Exploiting front end knowledge to effortlessly create Python modules

Gaius Mulley
⟨gaius@gnu.org⟩

these slides are available from
⟨http://www.nongnu.org/gm2/index.html⟩

Exploiting front end knowledge to effortlessly create Python modules

- GNU Modula-2 release 1.0 is imminent
  - current released version is 0.99 (gm2 is GPL v3)
  - grafted onto the front of gcc-4.1.2
  - implements Niklaus Wirth language definitions described in pim2, pim3, pim4 and also ISO Modula-2 (`-fpim2`, `-fpim3`, `-fpim4` and `-fiso`)
  - complete set of re-implemented ISO libraries (LGPL)
  - University of Ulm (pim) libraries
  - m2f libraries and partial set of Logitech libraries (pim)
  - coroutines implemented on top of `libpth`

- all 10086 regression tests pass on x86_32 and x86_64 Debian GNU/Linux

GNU Modula-2 features

- tries to fit in with the GCC family so it implements extensions such as:
  - users can run the C preprocessor on all source beforehand via (`-fcpp` which runs `cpp` using the options `-lang-asm -traditional-cpp`)
  - in-line assembly language syntax exactly follows GNU C (except it uses capital keywords)

- runtime checking can be enabled via: `-fsoft-check-all` which turns on `-fnil`, `-fcase`, `-fwholediv`, `-findex`, `-frange`, `-freturn`
  - `-O[0123s]` are supported together with `-shared`

- also provides users with `-fswig` which will automatically create a swig interface file for the current module

Effect of `-fswig`

- this option tells `gm2` to produce a `swig` interface file for the current implementation module

- it examines each parameter for each exported procedure
  - it works out whether a parameter is an: `INPUT`, `OUTPUT` or `INOUT` depending if the variable is read, written or both read and written in the initial basic block

- if the variable is manipulated in subsequent basic blocks then a comment is issued indicating that this is a guess

Consider an event driven collision simulation of numerous moving spheres library

- an abridged definition module ⟨`http://cvs.savannah.gnu.org/viewvc/*checkout*/gm2/examples/gravity/twoDsim.def?revision=1.8&root=gm2`⟩

Consider an event driven collision simulation of numerous moving spheres library

```
DEFINITION MODULE twoDsim ;

EXPORT UNQUALIFIED gravity, box, circle, fps, mass,
                   simulateFor, fix, replayRate ;

PROCEDURE gravity (g: REAL) ;
PROCEDURE box (x0, y0, i, j: REAL) : CARDINAL ;
PROCEDURE circle (x0, y0, r: REAL) : CARDINAL ;
PROCEDURE fps (f: REAL) ;
PROCEDURE mass (id: CARDINAL; m: REAL) : CARDINAL ;
PROCEDURE simulateFor (t: REAL) ;
PROCEDURE fix (id: CARDINAL) : CARDINAL ;
PROCEDURE replayRate (f: REAL) ;

END twoDsim.
```

Python application code

```
from twoDsim import *

b = box(0.0, 0.0, 1.0, 1.0)
b = fix(b)
c1 = circle(0.5, 0.7, 0.05)
c1 = mass(c1, 0.01)
for x in [0.1, 0.3, 0.5, 0.7, 0.9]:
    c = circle(x, 0.1, 0.1)
    c = mass(c, 0.01)
    c = fix(c)
```

Python application code

```
c = circle(0.1, 0.3, 0.1)
c = mass(c, 0.01)
c = fix(c)
c = circle(0.9, 0.3, 0.1)
c = mass(c, 0.01)
c = fix(c)
for x in [0.4, 0.7]:
    c = circle(x+0.01, 0.3, 0.05)
    c = mass(c, 0.01)
gravity(-9.81)
fps(24.0*4.0)
replayRate(24.0)
print "creating frames"
try:
    simulateFor(3.0)
    print "all done"
except:
    print "exception raised"
```

implement twoDsim.mod

- using discrete event simulation
- we use Newtonian mathematics to predict when moving spheres collide as we know the acceleration and velocity of each sphere
  - yields a quartic equation which is solved using Ferrari’s solution
- requires the type `COMPLEX`

Building the Python module using gm2

```
$ gm2 -g -fPIC -fiso -c roots.mod
$ gm2 -g -fPIC -fiso -c -fswig twoDsim.mod
```

- creates both a position independent object `twoDsim.o` and swig interface file `twoDsim.i`

Building the Python module using gm2

- now use swig to generate the wrapping code

```
$ swig -c++ -python twoDsim.i
$ gcc -c -fPIC twoDsim_wrap.cxx -I/usr/include/python2.5
```

- and use gm2 to create the shared library initialization/finalization code and satisfy all the linker dependencies

- in ISO Modula-2 each implementation module may have an initialization/finalization section of code

```
$ gm2 -fiso -fPIC -fshared -shared \
twoDsim.mod twoDsim_wrap.o -o _twoDsim.so
```

gm2 linker

- builds a list of all dependant modules by following the imports
- and uses this list to construct the initialisation/finalisation code

