EXT.1.9.5

ROBERT R. BRUNER

ABSTRACT. This document is a description of the use of the software included in `ext` and the changes made in this version. These include an important bug fix, a couple more small bug fixes, a lot of small improvements, the deletion of obsolete programs and documentation, and the provision of this new documentation which more accurately reflects the current state of the code. This document should be read together with the document [8], written with John Rognes.

CONTENTS

1. ACKNOWLEDGEMENTS

I owe thanks to many people, including, but not limited to, Don Davis, John Greenlees, Jeff Igo, Tyler Lawson, Mark Mahowald, Christian Nassau and John Rognes for contributions, suggestions and motivation over many years. Jeff Igo wrote the code that allows you to define a module by a *module definition file* instead of having to write a C program. Tyler Lawson wrote `dualizeDef`. John Rognes has made numerous contributions that greatly improve the usability of the code.

2. INTRODUCTION AND ORGANIZATION

I recommend that this document be read in conjunction with the paper [8] in which John Rognes and I describe the use of this software to compute a minimal resolution of $\mathbb{F}_2$ through $(s,t) = (128, 200)$. Many of the details of the algorithms

Date: 19 August 2022.

2010 *Mathematics Subject Classification.* 16E30, 16E40, 18G10, 18G15, 55S10, 55T15.

and data formats are not repeated here if they are given in detail there. An appendix describes the format of each of the data files that are used. A second appendix gives a sample run that a new user may wish to use to familiarize themselves with the system. *A user who is in a hurry may want to turn to that Appendix and do the sample computation outlined there before reading this and the rest of the documentation included here.*

Let $\mathcal{A}$ denote the classical mod 2 Steenrod algebra over $\mathbb{F}_2$ and let $\mathcal{A}(2)$ denote its subalgebra generated by Sq^1, Sq^2 and Sq^4.

This software is designed to be used from the command line in a Unix/Linux environment, including MacOSX. To start, download `ext.1.9.5.tar.gz` from the author's website. Create a directory, which I will call `ext`, in which to unpack everything. (I use the names `ext.1.9.5`, `ext.1.9.3`, *et cetera*, for the directories in which I install the software, to make clear which version is installed there.)

Then issue the commands

```
cd ext
tar xvf DIR/ext.1.9.5.tar.gz
cd A
./Install
cd ../A2
./Install
```

The shell script `Install` will compile various programs and place them where they belong. Here, `DIR` should be the name of the directory containing the downloaded tar file.

The directory `ext` will then contain subdirectories `A` and `A2` in which the work involving $\mathcal{A}$-modules and $\mathcal{A}(2)$-modules, respectively, takes place. Each of these will contain

- subdirectories `src`, `obj`, `bin`, `template`, and `map_template`, in which the source code, object files, and compiled code and shell scripts are located,
- commands `Install` and `Clean` which compile and install the code, and which remove the compiled code, respectively,
- a command `make_all_cocycles` which will be described in section 6,
- a directory `samples` which contains samples of module definitions and of chain map definitions, together with code to create or manipulate them, to be described in section 4,
- commands `newmodule`, `newmap`, and `cocycle` which create modules and chain maps, as described in sections 4 and 6,
- the text file `MAXFILT` whose contents are a single number which is the global default for the maximum homological degree (*filtration* in the Adams spectral sequence) through which the calculations are to run,
- the directory `F2`, which contains a precomputed resolution of $\mathbb{F}_2$ over $\mathcal{A}$ or $\mathcal{A}(2)$, respectively, through internal degrees 119 and 240, respectively. This latter allows users to calculate the action of $\mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$ on $\mathrm{Ext}_{\mathcal{A}}(M, \mathbb{F}_2)$ through this range, and serves as an example. The directory also contains some charts and tables which the software can compute,
- a symbolic link from `S0` to `F2` in `A`, and
- the directory `A2/C2`, which contains a precomputed resolution of H^*C2 over $\mathcal{A}(2)$ through internal degree 240, together with the Ext chart and some chain maps described in Section 8.

Of these, most users will need to interact primarily with the installation command `Install` (once for each of `A` and `A2`), and will repeatedly use the commands `newmodule`, `newmap`, `cocycle` and `make_all_cocycles`. The tools in the directory `samples` will be useful in creating module definition files, as described in section 4.

3. Notation for elements

Since we are computing minimal resolutions $C_* \longrightarrow M$, we have a natural isomorphism

$$\mathrm{Ext}^s_{\mathcal{A}}(M, \mathbb{F}_2) \cong \mathrm{Hom}_{\mathcal{A}}(C_s, \mathbb{F}_2).$$

We number the $\mathcal{A}$-generators of C_s sequentially starting with 0, and write them as s_0^*, s_1^*, ..., generically denoted s_g^*. This allows us to write the dual $\mathbb{F}_2$ basis elements of $\mathrm{Ext}^s_{\mathcal{A}}(M, \mathbb{F}_2)$ as s_0, s_1, ..., s_g, ... In plain text, rather than TEX, we write `s_g*` or `s_g`, respectively.

A typical element of C_2 might then be written

```
3
0 8 4 i(8)(2,2).
1 7 4 i(7)(4,1)(0,0,1).
3 1 1 i(1).
```

in the notation used by the programs described here. The first 3 indicates that this denotes a sum of three terms, namely

$$(Sq^8 + Sq^{(2,2)})2_0^* + (Sq^7 + Sq^{(4,1)} + Sq^{(0,0,1)})2_1^* + Sq^1 2_3^*.$$

Here, the $Sq^{(r_1,\ldots,r_k)}$ are Milnor basis elements, dual to $\xi_1^{r_1} \cdots \xi_k^{r_k}$. Each line following the 3 denotes a nonzero term in the sum in the format

```
g n d op
```

where `g` is the generator number, as in `s_g*`, the integer `n` is the degree of the Steenrod operation `op`, and `d` is the $\mathbb{F}_2$-dimension of the algebra in degree `n`. (Having this logically redundant value `d` available allows the input routines to allocate needed space without reference to the algebra over which we are working, and is a nice reminder to the user as well). The operation `op` can be expressed as a Milnor basis element, as above, or as a list of sequence numbers, as in

```
3
0 8 4 s0,2.
1 7 4 s0,1,3.
3 1 1 s0.
```

or as a bit string in hexadecimal, as in

```
3
0 8 4 xa0
1 7 4 xd0
3 1 1 x80
```

The sequence numbers are assigned to Milnor basis elements using grevlex, graded reverse lexicographic, ordering as described in a bit more detail in [8].

These formats are denoted '`x`' for hexadecimal, '`s`' for sequence numbers, or '`i`' for Milnor basis elements. There is a program `convert` described later which will convert one format to another. There is another format designation '`c`' for condensed, which uses the shortest notation, so that we could have an element written

```
5
0 35 58 s0,5,24,36.
1 34 55 xc950bc00240000
2 32 47 x90a004000000
3 28 35 s0,4,26.
5 4 2 x80
```

for example.

The module definition format, described in Section 4, describes a module by giving an $\mathbb{F}_2$ basis, with its elements numbered sequentially, starting with `0`. In order to use the same routines to read such elements, in contexts such as the definition of a finitely presented module or a cocycle, we will write elements of this $\mathbb{F}_2$-vector space as an element of the free $\mathcal{A}$-module which augments to it. That is, if generators `3`, `4`, `5`, and `6` all lie in the same degree of a module `M`, so that `3+4+6` is a homogeneous element of `M`, we might write this as

```
3
3 0 1 i(0).
4 0 1 i(0).
6 0 1 i(0).
```

or

```
3
3 0 1 x80
4 0 1 x80
6 0 1 x80
```

either of which denotes

$$Sq^0\,\mathtt{3} + Sq^0\,\mathtt{4} + Sq^0\,\mathtt{6}.$$

4. Creating modules

Each module over $\mathcal{A}$ or $\mathcal{A}(2)$ is assigned its own directory in `A` or `A2`, respectively. Our examples will be over $\mathcal{A}$, whose modules are kept in the directory `A`. The process is entirely analogous, but takes place in `A2`, for modules over $\mathcal{A}(2)$.

Two kinds of modules can be treated by the system. These are modules which are finite dimensional over $\mathbb{F}_2$, and modules which are finitely presented over $\mathcal{A}$ or $\mathcal{A}(2)$.

4.1. **Defining a finite module.** Modules which are finite dimensional over $\mathbb{F}_2$ are described by a `moddef`, or *module definition* file as follows. The header is

n
$d_0\ \ d_1\ \ \cdots\ \ d_{n-1}$

where n is the $\mathbb{F}_2$ dimension of the module, and d_0 through d_{n-1} are the degrees, *in non-decreasing order*, of a basis. (The utility `sortDef` will reorder the basis elements so that their degrees are in non-decreasing order and rewrite the rest of the module definition file accordingly.) The remainder of the module definition file describes the action of the Sq^i by lines of the form

$g\ \ i\ \ k\ \ g_1\ \ \cdots\ \ g_k$

where g and $g_1, \ldots, g_k$ are generator numbers (0 to $n-1$), and this line means that $Sq^i(g) = g_1 + \cdots + g_k$. For example, $M_1 = H^*C2$, the cohomology of the cofiber of $2 : S \to S$, has module definition file

```
2
0 1

0 1 1 1
```

while $M_2 = H^*C\eta$ has module definition file

```
2
0 2

0 2 1 1
```

and $M_1 \otimes M_1 = H^*C2 \times C2$ has module definition file

```
4
0 1 1 2

0 1 2 1 2
0 2 1 3

1 1 1 3

2 1 1 3
```

The blank lines are not required; it is only necessary that the integers defining the degrees and squaring operations be separated by white space, so that this last module definition file could be

```
4
0 1 1 2 0 1 2 1 2 0 2 1 3 1 1 1
3 2 1 1 3
```

but this would make it hard for a human to interact with.

N.B.: All nonzero Sq^i, $i > 0$, must be specified, even though the module is completely determined by just the Sq^{2^j}. If you enter just the Sq^{2^j}, the program `newconsistency` (described below) will help you find the remaining operations which must be added.

The directory `ext/A/samples` contains the following utilities.

- `makeP`, `makeCP`, and `makeHP` create module definition files for stunted real, complex and quaternionic projective spaces.
- `makeR` leaves out the 0-cell of the stunted real projective space, which gives the cohomology of the fiber of the Kahn-Priddy map when the bottom cell is in a negative degree.
- `consistency` and `newconsistency` check that the specified Steenrod operations are homogeneous and that the Adem relations are satisfied. The program `consistency` gives only the first term of each Adem relation which is violated, for a quick check, while `newconsistency` gives the full Adem relations.

