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".
- 2004: Technical Report 1 "TR1": non-normative list of new libraries.
- 2007-2008: Technical Report 2 "TR2".
- 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".
- 2004: Technical Report 1 "TR1": non-normative list of new libraries.
- 2007-2008: Technical Report 2 "TR2".
- 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".

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:
J. Maurer draft (2001), Boost.Interval (2002), Fortran effort.

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:
J. Maurer draft (2001), Boost.Interval (2002), Fortran effort.

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).

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 $\{z \in \mathbb{R} \mid \exists x \in X,\ \exists y \in Y,\ z = x \diamond y\}$ 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 $\{z \in \mathbb{R} \mid \exists x \in X,\ \exists y \in Y,\ z = x \diamond y\}$ 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 $\{z \in \mathbb{R} \mid \exists x \in X,\ \exists y \in Y,\ z = x \diamond y\}$ 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 $\{z \in \mathbb{R} \mid \exists x \in X,\ \exists y \in Y,\ z = x \diamond y\}$ 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 $\{z \in \mathbb{R} \mid \exists x \in X,\ \exists y \in Y,\ z = x \diamond y\}$ qualifies.
- Empty and unbounded intervals are supported.
- Silent (no exception) treatment of out-of-domain values: $\sqrt{[-1, 4]} \supseteq [0, 2]$ and $\sqrt{[-2, -1]} \supseteq \emptyset$.

Design overview

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

```
template < class T >
class interval
{
  interval();
  interval(T);
  interval(T, T);
  ...
};
```

Usage:

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

Design overview

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

```
template < class T >
class interval
{
  interval();
  interval(T);
  interval(T, T);
  ...
};
```

Usage:

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

Interval comparisons

No natural total order on intervals. Several schemes:

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

```
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:

```
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:

```
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:
  $([1,2] \prec [0,3]) = T$ $\quad ([0,2] \prec [1,3]) = F$ $\quad ([0,1] \prec [2,3]) = F$
- 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:

```
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:
  $([1,2] \prec [0,3]) = T$ $\qquad$ $([0,2] \prec [1,3]) = F$ $\qquad$ $([0,1] \prec [2,3]) = F$
- Set extension comparisons (`bool_set`):
  $([0,1] \prec [2,3]) = \{T\}$ $\qquad$ $([0,2] \prec [1,3]) = \{F, T\}$ $\qquad$ $([0,0] \prec \emptyset) = \emptyset$
- "Certain" comparisons:
  $([0,1] \prec [2,3]) = T$ $\qquad$ $([0,2] \prec [1,3]) = F$ $\qquad$ $([0,0] \prec \emptyset) = T$
- "Possible" comparisons:
  $([0,1] \prec [2,3]) = T$ $\qquad$ $([0,2] \prec [1,3]) = T$ $\qquad$ $([2,3] \prec [0,1]) = F$

No default comparison. Operators are selected by namespace:

```
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:
  $([1, 2] \prec [0, 3]) = T \qquad ([0, 2] \prec [1, 3]) = F \qquad ([0, 1] \prec [2, 3]) = F$
- Set extension comparisons (`bool_set`):
  $([0, 1] \prec [2, 3]) = \{T\} \qquad ([0, 2] \prec [1, 3]) = \{F, T\} \qquad ([0, 0] \prec \emptyset) = \emptyset$
- "Certain" comparisons:
  $([0, 1] \prec [2, 3]) = T \qquad ([0, 2] \prec [1, 3]) = F \qquad ([0, 0] \prec \emptyset) = T$
- "Possible" comparisons:
  $([0, 1] \prec [2, 3]) = T \qquad ([0, 2] \prec [1, 3]) = T \qquad ([2, 3] \prec [0, 1]) = F$

No default comparison. Operators are selected by namespace:

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

Interval comparisons

Canonical set extension of boolean comparisons:

```
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:

```
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`, ...

$$\mathtt{asin}(X) \supseteq \{r \in [-\tfrac{\pi}{2}, \tfrac{pi}{2}] \mid \sin r \in X\}$$

- Relational functions: `asin_rel`, `nth_rool_rel`, ...

$$\mathtt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$

(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`, ...

$$\mathtt{asin}(X) \supseteq \{r \in [-\frac{\pi}{2}, \frac{pi}{2}] \mid \sin r \in X\}$$

- Relational functions: `asin_rel`, `nth_rool_rel`, ...

$$\mathtt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$

(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`, ...

$$\mathtt{asin}(X) \supseteq \{r \in [-\frac{\pi}{2}, \frac{pi}{2}] \mid \sin r \in X\}$$

- Relational functions: `asin_rel`, `nth_rool_rel`, ...

$$\mathtt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$

(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`, ...

$$\mathtt{asin}(X) \supseteq \{r \in [-\tfrac{\pi}{2}, \tfrac{pi}{2}] \mid \sin r \in X\}$$

- Relational functions: `asin_rel`, `nth_rool_rel`, ...

$$\mathtt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$

(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`, ...

$$\mathtt{asin}(X) \supseteq \{r \in [-\tfrac{\pi}{2}, \tfrac{pi}{2}] \mid \sin r \in X\}$$

- Relational functions: `asin_rel`, `nth_rool_rel`, ...

$$\mathtt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$

(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`, ...
$$\texttt{asin}(X) \supseteq \{r \in [-\tfrac{\pi}{2}, \tfrac{pi}{2}] \mid \sin r \in X\}$$
- Relational functions: `asin_rel`, `nth_rool_rel`, ...
$$\texttt{asin\_rel}(X, R) \supseteq \{r \in R \mid \sin r \in X\}$$
(R may be $[17, 18]$)

Optimization

```
d = a + b;
e = d + c;
```

Boils down to something like:

```
save_current_rounding_mode();
set_rounding_mode_to_infinity();
d.inf = - (- a.inf - b.inf); // assume usual trick
d.sup = a.sup + b.sup;
restore_rounding_mode();           // useless

save_current_rounding_mode();      // useless
set_rounding_mode_to_infinity();   // useless
e.inf = - (- d.inf - c.inf);
e.sup = d.sup + c.sup;
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

```
d = a + b;
e = d + c;
```

Boils down to something like:

```
save_current_rounding_mode();
set_rounding_mode_to_infinity();
d.inf = - (- a.inf - b.inf); // assume usual trick
d.sup = a.sup + b.sup;
restore_rounding_mode();           // useless

save_current_rounding_mode();      // useless
set_rounding_mode_to_infinity();   // useless
e.inf = - (- d.inf - c.inf);
e.sup = d.sup + c.sup;
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

```
d = a + b;
e = d + c;
```

Boils down to something like:

```
save_current_rounding_mode();
set_rounding_mode_to_infinity();
d.inf = - (- a.inf - b.inf); // assume usual trick
d.sup = a.sup + b.sup;
restore_rounding_mode();            // useless

save_current_rounding_mode();       // useless
set_rounding_mode_to_infinity();    // useless
e.inf = - (- d.inf - c.inf);
e.sup = d.sup + c.sup;
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.

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`