Electronic Notes in Theoretical Computer Science 23 No. 3 (1999)
URL: `http://www.elsevier.nl/locate/entcs/volume23.html` 19 pages

# Building Up Hierarchical Mathematical Domains Using Functors in TH∃OREM∀

Wolfgang Windsteiger [1]

*RISC-Institute*
*SchloßHagenberg*
*A-4232 Hagenberg, Austria.*
*e-mail:* `Wolfgang.Windsteiger@RISC.Uni-Linz.ac.at`

**Abstract**

The world of mathematical domains is structured hierarchically. There are elementary domains and there are well-known techniques how to build up new domains from existing ones. Which of the domains to view as the actual basis of the hierarchy is the freedom of the mathematician who wants to work with these domains and it depends of course on the intention of their use. The strength of the concept lies, however, in the fact that a new domain is constructed from given domains by well-defined rules, which *do not depend* on the actually given domains but only rely on *certain properties* that hold *in the given domains*, and the construction rules guarantee *certain properties for the new domain* then.

In the *Theorema* system, B. Buchberger introduced the concept of *functors* to represent these domain construction rules. A functor can actually be viewed as a function that produces a new domain from given domains by *describing*, which objects and operations are available in the new domain, and by *defining*, how new objects can be constructed from known objects and how operations on the new objects can be carried out using known operations in the underlying domains. Following the general philosophy of the *Theorema* system, functors play a role both in *proving* and in *computing*. The domain description contained in the functor holds structural information, which can be used by a prover for the new domain in order to find an appropriate structure of proofs for formulae containing objects of the new domain. The domain definitions, on the other hand, contain rules, which can be used by an evaluator for executing computations involving objects of the new domain.

In the sequel, we will demonstrate the facilities that are provided in *Theorema* to define functors, to build up a hierarchy of domains using functors, and to do computations in the constructed domains.

[1] Supported by the Austrian Science Foundation (FWF) – project FO-1302 (SFB).

# 1 Introduction

In mathematics there is a rich repertoire of known principles how new mathematical domains can be constructed from existing domains. Applying this technique, one can build up hierarchies of domains starting with some elementary domains in the same spirit like building up theories starting from some axioms. It depends on the intended use of the domains and, of course, also on the personal taste, which domains are chosen to be the elementary domains to form the fundament of the hierarchy. For instance, one can represent "natural numbers" with "zero and the successor function", i.e. as "all objects of the form $0$, $0'$, $0''$, $0'''$, etc." and then give an inductive definition of addition and multiplication based on this representation, call this "the elementary domain of natural numbers", and build up all of mathematics from this starting point. Alternatively, if the basic construction of natural numbers is not in the focus of interest, one can represent "natural numbers" as "built into Mathematica", i.e. as "the objects 0, 1, 2, 3, etc." and define addition and multiplication to be the built-in Mathematica functions "+" ("Plus") and "*" ("Times"), call now this "the elementary domain of natural numbers", and start from this higher level.

Whatever one wants to choose as the point to start from, we want to concentrate on the method, how a new domain can be constructed from some given domain(s). Well known such constructions in mathematics are for instance the cartesian product of two or more domains, the polynomial ring over some domain of coefficients, or the factor structure w.r.t. some congruence relation. The common principle of these constructions is that new objects are constructed from existing objects – the elements in the cartesian product are *tuples of elements* of the involved domains – and that new operations in the new domain are defined in terms of existing operations on the components of the new objects – the sum of two tuples can be defined to be the *tuple containing the sums of corresponding components of the summands*. In [Buchberger 96] and [Buchberger 97] the concept of *functors* was introduced in this context as the common framework to represent *abstract domains* through the construction rules for new objects together with the definition of operations on these new objects. *Concrete domains* are either basic domains (pre-defined *Theorema* domains like natural numbers or tuples) or they can be produced by application of a functor to an already existing concrete domain.

In *Theorema*, a (very easy) functor has the form

$$\mathrm{Functor}[C, \mathrm{any}[a, b, a1, a2, b1, b2],$$

$$\mathbb{S} = \langle + : C \times C \to C \rangle$$

$$\underset{C}{\in}[\langle a, b \rangle] \Leftrightarrow \underset{A}{\in}[a] \wedge \underset{B}{\in}[b] \qquad \text{"}[D \in]\text{"}\ ]$$

$$\langle a1, b1 \rangle \underset{C}{+} \langle a2, b2 \rangle = \langle a1 \underset{A}{+} a2, b1 \underset{B}{+} b2 \rangle \quad \text{"}[D+]\text{"}$$

A functor defined in that way represents an abstract domain $C$ with the following properties: for any $a, b, a1, a2, b1$, and $b2$

**[D$\mathbb{S}$]** the signature of the domain contains a binary function on $C$ denoted "+",

**[D$\in$]** a pair $\langle a, b \rangle$ is element of $C$ if and only if $a$ is an element of $A$ and $b$ is an element of $B$,

**[D+]** the sum of $\langle a1, b1 \rangle$ and $\langle a2, b2 \rangle$ is the tuple $\langle a1 \underset{A}{+} a2, b1 \underset{B}{+} b2 \rangle$, where $\underset{A}{+}$ and $\underset{B}{+}$ represent addition in $A$ and $B$, respectively.

