Verification in Real Computation

Norbert Preining[1]

(joint work with Norbert Müller, Sewon Park, Martin Ziegler)

Accelia

Workshop on Mathematical Logic and its Application,
September 2016, Kyoto

[1] norbert@preining.info

Verification on discrete structures

well established

Verification on discrete structures

well established

- specification and verification
- models of computation
- cost analysis
- formal correctness proofs
- ...

Verification on continuous structures

just starting

Verification on continuous structures

just starting

- heuristics, numeric methods (very successful!)

Verification on continuous structures

just starting

- heuristics, numeric methods (very successful!)
- recursive analysis
  approximation via sequences of dense rationals, Turing machine, sound foundation, but not *programmer friendly*
- exact geometric computation
  nice idea, but hard to program, not compositionally
- reliable/interval numerics
  user is responsible for details, not programmer friendly
- validated numerics
  IEEE standard, axiomatized floating point, complex
- Real-PCF/Shrad
  strong theoretical foundation based on λ-calculus, good for declarative programming

Our approach

Interval semantics for Floyd-Hoare Logic

Our approach

Interval semantics for Floyd-Hoare Logic

- usability by relying on iRRAM
- verification power by well established Floyd-Hoare Logic

IRRAM

- C++ library
- new datatype REAL
- operator overloading ⇒ default behaviour
- interval semantics, automatic precision computation
- in development since 15+ years

Floyd-Hoare Logic

- deductive proof system based on triples:

$$\{P\}\, C\, \{Q\}$$

  where P is pre-condition, Q post-condition, and C command

- command examples
  - empty command: $\{P\}\, \varepsilon\, \{Q\}$
  - assignment: $\{P[a/x]\}\, x := a\, \{P\}$
  - conditional: $\{P \wedge b\}\, c_0\, \{Q\}$, $\{P \wedge \neg b\}\, c_1\, \{Q\}$
    $\Rightarrow \{P\}$ **if** b **then** c_0 **else** c_1 $\{Q\}$
  - while loop: $\{P \wedge b\}\, c\, \{P\}$
    $\Rightarrow \{P\}$ **while** b **do** c $\{P\}$

What is the problem?

What is the problem?

You shall not test for equality!

(Exact Real Scrolls 1:1)

What is the problem?

You shall not test for equality!

(Exact Real Scrolls 1:1)

- equivalent to the halting problem
- forbids also comparison operators $x < y$ etc.

Two sound semantics for comparison

partial comparison

$$x > y \;=\; \begin{cases} 1 & : x > y, \\ 0 & : x < y, \\ \downarrow & : x = y \end{cases}$$

Two sound semantics for comparison

partial comparison

$$x > y \;=\; \begin{cases} 1 & : x > y, \\ 0 & : x < y, \\ \downarrow & : x = y \end{cases}$$

multivalued comparison

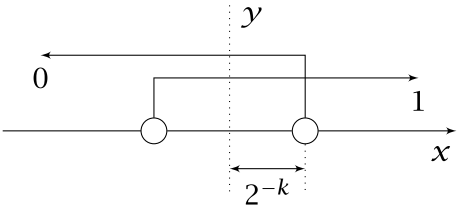
$$(x >_k y) \;=\; \begin{cases} \{1\} & : x \geq y + 2^k, \\ \{0\} & : x \leq y - 2^k, \\ \{0, 1\} & \text{else} \end{cases}$$

both supported in iRRAM

Multivalued semantics

$$(x >_k y) = \begin{cases} \{1\} & : x \geq y + 2^k, \\ \{0\} & : x \leq y - 2^k, \\ \{0, 1\} & \text{else} \end{cases}$$

$(x >_k y)$ can be understood as evaluating $x > y - 2^{-k}$ and $x < y + 2^{-k}$ in parallel and return one the value for one that has evaluated to true.



(actual implementation via `choose` function)

Three example verifications

- ilog_2
- Gaussian elimination
- simple root finding

Binary logarithm – trivial implementation

$$\operatorname{ilog}_2 : (0; \infty) \ \ni \ x \ \mapsto \ \{k \in \mathbb{Z} : 2^{k-1} < x < 2^{k+1}\} \ \in \ \mathbb{N}$$

Binary logarithm – trivial implementation

$$\operatorname{ilog}_2 : (0;\infty) \ni x \mapsto \{k \in \mathbb{Z} : 2^{k-1} < x < 2^{k+1}\} \in \mathbb{N}$$

function $\operatorname{ilog}_2(x : \mathbb{R})$ // **Require:** $x > 0$
    $\mathbb{Z} \ni l := 1$ // $\{0 < x,\ l = 1\}$
    if $x > 1$ **then** // $\{1 < x,\ l = 1\}$
        $\mathbb{R} \ni y := x$ // $\{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}$
        while $y > 2$ **do** // $\{2^{l} < y \cdot 2^{l-1} = x\}$
            $l := l + 1$; $y := y/2$ // $\{2^{l-1} < y \cdot 2^{l-1} = x\}$
        end while // $\{2^{l-1} < x = y \cdot 2^{l-1} \le 2^{l}\}$
    else // $\{0 < x \le 1 = 2^{l-1}\}$
        repeat
            $l := l - 1$ // $\{0 < x \le 2^{l}\}$
        until $x > 2^{l-1}$ // $\{2^{l-1} < x \le 2^{l}\}$
    end if
    return l // $\{2^{l-1} < x \le 2^{l}\}$
end function

Comments on the code

- no formal deduction
- given pre/post conditions give partial correctness
- Archimedian property gives total correctness

Comments on the code

- no formal deduction
- given pre/post conditions give partial correctness
- Archimedian property gives total correctness

But exchanging $\mathbb{R}$ with computable REAL,
we need to replace $>$ with $>_k$.

