Proving bounds on real-valued functions with computations

Guillaume Melquiond

Mathematical Components
INRIA–Microsoft Research

2008-08-12

Verifying inequalities, an example

While certifying an algorithm for preventing airplane collision, Carreño and Muñoz had to formally prove:

> *A plane flying at 250 knots and with a bank angle of 35° has a turn rate of at least 3° each second.*

In other words,

$$\frac{3\pi}{180} \leq \frac{g}{v} \cdot \tan\left(\frac{35\pi}{180}\right)$$

with $g = 9.8\ m/s^2$ and $v = 250 \cdot \frac{514}{1000}\ m/s$.

Verifying inequalities, an example

The inequality is trivially true since

$$\frac{3\pi}{180} \approx 0.052 \quad \text{and} \quad \frac{g}{v} \cdot \tan\left(\frac{35\pi}{180}\right) \approx 0.053.$$

Verifying inequalities, an example

The inequality is trivially true since

$$\frac{3\pi}{180} \approx 0.052 \quad \text{and} \quad \frac{g}{v} \cdot \tan\left(\frac{35\pi}{180}\right) \approx 0.053.$$

Proved in PVS by Lester and Muñoz with interval arithmetic.

What about Coq?

Verifying inequalities, an example

The inequality is trivially true since

$$\frac{3\pi}{180} \approx 0.052 \quad \text{and} \quad \frac{g}{v} \cdot \tan\left(\frac{35\pi}{180}\right) \approx 0.053.$$

Proved in PVS by Lester and Muñoz with interval arithmetic.

What about Coq?

```
Goal
  let v := 250 * (514 / 1000) in
  3 * pi / 180 <= g / v * tan (35 * pi / 180).
Proof.
  apply Rminus_le. (* transform into a - b <= 0        *)
  interval.        (* prove by interval computations *)
Qed.
```

Interval arithmetic

A few words about interval arithmetic:

- $x \in [a, b] \wedge y \in [c, d] \Rightarrow x + y \in [a + c, b + d] =: [a, b] + [c, d]$
- $x \in [a, b] \Rightarrow \exp x \in [\exp a, \exp b] =: \exp[a, b]$

Interval arithmetic

A few words about interval arithmetic:

- $x \in [a, b] \wedge y \in [c, d] \Rightarrow x + y \in [a + c, b + d] =: [a, b] + [c, d]$
- $x \in [a, b] \Rightarrow \exp x \in [\exp a, \exp b] =: \exp[a, b]$

Interval-based proofs are simple and sound,
but they rely on numerically-intensive computations.

E.g. Hales' proof of Kepler's conjecture depends on C programs.

Interval arithmetic

A few words about interval arithmetic:

- $x \in [a, b] \wedge y \in [c, d] \Rightarrow x + y \in [a + c, b + d] =: [a, b] + [c, d]$
- $x \in [a, b] \Rightarrow \exp x \in [\exp a, \exp b] =: \exp[a, b]$

Interval-based proofs are simple and sound,
but they rely on numerically-intensive computations.

E.g. Hales' proof of Kepler's conjecture depends on C programs.

Question: How to efficiently perform numerical computations inside a formal system?

Outline

Outline

1. Introduction

2. Conversion and computations
   - Example: Peano's arithmetic
   - Type theory and conversion
   - Evaluating expressions
   - Evaluating propositions
   - Implementation details

3. Proofs on real-valued expressions

4. Conclusion

Example: Peano's arithmetic

Inductive definition of natural numbers:

```
type nat = O | S of nat    (* 5 = SSSSSO *)
```

Axioms for addition:

`add0`: $\forall b \quad \mathrm{O} + b = b$

`addS`: $\forall a \forall b \quad (\mathrm{S}\ a) + b = a + (\mathrm{S}\ b)$

Example: Peano's arithmetic

Deductive proof of $4 + (2 + 3) = 9$: (9 steps)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \texttt{addS} \times 4}{4 + 5 = 9}}{4 + (0 + 5) = 9}\ \texttt{add0}}{4 + (1 + 4) = 9}\ \texttt{addS}}{4 + (2 + 3) = 9}\ \texttt{addS}
$$

$$
\dfrac{\dfrac{}{9 = 9}\ \text{reflexivity}}{0 + 9 = 9}\ \texttt{add0}
$$

Introducing computations into proofs

Recursive definition of addition:

```
let rec plus x y =
  match x with
  | O -> y
  | S x' -> plus x' (S y)
```

Lemma plus_xlate: $\forall a \forall b \quad a + b = \texttt{plus}\ a\ b$

Introducing computations into proofs

Recursive definition of addition:

```
let rec plus x y =
  match x with
  | O -> y
  | S x' -> plus x' (S y)
```

