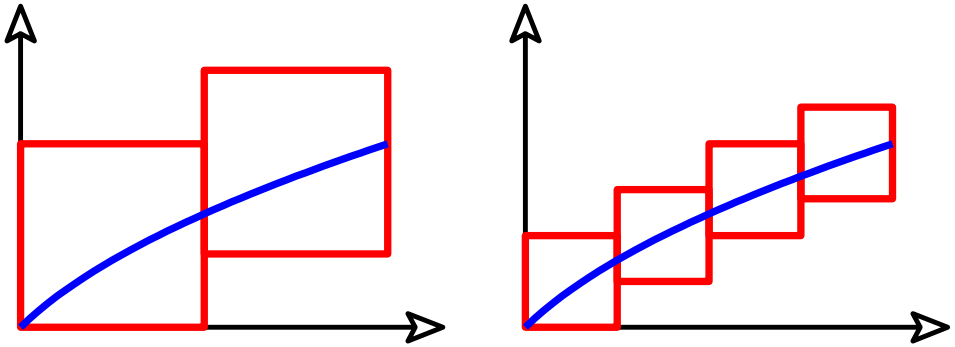
The current C++ interval standard effort

John D. Pryce
with George F. Corliss, R. Baker Kearfott,
Ned Nedialkov, Spencer Smith



Presented at Taylor Model Workshop
Boca Raton, Florida
16–19 Dec 2006

Background

A number of (not quite compatible) interval packages exist in Fortran & C++

Long running effort (incl Kearfott) to include intervals in Fortran standard – failed to get agreement for May 2004 standard

C++ effort spearheaded by Brönnimann, Melquiond, Pion (BMP) based on experience with Boost library

BMP C++ proposal Rev 0, Aug 2005; currently Rev 2, Sep 2006

Discussion forum
http://compgeom.poly.edu/mailman/listinfo/std-interval

To subscribe
mailto:std-interval-request@compgeom.poly.edu?subject=subscribe

ISL Group

A group of us (George Corliss, Baker Kearfott, Ned Nedialkov, John Pryce, Spencer Smith) aim to produce a high quality portable Interval Subroutine Library, ISL

Seed grant from EPSRC for 3 design meetings 2005–6

Grant application submitted to NSF for substantive funding to continue

Important ingredient of work so far: collaborate in Interval Standard work

ISL prefer a standard based on Containment Sets — **Csets**

But any well thought out standard is better than none

Interval Models

Intervals need an abstract model

Principle 1. *Interval arithmetic should be founded on standard set theory and real analysis*

Respect other approaches (abstract axiomatic, nonstandard analysis, . . .) but avoid them. Reasons

- Validated methods are becoming mainstream
- Principle 1 makes it easier for scientists/engineers to use them
- and to know they are doing it right

A consequence is

Principle 2. *An interval* $[a, b] = \{\, x \mid a \leq x \leq b \,\}$ *is a particular kind of subset of the number-system*

chosen because easy to **represent** and **manipulate**

Interval Models

An interval standard should start by defining the model:

- What is the number system: reals, or extended reals, . . . ?
- What are the allowed intervals, e.g. is $\emptyset$ an interval? do we allow unbounded intervals? wraparound intervals? . . .
- Give a mathematical definition of interval operations
- . . . and say how this maps to machine arithmetic

These are often skated over in documentation (hence probably design) of existing interval packages

Csets

Cset concept due to Walster and Hansen

Theory due mainly to myself in collaboration with them

Why no joint paper? Long-running disagreement over semantics

My view of Cset Interval Arithmetic has been refined by working with ISL

See *Interval Arithmetic with Containment sets* by J.D. Pryce & G.F. Corliss, Computing, Nov 2006 (online)

Cset version of BMP proposal circulated June 06 to `std-interval` but largely ignored . . . so far!

Cset mini-summary

The cset of a **real** (vector) function is obtained by taking the topological closure of its graph in a chosen **extended-real** space $(\mathbb{R}^*)^N$ and evaluating the result in the set-theory, or relation, sense

It encloses the exact range and equals it in "normal" cases

It offers a systematic way to handle interval computation with infinities.