- `dualizeDef` creates the module definition file for the Spanier-Whitehead dual of a module.
- `tensorDef` creates the module definition file for the tensor product of two modules.
- `collapse` and `truncate` create module definition files for the submodule containing all classes above a specified degree and the corresponding quotient module, respectively.
- `quotient` takes a more general quotient. There is no guarantee that the result it produces actually defines an $\mathcal{A}$-module, so has to be checked using `(new)consistency`.
- `sortDef` will rearrange the basis elements of a module definition so that their degrees are in non-decreasing order. It reports the permutation used.

In nearly all cases, the commands in this package give usage statements telling you the parameters required if invoked with the wrong number of arguments. You can use this to get more detail about any of them. For example,

```
ext/A/samples[1]: ./collapse

Usage: collapse <Def1> <bottom> <Def>
  will produce a module definition file named <Def>
  which defines the module obtained from <Def1> by
  taking the submodule containing all elements of degree
  greater than <bottom>, effectively collapsing to a point
  the classes up to and including this dimension.

ext/A/samples[2]: ./quotient

Usage: quotient <Def1> <exclusions> <Def>
  will produce a module definition file named <Def>
  which defines the module obtained from <Def1> by
  taking the quotient by all elements listed in
  the file <exclusions>.

  No attempt is made to check whether the result
  is a valid module definition, so it should be
  run through newconsistency.

ext/A/samples[3]: ./makeP

Usage: makeP <lo> <hi> [<file>]
  writes a Def file for stunted projective space with cells
  in dimensions <lo> to <hi> in <file> if specified, or Def
  if no file is specified.
```

4.2. **Installing a module.** Once you have a module definition file, say `M.def`, you prepare to calculate a minimal resolution for it by running the command `newmodule` in the directory `ext/A`. This will

(1) create a directory M,
(2) create files Diff.0 through Diff.smax in M to hold the resolution as it is computed,
(3) put links to the files in template into the module's directory, ext/A/M/,
(4) copy M.def to ext/A/M/Def, the location in which the resolution programs expect to find the definition of the module, and
(5) check the module definition for correctness.

If the file ext/A/MAXFILT exists, smax is set equal to its contents. As distributed, this is set to 40. You should change it according to your needs, For example, if you want a presentation of a finite module, you might replace 40 by 1. If the file ext/A/MAXFILT does not exist, the command newmodule will ask what value smax should be assigned.

For the following sample, assume that the definition is in ext/A/samples/M.def and you are in ext/A/.

```
ext/A/[4]: ./newmodule M samples/M.def
Setting the maximum filtration to 40
Making initial Diff files
Linking to executables in template directory
Moving module definition

Checking the module definition for correctness.


If any errors were reported, fix the definition before trying
to compute the minimal resolution. Note that the
program newconsistency will provide complete Adem relations
to simplify the task of correcting the definition.
```

Here we see that no errors were reported by consistency, so it is now possible to compute a minimal resolution, as in Section 5. Had there been errors in the definition, they would be reported as follows.

```
ext/A[4]: ./newmodule M samples/M.def
Setting the maximum filtration to 40
Making initial Diff files
Linking to executables in template directory
Moving module definition

Checking the module definition for correctness.

Degree error: Sq(1)(0) contains 3, but deg(0) = 2 while deg(3) = 4
Adem rel (Sq(1)Sq(1) + ... )(gen 0) =  4 , not zero.
Adem rel (Sq(2)Sq(2) + ... )(gen 0) =  7 , not zero.
Adem rel (Sq(1)Sq(2) + ... )(gen 1) =  7 , not zero.

If any errors were reported, fix the definition before trying
to compute the minimal resolution. Note that the
program newconsistency will provide complete Adem relations
to simplify the task of correcting the definition.
```

We see that the operation Sq^1 on generator 0 is inhomogeneous, containing a term of degree 4, rather than 3. We also see that several Adem relations were violated. Nonetheless, the directory M was created and it is now possible to edit the module definition file there. The program `newconsistency` will give the entire Adem relation, rather than just its first term, for each Adem relation which is violated.

```
ext/A[5]: cd M
ext/A[6]: ./newconsistency Def
Degree error: Sq(1)(0) contains 3, but deg(0) = 2 while deg(3) = 4
Adem rel (Sq(1)Sq(1) + ... )(gen 0) =  4 , not zero.
   Sq(1)Sq(1)  = 0.
Adem rel (Sq(2)Sq(2) + ... )(gen 0) =  7 , not zero.
   Sq(2)Sq(2) + Sq(3)Sq(1)  = 0.
Adem rel (Sq(1)Sq(2) + ... )(gen 1) =  7 , not zero.
   Sq(1)Sq(2) + Sq(3)Sq(0)  = 0.
```

You can then edit `Def` to fix the errors. If you do this, the erroneous file `ext/samples/M.def` will still exist, so you may prefer to `rm -r M`, edit the file `ext/samples/M.def`, and rerun `newmodule`. The directory `samples` contains a link to the executables `consistency` and `newconsistency` to make it easier to correct a module definition file there.

4.3. **Defining a finitely presented module.** Here are the steps to define a finitely presented $\mathcal{A}$-module.

(1) Create a module with the same connectivity using `newmodule`, as described above.
(2) In the module directory, remove `nextt` and link it to `nextt_fp`:

```
rm nextt
ln -s nextt_fp nextt
```

(3) Install the module's generators in `Diff.0` and the module's relations in `Diff.1`.

The easiest way to do the first step is to create a module definition file with 1 generator of degree `n`, where `n` is the connectivity. E.G., if the connectivity of the finitely presented module is 5, you might say

```
ext/A[1]: cat > S5F2.def
1
5
ext/A[2]: ./newmodule FPM S5F2.def
Setting the maximum filtration to 40
Making initial Diff files
Linking to executables in template directory
Moving module definition

Checking the module definition for correctness.


If any errors were reported, fix the definition before trying
to compute the minimal resolution. Note that the
```

```
program newconsistency will provide complete Adem relations
to simplify the task of correcting the definition.

ext/A[3]: head FPM/Diff.0
         0         4
ext/A[4]: head FPM/Diff.40
         0         4
```

Here, `S5F2.def` defines $\Sigma^5\mathbb{F}_2$, which is one dimensional over $\mathbb{F}_2$ with its single generator in degree 5.

The headers of the files `Diff.s`, $s = 0, 40$, show they initially contain 0 generators and are complete through degree 4, as is appropriate before starting to compute the resolution for a module whose connectivity is 5.

The last step, installing the presentation in `Diff.0` and `Diff.1` will be explained by an example. See Section 3 for more details about the format of elements of a free module, as in the files `Diff.s`.

Suppose our module is

$$H^*K(\mathbb{Z}) = \frac{\mathcal{A} \oplus \Sigma^3\mathcal{A}}{\left(\begin{pmatrix} Sq^1 \\ 0 \end{pmatrix}, \begin{pmatrix} Sq^2 \\ 0 \end{pmatrix}, \begin{pmatrix} Sq^4 \\ Sq^1 \end{pmatrix}, \begin{pmatrix} Sq^6 \\ Sq^{(0,1)} \end{pmatrix} \right)}$$

The connectivity of this is 0, so we could use `./newcommand HKZ F2/Def` to create a module directory `HKZ` with the correct connectivity.

The file `Diff.0` defining $\mathcal{A} \oplus \Sigma^3\mathcal{A}$ should be

```
        2      999


0
1
0 0 1 i(0).

3
1
1 0 1 i(0).
```

This actually defines the homomorphism $\mathcal{A} \oplus \Sigma^3\mathcal{A} \longrightarrow H^*K(\mathbb{Z})$ which takes the free $\mathcal{A}$ generators of `Diff.0` to distinct elements numbered 0 and 1. All that is important here is the domain of this homomorphism, which is the free $\mathcal{A}$-module on generators of degrees 0 and 3.

The file `Diff.1` defines the homomorphism

$$\Sigma\mathcal{A} \oplus \Sigma^2\mathcal{A} \oplus \Sigma^4\mathcal{A} \oplus \Sigma^6\mathcal{A} \longrightarrow \mathcal{A} \oplus \Sigma^3\mathcal{A}$$

which imposes the relations in $H^*K(\mathbb{Z})$. It can be written

```
        4      999


1
1
0 1 1 i(1).

2
```

```
1
0 2 1 i(2).

4
2
0 4 2 i(4).
1 1 1 i(1).

6
2
0 6 3 i(6).
1 3 2 i(0,1).
```

The headers in these files show that `Diff.0` and `Diff.1` have 2 and 4 generators over $\mathcal{A}$, respectively. The degree, 999, through which they are considered complete is simply a number large enough that it is effectively infinity. The images of the 4 elements in `Diff.1` are the relations between the generators of $H^*K(\mathbb{Z})$.

The version, `nextt_fp`, of the internal command `nextt`, which was put in place in step (2) above, will then compute `Diff.2`, `Diff.3`, ..., resolving the kernel of the map in `Diff.1`.

The text file `START_HERE` included in this distribution has another example and relevant details.

N.B., if your presentation is not minimal, the complex $\mathrm{Hom}_{\mathcal{A}}(\mathrm{Diff}.*, \mathbb{F}_2)$ may have nonzero differentials between homological degrees 0, 1 and 2. You will need to compute its homology in these degrees to get $\mathrm{Ext}_{\mathcal{A}}(M, \mathbb{F}_2)$.

5. Resolving modules

Once you have created the directory for your module, as in the preceding section, you can compute a minimal resolution through some finite internal degree, create the usual Adams spectral sequence charts, and create a TEX file of the action by the h_i as follows. Assume we are in the module's directory, `ext/A/XYZ` say, and that the connectivity of `XYZ` is 16.

Then we compute the resolution from internal degree 16 to internal degree 80 by

```
./dims 16 80 &
```

We use the ampersand at the end to put the process in the background so that we can issue other commands while it is running. On my current laptop, this took 1 minute and 54 seconds. When it has completed,

```
./report
```

will create files `Shape` and `himults` containing information about the $\mathbb{F}_2$-basis of the resolution and the action of the h_i on Ext. If you provide `report` with an argument, as in `./report summary`, it will write a brief text summary of the results in the named file (`summary` here). In either case

```
./chart 0 20 0 60 Shape himults XYZ.tex XYZ
```

will then create a TEX file `XYZ.tex` containing the Adams spectral sequence chart, with h_0, h_1 and h_2 multiplications shown. Apply your favorite TEX processor to this. (N.B.: for very large charts, the usual TEX processors may run out of memory. In this case, lualatex is a good option.)

If you wish to see the h_3 multiplications as well, there is a variant command `charth3` which will include them as dotted lines.

The file produced by `chart` has a very small wrapper around a tikzpicture, which you can easily edit and incorporate into other TeX documents.

If you wish to extend the resolution to degree 100, then run

```
./dims 81 100 &
```

wait until this has finished, and then

```
./report
./chart 0 20 0 80 Shape himults XYZ.tex XYZ
```

for example. If you had instead run

```
./dims 16 100 &
```

it would have taken a bit longer while it runs through degrees 16 to 80 again, but this would not cause any harm. Since the resolution was already exact in internal degrees up to 80, no changes would have been made in those degrees.

