Proposing Interval Arithmetic for the C++ Standard

Hervé Brönnimann      Guillaume Melquiond      Sylvain Pion

CIS Department, Polytechnic University
Arénaire, LIP, CNRS–ENSL–INRIA–UCBL
Geometrica, INRIA Sophia Antipolis

SCAN'2006: 12th GAMM - IMACS International Symposium on
Scientific Computing, Computer Arithmetic and Validated Numerics

2006-09-26
History

- 1984: C++ was born (first implementation by Stroustrup).
- 1994: The STL (Stepanov).
- 1998: C++ was standardized by ISO: “C++98” language and library.
- 1998: The Boost project was started to develop more libraries.
- 2003: A minor revision was made: “C++03”.
- 2009-2011: A new standard is planned: “C++0x”.

Major features of C++: general purpose, compatible with C, supports for abstraction and various programming paradigms, efficient “don’t pay for what you don’t use”.
History

- 1984: C++ was born (first implementation by Stroustrup).
- 1994: The STL (Stepanov).
- 1998: C++ was standardized by ISO: “C++98” language and library.
- 1998: The Boost project was started to develop more libraries.
- 2003: A minor revision was made: “C++03”.
- 2009-2011: A new standard is planned: “C++0x”.

Major features of C++: general purpose, compatible with C, supports for abstraction and various programming paradigms, efficient “don’t pay for what you don’t use”.

Hervé Brönnimann, Guillaume Melquiond, Sylvain Pion
Proposing Interval Arithmetic for the C++ Standard
The standardization process

- ISO groups 156 national standardization bodies together: AFNOR, ANSI, BSI, DIN, ...
- 95% of participants come from industry: compiler and library vendors, large C++ users, ...
- 2 meetings a year allow to make proposals, review and vote.
- Proposals are publicly available on the web: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/
- Libraries are first proposed for inclusion in TR.
- Some new features in the pipeline: multithreading, concepts, regular expressions, decimal f.p., filesystem, smart pointers, ...
Motivations for standardizing Interval Arithmetic

- Many applications: certified numerical computations, round-off error propagation control, global optimization, mathematical proofs, . . .
- Many existing implementations.
- Opportunity for better and more optimized implementations.
- Giving more exposure to reliable computations to the general programming community.
- Strengthen C++ as a language supporting numerical/scientific communities.
- Help grouping the interval community around a common basic implementation.

Previous works of standardization:
Motivations for standardizing Interval Arithmetic

- Many applications: certified numerical computations, round-off error propagation control, global optimization, mathematical proofs, . . .
- Many existing implementations.
- Opportunity for better and more optimized implementations.
- Giving more exposure to reliable computations to the general programming community.
- Strengthen C++ as a language supporting numerical/scientific communities.
- Help grouping the interval community around a common basic implementation.

Previous works of standardization:
Focus on basic interval arithmetic.
Leave out interval analysis, even linear algebra.
Only supports machine floating-point types (no MPFR).
Inclusion property verified by all functions.

Goals:
The functionality needs to be large enough to be useful.
But not too large to frighten standard library vendors.
A basic version can be done only with standard components (no need for auxiliary libraries).
It is a pure template extension to the Standard Library (no need for changes in compilers).
Functionality

- Focus on basic interval arithmetic.
- Leave out interval analysis, even linear algebra.
- Only supports machine floating-point types (no MPFR).
- Inclusion property verified by all functions.

Goals:

- The functionality needs to be large enough to be useful.
- But not too large to frighten standard library vendors.
- A basic version can be done only with standard components (no need for auxiliary libraries).
- It is a pure template extension to the Standard Library (no need for changes in compilers).
A few words on the mathematical model

- Intervals are (closed?) connected subset of real numbers: $[1, 1]$, $[3, \pi]$, $[-1.7, 5.1]$, $(-\infty, 42]$, $\emptyset$.

- Interval operations are defined by enclosing the canonical set extensions of operations on real numbers:
  \[
  \forall x \in X, \forall y \in Y, \quad x \diamond y \in X \diamond Y
  \]
  (for $X$ and $Y$ intervals and $\diamond \in \{+, -, \times, \div, \cdots\}$).

