# Symbolic Computation Software Composability

Sebastian Freundt[1], Peter Horn[2], Alexander Konovalov[3], Steve Linton[3] and Dan Roozemond[4]

[1] Fakultät II - Institut für Mathematik, Technische Universität Berlin, Berlin, Germany
freundt@math.tu-berlin.de
[2] Fachbereich Mathematik, Universität Kassel, Kassel, Germany
hornp@mathematik.uni-kassel.de
[3] School of Computer Science, University of St Andrews, Scotland
{alexk,sal}@mcs.st-and.ac.uk
[4] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands
d.a.roozemond@tue.nl

**Abstract.** We present three examples of the composition of Computer Algebra Systems to illustrate the progress on a composability infrastructure as part of the SCIEnce (Symbolic Computation Infrastructure for Europe) project[1]. One of the major results of the project so far is an OpenMath based protocol called SCSCP (Symbolic Computation Software Composability Protocol). SCSCP enables the various software packages for example to exchange mathematical objects, request calculations, and store and retrieve remote objects, either locally or accross the internet. The three examples show the current state of the GAP, KANT, and MuPAD software packages, and give a demonstration of exposing Macaulay using a newly developed framework.

## 1 Introduction

The SCIEnce project (Symbolic Computation Infrastructure for Europe) [25] brings together the developers of four powerful symbolic computation software packages (GAP [8], KANT [14], Maple [17], and MuPAD [19]), a major symbolic computation research institute (RISC-Linz [21]), and research groups expert in essential underpinning technologies (CNRS Palaiseau (France) [4], TU Eindhoven (Netherlands) [6], IeAT (Romania) [13] and Heriot-Watt University (UK) [5]). The aim is to unite the European community of researchers in, and users of, symbolic computation.

In this paper we report on one of the activities the SCIEnce project consists of, namely *NA3: Software Composability*. This activity focuses on the development and implementation of standards in order for the various Computer Algebra Systems (CASes) to communicate. The main goal of this activity is to allow these systems to be efficiently composed to solve complex problems.

[1] The project 026133 "SCIEnce—Symbolic Computation Infrastructure for Europe" is supported by the EU FP6 Programme.

S. Autexier et al. (Eds.): AISC/Calculemus/MKM 2008, LNAI 5144, pp. 285–295, 2008.

This part of the project has some common concerns with the well known SAGE project [22], as both projects try to unite several mathematical software packages. There are, however, important differences. SAGE presents itself as an integrated system in which users interact with the SAGE frontend and the contributing CASes are used as backend servers. Our goal in the SCIEnce project is to create a framework that will allow services to be both provided and consumed by any CAS. An important technical difference is that we use an existing language for representing mathematical objects, namely OpenMath [20], whereas SAGE uses a custom internal representation. We expect the use of OpenMath to facilitate third party developers to expose their software using SCSCP – although the conversion to and from the XML-based OpenMath format is potentially more time-consuming, we obtain a stable system-independent representation.

We illustrate the progress made in this activity by means of three examples. First, we introduce the Computer Algebra Systems involved in Section 2, and the OpenMath standard in Section 3. We give an overview of the newly designed protocol for the composition of symbolic computation software, SCSCP, in Section 4. The first example is the factorization in KANT of polynomials created in MuPAD (Section 5). In the second example (Section 6) we show a Gröbner basis computation executed in Macaulay on polynomials created in GAP. The third example (Section 7) demonstrates cross-platform use of GAP using SCSCP. Comments on the current status and intended future research can be found in Section 8.

## 2 The Computer Algebra Systems Involved

In this section, we briefly describe the four computer algebra systems involved in the SCIEnce project.

GAP [8] is a free, open and extendable system for computational discrete algebra, with particular emphasis on Computational Group Theory. GAP provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the GAP language as well as large data libraries of algebraic objects. GAP is developed by international cooperation of many contributors, and coordinated by the four GAP centers: Aachen (Germany), Braunschweig (Germany), Fort Collins (USA), and St Andrews (UK).

KANT [14] is a computer algebra system for sophisticated computations in algebraic number fields that has been developed at Technische Universität Berlin. The KANT functions are accessible through a user-friendly shell named KASH (KAnt SHell) that is freely available.

Maple [17] is the general purpose computer algebra system developed in Waterloo, Canada. Its latest features include an intuitive smart document environment and embedded GUI components such as buttons and sliders.

