Using FLINT's generics

Fredrik Johansson

2025-01-28
2025 FLINT workshop
École polytechnique, Palaiseau, France

What's in FLINT

Basic documentation: `https://flintlib.org/doc/gr.html`

Generic algorithms and data structures:
`gr_vec`
`gr_mat`
`gr_poly`
`gr_mpoly`
`gr_series`, `gr_series_mod` (semi-private)
`gr_generic` - basic algorithms (e.g. binary exponentiation)
`gr_special` - special functions

Context objects

A ring (or other structure) is defined by a `gr_ctx_t` which contains:

- `sizeof_elem`
- `which_ring` - e.g. `GR_CTX_FMPQ`
- Method table (generic fallbacks + overloads)
- Ring-specific data (currently 6 words to allow storing small data like an `nmod_t` or an `slong prec` inline; may use a pointer to a larger context object)

Initializing a context object is cheap ($O(1)$ cycles for simple types).

Note that `sizeof_elem` can be set on context initialization. This feature is currently used by `mpn_mod` ($\mathbb{Z}/n\mathbb{Z}$) and `nfloat`.

Not used (but supported in principle): dynamic method tables.

Compatibility

We can mix gr with ordinary FLINT types and methods:

```
gr_ctx_t ctx;
gr_ctx_init_fmpz(ZZ);

fmpz_t c;
fmpz_init(c);
fmpz_set_ui(c, 3);

gr_pow_ui(c, c, 100, ZZ);
```

Importantly, this also works for vectors, polynomials and matrices (some exceptions: `nmod_poly`, `nmod_mat`, `fmpq_poly`).

Creating generic elements

There is no `gr_t`. We must allocate `sizeof_elem` (or $n\times$ `sizeof_elem` for arrays) at runtime.

Fast temporary allocation (may use stack; avoid in loop bodies):

```
GR_TMP_INIT3(x, y, z, ctx);
GR_INIT_VEC(vec, n, ctx);
...
GR_TMP_CLEAR3(x, y, z, ctx);
GR_CLEAR_VEC(vec, n, ctx);
```

Persistent allocation (always mallocs):

```
x = gr_heap_init(ctx);
vec = gr_heap_init_vec(n, ctx);
...
gr_heap_clear(x, ctx);
gr_heap_clear_vec(vec, n, ctx);
```

Type stability

Elements cannot escape the ring! We have to convert explicitly.

```
// res has type (ctx)
gr_set_other(res, y, y_ctx, ctx);

// res and x have type (ctx)
gr_mul_other(res, x, y, y_ctx, ctx);
```

Warning: conversions work in basic cases but you may run into some issues.

Handling errors

Error flags:

- GR_DOMAIN (divide by zero, incompatible matrices, ...)
- GR_UNABLE (not enough memory, missing algorithm, undecidable equality...)

```
int status = GR_SUCCESS;
status |= gr_mul(x, a, b, ctx);
status |= gr_div(x, x, c, ctx);
return status;
```

Handling booleans

```
truth_t cond = gr_is_zero(x, ctx);

if (cond == T_TRUE)
    ...
else if (cond == T_FALSE)
    ...
else // (cond == T_UNKNOWN)
    ...
```

Enclosure semantics

Rings have enclosure semantics for inexact elements.

- $[3.14 \pm 0.01] = \pi$ gives `T_UNKNOWN`
- $[3.23 \pm 0.01] = \pi$ gives `T_FALSE`
- $[5 \pm 0] = 5$ gives `T_TRUE`

Another example: we distinguish between two kinds of power series (`series`, `series_mod`).

- In $R[[x]]$ with precision $O(x^3)$: $(2 - 3x + x^3) = (2 - 3x)$ gives `T_UNKNOWN`
- In $R[[x]]/\langle x^3 \rangle$: $(2 - 3x + x^3) = (2 - 3x)$ gives `T_TRUE`

Implementing rings

Methods required for basic functionality:

- ctx_init, ctx_clear, ctx_write, init, clear, swap, randtest, write, zero, one, equal, set, set_si, set_ui, set_fmpz, neg, add, sub, mul

Optional: fast vector operations

- vec_init, vec_clear, vec_swap, vec_zero, vec_set, vec_neg, vec_add, vec_sub, vec_mul_scalar_ui, vec_addmul_scalar_ui, vec_dot, vec_dot_rev, ...

Optional: fast algorithms

- poly_mullow, matrix_mul

Optional: more features and fine-tuning

- inv, div, sqrt, cmp, ctx_is_field, set_other, exp, poly_gcd, ...

Testing rings and algorithms

Testing rings:

```
myring_init(ctx);
gr_test_ring(ctx, iters, 0);
...
gr_mat_test_mul(myring_mat_mul, state, 5, iters, ctx);
...
myring_clear(ctx);
```

Testing generic algorithms:

```
gr_ctx_init_random(ctx, state);
...
if (status == GR_SUCCESS
    && gr_equal(R1, R2, ctx) == T_FALSE)
    // FAIL
```

New style rings

To implement new rings in FLINT, I recommend subtyping `gr_ctx_t` and following the `gr` interface.

See: `nfloat`, `mpn_mod`.

This gives us a lot for free.

Long term plan: gradually port finite fields, number fields, multivariate polynomials, ... to generics (making `fq_ctx_t` a subtype of `gr_ctx_t`, etc.). Currently we need wrapper layers.