Consequences:

- $X \diamond Y$ is not uniquely defined: any connected superset of \{\[z \in \mathbb{R} | \exists x \in X, \exists y \in Y, \ z = x \diamond y\}\} qualifies.

- Empty and unbounded intervals are supported.

- Silent (no exception) treatment of out-of-domain values: $\sqrt{[-1, 4]} \supseteq [0, 2]$ and $\sqrt{[-2, -1]} \supseteq \emptyset$. 
A few words on the mathematical model

- Intervals are (closed?) connected subset of real numbers: 
  \([1, 1], [3, \pi], [-1.7, 5.1], (-\infty, 42], \emptyset\).

- Interval operations are defined by enclosing the canonical set extensions of operations on real numbers:

  \[
  \forall x \in X, \forall y \in Y, \ x \diamond y \in X \diamond Y
  \]

  (for \(X\) and \(Y\) intervals and \(\diamond \in \{+, -, \times, \div, \cdots\}\)).

Consequences:

- \(X \diamond Y\) is not uniquely defined: any connected superset of 
  \(\{z \in \mathbb{R} \mid \exists x \in X, \exists y \in Y, \ z = x \diamond y\}\) qualifies.

- Empty and unbounded intervals are supported.

- Silent (no exception) treatment of out-of-domain values:
  \([\sqrt{-1, 4}] \supseteq [0, 2]\) and \([\sqrt{-2, -1}] \supseteq \emptyset\).
A few words on the mathematical model

- Intervals are (closed?) connected subset of real numbers: $[1, 1], [3, \pi], [-1.7, 5.1], (-\infty, 42], \emptyset$.
- Interval operations are defined by enclosing the canonical set extensions of operations on real numbers:

$$\forall x \in X, \forall y \in Y, \quad x \diamond y \in X \diamond Y$$

(for $X$ and $Y$ intervals and $\diamond \in \{+,-,\times,\div,\cdots\}$).

Consequences:
- $X \diamond Y$ is not uniquely defined: any connected superset of $\{z \in \mathbb{R} | \exists x \in X, \exists y \in Y, \ z = x \diamond y\}$ qualifies.
- Empty and unbounded intervals are supported.
- Silent (no exception) treatment of out-of-domain values: $\sqrt{[-1, 4]} \supsetneq [0, 2]$ and $\sqrt{[-2, -1]} \supseteq \emptyset$. 

Hervé Brönnimann, Guillaume Melquiond, Sylvain Pion

Proposing Interval Arithmetic for the C++ Standard
A few words on the mathematical model

- Intervals are (closed?) connected subset of real numbers: 

- Interval operations are defined by enclosing the canonical set extensions of operations on real numbers:
  \[ \forall x \in X, \forall y \in Y, \quad x \diamond y \in X \diamond Y \]
  (for $X$ and $Y$ intervals and $\diamond \in \{+,-,\times,\div,\cdots\}$).

Consequences:

- $X \diamond Y$ is not uniquely defined: any connected superset of 
  \{ $z \in \mathbb{R} \mid \exists x \in X, \exists y \in Y, \quad z = x \diamond y$ \} qualifies.

- Empty and unbounded intervals are supported.

- Silent (no exception) treatment of out-of-domain values: 
  $\sqrt{[-1, 4]} \supseteq [0, 2]$ and $\sqrt{[-2, -1]} \supseteq \emptyset$. 

Hervé Brönnimann, Guillaume Melquiond, Sylvain Pion
Proposing Interval Arithmetic for the C++ Standard
A few words on the mathematical model

- Intervals are (closed?) connected subset of real numbers: $[1, 1], [3, \pi], [-1.7, 5.1], (-\infty, 42], \emptyset$.

- Interval operations are defined by enclosing the canonical set extensions of operations on real numbers:

  $$\forall x \in X, \forall y \in Y, \hspace{1em} x \diamond y \in X \diamond Y$$

  (for $X$ and $Y$ intervals and $\diamond \in \{+, -, \times, \div, \cdots\}$).

Consequences:

- $X \diamond Y$ is not uniquely defined: any connected superset of $\{z \in \mathbb{R} \mid \exists x \in X, \exists y \in Y, \hspace{1em} z = x \diamond y\}$ qualifies.