COMPUTABLE VERSION

```
function ilog_2(x : REAL)                          // Require: x > 0
    INTEGER ∋ l := 1                               // {0 < x, l = 1}
    if x >_{-1} 3/2 then                           // {1 < x,;l = 1}
        ℝ ∋ y := x                                 // {1 = 2^{l-1} < y = y · 2^{l-1} = x}
        while y >_{-1} 5/2 do                      // {2^l < y · 2^{l-1} = x}
            l := l + 1 ; y := y/2                  // {2^{l-1} < y · 2^{l-1} = x}
        end while                                  // {2^{l-1} < x = y · 2^{l-1} < 2^{l+1}}
    else                                           // {0 < x < 2 = 2^l}
        repeat
            l := l − 1                             // {0 < x < 2^{l+1}}
        until x >_{l-2} 3 ∗ 2^{l-2}                // {2^{l-1} < x < 2^{l+1}}
    end if
    return l                                       // {2^{l-1} < x < 2^{l+1}}
end function
```

Computable version

function $\mathrm{ilog}_2(x : \mathrm{REAL})$ // **Require:** $x > 0$
  $\mathtt{INTEGER} \ni l := 1$ // $\{0 < x,\ l = 1\}$
  if $x >_{-1} 3/2$ **then** // $\{1 < x,;l = 1\}$
    $\mathbb{R} \ni y := x$ // $\{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}$
    while $y >_{-1} 5/2$ **do** // $\{2^{l} < y \cdot 2^{l-1} = x\}$
      $l := l + 1\ ;\ y := y/2$ // $\{2^{l-1} < y \cdot 2^{l-1} = x\}$
    end while // $\{2^{l-1} < x = y \cdot 2^{l-1} < 2^{l+1}\}$
  else // $\{0 < x < 2 = 2^{l}\}$
    repeat
      $l := l - 1$ // $\{0 < x < 2^{l+1}\}$
    until $x >_{l-2} 3 * 2^{l-2}$ // $\{2^{l-1} < x < 2^{l+1}\}$
  end if
  return l // $\{2^{l-1} < x < 2^{l+1}\}$
end function

Recall $x \geq 2 \ \Rightarrow\ x >_{-1} 3/2 \ \Rightarrow\ x > 1$,
$y \geq 3 \ \Rightarrow\ y >_{-1} 5/2 \ \Rightarrow\ y > 2$ and, for $l \leq 1$,
$x \geq 4 \cdot 2^{l-2} \ \Rightarrow\ x >_{l-2} 3 \cdot 2^{l-2} \ \Rightarrow\ x > 2 \cdot 2^{l-2}$.

Other examples

Gaussian elimination

Given a $n \times m$ matrix A, returns a row echelon form.

Gaussian elimination

Given a $n \times m$ matrix A, returns a row echelon form.

Problems with standard diagonalization procedure

- not compatible with REAL if $\mathrm{rank}(A)$ is not given (Kihara, Pauly *Dividing by zero – how bad is it, really?*, 2016)
- Partial pivoting is not computable over REAL, even if $\mathrm{rank}(A)$ is given

Gaussian elimination

Given a $n \times m$ matrix A, returns a row echelon form.

Problems with standard diagonalization procedure

- not compatible with REAL if $\mathrm{rank}(A)$ is not given (Kihara, Pauly *Dividing by zero – how bad is it, really?*, 2016)
- Partial pivoting is not computable over REAL, even if $\mathrm{rank}(A)$ is given
- …but with full pivoting and given rank it is computable!

Full pivot search using multivalued test

```
Procedure choosePivot
(n, k : INTEGER, B[n, n] : REAL, var pi, pj : INTEGER)
var i, j : INTEGER ;   var s, t : REAL ;   t := 0
                        // Require:  B′ := B[k ... n, k ... n] not all zero
              // Calculate max absolute value of square sub-matrix B′:
for i := k to n do
    for j := k  to n  do t := max (t, abs(B[i, j]))  end for
end for
          // Find index of some element whose abs exceeds half of t:
for i := k to n do
    for j := k to n do
        s := abs(B[i, j]) ; if choose(s > t/2 , t > s)  =  1  then
    pi := i ; pj := j  end if
    end for
end for       // Return index of some element of at least half the
    maximum absolute value in B′.
```

Simple root finding

Given continuous $f : [0, 1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$ and a unique root as black box as well as $n \in \mathbb{Z}$, produce some $c \in [0, 1]$ such that $|x - c| \leq 2^n$.

Simple root finding

Given continuous $f : [0,1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$ and a unique root as black box as well as $n \in \mathbb{Z}$, produce some $c \in [0,1]$ such that $|x - c| \leq 2^n$.

Bisection may fail when the testing value is exactly the root.

Use trisection

Conclusions and future work

- applied Floyd-Hoare logic for imperative, exact real computation
- three examples of verification
- no deductive formal proof, but possible

Conclusions and future work

- applied Floyd-Hoare logic for imperative, exact real computation
- three examples of verification
- no deductive formal proof, but possible
- extend to full matrix diagonalization
- proof assistants for automated verification
- computational complexity of multi-valued algebraic real verification

Conclusions and future work

- applied Floyd-Hoare logic for imperative, exact real computation
- three examples of verification
- no deductive formal proof, but possible
- extend to full matrix diagonalization
- proof assistants for automated verification
- computational complexity of multi-valued algebraic real verification

Thanks