On Implementing the C++ Interval Library libieeep1788

Marco Nehmeier

Institute of Computer Science
University of Würzburg
Germany

SWIM 2013

Outline

Outline

IEEE Interval Standard P1788

- Founded in 2008
  - to unify interval arithmetic
  - to enlarge the acceptance of interval arithmetic
  - it specifies
    - Interval types
    - Constructors and literals
    - Basic arithmetic operations
    - Elementary functions
    - Reverse functions
    - Numerical functions
    - Comparison relations
    - Conversions
    - IO
    - ...

libieeep1788

- C++ library
  - C++11
- Framework
  - Specified interface
  - Template programming
    - Template policy (strategy pattern)
    - Type checking
    - Mixed type operations
    - ...

Goal

- Easy to use
- Easy to understand
- Easy to extend

libieeep1788

- C++ library
  - C++11
- Framework
  - Specified interface
  - Template programming
    - Template policy (strategy pattern)
    - Type checking
    - Mixed type operations
    - ...

Goal

- Easy to use
- Easy to understand
- Easy to extend

Outline

Considerations . . .

What kind of interval arithmetic?

- Classical "Moore" interval arithmetic?
  - Unbounded or empty?
- Purely algebraic?
  - CSet?
- Only set-based?
  - Kaucher or modal interval arithmetic?

Agreement . . .

- Classical "Moore" interval arithmetic
  - extended by unbounded and empty intervals?
- Operations are purely algebraic
- Set-based **and** Kaucher interval arithmetic

Considerations . . .

What kind of interval arithmetic?

- Classical "Moore" interval arithmetic?
  - Unbounded or empty?
- Purely algebraic?
  - CSet?
- Only set-based?
  - Kaucher or modal interval arithmetic?

Agreement . . .

- Classical "Moore" interval arithmetic
  - extended by unbounded and empty intervals?
- Operations are purely algebraic
- Set-based **and** Kaucher interval arithmetic

Flavors

Flavor concept

- Different interval flavors
  - Different foundational approaches to intervals
- At least one flavor provided by an implementation
  - But only one active flavor in an execution block

All flavors

- Share the same set of required and recommended operations
  - Same interface
- Extend the classical interval arithmetic
  - **Common** basis

Currently

- Set-based interval arithmetic
- Kaucher interval arithmetic

Specification levels

Level 1	Number system used by flavor **F** Set of allowed intervals in **F** Operations on **F**-intervals	Mathematical	Abstract data type
Level 2	Finite subset **T** of **F**-intervals Operations on **T**-intervals	Interval datum	Implementation
Level 3	Representations of **T**-intervals	Representation	
Level 4	Encodings	Bit string	

$\Rightarrow$ Directly influence library design!

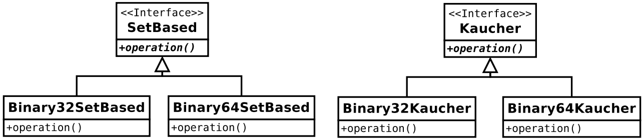
Specification levels

Level 1	Number system used by flavor **F** Set of allowed intervals in **F** Operations on **F**-intervals	Mathematical	Abstract data type
Level 2	Finite subset **T** of **F**-intervals Operations on **T**-intervals	Interval datum	Implementation
Level 3	Representations of **T**-intervals	Representation	
Level 4	Encodings	Bit string	

$\Rightarrow$ Directly influence library design!

Design decisions

Strictly following the specification levels

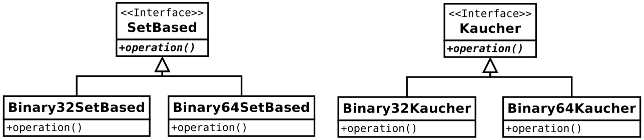


Drawbacks

- Code duplication
- No classical interval arithmetic as a **common** basis
    - No **common** interface

Design decisions

Strictly following the specification levels

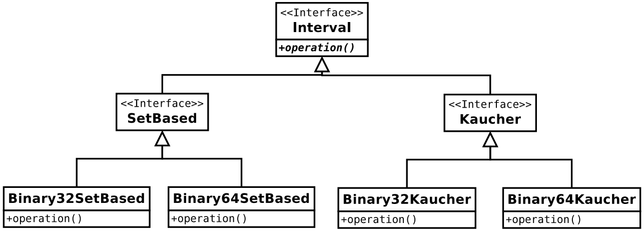


Drawbacks

- Code duplication
- No classical interval arithmetic as a **common** basis
  - No **common** interface

