Analyses de terminaison des calculs flottants

Termination Analysis of Floating-Point Computations

Fonenantsoa MAURICA

Thèse de doctorat

Soutenue le 8 décembre 2017 devant le jury composé de :

Amir BEN-AMRAM	Examinateur
Sylvie BOLDO	Rapporteuse
Christian DELHOMME	Examinateur
Laure GONNORD	Examinatrice
Salvador LUCAS	Rapporteur
Frédéric MESNARD	Directeur
Marianne MORILLON	Examinatrice
Etienne PAYET	Co-directeur

Version courte pour RAIM 2018

ImageMagick

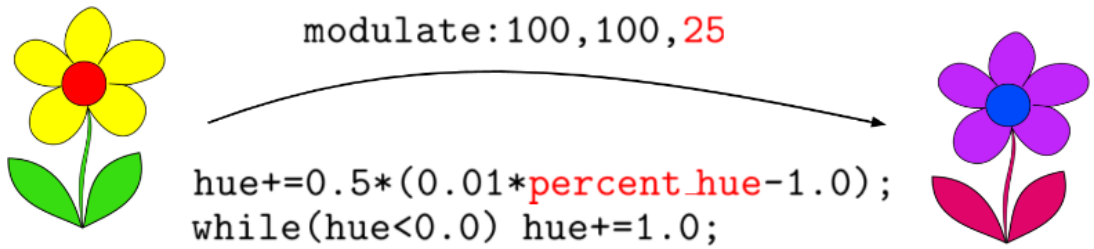


Fig 1: `ImageMagick v7.0.4-9` *non-terminate* `percent_hue=-1E25`

Care about termination.

Why does such bug occur?

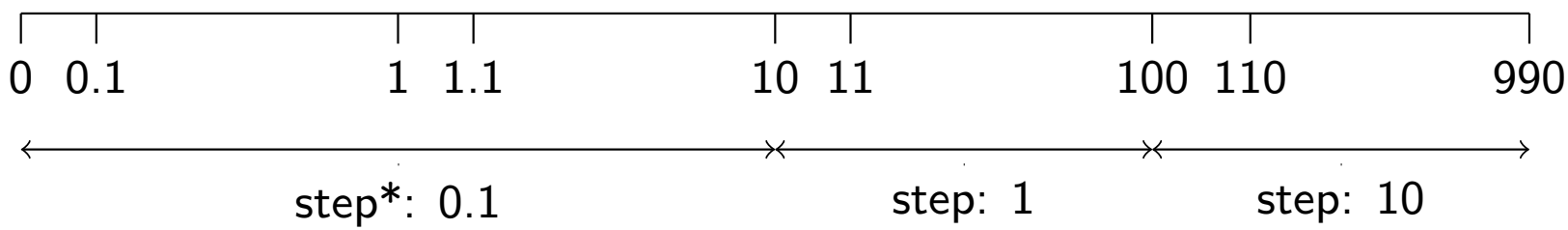
"95% of folks out there are completely clueless about FP" Gosling.

The best for these folks would be an easy-to-use (eg fully automatic) tool that prevents bugs from FPs.

FP Numbers

Similar to the "scientific notation":

$$\hat{x} \in \mathbb{F} : \hat{x} = \overbrace{(-1)^s}^{\text{thus } \pm 0} \quad \underbrace{\hat{m}}_{\substack{\text{finite expansion} \\ \text{thus rounding}}} \quad \beta \quad \overbrace{e}^{\substack{\text{finite thus} \\ \hat{x}_{\min} \leq \hat{x} \leq \hat{x}_{\max}}}$$



*difference between two consecutive FP numbers

Fig 2: A toy FP ty `myfloat`. *To* $x = 100\pi = 314.1592\cdots$ *corre* $\hat{x} = 310$ *if rounding to neare* $310 \oplus 1 = 310$.

The standard for FP numbers is IEEE-754. It defines: FP types, FP functions, rounding modes, exceptions.

Termination **IS** Decidable

And *many* non-trivial properties too!

Existence of RTE, most precise error bounds (even with transcendentals), synthesis of smallest possible programs...

This does not contradict Turing nor Rice. Contrarily to them, we talk here about machines with finite amount of memory.

But too costly:

Incompleteness is introduced for practical reasons.

