REPORT ON OpenDreamKit DELIVERABLE D3.3

Support for the SCSCP interface protocol in all relevant components (SAGE, GAP, etc.) distribution

LUCA DE FEO, ALEXANDER KONOVALOV, STEVE LINTON, TOM WIESING

Due on	02/28/2017 (Month 18)
Delivered on	02/28/2017
Lead	University of St Andrews (USTAN)
Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/62`	

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #62 ON 2017-02-28

- **WP3:** Component Architecture
- **Lead Institution:** University of St Andrews
- **Due:** 2016-08-31 (month 12)
- **Nature:** Other
- **Task:** T3.2 (#51)
- **Proposal:** p.43
- **Final report:** due 2017-02-28

SCSCP stands for the **Symbolic Computation Software Composability Protocol** - the remote procedure call framework by which different software components (primarily mathematical software systems) may offer computational services to a variety of possible clients using the OpenMath encoding both for the data and protocol instructions (see the **SCSCP specification** for further details).

SCSCP has been developed in the EU FP6 project 026133 SCIEnce - Symbolic Computation Infrastructure for Europe. In the duration of the project (2006-2011) and subsequent years, several native CAS implementations of SCSCP client and server, and also APIs for Java, C and C++ had appeared (see the complete list here). However, there were no Python OpenMath SCSCP implementations (except a prototype quality client supporting only lists of integers) and that hindered further extension of the SCSCP framework.

In this deliverable, we have extended support for SCSCP to other relevant systems involved in the project. This builds foundation to D3.9 “Semantic-aware Sage interface to GAP” (#68) and other activities outlined in our paper “Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach” (Intelligent Computer Mathematics. CICM 2016. Lecture Notes in Computer Science, vol 9791. Springer). More specifically, we have achieved the ability to communicate using SCSCP protocol to the following systems/languages:

- ✓ Python (both versions 2 and 3): via pure pip-installable packages
- ✓ openmath
  - PyPI: `https://pypi.python.org/pypi/openmath`
  - GitHub: `https://github.com/OpenMath/py-openmath`
- ✓ scscp
  - PyPI: `https://pypi.python.org/pypi/scscp`

  - GitHub: `https://github.com/OpenMath/py-scscp`
- ✓ SageMath: via Python packages listed above
- ✓ LMFDB: via Python packages listed above
- ✓ PARI:
- ✓ support via D4.1 “Python/Cython bindings for PARI and its integration in Sage” (#83)
- [ ] later via D4.10 “Second version of the PARI Python/Cython bindings” (#84)
- ✓ GAP: via (updated versions of) GAP packages:
- ✓ OpenMath
  - Website: `https://gap-packages.github.io/openmath/`
  - GitHub: `https://github.com/gap-packages/openmath`
- ✓ SCSCP
  - Website: `https://gap-packages.github.io/scscp/`
  - GitHub: `https://github.com/gap-packages/scscp`
- ✓ Singular: via GAP and/or SageMath
- ✓ MMT/MathHub: OpenMath and SCSCP client/server implementations in Scala

---

Remarks:

- Relevant tickets in Sage: `https://trac.sagemath.org/ticket/19970` and `http://trac.sagemath.org/ticket/19971`
- In view of the new DFG-funded project “Symbolic Tools in Mathematics and Their Application” which will develop the OSCAR system, based on Julia, it’s desirable to implement OpenMath and SCSCP in Julia and later use Singular through it (see the blog post `https://wbhart.blogspot.co.uk/2016/11/new-computer-algebra-system-oscar_20.html`)

CONTENTS

1. Introduction

SCSCP stands for the **Symbolic Computation Software Composability Protocol** – a remote procedure call framework by which different software components (primarily mathematical software systems) may offer computational services to a variety of possible clients, such as, for example:

- a computer algebra system (CAS) running on the same computer system or remotely;
- another instance of the same CAS (in a parallel computing context);
- a simplistic SCSCP client (e.g. C/C++/Python/etc. program) with a minimal SCSCP support needed for a particular application;
- an internet application providing user interface to the computational services;
- a Web server which passes on the same services as Web services to other applications;
- Cloud middleware

using the OpenMath encoding (see `http://www.openmath.org/`) both for the data and protocol instructions.

SCSCP has been developed in the EU FP6 project 026133 "SCIEnce – Symbolic Computation Infrastructure for Europe" (`http://www.symbolic-computing.org/`). Within the duration of that project (2006-2011) and over subsequent years, native implementations of SCSCP client and server have been developed in several CAS such as GAP, KANT, Macaulay2, Maple, Mathematica, MuPAD and TRIP; furthermore, APIs for Java, C and C++ were developed to facilitate SCSCP implementations in other applications (see the full list and links at `http://www.symbolic-computing.org/`). However, there were no Python implementations of OpenMath and SCSCP (except a prototype quality client supporting only lists of integers) and that hindered further extension of the SCSCP framework.

In this report we give an overview of work under task T3.2 to support SCSCP in all relevant components, necessary to be able to build interfaces between different systems.

2. Systems

Under task T3.2, our goal is to make SCSCP-compliant all relevant components. Since SCSCP is a protocol in which both data and instructions are encoded in OpenMath, its implementation is typically split into two components:

- OpenMath support to be able to parse and generate OpenMath code and encode/decode mathematical objects
- client and server functionality needed to establish connection and exchange with procedure calls and their results.

In order to achieve maximal impact on the Python ecosystem, instead of implementing SCSCP directly in Sage, our approach was to provide independent Python libraries, which may be used not only in Sage but also in a number of other Python applications including scientific computing libraries such as e.g. SciPy, NumPy, SymPy etc.

2.1. Python and Sage

OpenMath and SCSCP support in Python is provided by two libraries, openmath and scscp, which are available from the package repository PyPI at

- `https://pypi.python.org/pypi/openmath/`
- `https://pypi.python.org/pypi/scscp/`

and may be installed using the standard pip installer both under Python 2 and Python 3. The development repositories and issue trackers for these packages are hosted on GitHub at

- `https://github.com/OpenMath/py-openmath`
- `https://github.com/OpenMath/py-scscp`

The openmath package contains

- An object representation of the OpenMath standard,
- XML serialisers and deserialisers,
- An extendable mechanism for automatic conversions between Python and OpenMath objects.

We give details on the two latter components in the appendices.

The scscp package offers network components for implementing the SCSCP network protocol at various levels of abstraction. Among other components, it contains in particular

- An SCSCP base client,
- An SCSCP base server,
- A command line tool to interrogate SCSCP server, with automatic discovery of remote procedures, and automatic conversion between Python and OpenMath.
- A minimalistic demo server, and other examples illustrating how to use the package inside SAGE.

Both packages can easily be installed inside SAGE from PyPi. Although the client and server modules target generic Python code, they can easily be adapted to work seamlessly inside SAGE, as the included examples show. Since SAGE does not have extensive support for OpenMath yet, we have decided not to include these packages in SAGE by default for the moment. We will reevaluate this decision according to the progress made in WP6 in the next reporting period.

Appendices A and B contain README files for Python packages openmath and scscp respectively. Further appendices contain examples of using these packages for communication between different systems:

- from Python 2 to GAP (Appendix C)
- from Python 3 to GAP (Appendix D),
- from SAGE to GAP (Appendix E),
- from SymPy Python 3 library to SINGULAR via GAP (Appendix F)
- from GAP to NumPy Python 3 library (Appendix G)

All of these examples were prepared in the form of Jupyter notebooks, and could be reproduced using the original `.ipynb` files in the project repository.

2.2. GAP

OpenMath and SCSCP support in GAP is provided by two similarly named GAP packages, OpenMath and SCSCP. These packages are included in the standard GAP distribution and are also available from their websites:

- `https://gap-packages.github.io/openmath/`
- `https://gap-packages.github.io/scscp/`

The development repositories and issue trackers for these packages are hosted on GitHub:

- `https://github.com/gap-packages/openmath`
- `https://github.com/gap-packages/scscp`

Both packages have an extensive (20 and 54 pages respectively) documentation which is included in their distribution and is also available online:

- `https://gap-packages.github.io/openmath/doc/chap0.html`
- `https://gap-packages.github.io/scscp/doc/chap0.html`

The GAP package OpenMath provides an OpenMath phrasebook for GAP. It can import and export mathematical objects encoded in OpenMath for the purpose of exchanging them with other OpenMath-enabled applications. It supports both XML and binary OpenMath encodings, and allows users to extend it to support private OpenMath content dictionaries in case when

the official OpenMath content dictionaries do not exist or provide too verbose and inefficient encoding.

The SCSCP package implements both an SCSCP client and a server. The client is capable of establishing the connection with the specified server at the specified port; sending the `procedure_call` message to the server in both blocking and non-blocking modes; and fetching response in the form of the `procedure_completed` or `procedure_terminated` message.

The server reads all declarations from the specification file and starts to listen to the specified port (by default port 26133). Upon receiving and incoming connection, it starts the "receive-evaluate-return" loop: receive the `procedure_call` message; evaluate the result (or produce a side-effect); return the result in the form of the `procedure_completed` message, or returns an error in the form of `procedure_terminated` message.

The ability to issue concurrent non-blocking procedure calls permitted to build a framework for distributed parallel computations using the "master-worker" skeleton.

The SCSCP package had been established under the EU FP6 Programme project 026133 "SCIEnce – Symbolic Computation Infrastructure for Europe", and OpenMath project had been created with the support of the EU ESPRIT project EP 24969 "Accessing and Using Mathematical Information Electronically" and then had a major redevelopment under the SCIEnce project. However, they were not in active development after the SCIEnce project had finished.

Under OpenDreamKit, during the reported period we have prepared new versions of both packages and migrated their development repositories and websites to host them under the GAP packages virtual organisation on GitHub (see `https://gap-packages.github.io/`). Their new versions have now passed integration tests for the redistribution with GAP and will appear in the next GAP release. Major changes were upgrades and compatibility fixes for next GAP releases, improvements to improve the security and robustness when running public SCSCP servers, and fixes for problems discovered while testing Python packages for OpenMath and SCSCP.

Several examples of using GAP SCSCP client and server are given in the next section and Appendices. In addition to already mentioned examples of GAP SCSCP server called from Python application shown in Appendices C-G, we also demonstrate several use cases for the GAP SCSCP package in Appendices H, I and J.

2.3. **PARI/GP**

Given that the goal of PARI is to be a small, fast and lightweight library and interpreter, we have come to the conclusion that adding support for SCSCP directly to it is inadequate.

Instead, the Python library mentioned above, combined with the CyPari interface built in D4.1: "Python/Cython bindings for PARI and its integration in Sage", will enable building fast and reliable SCSCP client and servers for PARI through the Python platform.

Both CyPari, and the packages built for this deliverable are not mature enough to showcase such applications, but we may be able to do so by the time D4.10: "Second version of the PARI Python/Cython bindings" is completed.

2.4. SINGULAR

Both GAP and SAGE have very flexible interfaces to SINGULAR which permit to provide access SINGULAR functionality via SCSCP from these systems, for example, as shown in Appendix F which describes how one could call it from the SymPy Python 3 library through a GAP SCSCP server.

