Yet Another Cohomology Program

Sage Days 15, Seattle

Christian Nassau

May 19, 2009

Plan of talk

1. Mathematical background
2. Yacop/Sage architecture
3. The Steenrod Tcl library

Moduli spaces

A moduli space $\mathcal{M}$ is usually described by

P	space of parametrizations
C	space of parameter changes

The pair (P, C) is a groupoid scheme.

Want to understand $H^*(\mathcal{M}, \mathcal{L})$ for sheaves $\mathcal{L}$ on $\mathcal{M}$.

$H^0(\mathcal{M}, \omega^k)$ = some ring of modular forms.

Elliptic curves

Elliptic curve over $\mathbb{C}$: $E = \mathbb{C}/\Lambda$, $\Lambda = \mathbb{Z} + \mathbb{Z}\tau$

$$P = \mathfrak{h} = \{\tau \mid \mathrm{Im}\tau > 0\}$$

$$C = \mathrm{Sl}_2(\mathbb{Z})$$

Moduli space is the quotient $\mathfrak{h}/\mathrm{Sl}_2(\mathbb{Z})$.

Elliptic curves

Example: elliptic curves

$P = \operatorname{Spec} \mathbb{Z}[a_1, \ldots, a_6]$ represents Weierstrass curves

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

$C = \operatorname{Spec} \mathcal{O}_P[r, s, t]$ represents coordinate changes[1]

$$x \mapsto x + r, \qquad y \mapsto y + sx + t$$

(P, C) is a description of $\overline{\mathcal{M}_{\mathrm{Ell}}}$.

$H^0\left(\overline{\mathcal{M}_{\mathrm{Ell}}}, \omega^k\right) = \mathbb{Z}[c_4, c_6, \Delta]/(12^3\Delta = c_4^3 - c_6^2) +$ lots of torsion

[1] We ignore the not so interesting $x \mapsto u^2 x$, $y \mapsto u^3 y$.

Formal groups

A formal group is a power series $x +_F y = \sum_{i,j \geq 0} a_{i,j} x^i y^j$ with

$$0 +_F x = x +_F 0 = x, \qquad x +_F y = y +_F x$$
$$(x +_F y) +_F z = x +_F (y +_F z)$$

Examples:

:- $x +_F y = x + y$

:- $x +_F y = x + y + uxy$

:- elliptic curve addition law

$$\begin{aligned} x +_F y = x + y - a_1 xy - a_2(x^2 y + xy^2) \\ - (2a_3 x^3 y - (a_1 a_2 - 3a_3) x^2 y^2 + 2a_3 xy^3) + \cdots \end{aligned}$$

Formal groups

p-locally the universal formal group law is

$$x +_F y = x + y + \sum_{k \geq 1} v_k \Gamma_{p^k}(x, y) + \text{many more terms}$$

where $\Gamma_{p^k}(x, y) = \frac{1}{p}\left((x + y)^{p^k} - x^{p^k} - y^{p^k}\right)$.

Parameter space $P = \operatorname{Spec} \mathbb{Z}_{(p)}[v_1, v_2, \ldots]$.

Coordinate changes $C = \operatorname{Spec} \mathcal{O}_P[t_1, t_2, \ldots]$.

(P, C) describes $\mathcal{M}_{\mathrm{FG}}$.

Topologists are very interested in $H^*(\mathcal{M}_{\mathrm{FG}}, \mathcal{L})$.

Fermionic helpers

Can introduce anticommuting extra parameters to approach the cohomology:

$$EP = \operatorname{Spec} \mathbb{Z}_{(p)}[v_1, v_2, \ldots] \otimes E(\mu_0, \mu_1, \ldots),$$
$$EC = \operatorname{Spec} \mathcal{O}_{EP}[t_1, t_2, \ldots] \otimes E(\tau_0, \tau_1, \ldots),$$
$$\mu_n \mapsto \sum_{i=0}^{n} \mu_i t_{n-i}^{p^i} + \tau_n.$$

With $\partial\mu_n = v_n$ this gives a *differential* Hopf algebroid and there is a spectral sequence

$$\operatorname{Ext}_{(H(EP;\partial),H(EC;\partial))} \Rightarrow \operatorname{Ext}_{(EP,EC)} = H^*\left(\mathcal{M}_{\mathrm{FG}}\right)$$

One has $H(EP;\partial) = \mathbb{F}_p$. For $H(EC;\partial)$ see next page.

