ROMA TRE
Università degli Studi di Roma Tre
Dipartimento di Discipline Scientifiche
Via della Vasca Navale, 84 – 00146 Roma, Italy.

Uniform Representation of Basic Algebraic Structures in Computer Algebra

Carla Limongelli†, Giuseppina Malerba†, Marco Temperini‡

RT-INF-18-96 **Maggio 1996**

† Università di Roma Tre,
Via della Vasca Navale, 84
00146 Roma, Italy
e-mail: limongel@inf.uniroma3.it, malerba@dis.uniroma1.it

‡ Dip. di Informatica e Sistemistica - Università "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: marte@dis.unrioma1.it

This work was partially supported by MURST 40% "Calcolo Algebrico e Simbolico"

ABSTRACT

This work presents a methodological framework suitable for the design of innovative symbolic computation systems. A significant set of basic algebraic structures is defined and implemented by a uniform representation based on the algebraic structure of truncated power series. By following this approach we succeed in avoiding the need for explicit coercion in computing with the defined structures. We provide a CLOS prototype implementation.

1 Introduction

This work aims to propose a neat methodological framework suitable for the design of innovative symbolic computation systems.

We describe the specification and the prototype implementation of some basic algebraic structures such as algebraic numbers, multivariate polynomials and square matrices, represented in a *uniform* way. By uniform we intend that the mathematical entities at issue are represented by a set of inheritance related classes, in an object-oriented language, such that their data structure representation is *structurally compatible*. This compatibility allows for an important effect of uniform representation: the prototype software system we present can execute several legal computations over instances of our structures, without having to perform coercions.

In particular we can perform an operation between a number and a polynomial, as well as a polynomial and a matrix, without any need for explicit coercions.

We give a specification of the three basic algebraic structures cited above, by expressing their common algebraic nature of Ring and Truncated Power Series (TPS) structures.

This specification is translated in the definition of a hierarchy of data structures, in which numbers, polynomials and matrices are uniformly represented as specializations of the TPS.

The idea of a uniform representation for numbers and polynomials has been already presented in [5]. Here we develop further the notion of uniform representation and its application. In particular, we provide an interpretation of matrices over ring elements, as TPS. In order to obtain a sound specification of all these structures, we have to solve some problems, that mainly depend on the nature of matrices and of their operations. In particular, from our specification, it comes out that *external* multiplication[1] cannot be safely expressed in our framework. In order to accomplish this requirement we have to model explicitly our matrix structure as a (specialization of) the algebraic structure of R-module [6]. On the other hand, the ring operations between matrices (matrix plus a matrix, a matrix times a matrix) are simply expressed by their definition in the matrix structure.

After the specification of our structures is given (by following the formalism defined in [4]), we present their one-to-one implementation in suitable classes. In this implementation the derivation of a structure from another is expressed by specialization inheritance. CLOS [2] is the programming language used. It has been chosen for a number of reasons: because of its object orientation; because it has been already used within a project, in which we have been involved, aiming to the development of a language for symbolic computation systems [3], and for our involvement in another research activity aiming to provide an object-oriented language with *class reasoning* features [1]. In presenting the prototype implementation, we show some software details and some significant computational examples.

In the following section, we explain the formalism used for the specification of our algebraic structures. Sec. 3 describes the organization of implementation. In Sec. 4 we

[1] Given a matrix M with elements in R we mean the multiplication of $a \in R$ by M.

discuss some computational examples, and Sec. 5 previews future work.

The details about specification of our structures and some significant parts of the CLOS code are shown in appendix.

2 Classification of formal structures

The classification of formal structures can be given in an algebraic context, as proposed in [4], by defining three planes for their static definition: abstract, parametric and ground plane. In the abstract plane, classical algebraic structures are described by inherited properties. Sorts are not specified, and only symbolic computations are performed. In the parametric plane, parametric structures are defined: they enrich the definition of abstract structures by partial (parameterized) sorts, and by additional operations and properties. In the ground plane, the ground structures take place: they are completely specified, and both symbolic and numeric computations can be performed in this plane.

This classification points out the different specification requirements of structures laying on different planes, leading to a higher level of correctness in their treatment.

The whole discussion about this approach is given in [4]. In the following we provide an example of specification of the abstract structure *Ring* and of its derived structure $Unit - Ring$. Let us note that the specification includes also properties of operators even if they cannot be exploited at the moment. This choice is due to our work being developed in a general research framework [1] and doesn't affect the coherence of this exposition.

Ring	
from	*Abelian Group*
from	*Semiring*
properties	$\forall a, b, c \in \$, a * (b + c) = a * b + a * c$
	$\forall a, b, c \in \$, (a + b) * c = a * c + b * c$
	$\forall s_1, s_2 \in \$, s_1 - s_2 = s_1 + inv(s_2)$
Unit Ring	
from	*Ring*
operations	$1 : \rightarrow \$$
properties	$\forall a \in \$,\ a * 1 = 1 * a = a$

Usually an abstract class is a generalization of the properties that are common to a collection of more specialized classes. It can also be a particular specialization of other abstract classes (the main feature is the fact that methods either are not implemented or they are partially implemented). A parametric class inherits methods and properties from one or more abstract classes, but methods and sorts are parametrically specified. In a ground class all the sorts and methods are completely specified. Direct object instantiations of mathematical objects are possible only for ground classes (actually such an instantiation would be meaningless for abstract classes, and simply not possible for a parametric class, without providing further type arguments for the actualization parameters).

3 Algebraic numbers, polynomials and matrices by TPS

Let N, P and M symbols standing for the representation of, respectively, algebraic numbers, multivariate polynomials and matrices. In the following we will deal with structures of "type" $\mathcal{T}$, by the following notation:

$$\mathcal{T} ::= N|P\mathcal{T}|M\mathcal{T}.$$

We consider, for instance, polynomials over N (we will denote their set by PN), polynomials over polynomials over N (PPN) or matrices whose elements are polynomials over N (MPN).

