# Formal proofs of numerical programs

Nguyen Thi Minh Tuyen, Sylvie Boldo, Claude Marché

digiteo

Research in science and technology of information

Forum 2010

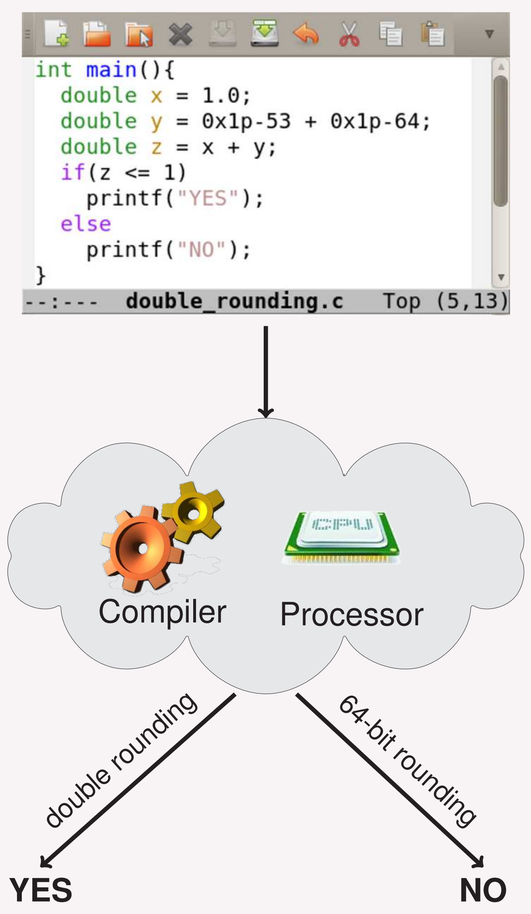
www.digiteo.fr

NGUYEN Thi Minh Tuyen

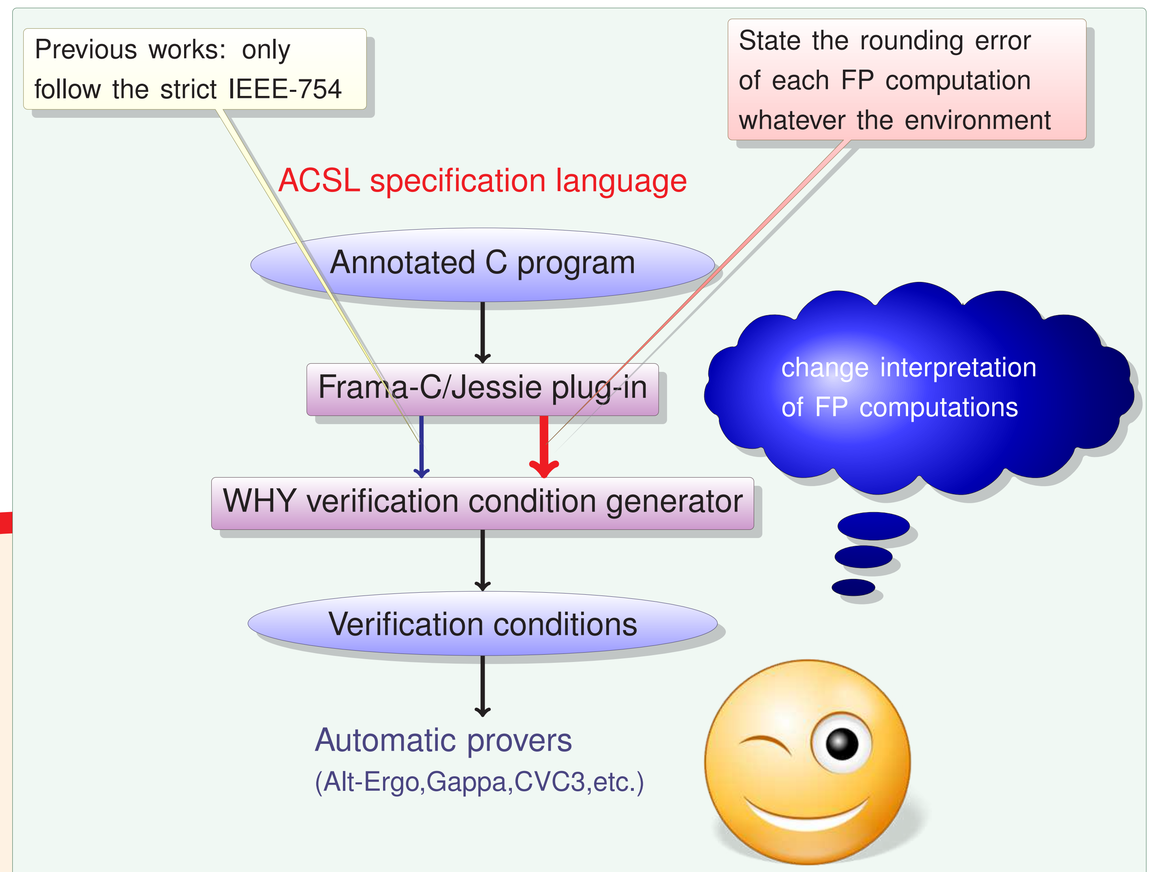
Nº 10

# FORMAL PROOFS OF NUMERICAL PROGRAMS

```
int main(){
  double x = 1.0;
  double y = 0x1p-53 + 0x1p-64;
  double z = x + y;
  if(z <= 1)
    printf("YES");
  else
    printf("NO");
}
--:---  double_rounding.c   Top (5,13)
```




