Why3 を用いた区間演算ライブラリの検証

藪 智仁　石井 大輔

区間演算ライブラリは浮動小数点数を境界とした区間同士の四則演算を提供する. 演算が返す値は, 引数の区間中の各値についての演算結果と境界の演算による丸め誤差を含む区間である. 一般に区間演算のアルゴリズムは多数の条件分岐や丸め演算などにより複雑で, その実装の信頼性は高くない. 本研究の目的は正しさが検証された区間演算ライブラリの提供である. 検証には Why3 プラットフォームを用い, アノテーション付き実装から検証条件を生成し, SMT ソルバーと Coq により証明する. 検証プロセスにおいて非線形項を含む検証条件や浮動小数点数の複雑な公理を扱うため, (1) 浮動小数点数定理の実数定理による言い直し, (2) 区間の上下限による定理分割, (3) 中間変数による補助定理, の三つの戦略を用いた. その結果, 区間演算ライブラリに関するすべての検証条件を証明することができた.

Interval arithmetic libraries provide the four operators on the argument intervals whose bounds are floating-point numbers. A returned value of the operators is another interval that contains every true values computed from the real values within the arguments, plus rounded errors caused by the computation at the bounds. In general, an actual implementation should be complicated because of large case analysis and round operations. As the existing code bases are implemented by human hands, their correctness is not obvious, which come to be critical for the reliability. This work provides a mechanically verified implementation of the interval arithmetic library. For the verification, we utilize the Why3 platform and generates the verification conditions (VCs) from the specification; then, the Alt-Ergo SMT solver and the Coq proof assistant are applied to the VCs to discharge all of them. To discharge all VCs, we use three strategies, (1) Restatement in the real theory, (2) Split into the lower and upper part, and (3) Lemmas on intermediate variables. Finally, a mechanically-verified and executable implementation of the interval arithmetic is provided.

1　はじめに

区間演算 (2 節) は実数を扱うための高信頼な手法である. 区間演算は数値の代わりに (一般的には浮動小数点数の) 区間を扱い, 真の解を逃さないように慎重な丸め方向の制御を行いながら演算を行う. 区間の四則演算は引数として与えられる区間の境界のみを計算する効率的な方法で実装できる. 実装では繰り返し処理は必要ないが, 多数の場合分けを行う. これは引数区間の境界同士, 0 との大小関係, 特殊な値 $\pm\infty$ と NaN , を区別するためである. 例えば kv ライブラリの乗算の実装は 17 個の分岐を持つ. 基本的に既存の区間演算プログラムは人間の手により実装しているため, 信頼性は低く, 当初の目的であった高信頼性を損なう恐れがある.

Why3 (3 節) は, 自動定理証明器 (例: SMT ソルバー) や対話定理証明器 (例: Coq) を統合するよう設計されたプログラム検証プラットフォームである. Why3 は検証対象を記述する仕様言語 `WhyML` と `Why3` を提供しており, アノテーション付きプログラムを記述することができる. ある仕様が与えられると, Why3 はその仕様を全て満たすことを保証するための検証条件 (VC) を生成する. そしてユーザーがバックエンド証明器を適用し, 生成されたすべての VC の証明することで, 与えられた仕様の正しさを証明する. Why3 と Why3 のバックエンド証明器は, 数値計算分野もサポートするように開発されてきた.

我々の目標 (4 節) は正しさの検証された区間の四則演算プログラムを提供することである. 目標のために, 我々は kv ライブラリの実装を `WhyML` に変換し, `Why3`

---

* Verification of Interval Arithmetic Library Using Why3. This is an unrefereed paper. Copyrights belong to the Author(s).

Tomohito Yabu, Daisuke Ishii, 福井大学, University of Fukui.

の事前・事後条件をアノテーションする. 生成されたすべての VC について, バックエンド証明器に SMT ソルバー (Alt-Ergo と Z3) と Coq を用い証明することで, 区間演算プログラムの正しさの検証を行う (5 節) .

検証プロセスは簡単ではない. 例えば SMT ソルバーは非線形項を含む検証条件を扱うことができず, また回避策としての対話証明は膨大な数の浮動小数点数定理により困難である. そのため Why3 の単純な利用では全ての VC を証明することはできない.

証明できない VC を検証するため, 我々は仕様に検証プロセスを支援する補助定理をアノテーションする三つの戦略を提案する (6–7 節) .

区間の四則演算の検証実験において, アノテーションされた補助定理は証明タスクを徐々に簡単なタスクに分割し, 最終的にはバックエンド証明器 (特に Alt-Ergo と Coq) によりすべての VC を証明する (8 節) . 結果として機械的に正しさが検証され, かつ実行可能な区間演算プログラムが提供できる.

2 区間演算

区間演算 [7] は, 実数が厳密には表現不可能な計算機上で高信頼な数値計算を行うための手法である. 区間演算は実数演算を区間の演算に拡張したもので, 数値の代わりに真解を含んだ区間値を計算し, 結果として得られる区間は演算がとりうるすべての結果を含む.

(有限) **区間** $\boldsymbol{x} = [\underline{x}, \overline{x}]$ は集合 $\{x \in \mathbb{R} : \underline{x} \le x \le \overline{x}\}$ として定義する. ただし $\underline{x} \le \overline{x}$ とする. 実際の区間の実装では, **下限** $\underline{x}$ と **上限** $\overline{x}$ は $\mathbb{F} \cup \{-\infty, +\infty\}$ に制限する. ただし $\mathbb{F}$ は $\pm\infty$ と NaN を除いた浮動小数点数の集合とし, $\mathbb{F} \cup \{-\infty, +\infty, \mathrm{NaN}\}$ を $\mathbb{F}^*$ とする. 実数を境界とする区間の集合は $\mathbb{IR}$ と表し, 浮動小数点数を境界とする場合は $\mathbb{IF}$ と表す. $\mathbb{IR}$ と $\mathbb{IF}$ を特に区別しない場合は $\mathbb{I}$ と表す. ある実数の集合 $S \subseteq \mathbb{R}$ に対して, **包囲** $\square S \in \mathbb{I}$ は $S \subseteq \square S$ を満たす最小幅の区間である. ここで区間の四則演算を定義する.

