The new ellinit

Karim Belabas
http://pari.math.u-bordeaux.fr/

The old ellinit (1/3)

```
E = ellinit([a1,a2,a3,a4,a6], flag = 0);
E = ellinit([a4,a6]);
```

returns an *ell* structure associated to E/K (K inferred from coefficients), passed as a first argument to elliptic curves functions. It is a vector containing:

- the curve coefficients and standard simple invariants $(b_2, b_4, b_6, b_8, c_4, c_6, \Delta, j)$
- approximations to $[e_1, e_2, e_3]$, $[\omega_1, \omega_2, \eta_1, \eta_2]$ if the a_i are real (`realprecision`).
- approximations to $[e_1, u, u^2, q, w]$ if the a_i belong to $\mathbb{Q}_p$ if the a_i are `t_PADIC` (precision of the a_i).

The flag allows *not* to compute the extended "domain-specific" components.

The old ellinit (2/3)

Drawbacks :

- Prime finite fields are somewhat supported (simple operations, no useful data stored)
- Non-prime finite fields are almost unsupported: point counting not even possible.
- `t_PADIC` supported only for $v_p(j) < 0$ (Tate curve), $p \neq 2$.
- `t_COMPLEX` unsupported (type error in `gsigne`)
- No other domains are supported. Functions individually try to guess the base field by considering `type(j)` or `type(`Δ`)` and act according to this, sometimes surprisingly (`ellisoncurve` for non exact input?)
- inexact data in *ell* structure is cached at an accuracy which is fixed at the time of `ellinit` call, and cannot be later updated.

The old ellinit (3/3)

Major problems :

- Useful data not cached (reduced period lattice basis, $\#E(\mathbb{F}_p)$, conductor and reduction type); useless data data included (E`.area`, E`.w`, η_1, η_2, the latter two being *very* expensive when `realprecision` is large). How to specify that some data must be precomputed, and some should not, depending on later applications?
- No way to specify a curve E/K and consider it over an extension. New functions for curves over $E/\mathbb{F}_q$ can't even be exported to GP in this model $\Rightarrow$ `ellffinit`, a new data type specific to curves over finite fields.
- Painful to change or extend (compatibility), cached data should depend on base field. And we would like to allow $\mathbb{F}_q$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, number field K, $\mathbb{Q}_p$, local field K_v...

Restart from scratch, new ellinit (1/1)

```
E = ellinit([a1,a2,a3,a4,a6], D);
E = ellinit([a4,a6], D);
```

where D encodes the “domain” over which we consider E. The result is mostly an empty shell: it includes only

- the standard simple invariants,
- the domain D, and a *default* accuracy for inexact data,
- *cheap* static domain-specific data (e.g. a morphism to a nice canonical model),
- dynamic domain-specific data, to be computed later, when and if needed. Any non-trivial information may (and will) be stored, when computed.
- if input is exact, E is exact; allowing to later compute approximate data to arbitrary accuracy.
- return approximate data at the accuracy requested by the user (`realprecision`) at the time of the call. If cached data too imprecise, recompute to higher accuracy and cache new value (same as `Pi`)

Restart from scratch, new ellinit (2/2)

Bug fixes (in progress) :

- non-prime finite fields are now fully supported.
- curves over $\mathbb{Q}_p$ are supported, for all p, and all reduction type.
- over $\mathbb{Q}_p$ (if multiplicative reduction), `ellpointtoz` now distinguishes between P and $-P$; we really return the parameter t in $\mathbb{Q}_{p^2}/q^{\mathbb{Z}}$, not $t + 1/t$ as before. The result lives in $\mathbb{Q}_{p^2}$ when the reduction is not split. Apparently, there remains a bug in the program since the result is sometimes obviously wrong.
- over $\mathbb{Q}_p$ (if multiplicative reduction), `ellztopoint` still not implemented.

Remaining Problems (1/1)

Data structure implementation and API, see `Records and Lazy vectors` in `libpari.dvi`.

Problem 1 : in GP, inserting new data into existing structures must be done via clones, inducing memory leaks *when* the structure is not stored into a GP variable: e.g.

```
ap = ellap(ellinit([1,1], Mod(1,p)))
```

instead of.

```
E = ellinit([1,1], Mod(1,p));
ap = ellap(E);
```

I see no good solution yet, besides telling GP users not to do this. Only storing data if struct is stored into a GP variable (as GP lists do) prevents library use! (Not a problem for lists, which are useless in library mode.)

Problem 2 : in libpari, such objects must be explicitly destroyed (`obj_free`) to avoid memory leaks. must

Remaining Problems (1/2)

Problem 3 : member function are not passed `realprecision`: use default precision (`ellperiods`(E) solves this by increasing that default precision in E). No way yet to do the same for p-adics: `elltateparametrization`(E) to be implemented. Maybe an `ellnewprec`, rather ?

What next ? (1/1)

Need to implement new domains: number fields, local fields (say, completions of number fields); need to implement new methods (e.g. Tate reduction and formal groups over local fields).

What next ? (1/1)

Need to implement new domains: number fields, local fields (say, completions of number fields); need to implement new methods (e.g. Tate reduction and formal groups over local fields).

Number fields ? Merging *nf* and *bnf* structures seems indicated: `nfinit` would compute trivial invariants, anything non-trivial would be computed on demand. No function would need to require a *bnf*. One should never have to restart a computation because some useful flag was omitted at initialization time: the missing data should be computed on the fly and inserted into the structure.

What next ? (2/2)

This is all very nice *if* we know that the variable value contains an elliptic curve. It would be nicer if we could “tag” a `GEN` so that it “knows” it is an elliptic curve. Then we wouldn’t have to rely on checking external type (`t_VEC` vs. `t_COL`, etc.) or lengths and making educated guesses. It also becomes trivial to implement

```
(08:58) gp > ?E
E is an elliptic curve defined over ℚ
(08:58) gp > ??E
E is the elliptic curve 15a1 defined over ℚ:
```

$$Y^2 + (X + 1) * Y = X^3 + X^2 - 10 * X - 10$$

```
E(Q) = []


(08:58) gp > ?K
K is a number field
```