The signature of the domain $C$, which is reflected in property [D$\mathbb{S}$], can be viewed as an abstract *domain description* of $C$, which does not refer to the actual implementation of $C$ . On the other hand, the properties [D$\in$] and [D+] reflect the *domain definitions*, which are given by the *implementation* of the operations in $C$ referring to certain operations in the underlying domains $A$ and $B$ . Functors occur typically in definitions like

$$\mathbf{Definition}[\text{"CartesianProduct"}, \mathrm{any}[A, B],$$

$$\mathrm{CartesianProduct}[A, B] = \mathrm{Functor}[C, \mathrm{any}[a, b, a1, a2, b1, b2], \ldots]]$$

which defines "the cartesian-product-functor". The definition as it is defines, for any (domains) $A$ and $B$, CartesianProduct$[A, B]$ to be a domain $C$ satifying the properties listed above. We can construct a variety of new domains by application of a functor to existing domains, e.g.

$$\mathbf{Definition}[\text{"NxN"}, \mathrm{NxN} = \mathrm{CartesianProduct}[\mathbb{N}, \mathbb{N}]]$$

Using this definition, we can do computations in this new domain "NxN" based on the operations in "the underlying domain $\mathbb{N}$". According to the given definition of the cartesian product, we have a predicate "$\in$" and a binary function "+" for objects in the domain "NxN", and we can do computations like

(1) $$\mathrm{Compute}[\underset{\mathrm{NxN}}{\in}[\langle 3, 1 \rangle], \mathrm{using} \to \mathrm{Definition}[\text{"NxN"}]]$$

(2) $$\mathbb{N}[\in][3] \wedge \mathbb{N}[\in][1]$$

**A note on the** *Theorema* **language:** $\underset{D}{\circ}$ ("an expression $D$ under an operator $\circ$") denotes "the operation $\circ$ in the domain $D$". Internally, the Mathematica expression representing such an operation is `D[∘]`. Each pair of consecutive displayed formulae (like (1) and (2) above) in the text from here on contains *Theorema* input with the corresponding output of the system (in the sequel, system in- and output will not be numbered). For doing computations the keyword "Compute" is used, i.e. calling the function Compute[$e$] computes the value of the expression $e$. A parameter "using $\rightarrow K$" expresses to use knowledge $K$ for the evaluation of the expression, a parameter "built-in $\rightarrow B$" provides the evaluator some built-in knowledge $B$. Typically, knowledge that goes into $K$ consists of definitions, axioms, propositions, etc. given by the user, whereas $B$ normally contains knowledge provided by the *Theorema* system.

$$\text{Compute}[\langle 2,4\rangle \underset{\text{NxN}}{+} \langle 3,1\rangle, \text{using} \rightarrow \text{Definition}[\text{“NxN”}]]$$

$$\langle 2 \underset{\mathbb{N}}{+} 3, 4 \underset{\mathbb{N}}{+} 1\rangle$$

Adding knowledge about the operations in $\mathbb{N}$, we get

$$\text{Compute}[\underset{\text{NxN}}{\in}[\langle 3,1\rangle], \text{using} \rightarrow \text{Definition}[\text{“NxN”}], \text{built-in} \rightarrow \text{Built-in}[\text{“}\mathbb{N}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle 2,4\rangle \underset{\text{NxN}}{+} \langle 3,1\rangle, \text{using} \rightarrow \text{Definition}[\text{“NxN”}], \text{built-in} \rightarrow \text{Built-in}[\text{“}\mathbb{N}\text{”}]]$$

$$\langle 5,5\rangle$$

We will show in the sequel, how we can build up hierarchies of computational domains using this approach, i.e. how computations can be done in domains that are built up by functors. Moreover, as already sketched in [Tomuta 97], we can use the information contained in a functor in order to structure proofs, i.e. the knowledge about the structure of a domain that is constructed using functors can be helpful in proofs in that domain.

## How To Use Functors in Theorema

In *Theorema*, we provide two variants of functors: the first variant – the *introduction functor* – is written in an example in the form

$$
\begin{aligned}
&\mathrm{Functor}\,[C, \mathrm{any}[a, b, a1, a2, b1, b2],\\
&\quad \mathbb{S} = \langle + : C \times C \to C\rangle\\
&\quad \underset{C}{\in}[\langle a, b\rangle] \Leftrightarrow \underset{A}{\in}[a] \wedge \underset{B}{\in}[b] \qquad\qquad \text{“[}D \in\text{]”}\ ]\\
&\quad \langle a1, b1\rangle \underset{C}{+} \langle a2, b2\rangle = \langle a1 \underset{A}{+} a2, b1 \underset{B}{+} b2\rangle \ \text{“[}D{+}\text{]”}
\end{aligned}
$$

The “reading instruction” for such a functor is as follows:

$$
\begin{aligned}
&\text{a domain } C, \text{ such that, for any } a, b, a1, a2, b1, b2\\
&\quad \text{the signature of } C \text{ consists of a function } + \text{ from } C \times C \text{ to } C\\
&\quad \langle a, b\rangle \text{ is an element of } C \text{ iff} \ldots \ \text{“label1”}\\
&\quad \langle a1, b1\rangle \underset{C}{+} \langle a2, b2\rangle = \ldots \qquad\quad \text{“label2”}
\end{aligned}
$$

Note that operations in domains constructed using functors always carry the name of the domain as an underscript. In such a situation, an operation $\underset{C}{+}$ should be read like “+ in C”. This gives the user total control over application of “overloaded operators”, since, whenever “+” is used one has to specify from which domain to use the “+”. The string labels appearing at the right margin can be omitted, they are used only as reference to the individual formulae in the functor. Such references appear for instance in the trace of a computation or in a proof.

