Generating formally certified bounds on values and round-off errors

Marc DAUMAS and Guillaume MELQUIOND

LIP Computer Science Laboratory

UMR 5668 CNRS–ENS Lyon–INRIA–UCBL

Lyon, France

Introduction

- Formal certification of software is spreading. But investigating numerical problems is still limited.
- Certifying a numerical code is tedious and error-prone.
- Tools for naive users are necessary.

Introduction

- Formal certification of software is spreading. But investigating numerical problems is still limited.
- Certifying a numerical code is tedious and error-prone.
- Tools for naive users are necessary.
- Certification goals:
  - variables are bounded (e.g. no square root of a negative number, etc), so that no exceptional behavior is triggered,
  - round-off errors are contained, so that the final result of an algorithm is sufficiently accurate.

Introduction

- Certifying numerical properties on a program f.

$$\begin{array}{ccccc} x \in & [a,b] & \Longrightarrow & \widetilde{f(x)} \in & [c,d] \\ & \uparrow & \uparrow & & \uparrow \\ & \text{given} & \text{Formal proof} & & \text{given or} \\ & & \text{automatically generated} & & \text{deduced} \end{array}$$

Introduction

- Certifying numerical properties on a program f.

$$x \in \quad [a,b] \quad\quad \Longrightarrow \quad\quad \widetilde{f(x)} \in \quad [c,d]$$

$\uparrow$ given — $\uparrow$ Formal proof automatically generated — $\uparrow$ given or deduced

- Constraints on f:
  - loops are static,
  - no branching,
  - equivalent to single assignment.
- These constraints are handled by other classical certification tools.

Outline

- The example of an aeronautical application.
  Robustness is critical, the implementation needs to be certified. Yet the algorithm is mathematically simple, its study should not require much human expertise.
- Methods used to analyze numerical programs.
  Bounds on variable are computed through interval arithmetic, and round-off errors are propagated through forward error analysis.
- Formal proofs.
  Why a formal proof? Conclusion and perspectives.

Safety distances between aircrafts

- Conversion from geodetic to euclidean data.

$$
\begin{aligned}
\phi_0 &\leftarrow (\phi_1 + \phi_2)/2 \\
r_{p0} &\leftarrow R_p(\phi_0) \\
s_{N1} &\leftarrow v_{N1}/R_m(\phi_1) \\
s_{N2} &\leftarrow v_{N2}/R_m(\phi_2) \\
p_x &\leftarrow (\lambda_1 - \lambda_2) * r_{p0} \\
p_y &\leftarrow (\phi_1 - \phi_2) * R_m(\phi_0) \\
v_{x1} &\leftarrow v_{E1} * r_{p0}/R_p(\phi_1 + s_{N1} * t_r) \\
v_{x2} &\leftarrow v_{E2} * r_{p0}/R_p(\phi_2 + s_{N2} * t_r)
\end{aligned}
$$

Safety distances between aircrafts

- Earth local radii (WGS84):

$$
\begin{aligned}
R_p(\phi) &= \frac{a}{1 + (1 - f)^2 \tan^2 \phi} \\
R_m(\phi) &= \frac{a(1 - e^2)}{(1 - e^2 \sin^2 \phi)^{3/2}}
\end{aligned}
$$

Safety distances between aircrafts

- Earth local radii (WGS84):

$$
\begin{aligned}
R_p(\phi) &= \frac{a}{1 + (1-f)^2 \tan^2 \phi} \\
R_m(\phi) &= \frac{a(1-e^2)}{(1 - e^2 \sin^2 \phi)^{3/2}}
\end{aligned}
$$

- Use of approximations (introduces truncation errors):

$$
\begin{aligned}
x &\leftarrow 511225 \times 2^{-18} - \phi^2 \\
\hat{R}_p(\phi) &\leftarrow 4439091 \cdot 2^{-2} + x \times (9023647 \cdot 2^{-2} + x \times (\\
&\quad 13868737 \cdot 2^{-6} + x \times (13233647 \cdot 2^{-11} + x \times (\\
&\quad -1898597 \cdot 2^{-14} + x \times (-6661427 \cdot 2^{-17})))))
\end{aligned}
$$

Containing the error

- Taking into account measurement errors on input, truncation errors on algorithm, and rounding errors during computations.
- Numerical properties of the program are described by a formal proof, automatically certified with the Coq proof checker.

