# bool_set and interval for the C++ standard

Sylvain Pion (INRIA Sophia Antipolis)

Hervé Brönnimann (Bloomberg, Polytechnic University Brooklyn)
Guillaume Melquiond (ENS Lyon)

# Plan

# Plan

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes ;-).

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes ;-).

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Why standardize Interval Arithmetic ?

- Has applications.
- Is mature and mathematically sound.
- Many existing implementations with slight variations (unifiable).
- Users care about the higher levels.
- Can be made more efficient if close to the hardware/compiler.
- Strengthen C++ as a language supporting numerical/scientific communities, and as a language supporting certified/prooved implementations.
- Help regrouping the interval community around a common basic implementation.
- Now you know why IEEE 754 provides rounding modes `;-)`.

More information at `http://www.cs.utep.edu/interval-comp/` .

# Plan

# Example : teaching floating-point to beginners

Explaining surprising behavior to beginners in floating-point computations.

```
double d = 1;
d /= 3;
if ( 3*d < 1 )
  ...
```

`interval` gives back *reliable* information.

```
using namespace std::bool_set_ops;
interval<double> d = 1;
d /= 3;
if ( 3*d < 1 )  // throws if not guaranteed !
  ...
```

or alternatively :

```
if ( certainly( 3*d < 1 ) )
```

or :

```
if ( possibly( 3*d < 1 ) )
```

# Example : teaching floating-point to beginners

Explaining surprising behavior to beginners in floating-point computations.

```
double d = 1;
d /= 3;
if ( 3*d < 1 )
  ...
```

`interval` gives back *reliable* information.

```
using namespace std::bool_set_ops;
interval<double> d = 1;
d /= 3;
if ( 3*d < 1 )  // throws if not guaranteed !
  ...
```

or alternatively :

```
if ( certainly( 3*d < 1 ) )
```

or :

```
if ( possibly( 3*d < 1 ) )
```

# Example : teaching floating-point to beginners

Explaining surprising behavior to beginners in floating-point computations.

```
double d = 1;
d /= 3;
if ( 3*d < 1 )
  ...
```

`interval` gives back *reliable* information.

```
using namespace std::bool_set_ops;
interval<double> d = 1;
d /= 3;
if ( 3*d < 1 )  // throws if not guaranteed !
  ...
```

or alternatively :

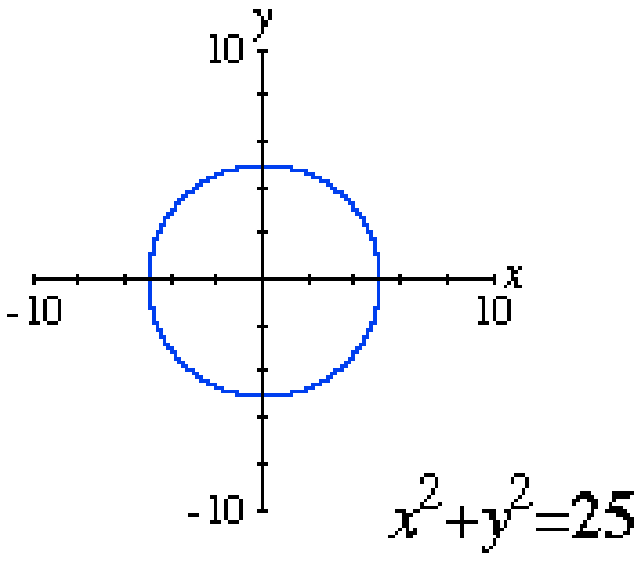
```
if ( certainly( 3*d < 1 ) )
```

or :

```
if ( possibly( 3*d < 1 ) )
```

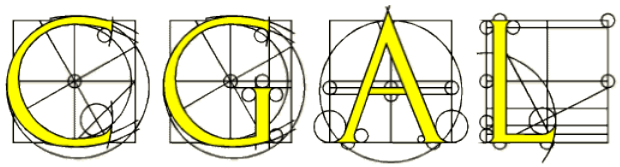
# Example : plotting functions

`GraphEq` uses interval arithmetic to reliably plot functions.




More generally : solving functions (or systems of functions) reliably and efficiently uses interval analysis (based on interval arithmetic).