Design decisions

Introducing a common base class

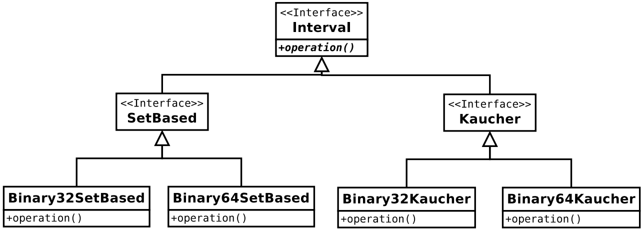


Drawbacks

- Code duplication
- Non generic class design

Design decisions

Introducing a common base class

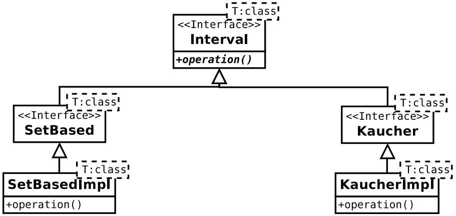


Drawbacks

- Code duplication
- Non generic class design

Design decisions

Introducing generic types

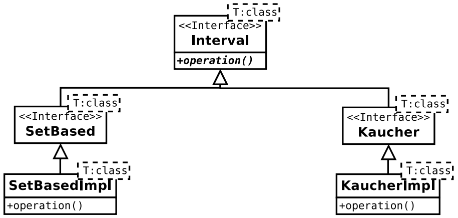


OK but ...

- Flavors are more like policies how to
  - Represent intervals
  - Perform operations
  - ...

Design decisions

Introducing generic types

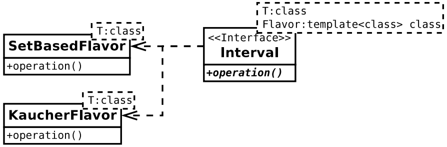


OK but ...

- Flavors are more like policies how to
  - Represent intervals
  - Perform operations
  - ...

Design decisions

Template policy pattern

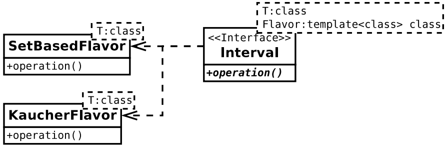


Satisfied ...

- Easy to use
- Easy to understand
- Easy to extend

Design decisions

Template policy pattern



Satisfied . . .

- Easy to use
- Easy to understand
- Easy to extend

Design decisions

Easy to use?

```
#include <iostream>

#include "p1788.hpp"
#include "flavor/setbased.hpp"

template<typename T>
using interval = p1788::interval<T, p1788::flavor::setbased_flavor>;

int main()
{
    interval<double> x(0,5);
    interval<double> y(1,2);

    interval<double> z1 = x + sqrt(y);

    // Mixed type
    interval<float>  z2 = add<interval<float>>(x + sqrt<interval<float>>(y));

    return 0;
}
```

Design decisions

Easy to understand?

- Flavor
    - Defines internal representation
        - Typedef (e.g. `std::pair<T,T>`)
    - Implements operations
- Interval
    - Interface
    - Skeleton
    - Parameterized with a Flavor policy
    - Internal representation defined by Flavor policy
    - Delegates all operations to Flavor policy

Design decisions

Easy to extend?

- Flavors are exchangeable
- Interval and Flavor are merged together at compile time using template meta programming
  - Type checking
  - Mixed type operations
  - . . .
  - Transparent for users and developers of new Flavor policies

How to implement a new Flavor policy

- Defines internal representation
- Implement required operations

Design decisions

Easy to extend?

- Flavors are exchangeable
- Interval and Flavor are merged together at compile time using template meta programming
  - Type checking
  - Mixed type operations
  - ...
  - Transparent for users and developers of new Flavor policies

How to implement a new Flavor policy

- Defines internal representation
- Implement required operations

New Flavor policy

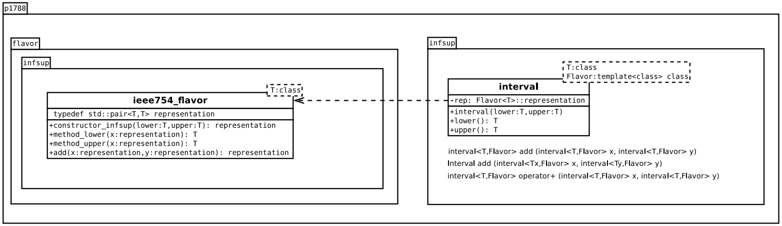
```
template<typename T>
class flavor
{
public:
    // Internal representation
    typedef std::pair<T,T> representation;

    // Constructors
    static representation constructor_infsup(T lower, T upper);

    // Methods
    static T method_lower(representation const& x);
    static T method_upper(representation const& x);
    static T method_mid(representation const& x);
    static T method_rad(representation const& x);

    // Operations
    static representation add(representation const& x, representation const& y);
    static representation sub(representation const& x, representation const& y);
}
```

Basic structure


p1788
flavor
infsup
T:class
ieee754_flavor
typedef std::pair<T,T> representation
+constructor_infsup(lower:T,upper:T): representation
+method_lower(x:representation): T
+method_upper(x:representation): T
+add(x:representation,y:representation): representation
infsup
T:class
Flavor:template<class> class
interval
-rep: Flavor<T>::representation
+interval(lower:T,upper:T)
+lower(): T
+upper(): T
interval<T,Flavor> add (interval<T,Flavor> x, interval<T,Flavor> y)
Interval add (interval<Tx,Flavor> x, interval<Ty,Flavor> y)
interval<T,Flavor> operator+ (interval<T,Flavor> x, interval<T,Flavor> y)