The second variant of a functor – the *extension functor* – can be written in the form

$$
\begin{aligned}
&\mathrm{Functor}\,[\langle D, \mathrm{extends}[C]\rangle, \mathrm{any}[a1, a2, b1, b2],\\
&\quad \mathbb{S} = \langle >: D \times D \to \mathbb{T}\rangle\\
&\quad \langle a1, b1\rangle \underset{D}{>} \langle a2, b2\rangle \Leftrightarrow \langle a2, b2\rangle \underset{C}{<} \langle a1, b1\rangle\ ]
\end{aligned}
$$

where the appropriate "reading instruction" is

a domain $D$, that extends the domain $C$ such that, for any $a1, a2, b1, b2$

the signature of $D$ consists of a predicate $>$ on $D \times D$

(in addition to the operations in $C$)

$\langle a1, b1\rangle \underset{D}{>} \langle a2, b2\rangle$ iff ...

These two variants of a functor differ in the fact that the introduction functor introduces a new data-structure for representing the elements of the new domain, whereas the extension functor uses an already existing data-structure and extends it by some new operations. Thus, they reflect the two main techniques for constructing new domains: *introducing* a new data-structure together with a membership predicate and some new operations on the new data-structure or*extending* an existing domain by some new operations. Moreover, comparing the actual *executable text* with the *informal reading instructions*, one can see that the representation of mathematical objects in the *Theorema* language differs only little from what a mathematician is used to read in a mathematical text. A "functor expression" (i.e. an expression of the form "Functor[...]", we will, however, often call such an expression simply "a functor") is a term in the language of "*Theorema* expressions", hence, in a "*Theorema* standard session" it only makes sense for a functor to occur inside some "*Theorema* formal text element", in fact, only the "Definition"-environment is capable of handling functors in an appropriate fashion (see [Buchberger 98] for an explanation of the *Theorema* session concept and [Windsteiger 99] for a survey on *Theorema* expressions, *Theorema* formal text elements, and *Theorema* user-commands). Thus, a functor is typically applied in definitions for domain constructors like the cartesian product of two domains $A$ and $B$:

$$\textbf{Definition}[\text{“CartesianProduct”}, \text{any}[A, B],$$
$$\text{CartesianProduct}[A, B] =$$
$$\text{Functor}[C, \text{any}[a, b, a1, a2, b1, b2], \ldots]]$$

Following the philosophy of the *Theorema* standard session, there are no side-effects at the moment of entering the definition during a *Theorema* session, i.e. the definition above is read as *one equality* for CartesianProduct$[A, B]$. Inspecting the internal representation [2] of this definition

$$\text{Definition}[\text{“CartesianProduct”}]$$

[2] The bullets are used in the internal representation of *Theorema* expressions as the first letter in the name of a data-structure. Roughly speaking, the definition consists of a label, a range for free variables, a condition on the free variables, and a formula list containing labeled formulae.

$$\bullet def[\text{"}CartesianProduct\text{"},$$

$$\bullet range[\bullet simpleRange[A], \bullet simpleRange[B]], True,$$

$$\bullet flist[\bullet lf[\text{""}, \mathrm{CartesianProduct}[A,B] = {}^{\mathrm{TM}}\,\mathrm{Functor}[\ldots]]]]$$

we see that $\mathrm{CartesianProduct}[A, B]$ is equal to a functor with the ${}^{\mathrm{TM}}\mathrm{Functor}[\ldots]$ object containing all information given in the functor. The knowledge represented by this is

**[Dom]** $\mathrm{CartesianProduct}[A, B]$ is a functor (of a certain structure).

A functor definition, however, implicitely contains much more information, namely an *algorithmic definition* of all operations available in the new domain. In the example above, this is

**[CP∈]** $\underset{\mathrm{CartesianProduct}[A,B]}{\in}[\langle a, b\rangle] \Leftrightarrow \ldots$ ("Membership in CartesianProduct[$A, B$]")

**[CP+]** $\langle a1, b1\rangle \underset{\mathrm{CartesianProduct}[A,B]}{+} \langle a2, b2\rangle = \ldots$ ("Plus in CartesianProduct[$A, B$]")

Uncovering this implicit knowledge does not happen immediately, it is delayed until the moment the *definition is used* in a proof or in a computation, where the ${}^{\mathrm{TM}}\mathrm{Functor}[\ldots]$ object is *expanded*. In this respect, the processing of a functor and the way the knowledge contained in a functor is applied differs drastically from earlier implementations in earlier development stages of the *Theorema* system, see [Buchberger 96].