Given that the perspective for SINGULAR has evolved in the meantime, and with the launch of the DFG-funded project "Symbolische Werkzeuge in der Mathematik und ihre Anwendung" ("Symbolic Tools in Mathematics and their Application") SINGULAR may later switch to Julia as a high level language for SINGULAR and other involved systems), we have come to the

conclusion that at this stage implementing a Singular-specific SCSCP client/server would be premature. Instead, it may be useful to establish an implementation of SCSCP in Julia, to be used uniformly by SINGULAR and other systems.

2.5. **MMT/MathHub**

MMT supports both OpenMath syntax and the SCSCP protocol. A separate SCSCP functionality for MathHub is not necessary, as it is already powered by MMT and thus any SCSCP communication can easily go via MMT instead.

Unlike with other systems MMT Terms are already OpenMath objects with a few custom extensions. As such, the translation from an MMT to an OpenMath object is just dropping the extra information provided by these extensions and the backward translation is simply the identity.

We have added a data structure that represents these "pure" (i.e. standards-conformant) OpenMath objects to MMT, the source code of which can be found at `https://github.com/UniFormal/MMT/tree/177a9fc3415856c426110a7e763c7d7843de4d5f/src/mmt-odk/src/info/kwarc/mmt/odk/OpenMath`. It in particular implements the translation procedure above as well as serialising and deserialising OpenMath from XML.

On top of this, we have also implemented a custom SCSCP Client as well as Server inside of MMT. They are written in Scala.

The client functionality allows MMT to first encode an MMT term (i.e. any object) as an OpenMath object, then have any SCSCP server perform a computation on this object, and finally receive the response and translate it back into an MMT term. Furthermore, the server functionality allows MMT to expose computational functionality to other systems. In this situation, it again receives an OpenMath object, translates it as an MMT object, performs some computation, and returns the result as an OpenMath object.

Having SCSCP functionality implemented in Scala has the advantage that we can directly and easily embed it into our existing codebase. However, as we had to implement it from scratch, there are certain corner cases that we have not been able to take into account so far. Furthermore it might be difficult to have to maintain this implementation and update in the future. Thus we are considering to migrate away from our own implementation and instead simply write a Scala wrapper around the pre-existing Java Implementation.

Appendices K and L show two examples of the implemented SCSCP Client functionality. The first one shows how the Scala implementation of the SCSCP Client can connect to an external server, the second one again shows such a connection, but this time starts out within MMT, goes through the procedure described above and then receives a result as an MMT term.

3. EXAMPLES AND USE CASES

In this section we give a brief overview of Appendices C-J and mention several highlights illustrating flexibility of our setup.

3.1. **Examples involving applications from Python ecosystem**

Appendices C-F show various examples of interactions between Python, SAGE, GAP and SINGULAR. All these examples are produced using Jupyter notebooks and may be reproduced by re-running the original `.ipynb` files contained in the project repository on GitHub.

In Appendix C an SCSCP client in Python2 communicates with the GAP SCSCP server. One of the highlights is creating remote objects on the GAP server and manipulating with them using remote references without retrieving actual objects.

Appendix D demonstrates SCSCP client in Python3 connecting to the GAP SCSCP server. Python 2 supports matrices, but it does not support matrix groups. When it needs to find the catalogue number of the group generated by given matrices, it sends those generators to the GAP server.

Appendix E shows SCSCP client in SAGE connecting to GAP server. There SAGE retrieves a group of order 512 represented in a private OpenMath dictionary, which only can be processed by GAP. However, it can be used as an argument in next procedure calls in order to investigate properties of this group.

Appendix F describes accessing SINGULAR from Python3 via a GAP SCSCP server. We show an example of calculating a Gröbner basis when remote procedure call to SINGULAR takes 6 seconds where as the same calculation with SymPy library in Python is taking 2 minutes. Besides the PDF version of the Jupyter notebook, we show the server configuration file for the GAP SCSCP server.

Finally, in Appendix G we show how SCSCP client in GAP communicates with the Python 3 server to operate with matrices and polynomials using the NumPy library. For example, the calculation of the rank and determinant of an integer matrix also has a well-performing implementation in GAP, but it may be interesting to run it in Python to check the reproducibility of the result. The other example in this Appendix uses NumPy to calculate floating-point approximations of complex roots of a polynomial, and such functionality is not available in GAP at all.

3.2. GAP SCSCP server in Docker container

A very efficient and secure way to run a public GAP SCSCP server is to run it in the GAP Docker container. We have established a pipeline of GAP Docker containers under Task T3.1 (see `https://hub.docker.com/u/gapsystem/`) to serve different use cases, and the one which is most suited for the SCSCP server is the `gapsystem/gap-docker` container available at `https://hub.docker.com/r/gapsystem/gap-docker/` and providing the most complete installation of the GAP system and all of the more than 130 packages currently redistributed with GAP, with satisfied dependencies on the external software libraries.

If the SCSCP server configuration file `myserver.g` is contained, for example, in the directory `path`, then one could mount that directory as `/scscp/` on the container and start the GAP SCSCP server as follows:

```
docker run --rm -i -t --net="host" -v path:/scscp gapsystem/gap-docker gap /scscp/myserver.g
```

A copy of the page `https://hub.docker.com/r/gapsystem/gap-docker/` with detailed instructions on using the GAP Docker container is provided in Appendix H.

3.3. The database of numbers of isomorphism types of finite groups

One of SCSCP use cases is accessing large and/or frequently updated databases, which should be rather accessed online that redistributed as add-ons for mathematical software packages. This is particularly relevant to work on mathematical databases under WP6.

Appendix I contains a description of a proof-of-concept example how such access may be organised to provide a crowdsourced database of numbers of isomorphism types of finite groups (see `https://github.com/alex-konovalov/gnu`). Using SCSCP, contributors may be able to get the latest information of known and unknown entries from the server, which runs in the GAP Docker container (see 3.2).

3.4. Parallel search in the GAP Small Groups Library using SCSCP

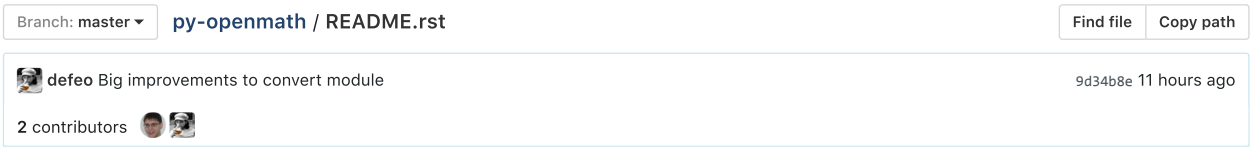
Appendix J contains a quick start guide for setting up the environment for parallel search in the GAP Small Groups Library. This guide and supplementary files could be found at `https://github.com/alex-konovalov/scscp-demo`. The example used in this guide is based on the GAP Software Carpentry lesson (`http://alex-konovalov.github.io/gap-lesson/`). This material was included in the programme of two CoDiMa training schools in Computational Discrete Mathematics held in the UK in 2015 and 2016 (`http://www.codima.ac.uk/schools/`).

4. Conclusions and future work

SCSCP implementation forms the 1st step of Task T3.2 “Interfaces between systems” and in particular builds foundation for D3.9 “Semantic-aware SAGE interface to GAP”, as well as feeding into many activities under Workpackage 6. While undertaking this work, we have identified a number of new activities which may flow from it, either within Workpackage 2 (dissemination) or as part of other projects beyond OpenDreamKit, increasing the impact of the project.

First, due to establishing pure Python packages for OpenMath and SCSCP instead of implementing them directly in SAGE, we are now able to reach out to wider Python communities, in particular to users and developers of NumPy, SciPy, SymPy and other scientific libraries that may need to provide or consume computational services via SCSCP. Also, we would like to establish similar implementations for other languages which may be used as interface languages to various scientific libraries. We plan to start this with the Julia language due to its importance for the new DFG-funded project “Symbolic Tools in Mathematics and Their Application” which will develop the OSCAR system, based on Julia. Finally, to ensure that the SCSCP standard will be properly maintained in the future, we plan to publish the stable SCSCP specification through the OpenMath society and license it for general use under the terms of the “W3C Software and document notice and license”.

APPENDIX A. README FILE FOR THE PYTHON OPENMATH PACKAGE



pyopenmath

build passing

Python OpenMath 2.0 implementation.

Description

OpenMath is an extensible standard for representing the semantics of mathematical objects.

Installation

```
pip install openmath
```

Usage

This package provides an object implementation of OpenMath, and XML parsing/serialization.

See py-scscp for an example of use.

XML Serialization

The modules `encoder` and `decoder` provide XML de-serialization for OpenMath objects.

```
>>> from openmath import encoder, decoder, openmath as om
>>> xml = encoder.encode_xml(om.OMString('hello world')); xml
<Element {http://www.openmath.org/OpenMath}OMSTR at 0x7fcb3cd82708>
>>> b = encoder.encode_bytes(om.OMString('hello world')); b
b'<OMSTR xmlns="http://www.openmath.org/OpenMath">hello world</OMSTR>'
>>> decoder.decode_xml(xml)
OMString('hello world', id=None)
>>> decoder.decode_bytes(b, snippet=True)
OMString('hello world', id=None)
```

Conversions between Python and OpenMath

This package provides facilities for easy conversions from Python to OpenMath and back. The module `convert` contains two functions, `to_python()` and `to_openmath()`, that do the conversion as their names suggest, or raise a `ValueError` if no conversion is known.

This module only implements conversions for basic Python types:

- bools,

- ints,
- floats,
- complex numbers,
- strings,
- bytes,
- lists (recursively),
- sets (recursively).

Furthermore, any object that defines an `__openmath__(self)` method will have that method called by `to_python`.

Finally, this module contains a mechanism for registering converters.

```
>>> from fractions import Fraction
>>> from openmath import convert, openmath as om
>>> def to_om_rat(obj):
...     return om.OMApplication(om.OMSymbol('rational', cd='nums1'),
...                             list(map(convert.to_openmath, [obj.numerator, obj.denominator])))
...
>>> def to_py_rat(obj):
...     return Fraction(convert.to_python(obj.arguments[0]), convert.to_python(obj.arguments[1]))
...
>>> convert.register(Fraction, to_om_rat, 'nums1', 'rational', to_py_rat)
>>> omobj = convert.to_openmath(Fraction(5, 6)); omobj
OMApplication(OMSymbol('rational', 'nums1', id=None, cdbase=None), [OMInteger(5, id=None), OMInteger(6, id=None)], ic
>>> convert.to_python(omobj)
Fraction(5, 6)
```

Contributing

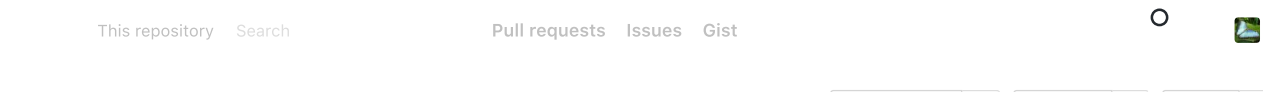
The source code of this project can be found on GitHub. Please use GitHub issues and pull requests to contribute to this project.