EXAMPLE 3.1 The matrix

$$M_1 = \begin{pmatrix} \frac{1}{9}x & \frac{7}{2}x \\ \\ \frac{4}{5}x & \frac{4}{3} \end{pmatrix}$$

is a MPN.

As a matter of fact algebraic numbers, polynomials with ring coefficients and matrices over ring elements are ring structures, hence $\mathcal{T}$ always denotes a ring structure. In order to reach a uniform representation of these structures, we need a suitable parametric structure that inherits its algebraic properties from the ring structure. To this purpose we use TPS.

In [5] the idea of the representation of numbers and polynomials via TPS is described. A rational number is viewed as a TPS with a numeric base p and coefficients in $\mathbb{Z}_p$, whereas a bivariate polynomial in x and y is a TPS with base y and coefficients in univariate polynomials with base x. However in the cited reference the formal specification and implementation of this structure was still undeveloped.

It is also possible to express a square matrix as a TPS, as follows: let's call I the formal base of the power series; the coefficients of the series represent the rows of the matrix. Any row of the matrix can be represented as a TPS with a formal base J. The coefficients of this last series represent the elements of the matrix itself.

So, given the following matrix

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

whose elements belong to $\mathcal{T}$, we can define a one-to-one correspondence with the following TPS:

$$(a\ J^0 + b\ J^1)I^0 + (c\ J^0 + d\ J^1)I^1.$$

EXAMPLE 3.2 Let us consider the matrix

$$M_2 = \begin{pmatrix} \frac{2}{3} & \frac{3}{4} \\ \\ \frac{1}{6} & \frac{1}{3} \end{pmatrix}.$$

We can represent it by the following series

$$((4 \cdot 5^0 + 1 \cdot 5^1 + 3 \cdot 5^2 + 1 \cdot 5^3)\, J^0 + (2 \cdot 5^0 + 1 \cdot 5^1 + 1 \cdot 5^2 + 1 \cdot 5^3)\, J^1))I^0 +$$
$$((1 \cdot 5^0 + 4 \cdot 5^1 + 0 \cdot 5^2 + 4 \cdot 5^3)\, J^0 + (2 \cdot 5^0 + 3 \cdot 5^1 + 1 \cdot 5^2 + 3 \cdot 5^3)\, J^1)\, I^1,$$

where we have chosen the base 5 and the truncation order 4, to represent the numerical coefficients.

EXAMPLE 3.3 The matrix M_1 is represented by the following series

$$(((0\ 0\ 0\ 0,\ 0)x\hat{}0\ +\ (4\ 2\ 0\ 1,\ 0)x\hat{}1)J^0 + ((0\ 0\ 0\ 0,\ 0)x\hat{}0\ +\ (1\ 3\ 2\ 2,\ 0)x\hat{}1)J^1)I^0 +$$
$$(((0\ 0\ 0\ 0,\ 0)x\hat{}0\ +\ (4\ 0\ 0\ 0, -1)x\hat{}1)J^0 + ((3\ 3\ 1\ 3,\ 0)x\hat{}0\ +\ (0\ 0\ 0\ 0,\ 0)x\hat{}1)J^1)I^1,$$

where we have represented the innermost series by Hensel codes, with base 5 and truncation order 4.

In order to describe a full specification of the structure $\mathcal{T}$, first we point out some observations related to the algebraic operations that has to be defined.

1. The addition operation and the representation of additive unit are the same for numbers, polynomials and matrices; on the other hand multiplication over matrices, as it is well known, cannot follow the *convolution rule* like numbers and polynomials do.

2. In case of matrices, we have to define the *external product*, that allows for multiplication between a matrix and an element (called *scalar*) that belongs to the ring of the matrix elements. So, the algebraic structure appropriate for representing matrices has to be a R-module, besides that a ring.

3. With respect to polynomials and numbers, the multiplicative unit has a different representation for matrices. Actually matrix unit appears having nothing to do with the polynomial (number) neutral element: it is the matrix having all 1 along the main diagonal, while the polynomial (number) unit is represented by the TPS having 1 as first coefficient and 0 as others.

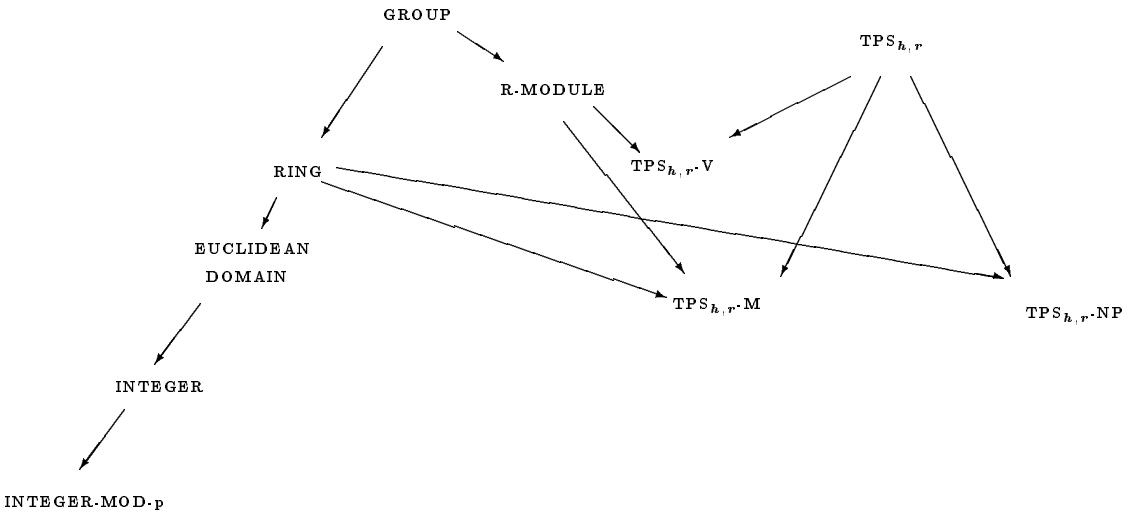

Figure 1: Structures organization

From these observations, it derives that we have to distinguish between two different rings: one for numbers and polynomials and another one for matrices.