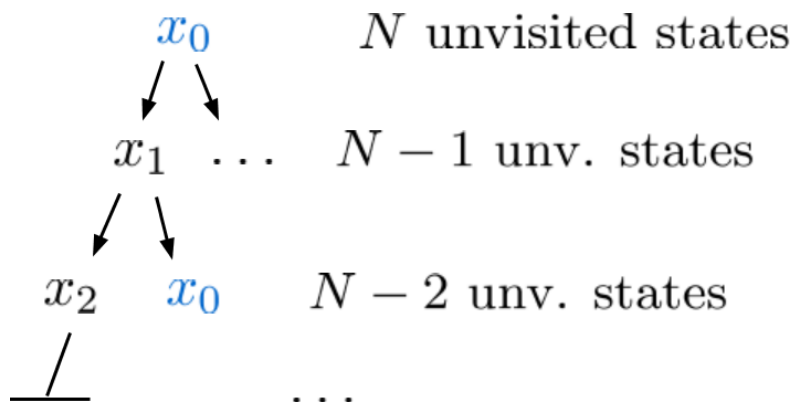
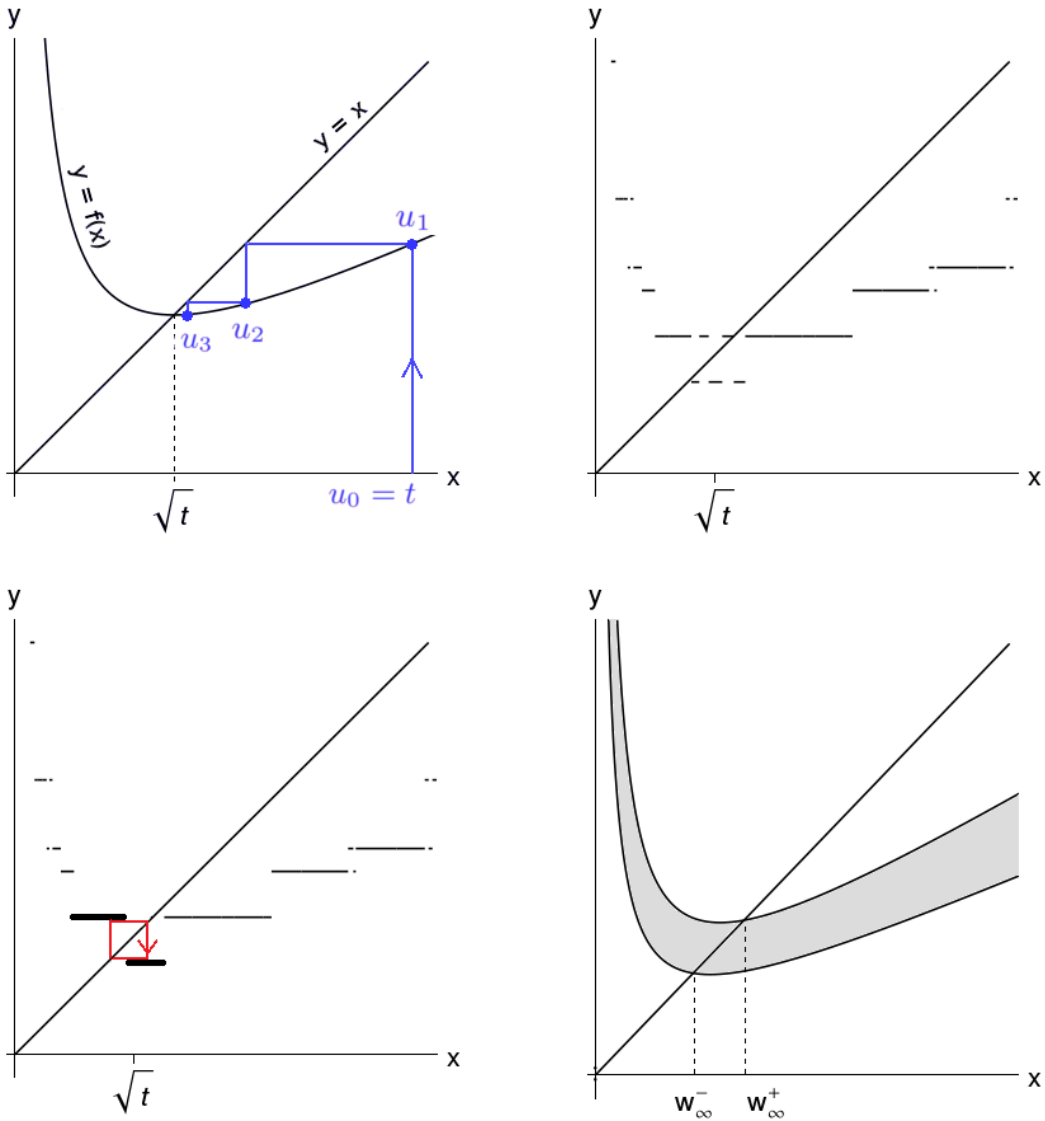


