Interval Comparisons and Lattice Operations based on the Interval Overlapping Relation

Marco Nehmeier, Jürgen Wolff von Gudenberg

October 26, 2010

1 Introduction

In this paper we start with the definition of the interval overlapping relation showing the 13 different cases that may occur when to compare two non-empty intervals. This definition and especially the naming follow Allen who, in [1] defined this relation in a temporal logic setting. Other names may be chosen.
This paper serves 2 purposes:

1. to define a solid foundation for interval comparisons and lattice operations
2. to provide an advanced interface for language embedding and application programs.

Based on the overlapping relation the paper then identifies 7 atomic independent interval operations, which form a base for the binary relation algebra (BRA) for intervals, [5].
The atomic operations can be used to compute the most commonly used interval comparisons, as those have been proposed in motion 13 [4]. Hence, this paper contributes to the rationale of motion 13.
The next section introduces the necessary lattice operations which are closely related with comparisons. These also can be derived from the overlapping relation.
In [6] we have shown that the general overlapping relation can be efficiently implemented in hardware. 4 bits are sufficient to encode the different states. Together with an object oriented interface allowing the direct access to the states (cases) this will presumably lead to a better performance of application programs.

1.1 Abstraction Levels

Concerning the levels of abstraction from level 1 that deals with real numbers and sets, via level 2 providing an abstract data type Interval with floating-point bounds, we come to the level 3 with executable representation, we try to stay as general or abstract as possible. Our main definition (Table 1) clearly is level 1. In our opinion that also holds for Table 2. Since P1788 defines the standard for floating-point intervals, we do need the level 2 tables 3 and 7.

2 Interval Overlapping Relation

Let Q be the finite set of states representing the 13 different possible situations of relative position of 2 non-empty intervals on the real line.

Definition 1 (Interval Overlapping) *The overlapping relation for two non-empty intervals is defined by the mapping*

$$\Join : \overline{\mathbb{IR}} \times \overline{\mathbb{IR}} \to Q \tag{1}$$

$$A \Join B \mapsto q_i \in Q\ ,\ \ i = 1 \ldots 13 \tag{2}$$

where the predicates characterizing the q_i are given in Table 1

State	Definition
before	$\forall_a \forall_b\ a < b$
meets	$\forall_a \forall_b\ a \leq b \wedge\ \exists_{a'} \forall_b\ a' < b\ \wedge\ \exists_{a''} \exists_{b'}\ a'' = b'$
overlaps	$\exists_{a'} \forall_b\ a' < b \wedge\ \exists_{b'} \forall_a\ a < b'\ \wedge\ \exists_{a''} \exists_{b''}\ b'' < a''$
starts	$\exists_{a'} \forall_b\ a' \leq b\ \wedge \exists_{b'} \forall_a\ b' \leq a\ \wedge \exists_{b''} \forall_a\ a < b''$
containedBy	$\exists_{b'} \forall_a\ b' < a \wedge \exists_{b''} \forall_a\ a < b''$
finishes	$\exists_{b'} \forall_a\ b' < a\ \wedge\ \exists_{a'} \forall_b\ b \leq a'\ \wedge\ \exists_{b''} \forall_a\ a \leq b''$
equal	$\forall_a \exists_{b'}\ a = b'\ \wedge\ \forall_b \exists_{a'}\ b = a'$
finishedBy	$\exists_{a'} \forall_b\ a' < b\ \wedge\ \exists_{b'} \forall_a\ a \leq b'\ \wedge\ \exists_{a''} \forall_b\ b \leq a''$
contains	$\exists_{a'} \forall_b\ a' < b\ \wedge\ \exists_{a''} \forall_b\ b < a''$
startedBy	$\exists_{b'} \forall_a\ b' \leq a\ \wedge\ \exists_{a'} \forall_b\ a' \leq b\ \wedge\ \exists_{a''} \forall_b\ b < a''$
overlappedBy	$\exists_{b'} \forall_a\ b' < a \wedge\ \exists_{a'} \forall_b\ b < a'\ \wedge\ \exists_{b''} \exists_{a''}\ a'' < b''$
metBy	$\forall_b \forall_a\ b \leq a \wedge\ \exists_{b'} \exists_{a'}\ b' = a'\ \wedge\ \exists_{b''} \forall a\ b'' < a$
after	$\forall_b \forall_a\ b < a$

Table 1: The 13 different states of interval overlapping situations

In order to keep the conditions readable we have omitted parentheses, the priority of the operators is assumed appropriately. We qualify each "for all" quantifier with the plain letter and each "exists" quantifier with a primed letter, we further do not spell out the interval, i.e. we write $\forall_b$ instead of $\forall_{b \in B}$ etc.

Remark 1 *The active or passive verb-forms are kind of inverse or reverse:* A activates $B \equiv B$ activatedBy A

Column 2 of Table 1 contains the set theoretic (level 1) predicates. The single states are obtained by shifting A from left to right and noticing each change of the predicate, see Figure 1. The three columns correspond (from left to right) to $width(A)(> \mid = \mid <) width(B)$, respectively.

Remark 2 *Since containment is important, we group the situation with the point interval $A = [a, a], a = \underline{b}$ under* "starts" *and not under* "meets". *For the same reason we prefer the situation $A = [a, a], a = \overline{b}$ to belong to* "finishes" *instead of* "metBy".

