *vcgpDefiniteness*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows ∃! *y*. *vcgp N* Ω *b r n* = *y*
⟨*proof*⟩

lemma *soldAllocationsFinite*:
assumes *finite N finite* Ω
shows *finite* (*soldAllocations N* Ω)
⟨*proof*⟩

The price paid by any participant is non-negative.

theorem *NonnegPrices*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *vcgp N* Ω *b r n* >= (*0*::*price*)
⟨*proof*⟩

lemma *allocationDisjointAuxiliary*:
assumes *a* ∈ *allocationsUniverse* **and** *n1* ∈ *Domain a* **and** *n2* ∈ *Domain a* **and** *n1* ≠ *n2*
shows *a*,,*n1* ∩ *a*,,*n2* = {}
⟨*proof*⟩

lemma *allocationDisjoint*:
assumes *a* ∈ *allocationsUniverse* **and** *n1* ∈ *Domain a* **and** *n2* ∈ *Domain a* **and** *n1* ≠ *n2*
shows *a*,,,*n1* ∩ *a*,,,*n2* = {}
⟨*proof*⟩

No good is assigned twice.

theorem *PairwiseDisjointAllocations*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N* *n1* ≠ *n2*
shows (*vcga′ N* Ω *b r*),,,*n1* ∩ (*vcga′ N* Ω *b r*),,,*n2* = {}
⟨*proof*⟩

Nothing outside the set of goods is allocated.

theorem *OnlyGoodsAllocated*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N g* ∈ (*vcga N* Ω *b r*),,,*n*
shows *g* ∈ *set* Ω
⟨*proof*⟩

11.2 Computable versions of the VCG formalization

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

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

r)

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

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

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

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

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

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

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

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

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

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

lemma *functionCompletion*:
assumes *x* ∈ *Domain f*
shows *toFunction f x* = (*f Elsee 0*) *x*
⟨*proof*⟩

lemma *lm09*:

assumes *fst pair* ∈ *N snd pair* ∈ *Pow* Ω − {{}}
shows *sum* (%*g*. (*toFunction* (*bidMaximizedBy a N* Ω)) (*fst pair*, *g*))
(*finestpart* (*snd pair*)) =
sum (%*g*. ((*bidMaximizedBy a N* Ω) *Elsee 0*) (*fst pair*, *g*))
(*finestpart* (*snd pair*))
⟨*proof*⟩

corollary *lm10*:
assumes *pair* ∈ *N* × (*Pow* Ω − {{}})
shows *summedBid* (*toFunction* (*bidMaximizedBy a N* Ω)) *pair* =
summedBid ((*bidMaximizedBy a N* Ω) *Elsee 0*) *pair*
⟨*proof*⟩

corollary *lm11*:
∀ *pair* ∈ *N* × (*Pow* Ω − {{}}).
summedBid (*toFunction* (*bidMaximizedBy a N* Ω)) *pair* =
summedBid ((*bidMaximizedBy a N* Ω) *Elsee 0*) *pair*
⟨*proof*⟩

corollary *lm12*:
(*summedBid* (*toFunction* (*bidMaximizedBy a N* Ω))) ‘ (*N* × (*Pow* Ω − {{}}))=
(*summedBid* ((*bidMaximizedBy a N* Ω) *Elsee 0*)) ‘ (*N* × (*Pow* Ω − {{}}))
(**is** *?f1* ‘ *?Z* = *?f2* ‘ *?Z*)
⟨*proof*⟩

corollary *lm13*:
summedBidVectorRel (*toFunction* (*bidMaximizedBy a N* Ω)) *N* Ω =
summedBidVectorRel ((*bidMaximizedBy a N* Ω) *Elsee 0*) *N* Ω
⟨*proof*⟩

corollary *maxbidEquivalence*:
summedBidVectorRel (*maxbid a N* Ω) *N* Ω =
summedBidVectorRel (*maxbidAlg a N* Ω) *N* Ω
⟨*proof*⟩

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

corollary *chosenAllocationEquivalence*:
assumes *card N* > *0* **and** *distinct* Ω
shows *chosenAllocation N* Ω *b r* = *chosenAllocationAlg N* Ω *b r*
⟨*proof*⟩

corollary *tiebidsBridgingLemma*:
assumes $x \in (N \times (Pow\ \Omega - \{\{\}\}))$
shows *tiebids a N Ω x = tiebidsAlg a N Ω x*
(**is** *?L=-*)
⟨*proof*⟩

definition *tiebids'=tiebids*
corollary *tiebidsBridgingLemma'*:
assumes $x \in (N \times (Pow\ \Omega - \{\{\}\}))$
shows *tiebids' a N Ω x = tiebidsAlg a N Ω x*
⟨*proof*⟩

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

lemma *resolvingBidEquivalence*:
assumes $x \in (N \times (Pow\ (set\ \Omega) - \{\{\}\}))$ *card N > 0 distinct Ω*
shows *resolvingBid' N Ω b r x = resolvingBidAlg N Ω b r x*
⟨*proof*⟩

lemma *sumResolvingBidEquivalence*:
assumes *card N > 0 distinct Ω a* $\subseteq (N \times (Pow\ (set\ \Omega) - \{\{\}\}))$
shows *sum (resolvingBid' N Ω b r) a = sum (resolvingBidAlg N Ω b r) a*
(**is** *?L=?R*)
⟨*proof*⟩

lemma *resolvingBidBridgingLemma*:
assumes *card N > 0 distinct Ω a* $\subseteq (N \times (Pow\ (set\ \Omega) - \{\{\}\}))$
shows *sum (resolvingBid N Ω b r) a = sum (resolvingBidAlg N Ω b r) a*
(**is** *?L=?R*)
⟨*proof*⟩

lemma *allAllocationsInPowerset*:
allAllocations N Ω $\subseteq Pow\ (N \times (Pow\ \Omega - \{\{\}\}))$
⟨*proof*⟩

corollary *resolvingBidBridgingLemmaVariant1*:
assumes *card N > 0 distinct Ω a ∈ allAllocations N (set Ω)*
shows *sum (resolvingBid N Ω b r) a = sum (resolvingBidAlg N Ω b r) a*
⟨*proof*⟩

corollary *resolvingBidBridgingLemmaVariant2*:
assumes *finite N distinct Ω a ∈ maximalStrictAllocations N (set Ω) b*
shows *sum (randomBids N Ω b r) a = sum (randomBidsAlg N Ω b r) a*
⟨*proof*⟩

corollary *tiebreakingGivesSingleton*:
assumes *distinct Ω set Ω ≠ {} finite N*
shows *card (argmax (sum (randomBidsAlg N Ω b r))*

(*maximalStrictAllocations N* (*set* Ω) *b*)) =
1
⟨*proof*⟩

theorem *maximalAllocationBridgingTheorem*:
assumes *finite N distinct* Ω
shows *maximalStrictAllocations N* (*set* Ω) *b* = *maximalStrictAllocationsAlg N* Ω *b*
⟨*proof*⟩

theorem *vcgaAlgDefinedness*:
assumes *distinct* Ω *set* Ω ≠ {} *finite N*
shows *card* (*argmax* (*sum* (*randomBidsAlg N* Ω *b r*)) (*maximalStrictAllocationsAlg N* Ω *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* =
{*map* (%*x*. (*x*, *sorted-list-of-set*(*a*,,*x*))) ((*sorted-list-of-set* ∘ *Domain*) *a*)}

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

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

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

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

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

file ‹*VCG*−*withoutWrapper.scala*›

end

References

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

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

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

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

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