Extending ConTEXt MkIV with PARI-GP

- Lua is a scripting language that can be used *to extend* an application
- LuaTEX is an application (TEX) that is *extended* with Lua
- ConTEXt `MkIV` is a macro format built on the top of LuaTEX. Macros are written in Lua and TEX

- To extend Lua with a C library it's necessary a specific software layer (the *binding*) between Lua and the library. The binding must adhere to the C API described in the ``Programming in Lua'' book
- in general build a binding is not simple, but Lua was designed to be an extensible language

- SWIG: **S**implified **W**rapper and **I**nterface **G**enerator.

- SWIG is a program that helps the developer to build a Lua binding by parsing the header files of the C library

- The developer lists into an *interface* file `*.i` the set of simbols he/she wants to access from Lua
- the `swig` program reads the interface file and produces the binding (a.k.a wrapper)
- the developer compiles the binding and produces the Lua module

- PARI/GP is a compact *Computer Algebra System* (CAS)
- it can do numerical and (to some extent) symbolic calculations
- it's a library (`libpari`) and an interpreted language (GP) similar to Lua
- it's stable and well written library available for most OS.

The interface file pari.i

```
%module pari
%{
#include "pari.h"
ulong overflow;
%}

%ignore gp_variable(char *s);
%ignore setseriesprecision(long n);
%ignore  killfile(pariFILE *f);

%include "pari/paritype.h";    %include "pari/paristio.h";
%include "pari/parisys.h";     %include "pari/paricom.h";
%include "pari/parigen.h";     %include "pari/parierr.h";
%include "pari/paricast.h";    %include "pari/paridecl.h";
%include "pari/paristio.h";    %include "pari/paritune.h";
%include "pari/paricom.h";     %include "pari/pariinl.h";

%inline %{
GEN uti_mael2(GEN m,long x1,long x2)
{return mael2(m,x1,x2);}
```

- Generating the binding

```
swig  -lua pari.i
```

- Compiling the module

```
gcc -ansi \
    -I./pari -I/opt/swig-2.0.2/include \
    -c pari_wrap.c  -o pari_wrap.o

gcc -Wall -ansi -shared -I./pari \
    -I/opt/swig-2.0.2/include -L./ \
    -L/opt/swig-2.0.2/lib pari_wrap.o \
    -lpari -lm -o pari.so
```

- The module is then available in Lua with `require("pari")` as `pari` table

Let's start with $\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1}$: the Lua code is

```
\startluacode
require("pari")
pari.pari_init(4000000,500000)
document = document or {}
document.lscarso= document.lscarso or {}
local function sum(X,a,b,expr,start)
   local avma = pari.avma
   local start = start or '0.'
   local res = pari.gp_read_str(string.format(
     "sum(%s=%s,%s,%s,%s)",X,a,b,expr,start))
   res = pari.GENtoTeXstr(res)
   pari.avma = avma
   return res
end
document.lscarso.sum = sum
\stopluacode
```

The TEX code is even simpler:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,30,"4*(-1)^k/(2*k+1)","0"))}
\stopformula
\stopTEXpage
\stoptext
```

It gives the *exact* result:

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = \frac{5863013579100197316985228 4}{1847292006410659792986502 5}$$

We can calculate an approximation:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,30,"4*(-1)^k/(2*k+1)","0."))}
\stopformula
\stopTEXpage
\stoptext
```

With a precision of 28 digits we have:

$$\sum_{k=0}^{30} \frac{4(-1)^k}{2k+1} = 3.17384233719074940869022414 0$$

and we can also calculate a symbolic sum:

```
\starttext
\startTEXpage
\startformula
\sum_{k=0}^{3}\frac{1}{x^2+k}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,3,"1/(x^2+k)","0"))}
\stopformula
\stopTEXpage
\stoptext
```

$$\sum_{k=0}^{3} \frac{1}{x^2+k} = \frac{4x^6+18x^4+22x^2+6}{x^8+6x^6+11x^4+6x^2}$$

Some advantages of the binding :