定義 2.1 (区間の四則演算) $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{I}, \odot \in \{+, -, *, /\}$ としたとき, 区間の四則演算は以下のように定義される:

$$\boldsymbol{x} \odot \boldsymbol{y} := \square\{x \odot y : x \in \boldsymbol{x}, y \in \boldsymbol{y}\} \tag{2.1}$$

ただし $\odot = /$ の際は $0 \notin \boldsymbol{y}$ と仮定する.

実際の実装は区間の境界同士のみを演算することで実装できる. 演算結果が表現不可能になった際は, 上下限をそれぞれ上向きと下向きに**丸める**. つまり理論値より広い区間に近似する. なお, ある実数 $x \in \mathbb{R}$ に対して $\nabla(x), \triangle(x) \in \mathbb{F} \cup \{-\infty, +\infty\}$ は, それぞれ x の**下向きと上向きの丸め**を表す. ここで実装のための基礎定理を導入する.

定理 2.2 (四則演算の実装) $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{I}$ について以下が成り立つ:

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= [\nabla(\underline{x} + \underline{y}), \triangle(\overline{x} + \overline{y})] \\
\boldsymbol{x} - \boldsymbol{y} &= [\nabla(\underline{x} - \overline{y}), \triangle(\overline{x} - \underline{y})] \\
\boldsymbol{x} * \boldsymbol{y} &= \square\{\underline{x}\,\underline{y}, \underline{x}\,\overline{y}, \overline{x}\,\underline{y}, \overline{x}\,\overline{y}\} \\
\boldsymbol{x} / \boldsymbol{y} &= \boldsymbol{x} * [\nabla(1 / \overline{y}), \triangle(1 / \underline{y})] \quad (0 \notin \boldsymbol{y})
\end{aligned} \tag{2.2}$$

2.1 区間演算の実装

区間演算を実装したライブラリは, Profil/Bias †1, filib †2, INTLAB †3, kv †4 などがある.

区間演算のうち, 乗除演算は複数の場合分けに基づいて実装されている. これは, 効率的な演算や, 浮動小数点数の演算結果が $\pm\infty$ や NaN になる場合を考慮するためである.

区間演算を実装コードのある分岐に着目し, その計算を取り出したものを以下のように形式化する:

定義 2.3 (実装関数とその正しさ) 区間の集合を $\mathbb{I} := \mathbb{IR}$ または $\mathbb{I} := \mathbb{IF}$, 演算 $\odot \in \{+, -, *, /\}$, 関係 $B \subseteq \mathbb{I}^2$ としたとき, **実装** $\boldsymbol{f}_{\odot,B}$ は $\mathbb{I} \times \mathbb{I}$ から $\mathbb{R} \cup \{-\infty, +\infty, \mathrm{NaN}\}$ の (連結) 部分集合への関数である.「$\boldsymbol{f}_{\odot,B}$ は **正しい**」の必要十分条件を以下に示す:

$$\begin{aligned}
&\forall \boldsymbol{x} \in \mathbb{I}, \forall \boldsymbol{y} \in \mathbb{I},\ B(\boldsymbol{x}, \boldsymbol{y}) \Rightarrow \boldsymbol{f}_{\odot,B}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{I} \wedge \\
&\quad \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \tilde{x} \odot \tilde{y} \in \boldsymbol{f}_{\odot,B}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned} \tag{2.3}$$

式 (2.3) では, 後件の一つ目の部分は演算結果が適切な区間かのチェックであり (例: 境界が NaN でない, かつ大小関係を満たす) , 二つ目の部分は真値を含ん

†1 http://www.ti3.tuhh.de/keil/profil/index_e.html
†2 http://www2.math.uni-wuppertal.de/~xsc/software/filib.html
†3 http://www.ti3.tu-harburg.de/rump/intlab/
†4 http://verifiedby.me/kv/

でいるかのチェックである.

例えば乗算のときに, B が

$$\underline{x} \geq 0 \wedge \overline{x} \neq 0 \wedge \underline{y} \geq 0 \wedge \overline{y} \neq 0 \tag{2.4}$$

で, 実装

$$f_{*,\text{式 (2.4)}}(\boldsymbol{x}, \boldsymbol{y}) := [\nabla(\underline{x}\,\underline{y}), \triangle(\overline{x}\,\overline{y})] \tag{2.5}$$

の場合が考えられる.

3 Why3 プラットフォーム

Why3 †5 [2] は演繹的プログラム検証のためのプラットフォームであり, 三つの機能 (1) 検証対象を記述するための仕様言語, (2) 仕様付きプログラムから最弱事前条件の計算に基づき **検証条件 (Verification Condition, VC)** を生成する機能, (3) VC を使用するバックエンド自動・対話定理証明器に中継する機能, を提供している. Why3 は OCaml 風の言語を二つ持つ. 命令型プログラムを記述するための **プログラミング言語 WhyML** と, プログラムに仕様として付与する一階述語論理式を記述するための **論理言語 Why3** である. Why3 のバックエンド証明器には, SMT ソルバー [8] (例: Alt-Ergo †6), 定理証明支援系 (例: Coq †7) などがある. なお, SMT ソルバーをバックエンド証明器として用いた場合の結果は, (1) 検証条件の妥当性を判定 (Valid / Invalid), (2) 制限時間までに停止しない (Timeout) , (3) 決定不能 (Unknown) を出力, のいずれかとなる. Why3 は GUI も提供しており, 生成された VC の一覧表示, VC の変形 (例: 複数の VC への分割), VC への証明器の割り当て, 証明器の入力・出力の確認などが行える.

`WhyML` と `Why3` 言語は実数用の `real` 型や, 64 ビット浮動小数点数を表す `double` 型, さらにユーザー定義のレコード型などを提供する型システムを備える. どちらの言語も仕様ごとに `module` または `theory` としてカプセル化するモジュールシステムを持つ. 標準モジュールは標準ライブラリとして備わっている.

Why3 は数値計算プログラムの検証に用いられてきた実績を持つ [1] [4] †8.

†5 `http://why3.lri.fr`
†6 `http://alt-ergo.lri.fr`
†7 `https://coq.inria.fr`
†8 参考文献は Why3 の前身での研究である.