Compare
Theorem 1 (Moore's Fundamental Theorem).
Let each elementary *function be given an interval version that for any interval inputs computes an enclosure of its exact range*
Then evaluating an arbitrary *(explicit) function* $f(x, y, \ldots)$*, using these interval elementaries, yields an enclosure of the exact range of* f *for any input intervals* $X, Y, \ldots$*,* **provided no exceptions occur**

with cset form
Theorem 2 (Fundamental Cset Theorem). *Let each* elementary *function be given an extended version that for any interval inputs computes an interval enclosure of its cset*
Then evaluating an arbitrary *explicit function* $f(x, y, \ldots)$*, using these extended elementary functions, yields an interval enclosure of the cset of* f *for any input intervals* $X, Y, \ldots$ **(exceptions do not occur)**

BMP proposal: basics

- *. . . a pure extension to the standard library. An efficient implementation of the proposal will rely on specific optimizations from the compiler [but] these are not compulsory.*

- *The header* `<interval>` *defines a class template, and numerous functions. . .*
  `interval<T>` with `T = float, double, long double` must be provided.
  User-supplied `T` permitted.

BMP proposal: Set of allowed intervals

Rev 0 had nothing about this.

Rev 1: *An object of type* `interval<T>` *represents a closed and contiguous subset of* $\mathbb{R}$*, which can be empty. If it is non-empty, it is specified by two values of type* `T`*, denoted by* $[\underline{x}, \overline{x}]$*, which can be finite or infinite, and* $\underline{x}$ *is never greater than* $\overline{x}$*. In this case, the set of real values represented is defined by* $\{\, x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x} \,\}$ *(thus excluding potential infinite values).*

Rev 2 (better): *Intervals are connected subsets of the set of real numbers. Which subsets are representable by* `interval<T>` *is implementation-defined. An implementation shall support at least the empty set* $\emptyset$*, the whole set of real numbers* $\mathbb{R}$*, and any singleton interval* $\{x\}$ *for* x *a real number representable by a floating-point number of type* `T`*.*

(Last clause creates some tie-up between a precision and the intervals supported in that precision)

This excludes the usual cset models.

It allows other representations beside "lower, upper", e.g.
- "midpoint, radius"
- special representations for infinite intervals on arithmetics without ∞

BMP proposal: definition of operations

- E.g. Addition:

  `template < class U> interval <T >& operator += (interval <U> rhs);`

  Effects: Stores an enclosure of $\{ x + y \mid x \in$ `*this` *and* $y \in$ `rhs` $\}$ *in* `*this`.
  *Returns: *this.*

  (This is basic defn from which other +'s are derived; similarly other ops)

- Much tighter than Rev 0. All operations/functions now defined in this semi-abstract way.

- **Loose evaluation** paradigm for operations not everywhere defined: see DISCTS flag below

  E.g. Division:

  `template < class U> interval <T >& operator /= (interval <U> rhs);`

  Effects: Stores an enclosure of $\{ x/y \mid x \in$ `*this` *and* $y \in$ `rhs` *and* $y \neq 0 \}$ *in* `*this`.
  *Returns: *this.*

  and Square Root:

  `template < class T > interval <T > sqrt ( interval <T > X);`

  Returns: an enclosure of $\{ \sqrt{x} \mid x \in X$ *and* $x > 0 \}$.