Nevertheless the concrete representation of these mathematical objects is always a TPS. We can exploit this similarity, by introducing a new parametric structure $TPS_{h,r}$, where h is the base of the series and r is the truncation order. By this structure we can unify matrices, numbers and polynomials from a formal point of view, but we cannot consider $TPS_{h,r}$ as an algebraic structure itself, since it lacks algebraic operations. These operations will be introduced in two different parametric structures, $TPS_{h,r}$-NP and $TPS_{h,r}$-M: they represent, respectively, polynomials (numbers) and matrices.

In the following, every actualization of TPS will be denoted by $TPS[\$, h, r]$, where \$ is the (actualized) sort of the series coefficients, h represents the base of the series and r the truncation order.

$TPS_{h,r}$, $TPS_{h,r}$-NP, $TPS_{h,r}$-M stand for the plain (not actualized) parametric structures. In Fig. 1 the hierarchy of our structures is shown: abstract structures, such as *R-module* and *Ring*, provide their algebraic characterization. Note that $TPS_{h,r}$ has no algebraic characterization, since it does not inherit from other structures; it only contributes to the structural representation of $TPS_{h,r}$-NP, $TPS_{h,r}$-M and $TPS_{h,r}$-V. Let us note that $TPS_{h,r}$-V is only used for supporting some internal operations of $TPS_{h,r}$-M, as suggested by the specification given in appendix A.

3.1 Implementation

In order to describe the proposed kernel we will focus on the main structures involved in the program: they are implemented by suitable classes. These classes define the necessary algebraic structures and represent the computational domains; they are abstract, parametric and ground classes corresponding to abstract, parametric and ground structures

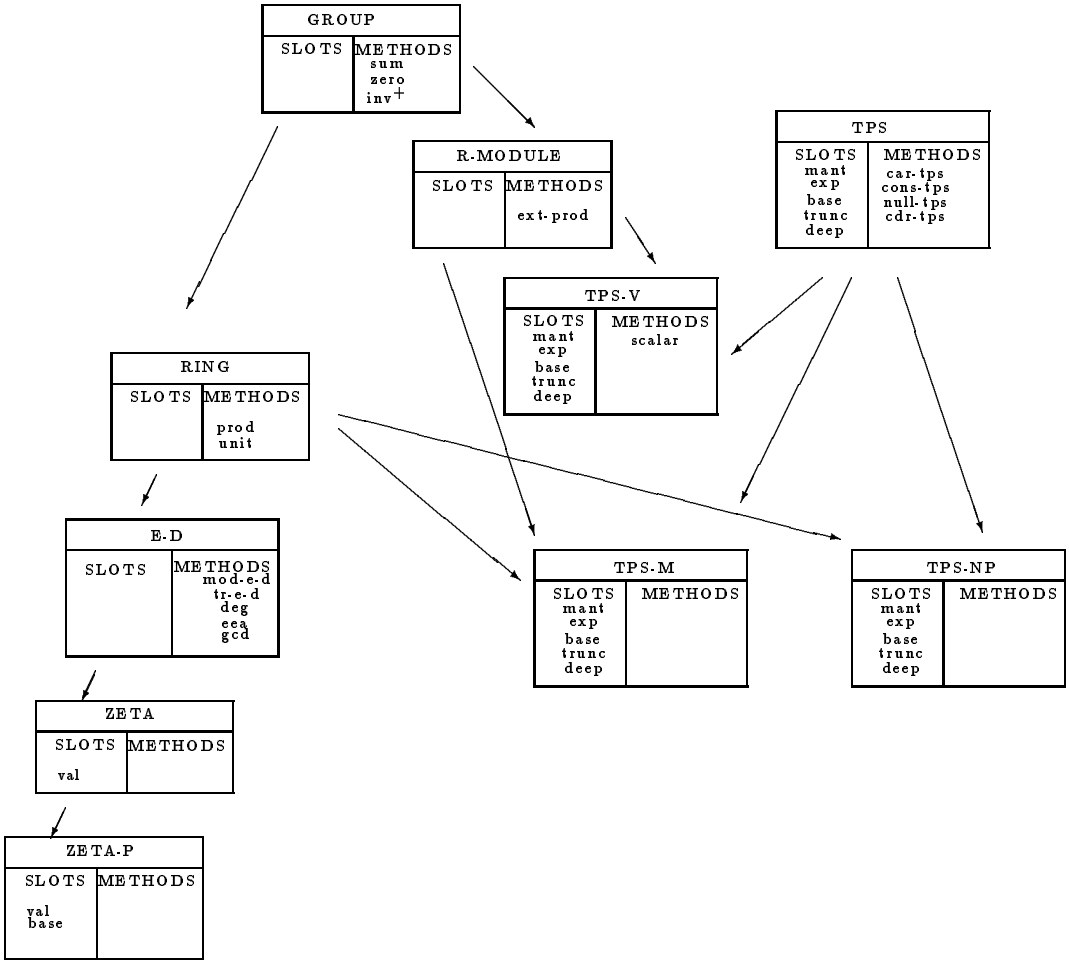

Figure 2: Classes organization.

presented in Sec. 2.