3.1 Flocq

Flocq †9 [3] は Coq 上で浮動小数点数を扱うためのライブラリである. Flocq は, 様々な基数 (2, 10 や, 奇数の基数も) やビット数の浮動小数点数, 様々な丸めの指定をサポートしており, 450 を超える定理を持ち, 約 12000 行からなる.

Why3 により浮動小数点数を用いた仕様から VC を生成し, Coq 上で対話証明を行う際, Why3 は Flocq に基づいた Coq の定理を生成する.

4 区間演算ライブラリの正しさの検証

本研究の目標は, 区間演算ライブラリの正しさの検証である. 正しいとは, 適切な区間が与えられた時に, プログラムの出力結果が, 理論値を包囲する適切な区間になっていること (式 (2.3)) である.

大まかな検証の流れは以下の四ステップからなる: (1) kv ライブラリの四則演算コードを `WhyML` に書き直し, 区間演算の定義を `Why3` でアノテーションし, (2) Why3 が記述コードから生成した VC に対して, SMT ソルバーにより自動証明を試み, (3) 自動証明できない VC を Coq により対話証明を試み, (4) それでも VC を証明できない場合は, 本研究の提案する三戦略を用い証明する.

5 節 で VC の検証基本プロセスを示し, 6–7 節で基本プロセスでは VC が証明できない場合に我々が提案する効果的な三戦略を示し, 8 節 で実験結果を示す.

本研究で検証する VC (式 (2.3) を言い換えたもの) は実数, 浮動小数点数, 区間の変数を含む一階述語論理式として形式化できる.

定義 4.1 (検証条件) $\mathbb{I} := \mathbb{IR}$ または $\mathbb{I} := \mathbb{IF}$ の区間 $\boldsymbol{x}$ と $\boldsymbol{y}$, 実装する演算子 $\odot$, そして関係 $B \subseteq \mathbb{I}^2$ を仮定し, 式 $VC^{I}_{\odot,B}$ と $VC^{C}_{\odot,B}$ を以下のように定義する:

$VC^{I}_{\odot,B} :\equiv B(\boldsymbol{x}, \boldsymbol{y}) \Rightarrow f_{\odot,B}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{I}$

$VC^{C}_{\odot,B} :\equiv B(\boldsymbol{x}, \boldsymbol{y}) \Rightarrow \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \tilde{x} \odot \tilde{y} \in f_{\odot,B}(\boldsymbol{x}, \boldsymbol{y})$

このとき $f_{\odot,B}$ は *2. 1* 節 の実装関数を表す.

$VC^{I}_{\odot,B}$ は演算結果が適切な区間であることをいい,

†9 `http://flocq.gforge.inria.fr`

```
1  module IntervalMulExact
2    use import real.Real
3
4    type interval = { inf: real; sup: real }
5      invariant { self.inf <= self.sup }
6
7    predicate _in (a: real) (x: interval) =
8      x.inf <= a <= x.sup
9
10   let make_zero () : interval
11     ensures { _in 0. result }
12   = { inf = 0.; sup = 0. }
13
14   let multiply (x y: interval) : interval
15     ensures { forall xx yy: real.
16       _in xx x /\ _in yy y ->
17         _in (xx * yy) result }
18   = if x.inf >= 0. then
19       if x.sup = 0. then make_zero ()
20       else
21         if y.inf >= 0. then
22           if y.sup = 0. then make_zero ()
23           else
24             { inf = x.inf * y.inf;
25               sup = x.sup * y.sup }
26         else
27           (* 省略 *)
28 end
```

図 1　実数境界区間の乗算コードの抜粋. スペースの関係で multiply の 13 個の分岐中 10 個は省略.

$VC^{C}_{\odot,B}$ は演算結果が演算 $\odot$ の真の結果を含むことをいう. 本研究の目標とする, 区間演算ライブラリの正しさの検証は, 検証対象から生成された $VC^{I}_{\odot,B}$ と $VC^{C}_{\odot,B}$ が正しいことを, Why3 のバックエンド証明器を用いて証明することで行う.

5　実数境界区間版の四則演算の検証

本節では, 検証の基本プロセスを説明し, 浮動小数点数を扱う検証との比較のために, 実数を境界に扱う「理想的な」区間演算のプログラムを考える.

5.1　仕様

Why3 の仕様言語による区間の乗算コードを図 1 に示す. まず, 2 行目で実数 theory (実数型 real や実数型の演算, 公理など) を現在の名前空間に導入する. つぎに, 4 行目で下限 (inf) と上限 (sup) を持つ区間型を WhyML のレコードとして定義する. このとき区間の上下限値の大小関係を不変条件 (invariant) としてアノテーションする (5 行目). そして, 7–8 行目である区間 x が, ある実数 a を含んでいるという述語 _in を定義する. さらに, 10–12 行目で 0 を含む区間を生成する WhyML 手続き make_zero を定義する. 最後に, 14–27 行目で WhyML の区間の乗算手続き multiply を定義する. multiply は kv ライブラリにならった実装となっており, 13 個の分岐を持つ.

5.2　検証

前節で書いた仕様の正しさを検証する. まず一つ目の検証ステップとして, Why3 プラットフォームを用いて手続き make_zero と multiply のそれぞれについて VC を生成し, バックエンド SMT ソルバーにより VC の証明を試みる. 5.1 節の例では, Alt-Ergo を用いて make_zero に対して生成された VC の証明を 1 秒以内で行うことができた. これはアノテーションされた手続きが正しいことを意味する. しかし, もう一方の multiply の VC を Alt-Ergo を用いて証明しようとすると, 1 分経っても終了しなかった. この場合, VC を 22 個の部分的な VC に **分割 (split)** し, 証明を試みる. SMT ソルバー (Alt-Ergo と Z3) は 22 個のうち 14 個を証明し, 残りは 8 個となった.

二つ目の検証ステップとして, Coq を用いた対話証明を行う. ここでは, 図 1 の最後の分岐を取り上げ説明する. 最後の分岐では multiply の引数が 2.1 節の条件式 (2.4) を満たしている. 8 個の未証明の VC はこの分岐に関するものであり, 以下の形になっている.

$$VC^{C}_{*,\text{式 (2.4)}} \equiv (\text{式 (2.4)}) \Rightarrow \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \tilde{x}\,\tilde{y} \in \boldsymbol{f}_{*,\text{式 (2.4)}}(\boldsymbol{x}, \boldsymbol{y}) \qquad (5.1)$$

ここで $\boldsymbol{f}_{*,\text{式 (2.4)}}$ の定義は 式 (2.5) である (丸め演算は冗長であるが).

