A contribution to an Auction Theory Toolbox through code and discussion

Marco B. Caminati

Abstract. I am contributing a further mechanization of Vickrey's theorem on weak equilibrium in second price auctions. Submission 6 of stage 1 is an invitation to discuss on early developments in formalizing a toolbox for auction theory, and I wish to add a further viewpoint. My formalization is in Mizar, which is nearer than other systems to the common mathematical language. Auctions being not yet present in the library, there is the chance of taking the first design decisions: one should put effort into making them suitable to build a large future library on them, and at the same time try his best to reduce as much material as possible to the most common mathematical objects, which are well supported in the existing library. I will illustrate how I faced this task.

1 What has been formalized

This AISB submission provides the Mizar formalization of a theorem by Vickrey about weak equilibrium in second price auctions. The mathematics is simply and clearly exposed in [3]. A quick summary follows. The initial data is a vector $\boldsymbol{b}$, containing the bids of each participant. Given this vector, the dynamics of the auction is simply modeled by assuming that each participant has, in his mind, a precise valuation of the auctioned good, which may or may not coincide with the amount of money he bids. The theorem in question says that, in this regard, the best strategy is to make them coincide, in the 'weak' sense: given a random $\boldsymbol{b}$, changing the bid of a participant to his valuation, the payoff of *that* participant does not decrease.

1.1 Defining the payoff to express the theorem

The best possible payoff for a single participant would be given by winning the whole auctioned good without paying anything, in which case it can be quantified by the subjective valuation v he deems the good worth. Given that, generally, any participant gets a fraction ranging from 0 (for losers) to 1 of the auctioned good, and that, of course, any decent auction scheme will impose at least to the winner(s) to disburse an amount of money, such payoff is defined by $vx - p$; here v is the valuation, x the fraction of the good obtained, and p the amount paid.

x and p depend on all the participants' bid, and as such each of them is a component of two distinct vectors having the same length as $\boldsymbol{b}$; $\boldsymbol{x}$ and $\boldsymbol{p}$ are respectively termed the *allocations* and the *payments*, and are calculated from $\boldsymbol{b}$ according to the auction algorithm. Thus, the theorem's wordy statement above can be put into this inequality:[1]

$$v \cdot \mathcal{X}^{II}_{\boldsymbol{b}}(i) - \mathcal{P}^{II}_{\boldsymbol{b}}(i) \leq v \cdot \mathcal{X}^{II}_{\boldsymbol{b}^v_i}(i) - \mathcal{P}^{II}_{\boldsymbol{b}^v_i}(i), \tag{1}$$

where

1. $\mathcal{X}^{II}$ and $\mathcal{P}^{II}$ calculate, from the bids vector $\boldsymbol{b}$, the allocations and payments vectors respectively, according to the second price auction algorithm;
2. i indexes the considered participant;
3. $\boldsymbol{a}^y_i$ is the vector obtained by changing the i-th component of $\boldsymbol{a}$ into y.

2 An overview of Mizar

The Mizar project (http://www.mizar.org) delivers a few provisions:

1. The Mizar *language* permits to write formulas in first-order set theory which read close to common mathematical language. For example, the formula

   $$X \neq \emptyset \Longrightarrow \exists x (x \in X)$$

   is written

   ```
   X <> {} implies ex x st x in X;
   ```

   In addition to the few reserved words pertaining to the first-order alphabet of set theory, the language specifies grammar and reserved words to invoke the verifier (see point 2) and to exploit advanced features of the system.
2. The Mizar *verifier* (PC Mizar) is a piece of software certifying whether one such formula can be deduced (according to some formal system for classical logic, see sections 2.2.1 and 3.5 of [2]) from other given formulas, specified via the keyword `by` of the Mizar language:

   ```
   A1: x in X;
   A2: for y being set holds y in X\/Y iff
                          (y in X or y in Y);
   x in X\/Y by A1, A2;
   ```