Credits

This work is supported by OpenDreamKit.

License

APPENDIX B. README FILE FOR THE PYTHON SCSCP PACKAGE



SCSCP

build passing

SCSCP 1.3 implementation for Python.

Description

The Symbolic Computation Software Composability Protocol (SCSCP) is a network protocol for software systems to exchange mathematical objects. Think RPC (Remote Procedure Call) for CAS (Computer Algebra Systems).

Installation

```
pip install scscp
```

Usage

This package provides a command-line SCSCP client and a base class that an SCSCP server may extend. An example implementation of an SCSCP server is also provided.

Server

The module `scscp.server` provides a class `SCSCPServer` that an SCSCP server may extend. Lower level classes are also available, for more details see the API docs.

This source distribution also contains an example server `examples/demo_server.py`, capable of performing very basic arithmetic operations. To run the demo server, simply run:

```
python examples/demo_server.py
```

Client

The module `scscp.client` provides a class `SCSCPClient` that an SCSCP client may extend. Lower level classes are also available, for more details see the API docs.

The package also contains a synchronous command-line client `scscp.SCSCPCLI` to query SCSCP servers. To connect to a server running on, e.g., localhost, type

```
>>> from scscp import SCSCPCLI
>>> c = SCSCPCLI('localhost')
```

The client automatically queries the server for the available functions, and populates the `heads` attribute:

```
>>> c.heads
{'arith1': ['minus', 'abs', 'power', 'divide', 'unary_minus', 'plus', 'times'], 'scscp2': ['get_allowed_heads', 'get_
```

Functions on the server can be queried via the syntax `c.heads.<cd>.<func>(args)` where `<cd>` is the name of the OpenMath *content dictionary*, `<func>` is the name of the function, and args is the list of arguments.

Integers, floats, complex numbers, booleans, strings, lists and binary data are automatically converted to and from Python native types.

```
>>> c.heads.arith1.power([2, 100])
1267650600228229401496703205376
```

The client also understands OpenMath data via the openmath package, which can be used to express more complex data.

```
>>> from openmath import openmath as om
>>> c.heads.arith1.power([om.OMInteger(2), om.OMInteger(100)])
1267650600228229401496703205376
```

To disconnect the client, simply use the `quit()` method.

```
>>> c.quit()
```

Contributing

The source code of this project can be found on GitHub. Please use GitHub issues and pull requests to contribute to this project.

Credits

This work is supported by OpenDreamKit.

License

APPENDIX C. EXAMPLE: SCSCP CLIENT IN PYTHON2 CONNECTING TO GAP SERVER

Example of SCSCP client in Python2 connecting to GAP server

```
In [1]: from scscp import SCSCPCLI
```

- Establishing connection with the demo SCSCP server

```
In [2]: c = SCSCPCLI('scscp.gap-system.org')
```

- Ask for the list of supported procedures

```
In [3]: c.heads
```

```
Out[3]: {'scscp_transient_1': ['SCSCPStartTracing', 'Addition', 'IO_UnpickleSt
        ringAndPickleItBack', 'NrConjugacyClasses', 'ConwayPolynomial', 'Small
        Group', 'GroupIdentification', 'AutomorphismGroup', 'IdGroup512ByCod
        e', 'Phi', 'Factorial', 'GnuExplained', 'MathieuGroup', 'TransitiveGro
        up', 'PrimitiveGroup', 'Multiplication', 'NextUnknownGnu', 'Identity',
        'IsPrimeInt', 'Gnu', 'Determinant', 'LatticeSubgroups', 'Length', 'Mat
        rixMultiplication', 'SCSCPStopTracing', 'AlternatingGroup', 'Symmetric
        Group', 'IdGroup', 'SylowSubgroup', 'GnuWishlist', 'Size']}
```

- A simplest "ping-pong" test which sends an object to the server and gets it back

```
In [4]: c.heads.scscp_transient_1.Identity([1])
```

```
Out[4]: 1
```

- Examples of some procedure calls

```
In [5]: c.heads.scscp_transient_1.Factorial([10])
```

```
Out[5]: 3628800
```

```
In [6]: c.heads.scscp_transient_1.IsPrimeInt([2**16+1])
```

```
Out[6]: True
```

- In the next example, we calculate the symmetric group of degree 3

```
In [7]: g = c.heads.scscp_transient_1.SymmetricGroup([3])
```

- This group does not map to an object defined in Python, so it is stored in its internal representation

```
In [8]: g
```

```
Out[8]: OMApplication(OMSymbol('group', 'permgp1', id=None, cdbase=None), [OMS
        ymbol('right_compose', 'permutation1', id=None, cdbase=None), OMApplic
        ation(OMSymbol('permutation', 'permut1', id=None, cdbase=None), [OMInt
        eger(2, id=None), OMInteger(3, id=None), OMInteger(1, id=None)], id=No
        ne, cdbase=None), OMApplication(OMSymbol('permutation', 'permut1', id=
        None, cdbase=None), [OMInteger(2, id=None), OMInteger(1, id=None)], id
        =None, cdbase=None)], id=None, cdbase=None)
```

- But we can use it as an argument in SCSCP procedure calls, for example, to find its order and the catalogue number in the GAP Small Groups Library

```
In [9]: c.heads.scscp_transient_1.Size([g])
```

```
Out[9]: 6
```

```
In [10]: c.heads.scscp_transient_1.NrConjugacyClasses([g])
```

```
Out[10]: 3
```

```
In [11]: c.heads.scscp_transient_1.IdGroup([g])
```

```
Out[11]: [6, 1]
```

However, this is not very efficient:

- OpenMath encoding for an object may be quite verbose
- Sending it to the GAP server will create a new object instead of using the existing one
- There may be situations when GAP may not be able to convert the result into OpenMath

For such scenarious, SCSCP specification defines remote objects

- Create a remote copy of the alternating group of degree 5 and ask to return a reference

```
In [12]: g=c.heads.scscp_transient_1.AlternatingGroup([5],cookie=True)
```

```
In [13]: g
```

```
Out[13]: OMReference('scscp://chrystal.mcs.st-andrews.ac.uk:26133/TEMPVarSCSCPJ
         XmNfISQ', id=None)
```

- The reference could be used as an arguments of the procedure call
- While the client does not "know" what this groups is, it can certainly "understand" various numerical properties of this group

```
In [14]: c.heads.scscp_transient_1.Size([g])
```

```
Out[14]: 60
```

- Now we will create an automorphism group of the given group and receive another reference

```
In [15]: a = c.heads.scscp_transient_1.AutomorphismGroup([g],cookie=True)

In [16]: a

Out[16]: OMReference('scscp://chrystal.mcs.st-andrews.ac.uk:26133/TEMPVarSCSCPT
         XMaSBJM', id=None)
```

- And then we are able to investigate various numerical properties of that group

```
In [17]: c.heads.scscp_transient_1.Size([a])

Out[17]: 120

In [18]: c.heads.scscp_transient_1.NrConjugacyClasses([a])

Out[18]: 7

In [19]: c.heads.scscp_transient_1.IdGroup([a])

Out[19]: [120, 34]
```

- Close the connection

```
In [20]: c.quit()
```

APPENDIX D. EXAMPLE: SCSCP CLIENT IN PYTHON3 CONNECTING TO GAP SERVER

Example of SCSCP client in Python3 connecting to GAP server

In [1]:

```
from scscp import SCSCPCLI
```

- Establishing connection

In [2]:

```
c = SCSCPCLI('scscp.gap-system.org')
```

- Ask for the list of supported procedures

In [3]:

```
c.heads
```

Out[3]:

```
{'scscp_transient_1': ['IdGroup512ByCode', 'IO_UnpickleStringAndPick
leItBack', 'IdGroup', 'ConwayPolynomial', 'Factorial', 'GroupIdentif
ication', 'Multiplication', 'Determinant', 'Phi', 'SCSCPStopTracin
g', 'AlternatingGroup', 'TransitiveGroup', 'Size', 'Identity', 'Auto
morphismGroup', 'SCSCPStartTracing', 'SymmetricGroup', 'MathieuGrou
p', 'Length', 'GnuExplained', 'NextUnknownGnu', 'Addition', 'GnuWish
list', 'IsPrimeInt', 'PrimitiveGroup', 'LatticeSubgroups', 'SmallGro
up', 'MatrixMultiplication', 'NrConjugacyClasses', 'SylowSubgroup',
 'Gnu']}
```

- A simplest test

In [4]:

```
c.heads.scscp_transient_1.Identity([1])
```

Out[4]:

```
1
```

- Determinant of a matrix

In [5]:

```
c.heads.scscp_transient_1.Determinant([ [ [ 1,2 ],[ 3,4] ] ])
```

Out[5]:

```
-2
```

- Number of groups of order 10000

In [6]:

```
c.heads.scscp_transient_1.Gnu([10000])
```

Out[6]:

```
4728
```

- What is the catalogue number of the group generated by matrices a and b

In [7]:

```
a = [ [ 0,-1], [1,-1] ]
```

In [8]:

```
b = [ [-1, 1], [0,1 ] ]
```

In [9]:

```
c.heads.scscp_transient_1.GroupIdentification([[a,b]])
```

Out[9]:

```
[6, 1]
```

- Close the connection

In [10]:

```
c.quit()
```

APPENDIX E. EXAMPLE: SCSCP CLIENT IN SAGE CONNECTING TO GAP SERVER

Example of SCSCP client in SageMath connecting to GAP server

```
In [1]: from scscp import SCSCPCLI
```

- Establish connection

```
In [2]: c = SCSCPCLI('scscp.gap-system.org')
```

```
In [3]: c.heads
```

```
Out[3]: {'scscp_transient_1': ['SCSCPStartTracing', 'Addition', 'IO_UnpickleStringAndPic
        kleItBack', 'NrConjugacyClasses', 'ConwayPolynomial', 'SmallGroup', 'GroupIdenti
        fication', 'AutomorphismGroup', 'IdGroup512ByCode', 'Phi', 'Factorial', 'GnuExpl
        ained', 'MathieuGroup', 'TransitiveGroup', 'PrimitiveGroup', 'Multiplication', '
        NextUnknownGnu', 'Identity', 'IsPrimeInt', 'Gnu', 'Determinant', 'LatticeSubgrou
        ps', 'Length', 'MatrixMultiplication', 'SCSCPStopTracing', 'AlternatingGroup', '
        SymmetricGroup', 'IdGroup', 'SylowSubgroup', 'GnuWishlist', 'Size']}
```

- The simplest example

```
In [4]: c.heads.scscp_transient_1.Identity([int(1)])
```

```
Out[4]: 1
```

- Working with GAP Small Groups Library

```
In [5]: g=c.heads.scscp_transient_1.SmallGroup([int(512),int(13)])
```

```
In [6]: g
```

```
Out[6]: OMApplication(OMSymbol('pcgroup_by_pcgscode', 'pcgroup1', id=None, cdbase=None),
        [OMInteger(1144084885715361616239395874018497928530277871 7L, id=None), OMInteger
        (512, id=None)], id=None, cdbase=None)
```

```
In [7]: c.heads.scscp_transient_1.NrConjugacyClasses([g])
```

```
Out[7]: 92
```

```
In [8]: c.heads.scscp_transient_1.NrConjugacyClasses([c.heads.scscp_transient_1.SmallGro
        up([int(512),int(13)])])
```

```
Out[8]: 92
```

- Close connection

```
In [9]: c.quit()
```

APPENDIX F. EXAMPLE: SCSCP CLIENT IN PYTHON3 CALCULATES GROÖBNER BASIS WITH SINGULAR

Example: SCSCP client in Python3 calculates Groebner basis with Singular

Python users needing an implementation of the Groebner basis algorithm may use SymPy (http://www.sympy.org/ (http://www.sympy.org/)) - a symbolic computation library that, among other features, contains a polynomial manupulation module.