Lemma plus_xlate: $\forall a \forall b \quad a + b = \texttt{plus}\ a\ b$

Proof of $4 + (2 + 3) = 9$: (4 steps)

$$\dfrac{\dfrac{\dfrac{\overline{9 = 9}\ \text{reflexivity}}{\texttt{plus}\ 4\ (\texttt{plus}\ 2\ 3) = 9}\ ???}{4 + (\texttt{plus}\ 2\ 3) = 9}\ \texttt{plus\_xlate}}{4 + (2 + 3) = 9}\ \texttt{plus\_xlate}$$

Type theory and conversion

Curry-Howard correspondence and type theory:

1. Proposition A holds if the type A is inhabited.
2. Convertible types have the same inhabitants.

$$\frac{p : A}{p : B}\ A \equiv_\beta B$$

Type theory and conversion

Curry-Howard correspondence and type theory:

1. Proposition A holds if the type A is inhabited.
2. Convertible types have the same inhabitants.

$$\frac{p : A}{p : B}\ A \equiv_\beta B$$

Proof of $4 + (2 + 3) = 9$: (4 steps)

$$\cfrac{\cfrac{\cfrac{\cfrac{}{p : 9 = 9}\ \text{reflexivity}}{p : \texttt{plus}\ 4\ (\texttt{plus}\ 2\ 3) = 9}\ \beta\text{-reduction}}{4 + (\texttt{plus}\ 2\ 3) = 9}\ \texttt{plus\_xlate}}{4 + (2 + 3) = 9}\ \texttt{plus\_xlate}$$

Encoding expressions

Inductive definition of expressions on natural numbers:

```
type expr = Nat of nat | Add of expr * expr
let rec interp_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    (interp_expr x) "+" (interp_expr y)
```

Proof of $4 + (2 + 3) = 9$:

$$\cfrac{\cfrac{???}{\texttt{interp\_expr}\ (\texttt{Add}\ (\texttt{Nat}\ 4, \texttt{Add}\ (\texttt{Nat}\ 2, \texttt{Nat}\ 3))) = 9}}{4 + (2 + 3) = 9}\ \beta\text{-reduction}$$

Evaluating expressions

Evaluating expressions on natural numbers:

```
let rec eval_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    plus (eval_expr x) (eval_expr y)
```

Lemma expr_xlate: $\forall e$ interp_expr $e =$ eval_expr e

Evaluating expressions

Evaluating expressions on natural numbers:

```
let rec eval_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    plus (eval_expr x) (eval_expr y)
```

Lemma expr_xlate: $\forall e \quad \texttt{interp\_expr}\ e = \texttt{eval\_expr}\ e$

Proof of $4 + (2 + 3) = 9$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{9 = 9}\ \text{reflexivity}}{\texttt{eval\_expr (Add (Nat 4,}\ldots\texttt{))} = 9}\ \beta\text{-reduction}}{\texttt{interp\_expr (Add (Nat 4,}\ldots\texttt{))} = 9}\ \texttt{expr\_xlate}}{4 + (2 + 3) = 9}\ \beta\text{-reduction}$$

Relational operators

Equality is usually a native concept, while comparisons are not.

Comparing natural numbers:

```
let rec le x y =
  match x, y with
  | O   , _    -> true
  | S _ , O    -> false
  | S x', S y' -> le x' y'
```

Lemma: $\forall a \forall b \quad \texttt{le}\ a\ b = \mathit{true} \Leftrightarrow a \leq b$

Encoding comparisons

Inductive definition of relations on natural expressions:

```
type prop = Le of expr * expr
let interp_prop p =
  match p with
  | Le (x, y) ->
    (interp_expr x) "<=" (interp_expr y)
let eval_prop p =
  match p with
  | Le (x, y) -> le (eval_expr x) (eval_expr y)
```

Encoding comparisons

Inductive definition of relations on natural expressions:

```
type prop = Le of expr * expr
let interp_prop p =
  match p with
  | Le (x, y) ->
    (interp_expr x) "<=" (interp_expr y)
let eval_prop p =
  match p with
  | Le (x, y) -> le (eval_expr x) (eval_expr y)
```

Proof of $4 + (2 + 3) \leq 5 + 6$:

$$\dfrac{\dfrac{\dfrac{}{\mathit{true} = \mathit{true}}\ \text{reflexivity}}{\texttt{eval\_prop (Le (Add } \ldots\texttt{, Add } \ldots\texttt{))} = \mathit{true}}\ \beta\text{-reduction}}{\dfrac{\texttt{interp\_prop (Le (Add } \ldots\texttt{, Add } \ldots\texttt{))}}{4 + (2 + 3) \leq 5 + 6}\ \beta\text{-reduction}}\ \texttt{prop\_xlate}$$

Implementation details

All the proofs are identical, except for the inductive object.