In order to organize the implementation in a way that matches the given design, we have to consider a class and its related methods as a unique entity. In this way the algebraic operations of a given structure correspond to polymorphic methods of the respective class. Fig. 2 shows this organization.

We will consider the classes above defined on the basis of their specialization level. Abstract classes (*GROUP*, *R-MODULE*, *RING*, *E-D*) have a purely declarative role; so they have no slots, and their methods are defined but not implemented, because these classes don't provide object instanciation.

Parametric classes (*TPS*, *TPS*-NP, *TPS*-V, *TPS*-M) implement parametric structures introduced in this paper. All these classes are structured in a similar way since all of them represent TPS. Their internal organization follows:

mant it is the main information contained in instances of *TPS* class, i.e. a list that represents the series coefficients. Each coefficient can be defined either, recursively,

as an instance of TPS or as an instance of $ZETA$-P, according with the specification of parametric structure $TPS_{h,r}$;

exp it contains the exponent (an integer) of the factor that multiplies the whole series;

base this slot contains the base of the series that we are going to represent. It can be either numeric (namely an instance of $ZETA$, while representing an Hensel code) or symbolic (when the base is not a ground structure);

trunc it is a natural number that represents the truncation order of the power series;

deep an auxiliary slot for indicating the partial length of the coefficients list manipulated during methods execution. It is a positive integer initialized to 0.

Ground classes ($ZETA$, $ZETA$-P) represent integers and mod$p$ integers. Their slots *val* contain the value of the represented object; $ZETA$-P objects have also slot *base* that contains the value p.

In appendix B we present the classes definitions. Moreover we show the implementation of a sample method included in an abstract class (the GCD algorithm).

Next section provides some significant examples of computations.

4 Examples

EXAMPLE 4.1 We want to represent the rational number 2/3 by fixing the numerical base equal to 5 and the truncation order for the series equal to 4.

The system creates an instance of the class TPS-NP, that represents the truncated power series:

$$2/3 = 4 \cdot 5^0 + 1 \cdot 5^1 + 3 \cdot 5^2 + 1 \cdot 5^3$$

where

mant contains a list formed by 4 instances of the class $ZETA$-P each of them has into the slot *val* respectively the values 4, 1, 3, 1 and into the slot *base* an instance of $ZETA$ that represents the number 5;

exp contains 0 since there are no powers as common factor in the series;

base contains an instance of $ZETA$ that represents the integer number 5;

deep is initialized to 0.

Program execution follows.

```
REPRESENTATION AND MANIPULATION OF NUMBERS, POLYNOMIALS AND MATRICES
BY TRUNCATED POWER SERIES


> Select one of the following options :
1. Object representation by TPS
2. Arithmetics operations
3. Quit
> 1
> matrices ? (y or n) n
> truncation order ? (natural number)  4
> base ? (prime number)  5
> A = ? 2/3
> Structure of A :TPS[Z5, 5,4]
> Show numeric coefficients as Hensel codes?
  (y or n) n
> Representation of A :4*5^0+1*5^1+3*5^2+1*5^3
```

EXAMPLE 4.2 Given the polynomial

$$P(x,y) = xy + 3x + 2y^2$$

with base 5 and truncation order 4, first the system organizes $P(x,y)$ as a TPS w.r.t. y

$$P(x,y) = (0 \cdot x^0 + 3 \cdot x^1)y^0 + (0 \cdot x^0 + 1 \cdot x^1)y^1 + (2 \cdot x^0)y^2$$

and then it creates P as an instance of the class TPS-NP where numeric coefficients are expresses as truncated power series by Hensel codes.

Let us see the slots of the obtained object, except slot *deep* that works only for very low level operations:

mant contains a list formed by 4 instances belonging to class TPS-NP;

exp contains the integer 0;

base contains string "y";

trunc contains the value 4;

Let us now consider the instances contained into the coefficients list of P. All of them have the same structure:

mant contains a list formed by 4 instances, belonging to the class TPS-NP;

exp contains the integer value 0;

base contains the string "x";

trunc contains the value 4;

If we go on with this "recursive visit" we arrive at the most nested level that is the *ground* level, related to numeric coefficients.

The program execution is the following:

```
REPRESENTATION AND MANIPULATION OF NUMBERS, POLYNOMIALS AND MATRICES
BY TRUNCATED POWER SERIES

> Select one of the following options :
1. Object representation by TPS
2. Arithmetics operations
3. Quit
> 1
> matrices ? (y or n) n
> truncation order ? (natural number)  4
> base ? (prime number)  5
> A = ? 3x^1+x^1*y^1+2y^2
> Structure of A : TPS[TPS[TPS[Z5,5,4],x,4],y,4]
> Show numeric coefficients as Hensel codes?
  (y or n) y
```

```
> Representation of A :
  ((0 0 0 0, 0)x^0 + (3 0 0 0, 0)x^1)y^0 +
 +((0 0 0 0, 0)x^0 + (1 0 0 0, 0)x^1)y^1 +
 +((2 0 0 0, 0)x^0 + (0 0 0 0, 0)x^1)y^2
```

EXAMPLE 4.3 With base 5 and truncation order 4, the representation of rational number $2/3$ is

$$s_1 = (4 \cdot 5^0 + 1 \cdot 5^1 + 3 \cdot 5^2 + 1 \cdot 5^3)$$

while the polynomial

$$P(x) = \frac{3}{2}x^0 + \frac{3}{4}x^1$$

is represented by

$$s_2 = (4 \cdot 5^0 + 2 \cdot 5^1 + 2 \cdot 5^2 + 2 \cdot 5^3)x^0 + (2 \cdot 5^0 + 1 \cdot 5^1 + 1 \cdot 5^2 + 1 \cdot 5^3)x^1.$$