まず始めに, Why3 は VC (5.1) と関連する公理を Coq の記述に変形する. これは以下のように Coq の証明コンテキスト†10 に表示される:

†10　正確には読み込んだ VC に簡略化 (例: 変数名の置換, 前提の選別, 区間コンストラクタの展開) を施したものである.

$$\frac{\begin{array}{c}\texttt{x.inf},\ \texttt{x.sup},\ \texttt{y.inf},\ \texttt{y.sup} \in \mathbb{R} \\ H_0 : \texttt{x.inf} \le \texttt{x.sup} \qquad H_1 : \texttt{y.inf} \le \texttt{y.sup} \\ H_2 : 0 \le \texttt{x.inf} \qquad H_3 : \texttt{x.sup} \ne 0 \\ H_4 : 0 \le \texttt{y.inf} \qquad H_5 : \texttt{y.sup} \ne 0 \\ H_6 : \texttt{x.inf} \cdot \texttt{y.inf} \le \texttt{x.sup} \cdot \texttt{y.sup} \\ \texttt{xx},\ \texttt{yy} \in \mathbb{R} \qquad H_7 : \texttt{x.inf} \le \texttt{xx} \le \texttt{x.sup} \\ H_8 : \texttt{y.inf} \le \texttt{yy} \le \texttt{y.sup}\end{array}}{\texttt{x.inf} \cdot \texttt{y.inf} \le \texttt{xx} \cdot \texttt{yy} \le \texttt{x.sup} \cdot \texttt{y.sup}}.$$

この VC は非線形項を含んでいる. 多くの SMT ソルバーは非線形項を扱うことができない. つぎに, 前提 H_0–H_8 の組み合わせと, Coq の Reals モジュールの定理 Rmult_le_compat (式 (5.2)) , Rmult_le_compat (式 (5.3)) を用いることで, このコンテキストのゴールは証明できる.

$$\forall r_1, r_2, r_3, r_4 \in \mathbb{R},\ 0 \le r_1 \ \Rightarrow\ 0 \le r_3 \ \Rightarrow\ r_1 \le r_2 \ \Rightarrow\ r_3 \le r_4 \ \Rightarrow\ r_1 r_3 \le r_2 r_4 \tag{5.2}$$

$$\forall r_1, r_2, r_3 \in \mathbb{R},\ r_1 \le r_2 \ \Rightarrow\ r_2 \le r_3 \ \Rightarrow\ r_1 \le r_3 \tag{5.3}$$

最終的な証明は 20 個のタクティック適用からなる†11.

この分岐に対応するもう一つの $VC^{\boldsymbol{I}}_{*,\ \text{式 (2.4)}}$ も同様に Coq で証明できる. さらに乗算の他の分岐も, また乗算以外の三演算についてもこの方法で証明できる.

これまでみてきたように, 実数演算プログラムの仕様の検証には SMT ソルバーによる自動証明ができるものもあれば, できないものもある. しかしできない場合も対話証明により検証を試みることができる.

6 浮動小数点数境界の区間演算の検証

つぎに浮動小数点数を境界とする区間演算の検証に進める.

6.1 仕様

WhyML と Why3 の乗算の仕様を 図 2 に示す. まず始めに, 浮動小数点数と丸め演算の定理を導入する (3–6 行目) . 導入する語彙の一部を 表 1 に示す. 残りの仕様は, double の値の扱いや特殊な値 (例: ±∞ や NaN) のための追加の分岐と丸め演算を除いて, 基本的に 図 1 と同じである. 9–10 行の invariant は区間の境界が NaN の場合や $[-\infty, -\infty]$ と $[+\infty, +\infty]$ の区間を禁止している. 述語 _in はある real の値がある double-境界区間に含まれていることをいう (12 行目) . 手続き whole は区間 $[-\infty, +\infty]$ を返す (18–20 行目) . 22–35 行目の手続き multiply は 17 個の分岐からなる. これは 図 1 の 4 個の分岐が, それぞれ無限大の境界を扱うため 26 行目のように分岐が増え, 8 個の分岐になるためである.

表 1　浮動小数点数と丸め演算の語彙の抜粋.

述語/関数	定義
is_finite x	$x \in \mathbb{F}$
is_plus_infinity x	$x = +\infty$
double_le_real x y	$x \le y$ $(x \in \mathbb{F}^*, y \in \mathbb{R})$
ge_zero x	$0 \le x$ $(x \in \mathbb{F}^*)$
is_gen_zero x	$x = 0$ $(x \in \mathbb{F}^*)$
mul_down x y	$\nabla(x\ y)$ $(\mathbb{F}^* \times \mathbb{F}^* \to \mathbb{F}^*)$
mul_post d x y r	r = mul_down x y (d = Down のとき)

6.2 検証の最初の試み

図 2 の丸め演算つき区間の乗算コードについて, Why3 は 3 個の VC を生成し, さらに GUI のインターフェイスを通して 32 個の VC に分割する. 図 2 の最後の分岐は, 5.2 節 の条件式 (2.4) の場合であり, 同じ $VC^{\boldsymbol{C}}_{*,\ \text{式 (2.4)}}$ が生成される. この VC の Why3 上での内部表現 (Why3 による仕様) を 図 3 に示す. 変数 xinf, xsup, yinf, および ysup は引数区間の境界を表し, 11–12 行目の rinf と rsup は乗算結果区間の境界を表す. 2–5 行目と 13–14 行目は適切な区間であることをいい†12, 15–17 行目は乗算結果が真値を含むことをいう.

5.2 節同様, 検証プロセスの一つ目のステップとして自動証明を適用し, Alt-Ergo は 15 個の VC を 150 秒

†11　証明は fourier タクティックによりさらに簡略化できる.