Fig 3: Non-termination if there is a state visited twice

Termination by Abstracting to the Rationals



From left to right, top to bottom:

- Computing $\sqrt{t}$
  Newton's method considers $f(x) = \frac{1}{2}(x + \frac{t}{x})$ and stops when $|u_{n+1} - u_n| < \delta$
- When using FP
  Computations become tricky
  Eg from `open-vm-tools`
- Risk of non-termination?
- No
  Under the condition that $\delta > \Delta$ where $\Delta = w_\infty^+ - w_\infty^-$ is found after linearly abstracting the rounding function.
  If $\delta \leq \Delta$, we don't know.

The PaaTerm Algorithm

[Mythesis, Algo 3.2] computes rational abstractions $\mathcal{R}^{\#}$ of the transition relation $\mathcal{R}$ by approximating the rounding function ζ_n with k-Piecewise Affine functions: $\forall x : \mu(x) \leq \zeta_n(x) \leq \nu(x)$.

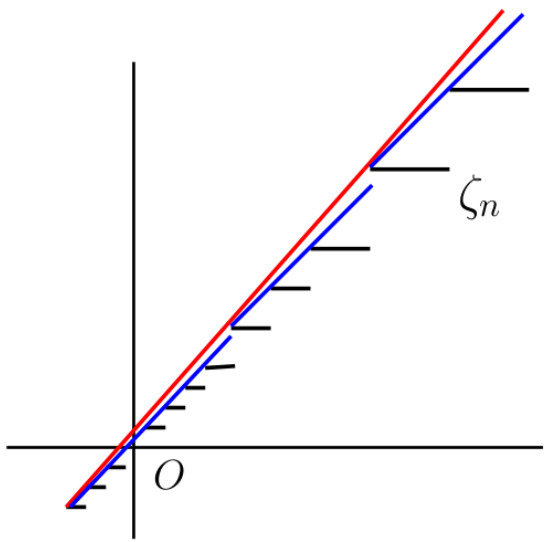


Fig 4: $k = 1$ *and* $k = 3$

Refine $\mathcal{R}^{\#}$ until we can prove termination or until we give up.

The refinement increases k.
It is such that the bigger k,
the bigger the set of Trace-Level Invariants,
the bigger the set of state-level variants,
the higher the "chance"
of proving termination.

F. Maurica, F. Mesnard, E. Payet. "Termination Analysis of Floating-Point Programs using Parameterizable Rational Approximations". ACM SAC 2016.

The PaaTerm Algorithm: Illustration

```
myfloat x = randFP();
        y = randFP();
while(x*y > 0 &
      y > -60) {
  x = x - 5.5;
  y = y + 3.5 + 3.5;
  if(x < -100) x = 100;
}
```

Fig 5: The program $\mathcal{P}_{my}$

$\mathcal{P}_{my}$ always terminates because:

(TLInv1) x is alternately $+$ and $-$

(TLInv2) y eventually becomes $+$

	(TLInv1)	(TLInv2)
$k = 1$	✓	✗
$k = 2$	✗	✓
$k = 3$	✗	✓
$k = 4$	✓	✓

Fig 6: TLInv captured when running PaaTerm

The fact that (TLInv1) is captured for $k = 1$ but lost for $k = 2$ is an exceptional case **[Thm 3.1-3.2]**

Practical Evaluation: the FTerm Analyzer

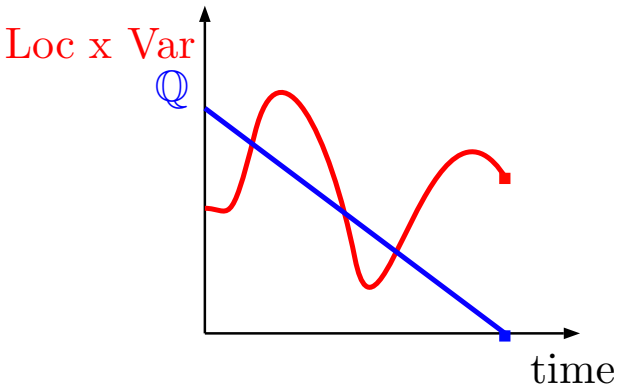
From	Verdict	Jugger-naut	2LS	AProVe Julia	**FTerm**	**Manual**
bcache-tools	✗	✗				
gauche	✗	✗				
ghostscript	?					
glibc	?					
libpano	✓	✓			✓	
maxlib	?					
maxlib	✓				✓	
postfix	✓				✓	
postfix	✓				✓	
bash	✓	✓	✓		✓	
git	✓					✓
open-vm-tools	✓					✓

