M2IF ENSL - 2015-16
CR10 : Program Analysis and Verification

TP1 : FramaC and Pagai

A `.tgz` archive of this lab is available on the course's website :

http://laure.gonnord.org/pro/teaching/verifM2.html

1 Exercises with Z3

https://github.com/Z3Prover/z3

Preparation Z3 is already installed on the ENS machines, in the following places :

```
/home/dmonniau/packages/z3/bin/z3
and
/home/dmonniau/packages/z3-unstable/bin/z3
```

You will have to set your `$PATH` and also your `$PYTHONPATH` :

```
export PYTHONPATH=/home/dmonniau/packages/z3/lib/python2.7/dist-packages/
```

Diamonds In the `Diamonds` directory of the archive, you will find :

- Two scripts `gen_diamond.py` and `gen_diamond2.py` to generate two different families of unsat "diamond" formula. For instance, `python gen_diamond.py 1` generates the following formula :

$$(y_0 \leq x_0 + 2) \wedge (z_0 \leq x_0 + 3) \wedge \big(x_1 \leq y_0 \vee x_1 \leq z_0\big) \wedge (x_1 > 3) \wedge (x_0 = 0)$$

- A script `gen_diamond3.py`that generates a family of sat formula.
- And also `gen_horn_diamond.py` `gen_horn_diamond_ungrouped.py`.

To check for sat/unsat, for instance :

```
$ python gen_diamond.py 1 > diam1.smt
$ z3 -smt2 diam1.smt
unsat
```

With Python/Gnuplot/whatever, try to caracterise the experimental complexity of Z3's algorithm on the first three classes of formula ($time = f(n)$ with an adequate n). You can use the `-st` option of Z3 to get some useful stats.

Pigeon formula A pigeon fancier has n nests and p pigeons, and want the following constraints to be respected :

- Each pigeon should be in a nest
- Each nest contains at most 1 pigeon.

The file `pigeons_partial.py` contains the begining of an encoding : the boolean variable $x_{i,j}$ represents the fact that pigeon i is in the jth nest. We encoded the first constraints. Complete the file and play with n and p.

2 Exercises with FramaC

http://frama-c.com/index.html

Preparation Put the directory in which we compiled FRAMA-C [1] in your `$PATH`, for instance insert the following line in your `.bashrc` :

```
PATH=$PATH:/home/lgonnord/.opam/4.02.1/bin/
```

In the archive, the `FramaC` directory contains :

— A Readme to use FRAMA-C : `ModeEmploiFramaC.txt`.

— A set of `.c` files.

— A Makefile.

First, compile all C files with `-Wall`, in order to be at least confident in their syntax (just type `make all`).

Do it yourself ! For each file [2], prove with FRAMA-C that the program has no incorrect execution, and that all the asserts are true (some indications are given as comments inside the files). Prove them in the following order :

— `mult.c` : a simple simple case, prove it with :

```
frama-c-gui -wp -wp-rte -wp-split mult.c &
```

— `arith1.c` : a first simple invariant

— `div*.c` : invariants + precondition + postcondition + function calls.

— `div*.c` : linear search in an array.

— `min_sort.c` : selection sort with a bit of pointer manipulation (`swap`)

3 Exercises with Pagai

http://pagai.forge.imag.fr/

Preparation Download a precompiled binary of PAGAI (on the ENS machines, the x86-64 version). Test it with the given example (`gopan.bc`), first compiled with `clang` (On the ENS machine, clang 3.4 is available in `/usr/bin`.)

```
clang -emit-llvm -g -c gopan.c -o gopan.bc
pagai -i gopan.bc
```

Finding invariants Play with Pagai (and the various abstract domains) to find numerical invariants :

— For some of the previous examples.

— For hand-written examples that show the need for the various abstract domains. (intervals, octogons, polyhedra).

— Do you manage to find a suitable invariant for the gaz burner example ? for the diamond example (see the course slides) ?

1. We used the opam installer
2. except `any.c`, which an auxiliary file for I/Os