

Prof. Burkhard Wolff
Parc Orsay Université
4, rue Jacques Monod
Building H / Room 012

Prof. Christine Paulin
Parc Orsay Université
4, rue Jacques Monod
Building N / Room 117

Proving Program Correctness

Date: 22.1.2009

The course website is www.lri.fr/~wolff/teach-material/2008-09/IFIPS-VnV/.

Generally, exercises will be posted online Monday afternoon. They can be (but need not be) handed in at the start of the exercises sessions, or in any other way that the assistants allows for.

Why

Why is an experimental environment for proving program correctness. We shall use the analyser of C code together with an automated theorem-prover Alt-Ergo. Both software are research tools developed at Université Paris-Sud.

Installing

1. The Why distribution is available from <http://why.lri.fr> and the Alt-Ergo distribution is available from <http://alt-ergo.lri.fr>.
2. To use them on linux PC (room 1 or 1bis at IFIPS) it is just necessary to add `/home/ensgnt/cpaulin/bin` in your path:
`export PATH=$PATH:/home/ensgnt/cpaulin/bin`

Simple test Why runs on C files annotated with specifications for instance:

```
/*@ ensures
  @   \result >= x && \result >= y
  @*/
int max(int x, int y) {
  if (x > y) return x; else return y;
}
```

- The post condition is introduced by the `ensures` clauses. Specification keywords like `\result` which represents the result of the function start with a `\` (other examples are `\forall`, `\true`).
- All specifications are special comments of C (so ignored by the compiler). They are written on a line starting with `/*@` or enclosed between `/*@` and `*/`. The character `@` is ignored inside the specification.
- If `max.c` contains the previous program, run `gwhy max.c`. It will open a window which displays a list of proof obligations. You can click on them to see the corresponding statement.
- Click on the `Alt-ergo` button to start proving the goals.
- Change the specification to capture that the result is the maximum of x and y and runs the tool again.

Exercise 1 (Testing *Square root*)

We start from the file `Sqrt0.c` which contains an implementation of the square-root example with the property `true` as pre and post-condition.

The loop is annotated with both an invariant and a variant (an expression which decreases in the body of the loop but stay greater than 0)

```

/*@ axiom square_sum : \forall int i; i * i + (2 * i + 1) == (i + 1) * (i + 1)

/*@ requires \true
   @ ensures \true
   @*/
int sqrt(int a) {
  int i = 0;
  int tm = 1;
  int sum = 1;
  /*@ invariant tm > 0
     @ variant (a - sum)
     @*/
  while (sum <= a) {
    i++;
    tm=tm+2;
    sum=sum+tm;
  };
  return i;
}

```

1. Run `gwhy` on this program and prove the obligations.

2. Change the specification to express the square root property and try to prove the corresponding proof obligations. It is possible to add assertions `//@ assert b` inside the program to help the prover.