A Small OpenMath Type System

James H. Davenport
Dept. Mathematical Sciences
University of Bath
Bath BA2 7AY
England
`jhd@maths.bath.ac.uk`*

Abstract

This paper describes the "light-weight" Small Type System of OpenMath. It is based on various discussion with the OpenMath Consortium, and notably with the NAG team.

Section 7 lists the open issues for debate.

1 Why a "Small Type System"?

Nothing in this document should be inferred as meaning that a Small Type System is "better" than ECC[1] theoretically: indeed clearly it is not. One should understand "Small" in the sense of "light-weight". There are two main uses of a potential Small Type System for OpenMath signatures.

- Tools which read "new" CDs automatically, e.g. a search engine which suddenly encountered as a symbol from a CD that it did not recognise. In the context of a "Small" type system, there is little more that can be offered to such systems than arity checking.
- Human beings reading the whole of a CD's fields, to determine how they ought to implement the CD, either as OpenMath-reading software, or, more interestingly, as OpenMath-writing software. These humans need to interpret the symbols in a way that (short of ECC or re-building an Axiom-like category system) cannot be totally formalised. However they "ought to be" imbued with that nebulous quality of "common sense".

To meet these goals, we define a system that encodes arity (and a little more) in a totally formal way, but leaves clues in names etc. that should help the human reader. If this were to be formalised somewhat more, it would meet the design goal [3] of being "typed annotations to set-theoretic specifications" (i.e. CDs).

2 Principles

1. A signature is an OpenMath object, and should, as such, possess an OpenMath XML encoding.
2. Every `OMS` must appear in a CD. For the sake of specificity, we chose a separate CD, `sts`, for those symbols specific to the Small Type System. This could clearly be changed, and clearly needs to be extended to add STS-specific but domain-specific symbols — see the experimental `polysts` CD. MathML version 2 has included[2] the names of certain sets (**N**, **Z**, **Q**, **R**, **C**, **H**), and OpenMath has a corresponding `setname1` CD, as well as an experimental `setname2` CD containing other symbols such as $\mathbf{Z}_m$. These symbols can also be used in signatures.
3. It is a truism of type theory that a constant can be replaced by a nullary function. Early precursors of Axiom did this, but this had to be changed, since the pragmatics of, say, `zero` (thought of as `zero()`) and `random()` are very different. Even equality cannot be the same. Hence we distinguish between the two. On the same lines, forcing all functions to have one argument by currying is tempting, but a pragmatic disaster[3]. This is not to say that a more adventurous type system could not do it, but this defeats both goals in the introduction at a "Small" level.
4. Clearly n-ary functions must be allowed. However, there are two types of these: those that simply take n-arguments, and those that are essentially binary associative functions normally written n-ary. As examples of these

   $$\texttt{list(1,list(2,3))} \neq \texttt{list(1,2,3)}$$

   even though `list` is an n-ary function, but

   $$\texttt{plus(1,plus(2,3))} = \texttt{plus(1,2,3)}$$

   since `plus` is also associative. It would be convenient (especially for search engines) to distinguish the two. Further justification for this view is given in Appendix A.
5. Some names (to be decided, and probably evolving with CD's) should refer to specific OpenMath known types, and some cannot be prescribed. For example, it may[4] be convenient to know that the second argument of `elt` (element of a list) must be a positive integer, and hence we could say this by writing

   ```
   <OMS name="PositiveInteger" cd="???"/>
   ```

*Supported by the OpenMath Esprit Project 24.696.

[1] A good reference for ECC is [4]. ECC is based on the Calculus of Constructions: see [1].

[2] Or has it? The latest draft seems to imply that **Q** would be `<ci type=''set''>Q</ci>`, so the list is potentially unbounded.

[3] But this is not to say that currying is always harmful: see appendix B.

[4] Or it may not, if we are writing about list algebras. This distinction is actually hard.

(though

```
<OMS name="N" cd="setname1"/>
```

might be better), whereas in other cases we may wish to convey to the human reader that the types have some generic property that the Small Type System does not formally encode, e.g. a gcd operation might wish to name its arguments and results without a formal definition, and therefore use

```
<OMV name="GCDDomain"/>
```

In particular, we use

```
<OMS name="Object" cd="sts"/>
```

to denote any OpenMath object (as far as the Small Type System is concerned). The symbol

```
<OMS name="NumericalValue" cd="sts"/>
```

is used to denote any numerical value, or symbol/expression that should represent such a value.