3. The Mizar Mathematical Library (MML) builds on the components (1) and (2) above to provide a mass of Mizar language formulas certified, by Mizar verifier, to be derivable from a handful

[1] Consistently with the fact that a vector is a function with a special domain (compare this with the beginning of section 3), we indicate the i-th component of vector $\boldsymbol{a}$ as $\boldsymbol{a}(i)$, reserving the use of sub- and superscripts for other cases.

of set-theoretical axioms affine to ZFC (Zermelo-Fraenkel with the axiom of choice) . The set theory resulting from these axioms, Tarski-Grothendieck (TG), is an extension of ZFC, and more on it can be found in [5].

MML is made up of Mizar source files called *articles*, and its latest version is always browsable at http://mizar.uwb.edu.pl/version/current/mml/. In the following, we will be using typewriter font for referencing articles and results inside MML: for example, XBOOLE_1:4 denotes the fourth theorem appearing in the MML article xboole_1.miz, which is thus viewable at http://mizar.uwb.edu.pl/version/current/mml/xboole_1.miz. We will also adopt typewriter font for Mizar code, as already done in point (1) of the numbered list above.

3 Formalization choices

In this section we pass from common mathematical language of section 1.1 to the corresponding Mizar definitions and statements.

We have to encode the objects appearing in (1): $\mathcal{X}_{\boldsymbol{b}}^{II}$ and $\mathcal{P}_{\boldsymbol{b}}^{II}$. It turns out that a first convenient step is to relax the requirement on $\boldsymbol{b}$: we just ask that it is a relation (not even a function) rather than a natural-indexed vector. A second convenient choice is to restrict the bids to be natural (as opposed to rational or real) numbers. This is not a hindrance, since in real world currency is indeed a positive integer. These choices allow to give the wanted definitions in simple terms of very general, low-level (from a set-theoretical point of view) objects; given a relation R representing our 'vector' of bids:

1. `union rng R` is the highest bid. `rng R` is the range of `R`, and `union rng R` is the union of all the elements of the set `rng R`. This works thanks to our requirement that the bids are natural, because of the so-called Von Neumann encoding of ordinals, which means that 0 is represented as the empty set, 1 is the set $\{0\}$, 2 is the set $\{0,1\}$, and $n+1$ is the set $\{0,1,\ldots,n\}$.
2. Then `R"(union rng R)`, aptly named `topbiddersof R`, is the set of the participants having placed the highest bid. Indeed, `R"Y` is the Mizar rendition of the preimage of the set `Y` under the relation `R`. So that
3. `winnerof R`, defined as

   ```
   the Element of (R"(union rng R))
   ```

   is the winner of the auction. Note that, in Mizar, the construct `the Element of` refers to a fixed element of a set without actually specifying it, hence implicitly employing the axiom of choice. In the present case, this gives a very quick way to randomly extract a winner in case of more than one top-bidder.
4. At this point, `losersof R` is trivially defined as `dom R \ {winnerof R}`. Here, `dom R` is the domain of the relation `R` and `X\Y` is the set-theoretical difference of `X` and `Y`, also called the relative complement of `Y` in `X`.
5. Note that `R|(losersof R)` is still a bid 'vector': this is true exactly because we gave up the requirement of it being a *proper* vector (i.e., having a domain like $\{1,2,\ldots,n\}$). Hence, we can repeat the calculation as from point 1., and conclude that `union (rng (R|(losersof R)))` is the highest non-winning bid, that is, the price to be paid by the winner, by definition of second-price auction. It is aptly named `priceof R`. Here, `R|X` is the restriction of the relation `R` to the set `X`.

Now the needed objects $\mathcal{P}_{\boldsymbol{b}}^{II}$ and $\mathcal{X}_{\boldsymbol{b}}^{II}$ are easy to express in Mizar (where, of course, the argument $\boldsymbol{b}$ becomes a generic relation `R`), respectively as

```
[:dom R,{0}:]+*[:{winnerof R},{priceof R}:]
```

and as

```
[:dom R,{0}:]+*[:{winnerof R},{1}:],
```

whose names are `R-pay` and `R-allocations`.