This object is the syntax tree of the proposition;
It has to be computed by an external oracle.

(= Simple parser written in the meta-language of Coq proofs.)

Implementation details

All the proofs are identical, except for the inductive object.

This object is the syntax tree of the proposition;
It has to be computed by an external oracle.

(= Simple parser written in the meta-language of Coq proofs.)

Proof verification:

$$\frac{\textit{true} = \textit{true}}{\texttt{eval\_prop}\ \ldots = \textit{true}} \left\langle \begin{array}{l}\text{typechecking fails}\\ \text{if the user is wrong}\end{array}\right.$$

$$\vdots$$

$$\frac{\texttt{interp\_prop}\ (\texttt{Le}\ldots)}{4 + (2 + 3) \le 5 + 6} \left\langle \begin{array}{l}\text{typechecking fails}\\ \text{if the oracle is wrong}\end{array}\right.$$

Outline

1. Introduction
2. Conversion and computations
3. Proofs on real-valued expressions
   - Enclosures of real-valued expressions
   - Interval arithmetic
   - Improving bounds
4. Conclusion

Enclosures of real-valued expressions

Definition (Real-valued expression)

Straight-line program of

- variables: x, y, ...
- unary operators: $-\square$, $\sqrt{\square}$, $|\square|$, $\square^2$, $\square^{-1}$,
- binary operators: $+$, $-$, $\times$, $\div$,
- transcendental functions: cos, sin, tan, arctan.

Straight-line programs are directed acyclic graphs;
Common sub-expressions are shared and evaluated only once.

Enclosures of real-valued expressions

Definition (Expression enclosures)

$$a \le f(x, y, \ldots) \le b$$

with a and b floating-point numbers $(m \cdot 2^e;\ m, e \in \mathbb{Z})$ or $\pm\infty$.

(Note: Undefined values, e.g. 1/0, are not bounded.)

Goal: Automatically prove the following proposition

$$\frac{a_x \le x \le b_x \quad a_y \le y \le b_y \quad \ldots}{a \le f(x, y, \ldots) \le b}$$

Interval arithmetic

Floating-point interval evaluations can serve as proofs of bounds, as long as they satisfy the containment property:

$$x \in I_x \wedge y \in I_y \quad \Longrightarrow \quad x \diamond y \in I_x \diamond I_y$$

for $\diamond \in \{+, -, \times, \div\}$. Also for unary functions: $\sqrt{\cdot}, \sin, \ldots$

Interval arithmetic

Floating-point interval evaluations can serve as proofs of bounds, as long as they satisfy the containment property:

$$x \in I_x \wedge y \in I_y \quad \Longrightarrow \quad x \diamond y \in I_x \diamond I_y$$

for $\diamond \in \{+, -, \times, \div\}$. Also for unary functions: $\sqrt{\cdot}, \sin, \ldots$

Theorem (Containment)

Given a SLP *prog* defined on expressions *inputs*,
if $\forall i,\ input_i \in range_i$, then

$$\text{eval}(prog, inputs) \in \text{eval}(prog, ranges)$$

Interval arithmetic

Floating-point interval evaluations can serve as proofs of bounds, as long as they satisfy the containment property:

$$x \in I_x \wedge y \in I_y \quad \Longrightarrow \quad x \diamond y \in I_x \diamond I_y$$

for $\diamond \in \{+, -, \times, \div\}$. Also for unary functions: $\sqrt{\cdot}, \sin, \ldots$

Theorem (Containment 2)

Given a SLP *prog* defined on expressions *inputs*,
if $\forall i,\ input_i \in range_i$, then

$\texttt{subset}\ (\texttt{eval}\ prog\ ranges)\ output = true \Rightarrow$
$\texttt{eval}\ prog\ inputs \in output$

Sharpening intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

For $x \in [0,1]$, $\quad x + (-x) \in [0,1] + [-1,0] = [-1,1]$.

Sharpening intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

For $x \in [0,1]$, $\quad x + (-x) \in [0,1] + [-1,0] = [-1,1]$.

Implemented improvements:

1. Bisection:
   Recursively split the domain into smaller sub-domains, until the proposition is proved on all the sub-domains.

Sharpening intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

$$\text{For } x \in [0,1], \quad x + (-x) \in [0,1] + [-1,0] = [-1,1].$$

Implemented improvements:

1. Bisection:
   Recursively split the domain into smaller sub-domains, until the proposition is proved on all the sub-domains.