6. Function arguments (or results) are first-class objects. This is certainly true of OpenMath, so should be true of its type system: see the discussion of currying in appendix B.

7. Parameterised types are probably too complex for the Small Type System. Use ECC instead!

3 Encoding

This attempts to write a quasi-BNF grammar for the OpenMath encoding of a signature `SIG`. I use `OMS!` as shorthand for an OpenMath symbol encoding, (possibly attributed) and similarly `OMV!`. Quotation marks ‘ ’ denote literal OpenMath, except that I quote one possible order for `name=` and `cd=`, but intend both. Similarly, I do not go into the blank space and comment rules.

1. A rule to represent principles 5 and 6.

```
OMSV = OMS! | OMV! | APPL
```

2. Rules to represent principle 4.

```
NARYS = ‘<OMS name="nary" cd="sts"/>’ |
        ‘<OMS name="nassoc" cd="sts"/>’

NARY = ‘<OMA>’
          NARYS
          OMSV
        ‘</OMA>’
OMSVN = OMSV | NARY
```

3. Rule for application[5].

```
APPL = ‘<OMA>’
         ‘<OMS name="mapsto" cd="sts"/>’
         OMSVN*
         OMSV
       ‘</OMA>’
```

The last `OMSV` is to be thought of as the target, the rest as source domains. There also needs to be a side-rule, which I cannot write in BNF, that at most one of the `OMSVN*` can be an `NARY`, the rest must be `OMSV`.

4. Finally,

```
SIG = OMS! | OMV! | APPL
```

(in fact `OMSV` would do, but this confuses the distinction between constants (the first two) and functions).

4 Examples

Here we give some examples of this encoding.

zero `<OMV name="AbelianMonoid">`

random
```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS name="Object" cd="sts"/>
</OMA>
```

minus
```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMV name="AbelianGroup"/>
  <OMV name="AbelianGroup"/>
  <OMV name="AbelianGroup"/>
</OMA>
```

There is an important point of interpretation here. `OMV`s represent the same object in a local context[6], so this means that `minus` takes two objects from the *same* AbelianGroup, and returns an element of the *same* AbelianGroup.

list
```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMA>
    <OMS name="nary" cd="sts"/>
    <OMS name="Object" cd="sts"/>
  </OMA>
  <OMS name="Object" cd="sts"/>
</OMA>
```

This allows for heterogenous lists, but I doubt that the Small Type System is up to enforcing anything else.

plus
```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMA>
    <OMS name="nassoc" cd="sts"/>
    <OMV name="AbelianSemiGroup"/>
  </OMA>
  <OMV name="AbelianSemiGroup"/>
</OMA>
```

[5]As usual in OpenMath, this is abstract application, and does not necessarilyimply computation.

[6]Currently meaning a single signature, but this does not pre-empt any change from the DOM work here.

determinant
```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMV name="SquareMatrix"/>
  <OMV name="CommutativeRing"/>
</OMA>
```

Note that principle 7 implies that we do not attempt to parameterise the input argument `SquareMatrix`, so that the human reader has to infer that we mean one over the following `CommutativeRing`. However, we can convey hints by writing `SquareMatrix` rather than `Matrix`, and not just writing `Ring`.

5 The Treatment of OMBIND

We now have to integrate `OMBIND` into this construct. The clue is provided by the fact that a signature with `mapsto` means, precisely, that that symbol can appear as the head of a `OMA`. By analogy, we need a symbol, say `binder`, to say that a symbol can appear as the head of a `OMBIND`. We therefore augment the grammar given above by

```
BIND = ‘<OMA>’
         ‘<OMS name="binder" cd="sts"/>’
         OMV!
         OMSV
       ‘</OMA>’
```

and changing the definition of `SIG` to be

```
SIG = OMS!| OMV! | APPL | BIND
```

(equivalent to `OMSV | BIND`).