Only two further set-theoretical definitions are involved here: first, the cartesian product $X \times Y$, which in Mizar is written `[:X,Y:]`. Recalling that, in set theory, relations and functions *are* their own graph, this means that `[:dom R,{0}:]` is the function constantly evaluating to 0 on the whole domain of `R`, and `[:{winnerof R},{y}:]` is the function associating the single element of its domain, `winnerof R`, to the value `y`. Finally, `f +* g` 'pastes' the functions `f` and `g`, evaluating to `g` on the intersection of their domains.[2] So that `R-pay` is a function evaluating to `0` for all the participants, except the winner, for which it evaluates to `priceof R`. Similarly, `R-allocations` is a function evaluating to `0` for all the participants, except the winner, for which it evaluates to `1`. This last occurrence of `1` reveals the simplified case we limit ourselves to: that of an indivisible auctioned good (we had not mentioned this limitation yet).

Luckily, the object `+*` just introduced naturally permits to define also the last operation we need to pass from the definitions to the theorem statement: the single-component alteration introduced in point 3. on page 1. Indeed, $\boldsymbol{f}_x^y$ is easily given by `f +* [:{x},{y}:]`, so that (1) has been formalized and proved in Mizar as:

```
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i -
(f +* [:{i},{v}:])-pay.i,
```

where the reader only needs to know that `g.x` is the function `g` evaluated in the point `x`: it is the Mizar way of denoting $g(x)$. Of course, having been so slack with the requirements on the bids `R` implies that some additional hypotheses are needed. The first one is already present in the code snippet above, hinted by the change of `R` into `f`: the bids is no longer only a relation, but a function. The remaining ones make the theorem actually read as:

```
for f being Function st
(rng f c= NAT\{0} & rng f is finite
& dom f is non trivial) holds
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i
- (f +* [:{i},{v}:])-pay.i
```

A byproduct of this approach is that it makes the points of the proof in which each additional hypothesis is needed self-evident: for example, the requirement of the bids being a relation is to be strengthened into the one that they are a function exactly in lemma `Lm7` (referring to the full Mizar source or to its essential version on page 6). We conclude with few last clarifications: `NAT` is $\mathbb{N}$, `c=` is the set-theoretical inclusion (so that the `rng f c= NAT\{0}` means that `f` is $\mathbb{Z}^+$-valued), and a set is trivial iff it has cardinality smaller than 2.

Table 1 sums up some Mizar notation for reader's convenience.

[2] Note that this still results in a function.

Table 1. Summary of Mizar notations.

Mizar notation	description	blackboard rendition
`X c= Y`	set inclusion	$X \subseteq Y$
`X c< Y`	strict set inclusion	$X \subset Y$
`{}`	the empty set	$\emptyset$
`X\/Y, X/\Y`	pairwise union, intersection	$X \cup Y$, $X \cap Y$
`X\Y`	difference of sets	$X \backslash Y$
`union X`	arbitrary union	$\bigcup_{x \in X} x$
`[:X,Y:]`	cartesian product	$X \times Y$
`NAT`	natural numbers	$\mathbb{N}$
`R"X`	preimage of set X through relation R	$R^{-1}[X]$
`R|X`	restriction of R to X	$R\|_X$
`dom R, rng R`	domain, range of R	
`f.x`	f evaluated in x	$f(x)$
`f +* g`	pasting of functions	$f\|_{\mathrm{dom}\, f \setminus \mathrm{dom}\, g} \cup g$
`+, -, *, /`	arithmetic operations	$+, -, \cdot, /$
`b-allocations`	allocations vector	$\mathcal{X}_{\boldsymbol{b}}^{II}$
`b-pay`	payments vector	$\mathcal{P}_{\boldsymbol{b}}^{II}$

4 Principles behind the formalization choices

Some points of this formalization expose a clash between two needs: that of writing Mizar code in a novice-friendly, closer-to-natural-language style, versus that of reducing the bloat and the coding time; the latter goal implies exploiting the features of both the Mizar system and its foundational axioms to create lean and efficient code. While Mizar language is reputedly among the ones which most resemble common mathematical language [6], and thus one of the most accessible to a novice to the field of mechanized proving, a proficient Mizar user will easily have priorities leading him to sacrifice this aspect.