In this example we demonstrate an alternative and much faster approach, which first uses SymPy to create multivariate polynomials, and calls GAP SCSCP server to pass them to Singular.

Because SymPy is presently unable to encode/decode polynomias in OpenMath, this requires desigining remote procedures and their calls to pass external representations of these polynomials in the form of lists of integers, which both systems support, demonstrating the flexibilty of our approach.

```
In [1]: import sympy
```

```
In [2]: from sympy.polys import ring, ZZ, QQ
```

```
In [3]: from scscp import SCSCPCLI
```

- Create a multivariate polynomial

```
In [4]: R, x, y, z = ring("x, y, z", ZZ)
```

```
In [5]: f = x*y*z+y**2*z+x**2*z+1
```

```
In [6]: f
```

```
Out[6]: x**2*z + x*y*z + y**2*z + 1
```

- Presently SymPy does not implement OpenMath support for polynomials, so we will be passing their external representation instead. The following lists describe monomials and corresponding coefficients.

```
In [7]: coeffs = f.coeffs()
```

```
In [8]: coeffs
```

```
Out[8]: [1, 1, 1, 1]
```

```
In [9]: mons = [ list(x) for x in f.monoms() ]
```

```
In [10]: mons
```

```
Out[10]: [[2, 0, 1], [1, 1, 1], [0, 2, 1], [0, 0, 0]]
```

- We will need the following two functions for conversion between SymPy polynomials and their external representations

In [11]:
```
def ext_rep_poly(f):
    return [ f.coeffs(), [ list(x) for x in f.monoms() ] ]
```

In [12]:
```
from numpy import prod
def construct_poly(R,extrep):
    g = R.gens
    coeffs = extrep[0]
    mons = extrep[1]
    return sum ( [ coeffs[m]*prod([g[i]**mons[m][i] for i in
range(len(g))]) for m in range(len(mons))] )
```

- Obviously, the following condition should always hold

In [13]:
```
g = construct_poly(R,ext_rep_poly(f))
```

In [14]:
```
f == g
```

Out[14]: True

- Similar functions for conversion between GAP polynomials and their external representation (as produced by SymPy) have been defined on the GAP SCSCP server. Let's test that we can send polynomials back and forth using the "Ping-Pong" test which encodes and decodes each polynomial twice - on the SymPy's side and on the GAP's side.

In [15]:
```
c = SCSCPCLI('localhost')
```

In [16]:
```
f == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext_rep_
poly(f)]))
```

Out[16]: True

In [17]:
```
c.quit()
```

- Now we show a small example of a Groebner basis computation with SymPy

In [18]:
```
R, x0, x1, x2, x3 = ring("x0, x1, x2, x3", ZZ)
```

In [19]:
```
f1=x0+x1+x2+x3
f2=x0*x1+x1*x2+x0*x3+x2*x3
f3=x0*x1*x2+x0*x1*x3+x0*x2*x3+x1*x2*x3
f4=x0*x1*x2*x3-1
```

```
In [20]: time(sympy.polys.groebnertools.groebner([f1,f2,f3,f4],R))
```

```
CPU times: user 9.42 ms, sys: 473 µs, total: 9.89 ms
Wall time: 9.64 ms
```

```
Out[20]: [x0 + x1 + x2 + x3,
          x1**2 + 2*x1*x3 + x3**2,
          x1*x2 - x1*x3 + x2**2*x3**4 + x2*x3 - 2*x3**2,
          x1*x3**4 - x1 + x3**5 - x3,
          x2**3*x3**2 + x2**2*x3**3 - x2 - x3,
          x2**2*x3**6 - x2**2*x3**2 - x3**4 + 1]
```

- To calculate it remotely, first we start new SCSCP session

```
In [21]: c = SCSCPCLI('localhost')
```

- Just another check for passing polynomials around

```
In [22]: all( t == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext
         _rep_poly(t)])) for t in [f1,f2,f3,f4] )
```

```
Out[22]: True
```

- Now call the remote procedure `GroebnerBasisWithSingular` with polynomials from the example above

```
In [23]: bas = c.heads.scscp_transient_1.GroebnerBasisWithSingular( [ [ ext_rep_p
         oly(x) for x in [f1,f2,f3,f4] ] ] )
```

- The result came in external representation, so we have to convert it to SymPy polynomials

```
In [24]: [ construct_poly(R,t) for t in bas ]
```

```
Out[24]: [x0 + x1 + x2 + x3,
          x1**2 + 2*x1*x3 + x3**2,
          x1*x2**2 - x1*x3**2 + x2**2*x3 - x3**3,
          x1*x2*x3**2 - x1*x3**3 + x2**2*x3**2 + x2*x3**3 - x3**4 - 1,
          x1*x3**4 - x1 + x3**5 - x3,
          x2**3*x3**2 + x2**2*x3**3 - x2 - x3,
          x1*x2 - x1*x3 + x2**2*x3**4 + x2*x3 - 2*x3**2]
```

- Finally, close SCSCP session

```
In [25]: c.quit()
```

- Now we present an example when remote calculation with Singular is much faster than local calculation with SimPy

```
In [26]: R, x0, x1, x2, x3, x4 = ring("x0, x1, x2, x3, x4", ZZ)
```

In [27]:
```
f1=x0+x1+x2+x3+x4
f2=x0*x1+x1*x2+x2*x3+x0*x4+x3*x4
f3=x0*x1*x2+x1*x2*x3+x0*x1*x4+x0*x3*x4+x2*x3*x4
f4=x0*x1*x2*x3+x0*x1*x2*x4+x0*x1*x3*x4+x0*x2*x3*x4+x1*x2*x3*x4
f5=x0*x1*x2*x3*x4-1
```

- Local calculation with SymPy takes about 2 minutes

In [28]:
```
time(sympy.polys.groebnertools.groebner([f1,f2,f3,f4,f5],R))
```

```
CPU times: user 1min 56s, sys: 915 ms, total: 1min 57s
Wall time: 1min 59s
```

Out[28]:
```
[x0 + x1 + x2 + x3 + x4,
 275*x1**2 + 825*x1*x4 + 550*x3**6*x4 + 1650*x3**5*x4**2 + 275*x3**4*x
4**3 - 550*x3**3*x4**4 + 275*x3**2 - 566*x3*x4**11 - 69003*x3*x4**6 +
 69019*x3*x4 - 1467*x4**12 - 178981*x4**7 + 179073*x4**2,
 275*x1*x2 - 275*x1*x4 + 275*x2**2 + 550*x2*x4 - 330*x3**6*x4 - 1045*x
3**5*x4**2 - 275*x3**4*x4**3 + 275*x3**3*x4**4 - 550*x3**2 + 334*x3*x4
**11 + 40722*x3*x4**6 - 40726*x3*x4 + 867*x4**12 + 105776*x4**7 - 1058
73*x4**2,
 275*x1*x3 - 275*x1*x4 - 110*x3**6*x4 - 440*x3**5*x4**2 - 275*x3**4*x4
**3 + 275*x3**3*x4**4 + 124*x3*x4**11 + 15092*x3*x4**6 - 15106*x3*x4 +
346*x4**12 + 42218*x4**7 - 42124*x4**2,
 55*x1*x4**5 - 55*x1 + x4**11 + 143*x4**6 - 144*x4,
 275*x2**3 + 550*x2**2*x4 - 550*x2*x4**2 + 275*x3**6*x4**2 + 550*x3**5
*x4**3 - 550*x3**4*x4**4 + 550*x3**2*x4 - 232*x3*x4**12 - 28336*x3*x4*
*7 + 28018*x3*x4**2 - 568*x4**13 - 69289*x4**8 + 69307*x4**3,
 275*x2*x3 - 275*x2*x4 + 440*x3**6*x4 + 1210*x3**5*x4**2 - 275*x3**3*x
4**4 + 275*x3**2 - 442*x3*x4**11 - 53911*x3*x4**6 + 53913*x3*x4 - 1121
*x4**12 - 136763*x4**7 + 136674*x4**2,
 55*x2*x4**5 - 55*x2 + x4**11 + 143*x4**6 - 144*x4,
 55*x3**7 + 165*x3**6*x4 + 55*x3**5*x4**2 - 55*x3**2 - 398*x3*x4**11 -
48554*x3*x4**6 + 48787*x3*x4 - 1042*x4**12 - 127116*x4**7 + 128103*x4*
*2,
 55*x3**2*x4**5 - 55*x3**2 - 2*x3*x4**11 - 231*x3*x4**6 + 233*x3*x4 -
 8*x4**12 - 979*x4**7 + 987*x4**2,
 x4**15 + 122*x4**10 - 122*x4**5 - 1]
```

- But remote calculation with Singular takes about 6 seconds

In [29]:
```
c = SCSCPCLI('localhost')
```

In [30]:
```
all( t == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext
_rep_poly(t)])) for t in [f1,f2,f3,f4,f5] )
```

Out[30]:
```
True
```

```
In [31]: time( [ construct_poly(R,t) for t in \
               c.heads.scscp_transient_1.GroebnerBasisWithSingular( [ [ ext_rep_
poly(x) for x in [f1,f2,f3,f4,f5] ] ] ) ] )
```

```
CPU times: user 5.85 s, sys: 21.9 ms, total: 5.87 s
Wall time: 6.01 s
```