†12　VC を分割したため, $VC^{\boldsymbol{C}}_{*,\ \text{式 (2.4)}}$ では式 (2.3) の後件の一つ目である $VC^{\boldsymbol{I}}_{*,\ \text{式 (2.4)}}$ は前提に入る.

```
module IntervalMulRounded
  use import real.Real
  use import floating_point.Rounding
  use import floating_point.SpecialValues
  use import floating_point.DoubleFull
  use import double_wrapper.DoubleWrapper

  type interval = { inf: double; sup: double }
    invariant { (is_finite self.inf \/ is_minus_infinity self.inf)
      /\ (is_finite self.sup \/ is_plus_infinity self.sup) /\ le self.inf self.sup }

  predicate _in (a: real) (x: interval) = double_le_real x.inf a /\ real_le_double a x.sup

  let make_zero () : interval
    ensures { _in 0. result }
  =   { inf = double_zero; sup = double_zero }

  let whole () : interval
    ensures { forall r:real. _in r result }
  =   { inf = minus_infinity; sup = plus_infinity }

  let multiply (x y: interval) : interval
    ensures { forall xx yy: real. _in xx x /\ _in yy y -> _in (xx * yy) result }
  =   if ge_zero x.inf then
        if is_gen_zero x.sup then
          if is_infinite y.inf || is_infinite y.sup then whole () else make_zero ()
        else
          if ge_zero y.inf then
            if is_gen_zero y.sup then
              if is_infinite x.inf || is_infinite x.sup then whole () else make_zero ()
            else
                (* 補助定理をここに追加する *)
                { inf = mul_down x.inf y.inf; sup = mul_up x.sup y.sup }
          else
    (* 省略 *)
end
```

図 2　浮動小数点数の乗算のコードの抜粋. スペース制限のため `multiply` は 5 個の分岐のみ示す.

以内で証明する. しかし $VC^{C}_{*,\text{式 (2.4)}}$ を含めた 17 個の VC は 4 分経っても証明できない†13.

二つ目のステップとして, Coq を起動し未証明の VC の対話証明を試みる. しかし浮動小数点数を扱う実装のため, Why3 によって変換された Coq の定理は 5.2 節の定理と比較して複雑になる. 例えば, $VC^{C}_{*,\text{式 (2.4)}}$ の Coq のコンテキストを 図 4 に示す. このように複雑になる理由は, 浮動小数点数の区間はユーザー定義オブジェクトで表現され, 表 1 の語彙を用いるすべての宣言は追加の前提を導入するためである. また $\mathbb{F}^*$ 値域は実数の述語を用いて表現されるため, 上の Coq のコンテキストはさらに複雑になる. そのため, 5.2 節同様に, 実数の定理 (例: 式 (5.2) や 式 (5.3)) を適用する場合, $\mathbb{F}^*$ を展開する必要があり複雑化する. 結果として, 例え基本演算を考えたときでさえ, 多くの VC の手による証明は困難である.

7　補助定理による検証

丸め区間演算コードから生成された VC の証明が難しい場合に, 証明に有効な戦略がある. 本研究のアプ

†13　他の SMT ソルバー (例: Z3 や CVC4) も同様に証明できない.

```
forall (xinf:double) (xsup:double) (yinf:double) (ysup:double),
  ((is_finite xinf) \/ (is_minus_infinity xinf)) /\
  ((is_finite xsup) \/ (is_plus_infinity xsup)) /\ (le xinf xsup) /\
  ((is_finite yinf) \/ (is_minus_infinity yinf)) /\
  ((is_finite ysup) \/ (is_plus_infinity ysup)) /\ (le yinf ysup) ->

  (* 分岐条件 *)
  (ge_zero xinf) -> (~ (is_gen_zero xsup)) -> (ge_zero yinf) -> (~ (is_gen_zero ysup)) ->

    (* 区間の乗算 *)
    forall (rinf:double), (mul_post Down xinf yinf rinf) ->
    forall (rsup:double), (mul_post Up xsup ysup rsup) ->
      ((is_finite rinf) \/ (is_minus_infinity rinf)) /\
      ((is_finite rsup) \/ (is_plus_infinity rsup)) /\ (le rinf rsup) ->
        forall (xx:R) (yy:R),
        (_in xx (mk_interval xinf xsup)) /\ (_in yy (mk_interval yinf ysup)) ->
          (_in (xx*yy) (mk_interval rinf rsup)).
```

図 3　Why3 でエンコードされた $VC^{C}_{*,\text{式 (2.4)}}$.

$$\frac{\begin{array}{c}\mathtt{x.inf},\mathtt{x.sup},\mathtt{y.inf},\mathtt{y.sup} \in \mathbb{F}^*\\ H_0 : \mathtt{x.inf} \in \mathbb{F} \vee \mathtt{x.inf} = -\infty\\ H_1 : \mathtt{x.sup} \in \mathbb{F} \vee \mathtt{x.sup} = +\infty\\ H_2 : \mathtt{y.inf} \in \mathbb{F} \vee \mathtt{y.inf} = -\infty\\ H_3 : \mathtt{y.sup} \in \mathbb{F} \vee \mathtt{y.sup} = +\infty\\ H_4 : \mathtt{x.inf} \le \mathtt{x.sup} \qquad H_5 : \mathtt{y.inf} \le \mathtt{y.sup}\\ H_6 : 0 \le \mathtt{x.inf} \qquad H_7 : \mathtt{x.sup} \neq 0\\ H_8 : 0 \le \mathtt{y.inf} \qquad H_9 : \mathtt{y.sup} \neq 0\\ \mathtt{r.inf},\mathtt{r.sup} \in \mathbb{F}^*\\ H_{10} : \mathtt{r.inf} = \nabla(\mathtt{x.inf}\cdot\mathtt{y.inf})\\ H_{11} : \mathtt{r.sup} = \triangle(\mathtt{x.sup}\cdot\mathtt{y.sup})\\ H_{12} : \mathtt{r.inf} \in \mathbb{F} \vee \mathtt{r.inf} = -\infty\\ H_{13} : \mathtt{r.sup} \in \mathbb{F} \vee \mathtt{r.sup} = +\infty\\ H_{14} : \mathtt{r.inf} \le \mathtt{r.sup}\\ \mathtt{xx},\mathtt{yy} \in \mathbb{R} \qquad H_{15} : \mathtt{x.inf} \le \mathtt{xx} \le \mathtt{x.sup}\\ H_{16} : \mathtt{y.inf} \le \mathtt{y.sup}\end{array}}{\mathtt{r.inf} \le \mathtt{xx}\cdot\mathtt{yy} \le \mathtt{r.sup}}.$$

図 4　$VC^{C}_{*,\text{式 (2.4)}}$ の Coq のコンテキスト.