We point out three of them, as seen useful to discuss the present matters:

1. Evaluating which are the most convenient mathematical objects to reduce the examined objects to.
2. Escaping the typing system when it is a hindrance.
3. Reformulating statements in a style more liable to automations, though possibly farther from natural language.

Let us deepen each point in a dedicated section.

4.1 Which objects to base on?

One of the referees stated a poignant observation which can be a very good starting point for this discussion:

> Aspects of the foundations of Mizar (which is a version of untyped set-theory with 'soft' type system on top) are leaking into the application: e.g., properties of natural numbers that happen to be encoded in the manner of von Neumann are used to express the idea of the maximum of a set. Do mathematicians do this in real life?

In a sense, yes: if they accept ZF as a foundation (as most do), they do such things all the time, albeit generally without thinking (or even knowing) about the underlying set-theoretical machinery. Indeed, the point in introducing all the 'higher level' encodings (as Von Neumann ordinals) is to be able to hide the complexities of ZF modeling, while keeping the comforts given by a solid, time-tested foundation as ZF, and this is usually very convenient in traditional mathematics. When doing *mechanized* mathematics, things change a bit, though: reasoning in terms of the underlying sets of course does not change the essence of the mathematical objects, and permits to get rid of typing when it is a hindrance (see section 4.2). Moreover, doing a formalization in terms of the encoded mathematical objects rather than in terms of the encoding sets usually prevents from taking advantage of the range of canned proofs in the library, which is unavoidably broader in the latter case. Even when a result is not available in the existing library, proving it for sets rather than for the particular objects one is dealing with is of course overall more desirable.

Thus how one encodes the new objects he need is no longer a mere matter of style; it is crucial to how effective and maintainable a formalization will be. The time initially spent to figure out how to translate the involved mathematical object into those already well supported (which typically means low-level with respect to the particular foundations) is a labor usually well repaid, both in terms of how easy the work will result and of the work's impact with regard to the global usefulness of the system's library. In the present case, as already illustrated, we used few general objects, like `union`, `\`, `"`, `|`, `/\`, `[:,:]`, `rng`, `+*`; this reflects in a somewhat long introductory section of `vickrey.miz`, containing general lemmas regarding those objects which are of wider interest. As a bottom line, considerations of a software engineering flavor as the ones above suggest that, in mechanized proving, there are occasions in which, opposedly to standard mathematics, it is convenient forgetting 'higher level' encodings and working on the underlying entities (sets, in the case of set theory).

As an example, let us dwell back on the definition of `priceof R` (see point 5 of page 2):

```
func priceof R equals
union rng (R | losersof R);
```

`union` is a universal set-theoretical operations, but in our cases it does exactly what we need, i.e. taking the maximum, thus obtaining the wanted second price. We focus on this last step, where two crucial choices emerge: first, rather than using the operator `max`, which Mizar provides, we use `union`. It should be noted that in our case the two operate in exactly the same way, with the big difference that `max` is only defined on *numerical* sets: this would imply constraining `R|losersof R` to be numerical-valued, which limits generality and invariably leads to additional work in the proofs involving this object. This work would be spent on something that is mathematically irrelevant, since `max` and `union` do exactly the same: we are discarding the typing because it is limiting us, in this case. Additionally, we would be obliged to import all the definitions regarding `max` and its operands, while `union` is such a basic operation that it needs much less dependencies.