Out[31]:
```
[x0 + x1 + x2 + x3 + x4,
 x1**2 + x1*x3 + 2*x1*x4 - x2*x3 + x2*x4 + x4**2,
 x1*x2*x3 - 2*x1*x3**2 - 2*x1*x3*x4 + 3*x1*x4**2 + x2**3 + 3*x2**2*x4
 - x2*x3**2 - 2*x2*x3*x4 + 3*x2*x4**2 - x3**3 - 3*x3**2*x4 - 2*x3*x4**
2 + 2*x4**3,
 x1*x2**2 - x1*x2*x3 + x1*x3*x4 - x1*x4**2 + x2**2*x3 - x2**2*x4 + x2*
x3*x4 - 2*x2*x4**2 + x3**2*x4 + x3*x4**2 - x4**3,
 14*x1*x2*x3*x4 - x1*x2*x4**2 - 27*x1*x3**2*x4 - 10*x1*x3*x4**2 + 24*x
1*x4**3 + 6*x2**2*x3*x4 + 7*x2**2*x4**2 + 2*x2*x3**2*x4 - 9*x2*x3*x4**
2 + 33*x2*x4**3 + x3**4 - 15*x3**3*x4 - 33*x3**2*x4**2 - 14*x3*x4**3 +
22*x4**4,
 32*x1*x2*x3*x4 - 24*x1*x2*x4**2 - 40*x1*x3**2*x4 - 12*x1*x3*x4**2 + 4
4*x1*x4**3 + 11*x2**2*x3*x4 - 3*x2**2*x4**2 + 19*x2*x3**3 + 10*x2*x3**
2*x4 - 45*x2*x3*x4**2 + 32*x2*x4**3 + 5*x3**4 + x3**3*x4 - 32*x3**2*x4
**2 - 32*x3*x4**3 + 34*x4**4,
 3*x1*x2*x3*x4 - 4*x1*x2*x4**2 + x1*x3**3 - 2*x1*x3**2*x4 - 2*x1*x3*x4
**2 + 4*x1*x4**3 + x2**2*x3*x4 - 2*x2**2*x4**2 + 3*x2*x3**3 + 3*x2*x3*
*2*x4 - 7*x2*x3*x4**2 - x2*x4**3 + x3**4 + 3*x3**3*x4 + x3**2*x4**2 -
 4*x3*x4**3 + 2*x4**4,
 -x1*x2*x3*x4 + 2*x1*x2*x4**2 - 2*x1*x3**3 + 2*x1*x3*x4**2 - x1*x4**3
 + x2**2*x3**2 - x2**2*x3*x4 + x2**2*x4**2 - x2*x3**3 - 2*x2*x3**2*x4
 + 3*x2*x3*x4**2 + 2*x2*x4**3 - x3**4 - 2*x3**3*x4 - 2*x3**2*x4**2 + 2
*x3*x4**3,
 x1*x2*x3**2 + x1*x2*x3*x4 - x1*x2*x4**2 - x1*x3**2*x4 - x1*x3*x4**2 +
x1*x4**3 + x2**2*x3*x4 + x2*x3**2*x4 + x2*x4**3 - x3**3*x4 - 2*x3**2*x
4**2 - x3*x4**3 + x4**4,
 2*x1*x2*x3*x4**2 - x1*x2*x4**3 - 2*x1*x3*x4**3 + x1*x4**4 + x2**2*x3*
x4**2 + 2*x2*x3**2*x4**2 - x2*x3*x4**3 + x2*x4**4 - x3**2*x4**3 - 2*x3
*x4**4 + x4**5 - 1,
 x1*x4**5 - x1 - x2*x4**5 + x2,
 -20*x1*x2*x4**4 + 5*x1*x3*x4**4 + 15*x1 - 20*x2**2*x4**4 + 15*x2*x3**
2*x4**3 - 25*x2*x3*x4**4 - 23*x2*x4**5 - 7*x2 + 10*x3**3*x4**3 + 30*x3
**2*x4**4 - 3*x3*x4**5 + 3*x3 - 4*x4**6 + 24*x4,
 -3*x1*x2*x4**4 + 11*x1*x3**2*x4**3 - 2*x1*x3*x4**4 - 6*x1 - 3*x2**2*x
4**4 + 5*x2*x3**2*x4**3 - x2*x3*x4**4 - 15*x2*x4**5 + 5*x2 + 7*x3**3*x
4**3 + 10*x3**2*x4**4 - x3*x4**5 + x3 - 5*x4**6 - 3*x4,
 2*x2*x3*x4**5 - 2*x2*x3 + 8*x2*x4**6 - 8*x2*x4 + x3**2*x4**5 - x3**2
 + x3*x4**6 - x3*x4 + 3*x4**7 - 3*x4**2,
 3*x2**2*x4**5 - 3*x2**2 - 2*x2*x3*x4**5 + 2*x2*x3 + x2*x4**6 - x2*x4
 - x3**2*x4**5 + x3**2 - x3*x4**6 + x3*x4,
 -9*x1*x2*x4**5 - x1*x2 + 11*x1*x3*x4**5 - x1*x3 - 3*x1*x4**6 + 3*x1*x
4 - 6*x2**2*x4**5 - 4*x2**2 - 5*x2*x3*x4**5 - 9*x2*x4**6 - 6*x2*x4 + 5
*x3**3*x4**4 + 14*x3**2*x4**5 + x3**2 + 5*x3*x4**6 - 5*x3*x4 - 3*x4**7
+ 13*x4**2,
 42*x1*x2*x3 - 76*x1*x2*x4 - 165*x1*x3**2 + 13*x1*x3*x4 + 186*x1*x4**2
+ 21*x2**2*x3 - 55*x2**2*x4 + 42*x2*x3**2 - 131*x2*x3*x4 + 21*x2*x4**2
- 55*x3**3 - 21*x3**2*x4 - 42*x3*x4**2 + x4**8 + 219*x4**3,
 -110*x1*x2*x3 + 29*x1*x2*x4 + 52*x1*x3**2 - 34*x1*x3*x4 + 63*x1*x4**2
- 55*x2**2*x3 - 26*x2**2*x4 + 60*x2*x3**2 - 102*x2*x3*x4 - 120*x2*x4**
2 + 39*x3**3 + 120*x3**2*x4 + x3*x4**7 + 109*x3*x4**2 - 26*x4**3,
 -112*x1*x2*x3 + 33*x1*x2*x4 + 61*x1*x3**2 - 35*x1*x3*x4 + 53*x1*x4**2
- 56*x2**2*x3 - 23*x2**2*x4 + 58*x2*x3**2 - 95*x2*x3*x4 + 8*x2*x4**7 -
129*x2*x4**2 + 42*x3**3 + 121*x3**2*x4 + x3*x4**7 + 111*x3*x4**2 + 3*x
4**8 - 41*x4**3,
 36*x1*x2*x3 - 11*x1*x2*x4 - 37*x1*x3**2 - 7*x1*x3*x4 + 19*x1*x4**2 +
 8*x2**3 + 14*x2**2*x3 + 27*x2**2*x4 - 20*x2*x3**2 + x2*x3*x4 + 53*x2*
x4**2 - 20*x3**3 + x3**2*x4**6 - 54*x3**2*x4 - 44*x3*x4**2 + 34*x4**3]
```

In [32]:
```
c.quit()
```

```
# GAP SCSCP server for the example of calling Singular from Python SCSCP client

LogTo(); # to close the log file in case it was opened earlier
LoadPackage("singular");
LoadPackage("scscp");

# create polynomial from its external representation
AssemblePolynomial := function( extrep )
local fam, rep, coeffs, mons, i, term, j, p;
fam := RationalFunctionsFamily(FamilyObj(1));
rep := [ ];
coeffs := extrep[1];
mons   := extrep[2];
for i in [1..Length(coeffs)] do
  term:=[];
  for j in [1..Length(mons[i])] do
    if mons[i][j]>0 then
      Append(term,[j,mons[i][j]]);
    fi;
  od;
  Append( rep, [ term, coeffs[i] ] );
od;
p:=PolynomialByExtRep(fam,rep);
return p;
end;

# produce external representation of a polynomial
DisassemblePolynomial:=function(f)
local rep, coeffs, mons, deg, t, r, i, term, mon, j;
rep := ExtRepPolynomialRatFun(f);
coeffs := [];
mons   := [];
deg    := Maximum( List( Filtered(rep{[1,3..Length(rep)-1]}, t -> Length(t)>0),
                         r -> Maximum(r{[1,3..Length(r)-1]}) ) );
for i in [1,3..Length(rep)-1] do
  term := rep[i];
  mon := ListWithIdenticalEntries(deg,0);
    for j in [1,3..Length(term)-1] do
      mon[term[j]]:=term[j+1];
    od;
  Add( mons, mon );
  Add( coeffs, rep[i+1]);
od;
return [coeffs,mons];
end;

# This is the main purpose of this server
GroebnerBasisWithSingular:=function( extreps )
# it accepts external representations of polynomials
local R, r, I, B;
# create polynomial ring of appropriate rank
R:=PolynomialRing( Rationals, Maximum(List( extreps, r -> Length(r[2]) ) ) );
# convert arguments to polynomials and get an ideal they generate
I:=Ideal( R, List( extreps, AssemblePolynomial ) );
# call local instance of Singular
B:=GroebnerBasis(I);
# return result in the form of external representations
return List(B,DisassemblePolynomial);
end;

# Procedures that the GAP SCSCP server provides

# Useful for simple tests
InstallSCSCPprocedure( "Identity", x -> x,
  "Identity procedure for tests", 1, 1 );
```

```
# Clearly, f = AssemblePolynomial( DisassemblePolynomial( f ) )
PingPongPoly := x -> DisassemblePolynomial( AssemblePolynomial ( x ) );
InstallSCSCPprocedure( "PingPongPoly", PingPongPoly,
  "Decode/encode polynomial and send it back", 1, 1 );

# Setting up calculation and calling Singular
InstallSCSCPprocedure( "GroebnerBasisWithSingular", GroebnerBasisWithSingular,
  "Groebner Basis with Singular", 1, 1 );

# Start GAP SCSCP server
RunSCSCPserver( SCSCPserverAddress, SCSCPserverPort : OMignoreMatrices);
```

APPENDIX G. EXAMPLE: SCSCP CLIENT IN GAP CONNECTING TO PYTHON 3 SERVER

Example of GAP SCSCP client connecting to Python 3 SCSCP server

In this example GAP SCSCP client communicates with the Python 3 SCSCP server. The Python code is based on https://github.com/OpenMath/py-scscp/blob/master/demo_server.py (https://github.com/OpenMath/py-scscp/blob/master/demo_server.py)

Simple calls

```
In [1]: EvaluateBySCSCP("plus",[2,2],"localhost",26133:cd:="arith1").object
```

```
4
```

```
In [2]: EvaluateBySCSCP("plus",[[1,2],[3,4]],"localhost",26133:cd:="arith1").obj
        ect
```

```
[ 1, 2, 3, 4 ]
```

In Python, addition of lists and strings is their concatenation

```
In [3]: EvaluateBySCSCP("plus",["abc","def"],"localhost",26133:cd:="arith1").obj
        ect
```

```
"abcdef"
```

Using NumPy linear algebra tools

In the next example, we extend Python server to offer some procedures from the NumPy package for scientific computing (http://www.numpy.org/ (http://www.numpy.org/)). To do that, we need only to add several more lines to the Python script to run the server:

```
import numpy

CD_SCSCP_TRANSIENT1 = {
    'numpy.linalg.det'         : numpy.linalg.det,
    'numpy.linalg.matrix_rank' : lambda x: int(numpy.linalg.matrix_rank
(x)),
}
```