- Empty and unbounded intervals are supported.

- Silent (no exception) treatment of out-of-domain values:

  $$\sqrt{[-1, 4]} \supseteq [0, 2]\text{ and } \sqrt{[-2, -1]} \supseteq \emptyset.$$
Design overview

A template class allowing float, double, and long double as parameter. Similar to `std::complex<T>`.

```cpp
1 template < class T >
2 class interval
3 {
4   interval();
5   interval(T);
6   interval(T, T);
7   ...
8 }
```

Usage:

```cpp
1 std::interval<double> I(1,2),
2   J("[3.1,4.7]"), K;
3 K = I + J;
4 std::cout << K << std::endl;
```
Design overview

A template class allowing float, double, and long double as parameter. Similar to std::complex<T>.

```
1 template < class T >
2 class interval
3 {
4    interval();
5    interval(T);
6    interval(T, T);
7    ...
8 }
```

Usage:

```
1 std::interval<double> I(1,2),
2    J("[3.1,4.7]"), K;
3 K = I + J;
4 std::cout << K << std::endl;
```
Interval comparisons

No natural total order on intervals. Several schemes:

- Set inclusion partial order:
  \([ [1, 2] \prec [0, 3] ] = T \)
  \([ [0, 2] \prec [1, 3] ] = F \)
  \([ [0, 1] \prec [2, 3] ] = F \)

- Set extension comparisons (bool_set):
  \([ [0, 1] \prec [2, 3] ] = \{T\} \)
  \([ [0, 2] \prec [1, 3] ] = \{F, T\} \)
  \([ [0, 0] \prec \emptyset \} = \emptyset \)

- “Certain” comparisons:
  \([ [0, 1] \prec [2, 3] ] = T \)
  \([ [0, 2] \prec [1, 3] ] = F \)
  \([ [0, 0] \prec \emptyset \} = T \)

- “Possible” comparisons:
  \([ [0, 1] \prec [2, 3] ] = T \)
  \([ [0, 2] \prec [1, 3] ] = T \)
  \([ [2, 3] \prec [0, 1] ] = F \)

No default comparison. Operators are selected by namespace:

```cpp
1   interval<
2   double> A, B;
3   ...
4   using namespace certainly_ops;
5   if (0. < A && A <= B) {
6       // ∀a ∈ A, ∀b ∈ B, 0 < a ≤ b
```
Interval comparisons

No natural total order on intervals. Several schemes:

- **Set inclusion partial order:**
  \( ([1, 2] \preceq [0, 3]) = T \)
  \( ([0, 2] \preceq [1, 3]) = F \)
  \( ([0, 1] \preceq [2, 3]) = F \)

- **Set extension comparisons (bool set):**
  \( ([0, 1] \preceq [2, 3]) = \{ T \} \)
  \( ([0, 2] \preceq [1, 3]) = \{ F, T \} \)
  \( ([0, 0] \preceq \emptyset) = \emptyset \)

- **“Certain” comparisons:**
  \( ([0, 1] \preceq [2, 3]) = T \)
  \( ([0, 2] \preceq [1, 3]) = F \)
  \( ([0, 0] \preceq \emptyset) = T \)

- **“Possible” comparisons:**
  \( ([0, 1] \preceq [2, 3]) = T \)
  \( ([0, 2] \preceq [1, 3]) = T \)
  \( ([2, 3] \preceq [0, 1]) = F \)

No default comparison. Operators are selected by namespace:

```cpp
1  interval<double> A, B;
2  ...
3  using namespace certainly_ops;
4  if (0. < A && A <= B) {
5    // ∀a ∈ A, ∀b ∈ B, 0 < a ≤ b
```
Interval comparisons

No natural total order on intervals. Several schemes:

- Set inclusion partial order:
  
  \([1, 2] \not\subset [0, 3]) = T \quad [0, 2] \not\subset [1, 3]) = F \quad [0, 1] \not\subset [2, 3]) = F

- Set extension comparisons (bool_set):
  
  \([0, 1] \not\subset [2, 3]) = \{ T \} \quad [0, 2] \not\subset [1, 3]) = \{ F, T \} \quad [0, 0] \not\subset \emptyset = \emptyset