ローチの鍵は SMT ソルバー, 特に Alt-Ergo の自動証明プロセスを, 多くの前提から可能な限り推論できるよう誘導することである. これは前提 $VC^{I}_{\odot,B}$ または $VC^{C}_{\odot,B}$ のどちらかに補助定理 L を

$$B(\boldsymbol{x},\boldsymbol{y}) \Rightarrow L(\boldsymbol{x},\boldsymbol{y}) \Rightarrow P$$

と追加することでできる (結論は P と省略している).

7.1　二つの戦略

この節では補助定理を追加する二つの戦略を提案する. 補助定理は仕様に WhyML コマンド assert で追加できる. 補助定理を追加することにより, 証明の一部が後回しにされ, 現在の証明プロセスが単純化されることがある. assert コマンドを追加すると, Why3 はコマンドに対する追加の VC を生成し, 自動または対話証明器のどちらかで証明する必要がある.

7.1.1　戦略 1: 浮動小数点数定理の実数定理による言い直し

我々は 5 節で証明した実数区間版の定理を補助定理として与えることを検討した. 実験では, $\bullet \in \{I, C\}$ としたとき, 丸め区間演算プログラムから生成された $VC^{\bullet}_{\odot,B}$ を証明する効果的な戦略は, 補助定理

$$\forall \boldsymbol{x} \in \mathbb{IR}, \forall \boldsymbol{y} \in \mathbb{IR}, VC^{C}_{\odot,B}$$

を追加することだと分かった. 例えば, 6.2 節の $VC^{I}_{*,\text{式 (2.4)}}$ と $VC^{C}_{*,\text{式 (2.4)}}$ を証明するために, 5.2 節で証明した $\forall \boldsymbol{x} \in \mathbb{IR}, \forall \boldsymbol{y} \in \mathbb{IR}, VC^{C}_{*,\text{式 (2.4)}}$ を前提とすることで Alt-Ergo の証明プロセスを助けることができ, 証明できるようになる. 最終的に, 図 2 の 32 行目に以下の assert コマンドを追加した:

```
assert { forall xinf xsup yinf ysup : real.
  xinf >= 0. -> xsup <> 0. ->
```

```
    yinf >= 0. -> ysup <> 0. ->
      forall xx yy : real.
        xinf <= xx <= xsup ->
          yinf <= yy <= ysup ->
            xinf*yinf <= xx*yy <= xsup*ysup };
```

実験では, この戦略により Alt-Ergo は 6.2 節で説明した未証明の VC の証明を補助することを確認した†14.

7.1.2 戦略 2: 区間の境界による定理分割

本実験では, VC を証明するためにさらなる補助定理を追加する必要があった. もう一つの補助定理は, `assert` を用いて VC を上限と下限に定理を分割することである. 下限の補助定理は以下のようになる:

$$\forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \tilde{x} \odot \tilde{y} \geq \underline{f_{\odot,B}(\boldsymbol{x},\boldsymbol{y})}$$

ただし $\underline{f_{\odot,B}(\boldsymbol{x},\boldsymbol{y})}$ は演算結果の下限を表す. 例えば $\underline{f_{*,\text{式 (2.4)}}(\boldsymbol{x},\boldsymbol{y})} = \nabla(\underline{x}\,\underline{y})$ である.

$VC^{C}_{*,\text{式 (2.4)}}$ を証明するために, 一つ目の戦略による補助定理に加えて,

```
assert { forall rinf : double.
  rinf = mul_down x.inf y.inf ->
    forall xx yy : real.
      _in xx x -> _in yy y ->
        double_le_real rinf (xx*yy) };
```

と, 上限の補助定理をプログラムに挿入する必要がある. Alt-Ergo はこれらの `assert` を 3.1 秒と 9.7 秒で証明し, 最終的に $VC^{C}_{*,\text{式 (2.4)}}$ を 16.5 秒で証明する.

7.2 もう一つの戦略

今までに導入した二つの戦略で, 四則演算の目標コードの 4 個以外の分岐の VC は証明できる. 二つの戦略でも証明できない場合に, 証明を補助するもう一つの戦略がある.

この節では `multiply` の別の分岐を考える. 図 5 は未証明の 4 個の分岐のうちの一つのみを記述するよう, コードを修正している. この分岐の条件は二段階からなる. 一つ目 (3–6 行目) は, 未証明の 4 個の分岐に共通する条件

$$\underline{x} < 0 < \overline{x} \wedge \underline{y} < 0 < \overline{y} \tag{7.1}$$

であり, これは引数の区間が 0 を含むことを述べている. 二つ目は, 11–12 行目の `if` 文から与えられる条件

```
1   let multiply (x y: interval) : interval
2     (* 事後条件は省略 *)
3   =   if not ge_zero x.inf &&
4            not le_zero x.sup then
5         if not ge_zero y.inf &&
6              not le_zero y.sup then
7           let inf1 = mul_down x.inf y.sup in
8           let inf2 = mul_down x.sup y.inf in
9           let sup1 = mul_up x.inf y.inf in
10          let sup2 = mul_up x.sup y.sup in
11          if lt inf1 inf2 &&
12             gt sup1 sup2 then
13            (* 補助定理をここに追加する *)
14            { inf = inf1; sup = sup1 }
15          else
16        (* 省略 *)
```

図 5　浮動小数点数区間に対する `multiply` の別の分岐.

$$\nabla(\underline{x}\cdot\overline{y}) < \nabla(\overline{x}\cdot\underline{y}) \wedge \triangle(\underline{x}\cdot\underline{y}) > \triangle(\overline{x}\cdot\overline{y}) \tag{7.2}$$

で, 境界の演算の四つの場合の一つを指定している. 以降, 条件 式 (7.1) ∧ 式 (7.2) は B' と省略する. つぎに, この分岐のための $VC^{I}_{*,B'}$ は以下である:

$$B'(\boldsymbol{x},\boldsymbol{y}) \Rightarrow [\nabla(\underline{x}\cdot\overline{y}), \triangle(\underline{x}\cdot\underline{y})] \in \mathbb{IF} \tag{7.3}$$