- Compute determinant and rank of a random 5x5 matrix

```
In [4]: m:=RandomMat(5,5);
```

```
[ [ 1, 0, -1, -1, -1 ], [ 1, -1, 1, -2, -1 ], [ -2, 0, -1, 2, -2 ], [
 -1, 2, -3, -1, 3 ], [ 0, -2, 1, -4, 0 ] ]
```

```
In [5]: EvaluateBySCSCP("numpy.linalg.det",[m],"localhost",26133:OMignoreMatrice
        s).object;
```

```
-36.
```

```
In [6]: EvaluateBySCSCP("numpy.linalg.matrix_rank",[m],"localhost",26133:OMignor
        eMatrices).object;
```

```
5
```

Let's try with matrices of larger dimensions

```
In [7]: EvaluateBySCSCP("numpy.linalg.det",
        [RandomMat(50,50)],"localhost",26133:OMignoreMatrices).object;
```

```
-7.67794e+49
```

```
In [8]: EvaluateBySCSCP("numpy.linalg.matrix_rank",[RandomMat(50,50)],"localhos
        t",26133:OMignoreMatrices).object;
```

```
50
```

Using NumPy to calculate complex roots of polynomials

Similarly, on the Python server we export another function that calculates (complext) roots of univariate polynomials and returns a list of their real and imaginary parts:

```
def polyroots( coeffs ):
    f = numpy.polynomial.polynomial.Polynomial( coeffs )
    r = f.roots()
    return [ [x.real,x.imag] for x in r]
```

- create polynomials with integer roots

```
In [9]: x:=X(Rationals,"x");
```

```
<object>
```

```
In [10]: f:=(x-10)*(x-1)*(x+5);
```

```
<object>
```

- calculate roots with GAP

```
In [11]: RootsOfUPol(f);
         [ 10, 1, -5 ]
```

- check that Python results agree

```
In [12]: coeffs:=CoefficientsOfUnivariatePolynomial(f)
         [ 50, -45, -6, 1 ]

In [13]: EvaluateBySCSCP("polyroots",[ coeffs ],"localhost",26133:OMignoreMatrice
         s).object;
         [ [ -5., 0. ], [ 1., 0. ], [ 10., 0. ] ]
```

- But GAP can not compute (approximations of) complex roots of another polynomial

```
In [14]: RootsOfUPol(1+2*x+3*x^2);
         [  ]
```

- However, Python with the help of NumPy is capable of doing this

```
In [15]: coeffs := CoefficientsOfUnivariatePolynomial(1+2*x+3*x^2)
         [ 1, 2, 3 ]

In [16]: EvaluateBySCSCP("polyroots",[ coeffs ],"localhost",26133:OMignoreMatrice
         s).object;
         [ [ -0.333333, -0.471405 ], [ -0.333333, 0.471405 ] ]
```

APPENDIX H. DOCUMENTATION FOR THE GAP DOCKER CONTAINER

PUBLIC | AUTOMATED BUILD

gapsystem (/u/gapsystem/)/gap-docker (/r/gapsystem/gap-docker/) ☆

Last pushed: a month ago

Repo Info (/r/gapsystem/gap-docker/)

Short Description

Docker container for GAP system (http://www.gap-system.org) and all packages redistributed with GAP.

Full Description

Docker container for GAP and packages

We have a prebuilt Docker image for GAP and packages at https://registry.hub.docker.com/u/gapsystem/gap-docker/ (https://registry.hub.docker.com/u/gapsystem/gap-docker/).

If you have installed Docker, first you need to download the GAP container using

```
docker pull gapsystem/gap-docker
```

(the same command is needed if you need to pull the new GAP container to get a new GAP release). After that, you can start the GAP container by typing the following in a terminal:

```
docker run --rm -i -t gapsystem/gap-docker
```

Note that you may have to run `docker` with `sudo`, particularly if you are on Ubuntu.

Once the GAP container is started, you can call `gap` inside it to start a new GAP session:

```
gap@11d9377db2bd:~$ gap
*********   GAP 4.8.2, 20-Feb-2016, build of 2016-03-01 01:08:48 (UTC)
*  GAP  *   http://www.gap-system.org
*********   Architecture: x86_64-pc-linux-gnu-gcc-default64
Libs used:  gmp, readline
Loading the library and packages ...
Components: trans 1.0, prim 2.1, small* 1.0, id* 1.0
Packages:   AClib 1.2, Alnuth 3.0.0, AtlasRep 1.5.0, AutPGrp 1.6,
            Browse 1.8.6, CRISP 1.4.1, Cryst 4.1.12, CrystCat 1.1.6,
            CTblLib 1.2.2, FactInt 1.5.3, FGA 1.3.0, GAPDoc 1.5.1, IO 4.
            IRREDSOL 1.2.4, LAGUNA 3.7.0, Polenta 1.3.5, Polycyclic 2.11
            RadiRoot 2.7, ResClasses 4.1.2, Sophus 1.23, SpinSym 1.5,
            TomLib 1.2.5
Try '?help' for help. See also  '?copyright' and  '?authors'
gap>
```

When you leave GAP, you will still be logged in to the container and will need to type `exit` to close it.

Alternatively, you can just type

```
docker run --rm -i -t gapsystem/gap-docker gap
```

to start GAP immediately (and return to the host filesystem after the end of the GAP session). You can put this command in a shell script and make it a default or optional way to start GAP on your system. GAP command line options can be appended after `gap`, for example `docker run --rm -i -t gapsystem/gap-docker gap -A`.

However, note that you will not be able to read a file from your local directory into GAP just by supplying the filename in the command line. Instead, this requires using the option `-v` to mount a local directory. For example, if the current directory contains the subdirectory `examples` with the file `examples/useful.g`, then the option `-v $PWD/examples:/data` will mount `examples` as `/data` on the Docker container. That is, to start GAP and read the file `examples/useful.g` into it, type:

```
docker run -v $PWD/examples:/data -t -i gapsystem/gap-docker gap /data/us
```

Note that the path to `useful.g` is the path in the container, and not in the GAP system.

If you need network access (for example, for packages downloading external data like AtlasRep), call `docker` with the option `--net="host"`, e.g.:

```
docker run --rm -i -t --net="host" gapsystem/gap-docker
```

Combining these options, the following command mounts the directory `pkg/scscp/example` from the GAP distribution as a directory `/scscp` on the container and starts the GAP SCSCP server using the configuration file `gap4rXpY/pkg/scscp/example/myserver.g` :

```
="host" -v ~/gap4rXpY/pkg/scscp/example:/scscp gapsystem/gap-docker gap /s
```

At the moment, almost all packages are in working order. External software needed by some packages at the moment includes:

- Ubuntu packages libmpfr-dev libmpfi-dev libmpc-dev libfplll-dev (needed by the float package)
- Polymake 2.14 (and dependencies, listed on polymake.org)
- Singular (git version of the day)
- 4ti2 1.6.3
- PARI/GP.

Work is in progress to configure the remaining packages that have non-standard installation procedures or dependencies on external components: Carat, ITC, Linboxing, ParGAP and XGAP.

Docker Pull Command

```
docker pull gapsystem/gap-docker
```

Owner

gapsystem

Source Repository

gap-system/gap-docker (https://github.com/gap-system/gap-docker)

Comments (0)

APPENDIX I. DOCUMENTATION FOR THE GAP SCSCP SERVER FOR THE NUMBER OF ISOMORPHISM TYPES OF GROUPS OF ORDER n

gnu

Crowdsourcing project for the database of numbers of isomorphism types of finite groups

What is gnu(n) ?

For an integer n, the number of isomorphism types of finite groups of order n is denoted by gnu(n), where "gnu" stands for the "Group NUmber". The problem of the determination of all groups of a given order up to isomorphism is very interesting and challenging. For example, the sequence (https://oeis.org/A000001) in OEIS is the number of groups of order n, with the first unknown entry being gnu(2048). Known values of gnu(n) for 0 < n < 2048 are summarised in "Counting groups: gnus, moas and other exotica" by John H. Conway, Heiko Dietrich and Eamonn A. O'Brien (https://www.math.auckland.ac.nz/~obrien/research/gnu.pdf) which also discusses some properties of gnu(n) and related functions. Both OEIS and the latter paper derive most of the entries of the gnu(n) table from the GAP Small Groups Library (http://www.gap-system.org/Packages/sgl.html) by Hans Ulrich Besche, Bettina Eick and Eamonn O'Brien. The group numbers in the SmallGroups library are to a large extent cross-checked, being computed using different approaches and also compared with theoretical results, where available (see [Hans Ulrich Besche, Bettina Eick and Eamonn O'Brien. A MILLENNIUM PROJECT: CONSTRUCTING SMALL GROUPS. Int. J. Algebra Comput. 12, 623 (2002), http://dx.doi.org/10.1142/S0218196702001115], in particular 4.1. Reliability of the data).

What is known for n > 2048 ?

For n > 2048, the calculation of gnu(n) is highly irregular. Certain orders, including some infinite series (groups of order p^n for n<=6; groups of order q^n*p where q divides 2^8, 3^6, 5^5 or 7^4 and p is a an arbitrary prime not equal to q; groups of squarefree order; groups or order which is a product of at most three primes) are covered by the GAP Small Groups Library (http://www.gap-system.org/Packages/sgl.html) so the gnu(n) is returned by `NrSmallGroups(n)`. The recently submitted GAP package SglPPow (http://www.gap-system.org/Packages/sglppow.html) by Michael Vaughan-Lee and Bettina Eick adds access to groups of order p^7 for p > 11 and to groups of order 3^8 (it should be loaded with `LoadPackage("sglppow")`. For groups of cube-free order, the Cubefee package (http://www.gap-system.org/Packages/cubefree.html) by Heiko Dietrich calculates gnu(n) with `NumberCFGroups(n)`.

How to calculate gnu(n) for arbitrary n ?

For groups of other orders one could try the GAP package GrpConst by Hans Ulrich Besche and Bettina Eick (http://www.gap-system.org/Packages/grpconst.html) to construct all groups of a given order using `ConstructAllGroups(n)`. As documented at http://www.gap-system.org/Manuals/pkg/grpconst/htm/CHAP003.htm, this function usually returns a list of groups, in which case gnu(n) is the length of this list. However, sometimes this list contains sublists. In this case, one has to check each such sublist contains groups which are pairwise non-isomorphic, or remove duplicates.

The runtime and memory requirements of `ConstructAllGroups` depend very much on n and may vary from minimalistic to practically unfeasible. The website of AG Algebra und Diskrete Mathematik (TU Braunschweig) provides the table containing gnu(n) for many n < 50000: http://www.icm.tu-bs.de/ag_algebra/software/small/number.html. These numbers were taken from the Small Groups Library or calculated with the GrpConst package. There is no information in the table for 1082 orders for which the computation have not yet been completed.

Goals of this package

As we see, currently there is no uniform access to the calculation of gnu(n) in GAP even in the case when it is feasible, since one has to call different functions in a different way, dependently on n. Even finding all known gnu(n) for a list of integers with `NrSmallGroups` is not straightforward, since GAP enters a break loop when the library of groups of order n is not available. Also, these data are accessible only from within the working GAP installation. Next, users who calculate new values of gnu(n) have no easy ways to share their data with others and record provenance details, i.e. who calculated them and when, using which hardware and which versions of GAP and relevant packages, and how much memory and runtime were needed. These missing details hinder verification of the results, while it will be useful to have them easily available to cross-check calculations using different approaches, to check the correctness and performance of new implementations that may emerge in the future, and to check that future versions of GAP do not break these calculations. Also, they may be useful for researchers who wants to calculate all groups of a given order with `ConstructAllGroups` and are interested to know in advance how much time it may take and whether someone else had already attempted this calculation.