BMP proposal: provided functions

- Point valued functions: `inf sup midpoint radius`
- Boolean functions: `is_empty_set is_singleton contains equals overlaps comparable`
- Set-type interval functions: `intersect hull`
- Functions returning a *pair* of intervals: `split bisect`
- Numerous mathematical functions

  Some return a pair of intervals, e.g. `atan2` has a version to handle the branch cut
- "Partial" mathematical functions, see later
- More boolean functions: `is_positively_bounded is_negatively_bounded is_bounded`

  These support arithmetics that lack infinity

BMP proposal: more functions

Comparison operators

Basic ops return a `bool_set` = subset of $\{\text{false}, \text{true}\}$ e.g.

$$X > Y = \{\, x > y \mid x \in X,\ y \in Y \,\} \subseteq \{\text{false}, \text{true}\}$$

— which I like

Also "possibly", "certainly", "set inclusion" . . . comparisons, implemented via name-spaces — import the one that suits the application

Interval math relations

To support constraint propagation, e.g.

`acos_rel(X,R)` returns enclosure of $\{\, x \in R \mid \cos(x) \in X \,\}$

BMP proposal: I/O

- Interval versions of stream I/O `<<, >>` that preserve enclosure
- Also a constructor that parses a string like `"[3.1415, 3.1416]"` to an enclosing interval:

  ```
  interval(const char *s);
  ```

  Effects: Constructs an interval by extracting an interval from the NTBS pointed by s.

  but this is meant for literals within a program, not for user I/O.

Controversial: conversions

Interval functions of points, and point functions of intervals, cause inherent difficulties. Much discussed in the Forum

- `interval(T lo , T hi);`
  Effects: Constructs an interval enclosing $\{\, x \mid \mathtt{lo} \le \mathtt{x} \le \mathtt{hi} \,\}$.
  Notes: Undefined if lo is neither a finite number nor $-\infty$*, or if hi is neither a finite number nor* $+\infty$*, or if lo is not less or equal to hi.*

  E.g. `interval (`$+\infty$`, `$+\infty$`);` & `interval (3,2);` are undefined tho' in Rev 1 the latter gave `empty`, I believe

- Precision-changing interval constructor: e.g.

  $$\mathtt{interval} < \mathtt{float} > (\mathtt{interval} < \mathtt{double} > \mathtt{x});$$

  gives whole line if `x` is double's RealMax, but undefined if `x` is $+\infty$

Controversial: conversions

- `template < class T > T midpoint (interval <T > x);`
  Returns: a finite number in x when x is not empty, and an implementation-defined value otherwise.
  Notes: When x is a bounded interval, the result should approximate the real number (inf(x)+sup(x))/2.

So midpoint of $[0, +\infty)$ is implementation-defined.

Scope for honest disagreement in these and similar cases

But I am (and ISL are) unhappy with the number of "undefined if . . ." in the proposal. However Rev 2 has a new concept of "uninitialized interval" intended to make undefined situations (more?) detectable.

Controversial: DISCTS flag

Support for using Brouwer's Theorem is in Rev 2. But in a way many users may dislike. Namely all standard functions that may be discontinuous have a "partial" form:

26.6.14: In addition to returning the same results as [ordinary standard functions], the following functions raise a flag passed as a parameter when their input intervals contain values outside the domain of the mathematical function on real numbers. They never clear the flag.

E.g. `sqrtx = sqrt(x, myflag);`

For division it is a `divide` function

Opposition to a global flag by compiler-writers was vitriolic, so we have this local flag that programmer must remember to include at each relevant operation.

Status

Email from Sylvain Pion to Forum, 21 Oct 06

Here are some news from the discussions that just took place at the ISO C++ (WG21) meeting in Portland . . .

The proposals have been discussed by the Library Working Group. . . . not much details, because the committee is very busy . . .

The LWG nevertheless ran the following 3 straw polls, which results are still positive for us. I guess a more formal vote for inclusion in TR2 will take place at one of the next meetings. (TR2 itself will probably take a few years to be closed, as work on C++0x has higher priority.)

	++	+	-	--
Interest in bool_set :	3	5	0	0
bool_set for C++0x (vs TR2) :	0	1	2	4
Interest in interval for TR2 :	2	5	2	1

So both proposals are still supported, targeting TR2 rather than C++0x. We're still on track!

Summary

Various details still to work on

The cset campaign continues

But this begins to look like a solid proposal for a non-cset interval system

Meantime . . . is the **Sun** shining on it?