Fig 7: ✓: *always terminates.* ✗: *there is at least one possible value of the variables that leads to non-termination.* ?: *can decide neither* ✓ *nor* ✗. *CO: time out after 300s on my i5-5200U 2.20GHz. Initial ranges of the variables are given.*

Other Approaches for Proving Termination

"Global approach"

Eg: find bounded linear descent



PR does that in P for simple rational loops.

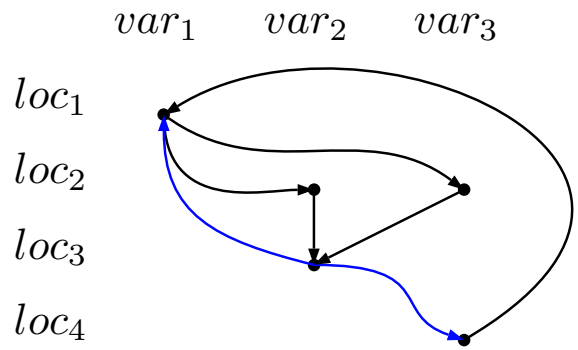
We show that the problem is coNP-hard for FP.
We give a FP adaptation of PR to FP.
P adaptation thus incomplete but optimal.

A. Podelski, A. Rybalchenko.
"A Complete Method for the Synth. of Lin. Rank Fun.". VMCAI 2004.

F. Maurica, F. Mesnard, E. Payet.
"On the Linear Ranking Pbl. for Simple FP Loops". SAS 2016.
"Optimal Approximation for Efficient Termination Analysis of FP Loops". IEEE NEXTCOMP 2017.

"Local approach"

Eg: check that all cycles of the *Size-Change graph* have a vertice of strict descent



[**Prop 5.4**] builds a domain of descent for any combination of FP types.
(⊃ machine integers)

[**Algo 5.2**] applies SC to FP programs.

C. Lee, N. Jones, A. Ben-Amram.
"The Size-Change Principle for Program Termination". POPL 2001.

A word on:
F. Zuleger. "Transition Invariants and SC". SAS 2018.

Termination by (the Dumbest Possible) Brute-Force

Juggernaut's brute-force[1]:
Uses advanced techniques: 2nd order SAT solver, genetic algorithms...
...Yet fails to decide termination of Newton's algorithm

What about the following brute-force?

```
float t, tnext = MIN_VALUE;
do {
   t = tnext;

  float v, v1 = t, d;
  do {
    v = v1;
    v1 = (v + t / v) / 2;
  } while (abs(v1 - v) > 0.5f);

tnext = nextAfter(t);
} while (t < MAX_VALUE);
```

It terminates within minutes!

How to combine cleverness with dumbness?

1. Reduce the search space through clever abstractions
2. Dumbly brute-force when the remaining space is small enough

[1] C. David et al. "Unrestricted Termin. and Non-term. Arguments for Bit-Vector Programs". ESOP 2015.

Exhibiting Non-Termination Bugs

Because people are more impressed by bug finding than by correctness proof

A simple template-based approach could be very effective.

For example `while(`$\underbrace{f + c_1 < c_2}_{\texttt{in } \mathbb{F}}$`){...}` is quite common in the wild.

For a fully automatic non-termination exhibitor:

1. Slicing[2] for keeping the information related to termination
2. Template-based non-termination checking (intra-procedural) for exhibiting non-termination condition
3. Backward value analysis (inter-procedural) for confirming that the bug is reproducible from the main program

[2] D. Monniaux. "The Pitfalls of Verifying FP Computations". TOPLAS 2008.

Concluding Words

"Computer scientists worldwide are working hard on schemes to debug and verify software, especially in the context of parallel computation, but not FP. Why not?"

Kahan in "Desperately Needed Remedies for the Undebuggability of Large FP Computations in Science and Engineering" 2014