MuPAD Pro [19] is a general purpose computer algebra system for exact symbolic and numeric computing with arbitrary precision. It provides a Pascal-like programming language allowing imperative, functional, and object-oriented

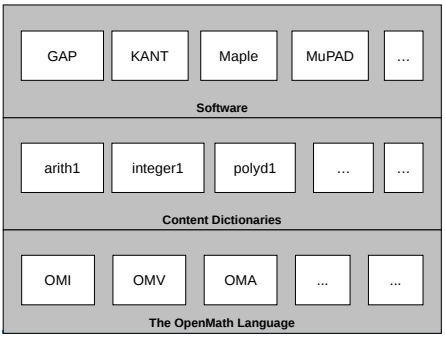


**Fig. 1.** OpenMath structure

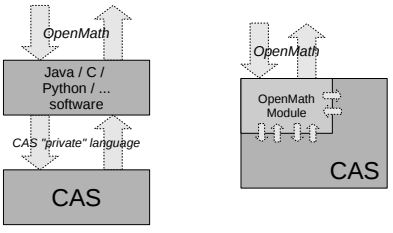


**Fig. 2.** CAS implementations

programming. MuPAD is developed by the SciFace company, based in Paderborn, Germany. The SCIEnce development with respect to MuPAD is performed at the University of Kassel, Germany [26].

## 3 OpenMath

The OpenMath standard is made for the representation of mathematics in such a way that mathematical objects can easily be exchanged between computer programs by way of rich semantics. A rough overview of this standard can be found in Figure 1. The 3 layers are explained as follows:

**Language.** The OpenMath language defines the grammar, i.e., notions such as Variables, Errors, Applications, Integers, etc.

**Content Dictionary.** A Content Dictionary (CD) is a document describing mathematical notions for some area of Mathematics. At the moment of writing about 180 content dictionaries are provided on the OpenMath homepage, both official and experimental. They cover not only general areas, such as basic arithmetic (for example the 'arith1' CD describes 'minus', 'plus', 'power', etc) or polynomials, but also more specific areas such as permutation groups, planar geometry, finite fields, and much more.

**Software.** This third layer consists of all software built using the basic language and content dictionaries, commonly referred to as "Phrasebooks". In Figure 2 the two most common approaches in our setting are described.

First the piece of translator software separate from the CAS: this software (commonly written by a third party) takes care of the translation from OpenMath into the CAS proprietary language, and back. Especially the translation back can be highly non-trivial, as the semantics of the CAS output cannot always be read off from the output itself.

The second option is a piece of translator software that is built into the CAS. The disadvantage is that one needs to have access to the source code, which is not always possible. On the other hand, the major advantage of

this approach is that generally the translation can be done using the internal representation of mathematical objects in the CAS.

The most common representation for OpenMath is the XML-representation. For example, $3 - \frac{4}{5}$ could be represented as follows:

```
<OMA><OMS cd="arith1" name="minus"/>
  <OMI>3</OMI>
  <OMA><OMS cd="nums1" name="rational"/>
    <OMI>4</OMI>
    <OMI>5</OMI>
  </OMA>
</OMA>
```

## 4 SCSCP

To simplify the communication between the various CASes, we have developed a protocol called "Symbolic Computation Software Composability Protocol", abbreviated SCSCP. The protocol is XML-based; in particular, the protocol messages are in the OpenMath language, and its TCP-sockets-based implementation uses XML processing instructions to delimit these messages and indicate major failures that may arise during the processing of a request. Communication takes place using port 26133, reserved for SCSCP by the Internet Assigned Numbers Authority (IANA).

We have developed two Content Dictionaries for SCSCP, called `scscp1` and `scscp2`. The protocol supports calling functions with mathematical objects as arguments, on either a local or remote system, and sending back successful results or failure reports. It also supports basic options such as limits on memory or CPU and information such as memory or CPU time used.

Moreover, SCSCP has support for remote objects. The client may indicate a preference for the reply to either contain a full mathematical object, or a reference to that same object. We envisage a scenario where a client can let almost all computations be performed remotely, possibly in another CAS, and where details of mathematical objects are not transmitted unless necessary.