**How to prove a program whatever the compiler and the architecture?**




## Theorem

Let $\varepsilon' = 2051 \cdot 2^{-60}$ and $\varepsilon = 2050 \times 2^{-64}$. Let $\eta' = 2049 \times 2^{-1082}$ and $\eta = 2049 \times 2^{-1086}$. If we define each operation result as any real such that

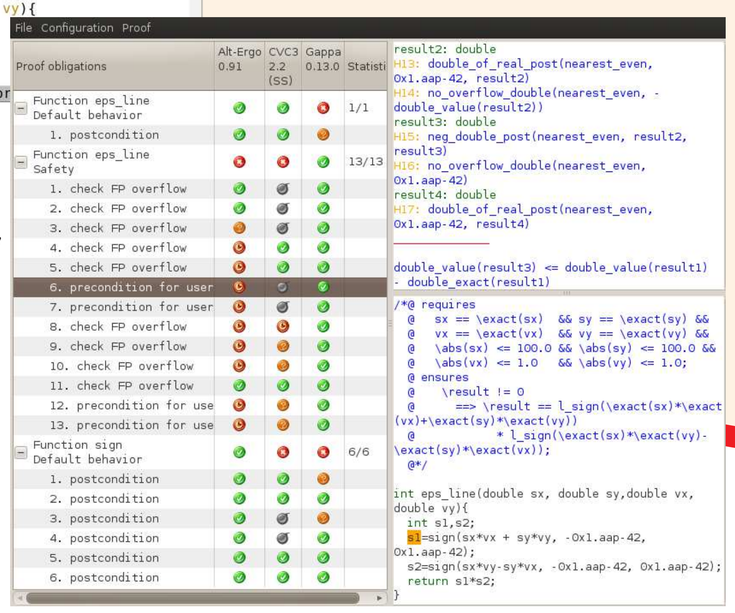
$$|x \oplus y - (x + y)| \leq \varepsilon' \cdot (|x| + |y|) + \eta'$$
$$|x \ominus y - (x - y)| \leq \varepsilon' \cdot (|x| + |y|) + \eta'$$
$$|x \otimes y - (x * y)| \leq \varepsilon \cdot |x * y| + \eta$$
$$|x \oslash y - (x/y)| \leq \varepsilon \cdot |x/y| + \eta$$

and if we are able to deduce a property (such as a rounding error), then this property holds whatever the architecture and the compiler optimizations among commutativity, addition/subtraction associativity (for less than 16 additions), use of extended registers, expression factorization and unfactorization.

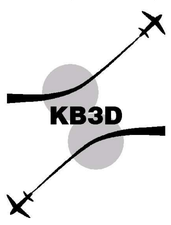
```
File  Edit  Options  Buffers  Tools  C  Help
/*@ logic integer l_sign(real x) = (x >= 0.0) ? 1 :-1;*/

/*@ requires e1<= x-\exact(x) <= e2;
  @ ensures  \abs(\result) <= 1 &&
  @    (\result != 0  ==> \result == l_sign(\exact(x)));
  @*/
int sign(double x, double e1, double e2) {
  if (x > e2)  return  1;
  if (x < e1)  return -1;
  return 0;
}

/*@ requires
  @   sx == \exact(sx)  && sy == \exact(sy) &&
  @   vx == \exact(vx)  && vy == \exact(vy) &&
  @   \abs(sx) <= 100.0 && \abs(sy) <= 100.0 &&
  @   \abs(vx) <= 1.0   && \abs(vy) <= 1.0;
  @ ensures
  @   \result != 0
  @     ==>\result==l_sign(\exact(sx)*\exact(vx) + \exact(sy)*\exact(vy))
  @              * l_sign(\exact(sx)*\exact(vy) - \exact(sy)*\exact(vx));
  @*/
int eps_line(double sx,double sy, double vx,double vy){
  int s1=sign(sx*vx+sy*vy,-0x1.aap-42,0x1.aap-42);
  int s2=sign(sx*vy-sy*vx,-0x1.aap-42,0x1.aap-42);
  return s1*s2;
}
--:---  eps_line.c     All (1,0)     SVN@0  (C/l Abbr
```




Case study
Part of KB3D NASA
http://research.nianet.org/fm-at-nia/KB3D/




## Future work

Look into multiplication/division reordering
Look into assembly

- know the order of the operations
- know the precision used for each operation
- know if the architecture supports FMA or not

This work was supported by the Hisseo project, funded by Digiteo (http://hisseo.saclay.inria.fr/)

Contact : Thi-Minh-Tuyen.Nguyen@inria.fr, Sylvie.Boldo@inria.fr, Claude.Marche@inria.fr