A new interface between GAP and Singular: libsing

Max Horn

Joint work with Mohamed Barakat, Frank Lübeck, Oleksandr Motsak, Max Neunhöffer, Hans Schönemann.

August 25, 2014

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN

Overview

The protagonists

- "GAP is a system for computational discrete algebra, with particular emphasis on Computational Group Theory.
- "Singular is a computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory. Singular provides highly efficient core algorithms, a multitude of advanced algorithms in the above fields, [...]"
- libsing provides an efficient bridge between these two systems.

Related work

- GAP package “singular” by Marco Costantini and Willem de Graaf
  - Last regular update in 2006, based on Singular 2
  - Uses “screen scraping” $\rightsquigarrow$ fragile; inefficient
  - No generic way to transfer complex data structures.
  - No easy access to complete functionality of Singular.
- Packages “IO_ForHomalg” & “RingsForHomalg by homalg project
  - Supports Singular 3 and 4
  - Based on IO package, but in the end also does “screen scraping”
  - Avoids data conversions where possible $\rightsquigarrow$ efficient at what it does
  - Does not attempt to give access to complete functionality of Singular “out of the box”.
- Sage includes both GAP and Singular, could be used as “interface”

Features

- libsing is a GAP package, loaded by the GAP interpreter, but also links against Singular 4 (compiled as a shared library).
- libsing consists of a part written in C++, and a part written in the GAP programming language.
- libsing grants access to the complete functionality of Singular from the high-level GAP programming language. With it you may ...
  - ... access Singular objects via GAP wrapper objects;
  - ... convert between Singular and GAP objects;
  - ... access Singular C++ kernel functions;
  - ... use Singular interpreter intrinsics;
  - ... use Singular libraries;

  all from within GAP.

Short example: Singular version

The following example is taken from the Singular manual, specifically from the entry for `stdfglm`, a library command for computing a Gröbner basis.

```
> ring r = 0,(x,y,z),lp;
> ideal i = y3+x2,x2y+x2,x3-x2,z4-x2-y;
> stdfglm(i);
_[1]=z12
_[2]=yz4-z8
_[3]=y2+y-z8-z4
_[4]=xy-xz4-y+z4
_[5]=x2+y-z4
```

Short example: Straight conversion to GAP

The following is a straight conversion to GAP code using the low-level access to Singular interpreter intrinsics and library functions.

```
gap> r := SI_ring(0,["x","y","z"],[["lp",3]]);
<singular ring>
gap> i := SI_ideal(r,"y3+x2,x2y+x2,x3-x2,z4-x2-y");
<singular ideal (mutable), 4 gens of deg <= 4>
gap> SIL_stdfglm(i);
<singular ideal (mutable), 5 gens of deg <= 12>
gap> Display(last);
z12,
yz4-z8,
y2+y-z8-z4,
xy-xz4-y+z4,
x2+y-z4
```

Short example: Better GAP version

Or we can do without those strings and use a more "GAP-style" way of entering the polynomials:

```
gap> r := SI_ring(0,["x","y","z"],[["lp",3]]);;
gap> AssignGeneratorVariables(r);
#I  Assigned the global variables [ x, y, z ]
gap> i := SI_ideal([y^3 + x^2, x^2*y + x^2,
>                    x^3 - x^2, z^4 - x^2 - y]);
<singular ideal (mutable), 4 gens of deg <= 4>
gap> SIL_stdfglm(i);
gap> Display(last);
z12,
yz4-z8,
y2+y-z8-z4,
xy-xz4-y+z4,
x2+y-z4
```

Troubled data exchange

- GAP and Singular deal with many similar kinds of data: machine integers, multiprecision integers, lists, matrices, polynomials, etc.
- ... but they represent them quite differently.
- They also use radically different memory management schemes:
- GAP: generational moving garbage collector
  - memory does not have to be explicitly released;
  - the exact location of an object in memory can change.
- Singular: traditional memory manager plus reference counting
  - memory must be freed explicitly;
  - objects live in a fixed position during their life time.
- Need to bridge these differences with as little overhead as possible.