The Gnu package addresses these problems by:

- Providing uniform access to the calculation of gnu(n) using a single function.
- Offering both the ability to install package locally and to access it remotely without its local installation.
- Providing remote data via SCSCP (Symbolic Computation Software Composability Protocol) to make them accessible to any SCSCP-compliant software (see the list at http://www.symbolic-computing.org).
- Using GitHub-based development model and storing in the revision history the provenance details such as runtime requirements, details of the software and hardware, etc.

Local installation

To use the package locally, first you have to install the GAP system using the source distribution from http://www.gap-system.org/Releases/. Please ensure that you build packages as described there as well. After that, the Gnu package could be installed in the same way like other GAP packages that do not require compilation. It is suggested to install it as a custom package in the `.gap/pkg` subdirectory of your home directory instead of placing it into the `gap4rN/pkg` directory of your GAP installation. Since the package is regularly updated with new data, you may use git to clone it and subsequently pull changes from the main repository. To do this, change to the `.gap/pkg` directory and call

```
git clone https://github.com/alex-konovalov/gnu.git
```

This will create the directory `gnu`. Later when you will need to pull changes, change to that directory and call

```
git pull
```

Alternatively, if you do not use git, you may download a zip-archive from https://github.com/alex-konovalov/gnu/archive/master.zip and later update it manually by downloading new zip-archive and unpacking it to replace the previous installation of the Gnu package.

After loading the package with `LoadPackage("gnu");` you should be able to use it as follows:

```
gap> Gnu(10000);
4728
gap> GnuExplained(10000);
[ 4728, "precomputed using GrpConst package" ]
gap> NextUnknownGnu(10000);
10080
gap> GnuWishlist([2000..3000]);
[ 2048, 2240, 2496, 2560, 2592, 2688, 2880, 2916 ]
gap> List([105,128,2004,10000,2304,3^8,7^2*5^2*11*19,50000],Gnu);
[ 2, 2328, 10, 4728, 15756130, 1396077, 8, false ]
```

You may see some more examples of explanations how the values of gnu(n) were obtained in the following example:

```
gap> List([105,128,2004,10000,2304,3^8,7^2*5^2*11*19,50000],GnuExplained);
[ [ 2, "using NrSmallGroups and the GAP Small Groups Library" ],
  [ 2328, "using NrSmallGroups and the GAP Small Groups Library" ],
  [ 10, "using NrSmallGroups and the GAP Small Groups Library" ],
  [ 4728, "precomputed using GrpConst package" ],
```

```
  [ 15756130, "http://dx.doi.org/10.1016/j.jalgebra.2013.09.028" ],
  [ 1396077, "using NrSmallGroups from SglPPow 1.1" ],
  [ 8, "using NumberCFGroups from CubeFree 1.15" ],
  [ false, "not stored in gnu50000 and no library of groups of size 50000" ] ]
```

Remote connection

It is also possible to access the data without local installation by accessing the dedicated GAP SCSCP server that runs in a Docker container in the Microsoft Azure cloud. This server is periodically restarted to pick up database updates. To access it from GAP, first you need to load the SCSCP package:

```
gap> LoadPackage("scscp");
```

Note that SCSCP package requires the IO package, and the IO package needs compilation on UNIX systems (for Windows, the GAP distributions comes with compiled binaries for the IO package).

After that, download and read (or copy and paste) the following file into GAP: https://raw.githubusercontent.com/alex-konovalov/gnu/master/lib/gnuclient.g

Now you are able to use remote counterparts of the commands shown in the previous section:

```
gap> GnuFromServer(50016);
1208
gap> GnuExplainedFromServer(50080);
[ 1434, "precomputed using GrpConst package" ]
gap> NextUnknownGnuFromServer(50080);
50112
gap> GnuWishlistFromServer([50000..50100]);
[ 50000, 50048 ]
```

If you have locally installed package, then the functions mentioned in this section will become available after its loading.

Note that the server is restarted periodically and may not contain the latest additions to the database. You may check when the server had been started and which version of the package it uses using the `GetServiceDescription` function from the SCSCP package:

```
gap> GetServiceDescription("scscp.gap-system.org",26133);
rec(
  description := "GAP SCSCP server for numbers of isomorphism types of finite \
groups. Gnu package version given by commit https://github.com/alex-konovalov/\
gnu/commit/6630e86ec7b1633b0afaeb7e35e8045561bb8e60. Server started on Sat Jun\
  4 20:46:10 UTC 2016", service_name := "gnu(n) SCSCP service",
  version := "GAP 4.8.3; CubeFree 1.15; Gnu 6630e86ec7b1633b0afaeb7e35e8045561\
bb8e60; GrpConst 2.5; SCSCP 2.1.4; SglPPow 1.1" )
```

To access the GAP SCSCP server from other SCSCP-compliant systems, follow their documentation for SCSCP client functionality and use the server name scscp.gap-system.org and port number 26133 similarly to the calls in https://github.com/alex-konovalov/gnu/blob/master/lib/gnuclient.g

Accessing provenance information

Using git, you can search in the version control history to find the details about the computation. For example, you can find the commit which adds gnu(4000) with the following command

```
git log --grep="gnu(4000)"
```

which will produce the following output:

```
commit dd9ae55743fe465389324bc44e54197bea146dc7
Author: Alexander Konovalov <alexk@mcs.st-andrews.ac.uk>
Date:   Sat May 21 15:33:01 2016 +0100

    gnu(4000)=6108

    GAP 4.8.3
```

```
GrpConst 2.5
Runtime: 975884 ms
Isomorphic groups eliminated

Submitted by @gnufinder. Validated by @alex-konovalov on
Ubuntu 16.04 on Azure cloud standard DS3 v2 instance (4 cores, 14 GB RAM)

Closes #59.
```

Contributing to the database

You can help to the development of this database with the following contributions:

- submitting new values of gnu(n)
- recording information about partial results to be pursued further (for example, when you run `ConstructAllGroups` but were unable to check non-isomorphism, or the computation was not completed after several days)
- verifying existing entries (possibly using other hardware, operating systems, new releases of GAP and related packages)
- improving the functionality of this package

You can submit new values of gnu(n) as new issues or pull requests to the GitHub repository https://github.com/alex-konovalov/gnu (you will have to create a GitHub account if you don't have one yet).

The template for the new issue/pull request will ask you to check that you provide the following details:

- Version of GAP and critical packages: GrpConst, Cubefree, etc.
- Brief description of the computer used for the calculation (operating system, processor, RAM)
- Runtime required for the calculation
- GAP commands used for the calculation
- Confirm that the output `r` of `ConstructAllGroups` is a *list* of groups (`ForAll(r,IsGroup)` should return `true`), or otherwise confirm that if the output contained lists of groups, then those groups were also shown to be pairwise non-isomorphic.

This information will be used to re-run your calculations and add gnu(n) to the database only after they will be verified.

Group orders that are currently not included in the database can be determined using `NextUnknownGnu` , `GnuWishlist` and their remote procedure call counterparts `NextUnknownGnuFromServer` and `GnuWishlistFromServer` as shown above. It may be also useful to look at currently open issues and pull requests since they may contain newly reported results awaiting to be added to the database after their validation. Yo do not need to worry that you may be duplicating someone's else computation, because in this case **DUPLICATION IS REPLICATION** and by checking that you can reproduce the same result with your GAP installation on your computer you will help to improve the quality of the software used in the experiment.

To submit partial results, please create new issues in this repository and tell what you have tried and at which step the calculation stopped. It will be useful to know, for example, about time-consuming cases that did not finish after substantial amount of time, or run out of memory, or where only the first step of the calculation had been completed, but checking the non-isomorphism has not been done.

You can also help with validating new submissions or rechecking existing ones, and with improving mathematical functionality of the package or its infrastructural part.

You may automatically generate almost all the text to submit using the function `GnuByConstructAllGroups` from the `grpconst.g` script located at https://raw.githubusercontent.com/alex-konovalov/gnu/master/lib/grpconst.g, and also included in the `lib` directory of the package. For example (note the double semicolon usage to suppress the output of the returned list):

```
gap> r:=GnuByConstructAllGroups(50024);;
****************************************
Constructing all groups of order 50024
****************************************
#I  computing groups of order [ 2, 2, 2, 13, 13, 37 ]:

#I  compute Frattini factors:
#I    compute ff groups with socle 2 and size 2
#I    compute ff groups with socle 4 and size 8
#I    compute ff groups with socle 8 and size 8
```

```
#I    compute ff groups with socle 13 and size 52
#I    compute ff groups with socle 26 and size 104
...
...
...
#I   extend candidate number 121 of 123 with size 50024
#I   extend candidate number 122 of 123 with size 50024
#I   extend candidate number 123 of 123 with size 50024
#I  found 187 extensions

**************************************
gnu(50024)=197

GAP      4.8.3
GrpConst 2.5
Runtime: 26335 ms
Isomorphic groups eliminated

In case this value is new, add the next line to data/gnudata.g
GNU_SAVE( 50024, 197, WITH_GC );

**************************************
gap>
```

In this case, instead of filling in the template you can copy and paste the last block of lines from the output into the description of an issue or a pull request with the added call to `GNU_SAVE` , and will only need to add the description of the computer used for the computation. In some cases, when `ConstructAllGroups` returns a list with sublists, the message will also contain report about further isomorphism checks. The function `GnuByConstructAllGroups` also returns a record with the output of `ConstructAllGroups` and timings in case it may require further analysis.

Finally, if you are submitting new values of gnu(n) as GitHub issues, please submit strictly one issue per group order. If you are submitting new values of gnu(n) in a pull request, you may submit one value (in which case the simplest way to submit a pull request is to edit the file https://github.com/alex-konovalov/gnu/blob/master/data/gnudata.g via the GitHub's web-interface) or multiple values, in which case each of them should be in an individual commit with the appropriate commit message looking like the summary produced by `GnuByConstructAllGroups` (see https://github.com/alex-konovalov/gnu/pull/58 for an example). Note however that it may take longer time to review such pull request.

Further details and formatting rules could be found in CONTRIBUTING.md: https://github.com/alex-konovalov/gnu/blob/master/CONTRIBUTING.md

Please take a look, and it will be great if you could be involved!