An example of a call to a CAS and a response follows:

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="scscp1" name="call_ID"/>
      <OMSTR>a1d0c6e83f027327d8461063f4ac58a6 </OMSTR>
      <OMS cd="scscp1" name="option_max_memory"/>
      <OMI>100000</OMI>
      <OMS cd="scscp1" name="option_return_object"/>
      <OMSTR/>
    </OMATP>
    <OMA>
      <OMS cd="scscp1" name="procedure_call"/>
```

```
      <OMSTR>Evaluate</OMSTR>
      <OMA>
         <OMS cd="arith1" name="plus"/>
         <OMI>16603777328095411</OMI>
         <OMI>9529248804930722</OMI>
      </OMA>
    </OMA>
  </OMATTR>
</OMOBJ>
```

with response

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="scscp1" name="call_ID"/>
      <OMSTR>a1d0c6e83f027327d8461063f4ac58a6</OMSTR>
      <OMS cd="scscp1" name="info_runtime"/>
      <OMI>3</OMI>
      <OMS cd="scscp1" name="info_memory"/>
      <OMI>2876</OMI>
    </OMATP>
    <OMA>
      <OMS cd="scscp1" name="procedure_completed"/>
      <OMI>26133026133026133</OMI>
    </OMA>
  </OMATTR>
</OMOBJ>
```

More details and examples can be found in the specification [15] and the two Content Dictionaries [23], [24].

Basic SCSCP support, both as server and as client, is now available for development versions of GAP, KANT, and MuPAD. This means that a user of one of these software packages can invoke one of the other systems (or the same system on a different machine) without leaving the software packages he is working in himself. In particular, one uses GAP syntax to use SCSCP from within GAP, for example, even though one may be calling out to KANT.

We have also created a Java implementation of SCSCP, intended as a framework to enable third party developers to expose their software easily to e.g. users of one of the systems involved.

Furthermore, as SCSCP is a specialized protocol, it would have to be implemented in a each software package to allow access to the capabilities of the systems involved. In order to offer access by means of a more widely used protocol than SCSCP, we have developed a WebProxy that connects to an arbitrary number of SCSCP-compliant systems and offers a SOAP-interface as well as a simplistic html-interface to these systems.

One of the other parallel activities in the SCIEnce project is *JRA1: Symbolic Computing on the Grid*, which focuses on developing a suitable framework for

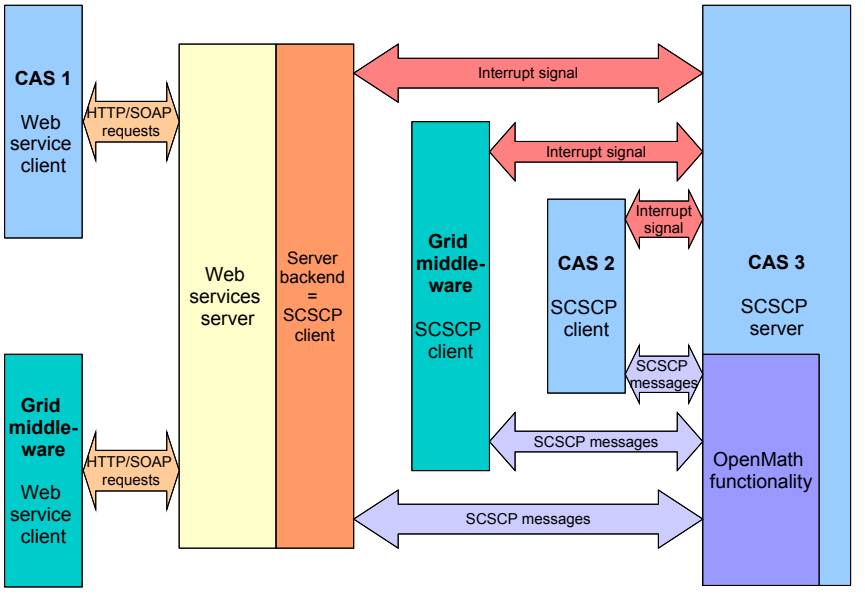


**Fig. 3.** SCSCP Overview

symbolic computing on computational grids. In this activity SCSCP is used for the communication between the server and the various systems.

## 5 Using KANT from MuPAD

As a first example, we consider factoring of polynomials defined similarly to Swinnerton-Dyer polynomials, but without the requirement that the primes are consecutive: for a set of distinct prime numbers $p_1, p_2, \ldots, p_n$, we define the polynomial $P_n(x)$ as

$$P_n(x) = \prod \left(x \pm \sqrt{p_1} \pm \sqrt{p_2} \ldots \pm \sqrt{p_n}\right),$$

where the product runs over all possible combinations of plus and minus signs, yielding $2^n$ factors in total. Hence the degree of such a polynomial is $2^n$.