Note that if you do not rerun the command `report` after computing the additional degrees, the files `Shape` and `himults` would not be updated, so that the chart produced would still stop at internal degree 80. Also note that the file `himults` contains all the h_i multiples reported in the form of lines

```
s g s0 g0 i
```

which mean that s_g is a term in the product $h_i \cdot s_{0_{g_0}}$. For example, the lines

```
...
7 13 6 10 3
7 13 6 13 2
7 13 6 14 1
7 14 6 10 3
...
```

tell us that $h_3 \cdot 6_{10} = 7_{13} + 7_{14}$ and that $h_2 \cdot 6_{13} = h_1 \cdot 6_{14} = 7_{13}$ in the cohomology of $\mathbb{F}_2$.

Here is an actual run.

```
~/ext/A/XYZ[314]: ./dims 16 80 &
[1] 87024
~/ext/A/XYZ[315]: date
Wed Aug  3 11:33:03 EDT 2022
~/ext/A/XYZ[316]: ls -lt | head
total 190192
-rw-r--r--   1 rrb  staff       0 Aug  3 11:34 OIm_4.76
-rw-r--r--   1 rrb  staff  313228 Aug  3 11:34 Ker_4.76
-rw-r--r--   1 rrb  staff  741604 Aug  3 11:34 Im_4.76
-rw-r--r--   1 rrb  staff  693640 Aug  3 11:34 OIm_3.76
-rw-r--r--   1 rrb  staff   25045 Aug  3 11:34 Diff.3
-rw-r--r--   1 rrb  staff  280413 Aug  3 11:34 Ker_3.76
-rw-r--r--   1 rrb  staff  683475 Aug  3 11:34 Im_3.76
-rw-r--r--   1 rrb  staff  329804 Aug  3 11:34 OIm_2.76
-rw-r--r--   1 rrb  staff    6282 Aug  3 11:34 Diff.2
~/ext/A/XYZ[317]: date
```

```
Wed Aug  3 11:34:57 EDT 2022
[1]  + Done                          ./dims 16 80
~/ext/A/XYZ[318]: ./report
Discarding summary
~/ext/A/XYZ[319]: ./chart 0 20 0 60 Shape himults XYZ.tex XYZ
~/ext/A/XYZ[320]: open XYZ.tex
~/ext/A/XYZ[321]: ./vsumm

Usage:  vsumm <title>

  Reads Shape, Maxt and lines and prints a stem by stem summary of
  the resolution on stdout, in LaTex, using <title> as the title.

~/ext/A/XYZ[322]: ./vsumm XYZ > XYZhi.tex
~/ext/A/XYZ[323]: open XYZhi.tex
```

The first date command was issued immediately upon starting `dims`, and the second as soon as I noticed that it seemed to have stopped. These are how I made the estimate of 1 minute and 54 seconds. Since `dims` was running in the background, I was able to issue the command `date` while it was running. I also issued the command `ls -lt | head` in order to see how far it had gotten. The presence of the files `OIm_4.76`, `Ker_4.76` and `Im_4.76` at the top of the list shows that it was in the second half of working on bidegree $(s, t) = (4, 76)$.

Finally, at the end, I used the command `vsumm` to create a TeX document containing a stem-by-stem listing of the generators together with the h_i products.

The commands `open X.tex` above should be replaced by `pdflatex X.tex` and `evince X.pdf` or other commands which process TeX files and display the results.

5.1. **Parallelization and an easy fix for an easy mistake.** The command `./dims tlo thi` runs `./nextt t` for `t` from `tlo` to `thi`. The program `./nextt t` computes each bidegree `(s,t)` for `s` from `0` to `MAXFILT`. Before starting bidegree `(s,t)`, it checks that bidegrees `(s-1,t)` and `(s,t-1)` have been completed. If they have not, the program sleeps a bit and checks again. If you inadvertently invoke `./dims tlo thi` with `tlo` too large, the program will simply sit in such a loop waiting for internal degreees up to `tlo-1` to be completed. You can simply invoke `./dims tlower tlo-1` to compute the missing degrees.

This mistake is most likely to happen with modules with bottom cell in a negative degree, when it is easy to 'start' by saying something like `./dims 0 60`, forgetting that you need to start in the bottom degree of the module, not always in degree `0`. A quick look at the start of `Diff.0` or `Def` will be sufficient to remind you where to start:

```
ext/A/testit[1]: cat Diff.0
         0        -5
ext/A/testit[2]: ./dims -4 20 &
[1] 51251
...

ext/A/testit[5]: <CR>
[1]    Done                          ./dims -4 20
```

```
ext/A/testit[6]: cat Diff.0
         1         20
-4

1
0 0 1 x80
```

We start by observing that the `Diff.0` file was initially complete through dimension `-5` with no entries, so we should start computing the resolution in degree `-4`. After `dims -4 20` has finished, we see that `Diff.0` is complete through `t=20`.

This feature allows for parallelization by running `N` processes, each of which computes `./nextt i` for one residue class modulo `N`. This makes good sense if you have a machine with `N` processors. As an extreme version, you can simply run all the `nextt` at once, letting the majority of them sleep until their turn arrives. For example, after the above sequence we could use the following set of instructions (in the `tcsh` shell):

```
ext/A/testit[7]: set t=21
ext/A/testit[8]: while ($t < 41)
while? ./nextt $t &
while? set t=`expr 1 + $t`
while? end
[1] 64132
[2] 64134
[3] 64137
[4] 64139
[5] 64143
[6] 64147
[7] 64150
[8] 64158
[9] 64166
[10] 64179
[11] 64195
[12] 64215
[13] 64229
[14] 64242
[15] 64255
[16] 64273
[17] 64286
[18] 64298
[19] 64310
[20] 64321
ext/A/testit[12]: jobs
[1]    Done                          ./nextt 21
[2]  + Running                       ./nextt 22
[3]    Running                       ./nextt 23
[4]    Running                       ./nextt 24
[5]    Running                       ./nextt 25
[6]    Running                       ./nextt 26
```

```
[7]     Running                         ./nextt 27
[8]     Running                         ./nextt 28
[9]     Running                         ./nextt 29
[10]    Running                         ./nextt 30
[11]    Running                         ./nextt 31
[12]    Running                         ./nextt 32
[13]    Running                         ./nextt 33
[14]    Running                         ./nextt 34
[15]    Running                         ./nextt 35
[16]    Running                         ./nextt 36
[17]    Running                         ./nextt 37
[18]    Running                         ./nextt 38
[19]    Running                         ./nextt 39
[20]  - Running                         ./nextt 40
```

Caution: It is important not to start two processes which will try to write to the same files at the same time. In the setup above, only `nextt t` will try to write any of the `Im`, `OIm` or `Ker` files for internal degree `t`. Also, since `nextt t` waits to compute bidegree `(s,t)` until bidegree `(s,t-1)` is completed, each of the non-sleeping `nextt` processes will be working on a separate homological degree. Hence, there will be no contention in writing to `Diff.s`. If, however, you were to start two processes, both running `nextt t` for the same value of `t`, they could interact destructively.

6. Creating chain maps and computing products

Consult the sections "Products" and "Chain Maps" in the document [8] included here as `ext/doc/CohomA2.pdf` for additional discussion.

6.1. **Map definition files.** We compute products in Ext by composing chain maps. Suppose that $C_* \longrightarrow M$ and $D_* \longrightarrow N$ are (partial) resolutions of the modules M and N, and suppose that the classes $[x] \in \mathrm{Ext}_{\mathcal{A}}^{s_0,t_0}(N,P)$ and $[y] \in \mathrm{Ext}_{\mathcal{A}}^{s_1,t_1}(M,N)$ are represented by cocycles

$$x : D_{s_0} \longrightarrow \Sigma^{t_0}P \qquad \text{and} \qquad y : C_{s_1} \longrightarrow \Sigma^{t_1}N.$$

Then $\Sigma^{t_1}x \circ y_{s_0}$ is a cocycle representing the product xy, where $\{y_s\}_s$ is a chain map lifting y. The chain map with components $\Sigma^{t_1}x_s \circ y_{s+s_0}$ is a lift of this cocycle.

$$\begin{tikzcd}
M & C_0 \arrow[l] & \cdots \arrow[l] & C_{s_1} \arrow[l] \arrow[dl, "y"'] \arrow[d, "y_0"] & \cdots \arrow[l] & C_{s_0+s_1} \arrow[l] \arrow[d, "y_{s_0}"] & \cdots \arrow[l] & C_{s+s_0+s_1} \arrow[l] \arrow[d, "y_{s+s_0}"'] \\
 & & \Sigma^{t_1}N & \Sigma^{t_1}D_0 \arrow[l] & \cdots \arrow[l] & \Sigma^{t_1}D_{s_0} \arrow[l] \arrow[dl, "\Sigma^{t_1}x"'] \arrow[d, "\Sigma^{t_1}x_0"] & \cdots \arrow[l] & \Sigma^{t_1}D_{s+s_0} \arrow[l] \arrow[d, "\Sigma^{t_1}x_s"'] \\
 & & & & \Sigma^{t_0+t_1}P & \Sigma^{t_0+t_1}E_0 \arrow[l] & \cdots \arrow[l] & \Sigma^{t_0+t_1}E_s \arrow[l]
\end{tikzcd}$$

A cochain $x : C_s \longrightarrow \Sigma^t N$ is defined to the system by a *map definition file* of the form

```
s t M N x k
```

```
g1
j1
x1_1 0 1 x80
...
x1_j1 0 1 x80


...

gk
jk
xk_1 0 1 x80
...
xk_jk 0 1 x80
```

Here, `M` and `N` are the names of the directories containing the modules M and N and their resolutions. The map name, `x`, will be the name of a subdirectory of the domain `M`. The rest of the file tells the value of the cochain x on the k generators s_{g1} through s_{gk}. If a generator s_g is not listed, then $x(s_g) = 0$. The values, `x(gi) = xi_1 + ...  + xi_ji` are sums of the $\mathbb{F}_2$ generators numbered `xi_j` in the module definition file for N.

Once you have a map definition file, say `x.def`, in the main directory `ext/A`, execute

```
./newmap x.def
```

This will create a directory named after the map in the domain directory, and add the name of that map/directory to the file `maps` in that domain directory.

As a special case, if `s_g*` is an $\mathcal{A}$-module generator of internal degree `t` in the resolution of `M`, the program `cocycle` will create the map definition file for the dual cocycle $s_g : C_s \to \Sigma^t\mathbb{F}_2$, then apply `newmap` to it. Thus, all you need to say is, in `ext/A`,

```
./cocycle M s g
```

The *command* `cocycle` thus refers to this special case with values in $\mathbb{F}_2$, dual to a single basis element found by the `ext` code. This is needed sufficiently often that it is useful to have.