Outline

Decorations

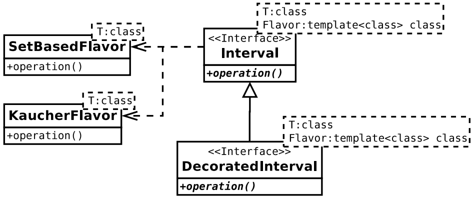
- Some algorithms are only valid if certain mathematical conditions are satisfied
  - E.g. fixpoint theorem
- Global flags vs. parallel computing
- Decorated intervals
  - Intervals are tagged with decorations
    - Pair of interval and decoration
  - Propagation order
    - History of computation

Decorations

com	common
	■ X is a bounded, nonempty subset of the domain of f
	■ f is continuous on X
	■ result is bounded
dac	defined & continuous
	■ X is a nonempty subset of the domain of f
	■ restriction of f to X is continuous
def	defined
	■ X is a nonempty subset of the domain of f
trv	trivial
	■ always true
ill	ill-formed
	■ Not an Interval

Design decisions

Should a decorated interval class be derived from the interval class?

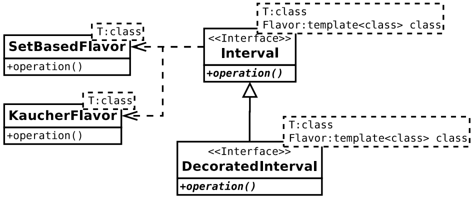


No

- Accidentally calls of bare interval operations with decorated intervals

Design decisions

Should a decorated interval class be derived from the interval class?

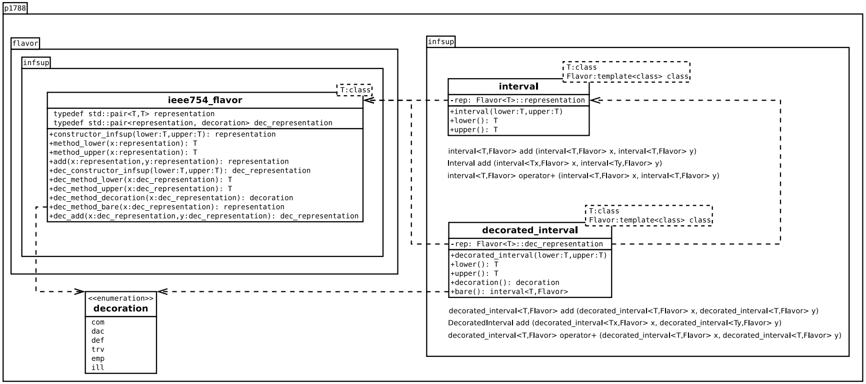


No

- Accidentally calls of bare interval operations with decorated intervals

Outline

Basic structure


p1788
flavor
infsup
T:class
ieee754_flavor
typedef std::pair<T,T> representation
typedef std::pair<representation, decoration> dec_representation
+constructor_infsup(lower:T,upper:T): representation
+method_lower(x:representation): T
+method_upper(x:representation): T
+add(x:representation,y:representation): representation
+dec_constructor_infsup(lower:T,upper:T): dec_representation
+dec_method_lower(x:dec_representation): T
+dec_method_upper(x:dec_representation): T
+dec_method_decoration(x:dec_representation): decoration
+dec_method_bare(x:dec_representation): representation
+dec_add(x:dec_representation,y:dec_representation): dec_representation
<<enumeration>>
decoration
com
dac
def
trv
emp
ill
infsup
T:class
Flavor:template<class> class
interval
-rep: Flavor<T>::representation
+interval(lower:T,upper:T)
+lower(): T
+upper(): T
interval<T,Flavor> add (interval<T,Flavor> x, interval<T,Flavor> y)
Interval add (interval<Tx,Flavor> x, interval<Ty,Flavor> y)
interval<T,Flavor> operator+ (interval<T,Flavor> x, interval<T,Flavor> y)
T:class
Flavor:template<class> class
decorated_interval
-rep: Flavor<T>::dec_representation
+decorated_interval(lower:T,upper:T)
+lower(): T
+upper(): T
+decoration(): decoration
+bare(): interval<T,Flavor>
decorated_interval<T,Flavor> add (decorated_interval<T,Flavor> x, decorated_interval<T,Flavor> y)
DecoratedInterval add (decorated_interval<Tx,Flavor> x, decorated_interval<Ty,Flavor> y)
decorated_interval<T,Flavor> operator+ (decorated_interval<T,Flavor> x, decorated_interval<T,Flavor> y)

Our Work

Current:

- Strong development
- Set based flavors
  - MPFR
  - libieee754-2008

Future:

- Kaucher flavors

Goal:

- Reference implementation
- Framework for other developers

Questions ?

Marco Nehmeier
nehmeier@informatik.uni-wuerzburg.de