6 The structure construct

Some computer algebra systems, notably Axiom, GAP and MAGMA, can actually refer to algebraic structures, rather than just elements of them: for example Axiom can refer to the structure

```
UnivariatePolynomial(x,Integer)
```

as well as the object $x^2 + 2$ that lies in it. The GAP predicate `IsAbelian` takes a group as an argument, and returns a Boolean. It *could not* simply take an element, for that element might lie in an Abelian subgroup of a non-Abelian group. More seriously, the results of several mathematical operations depend on the underlying domain. For example, in $\mathbf{Q}[x]$ $x^2 + 1$ is irreducible, but in $\mathbf{Q}[i][x]$ it factors as $(x + i)(x - i)$. A Gröbner base is only defined with respect to a particular ordering, i.e. not just in $\mathbf{Q}[x, y]$ but in $\mathbf{Q}_{\mathrm{revgradlex}}[x, y]|_{\{x>y\}}$.

In order to accommodate this requirement, we introduce a new element into the STS vocabulary:

```
<OMS name="structure" cd="sts"/>
```

Applying this (via `OMA`) means that one wants the structure, rather than an element of the structure[7].

With this operator, a version of “random” that took a domain and returned an element of that domain would have the signature

[7]Dr. Solomon (St. Andrews) points out that it might be more logical to make this the default behaviour, and to require
```
<OMA>
<OMS name="element" cd="sts"/>
<OMV name="AbelianSemiGroup"/>
</OMA>
```
to get an element. This is true, but since the majority of uses are of elements, it seems that the common case should be the short one.

```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMA>
    <OMS name="structure" cd="sts"/>
    <OMV name="Set"/>
  </OMA>
  <OMV name="Set"/>
</OMA>
```

Note that the convention that an OMV has the same meaning in a local context means that this function takes a set, and returns an element *of that set*. A more complex example is given by `convert` in the experimental `poly` CD.

```
<Signature name="convert" >
<OMOBJ>
 <OMA>
   <OMS name="mapsto" cd="sts" />
    <OMV name="Polynomial" />
    <OMA>
      <OMS name="structure" cd="sts" />
       <OMV name="Polynomial1" />
    </OMA>
    <OMV name="Polynomial1" />
 </OMA>
</OMOBJ>
</Signature>
```

This takes an element of the polynomial ring R, and (the specification of) a different polynomial ring S, and returns an element of S.

7 STS issues and developments

In this section we list various issues that have arisen, and outline possible developments of the STS.

1. Multiple signatures. At the moment, an operator like sin has only one signature:

   ```
   <OMA>
     <OMS name="mapsto" cd="sts"/>
     <OMS name="NumericalValue" cd="sts"/>
     <OMS name="NumericalValue" cd="sts"/>
   </OMA>
   ```

   While this does describe the behaviour of sin, it is not as complete as possible. One might wish to add more information by giving an *additional* signature such as the following, to convey the fact that sin always maps reals to reals.

   ```
   <OMA>
     <OMS name="mapsto" cd="sts"/>
     <OMS name="R" cd="setname1"/>
     <OMS name="R" cd="setname1"/>
   </OMA>
   ```

   One could even say the following, to be more precise.

   ```
   <OMA>
     <OMS name="mapsto" cd="sts"/>
     <OMS name="R" cd="setname1"/>
     <OMA>
       <OMS name="interval_cc" cd="interval1"/>
       <OMA>
   ```

```
      <OMS name="unary_minus" cd="arith1"/>
      <OMS name="one" cd="alg1"/>
    </OMA>
    <OMS name="one" cd="alg1"/>
  </OMA>
</OMA>
```

The area where the need for this is most evident[8], though special, is in power. This symbol has, unfortunately, two uses: powering as repeated multiplication (of the multiplicative inverse if necessary/possible) and powering of real and complex numbers, defined (in the arith1 CD) as $x \in C \Rightarrow x^a = \exp(a \ln x)$. However, with the "one signature" rule, we are torn between

```
<OMOBJ>
 <OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS name="Object" cd="sts"/>
  <OMS name="Z" cd="setname1"/>
  <OMS name="Object" cd="sts"/>
 </OMA>
</OMOBJ>
```

and

```
<OMOBJ>
 <OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
 </OMA>
</OMOBJ>
```

We may have to settle for

```
<OMOBJ>
 <OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS name="Object" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
  <OMS name="Object" cd="sts"/>
 </OMA>
</OMOBJ>
```

which would be unfortunate.