If you wish to compute the lifts of all cocycles, the command `make_all_cocycles` will create a shell script `makecocycle_<mod>.sh` which can be sourced to create all cocycles. First make certain that the `Shape` file is up to date in your module. Say the module name is `M` and you start in the directory `ext/A`. Then you should do the following:

```
cd M
./report
cd ..
./make_all_cocycles M
source makecocycles_M.sh
rm makecocycles_M.sh
```

This will install a map directory for every cocycle s_g in the resolution of M. The program `make_all_cocycles` does not take account of cocycles which have already been created, but the calls to `cocycle` in the shell script `makecocycles_M.sh` will detect the presence of any preexisting cocycles, and will skip them.

N.B., a map definition file can define any cochain, but only cocycles can be lifted. As a result, I often say *cocycle* when *cochain* would be more accurate. I have tried, in this document, to be careful about this distinction. If you define a cochain which is not a cocycle, you will get error messages saying that the lift cannot be computed.

6.2. **Computing the chain map.**

Once maps have been installed using `newmap` and/or `cocycle`,

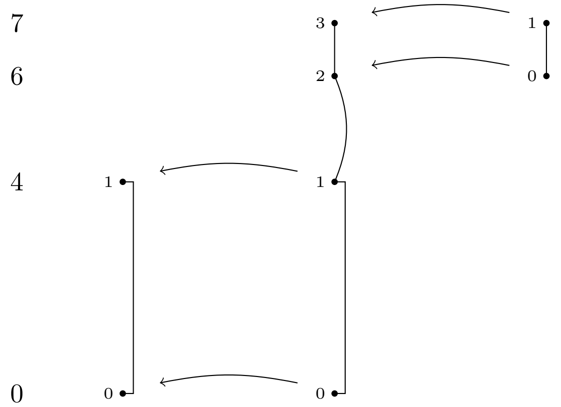
```
cd M
./dolifts slo shi maps
./collect maps all
```

will compute lifts for all the cocycles listed in the file `maps` and collect the resulting maps induced on Ext in the file named `all`.

See the sections titled "Products" and "Chain Maps" of [8] for more details. We give one more set of examples here, consisting of the three chain maps which induce the long exact sequence in Ext induced by the cofiber sequence

$$C\nu \xrightarrow{i} bo_1 \xrightarrow{p} \Sigma^6 C2 \xrightarrow{e} \Sigma C\nu$$

In cohomology we have a short exact sequence, and a lift of e to the first stage of an Adams resolution of $C\nu$ will induce the homomorphism of Adams spectral sequences detecting e.



$$\begin{tikzcd}
H^*C\nu & H^*bo_1 \arrow[l, two heads, "i^*"'] & H^*\Sigma^6C2 \arrow[l, tail, "p^*"'] \\
H^*C\nu \arrow[u, equal] & C_0 \arrow[l, two heads] \arrow[u] & C_1 \arrow[l, tail] \arrow[u, "e"']
\end{tikzcd}$$

We will name the three modules involved `Cnu`, `bo1` and `C2`, respectively. The modules C_0 and C_1 are the start of the resolution of $H^*C\nu$ (contained in the files `Cnu/Diff.0` and `Cnu/Diff.1`). The homomorphism e is the extension cocycle for the short exact sequence (i^*, p^*). The chain map lifting it will give the boundary homomorphism

$$\begin{tikzcd}
\mathrm{Ext}^{s,t}(H^*C2,\mathbb{F}_2) \arrow[r, "e_*"] \arrow[d, equal] & \mathrm{Ext}^{s+1,t+6}(H^*C\nu,\mathbb{F}_2)) \arrow[d, equal] \\
\mathrm{Ext}^{s,t+6}(H^*\Sigma^6C2,\mathbb{F}_2)) \arrow[r, "e_*"] & \mathrm{Ext}^{s+1,t+6}(H^*C\nu,\mathbb{F}_2))
\end{tikzcd}$$

The map definition file for i^* is

```
0 0 bo1 Cnu i 1

0
1
0 0 1 x80
```

since i^* sends the unique generator of `bo1/Diff.0` to generator `0` in the module `Cnu`.

The map definition file for p^* is

```
0 -6 C2 bo1 p 1

0
1
2 0 1 x80
```

since p^* sends the unique generator of `C2/Diff.0` to generator `2` in the module `bo1`.

The map definition file for e^* is

```
1 6 Cnu C2 e 1

2
1
0 0 1 x80
```

since `Cnu/Diff.1` shows that $C_1 = \Sigma\mathcal{A} \oplus \Sigma^2\mathcal{A} \oplus \Sigma^6\mathcal{A} \oplus \Sigma^8\mathcal{A} \oplus \cdots$, and all the generators go to 0 except for the degree 6 generator (generator number `2`), which goes to $Sq^6(0_0^*)$ in `Cnu/Diff.0`, and hence to generator `2` in `bo1`. This lifts to generator `0` in `C2`.

Suppose these three map definition files are named `i.def`, `p.def`, and `e.def`. The following sequence of commands will compute the chain maps lifting these cocycles through as much of the resolutions as has been computed thus far. (*Note that the command* `./checkmap s` *in a map directory will report on any unmapped generators in homological degrees up to* `s`.) Assume that these map definition files are located in `ext/A` and that you are currently in this directory.

```
./newmap i.def
./newmap p.def
./newmap e.def
cd bo1
./dolifts 0 40 maps
./collect maps all_i
cd ../C2
./dolifts 0 40 maps
./collect maps all_p
cd ../Cnu
```

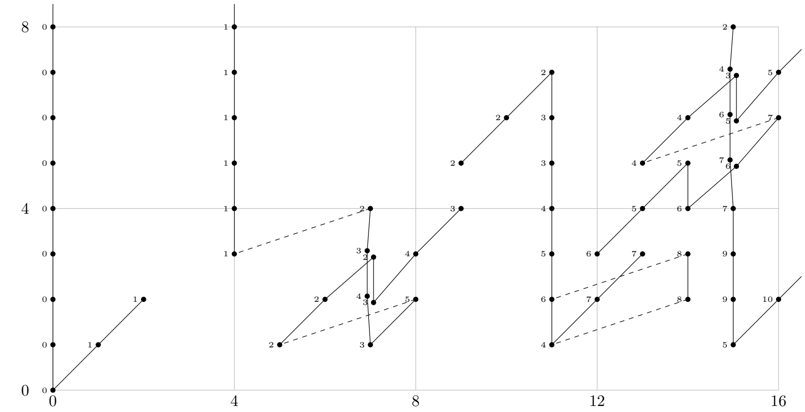


FIGURE 1. $\mathrm{Ext}_{\mathcal{A}}^{s,n+s}(H^*C\nu, \mathbb{F}_2)$, $0 \le n \le 16$, $0 \le s \le 8$

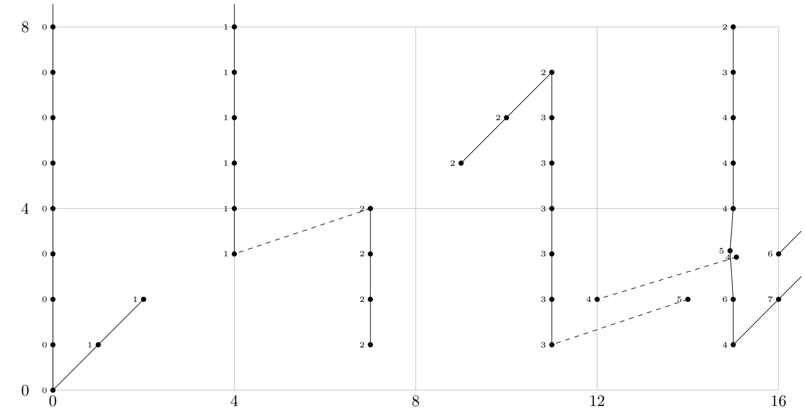


FIGURE 2. $\mathrm{Ext}_{\mathcal{A}}^{s,n+s}(H^*bo_1, \mathbb{F}_2)$, $0 \le n \le 16$, $0 \le s \le 8$

```
./dolifts 0 40 maps
./collect maps all_e
```

The file `bo1/all_i` contains

```
  0    0  (  0    0    Cnu)  i

  1    0  (  1    0    Cnu)  i

  1    1  (  1    1    Cnu)  i

  1    2  (  1    3    Cnu)  i

  1    3  (  1    4    Cnu)  i
```

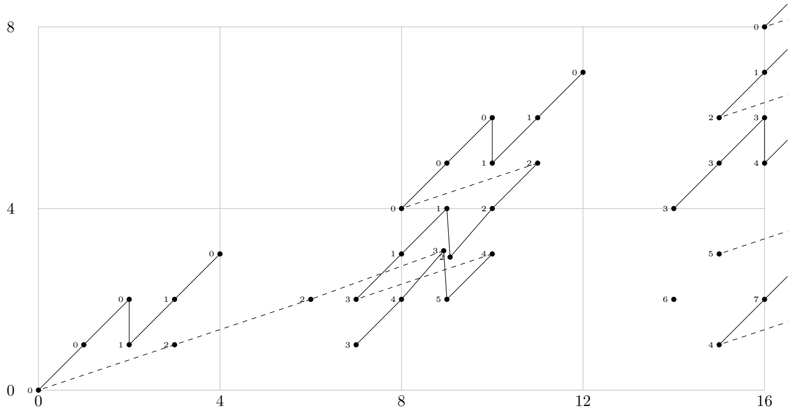


FIGURE 3. $\mathrm{Ext}_{\mathcal{A}}^{s,n+s}(H^*C2, \mathbb{F}_2)$, $0 \leq n \leq 16$, $0 \leq s \leq 8$

```
...

  5   17  (  5   21    Cnu)  i
  5   17  (  5   22    Cnu)  i
...
```

which shows that the homomorphism $i_* : \mathrm{Ext}(H^*C\nu, \mathbb{F}_2)) \longrightarrow \mathrm{Ext}(H^*bo_1, \mathbb{F}_2))$ takes values $i_*(0_0) = 0_0$, $i_*(1_0) = 1_0$, $i_*(1_1) = 1_1$, $i_*(1_3) = 1_2$, and, since 1_2 in `Cnu` is not listed, $i_*(1_2) = 0$. Further down, we see that $i_*(5_{21}) = i_*(5_{22}) = 5_{17}$, which follows from the fact that the chain map induced by `i` sends `5_17*` to `5_21* + 5_22* + decomposables`. (This can be seen in `ext/A/bo1/i/Map` as the value following the pair `5 17`.)