This polynomial is easily seen to be irreducible over $\mathbb{Z}$. On the other hand, suppose $\mathbb{F}$ is a finite field; then $P_n$ splits into linear factors over a quadratic extension of $\mathbb{F}$, so it will only have linear and quadratic factors over $\mathbb{F}$ itself. In particular, $P_n$ $(n > 1)$ is reducible over every finite field.

These polynomials are worst-case inputs for the Berlekamp-Zassenhaus algorithm for the factoring of polynomials over $\mathbb{Z}$. See [9, Section 15.3] for more information.

```
>> package("OpenMath"):
>> swindyer:=proc(plist) <some details omitted> :
>> R := Dom::UnivariatePolynomial(x, Dom::Rational):
>> p1 := R(expand(swindyer([2,3,5,7,11]))):
>> p2 := R(expand(subs(swindyer([2,3,5,7,13,17])), x=3*x-2)):
>> p := p1 * p2:
>> degree(p), nterms(p)
96, 49
```

So at this point we have constructed a univariate polynomial $p$, the product of two of these Swinnerton-Dyer like polynomials, with an affine transformation applied to the argument of the second one. It has 49 terms and degree $96 = 2^5 + 2^6$.

```
>> st := time(): F1 := factor(p): time()-st
38431
```

So factoring it in MuPAD takes 38 seconds.

On the other hand, KANT has one of the fastest univariate polynomial factorizers available. If we convert the polynomial into OpenMath, transmit it to a machine running KANT almost 400 kilometers away, convert it to KANT syntax, factor it, convert it back into OpenMath, transmit it back to the original machine, and finally convert it back into MuPAD syntax:

```
>> kant := SCSCP("scscp.math.tu-berlin.de", 26133):
>> st:=rtime(): F2:=kant::compute(hold(factor)(p)):rtime()-st
1221
```

So factoring in KANT only takes 1.2 seconds. To verify that the two results have the same factors:

```
>> FS1 := {op(Factored::factors(F1))}:
>> FS2 := {op(map(F2, X -> R(subs(expr((X[1])), `#1`=x))))}:
>> bool(FS1=FS2)
TRUE
```

The two-line conversion of the object KANT returns is necessary because one needs to explicitly state that this object $FS2$ is to be in the same polynomial ring that the original polynomial $p$ was in.

The OpenMath objects are transmitted in uncompressed XML syntax, a few kilobytes for polynomials of this order of magnitude. Moreover, even though at this stage of the project no particular effort has been put into optimizing the conversions between CAS syntax and OpenMath, in our case these translations take only about 20 milliseconds each.

## 6 Using Macaulay from GAP

Using the framework mentioned earlier, we have created an SCSCP interface to Macaulay 2 [10]. We use this interface to perform a Gröbner basis computation on polynomials created in GAP. These polynomials can be used to create an automatic proof of the circle theorem of Apollonius.

```
gap> R := PolynomialRing(Rationals,
>           ["a","b","s","y","m1","m2","p1","p2"]);;
gap> a := R.1;; b := R.2;; s := R.3;; y := R.4;;
gap> m1 := R.5;; m2 := R.6;; p1 := R.7;; p2 := R.8;;
gap> pols := [
>    (m1-a)^2 + m2^2 - s^2,
>    m1^2 + (m2-b)^2 - s^2,
>    (m1-a)^2 + (m2-b)^2 - s^2,
>    -2*a*p1+2*b*p2,
>    -2*a*p2-2*b*p1+2*a*2*b,
>    a*b*y-1
> ];;
```

We have created a polynomial ring in 8 variables over $\mathbb{Q}$, and a list of 6 polynomials. We can try to compute a Gröbner Basis (with respect to the graded reverse lexicographic ordering) in GAP:

```
gap> B := GroebnerBasis(pols, MonomialGrevlexOrdering());
```

but the computation does not end within 30 minutes. We can also use the Macaulay 2 interface we created:

```
gap> I := Ideal(R, pols);;
gap> B2 := EvaluateBySCSCP("Macaulay2-Groebner", [I],
>            "scscp.win.tue.nl", 26133);;
#I  Creating a socket ...
#I  Connecting to a remote socket via TCP/IP ...
#I  Got connection initiation message
#I  Request sent ...
#I  Waiting for reply ...
gap> B2.object;
[ b-2*m2, a-2*m1, -4*m2*p2+p1^2+p2^2, m1*p1-m2*p2,
  4*m1*m2-m1*p2-m2*p1, s^2-m1^2-m2^2, y*m1*p2+y*m2*p1-1,
  4*y*m2^2*p2-p1, 4*y*m1^2*p2-4*m1+p1 ]
```

This calculation took only a few seconds. Moreover, the majority of the time is actually spent in the conversion of OpenMath to Macaulay syntax and back, so an even larger improvement could be obtained by optimization of this translation.