Abridged initialization code

```
void __attribute__ ((constructor)) init (void)
{
   try {
       _M2_Storage_init (0, (char **)0);
       _M2_SYSTEM_init (0, (char **)0);
       _M2_M2RTS_init (0, (char **)0);
       _M2_RTExceptions_init (0, (char **)0);
       _M2_roots_init (0, (char **)0);
       _M2_twoDsim_init (0, (char **)0);
    }
    catch (...) {
       RTExceptions_DefaultErrorCatch();
    }
}
```

Abridged finalization code

```
void __attribute__ ((destructor)) finish (void)
{
   try {
      _M2_twoDsim_finish (0, (char **)0);
      _M2_roots_finish (0, (char **)0);
      _M2_RTExceptions_finish (0, (char **)0);
      _M2_M2RTS_finish (0, (char **)0);
      _M2_SYSTEM_finish (0, (char **)0);
      _M2_Storage_finish (0, (char **)0);
    }
    catch (...) {
       RTExceptions_DefaultErrorCatch();
    }
}
```

Manipulating the initialization/finalization order

- the option `-fmakelist` will generate a *modulename*`.lst`
- and the option `-fuselist` will request that `gm2` use *modulename*`.lst` rather than derive one from the import dependencies

Limitations of `-fswig`

- consider a heavily abridged Modula-2 definition module from the pim library, NumberIO.def ⟨`http://cvs.savannah.gnu.org/viewvc/*checkout*/gm2/gm2-libs/NumberIO.def?revision=1.7&root=gm2`⟩

-
```
DEFINITION MODULE NumberIO ;

EXPORT QUALIFIED ReadCard, StrToInt ;

PROCEDURE StrToHex (a: ARRAY OF CHAR; VAR x: CARDINAL) ;
PROCEDURE StrToInt (a: ARRAY OF CHAR; VAR x: INTEGER) ;
PROCEDURE ReadInt (VAR x: CARDINAL) ;

END NumberIO.
```

Limitations of `-fswig`

- compiling the implementation module with:

```
$ gm2 -fshared -I. -c -fPIC -g -fswig -I../../../gm2-libs \
../../../gm2-libs/NumberIO.mod
```

- creates a position independent object file `NumberIO.o` and swig interface file `NumberIO.i`

NumberIO.i

```
extern "C" void NumberIO_StrToHex (char *_m2_address_a,
    int _m2_high_a, unsigned int *OUTPUT);
/*  parameter: x is known to be an OUTPUT */

extern "C" void NumberIO_StrToInt (char *_m2_address_a,
    int _m2_high_a, int *OUTPUT);
/*  parameter: x is guessed to be an OUTPUT */

extern "C" void NumberIO_ReadInt (int *INOUT);
/*  parameter: x is guessed to be an INOUT */
```

- `StrToHex` parameter `x` is written to in the first basic block
- `StrToInt` parameter `x` is written to, but not during the first basic block
- `ReadInt` passes, x, by address to another procedure (`StrToInt`) in the first basic block

Strings, Python and Modula-2

- now our Python application can be written:

```
import NumberIO

print "1234 x 2 =", NumberIO.NumberIO_StrToInt("1234")*2
```

```
$ python testnum.py
1234 x 2 = 2468
```

- the ugly name space is due to the procedures being `EXPORT QUALIFIED` rather than `EXPORT UNQUALIFIED` as in our `twoDsim` example

Conclusion

- the technique breathes life into existing legacy code
- provided users are willing to use Modula-2 they can easily produce Python modules (or any other swig supported scripting language)
- further work could be done by examining the final basic block executed in a procedure
- future work, will be directed in grafting GNU Modula-2 onto a branch of gcc near the head!

Demos

- gm2-lightening1.avi ⟨http://floppsie.comp.glam.ac.uk/download/avi/gm2-lightening1.avi⟩
- gm2-lightening2.avi ⟨http://floppsie.comp.glam.ac.uk/download/avi/gm2-lightening2.avi⟩

Appendix

- given the distance between the circles can be calculated as:
  $\sqrt{(c_{0x} - c_{1x})^2 + (c_{0y} - c_{1y})^2}$

- using the formula for initial position, velocity and acceleration:
  $s = s_0 + ut + \frac{at^2}{2}$ it is also known that the position circle c_0 at time, t,
  is: $\left[c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}, c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2} \right]$

- correspondingly the position circle c_1 at time, t, is:
  $\left[c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}, c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2} \right]$

Appendix

- therefore we need to find the time at which the distance between both circles is $r_0 + r_1$ which is:

- $$r_0 + r_1 = \sqrt{\left[\left(c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}\right) - \left(c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}\right)\right]^2 + \left[\left(c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2}\right) - \left(c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2}\right)\right]^2}$$

- which can be easily rearranged as:

- $$0 = \left[\left(c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}\right) - \left(c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}\right)\right]^2 + \left[\left(c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2}\right) - \left(c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2}\right)\right]^2 - (r_0 + r_1)^2$$