The invocation of *sum* method over s_1 and s_2 gives the power series

$$(3 \cdot 5^0 + 4 \cdot 5^1 + 0 \cdot 5^2 + 4 \cdot 5^3)x^0 + (2 \cdot 5^0 + 1 \cdot 5^1 + 1 \cdot 5^2 + 1 \cdot 5^3)x^1$$

that is the representation of the following polynomial:

$$\frac{13}{6}x^0 + \frac{3}{4}x^1.$$

Program execution follows:

```
REPRESENTATION AND MANIPULATION OF NUMBERS, POLYNOMIALS AND MATRICES
BY TRUNCATED POWER SERIES


> Select one of the following options :
1. Object representation by TPS
2. Arithmetics operations
3. Quit
> 2
> matrices ? (y or n) n
> truncation order ? (natural number)  4
```

```
> base ? (prime number)  5
> A = ? 2/3
> Structure of A : TPS[Z5,5,4]
> Show numeric coefficients as Hensel codes?
  (y or n) n
> Representation of A:
  4*5^0+1*5^1+3*5^2+1*5^3
> B = ? 3/2x^0+3/4x^1
> Structure of B :TPS[TPS[Z5,5,4],x,4]
> Show numeric coefficients as Hensel codes?
  (y or n) n
> Representation of B :
  (4*5^0+2*5^1+2*5^2+2*5^3)x^0+
 +(2*5^0+1*5^1+1*5^2+1*5^3)x^1
> Select one of the following options :
1. A+B
2. A-B
3. -A
4. -B
5. A*B
> 1
> Structure of A+B : TPS[TPS[Z5,5,4],x,4]
> Show numeric coefficients as Hensel codes?
  (y or n) n
> Representation of A+B :
  (3*5^0+4*5^1+0*5^2+4*5^3)x^0+
 +(2*5^0+1*5^1+1*5^2+1*5^3)x^1
```

EXAMPLE 4.4 Let us consider the matrices

$$M_1 = \begin{pmatrix} \frac{1}{9}x & \frac{7}{2}x \\ \\ \frac{4}{5}x & \frac{4}{3} \end{pmatrix}$$

and

$$M_2 = \begin{pmatrix} \frac{2}{3} & \frac{3}{4} \\ \\ \frac{1}{6} & \frac{1}{3} \end{pmatrix}$$

given in Sect. 2; we want to execute the addition of these objects.

First the system creates M_1, an instance of the class TPS-M, whose slots are:

mant contains a list formed by 2 instances belonging to the class TPS-NP;

exp contains the integer 0;

base contains string `"I"`;

trunc contains the value 2

and M_2, an instance of the same class. Then the invocation of the method *sum* creates M_3, an instance belonging to the class TPS-M that contains the representation of the addition of M_1 and M_2.
The program execution is the following:

```
REPRESENTATION AND MANIPULATION OF NUMBERS, POLYNOMIALS AND MATRICES
BY TRUNCATED POWER SERIES
> Select one of the following options :
1. Object representation by TPS
2. Arithmetics operations
3. Quit
> 2
> matrices ? (y or n) y
> Select one of the following options :
1. A+B
2. A-B
3. -A
4. -B
5. A*B
6. v*A (ext-prod)
```

```
> 1
> order ? (natural number)  2
> a[11]=? 0x^0+1/9x^1
> a[12]=? 0x^0+7/2x^1
> a[12]=? 0x^0+4/5x^1
> a[22]=? 4/3x^0+0x^1
> Structure of A :TPS[TPS[TPS[TPS[Z5,5,4],x,4],J,2],I,2]
> Show numeric coefficients as Hensel codes?
  (y or n) y
> Representation of A:
  (((0 0 0 0, 0)x^0+(4 2 0 1, 0)x^1)J^0+
  +((0 0 0 0, 0)x^0+(1 3 2 2, 0)x^1)J^1)I^0+
  (((0 0 0 0, 0)x^0+(4 0 0 0,-1)x^1)J^0+
  +((3 3 1 3, 0)x^0+(0 0 0 0, 0)x^1)J^1)I^1
> b[11]=? 2/3
> b[12]=? 3/4
> b[21]=? 1/6
> b[22]=? 1/3
> Structure of B :TPS[TPS[TPS[Z5,5,4],J,2],I,2]
> Show numeric coefficients as Hensel codes?
  (y or n) y
> Representation of B :
  ((4 1 3 1, 0)J^0+(2 1 1 1, 0)J^1))I^0+
 +((1 4 0 4, 0)J^0+(2 3 1 3, 0)J^1))I^1
> Structure of A+B :TPS[TPS[TPS[TPS[Z5,5,4],x,4],J,2],I,2]
> Show numeric coefficients as Hensel codes?
  (y or n) y
> Representation of A+B :
  (((4 1 3 1, 0)x^0+(4 2 0 1, 0)x^1)J^0+
  +((2 1 1 1, 0)x^0+(1 3 2 2, 0)x^1)J^1)I^0+
  (((1 4 0 4, 0)x^0+(4 0 0 0,-1)x^1)J^0+
  +((2 3 1 3, 0)x^0+(0 0 0 0, 0)x^1)J^1)I^0
```

5 Conclusion

We have presented methodological and practical issues related to the design and the development of a symbolic computation system.

The methodological aspects concern the definition of basic mathematical structures under an object-oriented specification model. Such specification expresses and profits of a common (algebraic) nature of algebraic numbers, polynomials and matrices. This common nature is twofold: first, those structures are special Ring structures (and this gives the main common algebraic layer of our specification); moreover the instances of those structures are representable by TPS (and this allows us to speak in terms of *uniform representation*).

The practical aspect of our work is the prototype implementation, devised for exploiting the cited common nature of our structures.

We plan to develop further this work, aiming to define a more complete system in which also efficiency aspects are taken into account. This could not be done at the prototype stage of the implementation we have described in this paper. The choice of CLOS, was motivated by the need for a fast prototype programming tool, for developing our experiment.

