Verification in Real Computation

Norbert Preining\textsuperscript{1}
(joint work with Norbert Müller, Sewon Park, Martin Ziegler)

\begin{center}
\textbf{Accelia}
\end{center}

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

\textsuperscript{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

Verifikation 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: \( \{P \land b\} c_0 \{Q\}, \{P \land \neg b\} c_1 \{Q\} \)
    \[ \Rightarrow \{P\} \textbf{if} \ b \ \textbf{then} \ c_0 \ \textbf{else} \ c_1 \ \{Q\} \]
  - while loop: \( \{P \land b\} c \{P\} \)
    \[ \Rightarrow \{P\} \textbf{while} \ b \ \textbf{do} \ c \ \{P\} \]
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
\textbf{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.

(\textit{actual implementation via} \texttt{choose} function)
THREE EXAMPLE VERIFICATIONS

- $\log_2$
- Gaussian elimination
- simple root finding
**Binary logarithm – trivial implementation**

\[ \text{ilog}_2 : (0; \infty) \ni x \mapsto \{ k \in \mathbb{Z} : 2^{k-1} < x < 2^{k+1} \} \in \mathbb{N} \]
**Binary logarithm – trivial implementation**

\[
\text{ilog}_2 : (0; \infty) \ni x \rightarrow \{ k \in \mathbb{Z} : 2^{k-1} < x < 2^{k+1} \} \in \mathbb{N}
\]

1: `function` `ilog_2(x : \mathbb{R})` // **Require:** \( x > 0 \)
2: \( \mathbb{Z} \ni l := 1 \) // \( \{0 < x, l = 1\} \)
3: `if` \( x > 1 \) `then` // \( \{1 < x, l = 1\} \)
4: \( \mathbb{R} \ni y := x \) // \( \{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\} \)
5: `while` \( y > 2 \) `do` // \( \{2^l < y \cdot 2^{l-1} = x\} \)
6: \hspace{1em} `l := l + 1 ; y := y / 2` // \( \{2^{l-1} < y \cdot 2^{l-1} = x\} \)
7: `end while` // \( \{2^{l-1} < x = y \cdot 2^{l-1} \leq 2^l\} \)
8: `else` // \( \{0 < x \leq 1 = 2^{l-1}\} \)
9: `repeat` // \( \{0 < x \leq 2^l\} \)
10: \hspace{1em} `l := l - 1` // \( \{2^{l-1} < x \leq 2^l\} \)
11: `until` \( x > 2^{l-1} \) // \( \{2^{l-1} < x \leq 2^l\} \)
12: `end if` // \( \{2^{l-1} < x \leq 2^l\} \)
13: `return` `l` // \( \{2^{l-1} < x \leq 2^l\} \)
14: `end function`
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 $\mathbb{REAL}$, we need to replace $>$ with $\geq$. 
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 \( \geq_k \).
**Computable version**

1: function $\text{ilog}_2(x : \text{REAL})$  
2: \quad \text{INTEGER } \ni l := 1  
3: \quad \text{if } x > -1 \ 3/2 \ \text{then}  
4: \quad \quad \mathbb{R} \ni y := x  
5: \quad \quad \text{while } y > -1 \ 5/2 \ \text{do}  
6: \quad \quad \quad l := l + 1 ; y := y/2  
7: \quad \quad \text{end while}  
8: \quad \text{else}  
9: \quad \quad \text{repeat}  
10: \quad \quad \quad l := l - 1  
11: \quad \quad \quad \text{until } x > l-2 \ 3 \times 2^{l-2}  
12: \quad \text{end if}  
13: \quad \text{return } l  
14: \text{end function}

// Require: $x > 0$
// \{0 < x, l = 1\}
// \{1 < x, \ l = 1\}
// \{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}
// \{2^l < y \cdot 2^{l-1} = x\}
// \{2^{l-1} < y \cdot 2^{l-1} = x\}
// \{2^{l-1} < x = y \cdot 2^{l-1} < 2^{l+1}\}
// \{0 < x < 2 = 2^l\}
// \{0 < x < 2^{l+1}\}
// \{2^{l-1} < x < 2^{l+1}\}
// \{2^{l-1} < x < 2^{l+1}\}
**Computable Version**

1: function $\text{ilog}_2(x : \text{REAL})$  
2: \hspace{1em} INTEGER $\ni l := 1$  
3: \hspace{1em} if $x >_{-1} 3/2$ then  
4: \hspace{2em} $\exists y := x$  
5: \hspace{2em} while $y >_{-1} 5/2$ do  
6: \hspace{3em} $l := l + 1$ ; $y := y/2$  
7: \hspace{2em} end while  
8: \hspace{1em} else  
9: \hspace{2em} repeat  
10: \hspace{3em} $l := l - 1$  
11: \hspace{2em} until $x >_{l-2} 3 \cdot 2^{l-2}$  
12: \hspace{1em} end if  
13: \hspace{1em} return $l$  
14: end function

// Require: $x > 0$
// $\{0 < x, l = 1\}$
// $\{1 < x, l = 1\}$
// $\{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}$
// $\{2^l < y \cdot 2^{l-1} = x\}$
// $\{2^{l-1} < y \cdot 2^{l-1} = x\}$
// $\{2^{l-1} < x = y \cdot 2^{l-1} < 2^{l+1}\}$
// $\{0 < x < 2 = 2^l\}$
// $\{0 < x < 2^{l+1}\}$
// $\{2^{l-1} < x < 2^{l+1}\}$
// $\{2^{l-1} < x < 2^{l+1}\}$
// $\{2^{l-1} < x < 2^{l+1}\}$

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 $\text{rank}(A)$ is not given (Kihara, Pauly *Dividing by zero – how bad is it, really?*, 2016)

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

- Partial pivoting is not computable over REAL, even if \( \text{rank}(A) \) is given

- …but with full pivoting and given rank it is computable!
Full pivot search using multivalued test

Procedure choosePivot
\( (n, k : \text{INTEGER}, B[n,n] : \text{REAL}, \text{var } pi, pj : \text{INTEGER}) \)

1: \text{var } i, j : \text{INTEGER}; \text{var } s, t : \text{REAL}; t := 0

2: \text{for } i := k \text{ to } n \text{ do}
3: \quad \text{for } j := k \text{ to } n \text{ do } t := \max (t, \text{abs}(B[i,j])) \text{ end for}
4: \text{end for}

5: \text{for } i := k \text{ to } n \text{ do}
6: \quad \text{for } j := k \text{ to } n \text{ do}
7: \quad \quad s := \text{abs}(B[i,j]); \text{if } \text{choose}(s > t/2 \ , \ t > s) = 1 \text{ then}
8: \quad \quad \quad pi := i \ ; \ pj := j \text{ end if}
9: \quad \text{end for}
10: \text{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 $|x - c| \leq 2^n$. 

Bisection may fail when the testing value is exactly the root. Use trisection.
**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

Thanks
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