## 2 How Functors are Processed in Theorema

Functors can be used in proving and computing. Both proving and computing are activities that involve a knowledge base that has to be processed accordingly, where, in case of proving, this means to translate the possibly nested knowledge base into a flat list of assumptions (i.e. *Theorema* formulae), and, in case of computing, this amounts to translate the possibly nested knowledge base into a flat list of Mathematica transformation rules to be applied during evaluation. In both cases, the crucial operation is "flattening the nested structure", which will also be the appropriate place for expansion of functors.

There are at least two possibilities how to process a functor:

(i) one can *adjoin* the knowledge about the domain operations ([CP∈] and [CP+]) to the definition of the domain ([Dom]) or

(ii) the knowledge about the domain operations can *replace* the definition of the domain.

Flattening the nested structure is done as a preprocessing step both for proving and computing by the *Theorema*-command "FlattenKB" [3], which recognizes

[3] FlattenKB produces, roughly, an assumption list consisting of labeled formulae, which

an option named "ExpandFunctors". Reflecting the possible choices how to process functors, the values for this option can be either "adjoin" (default) or "replace".

$$\text{FlattenKB}[\text{Definition}[\text{“CartesianProduct”}], \text{ExpandFunctors} \rightarrow \text{“adjoin”}]$$

$$\bullet asml\,[\bullet lf\,[\text{“}[D\in]\text{”}, \forall_{A,B,a,b}(\text{CartesianProduct}[A,B][\in][\langle a,b\rangle] \Leftrightarrow A[\in][a] \wedge B[\in][b])],$$

$$\bullet lf\,[\text{“}[D+]\text{”}, \forall_{A,B,a1,a2,b1,b2}(\text{CartesianProduct}[A,B][+][\langle a1,b1\rangle,\langle a2,b2\rangle] = \langle A[+][a1,a2], B[+][b1,b2]\rangle)],$$

$$\bullet lf\,[\text{“”}, \forall_{A,B}(\text{CartesianProduct}[A,B] =^{\mathrm{TM}} \text{Functor}[C,\ldots])]]$$

*Adjoining* the domain operations is the choice in proving since in a proof one can benefit from the knowledge about the domain structure that can be extracted from the functor. Furthermore, we do not only want to prove *properties of the operations in a domain*, but we might want to prove properties of an entire domain itself, e.g. we might want to prove that CartesianProduct$[\mathbb{N},\mathbb{N}]$ is a vector space. Moreover, from this list of assumptions one can detect that CartesianProduct$[A,B]$ is a domain constructed by a functor, which can be used in order to call a special prover that is designed for proving properties in functor-defined structures. Typically, this type of knowledge is not relevant for computation, since neither there is a special evaluation technique for operations defined by a functor nor is there the desire to compute CartesianProduct$[A,B]$. Thus, for computation we choose to *replace* the domain definition by the domain operations.

$$\text{FlattenKB}[\text{Definition}[\text{“CartesianProduct”}], \text{ExpandFunctors} \rightarrow \text{“replace”}]$$

$$\bullet asml\,[\bullet lf\,[\text{“}[D\in]\text{”}, \forall_{A,B,a,b}(\text{CartesianProduct}[A,B][\in][\langle a,b\rangle] \Leftrightarrow A[\in][a] \wedge B[\in][b])],$$

$$\bullet lf\,[\text{“}[D+]\text{”}, \forall_{A,B,a1,a2,b1,b2}(\text{CartesianProduct}[A,B][+][\langle a1,b1\rangle,\langle a2,b2\rangle] = \langle A[+][a1,a2], B[+][b1,b2]\rangle)]]$$

Note again, that the domain underscipt of the operator symbols like $\underset{C}{\in}$ *and*

contain a label and a formula.

$\underset{C}{+}$ is represented internally as C[∈] and C[+], respectively. This will work together neatly with the default evaluation mechanism of Mathematica when we want to use functors in a computational session as well.

# 3 Examples: System Demonstration

In this section, we will show, how the concepts described above can be used to built up a hierarchy of domains for doing computations. As an example, we use the domain of multivariate polynomials over some coefficient domain (a similar example is presented in [Buchberger 99]). We represent multivariate polynomials as ordered tuples of monomials, where each monomial is a pair consisting of a coefficient and a power product. Power products are represented as tuples of exponents. The example will show that this approach combines the techniques used in mathematics to elegantly define mathematical domains by well-known constructions with common techniques used in modern programming environments like overloading, inheritance, etc. We start off with the rationals, which will later be used as the coefficient domain in the construction of the polynomial domain. For deciding membership in the domain of rationals we use the *Theorema*-predicate "IsRational" and we define the arithmetic operations in such a way that the available operations provided by Mathematica are used. (Note, however, that some basic domains like natural numbers, integer numbers, or rational numbers will be provided as built-in knowledge in *Theorema*.)