We are confident that the set of structures we have implemented so far is large enough to allow for experimenting with several algebraic computational problems.

References

[1] G. Cioni, A. Colagrossi, and M. Temperini. Class reasoning in object oriented languages for symbolic computation. Technical report, Istituto di Analisi dei Sistemi ed Informatica IASI-CNR, November 1995. Rep. n. 416.

[2] Sonya E. Keene. *Object Oriented Programming in Common Lisp.* Addison Wesley Publishing Company, 1989.

[3] C. Limongelli, A. Miola, and M. Temperini. Design and implementation of symbolic computation systems. In P.W. P.W. Gaffney and E. N. Houstis, editors, *IFIP TC2/WG 2.5 Working Conference on Programming Environments For High Level Scientific Problem Solving.* North–Holland, september 1991.

[4] C. Limongelli and M. Temperini. Abstract specification of structures and methods in symbolic mathematical computation. *Theoretical Computer Science*, 104:89–107, October 1992. Elsevier Science Publisher.

[5] C. Limongelli and M. Temperini. On the uniform representation of mathematical data structures. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems, International (Symposium Disco '93)*, volume 722 of *LNCS*. Springer Verlag, 1993.

[6] Henryk Minc and Marvin Marcus. *Introduction to Linear Algebra.* Macmillan, New York, 1965.

A Specification description

A.1 Specification description

The basic sets useful for our specification are defined as follows:

- $\mathbb{N} = \{\text{set of naturals}\}$
- $\mathcal{I}_n = \{i \in \mathbb{N} : 0 \leq i \leq n-1\}$ with $n \in \mathbb{N}$
- $\mathbb{Z} = \{\text{set of integers}\}$
- $\mathbb{Z}^+ = \{z \in \mathbb{Z} : z \geq 0\}$
- $\mathcal{P} = \{s \in \mathbb{Z}^+ \setminus \{0,1\} : \forall\, t \in \mathbb{Z}^+ \quad \text{with either} \quad s \; mod \; t = 0 \;\rightarrow\; t = s \quad \text{or} \quad t = 1\}$
- $\mathbb{Z}_m = \{i \in \mathbb{Z}^+ : 0 \leq i \leq m-1\}$ with $m \in \mathbb{Z}^+$

Given $p \in \mathcal{P}$ and $r \in \mathbb{N}$, the parametric structures $TPS_{h,r}$, $TPS_{h,r}$-NP, $TPS_{h,r}$-V, $TPS_{h,r}$-M are defined as follows:

SORTS: here we give the sorts involved in the specification.

$\mathcal{I}_r,\ \mathbb{Z},\ \mathbb{Z}^+,\ \mathbb{Z}_p$	They represent, respectively, the index set, the set of integer numbers, the set of positive integers and the set of integers modulo p,
$X = \{x_0, x_1, \ldots x_{r-1}\}$	the alphabet containing the symbolic bases of TPS representing polynomials,
$X' = X \bigcup \{p\}$	the alphabet containing the bases of TPS representing numbers and polynomials.
$\mathcal{X} = X' \bigcup \{I, J\}$	Finally here are the bases of TPS representing numbers, polynomials and matrices

Given $h \in \mathcal{X}$, $then$ $\$ = \begin{cases} \mathbb{Z}_p & \text{if} \quad h = p \\ TPS_{h,r} & \text{if} \quad h \in \mathcal{X} \setminus \{p\} \end{cases}$

Note that, in case of nested power series, the base of the innermost one is a number $p \in \mathbb{Z}^+$.

In the following Fig. 3 we will give the specification of the operations related to $TPS_{h,r}$. We have already noted that this set does not have any algebraic characteristic.

The two following figures 4 and 5 represent the operations and properties related to the specification of $TPS_{h,r}$-NP.

In Fig. 6 the specification of $TPS_{h,r}$-V is presented. Let us note that this set contains some operations necessary to handle rows in a matrix.

Fig. 7 shows the specification of square matrices.

$TPS_{h,r} = \{< a_i, h^i >_{i \in \mathcal{I}_r} \quad h \in \mathcal{X}, \quad a_i \in \$\}$

OPERATIONS:

nth : $TPS_{h,r} \times \mathcal{I}_r \longrightarrow \$$ selects an element of the series

base: $TPS_{h,r} \longrightarrow \mathcal{X}$ returns the base of the series

PROPERTIES:

$\forall i \in \mathcal{I}_r, \quad \forall s = < a_i, h^i >_{i \in \mathcal{I}_r} \in TPS_{h,r}$

1. $nth(s, i) = a_i$
2. $base(s) = h$

Figure 3: Operations and properties of $TPS_{h,r}$

$TPS_{h,r}$-NP= $\{< a_i, h^i >_{i \in \mathcal{I}_r} \quad h \in X', \quad a_i \in \$\}$

OPERATIONS:

ZERO	*from RING on* $TPS_{h,r}$-NP *redefining* 0 *as* 0_{NP}
UNIT	*from RING on* $TPS_{h,r}$-NP *redefining* 1 *as* 1_{NP}
ADDITION	*from RING on* $TPS_{h,r}$-NP *redefining* $+$ *as* $+_{NP}$
MULT	*from RING on* $TPS_{h,r}$-NP *redefining* $*$ *as* $*_{NP}$
INVERSE	*from RING on* $TPS_{h,r}$-NP *redefining inv as* inv_{NP}

Figure 4: Operations for $TPS_{h,r}$-NP

PROPERTIES: all the properties defined in RING are inherited. The properties needed to express the semantics of the operators listed above are shown. $\forall i \in \mathcal{I}_r,\ \forall s \in TPS_{h,r}$-NP

ZERO A series is zero when all its coefficients are zero: $nth(0_{NP}, i) = 0_{\$}$