Containing the error

- Parts of the Coq script:

```
Variable V_vx_2:  float.
Hypothesis H_vx_2:  (Div_float V_T8 V_Rp_2_c0 V_vx_2).
Definition B_16 := (Eint_bound Cl_16 Cu_16 V_Rp_a3).
...
Definition Ce_195 := (Float ‘1‘ ‘-11‘).
Lemma E_195:  ... -> B_16 -> ... ->
             (Eint_error Ce_195 V_vx_2 R_vx_2).
```

- The last lemma can be read as follow.
  - Hypotheses:
    $H_{vx2} : \tilde{v}_{x2} \leftarrow \tilde{t}_8 \oslash R_{p2c}$,
    $B_{16} : R_{p3} = 13233647 \times 2^{-11}$,
    $\ldots$
  - Conclusion: $|v_{x2} - \tilde{v}_{x2}| \leq 2^{-11}$.

Outline

- The example of an aeronautical application.
  Robustness is critical, the implementation needs to be certified. Yet the algorithm is mathematically simple, its study should not require much human expertise.
- Methods used to analyze numerical programs.
  Bounds on variable are computed through interval arithmetic, and round-off errors are propagated through forward error analysis.
- Formal proofs.
  Why a formal proof? Conclusion and perspectives.

Bounding the variables

- Compute the bounds by interval arithmetic.
  For each floating point operator $\square$, define an interval operator such that
  $\forall A, B \in \mathbb{IF}, A \square B \supseteq \{\tilde{c} \in \mathbb{F} | \tilde{a} \in A, \tilde{b} \in B, c = \tilde{a} \square \tilde{b}\}$

Bounding the variables

- Compute the bounds by interval arithmetic.
  For each floating point operator $\square$, define an interval operator such that
  $\forall A, B \in \mathbb{IF}, A \square B \supseteq \{\tilde{c} \in \mathbb{F} | \tilde{a} \in A, \tilde{b} \in B, c = \tilde{a} \square \tilde{b}\}$
- Floating point operators are monotone:
  if $A = [\underline{a}, \overline{a}]$ and $B = [\underline{b}, \overline{b}]$, then we define
  - $A \oplus B = [\underline{a} \oplus \underline{b}, \overline{a} \oplus \overline{b}]$,
  - $A \ominus B = [\underline{a} \ominus \overline{b}, \overline{a} \ominus \underline{b}]$,
  - $A \otimes B = \begin{array}{l}[\min(\underline{a} \otimes \underline{b}, \underline{a} \otimes \overline{b}, \overline{a} \otimes \underline{b}, \overline{a} \otimes \overline{b}), \\ \quad \max(\underline{a} \otimes \underline{b}, \underline{a} \otimes \overline{b}, \overline{a} \otimes \underline{b}, \overline{a} \otimes \overline{b})],\end{array}$
  - etc.

Bounding the variables

- Floating-point interval operators:
  - $A \oplus B = [\underline{a} \oplus \underline{b}, \overline{a} \oplus \overline{b}]$,
  - $A \ominus B = [\underline{a} \ominus \overline{b}, \overline{a} \ominus \underline{b}]$,
  - $A \otimes B = \begin{array}{l}[\min(\underline{a} \otimes \underline{b}, \underline{a} \otimes \overline{b}, \overline{a} \otimes \underline{b}, \overline{a} \otimes \overline{b}), \\ \quad \max(\underline{a} \otimes \underline{b}, \underline{a} \otimes \overline{b}, \overline{a} \otimes \underline{b}, \overline{a} \otimes \overline{b})],\end{array}$
- This is an exotic interval arithmetic implementation: the computed bounds are not to be rounded up or down.
  - Hence we use the Boost[a] interval arithmetic library.
  - It is combined with the SoftFloat library to provide support for all floating point formats.

[a] Brönnimann, Melquiond, Pion, `http://www.boost.org/libs/numeric/interval/`

Bounding the errors

- Errors are also bounded by intervals ($\tilde{v} \in \mathbb{F}$, $v \in \mathbb{R}$):
  - absolute error: $v - \tilde{v} \in A_{\tilde{v},v}$,
  - relative error: $v/\tilde{v} - 1 \in R_{\tilde{v},v}$ for v and $\tilde{v}$ of same sign.