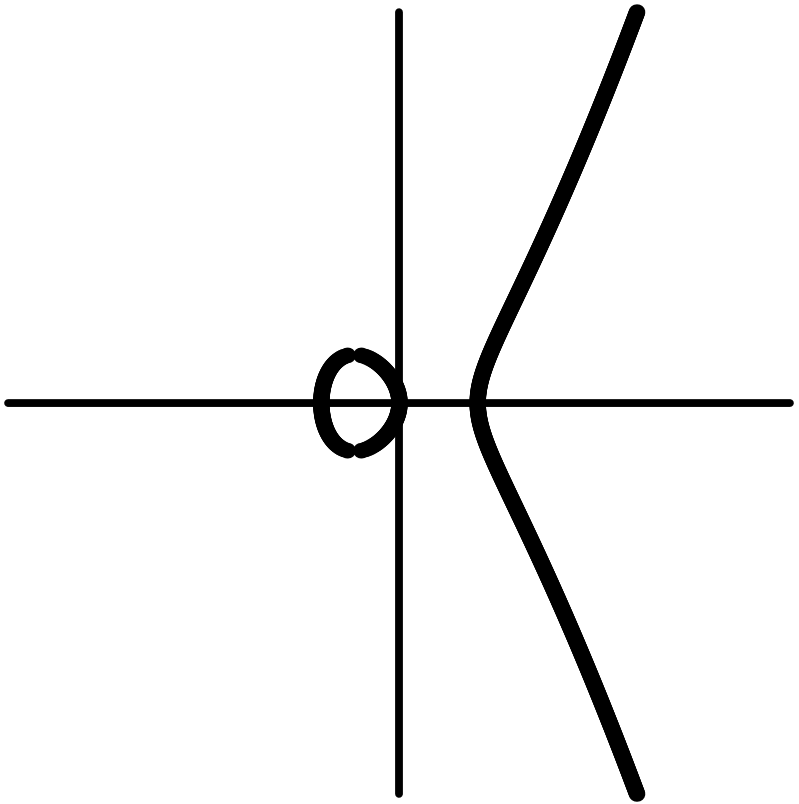
- clear separation between Lua code (logic) and ConTEXt MkIV code (presentation) because ConTEXt MkIV has an advanced Lua layer
- reuse of Lua code
- reuse of GP scripts
- easy access to the functions of libpari

Another advantage of ConTEXt MkIV: tight integration with METAPOST.

The next example will show how to plot the real roots of $x^3 - x - y^2$ for $y \in [-5,5]$ with step 2^{-6}.

The paper explains how to implement the function `polroots` (it is basically a wrapper around the `roots` function of `libpari`): the code is an example of integration of Lua and METAPOST code in ConTEXt MkIV.

```
\startluacode
local poly = "x^3-x-y^2";local step= 1/2^6
local results = {}      ;local L = 5
local zero = '0.E-96'   ;local prec = 12
get_value = document.lscarso.get_value
polroots  = document.lscarso.polroots
context("\\startMPpage")
context("pickup pencircle scaled 0.1pt;")
context(string.format("draw (-%s,0)--(%s,0);", L, L))
context(string.format("draw (0,-%s)--(0,%s);", L, L))
context("pickup pencircle scaled 0.2pt;")
for y=-L,L,step do
 local poly_x = get_value(poly,'y',y,prec)
 local roots = polroots(poly_x,prec)
 for _,root in pairs(roots) do
  local real,imag = root[1],root[2]
  if imag == zero then
     if real ==  zero then real = '0' end
     context(string.format("draw (%s,%s);",real,y))
  end  end end
context("\\stopMPpage")
\stopluacode
```



Another example: implicitization of a cubic Bezier curve. Given the parametric form of a cubic Bezier

$$\begin{aligned}\mathcal{C} = \{&(1-t)^3\mathbf{p} + 3(1-t)^2 t\mathbf{c_1} \\ &+ 3(1-t)t^2\mathbf{c_2} + t^3\mathbf{q}, \\ &0 \le t \le 1\}\end{aligned}$$

for a point $(x_t, y_t) \in \mathcal{C}$ we have

$$\begin{aligned} x_t &= a_3t^3 + a_2t^2 + a_1t + a_0 = \mathrm{a}(t) \\ y_t &= b_3t^3 + b_2t^2 + b_1t + b_0 \ = \mathrm{b}(t) \end{aligned}$$