The file `Cnu/all_e` contains

```
  1    2  (  0    0     C2)  e

  2    2  (  1    0     C2)  e

  2    3  (  1    1     C2)  e

  2    5  (  1    2     C2)  e

  2    7  (  1    3     C2)  e

  2   12  (  1    4     C2)  e

  2   17  (  1    5     C2)  e

  2   23  (  1    6     C2)  e
  3    2  (  2    0     C2)  e

  3    4  (  2    1     C2)  e

  3    6  (  2    3     C2)  e
...
```

which shows that the homomorphism $e_* : \mathrm{Ext}^{s,t}(H^*C2, \mathbb{F}_2)) \longrightarrow \mathrm{Ext}^{s+1,t+6}(H^*C\nu, \mathbb{F}_2))$ takes values $e_*(0_0) = 1_2$, $e_*(1_0) = 2_2$, $e_*(1_1) = 2_3$, $e_*(1_2) = 2_5$, and, since 2_2 in C2 is not listed, $e_*(2_2) = 0$.

N.B., the information contained in the files produced by collect is the indecomposable quotient of the chain map, and the induced homomorphism on Ext is that induced on cocycles, i.e., on homomorphisms from the resolution to $\mathbb{F}_2$. This dualization is left to the user, since it is easy, but should not be overlooked.

In general, the value of the chain map on a given generator can only be computed if the resolution of the codomain has been extended far enough to include its image. (Occasionally, startmap can determine that the chain map can send a generator to 0 before this.) Before deciding that a class goes to 0, it may help to run checkmap in the map's directory, to be certain that its image is 0 rather than simply not computed yet. It will appear in the Map file, and hence will not appear in the output of checkmap, if it has been computed, even if its image is 0 (or simply decomposable). The summary produced by collect conveys this fact by omission, hence this caution.

6.3. **Missing generators.** There is a utility, missing, which reads the Shape file and the all file and reports any s_g which do not appear as the first entry of a line in the all file. If the all file has been created from a list of cocycles s_g, this is a rough and ready way to find cocycles s_g which are not in the $\mathrm{Ext}(\mathbb{F}_2, \mathbb{F}_2)$-submodule of $\mathrm{Ext}(M, \mathbb{F}_2)$ generated by those cocycles.

For example, in ext/A2/C2, suppose we have computed the chain map lifting 0_0 and that we wish to determine the $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$-submodule of $\mathrm{Ext}_{\mathcal{A}(2)}(M, \mathbb{F}_2)$ spanned by 0_0. We create a file ext/A2/C2/maps0 which contains just this one chain map. Then, if we collect the action on ext/A2/C2/maps0 and check what is missing, we see

```
ext/A2/C2[2]: ./collect maps0 all0
ext/A2/C2[3]: ./missing all0 | head

Missing generators:
  1   1
  2   1
  2   3
  3   0
  3   2
  4   2
  5   1
  5   3
```

From this, we see that 1_1, 2_1, 2_3, 3_0, ... are not in the submodule of $\mathrm{Ext}_{\mathcal{A}(2)}(C2, \mathbb{F}_2)$ generated by 0_0. If we add the first of these, 1_1, to maps0 to

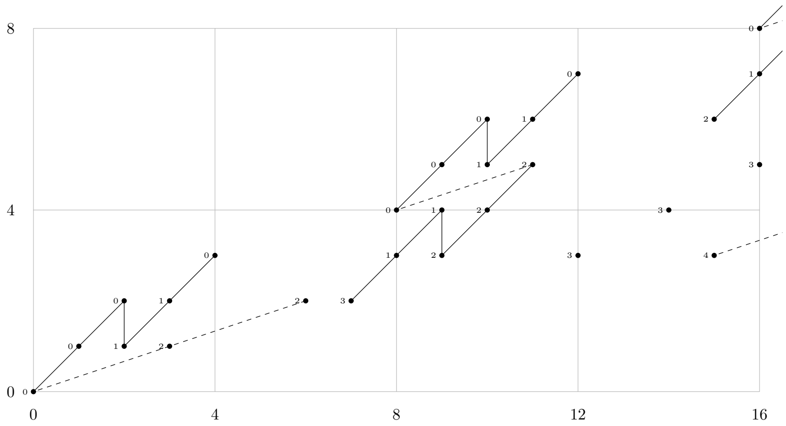


FIGURE 4. $\mathrm{Ext}_{\mathcal{A}(2)}^{s,n+s}(H^*C2,\mathbb{F}_2)$, $0 \le n \le 16$, $0 \le s \le 8$

get `maps1` with both maps in it, do the lifts for `1_1` if we haven't already, and then repeat the process above, we find

```
ext/A2/C2[4]: ./collect maps1 all1
ext/A2/C2[5]: ./missing all1 | head

Missing generators:
  2   3
  3   2
  5   8
  6   2
  6   3
  6   4
  6  10
  7   2
```

Evidently, adding `1_1` to our list of generators has now made `2_1` decomposable, and a glance at the chart, Figure 4, for this module shows this is so. In addition, `3_0`, `4_2`, `5_1` and `5_3` are in the submodule generated by `0_0` and `1_1`. The element `2_3` is the only filtration 2 generator still missing. If we now create `maps2` by adding `2_3` to `maps1`, and do the lifts for `2_3`, we find

```
ext/A2/C2[6]: ./collect maps2 all2
ext/A2/C2[7]: ./missing all2 | head

Missing generators:
  3   2
  5   8
  6   3
  6  10
  7   2
  7  10
  7  12
  8   7
```

Continuing in this way, we find 13 indecomposables, so that $\mathrm{Ext}_{\mathcal{A}(2)}(C2, \mathbb{F}_2)$ is generated as a module over $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$ by these 13 cocycles. With those 13 in `maps`, we find

```
ext/A2/C2[22]: cat maps
0_0
1_1
2_3
3_2
5_8
6_3
6_10
7_10
7_12
8_7
8_12
8_14
10_12
ext/A2/C2[23]: ./collect maps allmaps
ext/A2/C2[24]: ./missing allmaps

No missing generators
```

This almost shows our list is complete. N.B., it is necessary to carefully check bidegrees of dimension more than 1 over $\mathbb{F}_2$ to be certain that this is so. The quick textual check done by `missing` would take a relation like $4_8 \cdot 2_3 = 6_7 + 6_8$, for example, which would be shown in the file produced by `collect` as

```
  6    7  (  4    8     F2)  2_3
  6    7  (  5   11     F2)  1_1
  6    7  (  6   10     F2)  0_0

  6    8  (  1    1     F2)  5_8
  6    8  (  4    8     F2)  2_3
  6    8  (  6   10     F2)  0_0
```

and conclude that both 6_7 and 6_8 have been found. If the 4_8-multiple $6_7 + 6_8$ were the only product in this bidegree, then the sum would be decomposable, but each individual term would be indecomposable. Here, the product $5_{11} \cdot 1_1 = 6_7$ does show that 6_7 and, hence, 6_8 are decomposable.

Using this process we are able to find finite presentations of $\mathrm{Ext}_{\mathcal{A}(2)}(M, \mathbb{F}_2)$ over $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$ when M is finite. Over $\mathcal{A}$ this is only possible in a finite range of degrees.

6.4. **Parallelization.** If multiple instances of the program `dolifts` are run at the same time, they will interact destructively. Some of the processes may remove files that others still need, and worse, multiple processes may write to files simultaneously, producing meaningless results. A version of `dolifts` called `pdolifts` handles the parallization to avoid these difficulties. It is invoked by

```
./pdolifts slo shi maps n
```

in order to do the lifts in n parallel processes. The code makes no attempt to balance the load between these processes. It simply distributes the maps listed in `maps` round-robin into n groups. Some balancing can be achieved by sorting the `maps` file by homological and internal degree, or by the size of their current `Map` file before invoking `pdolifts`, either from largest to smallest or from smallest to largest. If the maps file was created by running `make_all_cocycles`, the maps file will be sorted by homological, then internal degree.

7. Toda brackets

From the chain map lifting a cocycle x, it is possible to compute all brackets of the form $\langle h_i, s_g, x\rangle$. This is discussed carefully in Section 10, "Toda brackets" in [8]. (See also Section 14 there on the use of brackets to define operators in the Adams spectral sequence.)

We will give one example here. The brackets ending in the cocycle `i` above can be found in the file `ext/A/bo1/i/brackets.sym`, which begins

```
2_3 in < h2, 4, i >
2_4 in < h2, 5, i >
2_6 in < h3, 3, i >
2_6 in < h3, 4, i >
2_7 in < h4, 0, i >
2_7 in < h3, 5, i >
```

Since $i : H^*bo_1 \longrightarrow H^*C\nu$, the middle entry in each bracket is a cocycle s_g in $\mathrm{Ext}(H^*C\nu, \mathbb{F}_2)$. The left hand `hi` is $h_i \in \mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$. We can determine s because the homological degree of the bracket is 1 less than the sum of the homological degrees of the terms. In these examples, we have $2 = 1 + s + 0 - 1$, so that the middle terms are 2_g for the indicated g.

The lines indicate possible brackets: they are not necessarily defined. The first one is not, since $i_*(2_4) = 2_2 \neq 0$. The third bracket is not defined because $h_3 \cdot 2_3 = 3_8 \neq 0$ in $\mathrm{Ext}_{\mathcal{A}}(H^*C\nu, \mathbb{F}_2)$. The second bracket is defined, though, and we see that $2_4 = \langle h_2, 2_5, i\rangle \in \mathrm{Ext}_{\mathcal{A}}^{2,14}(H^*bo_1, \mathbb{F}_2)$, where $2_5 \in \mathrm{Ext}_{\mathcal{A}}^{2,10}(H^*C\nu, \mathbb{F}_2)$. There is no indeterminacy: the image of i_* and the image of multiplication by h_2 are both 0 in this bidegree. By Moss' Theorem and the fact that there are no crossing differentials here, this shows us that the unique nonzero homotopy class $\{2_4\} \in \pi_{12}(bo_1)$ is the bracket of the three maps

$$S^{11} \xrightarrow{\nu} S^8 \xrightarrow{\eta\sigma} C\nu \xrightarrow{i} bo_1$$

since 2_5 detects the homotopy class $\eta\sigma$ on the bottom cell of $C\nu$. That is, the diagram

$$\begin{tikzcd}
 & & & \Sigma^8 C\nu \arrow[rr] \arrow[dr, dashed] & & S^{12} \arrow[dr, dashed, "\{2_4\}"] & \\
S^{11} \arrow[rr, "\nu"] \arrow[dr, dashed, "\{2_4\}"'] & & S^8 \arrow[ur] \arrow[rr, "\eta\sigma"] \arrow[dr, dashed] & & C\nu \arrow[rr, "i"] & & bo_1 \\
 & \Sigma^{-1} bo_1 \arrow[rr] & & \Sigma^5 C2 \arrow[ur, "e"'] & & &
\end{tikzcd}$$

commutes.

8. Precalculated contents

There is a trade-off between including precalculated resolutions and chain maps, to save time, and the size of the distribution. The three resolutions included here take up about 43 MB, the majority of the space required. Since calculations out to internal degree 80 or 100 take only a few minutes on modern machines, including any more seemed unnecessary.