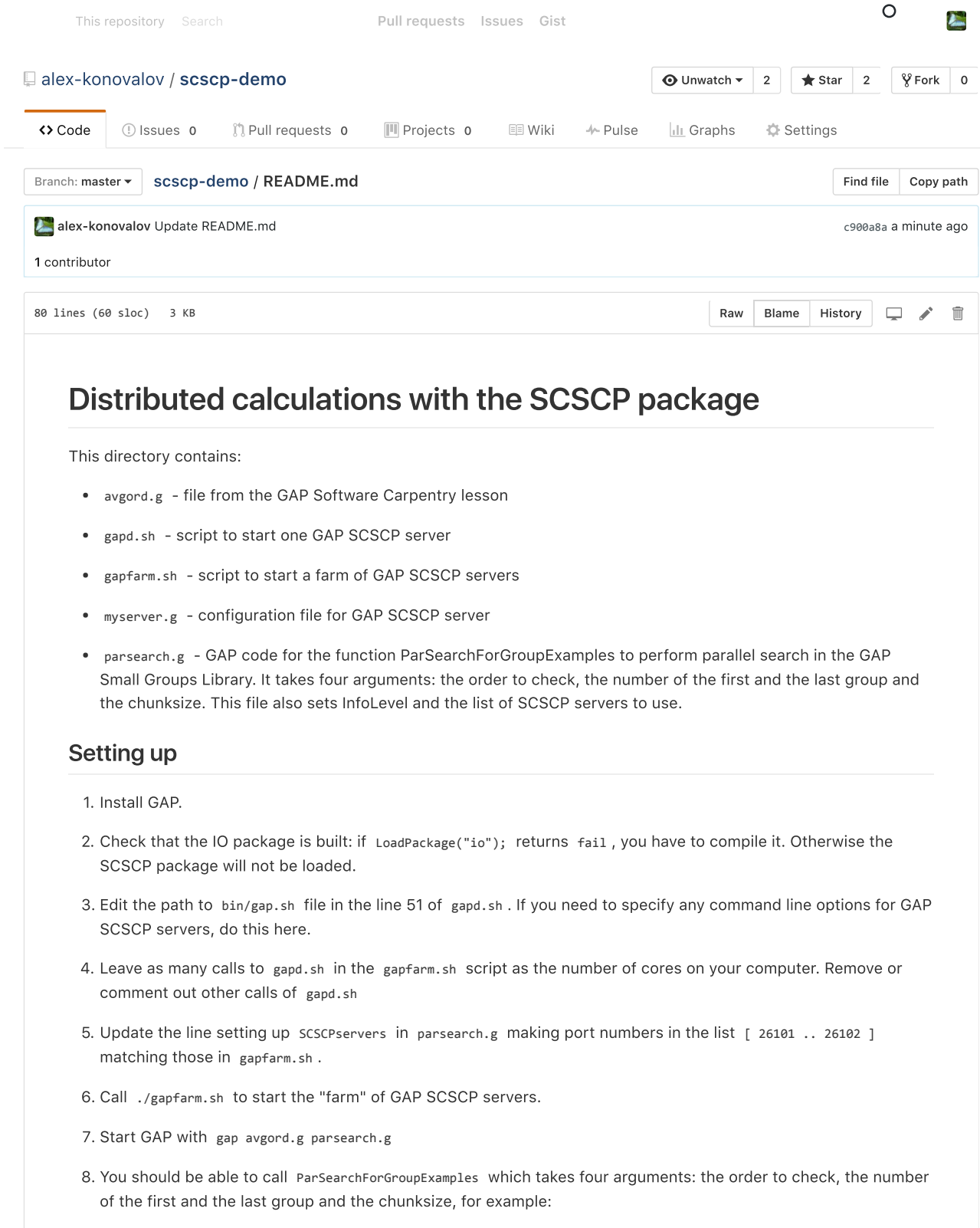
Alexander Konovalov

May 2016

Acknowledgements

We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541). We also acknowledge computational resources that were made available via the Microsoft Azure for Research award.

APPENDIX J. PARALLEL SEARCH IN THE GAP SMALL GROUPS LIBRARY WITH SCSCP



Distributed calculations with the SCSCP package

This directory contains:

- `avgord.g` - file from the GAP Software Carpentry lesson
- `gapd.sh` - script to start one GAP SCSCP server
- `gapfarm.sh` - script to start a farm of GAP SCSCP servers
- `myserver.g` - configuration file for GAP SCSCP server
- `parsearch.g` - GAP code for the function ParSearchForGroupExamples to perform parallel search in the GAP Small Groups Library. It takes four arguments: the order to check, the number of the first and the last group and the chunksize. This file also sets InfoLevel and the list of SCSCP servers to use.

Setting up

1. Install GAP.
2. Check that the IO package is built: if `LoadPackage("io");` returns `fail`, you have to compile it. Otherwise the SCSCP package will not be loaded.
3. Edit the path to `bin/gap.sh` file in the line 51 of `gapd.sh`. If you need to specify any command line options for GAP SCSCP servers, do this here.
4. Leave as many calls to `gapd.sh` in the `gapfarm.sh` script as the number of cores on your computer. Remove or comment out other calls of `gapd.sh`
5. Update the line setting up `SCSCPservers` in `parsearch.g` making port numbers in the list `[ 26101 .. 26102 ]` matching those in `gapfarm.sh`.
6. Call `./gapfarm.sh` to start the "farm" of GAP SCSCP servers.
7. Start GAP with `gap avgord.g parsearch.g`
8. You should be able to call `ParSearchForGroupExamples` which takes four arguments: the order to check, the number of the first and the last group and the chunksize, for example:

```
gap> n:=96;ParSearchForGroupExamples(n,1,NrSmallGroups(n),30);
96
#I  1/8:master --> localhost:26101 : [ 96, 1, 30 ]
#I  2/8:master --> localhost:26102 : [ 96, 31, 60 ]
#I  localhost:26102 --> 2/8:master : [  ]
#I  3/8:master --> localhost:26102 : [ 96, 61, 90 ]
#I  localhost:26101 --> 1/8:master : [  ]
#I  4/8:master --> localhost:26101 : [ 96, 91, 120 ]
#I  localhost:26102 --> 3/8:master : [  ]
#I  5/8:master --> localhost:26102 : [ 96, 121, 150 ]
#I  localhost:26101 --> 4/8:master : [  ]
#I  6/8:master --> localhost:26101 : [ 96, 151, 180 ]
#I  localhost:26102 --> 5/8:master : [  ]
#I  7/8:master --> localhost:26102 : [ 96, 181, 210 ]
#I  localhost:26101 --> 6/8:master : [  ]
#I  8/8:master --> localhost:26101 : [ 96, 211, 231 ]
#I  localhost:26101 --> 8/8:master : [  ]
#I  localhost:26102 --> 7/8:master : [  ]
[ [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ] ]
```

And here is another group with integer average order of its elements:

```
gap> n:=1785;ParSearchForGroupExamples(n,1,NrSmallGroups(n),1);
1785
#I  1/2:master --> localhost:26101 : [ 1785, 1, 1 ]
#I  2/2:master --> localhost:26102 : [ 1785, 2, 2 ]
#I  localhost:26101 --> 1/2:master : [ 1785, 1 ]
#I  localhost:26102 --> 2/2:master : [  ]
[ [ 1785, 1 ], [  ] ]
```

(Of course, no parallelisation is needed to check groups of order 1785, but it is needed to check whether there are other groups with such property among those available in the Small Groups Library).

APPENDIX K. EXAMPLE: SCSCP CLIENT IN SCALA CONNECTING TO GAP SERVER

```
// import the SCSCPClient and OpenMath libraries
import info.kwarc.mmt.odk.SCSCP.Client.SCSCPClient
import info.kwarc.mmt.odk.OpenMath._

// establish a connection
val client = SCSCPClient("scscp.gap-system.org")

// get a list of supported symbols
/**
 * List(OMSymbol(Size,scscp_transient_1,None,None),
 * OMSymbol(Length,scscp_transient_1,None,None),
 * OMSymbol(LatticeSubgroups,scscp_transient_1,None,None),
 * OMSymbol(NrConjugacyClasses,scscp_transient_1,None,None),
 * OMSymbol(AutomorphismGroup,scscp_transient_1,None,None),
 * OMSymbol(Multiplication,scscp_transient_1,None,None),
 * OMSymbol(Addition,scscp_transient_1,None,None),
 * OMSymbol(IdGroup,scscp_transient_1,None,None),
 * ...,
 * OMSymbol(NextUnknownGnu,scscp_transient_1,None,None))
 */
println(client.getAllowedHeads)


// We make a simple example: Apply the identity function to an integer 1
val identitySymbol = OMSymbol("Identity","scscp_transient_1", None, None)
val identityExpression = OMApplication(identitySymbol, List(OMInteger(1, None)), None, None)
/**
 * OMInteger(1,None)
 */
println(client(identityExpression).fetch().get)

// We also try to compute 1 + 1
val additionSymbol = OMSymbol("Addition", "scscp_transient_1", None, None)
val additionExpression = OMApplication(additionSymbol, OMInteger(1, None) :: OMInteger(1, None) :: Nil, None,
None)

/**
 * OMInteger(2,None)
 */
println(client(additionExpression).fetch().get)

// and close the connection
client.quit()
```

APPENDIX L. EXAMPLE: SCSCP CLIENT IN MMT CONNECTING TO GAP SERVER

```
// import mmt terms
import info.kwarc.mmt.api.Path
import info.kwarc.mmt.api.uom.OMLiteral._
import info.kwarc.mmt.api.objects._

// create a group inside of MMT
val mmt_term = OMA(
 OMS(Path.parseS("http://www.gap-system.org?pcgroup1?pcgroup_by_pcgscode")),
 List(
   OMI(BigInt("1144084885715361616239395874018497928530277871 7")),
   OMI(512)
 )
)

/**
 * (http://www.gap-system.org?pcgroup1?pcgroup_by_pcgscode
1144084885715361616239395874018497928530277871 7 512)
 */
println(mmt_term)



// encode it into an OpenMath term (for GAP in this case)
import info.kwarc.mmt.odk.OpenMath.Coding.GAPEncoding
val om_term = GAPEncoding.decodeExpression(mmt_term)

/**
 * OMApplication(
 *   OMSymbol(pcgroup_by_pcgscode,pcgroup1,None,None),
 *   List(
 *     OMInteger(1144084885715361616239395874018497928530277871 7,None),
 *     OMInteger(512,None)
 *   ),None,None)
 */
println(om_term)

// prepare a computation for GAP
// here we compute the nr of conjugacy classes
import info.kwarc.mmt.odk.OpenMath._
val NrConjugacyClasses = OMSymbol("NrConjugacyClasses", "scscp_transient_1", None, None)
val computation = OMApplication(NrConjugacyClasses, List(om_term), None, None)

// fetch the resulting expression from GAP
val client = SCSCPClient("scscp.gap-system.org")
val om_result = client(computation).fetchExpression()
client.quit()

/**
 * OMInteger(92,None)
 */
```

```
println(om_result)

// and turn the result back into an MMT term
val mmt_result = GAPEncoding.encode(om_result)

/**
 * 92
 */
println(mmt_result)
```

Disclaimer: this report, together with its annexes and the reports for the earlier deliverables, is self contained for auditing and reviewing purposes. Hyperlinks to external resources are meant as a convenience for casual readers wishing to follow our progress; such links have been checked for correctness at the time of submission of the deliverable, but there is no guarantee implied that they will remain valid.