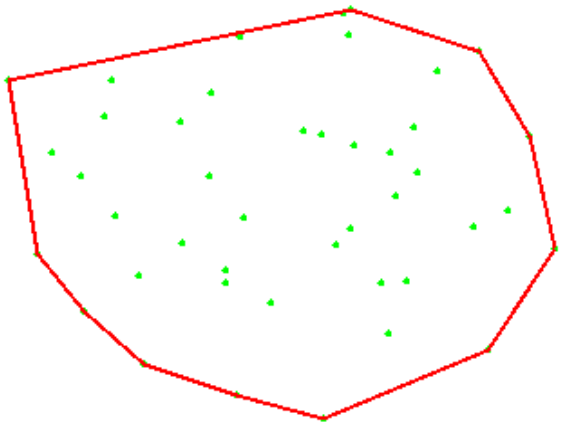
# Example : computational geometry

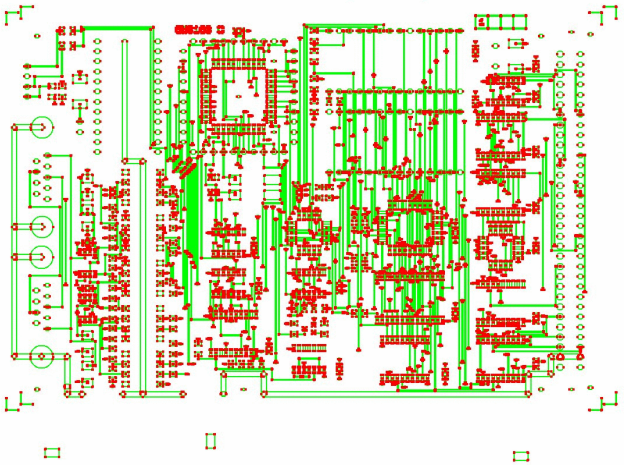


*Company* : GeometryFactory
CGAL uses interval arithmetic for `robust` and `efficient` geometric computations.

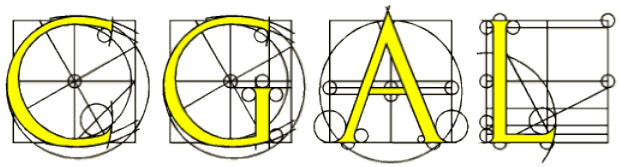
- Almost all algorithms simply don't work if numerical computations are not robust.
- It is way too slow without interval arithmetic (because of multiprecision).

Imagine you want to `sort` values, with $f(x, y) < 0$ as comparator instead of $x - y < 0$.
How would the `std::sort` function behave in presence of approximations ?
Even the most basic geometric algorithm (convex hull) does just that.





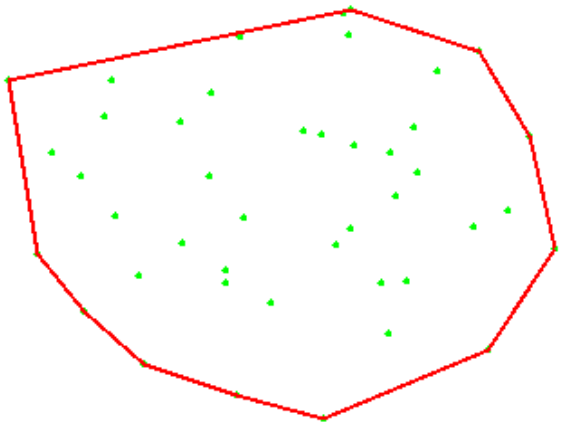
# Example : computational geometry

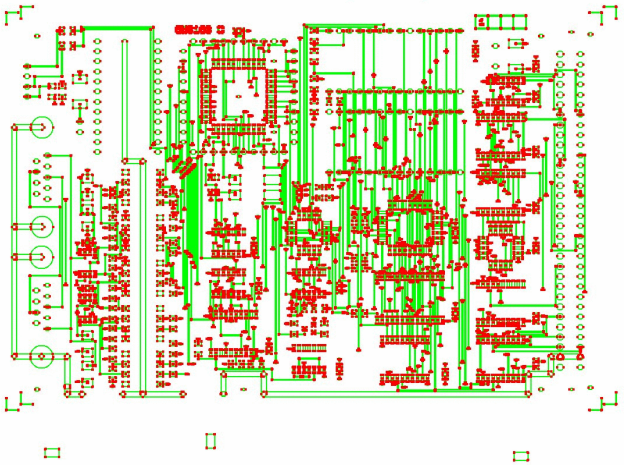


*Company* : GeometryFactory
CGAL uses interval arithmetic for `robust` and `efficient` geometric computations.

- Almost all algorithms simply don't work if numerical computations are not robust.
- It is way too slow without interval arithmetic (because of multiprecision).