Implicit in such a definition is the second choice: that of limiting the bids to natural numbers. Indeed, the identification of `max` and `union` only holds for *natural* numbers. This is not accidental: natural numbers are simple objects in most formal frameworks, and hence admit simple encodings in term of low-level objects. Thus, this is a symptom of a more general issue: integer, rational and real numbers present escalating complexity in definitions and proofs. Correspondingly, assessing which of these numerical families one will base future code should occur early in the endeavour. In the present case, the first consideration was that currency has always an atomic quantum even on financial exchanges where 'sub-pennying' is allowed. The second consideration was that even if, for some currently unforeseen reason, the case of fractional or even continuous 'currency'

were needed, definitions and theorems could be generalized by embedding them into the correspondent enlargements.

4.2 How much typing?

Contrary to most other systems, typing has no foundational role in Mizar (and in set theory). Assigning a type to each term one talks about was an early response to the foundational crisis of naive set theory. However, although type systems subsequently found extremely widespread application in programming language design to catch errors in software, in its original goal they were largely displaced by ZF set theory, which fixed naive set theory in other ways.

But while programming languages are used to produce executables, formalization languages are used to verify correctness mathematics: correspondingly, the role of a type system changes. In the case of Mizar, this role is twofold: to make the text read more natural (e.g., to be able to write `n is natural number` instead of the equivalent, plainly set-theoretical statement `n in NAT`) and to embed several automation mechanisms in it. When these two aspects are not top-priority, typing should be reduced to a minimum. For a concrete example, let us go back once again to the definition of `priceof`, and assume we ignored the considerations above, conceding a more immediate and typed definition:

```
let R be REAL-valued Relation;
func priceof R equals max rng (R|loserof R);
```

This version would probably be slightly more intelligible than ours for a newcomer, because it saves him to be warned about the proviso that taking the union over finitely many natural numbers equates to taking the maximum. On the other hand, to make such a definition be accepted by Mizar, one first of all has to look for and import all the results making all the types right: one has to know that

- `R|losersof R` is still `REAL-valued`
- the range of a `REAL-valued` is a subset of `REAL`, and thus
- `max` can be applied

This burden is faced every time one invokes `priceof R` in subsequent proofs, while it is utterly bypassed by breaking the abstraction layer and looking down at the sets embodying it.

4.3 Automatically accepted statements

Due to how automations work in Mizar, often there are mathematically equivalent statements presenting different amounts of justification needed to have them accepted by the checker. Usually, the most natural and readable rendition (i.e., the one closest to natural language and easiest to digest for a layman) is not the one minimizing the justifying effort.

We review here few concrete cases, while a general treatment of these kinds of custom exploitation of Mizar automation mechanisms is exposed in [1]. The simplest and most common case is that of statements about the equality of two sets. The equality symbol, =, can be rendered through the set theoretical operation `\+\` and the attribute `empty`, via the result `FOMODEL0:29`:

```
for X, Y being set holds
X \+\ Y is empty iff X=Y;
```

This means that for every theorem whose statement has the form

```
B1: term1 = term2;                (2)
```

one can produce a translation like

```
term1 \+\ term2 is empty;         (3)
```

This latter version presents the advantage of being usable without justification in subsequent proofs, while the former ones require the user to explicitly refer to the label `B1`. This implies consuming a substantial amount of time to locate this label inside the huge MML. For this reason, in the present formalization, the second form prevails over the first, which is nonetheless much more immediate to read. For example, one finds

```
dom f \/ dom g \+\ dom (f +* g) ={};
```

in lieu of

```
dom(f +* g)=dom f \/ dom g by FUNCT_4:def 1;
```

along with more intricate schemes (all documented in [1]) which permit to save lookups to MML at the expense of clarity (because the formula reads less natural *and* because an explicit justification is missing). On the other hand, these developer's shortcut schemes tend to be applied to such trivial passages that the reader, once he is aware of the base idea, should have no problem with them.