Steenrod algebra

$H(EC;\partial)$ is the dual of the (odd) *Steenrod algebra*:

$$H(EC;\partial) =: A_*^{\mathrm{odd}} = \underbrace{\mathbb{F}_p[\zeta_1,\zeta_2,\ldots]}_{=:A_*^{\mathrm{red}}\text{(reduced part)}} \otimes E(\tau_0,\tau_1,\ldots)$$

A_*^{red} represents the group of automorphisms

$$x \mapsto x + \xi_1 x^p + \xi_2 x^{p^2} + \cdots$$

of the additive formal group.

Yacop computes resolutions for subalgebras of A^{odd}.

The subalgebra $A(2)^{\mathrm{odd}}$ dual to $A_*^{\mathrm{odd}}/(\zeta_1^4,\zeta_2^2,\zeta_{k\geq 3},\tau_{l\geq 3})$ corresponds to $\overline{\mathcal{M}}_{\mathrm{Ell}}$ for $p=2$.

A resolution

Plan

Background

:- Moduli spaces

:- Elliptic curves

:- Formal groups

:- Fermionic helpers

:- Steenrod algebra

:- A resolution

:- A chart

Yacop/Sage

Steenrod

The following code computes a resolution of $A(2)^{\mathrm{odd}}$ and opens a GUI window to look at the result:

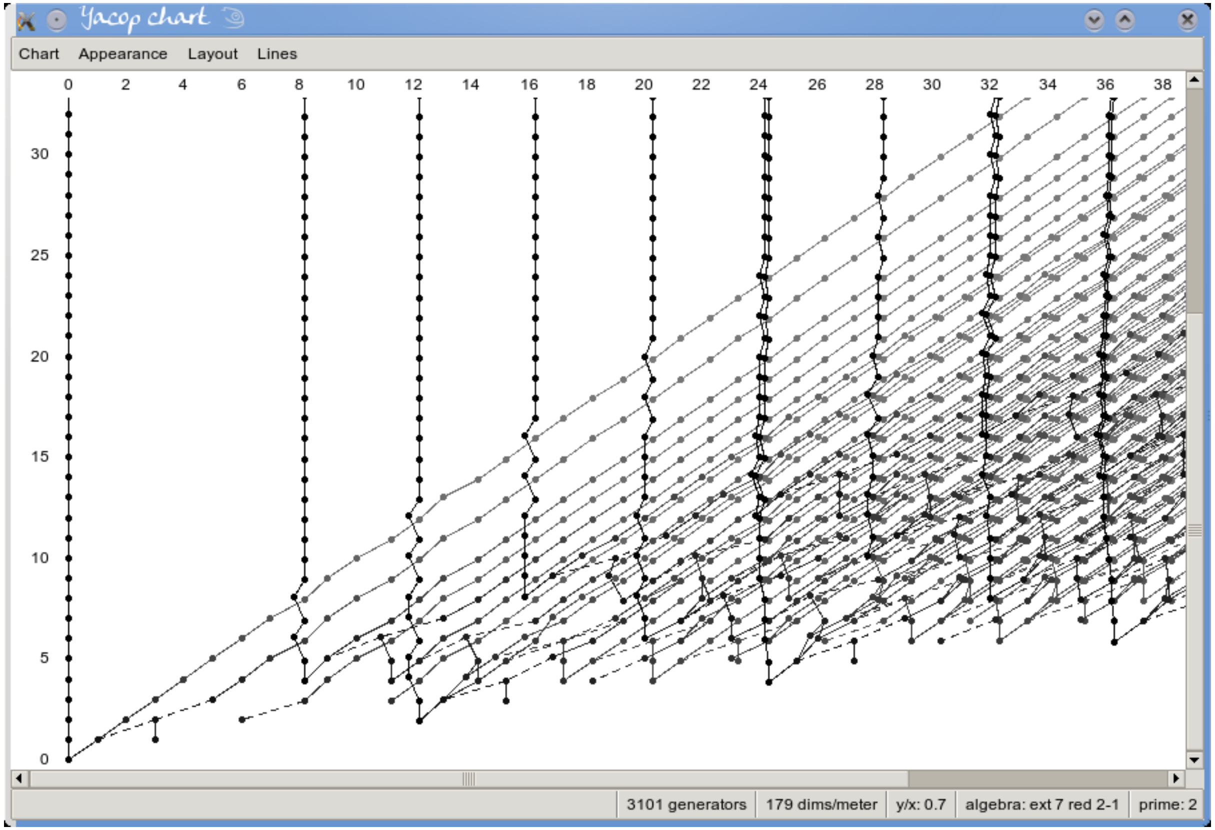
```
sage: # create resolution object with
sage: # an im-memory database
sage: SGFR = SteenrodAlgebraGroundFieldResolution
sage: C = SGFR(prime=2,profile=’7 {2 1}’,filename=":memory:

sage: # compute up to dimension 50, filtration 40
sage: time C.extend(s=40,n=50)
CPU times: user 48.82 s, sys: 4.42 s, total: 53.24 s
Wall time: 53.55 s

sage: # open window
sage: C.gui()
```