8.1. $\mathbb{F}_2$ **over** $\mathcal{A}$**.** The directory `ext/A/F2` contains a resolution of $\mathbb{F}_2$ over $\mathcal{A}$ for the range $0 \leq s \leq 60$, $0 \leq t \leq 119$. This is complete through the 80 stem. It also contains chain maps for all 29 indecomposable cohomology classes in the range $0 \leq s \leq 4$, $0 \leq t \leq 119$. These 29 chain maps are listed in `ext/A/F2/maps` and products with the corresponding cocycles can be found in `ext/A/F2/all`.

8.2. $\mathbb{F}_2$ **over** $\mathcal{A}(2)$**.** The directory `ext/A2/F2` contains a resolution of $\mathbb{F}_2$ over $\mathcal{A}(2)$ for the range $0 \leq s \leq 100$, $0 \leq t \leq 240$. This is effectively a complete calculation, since $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$ is generated by classes within this range satisfying known relations. See [6].

The only precalculated chain map with domain `A2/F2` is the one induced by projection onto the top cell of the Moore spectrum, $C2 \to S^1$. This induces the cocycle C2top : $\mathrm{Diff}.0 \to \mathbb{F}_2 \to \Sigma^{-1} H^* C2$. The map definition file `A2/F2/C2top/Def` contains

```
0 -1 F2 C2 C2top 1

0


1
1 0 1 x80
```

The file `C2topcell` was produced by `./collect maps C2topcell`. It determines the homomorphism $\mathrm{Ext}_{\mathcal{A}(2)}(H^*C2, \mathbb{F}_2) \to \mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$ induced by this projection map. (Here, `maps` contains only the one chain map, `C2top`).

8.3. $C2$ **over** $\mathcal{A}(2)$**.** The directory `ext/A2/C2` contains a resolution of H^*C2 over $\mathcal{A}(2)$ for the range $0 \leq s \leq 120$, $0 \leq t \leq 240$. This is effectively a complete calculation, since $\mathrm{Ext}_{\mathcal{A}(2)}(H^*C2, \mathbb{F}_2)$ is generated by 13 classes, found in `A2/C2/maps`, over the coefficient ring $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$. See [6]. Chain maps have been calculated for each of these 13 cocycles, and the results collected in the file `ext/A2/C2/all`. We see for example,

```
  5    2  (  2    1      F2)  3_2
  5    2  (  4    2      F2)  1_1
  5    2  (  5    3      F2)  0_0
```

showing that $5_2 = h_1^2 \cdot 3_2 = h_1 c_0 \cdot 1_1 = h_2 w_1 \cdot 0_0$ in $\mathrm{Ext}_{\mathcal{A}(2)}(H^*C2, \mathbb{F}_2)$.

8.4. **A long exact sequence.** These data suffice to give the long exact sequence in $\mathrm{Ext}_{\mathcal{A}(2)}(-, \mathbb{F}_2)$ induced by the cofiber sequence $S \to C2 \to S^1$, since

(1) the inclusion of the bottom cell induces the cocycle `0_0`,
(2) the projection onto the top cell induces the cocycle `C2top`,
(3) the boundary map is multiplication by h_0, which we have calculated without the need to compute a chain map.

8.5. **Chain maps and edge effects.** The chain map `C2top` nicely illustrates a couple of issues involving chain maps related to the fact that we are only working with initial segments of resolutions. If we go to `ext/A2/F2/C2top` and run `./checkmap 40` we find

```
ext/A2/F2/C2top[1]: ./checkmap 40 | head
  36  206
  37  212
  37  213
  38  203
  38  204
  39  206
  40  218
  40  219
```

Each of these classes, 36_{206}, et cetera, lies in degree $t = 240$ for $\mathbb{F}_2$, hence in degree $t = 241$ for $\Sigma\mathbb{F}_2$. Since the resolution for H^*C2 has only been calculated through $t = 240$, we cannot expect to determine the value of `C2top` on such classes. However, looking at the chart for $\mathrm{Ext}^{36,240}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$, we see that it is two dimensional, spanned by 36_{206} and 36_{207}. Checkmap did not report that 36_{207} was missing, and when we look at `C2top/Map`, we see that 36_{207} is mapped to 0. This is already known, because, when the program `./startmap 36` calculated `C2top(d(36_207))`, the result was 0, so that there was no need to run `./liftmap` to see that the lift `C2top(36_207)` can be chosen to be zero.

8.6. **Editing the charts.** The charts in each of these three directories have been edited to include a dashed line showing the edge of the calculation. In addition, the chart in `ext/A2/C2` has been edited to show the 13 generators of $\mathrm{Ext}_{\mathcal{A}(2)}(H^*C2, \mathbb{F}_2)$ over $\mathrm{Ext}_{\mathcal{A}(2)}(\mathbb{F}_2, \mathbb{F}_2)$ as larger red dots. A few Adams differentials which follow easily by naturality have been included in `ext/A2/C2/himults`. These are the 4 lines

```
3 3 5 2 0
4 4 7 1 0
6 3 8 1 0
7 3 9 2 0
```

at the end. If you rerun the program `report`, these lines will be lost. They are included here to show how to add differentials to the TEX file produced by `chart`.

9. Bug fixes and improvements

Here are the changes in `ext.1.9.5` from the last released version, `ext.1.9.3`.

9.1. **Bug in `addgen.c`.** This bug does not affect anyone who uses the code as distributed. However, if you modified the programs `nextt` and/or `nextt_fp` to save the `OIm_s.t` files generated during the computation of a resolution, this bug would affect chain maps into directories where this was done. It would be advisable to run `dmeqmd` or `ccmap` to check the validity of such chain maps.

The bug occurs in the stage when new generators are added to the resolution to make it exact. In each bidegree (s, t), we first compute $d(a \cdot s_g^*)$ for each Milnor basis element a and each existing generator s_g^*, in order, as described in Section 7, "A Canonical Basis", of [8]. We row reduce these as we proceed, producing a list of pairs (dx_i, x_i) such that the leading terms of the dx_i are disjoint. Then, for each c in our basis for the kernel in bidegree $(s-1, t)$, we row reduce c, producing $c_{\mathrm{red}} = c - \sum \alpha_i dx_i$, $\alpha_i \in \mathbb{F}_2$, so that either $c_{\mathrm{red}} = 0$ or c_{red} has a leading term disjoint from those of the dx_i. In the latter case, we add a new generator s_g^* in bidegree s with $d(s_g^*) = c$. We could instead set $d(s_g^*) = c_{\mathrm{red}}$, but experience has shown that the product structure is closer to monomial with $d(s_g^*) = c$, and this has been the algorithm in use since around 2000. The first difference in the bases chosen for $\mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$ by the two algorithms occurs in bidegree $(s, t) = (9, 32)$. The older algorithm (used in the 1997 document [3]) gave the basis $\langle h_1 P d_0 + h_0^2 i, h_0^2 i \rangle$, while the new algorithm gives $\langle h_1 P d_0, h_0^2 i \rangle$.

With the old algorithm, we would then simply insert $(c_{\mathrm{red}}, s_g^*)$ into our list of pairs (dx_i, x_i), at the position corresponding to the leading term of dx_i, and proceed to the next kernel basis element. With the new algorithm, we must instead insert $(c_{\mathrm{red}}, s_g^* - \sum \alpha_i x_i) = (c - \sum \alpha_i dx_i, s_g^* - \sum \alpha_i x_i)$ into the list of pairs (dx_i, x_i). Unfortunately, when the new algorithm was adopted, the incorrect pair $(c_{\mathrm{red}}, s_g^*)$ was used. This is the fix in `ext.1.9.5`: we now insert the correct pair $(c_{\mathrm{red}}, s_g^* - \sum \alpha_i x_i) = (c - \sum \alpha_i dx_i, s_g^* - \sum \alpha_i x_i)$ into the list of pairs (dx_i, x_i).

The bug did not affect the exactness of the resolution, but if this faulty `OIm_s.t` file, containing $(c_{\mathrm{red}}, s_g^*)$ was saved and later used to compute chain maps, they could be in error, since $d(s_g^*) \neq c_{\mathrm{red}}$.

If the scripts, as distributed, were used, then the faulty `OIm_s.t` files would be discarded without being used. The `OIm_s.t` files which are created by the programs which compute chain maps do not have the flaw described above, since they work on bidegrees in which the resolution is already exact, so that new generators do not need to be added.

In addition to carefully inspecting the new code in `addgen.c`, the program which contained this error, to see that it is now doing the correct thing, I took account of Donald Knuth's adage

> "Beware of bugs in the above code; I have only proved it correct,
> not tried it."

and compared the `OIm_s.t` files produced by the new `addgen.c` and those produced after the resolution has been computed, as when chain maps are being calculated. They were identical, character by character, for $t \leq 100$ in a resolution of $\mathbb{F}_2$. I have also run extensive tests using `dmeqmd` or `ccmap` to verify that the chain maps the program has computed really are chain maps.

9.2. **Bug in io.c.** I changed `io.c` to use XOR rather than OR when reading elements formatted as type `i`. The use of OR was based on the expectation that the terms in

$$i(r_{1,1}, ..., r_{1,k_1})(r_{2,1}, ..., r_{2,k_2})...(r_{n,1}, ..., r_{n,k_n}).$$

would be distinct so that OR or XOR of the corresponding bits would produce the same results. This is true of the ouput of routines in io.c, so makes no difference if all files read by `ext` are generated by the `ext` code, but it might be false if a user generates entries by hand or by another program. This fix makes, e.g.,

```
1
0 3 1 i(0,1)(0,1)(3).
```

produce the same result as

```
1
0 3 1 i(3).
```

rather than the same result as

```
1
0 3 1 i(0,1)(3).
```

Clearly this is better.

9.3. **Bug in `summarize.c`.** Previous versions left out h_7 multiples.

9.4. **Improvements.**

(1) The program `pdolifts` has been added to parallelize the calculation of chain maps. See Subsection 6.4.

(2) The program `collect` has incorporated John Rognes' improved sorting of the output. There is also a better usage statement when it is called without arguments.

(3) There are numerous fixes to `chart.c`, many contributed by John Rognes, which improve the tikz code and improve the appearance of the charts. These are improvements to margins, clipping, and the spacing of labels. Comments are added to the tikz code so that the code generating the dots is labelled by their `s_g` name, and and the code generating the lines is labelled by their source and target names. This facilitates adding differentials or other modifications to the tikz chart, since it allows searching for `s_g` to find the code involving s_g. I also added `charth3`, which adds dotted lines indicating h_3-multiples.