- “Certain” comparisons:
  
  \([0, 1] \not\subset [2, 3]) = T \quad [0, 2] \not\subset [1, 3]) = F \quad [0, 0] \not\subset \emptyset = T

- “Possible” comparisons:
  
  \([0, 1] \not\subset [2, 3]) = T \quad [0, 2] \not\subset [1, 3]) = T \quad [2, 3] \not\subset [0, 1]) = F

No default comparison. Operators are selected by namespace:

```cpp
1 interval<double> A, B;
2 ...
3 using namespace certainly_ops;
4 if (0. < A && A <= B) {
5     // ∀a ∈ A, ∀b ∈ B, 0 < a ≤ b
```
Interval comparisons

No natural total order on intervals. Several schemes:

- Set inclusion partial order:
  
  \((1, 2) \prec (0, 3)\) = \(T\) \quad \((0, 2) \prec (1, 3)\) = \(F\) \quad \((0, 1) \prec (2, 3)\) = \(F\)

- Set extension comparisons (bool_set):
  
  \((0, 1) \prec (2, 3)\) = \{\(T\}\} \quad \((0, 2) \prec (1, 3)\) = \{\(F, T\}\} \quad \((0, 0) \prec \emptyset\) = \(\emptyset\)

- “Certain” comparisons:
  
  \((0, 1) \prec (2, 3)\) = \(T\) \quad \((0, 2) \prec (1, 3)\) = \(F\) \quad \((0, 0) \prec \emptyset\) = \(T\)

- “Possible” comparisons:
  
  \((0, 1) \prec (2, 3)\) = \(T\) \quad \((0, 2) \prec (1, 3)\) = \(T\) \quad \((2, 3) \prec (0, 1)\) = \(F\)

No default comparison. Operators are selected by namespace:

```cpp
1    interval<double> A, B;
2    ...
3    using namespace certainly_ops;
4    if (0. < A && A <= B) {
5        // \(\forall a \in A, \forall b \in B, 0 < a \leq b\)
```
Interval comparisons

No natural total order on intervals. Several schemes:

- Set inclusion partial order:
  \([1, 2] \prec [0, 3]) = T \quad ([0, 2] \prec [1, 3]) = F \quad ([0, 1] \prec [2, 3]) = F

- Set extension comparisons (bool_set):
  \([0, 1] \prec [2, 3]) = \{ T \} \quad ([0, 2] \prec [1, 3]) = \{ F, T \} \quad ([0, 0] \prec \emptyset) = \emptyset

- “Certain” comparisons:
  \([0, 1] \prec [2, 3]) = T \quad ([0, 2] \prec [1, 3]) = F \quad ([0, 0] \prec \emptyset) = T

- “Possible” comparisons:
  \([0, 1] \prec [2, 3]) = T \quad ([0, 2] \prec [1, 3]) = T \quad ([2, 3] \prec [0, 1]) = F

No default comparison. Operators are selected by namespace:

```cpp
int interval<double> A, B;
...
using namespace certainly_ops;
if (0. < A && A <= B) {
    // ∀a ∈ A, ∀b ∈ B, 0 < a ≤ b
```
Interval comparisons

No natural total order on intervals. Several schemes:

- **Set inclusion partial order:**
  \([1, 2] \prec [0, 3] = T\)  \([0, 2] \prec [1, 3] = F\)  \([0, 1] \prec [2, 3] = F\)

- **Set extension comparisons (bool_set):**
  \([0, 1] \prec [2, 3] = \{T\}\)  \([0, 2] \prec [1, 3] = \{F, T\}\)  \([0, 0] \prec \emptyset = \emptyset\)

- **“Certain” comparisons:**
  \([0, 1] \prec [2, 3] = T\)  \([0, 2] \prec [1, 3] = F\)  \([0, 0] \prec \emptyset = T\)

- **“Possible” comparisons:**
  \([0, 1] \prec [2, 3] = T\)  \([0, 2] \prec [1, 3] = T\)  \([2, 3] \prec [0, 1] = F\)