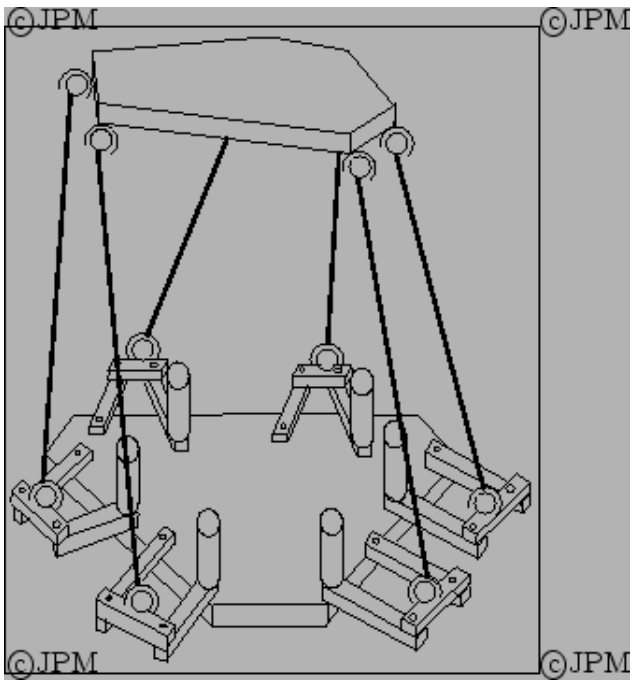
Imagine you want to `sort` values, with $f(x, y) < 0$ as comparator instead of $x - y < 0$.
How would the `std::sort` function behave in presence of approximations ?
Even the most basic geometric algorithm (convex hull) does just that.





# Example : robotics

Robots are modeled by their parameters (length of arms, angles...).
The range of these variables defines the configuration space of the robot.
Robots move, hence the parameters values change, and we need to check the feasibility of this path.
Relations are defined by some equations, and we need to proove that there is no singularity (existence of solutions to equations).
*Company* : LHR Technologies



# Example : Interval Global Solver for MS Excel

*Company* : Frontline Systems Inc.
Use intervals and functions like `sin`, `cos`, `exp`, `log`...
Links: `http://www.solver.com/press200202.htm`,
`http://www.solver.com/xlsplatform9.htm`

*Company* : Lindo Systems Inc.
Selling Optimization Modeling Tools for economics

# Plan

# Many slightly different implementations

- Sun, CGAL, GAOL, PROFIL, FILIB++, Boost.Interval...
- Building blocks : IEEE754, MPFR, CRLIBM... (directed rounding modes)

It is very easy to implement the basic functions ($+, -, \times, \div, \sqrt{}$) thanks to IEEE 754.

Using existing libraries like `MPFR`, it is easy to build other functions (`<cmath>`).

Note : IEEE 754 is under revision and plans to include more functions (with directed rounding modes)

# Many slightly different implementations

- Sun, CGAL, GAOL, PROFIL, FILIB++, Boost.Interval...
- Building blocks : IEEE754, MPFR, CRLIBM... (directed rounding modes)

It is very easy to implement the basic functions ($+, -, \times, \div, \sqrt{}$) thanks to IEEE 754.

Using existing libraries like `MPFR`, it is easy to build other functions (`<cmath>`).

Note : IEEE 754 is under revision and plans to include more functions (with directed rounding modes)

# Many slightly different implementations

- Sun, CGAL, GAOL, PROFIL, FILIB++, Boost.Interval...
- Building blocks : IEEE754, MPFR, CRLIBM... (directed rounding modes)

It is very easy to implement the basic functions ($+, -, \times, \div, \sqrt{}$) thanks to IEEE 754.

Using existing libraries like `MPFR`, it is easy to build other functions (`<cmath>`).

Note : IEEE 754 is under revision and plans to include more functions (with directed rounding modes)

# Many slightly different implementations

- Sun, CGAL, GAOL, PROFIL, FILIB++, Boost.Interval...
- Building blocks : IEEE754, MPFR, CRLIBM... (directed rounding modes)

It is very easy to implement the basic functions ($+, -, \times, \div, \sqrt{}$) thanks to IEEE 754.

Using existing libraries like `MPFR`, it is easy to build other functions (`<cmath>`).

Note : IEEE 754 is under revision and plans to include more functions (with directed rounding modes)

# Plan

# History

- **Mont-Tremblant** : First draft N1843 : positive feedback from LWG
- **Berlin** : informal update : positive feedback, LWG prefers TR2 to C++0x.
- **Portland** : split `bool_set` (N2046),
  and improved `interval` proposal (N2067 + update on wiki).

# Plan

# `bool_set` (N2046): Summary

- Very small utility tool
- Represents : {true}, {false}, {true,false} and {}.
- Boolean operations, and set operations on them.
- Relation with `interval` : serves as return type of comparisons
- Similar, but superset of Boost.Tribool (which has users)
  (more complete/consistent)
- The SQL standard has such a type
- It is "Boolean-ish"

# Questions

- Is there interest ?
- Is I/O needed ?
- && and || ?
- Throwing conversion to `bool` ?
- Name ?