(4) A utility `ext/A2/induceup` takes an $\mathcal{A}(2)$-resolution of an $\mathcal{A}(2)$-module M in `ext/A2/` and applies $\mathcal{A} \otimes_{\mathcal{A}(2)} -$ to get the induced up $\mathcal{A}$-resolution of $\mathcal{A} \otimes_{\mathcal{A}(2)} M$, which is placed in `ext/A/`.

```
Usage: induceup <A2-module> <A-module>

  - takes an A2 resolution in <A2-module> and converts
    it into an A-resolution in ../A/<A-module>.    All
    it does is to replace each A(2) by a copy of A,
    leaving the differentials the same.
    It should be invoked in the directory A2, after
    computing the A2 resolution of M.
NOTE: the Def file for the new module will not be an actual
 Def file.   Instead it will tell that it was induced up.
```

(5) A utility, `ext/A/bin/ccmap`, checks that a chain map really is a chain map by computing the difference $dm - md$. It isn't perfect in dealing with missing generators, so it is best to use checkmap first.

(6) Fixes to `startmap` and `fstartmap` to handle `s1 == 0` better, add missing cast to int, and (`fstartmap.c`) move fclose into the block with the accompanying fopen to get rid of an excess fclose error. (These fixes were in the unreleased 1.9.4)

(7) `liftmap` (called by `dolifts`) is now a little bit more informative when handling the `s==0` case, to help in debugging malformed map/Def files. Precisely, `liftmap` now prints the internal degrees of the generators of the codomain when a chain map is being lifted from the module to the 0th stage of its resolution (for sanity/error checking purposes). An obsolete comment from pre moddef days has been removed.

(8) The new `make_all_cocycles` in `ext/A` and `ext/A2` creates all cocycles of the form `s_g` for a specified module. See Section 6.

(9) The output of `convert` now has slightly better formatting: no blank spaces at the ends of lines, and a reasonable number of blank lines between elements and blocks.

(10) There is a new utility `prune.c` for pruning `Diff` files and similar. It is not included in `Install`, so if you want to use it you have to compile it yourself: in `ext/A` say

```
gcc -o DIR/prune -Isrc obj/*.o src/prune.c
```

where `DIR` is the directory you want it to end up in.

9.5. Minor cleanup.

(1) `util.c`: fixed typos in comments.

(2) The program `vsummarize` has been removed because it is superceded by `vsumm`. If for some reason it is wanted, it is still in `ext.1.9.3` and earlier.

(3) The script `seeres` has been removed, since the optional output of `report` has the same information in a more readable format. It is still available in older versions if wanted.

10. Summary of programs

This section contains a list of the files a user might use in each of the directories.

`ext/A` or `ext/A2`:

- Install
- Clean
- newmodule
- MAXFILT
- newmap
- cocycle
- make_all_cocycles
- samples
- F2
- S0
- induceup (`ext/A2` only)
- C2 (`ext/A2` only)

`ext/A/samples`:

- consistency
- newconsistency
- tensorDef
- dualizeDef
- quotient
- truncate
- collapse
- sortDef
- makeP
- makeCP
- makeHP
- makeR
- Qi

`ext/A/M` **for a module** `M`**:**

- Def
- consistency, newconsistency
- MAXFILT (optional)
- Diff.s, for $s = 0, \ldots,$ `MAXFILT`
- dims
- report
- chart, charth3
- vsumm
- convert
- dolifts
- collect
- missing
- extend_dims
- Shape, himults, lines

`ext/A/M/m` **for a chain map m defined on the resolution of** `M`**:**

- checkmap
- ../../bin/ccmap

For a more complete listing of the programs see the file `ext/doc/Organization`. There are still a few items not listed here or there which are useful in special circumstances (adem, fixedmul, fstartmap, . . .). Feel free to play with them to see what they do.

Appendix A. File formats

In the following sections we describe the format of each of the data files used by `ext`. Recall from Section 3 that elements of free modules are written as follows:

```
k
g1 n1 d1 op1
...
gk nk dk opk
```

This represents the sum of `k` terms, `op1 * g1 + ...  + opk * gk`. Each `gi` is an integer representing a basis element of the free module, numbered from `0` to `N-1`, where `N` is the $\mathcal{A}$-dimension (resp., $\mathcal{A}(2)$-dimension) of the free module. Each opi is a Steenrod operation of degree ni, and the $\mathbb{F}_2$-dimension of the algebra ($\mathcal{A}$ or $\mathcal{A}(2)$) in degree ni is di.

The operations opi can be written in any of three formats.

- **internal:** The letter 'i' followed by any number of Milnor basis elements (r1,...,rj), followed by a period.
- **sequence numbers:** The letter 's' followed by the sequence numbers of the Milnor basis elements in the sum, ordered by graded reverse lexicographic order, numbered starting at 0, separated by commas and followed by a period.
- **hexadecimal:** The letter 'x' followed by a bit string in hexadecimal representing a whole number of bytes sufficient to hold the bitstring.

For example, the term $(Sq^9 + Sq^{(6,1)} + Sq^{(2,0,1)}) \cdot g$ could be written as any of

```
g 9 5 i(9)(6,1)(2,0,1).
```

```
g 9 5 s0,1,4.
```

or

```
g 9 5 xc8
```

since grevlex ordering in degree 9 of $\mathcal{A}$ is

$$(9) < (6,1) < (3,2) < (0,3) < (2,0,1).$$

A term $Sq^n \cdot g$ could therefore be written

```
g n d i(n).
```

```
g n d s0.
```

or

```
g n d x80...0
```

where d is the $\mathbb{F}_2$-dimension of the degree n part of the algebra ($\mathcal{A}$ or $\mathcal{A}(2)$) and the hexadecimal number 80...0 has the smallest even number of hexadecimal digits which contains d bits.

Several of the data files described in this appendix can be converted to any of these formats by the program convert. It also accepts a format designation 'c', meaning 'condensed', which will write the element in the format using the fewest characters. Each of these files has a header consisting of a certain number of integers, followed by a sequence of *blocks*. Each block has a *prefix* consisting of some number of integers, followed by some number of elements. The usage statement for convert tells these sizes.

```
Usage: convert infile outfile hdr_size prefix_size block_size [type]
where type, if included, is one of c, n, x, s, i, or b (default = c)
Types are c=condensed, x=hex, s=sequence numbers, i=Milnor basis,
b=binary, n=numbered and Milnor basis.

Size parameters:
File type    hdr_size prefix_size block_size
---------    -------- ----------- ----------
Diff.s       2        1           1
Map          0        2           1
OIm_s.t      1        4           2
```

```
Im_s.t       1        2           2
Ker_s.t      1        2           1
```

For example, we might say

```
./convert Diff.2 hDiff.2 2 1 1 i
```

or

```
./convert Map hMap 0 2 1 i
```

to convert `Diff.2` or `Map` to the 'humanly readable' files `hDiff.2` or `hMap` respectively.

To avoid duplication, in what follows, we will describe the files in `ext/A/`. The information applies equally well to `ext/A2/`.

Let `M` stand for a module and let `x` and `y` stand for cocycles and associated chain maps with domain `M`.

A.1. **Module Definition file `ext/A/M/Def`.** Modules which are finite dimensional over $\mathbb{F}_2$ are described by a *module definition* file as follows. The header is

n
$d_0 \;\; d_1 \;\; \cdots \;\; d_{n-1}$

where n is the $\mathbb{F}_2$ dimension of the module, and d_0 through d_{n-1} are the degrees, *in non-decreasing order*, of elements of a basis. The utility `sortDef` will reorder the basis elements to put them in non-decreasing order, and rewrite the rest of the module definition file accordingly. The remainder of the module definition file describes the action of the Sq^i, $i > 0$, by lines of the form

$g \;\; i \;\; k \;\; g_1 \;\; \cdots \;\; g_k$

where g and $g_1, \ldots, g_k$ are generator numbers (0 to $n-1$), and this line means that $Sq^i(g) = g_1 + \cdots + g_k$.

A.2. **Differential file `ext/A/M/Diff.s`.** The file `Diff.s` contains the generators in homological degree `s` of the resolution together with the value of the differential on them.

Each file `Diff.s` has a 20 character header. The first 10 characters contains the number of elements in the file. The second 10 characters contains the internal degree through which the resolution has been calculated in homological degree `s`. The remainder of the file consists of blocks containing the degree of each generator and its image under the differential. Precisely, if `Diff.s` contains

```
         n        D

d0
E0

d1
E1

...

d{n-1}
E{n-1}
```

then the header tells us that, at this stage of the calculation, homological degree `s` has $\mathcal{A}$-basis s_0, s_1, ..., s_{n-1}, and this is complete in degrees up to and including $t = D$. It says that the internal degree of s_i is `di`, and that $d(s_i)$ is `Ei`. The elements `Ei` are written in the format described in the introduction to this appendix.

A.3. **Shape file** `ext/A/M/Shape`**.** Produced by `report`, the Shape file consists of integers separated by white space as follows.

```
M
n0 n1 ... nM
t00 t01 ... t0n0
t10 t11 ... t1n1
...
tM0 tM1 ... tMnM
```

These give

(1) the maximum homological degree (filtration) `M`,
(2) the $\mathcal{A}$-dimension `ns` of `ext/A/M/Diff.s`, and
(3) the internal degree `tsg` of each basis element s_g.

This information specifies exactly where the dots are located in the Adams chart.

A.4. h_i **multiples** `ext/A/M/himults`**.** The initial version of this file is produced by `report`, and shows which dots should be connected by lines representing h_i-multiples. The user can add entries, intended to represent differentials. The file consists of a sequence of lines

```
s0 g0 s1 g1 i
```

If `s0 > s1` this represents an h_i from `s1_g1` to `s0_g0`. The entries produced by `report` all have this form.

If the user adds an entry with `s0 < s1`, this is treated as a differential from `s0_g0` to `s1_g1`. In these, the value of `i` is ignored.

A.5. **Map Definition file** `ext/A/M/x/Def`**.** Suppose that

$$\cdots \to C_2 \to C_1 \to M \to 0$$

is the minimal resolution of M calculated by `ext`. A cochain $x : C_s \longrightarrow \Sigma^t N$ is defined to the system by a *map definition file* of the form

```
s t M N x k

g1
j1
x1_1 0 1 x80
...
x1_j1 0 1 x80



...

gk
jk
xk_1 0 1 x80
...
```

```
xk_jk 0 1 x80
```

Here, `M` and `N` are the names of the directories containing the modules M and N and their resolutions. The map name, `x`, will be the name of a subdirectory of the domain `M`. The rest of the file tells the value of the cocycle x on the k generators s_{g1} through s_{gk}. If a generator s_g is not listed, then $x(s_g^*) = 0$. The values, `x(gi) = xi_1 + ... + xi_ji` are sums of the $\mathbb{F}_2$ generators numbered `xi_1`, …, `xi_ji` in the module definition file for N.

These elements of N,

```
j
x1 0 1 x80
...
xj 0 1 x80
```

are written as elements in the image of the section $N \to \mathcal{A} \otimes N$ given by the unit $\mathbb{F}_2 \to \mathcal{A}$ and are to be interpreted as the elements of N obtained from them by applying the action $\mu : \mathcal{A} \otimes N \to N$ or the augmentation $\epsilon \otimes 1 : \mathcal{A} \otimes N \to N$.