A chart

E_2-term of a spectral sequence for $H^*\left(\overline{\mathcal{M}_{\mathrm{Ell}}}\right) \otimes \mathbb{Z}_{(2)}$:



The classes in $(x, y) = (8, 4)$ resp. $(12, 2)$ represent c_4 and c_6.

Features

1. Resolutions C_* of the ground field $\mathbb{F}_p$ over $B \subset A^{\mathrm{odd}}$
2. For a right differential A^{odd}-module X computation of

$$\mathbb{F}_p \otimes_A (X \wedge C_*) \quad \text{and its homology} \quad \mathrm{Tor}_*^A(X, \mathbb{F}_p)$$

3. Chain maps $C_* \to D_*$.

   This includes

   (a) Multiplication on $\mathrm{Ext}_B(\mathbb{F}_p, \mathbb{F}_p)$
   (b) Change of rings maps $\mathrm{Ext}_A \to \mathrm{Ext}_B$
   (c) Frobenius map $\mathrm{Ext}_B^{s,t} \to \mathrm{Ext}_B^{s,pt}$

Architecture

Yacop is written in Tcl, so Sage needs its own Tcl installation.

Tcl packages used:

Tcl/Tk 8.5.7	“Tool command language” Tcl + graphical Toolkit Tk
TclOO 0.6	A new Tcl Object system, will be standard in Tcl 8.6
sqlite 3.6.13	SQLite database package
Steenrod	Steenrod algebra package, written in C
Yacop	Platform independent, written in Tcl

The Yacop code is in `./local/lib/yacop` if you want to hack into it.

Tkinter

The communication between Python and Tcl uses the Tkinter Python module.

Python can call Tcl and get a string result back.

Tcl *cannot* call back into Python.

```
sage: import Tkinter
sage: interp = Tkinter.Tcl()
sage: interp.eval("package require Steenrod")
’1.0’
sage: interp.eval("steenrod::prime 5 inverse 2")
’3’
```

Yacop objects

Sage uses a single Tcl interpreter for all objects.

Every Yacop object has its own namespace in that interpreter.

There is just one thread for all Tcl actions.

```
sage: SGFR = SteenrodAlgebraGroundFieldResolution
sage: C = SGFR(prime=2,profile='7 {2 1}')
sage: C.tcl.eval("namespace current")
'::yacop-1242066772-1'
sage: C.tcl.eval("""
    catch {resolution whatever} errmsg
    set errmsg
""")
'unknown method "whatever": must be algebra, config, db,
destroy, extend-to, isComplete, profmode or viewtype'
```

Data files

Yacop stores its resolutions in SQLite database files.

Files are under `data/yacop` (use `SAGE_DATA` to customize).

You can access the data through

- the `db` subcommand of the Yacop Tcl object,
- the SQL-console of the GUI,
- any SQLite capable application.

```
sqlite3 data/yacop/gfr-steenrod-3-E-1Rfull.db
sqlite> select * from generators where ideg-sdeg = 42;
rowid   id      basid   sdeg    ideg    edeg
26      10      0       3       45      1
128     8       0       8       50      6
152     6       0       9       51      3
173     3       0       10      52      4
```

Fragments

The differentials of the resolution (or the constituents of a chain map) are usually stored in a *fragments* table:

```
CREATE TABLE fragments(/* the summands of the differential
    rowid integer primary key,
    srcgen integer, /* rowid of source generator */
    targen integer, /* rowid of target generator */
    opideg integer, /* internal degree of this piece */
    format text,    /* how data is encoded */
    data text       /* the data */);
```

To get the full differential of a generator g you have to sum up all fragments with `srcgen` = $\mathrm{id}(g)$. The `data` might need to be decoded if `format` is not `tcl`. `targen` should replace the target information from `data`

The `frag_decode(...)` function does all this.

