The GP-to-C compiler

GP2C

GP2C basics

General information

GP2C is a software package to automatically convert GP scripts to libpari programs.

The current version is 0.0.3pl1.

It it available from the main PARI website:

`http://pari.math.u-bordeaux.fr/`

including the CVS version. (Just enter cvs co gp2c inside pari CVS)

The tools

The GP2C package provides 3 tools:

gp2c	:	translate GP code to C code.
gp2c-run	:	compile and run a GP script under GP.
gp2c-dbg	:	compile and run a GP script under GP runing under GDB.

Design goals

GP2C was designed with the following goals in mind:

- To produce nice readable C code close to hand-written code found in the libpari code.

- The generated code should give the exact same result as the GP code in all normal circumstances.

A tutorial to GP2C

1. How can I compile and run my scripts?

2. Using option -g to enable garbage collecting.

3. How can I compile directly with gp2c ?

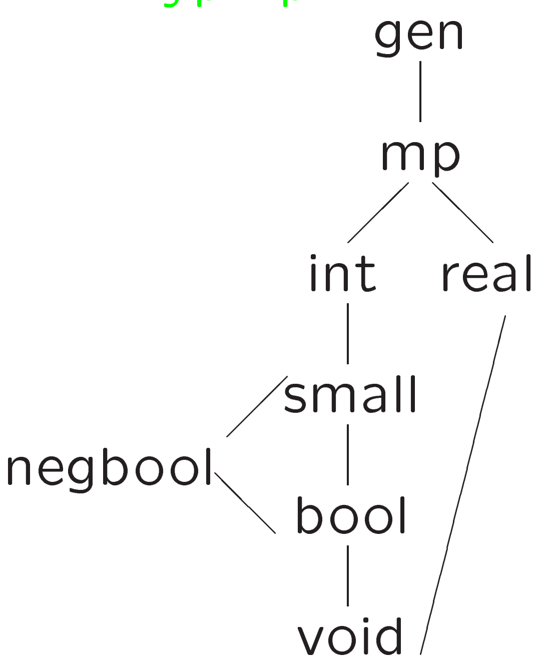
4. Using GP2C to find errors in GP scripts.

5. Using compiled functions in a new program.

Advanced GP2C use

Typing

Type preorder



nom	description
`void`	like in C
`bool`	boolean, true (1) or false (0)
`negbool`	antiboolean, true (0) or false (1)
`small`	C integer `long`
`int`	multiprecision integer
`real`	multiprecision floating point
`mp`	multiprecision number.
`gen`	generic PARI object.

Description of GP functions and operators

PARI provides a database of all functions and operators known to GP together with a 'description' (in src/functions).

Example: The description of the operator **+**:

```
_+_:
(small,small):small:parens     $(1) + $(2)
(small,int):int            addsi($1, $2)
(int,small):int            addis($1, $2)
(int,int):int              addii($1, $2)
(real,mp):real             mpadd($1, $2)
(mp,mp):mp                 mpadd($1, $2)
(gen,small):gen            gaddgs($1, $2)
(gen,gen):gen              gadd($1, $2)
```

Type declaration and casting

To declare that a variable belongs to type `type` use:
function(x:type)
or
local(x:type)

To declare that an expression value belongs to type `type`, use:
expr:type

GP2C will check types consistency and output warnings if necessary.

Example of optimisation

```
rho(n)=
{
  local(x,y);

  x=2; y=5;
  while(gcd(y-x,n)==1,
    x=(x^2+1)%n;
    y=(y^2+1)%n; y=(y^2+1)%n
   );
  gcd(n,y-x)
}
```

```
GEN rho(GEN n)
{
  GEN x;
  GEN y;
  x = gdeux;
  y = stoi(5);
  while (gegalgs(ggcd(gsub(y, x), n), 1))
  {
    x = gmod(gaddgs(gsqr(x), 1), n);
    y = gmod(gaddgs(gsqr(y), 1), n);
    y = gmod(gaddgs(gsqr(y), 1), n);
  }
  return ggcd(n, gsub(y, x));
}
```

We declare that the variables n, x et y will contain integers as follows:

```
rho(n:int)=
{
local(x:int,y:int);

  x=2; y=5;
  while(gcd(y-x,n)==1,
    x=(x^2+1)%n;
    y=(y^2+1)%n; y=(y^2+1)%n
   );
  gcd(n,y-x)
}
```

```
GEN rho(GEN n)   /* int */
{
  GEN x;           /* int */
  GEN y;           /* int */
  x = gdeux;
  y = stoi(5);
  while (cmpis(gcdii(subii(y, x), n), 1) == 0)
  {
    x = modii(addis(sqri(x), 1), n);
    y = modii(addis(sqri(y), 1), n);
    y = modii(addis(sqri(y), 1), n);
  }
  return gcdii(n, subii(y, x));
}
```

The code now uses more specific functions `sqri`, `addis`, `modii`, and `gcdii`.