CCL – an approach to verifiable multi-valued computations on $\mathbb{R}$

Franz Brauße\textsuperscript{1}  Norbert Th. Müller\textsuperscript{1}  Robert Rettinger\textsuperscript{2}

Universität Trier, Germany
Hochschule Dortmund, Germany

iRRAM/MPFR/MPC workshop, Schloß Dagstuhl, 2018-04-20
Observations

- C++ not formally verifiable (yet?).
- Programs in iRRAM need to run in its C++ loop and throw C++ exceptions.

\[ \mapsto \] iRRAM-algorithms on reals not verifiable.

- don’t want to think in convergent sequences!
- aim twofold:
  - compile to iRRAM \(\mapsto\) fast execution
  - compile to Coq \(\mapsto\) verified exact real algorithm
- computable constructs \(\mapsto\) operations have multi-valued semantics, especially *composition*!
- semantics of expressions \(e\) defined through multi-valued function
  \[ \vdash e : \Gamma \Rightarrow \Gamma \times \text{dom type } e \text{ directly in Coq} \]

\(^1\)Based on Clerical by Bauer, Park, Simpson,: [https://github.com/andrejbauer/clerical](https://github.com/andrejbauer/clerical)
Multi-valued functions on \( \mathbb{R} \)

When computing \( f : \mathbb{R} \to \mathbb{R} \), in order to decide on an approximation of \( f(x) \), only finitely many bits of \( x \) are known. Thus, continuity:

\[
\forall x, \varepsilon > 0 \exists \delta > 0 : f((x \pm \delta)) \subseteq (f(x) \pm \varepsilon)
\]

(Computable) way around discontinuities: multi-valued ‘functions’:

- \( f : \mathbb{R} \Rightarrow A \) is a relation on \( \mathbb{R} \times A \), really.
- but: composition \( (f \circ g)(x) \) only defined iff \( \emptyset \neq g(x) \subseteq \text{dom} \ f \).

**Example: Test for \( x \) close to zero**

E.g. compute \( x \leq 2^{-n} \) multi-valued as \( x \mapsto \begin{cases} 
\{1\} & x < 2^{-n}/2 \\
\{0\} & x > 2^{-n} \\
\{0, 1\} & x \in [2^{-n}/2, 2^{-n}] 
\end{cases} \)

During computation, just one of the values \( \in f(x) \) is selected in an inherently implementation-defined manner.
CCL

- intro variables: `let x := e in f`
- loops: `while b do e`
- assignments: `x := e`
- + std arithmetic

- `'lim' n =>' e`
  - `n` new variable, `e` an expression depending on `n`
  - `e` computes an approx. to accuracy $2^{-n}$ of limit

- `'case' b1 =>' e1 || b2 =>' e2`
  - at least one of `b1`, `b2` must be `'True'`, otherwise undefined
  - more than one branch `'True'` multi-valued execution semantics
  - every state transition is potentially multi-valued

- Function contracts / expression invariants as annotations
  - translated to proof obligations
  - documentation to users
Limit expression – example: Heron’s method

\{ \forall x : \text{Coq.Reals.R}, \text{abs } x = \text{Coq.Reals.Rabs } x \}\}

external \text{abs(Real) } \rightarrow \text{Real} \\
external \text{bounded(Real, Int) } \rightarrow \text{Bool}

\{ \forall x : \text{Coq.Reals.R}, x \geq 0 \Rightarrow \text{sqrt } x = \text{Coq.Reals.Rsqrt } x \}\}

function \text{sqrt(Real } x): \\
\quad \lim n \Rightarrow \\
\quad \quad \text{var } y := 1.0 \text{ in (} \\
\quad \quad \quad \text{var } z := x/y \text{ in (} \\
\quad \quad \quad \quad \text{while } !\text{bounded(abs(y-z), n)} \\
\quad \quad \quad \quad \quad \text{do (} \\
\quad \quad \quad \quad \quad \quad \quad \quad y := (y+z)/2.0; \\
\quad \quad \quad \quad \quad \quad \quad \quad z := x/y \\
\quad \quad \quad \quad \quad \quad ) \\
\quad \quad \quad \); y) \\
\quad \); y)

\text{do } \\
\quad \text{sqrt(2.0)}
Case expression – intro multi-valuedness

- Multi-valued test for \( x \) bounded by \( 2^{-n} \):

\[
\{ \forall x \in \mathbb{N}, x < (\frac{1}{2^n}) \Rightarrow \text{bounded} x n = '\text{True}' \}
\]

function bounded(Real x, Int n):
  var eps := 2.0^{\text{real}(-n)} in
  case x < eps => True
  || x > eps/2.0 => False
end

- single-valued absolute value via limit of a multi-valued sequence

\[
\{ \forall x : \text{Coq.ZArith.Z, real } x = \text{Coq.Reals.IZR } x \}
\]

external real(Int) -> Real

\[
\{ \forall x, \text{abs } x = \text{Coq.Reals.Rabs } x \}
\]

function abs(Real x):
  lim n =>
  var eps := 2.0^{\text{real}(-n)} in
  case x < eps => -x
  || x > -eps => x
end

\(^2\text{IZR injects Coq's integers into Coq's axiomatic Reals}\)
Thank you!


CCL → iRRAM: https://github.com/fbrausse/cclerical
(** Multi-valued functions **)  
Definition mv_f (A B : Type) := A -> B -> Prop.

Definition mv_compose {A B C : Type} (f : mv_f A B) (g : mv_f B C)
  : mv_f A C
  := fun a c => (forall b : B, f a b -> exists c : C, g b c) /
                     exists b : B, f a b /\n                     g b c.

(** vdash relates "previous" contexts to "later" contexts and a result value **)  
Definition vdash {sigma : scope} {t : ccl_type} :=
  context sigma -> context sigma -> ccl_dom t -> Prop.

Definition case {sigma : scope} {t : ccl_type}
  (b0 b1 : @vdash sigma Kleenean)
  (e0 e1 : @vdash sigma t)
  : vdash := fun g g' v => b0 g g (kl Bool true) /
                        e0 g g' v / b1 g g (kl Bool true) /
                        e1 g g' v.

Definition lim {sigma : scope} {t : ccl_type} (id : string)
  (e : @vdash (mk_scope_record id Integer :: sigma) t)
  := fun g g' v
     => g = g' /
        let (d,_) := ccl_met t in
        let s := fun n w => let h := (g,Z.of_nat n) in e h h w in
        mv_total s /
        mv_converges_fast (ccl_dom t) d s v.