Functions

`frag_decode` and other convenience functions are only available if you access the database through Yacop/Sage.

Some examples:

```
select list(1,2,3) -- Tcl list
1 2 3
select pylist(1,2,3) -- Python list
(1,2,3)
select pydict('kurt','cobain','sd',15) -- dictionary
{"kurt":cobain,"sd":15}
```

Fragments

The differentials in Yacop internally look like this:

```
-- differential of generator for c_6
select frag_decode(format,data,targen) from fragments
where srcgen in (select rowid from generators
                 where sdeg=2 and ideg-sdeg=12)
{1 4 {0 1} 2} {1 1 {3 1} 2}
{1 5 2 3}
{1 3 {0 1} 4}
```

In the usual notation this means

$$d(c_6) = (Q_2P(0,1) + Q_0P(3,1))\, g_2 + Q_0Q_2P^2 g_3 + Q_0Q_1P(0,1) g_4$$

A decorated Milnor basis element (a.k.a. “monomial”) $c \cdot Q(\epsilon)P(R)g_k$ is represented by the Tcl list $\{c\ \epsilon\ \{R\}\ k\}$.

An element of the Steenrod algebra (a.k.a. “polynomial”) is a list of such monomials.

Overview

The Steenrod library provides optimized implementations for some specialised tasks:

- Steenrod algebra multiplication
- Enumeration of basis of $A^{\mathrm{odd}}//B$ where $B \subset A$ Hopf subalgebra.
- Linear algebra over $\mathbb{F}_p$

Multiplication


```
local/bin/tclsh8.5
% package require Steenrod
1.0
% steenrod::poly steenmult {{2 0 1 0}} {{1 1 {} 0}} 3
{2 1 1 0} {2 2 {} 0}
% steenrod::poly steenmult {{1 0 4 0}} {{1 -5 {-3 -5 -2} 0}}
{1 -5 {-3 -6} 0} {1 -3 {-1 -5 -2} 0}
```

The first computation means

$$2P^1 \cdot Q_0 = 2Q_0P^1 + 2Q_1 \quad (p = 3).$$

The second uses $\mathrm{Sq}(-1 - R) = \zeta^R$ and stands for

$$P^4 \cdot \tau_2\zeta_1^2\zeta_2^4\zeta_1 = \tau_2\zeta_1^2\zeta_2^5 + \tau_1\zeta_2^4\zeta_3 \quad (p = 2).$$

Enumeration


```
local/bin/tclsh8.5
% package require Steenrod
1.0
% steenrod::enumerator C -prime 2 -algebra {0 -1 {10 10 10
% C configure -genlist {{0 0 0} {1 5 1}}
% C configure -ideg 10 -edeg 2
% C basis
{1 1 2 1} {1 2 1 1} {1 3 {0 1} 0} {1 3 3 0} {1 5 1 0} {1 6
% C dimension
6
% C seqno {1 3 3 0}
3
```

This code creates a free module on generators g_0 with $(i, e) = (0, 0)$ and g_1 with $(i, e) = (5, 1)$. In bidegree $(10, 2)$ that module has dimension 6 and $Q_0 Q_1 P^3 g_0$ is the 4th basis element.

Enumeration II

The efficient computation of a resolution uses the decomposition $A = B \otimes A//B$.
Here we use the same C but with prescribed $E(Q_0) = B$-components:

```
% C configure -profile {0 1 {} 0}
% C sigreset
% C cget -signature
0 0 {} 0
% C basis
{1 2 1 1} {1 6 {} 0}
% C signext
1
% C cget -signature
0 1 {} 0
% C basis
{1 1 2 1} {1 3 {0 1} 0} {1 3 3 0} {1 5 1 0}
```

Monomaps

The function `steenrod::ComputeMatrix` computes a map $\phi : C \to D$ between free modules. The map must be given as a "monomap" object.

`steenrod::ComputeMatrix` and `steenrod::ComputeImage` are more efficient than a straightforward iteration using `steenrod::poly steenmult`.

Linear algebra

A simple example:

```
% set mat {{1 0 0 1} {3 1 2 2} {1 2 4 4} {1 2 2 3}}
{1 0 0 1} {3 1 2 2} {1 2 4 4} {1 2 2 3}
% steenrod::matrix orthonormalize 5 mat ker
% set mat
{1 0 0 1} {0 1 2 4} {0 0 3 4}
% set ker
{0 3 1 0}
```