さらに結論を分解したものを以下に示す:

$$\begin{aligned} &B'(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \\ &\quad \nabla(\underline{x}\cdot\overline{y}) \in \mathbb{IF}\cup\{-\infty\} \wedge \triangle(\underline{x}\cdot\underline{y}) \in \mathbb{IF}\cup\{+\infty\} \wedge \\ &\quad \nabla(\underline{x}\cdot\overline{y}) \leq \triangle(\underline{x}\cdot\underline{y}) \end{aligned} \tag{7.4}$$

また, $VC^{C}_{*,B'}$ を以下に示す:

$$B'(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \nabla(\underline{x}\,\overline{y}) \leq \tilde{x}\,\tilde{y} \leq \triangle(\underline{x}\,\underline{y}) \tag{7.5}$$

VC を証明するために, 一つ目の戦略に従って以下の補助定理を 13 行目に `assert` する:

$$\begin{aligned} &\forall \boldsymbol{x} \in \mathbb{IR}, \forall \boldsymbol{y} \in \mathbb{IR}, \\ &B'(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y},\ \underline{x}\,\overline{y} \leq \tilde{x}\,\tilde{y} \leq \underline{x}\,\underline{y} \end{aligned}$$

これは Coq により証明できる. この補助定理により $VC^{I}_{*,B'}$ は Alt-Ergo により 91 秒で証明できる. しかし $VC^{C}_{*,B'}$ は 300 秒でも証明できない.

図 5 の目標のプログラムの検証を達成するために, 7–10 行目で導入されている中間変数に着目する. VC はこれらの変数を用いて書き換えられる. 例えば, VC (7.5) は以下のように書き換えられる:

†14　この補助定理は, 6.2 節 同様に Coq で証明できる.

$$
\begin{aligned}
&(\text{式 (7.1)}) \Rightarrow \forall \mathtt{inf1} \in \mathbb{IF}, \forall \mathtt{sup1} \in \mathbb{IF}, \\
&\qquad \forall \mathtt{inf2} \in \mathbb{IF}, \forall \mathtt{sup2} \in \mathbb{IF}, \\
&\qquad \mathtt{inf1} = \nabla(\underline{x} \cdot \underline{y}) \Rightarrow \mathtt{inf2} = \nabla(\overline{x} \cdot \overline{y}) \Rightarrow \\
&\qquad \mathtt{sup1} = \triangle(\underline{x} \cdot \underline{y}) \Rightarrow \mathtt{sup2} = \triangle(\overline{x} \cdot \overline{y}) \Rightarrow \\
&\qquad \mathtt{inf1} < \mathtt{inf2} \wedge \mathtt{sup1} > \mathtt{sup2} \Rightarrow \\
&\qquad \forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y}, \ \mathtt{inf1} \leq \tilde{x}\,\tilde{y} \leq \mathtt{sup1} \quad (7.6)
\end{aligned}
$$

この変形をより一般化した形式で述べ, 三つ目の戦略を導入する.

7.2.1 戦略 3: 中間変数による補助定理

ある実装 $\boldsymbol{f}_{\odot,B}(\boldsymbol{x},\boldsymbol{y})$ があるとき, 合成関数 $\boldsymbol{h}_{\odot,B}(\boldsymbol{g}_{\odot,B}(\boldsymbol{x},\boldsymbol{y}))$ に分解することができる. ただし $\boldsymbol{g}_{\odot,B} : \mathbb{IF}^2 \to \mathbb{IF}^n$ かつ $\boldsymbol{h}_{\odot,B} : \mathbb{IF}^n \to \mathbb{IF}$ である. したがって VC は合成式と (ベクトル値の) 中間変数で書き直すことができる. 例えば $VC^C_{\odot,B}$ は

$$
\begin{aligned}
&B(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \\
&\forall \mathtt{v} \in \mathbb{IF}^n, \ \mathtt{v} = \boldsymbol{g}_{\odot,B}(\boldsymbol{x},\boldsymbol{y}) \Rightarrow L(\boldsymbol{x},\boldsymbol{y},\mathtt{v}) \Rightarrow \\
&\forall \tilde{x} \in \boldsymbol{x}, \forall \tilde{y} \in \boldsymbol{y}, \ \boldsymbol{h}_{\odot,B}(\mathtt{v})
\end{aligned}
$$

と書き直せる. ここで L は引数と中間変数の論理式である. 三つ目の戦略は上の形式の VC に補助定理 L として追加する.

式 (7.6) の条件 $\mathtt{inf1} < \mathtt{inf2} \wedge \mathtt{sup1} > \mathtt{sup2}$ は検証を単純化する補助定理の例である. 実験ではいくつかの補助定理 L を追加することにより $VC^C_{*,B'}$ を証明できた. 例えば inf1 について, 以下を追加した:

$$
\begin{aligned}
&\mathtt{inf1} \neq \mathrm{NaN} \\
&\mathtt{inf1} \in \mathbb{F} \Rightarrow \underline{x} \in \mathbb{F} \wedge \overline{y} \\
&\mathtt{inf1} \in \mathbb{F} \Rightarrow \mathtt{inf1} \leq \underline{x}\,\overline{y} \\
&\mathtt{inf1} \in \mathbb{F} \Rightarrow \forall \tilde{x} \in \mathtt{x}, \forall \tilde{y} \in \mathtt{y}, \ \mathtt{inf1} \leq \tilde{x}\,\tilde{y} \\
&\forall \tilde{x} \in \mathtt{x}, \forall \tilde{y} \in \mathtt{y}, \ \mathtt{inf1} \leq \tilde{x}\,\tilde{y}
\end{aligned}
$$

この補助定理は Alt-Ergo によりそれぞれ 1.5 秒, 4.8 秒, 32 秒, 200 秒, 13.7 秒で証明できた. 同様にして sup1 についての補助定理を追加し, Alt-Ergo によりそれぞれ 1.5 秒, 8.9 秒, 14.1 秒, 216 秒, 6.0 秒で証明できた. 最終的に $VC^I_{*,B'}$ と $VC^C_{*,B'}$ は Alt-Ergo によりそれぞれ 14.6 秒と 0.1 秒で証明できた.