Wrapping objects

- Singular objects to be used from GAP are put into wrapper objects.
- Singular objects stay in a fixed place, the GAP wrapper may move.
- Benefit from the garbage collector housekeeping: if the wrapper is garbage collected, we free the wrapped object.
- Singular has a notion of an "active ring", implicitly used for computations. libsing hides this concept from GAP and the user.

Wrapping objects II

There are three kinds of wrappers, consisting of two or fours machine words (32 or 64 bit), respectively. Namely for . . .

ring free objects:

Attribute bits & type
C++ pointer to Singular object

ring dependant objects:

Attribute bits & type
C++ pointer to Singular object
GAP reference to wrapper for ring
C++ pointer to Singular ring

rings:

Attribute bits & type
C++ pointer to Singular object
GAP reference to wrapper for ring's zero
GAP reference to wrapper for ring's one

There can also be an additional word for the extended attributes.

Converting data

- (Un)wrapping objects is fast, has low memory overhead.
- But sometimes one needs to convert data, e.g.: given some GAP data, convert it to Singular, perform heavy computations, convert the result back to GAP.
- Machine ints are converted directly.

```
gap> SI_int(3);
3
```

- Multi precisions ints: GAP uses low-level, Singular high-level GMP interface $\implies$ conversion is necessary, requires copying data.

```
gap> x := SI_bigint(3^20);
<singular bigint:3486784401>
gap> _SI_Intbigint(x);
3486784401
```

Converting data II

- Containers (intvecs, intmats, lists, etc.) are converted recursively.

```
gap> v := SI_intvec([1,2,3]);
<singular intvec:[ 1, 2, 3 ]>
gap> _SI_Plistintvec(v);
[ 1, 2, 3 ]
```

- Rings, their elements, ideals, ..., are not (automatically) converted, (exception: rationals, $\mathbb{Z}/p\mathbb{Z}$)

```
gap> r := SI_ring(0,["x","y","z"],[["lp",3]]);
<singular ring>
```

- Instead of converting e.g. polynomials, the goal is to implement high level GAP APIs for such objects (e.g. querying the degree, coefficients, etc.).

Low level: Calling into the Singular kernel

- Wrappers for select C / C++ functions from the Singular kernel.
- For each supported function, we copy its declaration from the Singular kernel, augmenting it with some "hints":

```
poly pp_Mult_nn(poly p, number n, const ring r);
poly p_Mult_nn(DESTROYS poly p, number n, const ring r);
```

- When compiling libsing (via `make`), this is read by a code generator, which produces C wrappers accessible from GAP:

```
_SI_pp_Mult_nn
_SI_p_Mult_nn
```

- Right now only a small set of functions is mapped, but it is very easy to extend this.

Middle level: Using interpreter intrinsics

- "Interpreter intrinsics": functions provided by the Singular interpreter, but implemented in C++ (such as `betti`, `std`, `ncols`).
- Take different data types as input as the kernel functions.
- Singular interpreter contains a table describing all intrinsics (including names, parameter types; overloading possible)
- When compiling libsing, this table is read and appropriate wrappers are created as GAP functions

```
BindGlobal("SI_degree",
  function(a)
    return _SI_CallFunc1(285,a);
  end );
```

Middle level: Using Singular libraries

- Functions and libraries written in the Singular language are usable.

```
gap> SIL_submat;
Error, Variable: 'SIL_submat' must have a value
not in any function at line 44 of *stdin*
gap> SI_LIB("matrix.lib");
gap> s := SI_ring(0,["a","b"]);;
gap> m := SI_matrix(s,2,2,"a,b,ab,1");
<singular matrix (mutable):
a, b,
ab,1 >
gap> y := SIL_submat(m,SI_intvec([1,2]),SI_intvec([2]));
<singular matrix (mutable):
b,
1 >
```

- Currently can not use GAP functions from Singular.

High level: Dressing it up as GAP objects

- Finally, we also intend to provide high-level GAP wrappers for Singular functionality.
- For example, operators like +, -, * are overloaded.

```
gap> SI_bigint(3^25) + SI_bigint(7^36);
<singular bigint:265173084585965347262631199104 4>
```

- This is still a lot of work if one wants to cover everything.