**Definition**[“Basic Rationals”,

Basic-Rationals[] = Functor[$F$, any[$c, d$],

$$\mathbb{S} = \langle 0 : F, + : F \times F \to F, - : F \to F, - : F \times F \to F, 1 : F, * : F \times F \to F,$$
$$/ : F \times F \to F, | : F \times F \to \mathbb{T},$$
$$>: F \times F \to \mathbb{T}, \succ: F \times F \to \mathbb{T}\rangle$$

$$\underset{F}{\in}[c] \Leftrightarrow \mathrm{IsRational}[c]$$

$$\underset{F}{0} = 0$$

$$\underset{F}{1} = 1$$

$$c \underset{F}{+} d = c + d$$

$$\underset{F}{-} c = -c \qquad ]]$$

$$c \underset{F}{-} d = c - d$$

$$c \underset{F}{*} d = c * d$$

$$c \underset{F}{/} d = c/d$$

$$c \underset{F}{|} d \Leftrightarrow (c \neq 0 \vee (d = 0))$$

$$c \underset{F}{>} d \Leftrightarrow c > d$$

$$c \underset{F}{\succ} d \Leftrightarrow (c \neq 0 \wedge (d = 0))$$

Whenever we have an order relation “>” in a domain, we can easily define “≥”, “<”, “≤” in terms of “>”. An extension functor can be used for this purpose.

$$\textbf{Definition}\,[\text{“Operator Variants”}, \text{any}[S],$$

$$\text{Operator-Variants}[S] = \text{Functor}\,[\langle T, \text{extends}[S]\rangle, \text{any}[x, y],$$

$$\mathbb{S} = \langle \geq\colon T \times T \to \mathbb{T}, \not>\colon T \times T \to \mathbb{T}, \not\geq\colon T \times T \to \mathbb{T},$$
$$<\colon T \times T \to \mathbb{T}, \leq\colon T \times T \to \mathbb{T}, \not<\colon T \times T \to \mathbb{T}, \not\leq\colon T \times T \to \mathbb{T}\rangle$$

$$x \underset{T}{\geq} y \Leftrightarrow x \underset{S}{>} y \vee x = y$$

$$x \underset{T}{\not>} y \Leftrightarrow \neg(x \underset{S}{>} y)$$

$$x \underset{T}{\not\geq} y \Leftrightarrow \neg(x \underset{S}{>} y \vee x = y) \qquad ]]$$

$$x \underset{T}{<} y \Leftrightarrow y \underset{S}{>} x$$

$$x \underset{T}{\leq} y \Leftrightarrow y \underset{S}{>} x \vee x = y$$

$$x \underset{T}{\not<} y \Leftrightarrow \neg y \underset{S}{>} x$$

$$x \underset{T}{\not\leq} y \Leftrightarrow \neg y \underset{S}{>} x \vee x = y$$

Now we can construct a new domain $\mathbb{Q}$ as the rational numbers extended by the operator variants of “$>$”.

$$\textbf{Definition}[\text{“}\mathbb{Q}\text{”}, \mathbb{Q} = \text{Operator-Variants}[\text{Basic-Rationals}[]]]$$

Immediately, we can do some easy computations in $\mathbb{Q}$ using this definition as knowledge base. In a *Theorema* standard session, this is done as follows:

$$\text{Compute}[\underset{\mathbb{Q}}{\in}[3], \text{using} \to \text{Definition}[\text{“}\mathbb{Q}\text{”}]]$$

$$True$$

$$\text{Compute}[\underset{\mathbb{Q}}{0}, \text{using} \to \text{Definition}[\text{“}\mathbb{Q}\text{”}]$$

$$0$$

$$\text{Compute}[2 \underset{\mathbb{Q}}{*} 5/3, \text{using} \to \text{Definition}[\text{“}\mathbb{Q}\text{”}]]$$

$$\frac{10}{3}$$

$$\text{Compute}[2 \underset{\mathbb{Q}}{>} 5/3, \text{using} \to \text{Definition}[\text{“}\mathbb{Q}\text{”}]]$$

$$True$$

$$\text{Compute}[2 \underset{\mathbb{Q}}{<} 5/3, \text{using} \to \text{Definition}[\text{“}\mathbb{Q}\text{”}]]$$

$$False$$

$$\text{Compute}[2 \underset{\mathbb{Q}}{\not\leq} 2, \text{using} \rightarrow \text{Definition}[\text{“}\mathbb{Q}\text{”}]]$$

$$False$$

Next, we define exponent tuples over a domain $S$ of length $n$. In order to be used as power products in the domain of polynomials, we need an order relation “$>$” on exponent tuples, furthermore, we define a binary predicate “$|$” as “divisibility of exponent tuples”.

$$\textbf{Definition}[\text{“Exponent Tuples”}, \text{any}[S, n],$$

$$\text{Exponent-Tuples}[S, n] = \text{Functor}[T, \text{any}[\bar{e}, e, E, F],$$

$$\mathbb{S} = \langle 1 : T, * : T \times T \rightarrow T, / : T \times T \rightarrow T, | : T \times T \rightarrow \mathbb{T}, >: T \times T \rightarrow \mathbb{T} \rangle$$

$$\underset{T}{\in}[\langle \bar{e} \rangle] \Leftrightarrow |\langle \bar{e} \rangle| = n \wedge \forall_{i=1,\ldots,n} \underset{S}{\in}[\langle \bar{e} \rangle_i]$$

$$\neg \underset{T}{\in}[e]$$

$$\underset{T}{1} = \langle \underset{S}{0} |_{i=1,\ldots,n} \rangle$$

$$]]$$

$$E \underset{T}{*} F = \langle E_i \underset{S}{+} F_i |_{i=1,\ldots,n} \rangle$$

$$E \underset{T}{/} F = \langle E_i \underset{S}{-} F_i |_{i=1,\ldots,n} \rangle$$

$$E \underset{T}{|} F \Leftrightarrow \forall_{i=1,\ldots,n} E_i \underset{S}{\leq} F_i$$

$$E \underset{T}{>} F \Leftrightarrow \exists_{i=1,\ldots,n} \left( \forall_{j=1,\ldots,i-1} E_j = F_j \wedge E_i \underset{S}{>} F_i \right)$$