Following Sederberg(chap. "Algebraic Geometry for CAGD"), let

$$\mathrm{f} = \mathrm{f}(t,x) = \mathrm{a}(t) - x$$

$$\mathrm{g} = \mathrm{g}(t,y) = \mathrm{b}(t) - y$$

and

$$\mathrm{h}_1(t,x,y) = (a_3\mathrm{g} - b_3\mathrm{f})$$

$$\mathrm{h}_2(t,x,y) = (a_3t + a_2)\mathrm{g} - (b_3t + b_2)\mathrm{f}$$

$$\mathrm{h}_3(t,x,y) = (a_3t^2 + a_2t + a_1)\mathrm{g} - (b_3t^2 + b_2t + b_1)\mathrm{f}$$

It's better to rename $(t,x,y) \to (x,X,Y)$ so that each h_j can be seen as a polynomial $(h_j[X,Y])[x]$ with at most degree 2 with respect to x.

If we are able to find
$h_1[x] = h_2[x] = h_3[x] = 0$
(the *null polynomial*) then we have found the implicit form of our curve.

It can be demonstrated that, if $h_{j,n}$ is the coefficient of x^n of h_j, the product

$$\begin{pmatrix} h_{1,2}[X,Y] & h_{1,1}[X,Y] & h_{1,0}[X,Y] \\ h_{2,2}[X,Y] & h_{2,1}[X,Y] & h_{2,0}[X,Y] \\ h_{3,2}[X,Y] & h_{3,1}[X,Y] & h_{3,0}[X,Y] \end{pmatrix} \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix}$$

is identically equal to $(0,0,0)^T$

if and only if

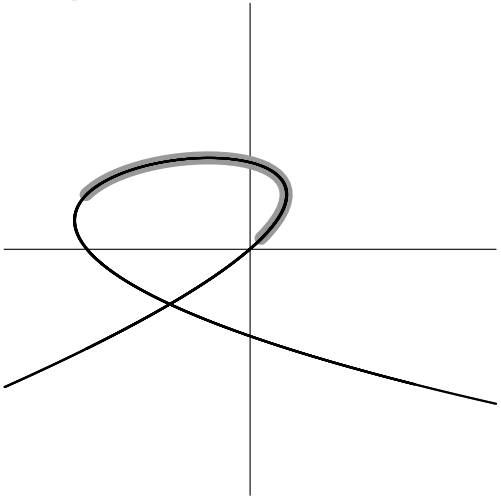
$$\begin{vmatrix} h_{1,2}[X,Y] & h_{1,1}[X,Y] & h_{1,0}[X,Y] \\ h_{2,2}[X,Y] & h_{2,1}[X,Y] & h_{2,0}[X,Y] \\ h_{3,2}[X,Y] & h_{3,1}[X,Y] & h_{3,0}[X,Y] \end{vmatrix} = 0$$

and hence this determinant is our $P[X,Y]$.

The key point here is that it's possible to implement an algorithm for implicitization literally following the informal steps described above: for example, for the curve $\mathcal{C}$ with $\mathbf{p} = (1,1)$, $\mathbf{c_1} = (10,10)$, $\mathbf{c_2} = (-10,10)$, $\mathbf{q} = (-15,5)$ we have

$$P[X,Y] = -64X^3 + (2112Y + 312360)X^2 + (-23232Y^2 - 67920Y + 4711200)X + (85184Y^3 - 4440Y^2 - 5383200Y + 368000)$$

The next picture shows the METAPOST curve (thick, color gray) and the roots of $P[X,Y]$ for $-15 \leq x \leq 15$, $-15 \leq y \leq 15$ (thin, color black):



- PARI/GP is useful *to translate* mathematical ideas into concrete algorithms
- ConTEXt MkIV is useful *to present* mathematical formulae and results
- METAPOST is useful *to draw* mathematical relations

Conclusions:

- PARI/GP is a well-written C library and build a bindind is easy — *but in general it's a difficult task, even with SWIG*
- the Lua code is platform independent and not strictly tied to ConTEXt MkIV— *but the binding is specific for each platform*
- it's easy to use for simple calculations — *but for a serious use it's necessary to study number theory at graduate level*

That’s all

Thank you !