Bounding the errors

- Errors are also bounded by intervals ($\tilde{v} \in \mathbb{F}$, $v \in \mathbb{R}$):
  - absolute error: $v - \tilde{v} \in A_{\tilde{v},v}$,
  - relative error: $v/\tilde{v} - 1 \in R_{\tilde{v},v}$ for v and $\tilde{v}$ of same sign.
- Absolute error of the multiplication:

$$\begin{aligned} x \times y - \tilde{x} \otimes \tilde{y} &= (x \times y - \tilde{x} \times \tilde{y}) + (\tilde{x} \times \tilde{y} - \tilde{x} \otimes \tilde{y}) \\ &= (x - \tilde{x})\tilde{y} + (y - \tilde{y})\tilde{x} + (x - \tilde{x})(y - \tilde{y}) + \epsilon_0 \\ A_{x \times y, \tilde{x} \otimes \tilde{y}} &\subseteq A_{x,\tilde{x}}\tilde{Y} + A_{y,\tilde{y}}\tilde{X} + A_{x,\tilde{x}}A_{y,\tilde{y}} + A^0_{\tilde{x} \otimes \tilde{y}} \end{aligned}$$

$\epsilon_0 \in A^0_{\tilde{x} \otimes \tilde{y}}$ is the rounding error.

Bounding the errors

- Absolute error of the multiplication:

$$
\begin{aligned}
x \times y - \tilde{x} \otimes \tilde{y} &= (x \times y - \tilde{x} \times \tilde{y}) + (\tilde{x} \times \tilde{y} - \tilde{x} \otimes \tilde{y}) \\
&= (x - \tilde{x})\tilde{y} + (y - \tilde{y})\tilde{x} + (x - \tilde{x})(y - \tilde{y}) + \epsilon_0 \\
A_{x \times y, \tilde{x} \otimes \tilde{y}} &\subseteq A_{x,\tilde{x}}\tilde{Y} + A_{y,\tilde{y}}\tilde{X} + A_{x,\tilde{x}}A_{y,\tilde{y}} + A^0_{\tilde{x} \otimes \tilde{y}}
\end{aligned}
$$

- This is traditional interval arithmetic:
  - interval operators deal with real numbers,
  - interval bounds are rounded up and down through MPFR.

Outline

- The example of an aeronautical application.
  Robustness is critical, the implementation needs to be certified. Yet the algorithm is mathematically simple, its study should not require much human expertise.
- Methods used to analyze numerical programs.
  Bounds on variable are computed through interval arithmetic, and round-off errors are propagated through forward error analysis.
- Formal proofs.
  Why a formal proof? Conclusion and perspectives.

Formal proof

- Automatically computing bounds on variables and errors may suffer from limitations or bugs from the tools. Examples:
  - no support for subnormal numbers,
  - a problem in the underlying arithmetic libraries.

Formal proof

- Automatically computing bounds on variables and errors may suffer from limitations or bugs from the tools. Examples:
  - no support for subnormal numbers,
  - a problem in the underlying arithmetic libraries.
- Two solutions:
  - 1. certify the tools and their libraries,
  - 2. generate a formal proof along the computations.

Oracles

- The tool is external, not tied to any proof checker. It can act as an oracle and simplify the proof.
  - If the user just wants the tool to prove $|x - \tilde{x}| < 1.5$, no need to generate the complex proof of the optimal bound $|x - \tilde{x}| < 1.43569726$.
  - The shorter the numbers are, the faster multi-precision floating point arithmetic is. Simpler proofs are validated faster.

Conclusion

- Certifying the numerical behavior of a program is a tedious task:
  - tools are a necessity,
  - they should not require extensive knowledge on the domain.
- Advantage of our approach to formal proof:
  - no need for a blind faith in the tools,
  - results are usable in an extensive formal certification of a program.

Perspectives

- Interface our tools with Why:
  - a software verification tool,
  - it generates proof obligations for Coq, PVS, etc.
- Handle alternate computer arithmetics:
  - floating point, double-double,
  - fixed point arithmetic.
- Generate formal proofs for other proof checkers.

Questions?

Url: `http://lipforge.ens-lyon.fr/www/gappa/`

Email: `guillaume.melquiond@ens-lyon.fr`