The domain $\mathbb{E}$ from now on denotes the exponent tuples over $\mathbb{N}$ of length 4.

$$\textbf{Definition}[\text{“}\mathbb{E}\text{”}, \mathbb{E} = \text{Exponent-Tuples}[\mathbb{N}, 4]]$$

$$\text{Compute}[\underset{\mathbb{E}}{1}, \text{using} \rightarrow \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$\langle 0, 0, 0, 0 \rangle$$

$$\text{Compute}[\langle 0, 1, 2, 11 \rangle \underset{\mathbb{E}}{*} \langle 2, 3, 2, 1 \rangle, \text{using} \rightarrow \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$\langle 2,4,4,12\rangle$$

$$\text{Compute}[\langle 0,1,2,11\rangle \underset{\mathbb{E}}{|} \langle 2,3,2,1\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$False$$

$$\text{Compute}[\langle 0,1,2,11\rangle \underset{\mathbb{E}}{|} \langle 2,3,2,12\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle 0,1,2,11\rangle \underset{\mathbb{E}}{>} \langle 2,3,2,12\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$False$$

$$\text{Compute}[\langle 10,1,2,11\rangle \underset{\mathbb{E}}{>} \langle 2,3,2,12\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle 2,3,2,13\rangle \underset{\mathbb{E}}{>} \langle 2,3,2,12\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{E}\text{”}]]$$

$$True$$

The computations with “|” and “>” also show the power of the *Theorema* language: the definitions given with quantifiers appear as they would be written in a mathematical text. However, as long as they have a finite range they can be used in computations immediately. Monomials can now be defined as pairs of a coefficient and a power product. For carrying on the example, we define $\mathbb{M}$ to be the domain of monomials over $\mathbb{Q}$ and $\mathbb{E}$ from above.

**Definition** [“Monomials as Pairs”, any$[C, P]$,

MonsP$[C, P]$ = Functor $[M$, any$[c, d, p, q]$,

$$\mathbb{S} = \langle 0 : M, + : M \times M \to M, - : M \to M, 1 : M, * : M \times M \to M,$$
$$/ : T \times T \to T, | : T \times T \to \mathbb{T}, >: T \times T \to \mathbb{T}\rangle$$

$$\underset{M}{\in}[\langle c, p\rangle] \Leftrightarrow \underset{C}{\in}[c] \wedge \underset{P}{\in}[p]$$

$$\underset{M}{0} = \langle \underset{C}{0}, \underset{P}{1}\rangle$$

$$\langle c, p\rangle \underset{M}{+} \langle d, p\rangle = \langle c \underset{C}{+} d, p\rangle$$

$$\underset{M}{-} \langle c, p\rangle = \langle \underset{C}{-} c, p\rangle \qquad ]]$$

$$\underset{M}{1} = \langle \underset{C}{1}, \underset{P}{1}\rangle$$

$$\langle c, p\rangle \underset{M}{*} \langle d, q\rangle = \langle c \underset{C}{*} d, p \underset{P}{*} q\rangle$$
$$\langle c, p\rangle \underset{M}{/} \langle d, q\rangle = \langle c \underset{C}{/} d, p \underset{P}{/} q\rangle$$

$$\langle c, p\rangle \underset{M}{|} \langle d, q\rangle \Leftrightarrow c \underset{C}{|} d \wedge p \underset{P}{|} q$$
$$\langle c, p\rangle \underset{M}{>} \langle d, q\rangle \Leftrightarrow p \underset{P}{>} q$$

**Definition**[“$\mathbb{M}$”, $\mathbb{M}$ = MonsP$[\mathbb{Q}, \mathbb{E}]$]

As expected, we can now compute with monomials.

$$\text{Compute}[\underset{\mathbb{M}}{\in}[\langle 3, \langle -23, 0\rangle\rangle], \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$False$$

$$\text{Compute}[\underset{\mathbb{M}}{\in}[\langle 3, \langle 2, 0, 1, 1\rangle\rangle], \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle 1, \langle 0, 1, 2, 4\rangle\rangle \underset{\mathbb{M}}{+} \langle 1, \langle 1, 2, 2, 4\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$\langle 1, \langle 0, 1, 2, 4\rangle\rangle \underset{\mathbb{M}}{+} \langle 1, \langle 1, 2, 2, 4\rangle\rangle$$

(This is perfectly alright since only monomials with identical power products can be added according to our definition!)