# Plan

# Summary

- Template class `interval<T>`, where `T` is `float, double, long double`
- Represents the set of real values enclosed in 2 values of type T (potentially unbounded)
- Basic *inclusion* property : given $f : R \rightarrow R$, an interval extension $f(X)$ needs to verify that the resulting interval contains $\{f(x)|x \in X\}$.
- Basic functions : $+, -, \times, \div, sqrt$ (implementable using IEEE754)
- 4 different sets of comparison operators (in different namespaces)
- More functions defined : almost all (C++98) `<cmath>` functions to get a coherent more useful set. Is only more complicated if not re-using libraries like MPFR.
- Functionalities rejected : complex intervals and functions, other base type (integral, multiprecision, user-defined...), other exotic semantics of interval arithmetic, special math functions, higher levels (linear algebra, interval analysis...)...

# Summary

- Template class `interval<T>`, where `T` is `float, double, long double`
- Represents the set of real values enclosed in 2 values of type `T` (potentially unbounded)
- Basic *inclusion* property : given $f : R \rightarrow R$, an interval extension $f(X)$ needs to verify that the resulting interval contains $\{f(x)|x \in X\}$.
- Basic functions : $+, -, \times, \div, sqrt$ (implementable using IEEE754)
- 4 different sets of comparison operators (in different namespaces)
- More functions defined : almost all (C++98) `<cmath>` functions to get a coherent more useful set. Is only more complicated if not re-using libraries like MPFR.
- Functionalities rejected : complex intervals and functions, other base type (integral, multiprecision, user-defined...), other exotic semantics of interval arithmetic, special math functions, higher levels (linear algebra, interval analysis...)...

# Summary

- Template class `interval<T>`, where `T` is `float, double, long double`
- Represents the set of real values enclosed in 2 values of type `T` (potentially unbounded)
- Basic *inclusion* property : given $f : R \rightarrow R$, an interval extension $f(X)$ needs to verify that the resulting interval contains $\{f(x)|x \in X\}$.
- Basic functions : $+, -, \times, \div, sqrt$ (implementable using IEEE754)
- 4 different sets of comparison operators (in different namespaces)
- More functions defined : almost all (C++98) `<cmath>` functions to get a coherent more useful set. Is only more complicated if not re-using libraries like MPFR.
- Functionalities rejected : complex intervals and functions, other base type (integral, multiprecision, user-defined...), other exotic semantics of interval arithmetic, special math functions, higher levels (linear algebra, interval analysis...)...

# Summary

- Template class `interval<T>`, where `T` is `float, double, long double`
- Represents the set of real values enclosed in 2 values of type `T` (potentially unbounded)
- Basic *inclusion* property : given $f : R \rightarrow R$, an interval extension $f(X)$ needs to verify that the resulting interval contains $\{f(x)|x \in X\}$.
- Basic functions : $+, -, \times, \div, sqrt$ (implementable using IEEE754)
- 4 different sets of comparison operators (in different namespaces)
- More functions defined : almost all (C++98) `<cmath>` functions to get a coherent more useful set. Is only more complicated if not re-using libraries like MPFR.
- Functionalities rejected : complex intervals and functions, other base type (integral, multiprecision, user-defined...), other exotic semantics of interval arithmetic, special math functions, higher levels (linear algebra, interval analysis...)...

# Summary

- Template class `interval<T>`, where `T` is `float, double, long double`
- Represents the set of real values enclosed in 2 values of type `T` (potentially unbounded)
- Basic *inclusion* property : given $f : R \rightarrow R$, an interval extension $f(X)$ needs to verify that the resulting interval contains $\{f(x)|x \in X\}$.
- Basic functions : $+, -, \times, \div, sqrt$ (implementable using IEEE754)
- 4 different sets of comparison operators (in different namespaces)
- More functions defined : almost all (C++98) `<cmath>` functions to get a coherent more useful set. Is only more complicated if not re-using libraries like MPFR.
- Functionalities rejected : complex intervals and functions, other base type (integral, multiprecision, user-defined...), other exotic semantics of interval arithmetic, special math functions, higher levels (linear algebra, interval analysis...)...

# Code example

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

Usage:

```
std::interval<double> I(1,2), J(2,3), K;
K = I+J;
std::cout << K << std::endl;
```

# Code example

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

Usage:

```
std::interval<double> I(1,2), J(2,3), K;
K = I+J;
std::cout << K << std::endl;
```

# Differences with N1843

Actions :

- More discussion with interval experts, to get semantics and needs right.
- `std-interval` mailing list very active (70). Committee members present.
- Explicit support from the GAMM interval experts
  (forwarded by Steve to the LWG reflector)

Changes :

- More precise semantics
- More functions (all C++98's `<cmath>` basically)
- `interval<bool>` split in `bool_set` (N2046)
- Improvements on some details
- Better support for C++ implementations missing Infinity
- "error" flag to detect discontinuities and out-of-range input.

# Questions

- Is LWG still interested in it ?
- Basic set of functions versus more useful/ambitious one ?
- Detail : how to more cleanly report a discontinuity ?
- Is is ready for getting included ?
- TR2 vs C++0x ?