5 Formal proof

The Mizar source is organized into three thematic sections[3]: the first one, as mentioned in section 4.1, contains those results which could be induced down to low-level objects, thanks to the approach discussed in section 4.1. For example, `union {x} = x;`, or

```
y<>0 implies
(x=z iff ([:X,{0}:]+*[:{z},{y}:]).x<>0);
```

which says that the function constantly zero except in a point z yields a non-zero value exactly when evaluated in z.

Given their triviality, this whole first section can be omitted when illustrating the proof. The second sections defines a second price auctions and the related concepts, also stating few simple characterizing properties. Third section contains the significant results. Stripping off the proofs, second and third section fit on one single page, attached here (page 6) for the reader to have an overview of the proof design. This make sense because each single Mizar lemma results quite elementary, having proof never exceeding thirty lines.

As already remarked, in most theorems the hypotheses on the bids 'vector', denoted with `R` or `f`, are restrictions which restore some very natural properties we gave up in name of generality and simplicity of definitions. They are usually implicitly assumed in standard treatments, and require, for example, that the bids are numerical, that they are finite and finitely many, or that the participants are more than one.

The key result is `Lm21`. The subsequent theorem, `Lm20`, is introduced only to translate the hypotheses of `Lm21` on the bids in a more direct and usable form, thus producing `Lm22`.

[3] The Mizar keyword `begin` permits starting a new section for the writer's convenience; it is ignored by the verifier.

6 Informal proof

The proof mechanism is slightly different from the ones found in [3, 4]. Here, the cases are parsed not on, e.g., whether the given participant i wins or loses in the original and modified auction. Rather, we focus on the sign of left-hand side (LHS) of inequality (1): if it is non positive, then (1) is obviously true because of lemma 1. Thus, we reduced to the case LHS being positive, which means that i is the winner of the auction $\boldsymbol{b}$ and that $v > \mathcal{P}^{II}_{\boldsymbol{b}}(i) > 0$. Then,

$$\mathcal{P}^{II}_{\boldsymbol{b}^v_i}(i) = \mathcal{P}^{II}_{\boldsymbol{b}}(i) > 0 \tag{4}$$

by lemma 2. This also implies the equality $\mathcal{X}^{II}_{\boldsymbol{b}^v_i}(i) = \mathcal{X}^{II}_{\boldsymbol{b}}(i) = 1$, which, reusing (4) with it, yields[4] LHS=RHS in (1).

This concludes the proof, leaving only the following couple of lemmas to be justified; in what follows, the winner and the number representing the price (given in Mizar by `winnerof` and `priceof`, respectively) of a second price auction starting with input $\boldsymbol{b}$ will be denoted with $\underline{\boldsymbol{b}}$ and $\overline{\boldsymbol{b}}$.

Lemma 1 (`Lm4`)**.** *RHS of* (1) *is not negative.*

Proof. Consider the function

$$\tau := (v - \overline{\boldsymbol{b}^v_i}) \cdot \mathcal{X}^{II}_{\boldsymbol{b}^v_i}.$$

The only possible point at which such a function can yield a negative value is $\underline{\boldsymbol{b}^v_i}$, and this happens if and only if $v < \overline{\boldsymbol{b}^v_i}$. Since RHS is $\tau(i)$, if thesis were false we should then assume $i = \underline{\boldsymbol{b}^v_i}$ and $v < \overline{\boldsymbol{b}^v_i}$, which leads to contradiction: $\overline{\boldsymbol{b}^v_i} \leq \boldsymbol{b}^v_i\left(\underline{\boldsymbol{b}^v_i}\right) = v$. □

Lemma 2 (`Lm5`)**.** *Changing the bid of the winner of a second price auction into a value strictly higher than the auction price does not change the payments vector:*

$$\epsilon > 0 \Longrightarrow \mathcal{P}^{II}_{\boldsymbol{b}^{\overline{\boldsymbol{b}}+\epsilon}_{\underline{\boldsymbol{b}}}} = \mathcal{P}^{II}_{\boldsymbol{b}}$$

Proof. As long as the winner bids strictly more than the second price, he will remain the winner, so that the allocation vector will not vary; moreover, changing the winner's bid will not change the second price, by definition. Hence the thesis, because the payments vector is the allocations vector scalar-multiplied by the price. □

The reader can compare these informal lemmas with their Mizar counterparts, whose labels are reported bracketed.