Note that we are *not* proposing conflicting signatures or different arities. In fact we will require the signatures to be consistent, as defined in Appendix C. Note that this will require, in addition to the two signatures for power proposed above, the signature

```
<OMOBJ>
 <OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
  <OMS name="Z" cd="sts"/>
  <OMS name="NumericalValue" cd="sts"/>
 </OMA>
</OMOBJ>
```

[8] Bill Naylor has pointed out the instance of Laplacian. This has two possible signatures: (vector→scalar)→(vector→scalar) and (vector→vector)→(vector→vector). The only choice available to us: (vector→object)→(vector→object) is unsatisfactory, since it would also include
(vector→scalar)→(vector→vector).

to make the system regular.

2. Extending STS — a "not-so-small" type system?

   It is obvious from Appendix D that some names are very frequent. The most common is Boolean, and I therefore **propose** that this be moved into setname2 CD. Whether we can do this in time for the next release of CDs depends on whether setname2 itself is in that release. Since MathML2 appears to have moved away from a fixed collection of set names to allowing <ci type="set">, there seems to be no reason not to add this to the core small typing system.

   The next one is Set. Two occurrences of this in the group1 CD clearly mean "set of (members of) the same group", and would therefore cry out for parameterisation, as do the two from the permgrp CD. Similarly VariableList and ObjectList, the five occurrences of PolynomialRingList and the occurrence of SLInstructionList could do with being replaced by a parameterised structure, as could some of Matrix constructs, and, as discussed above, the SquareMatrix use.

   Is this enough demand to build in a mechanism for parameterisation?

3. OpenMath Errors.

   There is one element of the OpenMath language that does not have a signature mechanism described so far. This is the OpenMath "Error" construct. These errors are defined to have return type the special symbol Error in the STS content dictionary. Thus the signature for an error such as unhandled_symbol would be as follows.

```
<OMOBJ>
  <OMA>
    <OMS cd="sts" name="mapsto"/>
    <OMV name="OMSymbol"/>
    <OMS name="Error" cd="sts"/>
  </OMA>
</OMOBJ>
```

References

[1] Coquand,Th. & Huet,G., The Calculus of Constructions. *Information and Computation*, **76** (1988).

[2] Goguen,J.A. & Meseguer,J., Order-sorted Algebra I: Equational deduction for multiple inheritance, polymorphism and partial operations. *Theor. Comp. Sci.* **105** (1992) pp. 217–293.

[3] Lamport,L. & Paulson,L.C., Should Your Specification Language Be Typed? *ACM TOPLAS* **21** (1999) pp. 502–526.

[4] Luo,Z. *An Extended Calculus of Constructions*. Ph.D. thesis, University of Edinburgh, 1990.

A Justification for nassoc

It would be possible not to describe nassoc functions differently from nary ones. This would be possible if OpenMath were not a semantic system. In particular, the Formal Mathematical Properties (FMPs) are hard to write. The commutativity of a binary addition can be addressed, as in the arith CD, as follows.

```
<FMP>
<OMOBJ>
  <OMBIND>
    <OMS cd="quant" name="forall"/>
    <OMBVAR>
      <OMV name="a"/>
      <OMV name="b"/>
    </OMBVAR>
    <OMA>
      <OMS cd="relation" name="eq"/>
      <OMA>
        <OMS cd="arith" name="plus"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
      <OMA>
        <OMS cd="arith" name="plus"/>
        <OMV name="b"/>
        <OMV name="a"/>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
</FMP>
```

However, this does not even state that a ternary plus could be commutative in the first two arguments, and certainly does not state that, say, $a+b+c=a+c+b$.

A similar problem is posed by associativity: one can write a property equivalent to $a+(b+c)=(a+b)+c$, but again this does not cover n-ary functions. It would be possible to cope with ternary usages, e.g. by writing an FMP equivalent to $(a+b)+c=a+b+c=a+(b+c)$, but even this would not cope with quaternary usages, as in $a+b+c+d$.