中間変数の導入は実装関数 $\boldsymbol{f}_{\odot,B}$ を二段階に分解することによって VC を単純化する. しかし, この分解は分岐の条件と結果の条件の間の推論を困難にする. 追加した補助定理は, 引数と中間変数の間の関係を明示的に述べ, 結びつける役割をする.

8 実験結果

6 節から 7 節で述べた浮動小数点数の区間演算プログラムの検証結果を表 2 に示す. それぞれのカラムは以下である:

- 演算子
- Why3 が生成した VC (および補助定理) の数
- SMT ソルバー (Alt-Ergo) で証明した VC の数
- 対話証明器 (Coq) で証明した VC の数
- Alt-Ergo による自動証明と, Coq による型検査にかかった実行時間 (秒)

本実験は RAM 16 GB を持つ 2.4-GHz Intel Core i5 プロセッサで行った. また検証には Why3 (バージョン 0.87.3) 上で Alt-Ergo (バージョン 1.30) と Z3 (バージョン 4.5.0), 定理証明支援系 Coq (バージョン 8.6) を用いて行い, 実行メモリは 2.5 GB に設定した.

9 関連研究

9.1 区間演算プログラムの検証

Ayad ら [1] による区間演算プログラムの検証事例がある. Ayad らは Why3 と同様な検証プラットフォーム Frama-C 上で, バックエンド証明器に SMT ソルバーと Coq を用い, 本研究とは若干異なる実装の $\mathbb{IF}$ 四則演算コードの検証に成功している.

上記のコードの乗算部分を Why3 に書き換え, 検証実験を行ったところ, 同様にすべての VC を証明できることが確認できた.

Ayad らと我々のコードを比較してみたところ, (1) Ayad らも本研究で提案する戦略 1 のように, 式 (5.2) に相当する定理を追加し証明の補助をしているが, 戦略 2–3 の補助を必要としていない, (2) Ayad らのコードはいくつかの場合を考慮しておらず限定的である, ことが分かった.

そのため, 別の区間演算プログラムや応用プログラムについて, 自動証明がうまくいかなかった場合に, 我々の提案する三戦略の効果が期待される.

9.2 CRlibm

一般の浮動小数点数を用いた数値計算プログラム

表 2　浮動小数点数境界の区間演算プログラムの検証結果

演算	VC (内, 補助定理)	Alt-Ergo (1.30)	Coq (8.6)	CPU 時間
加算	2 (0)	2	0	11.37s
減算	2 (0)	2	0	13.29s
乗算	102 (72)	86	16	947.29s
除算	37 (21)	29	8	104.47s

の証明が多数報告されている.

CRlibm [6] は効率的かつ丸めの正しさが証明された数学ライブラリであり, C の標準数学ライブラリ libm の置き換えを目的としたものである. CRlibm での正しさの証明は, 自動定理証明器 Gappa などを用いて区間解析により数値誤差を求め証明している.

CRlibm と本研究の違いは 3 つある: (1) CRlibm は演算結果が指定した丸め方向の最も近い値であるかどうかを証明しているが, 本研究の軽量かつ半自動的な証明とは違い, 複雑な証明からなる, (2) CRlibm は主に初等関数の丸めの正しさの証明であり, 本研究とは異なる性質を証明している. 本研究のアプローチを初等関数の実装コードの検証に応用することも期待出来る, (3) CRlibm は C で実装してありそのまま実行可能であるが, 本研究の検証コードはそのまま実行できない. ただし Why3 の `extract` 機能を用いれば実行可能コードへの変換は可能である.

9.3　自動証明器の浮動小数点数理論のサポート

最近, SMT ソルバーにおける浮動小数点数理論のサポートの開発が進んでいる. Conchon ら [5] は, 実数理論に基づいた浮動小数点数に関する公理系を用意し, 浮動小数点数理論に基づく述語論理式を扱う手法を提案している. その求解処理では, 戦略の一つとして非線形実数算術の求解系を呼び出して利用している. 同様の仕組みにより, 本研究の戦略も自動化することが考えられる.

10　まとめと今後の課題

自動的・対話的な定理証明器を用いた, 区間演算ライブラリの検証事例の提案手法と実験結果を示した. 本研究により, 高信頼な数値計算ライブラリに対して, それ自体の信頼性も高めることが可能になった. また, 自動証明では全ての検証条件ができない他のプログラム検証にも, 本研究で提案した三戦略による検証技法を適用することが期待される.

今後の課題としては, 証明した `WhyML` から実行コードへの変換, 同様のアプローチによる区間演算の初等関数や応用プログラムの検証, 提案した三戦略の適用の自動化がある.

謝辞 本研究の一部は科研費 15K15968 の補助を得て行った.

参考文献

[1] A. Ayad and C. Marché: Multi-Prover Verification of Floating-Point Programs, *IJCAR*, LNAI 6173, pp. 127–141, 2010.

[2] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich.: Why3: Shepherd your herd of provers, *International Workshop on Intermediate Verification Languages (Boogie)*, pp. 53–64, 2011.

[3] S. Boldo and G. Melquiond.: Flocq: A Unified Library for Proving Floating-point Algorithms in Coq, E. Antelo, D. Hough, and P. Ienne.: *IEEE Symposium on Computer Arithmetic*, pp. 243–252, 2011.

[4] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis.: Wave Equation Numerical Resolution: A Comprehensive Mechanized Proof of a C Program, *Journal of Automated Reasoning*, 50(4):423–456, 2013.

[5] S. Conchon, M. Iguernlala, K. Ji, G. Melquiond, and C. Fumex.: A Three-Tier Strategy for Reasoning About Floating-Point Numbers in SMT, *CAV*, LNCS 10427, pp. 419–435, 2017.

[6] C. Daramy-Loirat, D. Defour, F. de Dinechin, M. Gallet, N. Gast, C. Q. Lauter, and J.-M. Muller.: CR-LIBM - A library of correctly rounded elementary functions in double-precision, Version 1.0 beta 5, 2016.

[7] Moore, R.E., 1966. Interval Analysis, *Prentice-Hall*.

[8] 梅村晃広: SMT ソルバ・SMT ソルバの技術と応用, コンピュータソフトウェア, 2010, 27 (3):24–35.