$$\text{Compute}[\langle 1, \langle 0, 1, 2, 4\rangle\rangle \underset{\mathbb{M}}{+} \langle 1, \langle 0, 1, 2, 4\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$\langle 2, \langle 0, 1, 2, 4\rangle\rangle$$

$$\text{Compute}[\langle 34, \langle 17, 2, 0, 1\rangle\rangle \underset{\mathbb{M}}{*} \langle 22, \langle 17, 3, 8, 24\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$\langle 748, \langle 34, 5, 8, 25\rangle\rangle$$

$$\text{Compute}[\langle 34, \langle 17, 2, 0, 1\rangle\rangle \underset{\mathbb{M}}{|} \langle 22, \langle 17, 3, 8, 24\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle 34, \langle 17, 2, 0, 1\rangle\rangle \underset{\mathbb{M}}{>} \langle 22, \langle 17, 3, 8, 24\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$False$$

$$\text{Compute}[\langle 22, \langle 17, 3, 8, 24\rangle\rangle \underset{\mathbb{M}}{>} \langle 34, \langle 17, 2, 0, 1\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{M}\text{”}]]$$

$$True$$

Finally, we go for polynomials as ordered tuples of monomials. The following functor defines some basic operations on polynomials[4]:

**Definition**[“Polynomials as Ordered Tuples of Monomials”, any$[M]$,

$$\mathrm{PolyOTM}[M] = \mathrm{Functor}\,[P, \mathrm{any}[m, m1, m2, \bar{m}, n, \bar{n}, p, q],$$

$$\mathbb{S} = \langle 0 : P, + : P \times P \to P, - : P \to P, * : P \times P \to P, 1 : P, >: P \times P \to \mathbb{T}\rangle$$

$$\underset{P}{\in}[\langle\rangle]$$

$$\underset{P}{\in}[\langle m\rangle] \Leftrightarrow \underset{M}{\in}[m] \wedge m \neq \underset{M}{0}$$

$$\underset{P}{\in}[\langle m1, m2, \bar{m}\rangle] \Leftrightarrow \underset{M}{\in}[m1] \wedge m1 \neq \underset{M}{0} \wedge m1 \underset{M}{>} m2 \wedge \underset{P}{\in}[\langle m2, \bar{m}\rangle]$$

$$\underset{P}{0} = \langle\rangle$$

$$\langle\rangle \underset{P}{+} q = q \qquad ]]$$

$$p \underset{P}{+} \langle\rangle = p$$

$$\langle m, \bar{m}\rangle \underset{P}{+} \langle n, \bar{n}\rangle =$$

$$\| \; m \smile (\langle\bar{m}\rangle \underset{P}{+} \langle n, \bar{n}\rangle) \Longleftarrow m \underset{M}{>} n,$$

$$n \smile (\langle m, \bar{m}\rangle \underset{P}{+} \langle\bar{n}\rangle) \Longleftarrow n \underset{M}{>} m,$$

$$\text{where}\left[s = m \underset{M}{+} n, \| \; s \smile (\langle\bar{m}\rangle \underset{P}{+} \langle\bar{n}\rangle) \Longleftarrow s \neq \underset{M}{0}, \langle\bar{m}\rangle \underset{P}{+} \langle\bar{n}\rangle \; \|\right] \|$$

$\vdots$ (to be continued)

[4] The expression $\| \; e_1 \Longleftarrow c_1, \ldots, e_n \Longleftarrow c_n \; \|$ denotes a case distinction: if $c_1$ then $e_1$, else if $\ldots$, else if $c_n$ then $e_n$. In the last alternative, the condition can be ommitted, in which case *True* is assumed. $e \smile t$ prepends $e$ to the tuple $t$, i.e. $e \smile \langle t_1, \ldots, t_n\rangle := \langle e, t_1, \ldots, t_n\rangle$

$\vdots$ (continuation)

$$\underset{P}{-}\langle\rangle = \langle\rangle$$

$$\underset{P}{-}\langle m, \bar{m}\rangle = \underset{M}{-} m \smile \underset{P}{-}\langle \bar{m}\rangle$$

$$\underset{P}{1} = \langle \underset{M}{1} \rangle$$

$$\langle\rangle \underset{P}{*} q = \langle\rangle$$

$$p \underset{P}{*} \langle\rangle = \langle\rangle$$

$$\langle m, \bar{m}\rangle \underset{P}{*} \langle n, \bar{n}\rangle = \text{where}\Big[ mn = m \underset{M}{*} n, r = \langle m\rangle \underset{P}{*} \langle \bar{n}\rangle \underset{P}{+} \langle \bar{m}\rangle \underset{P}{*} \langle n, \bar{n}\rangle,$$
$$\| \, mn \smile r \Leftarrow mn \neq \underset{M}{0}, r \, \| \Big]$$

$$\neg \left( \langle\rangle \underset{P}{>} p \right)$$

$$\langle m, \bar{m}\rangle \underset{P}{>} \langle\rangle$$

$$\langle m, \bar{m}\rangle \underset{P}{>} \langle n, \bar{n}\rangle \Leftrightarrow m \underset{M}{>} n \vee (m = n \wedge \langle \bar{m}\rangle \underset{P}{>} \langle \bar{n}\rangle)$$

$$\textbf{Definition}[\text{“}\mathbb{P}\text{”}, \mathbb{P} = \text{PolyOTM}[\mathbb{M}]]$$

$$\text{Compute}[\langle\langle 1, \langle 2, 0, 1, 3\rangle\rangle\rangle \underset{\mathbb{P}}{+} \langle\langle 3, \langle 2, 0, 1, 3\rangle\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{P}\text{”}]]$$