REFERENCES

[1] M.B. Caminati and G. Rosolini, 'Custom automations in mizar', *Journal of Automated Reasoning*, (2012).

[2] A. Grabowski, A. Korniłowicz, and A. Naumowicz, 'Mizar in a Nutshell', *Journal of Formalized Reasoning*, **3**(2), 153–245, (2010).

[3] M. Kerber, C. Lange, and C. Rowat, 'Vickrey auctions', *http://www.cs.bham.ac.uk/research/projects/formare/code/auction-theory/*, (2013).

[4] Eric Maskin, 'The unity of auction theory: Milgrom's masterclass', *Journal of Economic Literature*, **42**(4), 1102–1115, (2004).

[5] P. Rudnicki and A. Trybulec, 'On equivalents of well-foundedness', *Journal of Automated Reasoning*, **23**(3), 197–234, (1999).

[6] J. Urban, 'MizarMode–an integrated proof assistance tool for the Mizar way of formalizing mathematics', *Journal of Applied Logic*, **4**(4), 414–427, (2006).

[4] Actually, we just proved something stronger than inequality (1); i.e., that LHS $> 0 \Longrightarrow$ LHS $=$ RHS and LHS $< 0 \Longrightarrow$ LHS $<$ RHS

```
begin
:::definitions related to second price auctions, and basic facts

reserve x,y,z,X,Y,i for set, P,Q,R for Relation, f,g for Function, v for Nat;

definition let R;
func topbiddersof R -> Subset of dom R
equals R"{union rng R};
func winnerof R equals
the Element of topbiddersof R;
func losersof R -> Subset of dom R
equals dom R \ {winnerof R};
func priceof R equals
union rng (R | losersof R);
func priceof R -> Element of NAT equals
the Element of {priceof R}/\NAT;
func R-pay -> Function equals
[:dom R,{0}:]+*[:{winnerof R},{priceof R}:];
func R-allocations -> Function equals
[:dom R,{0}:]+*[:{winnerof R},{1}:];
end;

Lm16: priceof R=0 or priceof R=union rng (R|losersof R)

Lm8: (rng R<>{} & rng R c= NAT & rng R is finite)
implies winnerof R in topbiddersof R

Lm7: winnerof f in topbiddersof f
implies f.(winnerof f)=union rng f &
priceof f c= f.(winnerof f)

begin :::the core theorems

Lm3: R-pay.(winnerof R)=priceof R

Lm0: union rng (P|(dom P \ dom Q)) c< union rng Q
implies topbiddersof (P +*1 Q) = Q"{union rng Q}

Lm1: union rng (P | (dom P\{x})) c< X implies
topbiddersof (P +*1 [:{x},{X}:])={x}

Lm5: (priceof R<>0 & i=winnerof R & R-pay.i c< y)
implies (R +*1 [:{i},{y}:])-pay.i = R-pay.i

Lm4: f is NAT-valued & rng f is finite implies
(f+*[:{i},{v}:])-pay.i <= v*((f+*[:{i},{v}:])-allocations.i)

Lm21: :::Vickrey's theorem, version 1
f is NAT-valued & rng f is finite & priceof f <> 0 implies
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i - (f +* [:{i},{v}:])-pay.i

Lm20: (rng R c= NAT & rng R is finite & dom R is non trivial)
implies priceof R in rng R

Lm22: rng f c= NAT\{0} & rng f is finite & dom f is non trivial
implies :::Vickrey's theorem, version 2
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i - (f +* [:{i},{v}:])-pay.i
```