UNIT is a series that has unit element as first coefficient and zero for further coefficients:

$$nth(1_{NP}, i) = \begin{cases} 1_{\$} & \text{if } i = 0 \\ 0_{\$} & \text{if } i > 0 \end{cases}$$

ADDITION it behaves as usual by adding the coefficients of terms that have the same power:

$\forall s_1, s_2 \in TPS_{h,r}$-NP

$$s_1 +_{NP} s_2 = < a_i, h^i >_{i \in \mathcal{I}_r} \text{ where:}$$

$$a_i = \begin{cases} nth(s_1, i) +_{\$} nth(s_2, i) & \text{if } \$ = TPS_{h,r} \\ nth(s_1, i) +_{\mathbb{Z}_p} nth(s_2, i) +_{\mathbb{Z}_p} rip_i & \text{if } \$ = \mathbb{Z}_p \end{cases}$$

with

$$rip_j = \begin{cases} 0 & \text{if } j = 0 \\ (nth(s_1, j-1) +_{\mathbb{Z}} nth(s_2, j-1) +_{\mathbb{Z}} rip_{j-1}) \ DIV \ p & \text{if } j > 0 \end{cases}$$

Here the operation DIV is the modular division that computes the integer quotient. Note that if the coefficients are $TPS_{h,r}$ we call the addition recursively. If they are ground elements, i.e. in $\mathbb{Z}_p$, we have to apply the usual modular addition that computes also the carry (rip_j).

MULTIPLICATION it follows the convolution rule. When the coefficients are again elements of $TPS_{h,r}$ the multiplication is applied recursively. When the coefficients are ground elements the multiplication in $\mathbb{Z}_p$ is applied and the carry has to be computed.

$\forall s_1, s_2 \in TPS_{h,r}$-NP

$$s_1 *_{NP} s_2 = < b_{0,i}, h^i >_{i \in \mathcal{I}_r} +_{NP} \cdots +_{NP} < b_{r-1,i}, h^i >_{i \in \mathcal{I}_r}$$

where $\forall k \in \mathcal{I}_r$

$b_{k,i} = 0$ if $k > i$

$$b_{k,i} = \begin{cases} nth(s_2, k) *_{\$} nth(s_1, i-k) & \text{if } \$ = TPS_{h,r} \\ nth(s_2, k) *_{\mathbb{Z}_p} nth(s_1, i-k) +_{\mathbb{Z}_p} rip_{k,i-k} & \text{if } \$ = \mathbb{Z}_p \end{cases}$$

otherwise

with

$$rip_{l,m} = \begin{cases} 0 & \text{if } m = 0 \\ (nth(s_2, l) *_{\mathbb{Z}} nth(s_1, m-1) +_{\mathbb{Z}} rip_{l,m-1}) DIV p & \text{if } m > 0 \end{cases}$$

INVERSE (additive inverse):

$\forall s \in TPS_{h,r}$-NP

$$inv_{NP}(s) = \begin{cases} < inv_{\$}(nth(s,i)), h^i >_{i \in \mathcal{I}_r} & \text{if } \$ = TPS_{h,r} \\ < a_i, p^i >_{i \in \mathcal{I}_r} & \text{if } \$ = \mathbb{Z}_p \end{cases}$$

with

$$a_i = \begin{cases} -_{\mathbb{Z}_p} nth(s,i) & \text{if } i = 0 \\ -_{\mathbb{Z}_p} nth(s,i) -_{\mathbb{Z}_p} 1 & \text{if } i > 0 \end{cases}$$

Figure 5: Properties for $TPS_{h,r}$-NP

$TPS_{h,r}$-V= $\{< a_i, h^i >_{i \in \mathcal{I}_r} \quad h = J, \quad a_i \in TPS_{h,r}\}$

OPERATIONS:

ZERO	*from R-MODULE on* $TPS_{h,r}$-V *redefining* 0 *as* 0_V
UNIT	*from R-MODULE on* $TPS_{h,r}$-V *redefining* 1 *as* 1_V
ADDITION	*from R-MODULE on* $TPS_{h,r}$-V *redefining* $+$ *as* $+_V$
INVERSE	*from R-MODULE on* $TPS_{h,r}$-V *redefining inv as* inv_V
EXT-PROD	*from R-MODULE on* $TPS_{h,r}$-V *redefining ext-prod as* $ext\text{-}prod_V$
SCALAR	: $TPS_{h,r}$-V $\times$ $TPS_{h,r}$-V $\longrightarrow$ \$

PROPERTIES: all the operations defined in *R-MODULE* are inherited. The function *scalar* is only defined in the $TPS_{h,r}$-V structure. The *zero* and *unit* operations are defined as for $TPS_{h,r}$-NP

ADDITION $\forall s_1, s_2 \in TPS_{h,r}\text{-V}\ s_1 +_V s_2 = < a_i, J^i >_{i \in \mathcal{I}_r}$ where $a_i = nth(s_1, i) +_\$ nth(s_2, i)$

INVERSE $\forall s \in TPS_{h,r}\text{-V}\ inv_V(s) =< inv_\$(nth(s,i)), J^i >_{i \in \mathcal{I}_r}$

EXT-PROD $\forall i \in \mathcal{I}_r, \quad \forall s_1 \in TPS_{h,r}, \quad \forall s_2 \in TPS_{h,r}\text{-V}\ nth(s_1\ ext\text{-}prod_V\ s2,\ i) = s_1 *_\$ nth(s_2, i)$

SCALAR $\forall s_1, s_2 \in TPS_{h,r}\text{-V}\ s_1\ scalar\ s_2 = nth(s_1, 0) *_\$ nth(s_2, 0) +_\$ \ldots +_\$ nth(s_1, r-1) *_\$ nth(s_2, r-1)$