No default comparison. Operators are selected by namespace:

```cpp
1  interval<double> A, B;
2  ...
3  using namespace certainly_ops;
4  if (0. < A && A <= B) {
5      // ∀a ∈ A, ∀b ∈ B, 0 < a ≤ b
```
Interval comparisons

Canonical set extension of boolean comparisons:

```cpp
namespace bool_set_ops {
    template <class T>
    bool_set operator< (interval<T> a, interval<T> b);
}
```

bool_set implements a multi-valued logic:

- Four states: false, true, both, none.
- Convertible to bool, possibly with an exception.
- Logical operations: |, &, ^, !, ...
Interval comparisons

Canonical set extension of boolean comparisons:

```cpp
namespace bool_set_ops {
    template < class T >
    bool_set operator< (interval<T> a, interval<T> b);
}
```

bool_set implements a multi-valued logic:

- Four states: false, true, both, none.
- Convertible to bool, possibly with an exception.
- Logical operations: |, &, ^, !, ...
List of free functions

- **Value functions:** inf, sup, midpoint, width.
- Arithmetic operators: +, -, *, /, square, sqrt, abs.
- Set operations: is_singleton, contains, overlaps, equals, intersect, hull, split, bisect.
- I/O operators << and >>.
- Forward functions: cos, log10, hypot, ...
  \[
  \text{asinh}(X) \supseteq \{ r \in [-\frac{\pi}{2}, \frac{\pi}{2}] \mid \sin r \in X \} 
  \]
- Relational functions: asin_rel, nth_root_rel, ...
  \[
  \text{asin}_{\text{rel}}(X, R) \supseteq \{ r \in R \mid \sin r \in X \} 
  \]
  \((R \text{ may be } [17, 18])\)
List of free functions

- **Value functions:** inf, sup, midpoint, width.
- **Arithmetic operators:** +, -, *, /, square, sqrt, abs.
- **Set operations:** is_singleton, contains, overlaps, equals, intersect, hull, split, bisect.
- **I/O operators** << and >>.
- **Forward functions:** cos, log10, hypot, ...
  \[
  \text{asinf}(X) \supseteq \{ r \in [-\frac{\pi}{2}, \frac{\pi}{2}] \mid \sin r \in X \}
  \]
- **Relational functions:** asinf_rel, nth_rool_rel, ...
  \[
  \text{asinf_rel}(X, R) \supseteq \{ r \in R \mid \sin r \in X \}
  \]
  
  (\(R\) may be [17, 18])
List of free functions

- Value functions: \( \text{inf}, \text{sup}, \text{midpoint}, \text{width} \).

- Arithmetic operators: +, -, *, /, square, sqrt, abs.

- Set operations: is_singleton, contains, overlaps, equals, intersect, hull, split, bisect.

- I/O operators << and >>.

- Forward functions: \( \cos, \log10, \text{hypot} \), . . .

\[
\text{asinf}(X) \supseteq \{ r \in [-\frac{\pi}{2}, \frac{\pi}{2}] \mid \sin r \in X \}
\]

- Relational functions: \( \text{asinf}_\text{rel}, \text{nth}_\text{root}_\text{rel} \), . . .

\[
\text{asinf}_\text{rel}(X, R) \supseteq \{ r \in R \mid \sin r \in X \}
\]

\((R \text{ may be } [17, 18])\)
List of free functions

- **Value functions:** $\inf$, $\sup$, midpoint, width.
- **Arithmetic operators:** $+$, $-$, $\times$, $/$, square, sqrt, abs.
- **Set operations:** is_singleton, contains, overlaps, equals, intersect, hull, split, bisect.
- **I/O operators** $\ll$ and $\gg$.
- **Forward functions:** cos, log10, hypot, ...
  \[
  \text{asin}(X) \supseteq \{ r \in [-\frac{\pi}{2}, \frac{\pi}{2}] | \sin r \in X \} 
  \]
- **Relational functions:** asin_rel, nth_root_rel, ...
  \[
  \text{asin}_\text{rel}(X, R) \supseteq \{ r \in R | \sin r \in X \} 
  \]
  ($R$ may be $[17, 18]$)
List of free functions