There is a fundamental dichotomy between allowing n-ary operations for notational and practical convenience, on the one hand, and a formal semantics system, which generally only allows operations of fixed arity. Treating such operations as "special" is the easiest way of solving this.

B Currying

Currying is not always harmful, despite the views in section 2. Consider the Bessel function $J_\nu(z)$ for simplicity: the same points are true for other functions. This could be defined as a function $(\mathbf{C}\times\mathbf{C})\rightarrow\mathbf{C}$, i.e.

```
<OMA>
  <OMS name="mapsto" cd="sts"/>OMV
  <OMS cd="sts" name="NumericalValue"/>
  <OMS cd="sts" name="NumericalValue"/>
  <OMS cd="sts" name="NumericalValue"/>
</OMA>
```

but it makes more sense to define it as $\mathbf{C}\rightarrow(\mathbf{C}\rightarrow\mathbf{C})$, i.e.

```
<OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMS cd="sts" name="NumericalValue"/>
  <OMA>
    <OMS name="mapsto" cd="sts"/>
    <OMS cd="sts" name="NumericalValue"/>
    <OMS cd="sts" name="NumericalValue"/>
  </OMA>
</OMA>
```

so that we can say that $J(\nu)$ satisfies Bessel's equation, rather than having to say that $\lambda x.J(\nu,x)$ satisfies Bessel's equation.

C Consistency of Signatures

While it is "intuitive" what one means by a set of consistent signatures, writing it down is more complicated, and we need the support of order-sorted algebra [2]. From the point of view of STS, a "sort symbol" is just an STS `OMSV`, i.e. an STS type. Some of these sort symbols represent specific mathematical objects, e.g. `<OMS name="Q" cd="setname1"/>` represents $\mathbf{Q}$, whereas others represent families of mathematical objects, such as `<OMV name="AbelianSemiGroup"/>`.

Definition 1 *Let S be a set of sort symbols such that there is a partial order $\prec$ defined on S, and a "top" element u of S such that $s\preceq u$ for all s in S.*

u is in fact `<OMS name="Object" cd="sts"/>`, and the relation $\prec$ can be read as "is contained in", interpreted as follows (for $s\prec t$):

- s, t specific: $s\subset t$;
- s specific, t generic: $s\in t$, as in

  `<OMS name="Q" cd="setname1"/>` $\in$ `<OMV name="Field"/>`;
- s, t generic: $s\subset t$, as in

  `<OMV name="AbelianGroup/>` $\prec$ `<OMV name="AbelianSemiGroup/>`

  (note that this generally means that s will have more operations/axioms than t).

Actually documenting the relation $\prec$ is hard, and can be regarded as much of the job of the Axiom category system.

Definition 2 *For the purpose of this appendix, an S-arity of rank n is an ordered n-tuple of elements of S. We extend $\preceq$ from S to the S-arities of rank n by defining $(s_1,\ldots,s_n)\preceq(t_1,\ldots,t_n)$ if, and only if $\forall i \quad s_i\preceq t_i$.*

Definition 3 *An S-operator set or S-sorted signature is a set Σ of sets $\Sigma_{n,q,s}$ (indexed by a natural number n, an arity q of rank n and an element s of S) such that all $\Sigma_{n,q,s}$ are subsets of the set of OpenMath symbols (symbols in different CDs are considered different).*

The OpenMath condition that all symbols have a fixed arity (in the usual sense) means that, if $n\neq n'$, then $\Sigma_{n,q,s}\cap\Sigma_{n',q',s'}=\emptyset$.

Definition 4 *A Σ-algebra is an ordered triple $\langle A,\{A_s:s\in S\},\alpha\rangle$ where A is a set, known as the universe, $\{A_s:s\in S\}$ is an S-indexed family of subsets of A, known as the carriers of the algebra, and α is a set of sets of functions $\alpha_{n,q,s}=\{\alpha_{n,q,s,\sigma}:A^q\rightarrow A_s\forall\sigma\in\Sigma_{n,q,s}\}$, such that:*

1. *$A_u=A$;*
2. *If $s\preceq s'$ in S, then $A_s\subseteq A_{s'}$;*
3. *If $\sigma\in\Sigma_{n,q,s}\cap\Sigma_{n,q',s'}$ with $s'\preceq s$ and $q'\preceq q$, then $\alpha_{n,q,s,\sigma}$ agrees with $\alpha_{n,q',s',\sigma}$ on $A^{q'}$.*

Here $A^q = (A_{q_1}, \ldots, A_{q_n})$ if $q = (q_1, \ldots, q_n)$.