Figure 6: Operations and properties for $TPS_{h,r}$-V

B Classes definition

```
;;*************************************************************************
;;                          CLASSES DEFINITIONS
;;*************************************************************************

(defclass group ()
  ()
 (:documentation "representation of abstract structure Group"))
(defclass ring (group)
  ()
 (:documentation "representation of abstract structure Unit-Ring;
                  subclass of group"))
(defclass e-d (ring)
  ()
 (:documentation "representation of abstract structure Euclidean Domain;
                  subclass of ring"))
(defclass zeta (e-d)
  ((val :reader read-val :initarg :val :type integer
        :documentation "integer value of represented object"))
 (:documentation "representation of ground structure Integers Numbers;
                  subclass of e-d"))
(defclass zeta-p (zeta)
```

$TPS_{h,r}$-M= $\{< a_i, h^i >_{i \in \mathcal{I}_r} \quad h = I, \quad a_i =< b_j, h^j >_{j \in \mathcal{I}_r} \quad h = J\}$

OPERATIONS:

ZERO *from RING on* $TPS_{h,r}$-M *redefining* 0 *as* 0_M

UNIT *from RING on* $TPS_{h,r}$-M *redefining* 1 *as* 1_M

ADDITION *from RING on* $TPS_{h,r}$-M *redefining* $+$ *as* $+_M$

MULT *from RING on* $TPS_{h,r}$-M *redefining* $*$ *as* $*_M$

INV *from RING on* $TPS_{h,r}$-M *redefining inv as* inv_M

EXT-PROD *from* $R-MODULE$ *on* $TPS_{h,r}$-M *redefining ext-prod as* $ext\text{-}prod_M$

PROPERTIES: the operations defined in *Ring* and *ext-prod* defined in *R-Module* are inherited. $\forall i, j \in \mathcal{I}_r, \quad \forall s \in TPS_{h,r}$-M:

ZERO $nth(0_M, i) = 0_\$$

UNIT $nth(1_M, i) = i\text{-}base(i)$

where

$$nth(i\text{-}base(i), j) = \begin{cases} 1_\$ & \text{if } i = j \\ 0_\$ & \text{if } i \neq j \end{cases}$$

ADDITION $\forall s_1, s_2 \in TPS_{h,r}$-M

$$s_1 +_M s_2 = < a_i, I^i >_{i \in \mathcal{I}_r}$$

where:

$$a_i = nth(s_1, i) +_\$ nth(s_2, i)$$

MULT $\forall s_1, s_2 \in TPS_{h,r}$-M $s_1 *_M s_2 = s_1$ *calc* $transpose(s_2)$

where

$\forall s \in TPS_{h,r}$-M, since $s =< a_i, I^i >_{i \in \mathcal{I}_r}$ with $a_i =< b_i^j, J^j >_{j \in \mathcal{I}_r}$, $b_i^j \in TPS_{h,r}$, then $transpose\ (s) =< c_i, I^i >_{i \in \mathcal{I}_r}$ with $c_i =< b_j^i, J^j >_{j \in \mathcal{I}_r}$

$\forall s_1, s_2 \in TPS_{h,r}$-M, $\forall i \in \mathcal{I}_r \quad nth(s_1\ calc\ s_2, i) =< b_i^j, J^i >_{j \in \mathcal{I}_r}$ where $b_i^j = nth(s_1, j)\ scalar\ nth(s_2, i)$

INVERSE $\forall s \in TPS_{h,r}$-M

$$inv_M(s) =< inv_\$(nth(s, i)), h^i >_{i \in \mathcal{I}_r}$$

EXT-PROD $\forall i \in \mathcal{I}_r, \quad \forall s_1 \in TPS_{h,r}, \quad \forall s_2 \in TPS_{h,r}$-M

$$nth(s_1\ ext\text{-}prod_M\ s2,\ i) = s_1\ *_\$\ nth(s_2, i)$$

Figure 7: Operations and properties for $TPS_{h,r}$-M

```
  ((base :reader read-base :initarg :base :type integer
         :documentation "base (positive integer)")
 (:documentation "representation of ground structure Integers mod p;
                  subclass of zeta"))
(defclass r-module (group)
  ()
 (:documentation "representation of abstract structure R-module;
                  subclass of group"))
(defclass tps ()
  ((mant :reader read-mant  :initarg :mant :type list
         :documentation "list of the power series coefficients ordered w.r.t.
                         increasing powers")
   (exp  :reader read-exp   :initarg :exp  :type integer
         :documentation "numeric value of the possible exponent of the base")
   (base :reader read-base  :initarg :base
         :documentation "base (symbol or number)")
   (trunc :reader read-trunc :initarg :trunc
          :documentation "truncation order")
   (deep :reader read-deep  :initarg :deep
         :documentation "counter of mantissa's length for recursive algorithms "))
 (:documentation "representation of parametric structure
                  TPS"))
(defclass tps-v  (r-module tps)
  ()
 (:documentation "representation of parametric structure
                  TPS-V"))
(defclass tps-np (ring tps)
  ()
 (:documentation "representation of parametric structure
                  TPS-NP"))
(defclass tps-m  (ring r-module tps)
  ()
 (:documentation "representation of parametric structure
                  TPS-M"))
(defmethod gcd-e-d ((obj1 e-d) (obj2 e-d))
  (cond
       ((equal obj2 (zero obj2)) (unit obj1))
       ((< (deg obj1) (deg obj2)) (gcd-e-d obj2 obj1))
       (t (gcd-e-d* obj1 obj2))))
(defmethod gcd-e-d* ((obj1 e-d) (obj2 e-d))
  (cond
       ((equal obj2 (zero obj2)) obj1)
       (t (gcd-e-d* obj2 (mod-e-d obj1 obj2)))))
```