## 7 Further GAP Examples

The previous section demonstrated GAP as an SCSCP client. Here we would like to give one more example of the use of GAP as an SCSCP server with the development version of the GAP package SCSCP [16]. We will outline simple steps needed for the design and provision of the SCSCP service within the framework provided by SCSCP.

The GAP Small Groups Library [2] contains all groups of orders up to 2000, except groups of order 1024. The GAP command `SmallGroup(n,i)` returns the `i`-th group of order `n`. Moreover, for any group $G$ of order $1 \leq |G| \leq 2000$ where $|G| \notin \{512, 1024\}$, GAP can determine its *library number*: the pair `[n,i]` such

that $G$ is isomorphic to `SmallGroup(n,i)`. This is in particular the most efficient way to check whether two groups of "small" order are isomorphic or not.

For groups of order 512 the Small Groups Library contains all 10494213 non-isomorphic groups of this order and allows the user to retrieve any group by its library number, but it does not provide an identification facility. However, the GAP package ANUPQ [7] provides a function `IdStandardPresented512Group` that performs the latter task. Because the ANUPQ package only works in a UNIX environment it is useful to design an SCSCP service for identification of groups of order 512 that can be called from within GAP sessions running on other platforms (note that the client version of the SCSCP package for GAP does work under Windows).

First we need to decide how the client should transmit a group to the server: How should the group be encoded in OpenMath? Should it be converted into a permutation representation, which can be encoded using existing content dictionaries? Or should we develop new content dictionaries for other kinds of groups? Luckily, the SCSCP protocol provides enough freedom for the user to select his own data representation, and since we are interfacing between two copies of the GAP system, we are free to use a GAP-specific data format, namely the *pcgs code*: an integer, describing the *polycyclic generating sequence* (*pcgs*) of the group, to pass the data to the server. See the GAP manual and [1] for more details about the pcgs code.

First we create a function that takes the pcgs code of a group of order 512 and returns the number of this group in the GAP Small Groups library:

```
gap> IdGroup512ByCode := function( code )
>    local G, F, H;
>    G := PcGroupCode( code, 512 );
>    F := PqStandardPresentation( G );
>    H := PcGroupFpGroup( F );
>    return IdStandardPresented512Group( H );
>    end;;
```

After such a function was created on the server, we need to make it "visible" as an SCSCP procedure under the name `IdGroup512`:

```
gap> InstallSCSCPprocedure("IdGroup512", IdGroup512ByCode );
InstallSCSCPprocedure : procedure IdGroup512 installed.
```

For the convenience of the user, we provide the client's counterpart for this service, carrying all technical details (server, port) and also checking that the group is of order 512:

```
gap> IdGroup512:=function( G )
>    local code, result;
>    if Size( G ) <> 512 then
>      Error( "|G|<>512\n" );
>    fi;
>    code := CodePcGroup( G );
>    result := EvaluateBySCSCP("IdGroup512ByCode", [ code ],
>                              "scscp.st-and.ac.uk", 26133);
```

```
>    return result.object;
>    end;;
```

Now when the client calls the function `IdGroup512`, it looks almost like the standard GAP function `IdGroup` (the user may switch off intermediate information messages):

```
gap> IdGroup512( DihedralGroup( 512 ) );
#I  Creating a socket ...
#I  Connecting to a remote socket via TCP/IP ...
#I  Got connection initiation message
#I  Request sent ...
#I  Waiting for reply ...
[ 512, 2042 ]
```

The GAP package SCSCP also offers functionality for parallel computations that may be used for example on a multi-core machine. It provides convenient functions for the user to parallelize computation by sending out two or more requests, and then collect either all results or (in the case when several methods are used for the same computation and it is not a priori clear which one will be fastest) get the first available result. More higher-level examples are contained in the package's manual which is available upon request and will be a part of the official release of the package.