- **Value functions:** inf, sup, midpoint, width.
- **Arithmetic operators:** +, -, *, /, square, sqrt, abs.
- **Set operations:** is_singleton, contains, overlaps, equals, intersect, hull, split, bisect.
- **I/O operators** << and >>.
- **Forward functions:** cos, log10, hypot, ...
  \[ \text{as}
  \frac{\pi}{2}, \text{as}\frac{pi}{2} | \sin r \in X} \]
- **Relational functions:** asin_rel, nth_root_rel, ...
  \[ \text{as}
  \frac{\pi}{2}, \text{as}\frac{pi}{2} | \sin r \in X} \]

(R may be [17, 18])
List of free functions

- Value functions: `inf`, `sup`, `midpoint`, `width`.
- Arithmetic operators: `+`, `−`, `∗`, `/`, `square`, `sqrt`, `abs`.
- Set operations: `is_singleton`, `contains`, `overlaps`, `equals`, `intersect`, `hull`, `split`, `bisect`.
- I/O operators `<<` and `>>`.
- Forward functions: `cos`, `log10`, `hypot`, ...
  \[ \text{asinh}(X) \supseteq \{ r \in [-\frac{\pi}{2}, \frac{\pi}{2}] \mid \sin r \in X \} \]
- Relational functions: `asinh_rel`, `nth_root_rel`, ...
  \[ \text{asinh}_\text{rel}(X, R) \supseteq \{ r \in R \mid \sin r \in X \} \]
  \((R \text{ may be } [17, 18])\)
Optimization

Boils down to something like:

```cpp
1 d = a + b;
2 e = d + c;
```

```cpp
1 save_current_rounding_mode();
2 set_rounding_mode_to_infinity();
3 d.inf = -(-a.inf - b.inf); // assume usual trick
4 d.sup = a.sup + b.sup;
5 restore_rounding_mode(); // useless
6
7 save_current_rounding_mode(); // useless
8 set_rounding_mode_to_infinity(); // useless
9 e.inf = -(-d.inf - c.inf);
10 e.sup = d.sup + c.sup;
11 restore_rounding_mode();
```

How to optimize away redundant rounding mode changes?

- Ask the user to preset the rounding mode.
- Ask the **compiler** to do the job: user-friendly and optimistic.
Optimization

1. \( d = a + b; \)
2. \( e = d + c; \)

Boils down to something like:

1. `save_current_rounding_mode();`
2. `set_rounding_mode_to_infinity();`
3. \( d.\text{inf} = -(-a.\text{inf} - b.\text{inf}); \) // assume usual trick
4. \( d.\text{sup} = a.\text{sup} + b.\text{sup}; \)
5. `restore_rounding_mode();` // useless
6.
7. `save_current_rounding_mode();` // useless
8. `set_rounding_mode_to_infinity();` // useless
9. \( e.\text{inf} = -(-d.\text{inf} - c.\text{inf}); \)
10. \( e.\text{sup} = d.\text{sup} + c.\text{sup}; \)
11. `restore_rounding_mode();`

How to optimize away redundant rounding mode changes?
- Ask the user to preset the rounding mode.
- Ask the compiler to do the job: user-friendly and optimistic.
Optimization

\begin{verbatim}
1  d = a + b;
2  e = d + c;
\end{verbatim}

Boils down to something like:

\begin{verbatim}
1  save_current_rounding_mode();
2  set_rounding_mode_to_infinity();
3  d.inf = - (- a.inf - b.inf);  // assume usual trick
4  d.sup = a.sup + b.sup;
5  restore_rounding_mode();    // useless
6
7  save_current_rounding_mode();  // useless
8  set_rounding_mode_to_infinity();  // useless
9  e.inf = - (- d.inf - c.inf);
10 e.sup = d.sup + c.sup;
11 restore_rounding_mode();
\end{verbatim}

How to optimize away redundant rounding mode changes?

- Ask the user to preset the rounding mode.
- Ask the \texttt{compiler} to do the job: user-friendly and optimistic.
Conclusion

Current work:

- Collect comments on the first revision.
- Collect support.
- Prepare for the next meeting in October.

Mailing-list: std-interval@compgeom.poly.edu

Proposals: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/
  - n2046.pdf: bool_set
  - n2067.pdf: interval

Mail: guillaume.melquiond@ens-lyon.fr