Comments and Suggestions for the Proposed Standard Interval Library for C++

Workshop on New Perspectives in Enclosure Methods
J. Wolff v. Gudenberg
Dagstuhl Oct 2005

The participants of the workshop - including some of the authors of C-XSC and filib++ - have been discussing the proposal in moderate detail and came to the decision to strongly support the proposal for standardizing interval arithmetic in C++.

Some comments in favour of the proposal

1. We support the decision to offer a template with a typename parameter for one of the real arithmetic types.
2. Hardware rounding should not be visible to the user.
3. Least bit accuracy is not necessary but inclusion is.
4. The empty interval is ok, but operations with empty intervals must be clearly defined. (see below)
5. exception free mode producing result intervals containing infinities is an acceptable decision, but we must be sure that no containment errors occur (see below)
6. We accept the interval bool type

Some problems with the proposal

1. The output operator << should satisfy inclusion property, as well as input operator should perform proper outwardly directed rounding
   Reason: Users will be puzzled otherwise.
2. Comparison operators <= etc. for intervals should not be defined, only functions
   Reason: We have different comparisons and the user should be able to choose his favorite version. Filib++ users, e.g. are used that <= means subset inclusion.
3. Some cases are not considered in the specification of the arithmetic operations
   See addition for example:
   If the value *lhs* of *this prior to the addition is non empty, *this contains [*xl*+*yl*,*xu*+*yu*] where *lhs* = [*xl*; *xu*] and *rhs* = [*yl*;*yu*] and all operations are computed exactly, and this->empty() is true otherwise.
   Otherwise is unclear, rhs = empty would be clearer

4. The specification of the division is wrong, at least incomprehensible.
   Stores an empty interval<T> in *this if rhs is empty or the singleton interval<T>(T(0)),
   otherwise does not change *this if it already contains interval<T>(T(0))
   otherwise stores interval<T>::whole() in *this if rhs strictly contains T(0),
   otherwise divides the interval *this by the interval value rhs and stores the result in *this.

   What does that mean ?? the second line ?
   It should return whole, if 0 is contained in this and in rhs !

Some further suggestions

1. We think it is very important that the existing libraries boost and filib++,e.g. can be rebuild on top of the new standard and that users of the advanced features of

these libraries will not be compromised. From the filib++ point of view we will define aliases if the function name differs and implement the new functions.

2. We think it will be appropriate to give a reference to C-XSC in the introduction of the proposal. Here is a link
   http://www.math.uni-wuppertal.de/org/WRST/literatur/cxsc_docu.html
3. The feature that std::set<interval> is possible is of minor importance.
4. We also would like to propose elementary functions. Filib++ provides a reference implementation. But perhaps it is cleverer to postpone this for the second step.