## 8 Status and Future Research

At the moment of writing (March 2008) we implemented support for communications using SCSCP, both as servers and as clients, for the development versions of GAP, KANT, and MuPAD. Also, progress is being made with respect to OpenMath and SCSCP support in Maple. A part of ongoing work is to extend the range of mathematical objects that are understood by our systems, i.e., to enable translations from and to OpenMath for a wider set of OpenMath content dictionaries. We expect OpenMath support to be improved over the year 2008, greatly increasing the possibilities for exchanging mathematics between the various computer algebra systems.

As demonstrated, currently we expose the systems involved as SCSCP services. Future research includes using SCSCP to expose these systems as proper Web services, i.e. extending the WebProxy. We may also look into experience accumulated in the MONET project [18] and other existing technologies such as MathServe [11] and MathBroker II [12].

Furthermore, while developing this protocol we discovered that we have some need for representing mathematical objects in OpenMath that are not met by the current set of content dictionaries. This includes, for example, finitely presented groups, character tables of finite groups, and efficient representation of large matrices over finite fields. We plan to investigate these difficulties and create new content dictionaries where necessary.

**Acknowledgement.** The authors wish to thank the anonymous reviewers for their useful comments.

## References

1. Besche, H.U., Eick, B.: Construction of finite groups. J. Symbolic Comput. 27(4), 387–404 (1999)
2. Besche, H.U., Eick, B., O'Brien, E.: The Small Groups Library, http://www-public.tu-bs.de:8080/~beick/soft/small/small.html
3. The Centre for Interdisciplinary Research in Computational Algebra (St Andrews, Scotland), http://www-circa.mcs.st-and.ac.uk/
4. CNRS, École Polytechnique (Palaiseau, France), http://www.polytechnique.fr/
5. The Dependable Systems Research Group at Heriot-Watt University, Edinburgh, Scotland, http://www.macs.hw.ac.uk/~dsg/content/public/home/home.php
6. The Discrete Algebra and Geometry group at the Technical University of Eindhoven, Netherlands, http://www.win.tue.nl/dw/dam/
7. Gamble, G., Nickel, W., O'Brien, E.: ANUPQ — ANU p-Quotient, GAP4 package, http://www.math.rwth-aachen.de/~Greg.Gamble/ANUPQ/
8. The GAP Group: GAP — Groups, Algorithms, and Programming, http://www.gap-system.org
9. Von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (1999)
10. Grayson, D.R., Stillman, M.E.: Macaulay 2, a software system for research in algebraic geometry, http://www.math.uiuc.edu/Macaulay2/
11. The MathServe Framework, http://www.ags.uni-sb.de/~jzimmer/mathserve.html
12. MathBroker II: Brokering Distributed Mathematical Services, http://www.risc.uni-linz.ac.at/research/parallel/projects/mathbroker2/
13. Institute e-Austria Timisoara, Romania, http://www.ieat.ro/
14. The KANT group at the Technical University of Berlin, Germany, http://www.math.tu-berlin.de/~kant/
15. Freundt, S., Horn, P., Konovalov, A., Linton, S., Roozemond, D.: Symbolic Computation Software Composability Protocol (SCSCP) Specification, Version 1.1. CIRCA (preprint, 2008), http://www.symbolic-computation.org/scscp/
16. Konovalov, A., Linton, S.: SCSCP — Symbolic Computation Software Composability Protocol. GAP 4 package
17. Maplesoft, Inc, Waterloo, Canada, http://www.maplesoft.com/
18. MONET, http://monet.nag.co.uk/
19. MuPAD, http://www.sciface.com
20. OpenMath, http://www.openmath.org
21. RISC-Linz, Austria, http://www.risc.uni-linz.ac.at/
22. SAGE: Open Source Mathematics Software, http://www.sagemath.org/
23. Roozemond, D.: OpenMath Content Dictionary: scscp1, http://www.win.tue.nl/SCIEnce/cds/scscp1.html
24. Roozemond, D.: OpenMath Content Dictionary: scscp2, http://www.win.tue.nl/SCIEnce/cds/scscp2.html
25. Symbolic Computation Infrastructure for Europe, http://www.symbolic-computation.org/
26. Research Group Computational Mathematics, Department of Mathematics, University of Kassel, Germany, http://www.mathematik.uni-kassel.de/compmath