This last condition removes any syntactic ambiguity about the application of the operator corresponding to σ to arguments typed according to q': should we use the function that takes arguments from $A^{q'}$ or from A^q? The answer given by condition (3) is that it does not matter. Naïvely, this should seem to suffice. However, it turns out that we need a further requirement on the signatures.

Definition 5 *The order-sorted signature Σ is regular if, whenever q^* is an arity and $\sigma \in \Sigma_{n,q,s}$ with $q^* \preceq q$, there is a least pair q', s' such that $\sigma \in \Sigma_{n,q',s'}$ and $q^* \preceq q'$ and $s' \preceq s$.*

As an example of the need for this condition, consider a system of sorts Z, G, R, C with $Z \prec G \prec C$ and $Z \prec R \prec C$, and a binary operation $\cdot$ defined on G and R (so that $\Sigma_{2,(G,G),G} = \Sigma_{2,(R,R),R} = \{\cdot\}$). As a specific example of this, think of the algebra whose C sort is the complex numbers $\mathbf{C}$, whose R sort is the real numbers $\mathbf{R}$, whose G sort is the complex integers $\mathbf{Z} + i\mathbf{Z}$ and whose Z sort is the integers $\mathbf{Z}$. These carriers satisfy the appropriate inclusions. We could define the operators so that $x \cdot y$ was xy for $x, y \in \mathbf{R}$, but $x \cdot y$ was $x + y$ for $x, y \in \mathbf{Z} + i\mathbf{Z}$. We would then be confused about $2 \cdot 3$: is it 6 because $2, 3 \in \mathbf{R}$, or 5 because $2, 3 \in \mathbf{Z} + i\mathbf{Z}$? This confusion arises because the signature is not regular, so there is no least signature for $2 \cdot 3$. To make the signature regular, we have to add $\Sigma_{2,(Z,Z),Z} = \{\cdot\}$, and then the compatibility constraint (3) forces this to have the same meaning on the Z sort as both the other definitions of $\cdot$, which shows that the example cannot be completed to an algebra with a regular signature.

Definition 6 *A set of signatures for an OpenMath symbol is* consistent *if it is regular and every collection of mathematical sets on which the mathematical operation defined by the symbol is an algebra in the sense of definition 3, in particular satisfies clause (3).*

D The `OMV` elements used so far

This excludes 31 specialised names from `meta.sts`.

AbelianGroup	8
AbelianMonoid	5
AbelianSemiGroup	4
BasedInteger	1
BigFloat	3
Boolean	34
CardinalNumber	1
Category	1
ComplexField	1
Domain	1
EuclideanRing	2
FactoredPolynomial	5
Field	15
FunctionType	1
GroebnerBasis	3
Group	24
GroupGenerators	1
IEEEFloat	1
Infinity	1
IntegerInterval	1
Integer-or-Infinity	2
IntegerRange	2
IntegralDomain	2
List	5
MathMLNumericType	6
MathMLType	5
Matrix	9
Monoid	1
MonomialD	2
MonomialR	2
NonZeroInteger	1
ObjectList	1
OrderedSet	12
Permutation	1
PermutationGroup	4
PGroup	1
Polynomial1	1
PolynomialPower	3
PolynomialRing	1
PolynomialRingList	5
PolynomialVariable	6
PolyRrep	2
PositiveInteger	5
PositiveReal	1
RangeOfIntegration	1
RealInterval	5
Ring	7
SDMPObject	2
SemiGroup	2
Set	30
SLInstructionList	1
SLPInstruction	8
SLPPolynomial	12
SLProgram	2
SquareMatrix	1
Symbol	4
TendsTo	5
Tuple	2
Type	3
VariableList	1