$$\langle\langle 4, \langle 2, 0, 1, 3\rangle\rangle\rangle$$

$$\text{Compute}[\underset{\mathbb{P}}{\in}[\langle\langle 4, \langle 2, 0, 1, 3\rangle\rangle\rangle], \text{using} \to \text{Definition}[\text{“}\mathbb{P}\text{”}]]$$

$$True$$

$$\text{Compute}[\langle\langle 1, \langle 2, 0, 1, 3\rangle\rangle\rangle \underset{\mathbb{P}}{+} \langle\langle 3, \langle 3, 0, 1, 3\rangle\rangle\rangle, \text{using} \to \text{Definition}[\text{“}\mathbb{P}\text{”}]]$$

$$\langle\langle 3, \langle 3, 0, 1, 3\rangle\rangle, \langle 1, \langle 2, 0, 1, 3\rangle\rangle\rangle$$

(From now on, we omit the specification of the used knowledge, since it does not change anymore.)

$$\text{Compute}[\langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 1,\langle 2,0,1,3\rangle\rangle\rangle \underset{\mathbb{P}}{+} \langle\langle 3,\langle 2,1,1,0\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$\langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 3,\langle 2,1,1,0\rangle\rangle,\langle 13,\langle 2,0,1,3\rangle\rangle\rangle$$

$$\text{Compute}[\langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 1,\langle 2,0,1,3\rangle\rangle\rangle \underset{\mathbb{P}}{>} \langle\langle 3,\langle 2,1,1,0\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$True$$

$$\text{Compute}[\langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 1,\langle 2,0,1,3\rangle\rangle\rangle \underset{\mathbb{P}}{>} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$False$$

$$\text{Compute}[\langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 1,\langle 2,0,1,4\rangle\rangle\rangle \underset{\mathbb{P}}{>} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$True$$

$$\text{Compute}[\underset{\mathbb{P}}{0} \underset{\mathbb{P}}{>} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$False$$

$$\text{Compute}[\underset{\mathbb{P}}{0} \underset{\mathbb{P}}{<} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$\langle\rangle \underset{\substack{\text{PolyOTM[MonsP[Operator-Variants[Basic-Rationals[]],} \\ \text{Exponent-Tuples}[\mathbb{N},4]]]}}{<} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle$$

Clear, we have no "<" in the domain $\mathbb{P}$, but we can get one if we extend the domain by the operator variants of ">".

$$\text{Compute}[\underset{\mathbb{P}}{0} \underset{\text{Operator-Variants}[\mathbb{P}]}{<} \langle\langle 3,\langle 3,0,1,3\rangle\rangle,\langle 12,\langle 2,0,1,3\rangle\rangle\rangle]$$

$$True$$

## 4 Conclusion

*Theorema* functors provide a way to define new domains in an elegant way. In fact, a functor *describes* how to *construct* a new domain from zero or more given domains. The elegance of this concept lies in the combination of *mathematical style* used in the definitions contained in the functor (quantifiers, logical connectives, etc.) and – at the same time – *immediate executability* of those operations in the new domain, that are given in an algorithmic way. Compared to object-oriented programming languages, the functor concept is very similar to the "class template" concept of C++, which describe parametrized classes. The functor definition *describes* the construction of a new domain as much as a class template definition *describes* the construction of a new class. On the other hand, application of a functor actually *constructs* a new domain just like a new class is generated from a class template as soon as the template is instanciated. Whereas instanciation of templates is done at compile-time the evaluation of operations defined by functors is done at run-time. Even more, in order to have more control over evaluation we employ our own evaluation mechanism written in the Mathematica language instead of using Mathematica's built-in evaluator. Due to this, we have enourmous computation times for comparably trivial operations as soon as domains are deply nested. However, for tutorial purposes the *Theorema* functors can be used nicely to show "how mathematics works". For the future, a compiler for parts of the *Theorema* language is planned, which can help to speed up such computations significantly.

## References

[Buchberger 96] B. Buchberger: *Symbolic Computation: Computer Algebra and Logic*. In: Frontiers of Combining Systems (F. Baader, K.U. Schulz eds.), pp. 193 - 220. Applied Logic Series. Kluwer Academic Publishers, 1996.

[Buchberger 97] B. Buchberger: *The Theorema prover for equalities over the natural numbers*. In: Proceedings of "The First *Theorema* Workshop", RISC-report 97-20, 1997.

[Buchberger 98] B. Buchberger: *Theorema: Theorem Proving for the Masses*. Invited Talk at the Worldwide *Mathematica* Conference, Chicago, June 18-21, 1998.

[Buchberger 99] B. Buchberger: *Theorema: A Progress Report*. Colloquium Talk at GMD, Bonn, June 1999.

[Tomuta 97] E. Tomuta: *Using Functors in Organizing Proofs*. In: Proceedings of "The First *Theorema* Workshop", RISC-report 97-20, 1997.

[Windsteiger 99] W. Windsteiger: *Theorema: Overview on Using the System and Details on Composing Hierarchical Knowledge Bases*. Presentation in the frame of the "School for Logic and Computation", Edinburgh, April 9-13, 1999. RISC-report 99-15, 1999.