2. First-order approximation:
   $f(x_0 + h) = f(x_0) + h \cdot f'(x_0 + \xi) \quad \text{with } \xi \in [0,h] \text{ (or } [h,0]),$
   so $\forall x \in X, \quad f(x) \in f(x_0) + (X - x_0) \cdot f'(X) \quad \text{with } x_0 \in X.$

Outline

Verifying inequalities, another example

Global positioning requires knowing the local radius of Earth:

$$r_p(\phi) = \frac{a}{\sqrt{1 + (1-f)^2 \cdot \tan^2 \phi}}$$

This can be approximated by a degree-5 polynomial P with single-precision coefficients: $P(\phi_m^2 - \phi^2) = \tilde{r_p}(\phi)$.

$$P(x) = \frac{4439091}{4} + x \cdot \left(\frac{9023647}{4} + x \cdot \left(\cdots \frac{6661427}{131072} \right) \right)$$

Goal: Prove $\left| \frac{r_p(\phi) - \tilde{r_p}(\phi)}{r_p(\phi)} \right| \le 23 \cdot 2^{-24}$ when $0 \le \phi \le \phi_m = \frac{715}{512}$.

Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, and Muñoz, in PVS (2005).

   An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval.

   Verification: several hours on a 48-core parallel computer.

Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, and Muñoz, in PVS (2005).

   An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval.

   Verification: several hours on a 48-core parallel computer.

2. In Coq (2008), a few minutes on a laptop computer.

Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, and Muñoz, in PVS (2005).

   An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval.

   Verification: several hours on a 48-core parallel computer.

2. In Coq (2008), a few minutes on a laptop computer.

   Differences: No oracle, but floating-point numbers.

Verifying inequalities, another example

```
Require Import Reals.
Require Import tactics.
Open Local Scope R_scope.

Definition a := 6378137.
Definition f := 1000000000/298257223563.
Definition umf2 := (1 - f)².
Definition max := 715/512.
Definition rp phi := a / sqrt (1 + umf2 * (tan phi)²).
Definition arp phi :=
  let x := max² - phi² in
  4439091/4 + x * (
    9023647/4 + x * (
      13868737/64 + x * (
        13233647/2048 + x * (
          -1898597/16384 + x *
            (-6661427/131072))))).

Goal
  forall phi, 0 <= phi <= max ->
  Rabs ((rp phi - arp phi) / rp phi) <= 23/16777216.
Proof.
  unfold rp, arp, umf2, a, f, max. intros.
  Time interval with (i_bisect_diff phi, i_nocheck).  (*  4s *)
Time Qed.                                             (* 96s *)
```

Tactic parameters:

- bisection and order-1 evaluation on ϕ,
- minimal relative width of intervals: 2^{-15},
- check delayed at `Qed` time,
- precision: 30 bits.

What about Gappa?

Toy implementation of cosine in C

```
/*@ requires |x| ≤ 2^-5
 *@ ensures |\result − cos x| ≤ 2^-23 */
float mycos(float x)
{ return 1 - x * x * 0.5; }
```

What about Gappa?

Toy implementation of cosine in C

```
/*@ requires |x| ≤ 2^-5
 *@ ensures |\result − cos x| ≤ 2^-23 */
float mycos(float x)
{ return 1 - x * x * 0.5; }
```

When certifying the function, Why generates a proof obligation:

```
Theorem mycos17:
  forall x, Rabs x <= 1/32 ->
  Rabs (round (1 - round (round (x*x) * (5/10)))
         - cos x) <= powerRZ 2 (-23).
```

Impossible for Gappa: what is cos?

Impossible for interval: highly correlated, yet not even continuous.

What about Gappa?

```
Theorem mycos17:
  forall x, Rabs x <= 1/32 ->
  Rabs (round (1 - round (round (x*x) * (5/10)))
        - cos x) <= powerRZ 2 (-23).
Proof.
  intros.
  assert (Rabs ((1 - x*x * (5/10)) - cos x)
            <= 7/134217728)
    by interval with (i_bisect_diff x).
  gappa.
Qed.
```

Conclusion

This prover is:

- Pure library. No modifications to Coq are needed.
- Implemented as a computable boolean function with an associated correctness theorem.

  This is a pure Coq λ-term!

It relies on standard numerical techniques:

- multi-precision floating-point arithmetic,
- interval arithmetic,
- automatic differentiation,
- bisection.

Questions?

Mail: `guillaume.melquiond@inria.fr`

Web: `http://www.msr-inria.inria.fr/soft/coq-interval/`