A.6. **Map file `ext/A/M/x/Map`.** The map file consists of blocks

```
s g
E
```

which mean that $x(s_g^*) =$ `E`. The element `E` is written in the format described in the introduction to this appendix. If `E` $= 0$ this can be shortened to

```
s g 0
```

The blocks do not have to be in any particular order.

A.7. **List of maps `ext/A/M/maps`.** This is a text file containing the names of directories holding chain maps with domain `M`. The programs `newmap` and `cocycle` add the chain maps they create to the default file `maps`. To focus attention on specific maps, you can create other lists of chain maps, naming them (nearly) anything you like, and use those as the arguments to `dolifts` and `collect`.

The default, and recommended, format is one chain map per line, but all that matters is that the map names are separated by white space. Order does not matter.

A.8. **Products and induced maps, `ext/A/M/all`.** Suppose that $C_* \longrightarrow M$ is a resolution of the module M, that $D_* \longrightarrow N$ is a resolution of the module N, and suppose that $y : C_{s_1} \longrightarrow \Sigma^{t_1} N$ is a cocycle whose lift to a chain map has been computed in `ext/A/M/y/`. Running the command

```
./collect maps1 all1
```

in `ext/A/M/` (assuming that `y` is listed in the file `ext/A/M/maps1`) will create a file named `all1` containing entries of the form

```
  s   g   (  s0   g0   N)  y
```

Here $s = s_0 + s_1$, and this line in `all1` says that $y_{s_0}(s_g^*) = {s_0}_{g_0}^* + \cdots$.

$$\begin{tikzcd}
M & C_0 \arrow[l] & \cdots \arrow[l] & C_{s_1} \arrow[l] \arrow[dl, "y"'] \arrow[d, "y_0"] & \cdots \arrow[l] & C_{s_0+s_1} \arrow[l] \arrow[d, "y_{s_0}"] & s_g^* \arrow[d, mapsto] \\
 & & \Sigma^{t_1} N & \Sigma^{t_1} D_0 \arrow[l] & \cdots \arrow[l] & \Sigma^{t_1} D_{s_0} \arrow[l] & {s_0}_{g_0}^* + \cdots
\end{tikzcd}$$

Dually, this says that

$$y^* : \mathrm{Ext}^{s_0,t_0}(N, \mathbb{F}_2) \longrightarrow \mathrm{Ext}^{s_0+s_1,t_0+t_1}(M, \mathbb{F}_2)$$

satisfies $y^*({s_0}_{g_0}) = s_g + \cdots$. If $y = {s_1}_{g_1}$, this says that ${s_0}_{g_0} \cdot {s_1}_{g_1} = s_g + \cdots$.

A.9. **Brackets `ext/A/M/x/brackets.sym`.** When `dolifts` finishes computing the part of the chain map lifting a cocycle $x : C_{s_1} \to \Sigma^{t_1} N$ that it has been asked to compute, it extracts a file `ext/A/M/x/brackets.sym` from which deductions about Toda brackets can be made. The entries have the form

```
s_g in < hi, g0, x >
```

meaning that the bracket $\langle h_i, {s_0}_{g_0}, x\rangle$, if defined, contains an element $s_g + \cdots \in \mathrm{Ext}(M, \mathbb{F}_2)$. Here, $s = s_0 + s_1$. More precisely, the bracket, if defined, contains an element which is the sum of all such s_g.

A.10. **Image file `ext/A/M/Im_s.t`.** You will not need to look at these *image files* unles you are getting deep into the weeds. The discussion of the algorithm in the Section "A Canonical Basis" in [8] gives useful detail for understanding the following brief description. The file `Im_s.t` is created by either `genimker s t` or `genim s t`.

An `Im` file has a 10 character header containing either `0`, indicating that it is still being created, or `1`, meaning that it is complete. This is followed by blocks of the form

```
g k
DX
X
```

where `DX` and `X` are elements with $d(\mathtt{X}) = \mathtt{DX}$. The pair $(\mathtt{DX}, \mathtt{X})$ was found by row reducing $d(op_k \cdot s_g)$ until its leading term was distinct from all the leading terms seen before this. Here, op_k is the k^{th} Milnor basis element in grevlex ordering of the degree in question. Hence, the `DX` in `Im_s.t` form a basis for the image of the differential $d : C_{s,t} \to C_{s-1,t}$ in the resolution of the module under consideration and the `X` are their lifts to $C_{s,t}$.

A.11. **Kernel file `ext/A/M/Ker_s.t`.** The file `Ker_s.t` is created by `genimker` at the same time as `Im_s.t`, and contains those terms such that $d(op_k \cdot s_g)$ row reduced to `0`, so that the corresponding `X` form a basis for the kernel of $d : C_{s,t} \to C_{s-1,t}$.

The format is similar. There is a 10 character header indicating "in progress" (0) or "complete" (1), followed by blocks

```
g k
X
```

A.12. **Ordered image file `ext/A/M/OIm_s.t`.** This is created by `addgen s t` during calculation of the resolution, or later by `orderim` when it is needed in order to compute lifts of chain maps. In the former case, it consists of the terms in `Im_s.t` together with additional terms created by adding new generators to $C_{s,t}$ to make the image equal to the kernel. In the latter case, it is simply a variant of `Im_s.t`, since the new generators have already been added. It has no header, and consists of blocks

```
lg lk g k
DX
X
```

in which the entries `g`, `k`, `DX` and `X` are the same as corresponding entries in `Im_s.t`, while `lg` and `lk` are the leading generator and leading operation, respectively, in `DX`. The file is ordered: a block with prefix `lg1 lk1 g1 k1` comes before a block with prefix `lg2 lk2 g2 k2` iff `lg1 < lg2` or `lg1 == lg2` and `lk1 < lk2`.

Appendix B. A sample run

The following instructions will install the system into a direcctory `195test`, calculate enough of the resolution to create Figure 1, and calculate enough chain maps that every class in that chart is an $\mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$-multiple of at least one of them. A new user is encouraged to execute these commands and examine the results along the way, as a quick introduction to the use of this code. From start to finish, the calculation in this Appendix took about 6 minutes on my laptop. Naturally, if you stop to look at the results along the way, your elapsed time will be commensurately greater.

B.1. **Installing the program.** Create a directory, untar the program into it and run the Install script. (Start in a directory containing `ext.1.9.5.tar`.)

```
mkdir 195test
cd 195test
tar xf ../ext.1.9.5.tar
cd A2
./Install
cd ../A
./Install
```

B.2. **Installing and resolving the cofiber of ν.** Create the module `Cnu` and compute the resolution through internal degree 24, to include all of Figure 1. Create the TeX file and pdf chart. (You should be in the directory `195test/A` now.)

```
./newmodule Cnu samples/Cnu.def
cd Cnu
./dims 0 24 &
./report
./chart 0 8 0 16 Shape himults Cnu.tex Cnu
pdflatex Cnu.tex
open Cnu.pdf
```

B.3. **Create a generating set of chain maps.** Create cocycles, and use `missing` to find cocycles that are not $\mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$-multiples of earlier cocycles, in order to get a generating set. (You should be in the directory `195test/A/Cnu` now.)

```
cd ..
./cocycle Cnu 0 0
cd Cnu
./dolifts 0 40 maps
./collect maps all
cat all
./missing all | head
```

```
cd ..
./cocycle Cnu 1 2
./cocycle Cnu 1 4
cd Cnu
./dolifts 0 39 maps
rm all
./collect maps all
./missing all | head
cd ..
./cocycle Cnu 2 3
./cocycle Cnu 3 1
./cocycle Cnu 3 6
cd Cnu
./dolifts 0 38 maps
rm all
./collect maps all
./missing all | head
cd ..
./cocycle Cnu 5 4
./cocycle Cnu 6 5
cd Cnu
./dolifts 0 35 maps
rm all
./collect maps all
./missing all | head
```

This last call to `missing` should result in the message

```
No missing generators
```

Note that calls to `dolifts` above take account of the homological degree of the maps being lifted. For example, when lifting `5_4` and `6_5`, we run

```
./dolifts 0 35
```

since `5_4` would require homological degrees above 40 to map into homological degrees above 35.

B.4. **Conclusions.** The last call to `missing` will show that the 8 cocycles we computed suffice to generate $\mathrm{Ext}_{\mathcal{A}}(H^*C\nu, \mathbb{F}_2)$ as an $\mathrm{Ext}_{\mathcal{A}}(\mathbb{F}_2, \mathbb{F}_2)$-module in the range we have calculated. Examination of the file `all` reveals relations not evident in the chart. For example, we find

```
  3    8   (  1    3     F2)  2_3
  3    8   (  2    2     F2)  1_4
  3    8   (  3    5     F2)  0_0
```

showing that $3_8 = h_3 \cdot 2_3 = h_0 h_2 \cdot 1_4 = h_0 h_3^2 \cdot 0_0$. Similarly,

```
  4    5   (  1    1     F2)  3_6
  4    5   (  3    3     F2)  1_2
```

shows that $4_5 = h_1 \cdot 3_6 = c_0 \cdot 1_2$.

References

[1] Robert R. Bruner, *Calculation of large Ext modules*, Computers in geometry and topology (Chicago, IL, 1986), Lecture Notes in Pure and Appl. Math., vol. 114, Dekker, New York, 1989, pp. 79–104.

[2] Robert R. Bruner, Ext *in the nineties*, Algebraic topology (Oaxtepec, 1991), Contemp. Math., vol. 146, Amer. Math. Soc., Providence, RI, 1993, pp. 71–90, DOI 10.1090/conm/146/01216.

[3] Robert R. Bruner, *The cohomology of the mod 2 Steenrod algebra: A computer calculation*, Research Reports, vol. 37, Wayne State University, 1997.

[4] Robert R. Bruner, *Some root invariants and Steenrod operations in* $\mathrm{Ext}_A(F_2, F_2)$, Homotopy theory via algebraic geometry and group representations (Evanston, IL, 1997), Contemp. Math., vol. 220, Amer. Math. Soc., Providence, RI, 1998, pp. 27–33, DOI 10.1090/conm/220/03092.

[5] Robert Bruner, Christian Nassau, and Sean Tilson, *Steenrod operations and A-module extensions.* arXiv:1909.03117v3.

[6] Robert R. Bruner and John Rognes, *The Adams Spectral Sequence for Topological Modular Forms*, Mathematical Surveys and Monographs, vol. 253, American Mathematical Society, Providence, RI, 2021.

[7] Robert R. Bruner and John Rognes, *The cohomology of the mod 2 Steenrod algebra* (2021), `https://doi.org/10.11582/2021.00077`. [Dataset]. Norstore.

[8] Robert R. Bruner and John Rognes, *The cohomology of the mod 2 Steenrod algebra.* arXiv:2109.13117v2 [math.AT].

Department of Mathematics, Wayne State University, USA
Email address: `robert.bruner@wayne.edu`