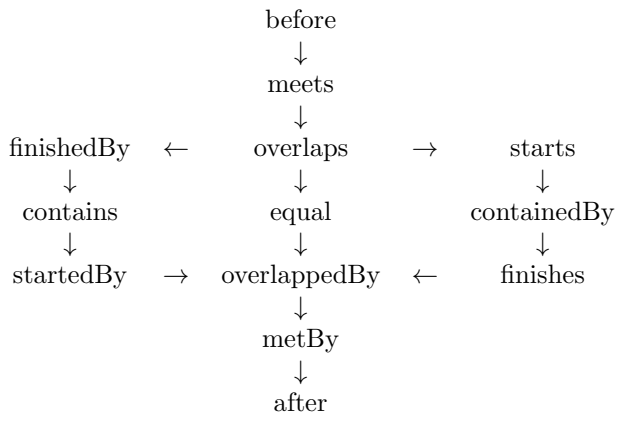


Figure 1: Shifting A from left to right

Remark 3 *Empty sets are commonly used only together with containment.*

$$\begin{aligned} \emptyset \circledcirc B \mapsto &\quad \text{containedBy} \\ A \circledcirc \emptyset \mapsto &\quad \text{contains} \\ \emptyset \circledcirc \emptyset \mapsto &\quad \text{equal} \end{aligned}$$

These values can also be obtained by the formulas in Table 1.

When we extend our mapping to empty intervals, we technically add three new states (fstEmpty, sndEmpty, and bothEmpty) to the range Q. Note that these new states are some kind of intermediate in the sense that they are internal states only used for the implementation of the exceptional occurence of an empty set. We thus obtain a complete partition of the domain $\overline{\mathbb{IR}} \times \overline{\mathbb{IR}}$ into 16 parts or cases.
Table 2 presents this partition and illustrates the definition by deducing conditions for the endpoints and drawing sketches for each state. Those old states that are meaningful for empty intervals, have been split and renamed by appending a "P".

Remark 4 *The states where the characteristic predicates deliver "true", if either operand is empty, are split into 2 or more states.*

$$\begin{aligned} \text{before} &= \text{beforeP} \vee \text{fstEmpty} \vee \text{sndEmpty} \vee \text{bothEmpty} \\ \text{containedBy} &= \text{containedByP} \vee \text{fstEmpty} \\ \text{equal} &= \text{equalP} \vee \text{bothEmpty} \\ \text{contains} &= \text{containsP} \vee \text{sndEmpty} \\ \text{after} &= \text{afterP} \vee \text{fstEmpty} \vee \text{sndEmpty} \vee \text{bothEmpty} \end{aligned}$$

The illustrations in column 3 and following of Table 2 display the interval A on the bottom and the interval B on the top. This table provides the level 2 interface that exactly specifies the conditions on the endpoints. Note that we are using the endpoints for specification of the relation, and not necessarily for representation of the intervals.

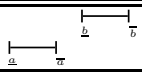
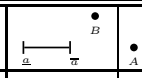
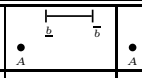
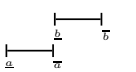
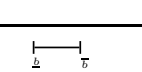
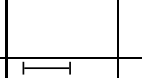
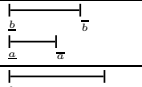
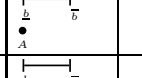
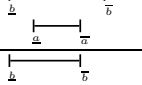
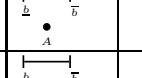
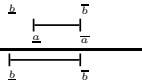
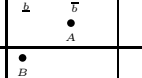
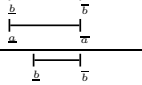
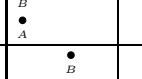
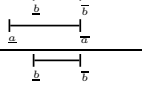
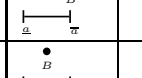
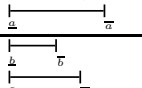
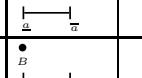
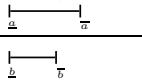
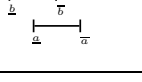
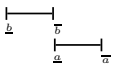
State	Condition				
beforeP	$\overline{a} < \underline{b}$				
meets	$\underline{a} < \overline{a}$ $\overline{a} = \underline{b}$ $\underline{b} < \overline{b}$				
overlaps	$\underline{a} < \underline{b}$ $\underline{b} < \overline{a}$ $\overline{a} < \overline{b}$				
starts	$\underline{a} = \underline{b}$ $\overline{a} < \overline{b}$				
containedByP	$\underline{b} < \underline{a}$ $\overline{a} < \overline{b}$				
finishes	$\underline{b} < \underline{a}$ $\overline{a} = \overline{b}$				
equalP	$\underline{a} = \underline{b}$ $\overline{a} = \overline{b}$				
finishedBy	$\underline{a} < \underline{b}$ $\overline{b} = \overline{a}$				
containsP	$\underline{a} < \underline{b}$ $\overline{b} < \overline{a}$				
startedBy	$\underline{b} = \underline{a}$ $\overline{b} < \overline{a}$				
overlappedBy	$\underline{b} < \underline{a}$ $\underline{a} < \overline{b}$ $\overline{b} < \overline{a}$				
metBy	$\underline{b} < \overline{b}$ $\overline{b} = \underline{a}$ $\underline{a} < \overline{a}$				
afterP	$\overline{b} < \underline{a}$				
fstEmpty	$A = \emptyset$ $B \neq \emptyset$				
sndEmpty	$A \neq \emptyset$ $B = \emptyset$				
bothEmpty	$A = \emptyset$ $B = \emptyset$				

Table 2: Partition of the input space

The 16 states of Table 2 can be represented by a 4-bit string. The evaluation of the conditions can be accelerated by a hardware unit consisting out of parallel comparators, as have been shown in [6]. That leads to an efficient evaluation of interval comparisons by table lookup, see Table 4.
Because our definition of unbounded intervals allows to use $-\infty$, or $+\infty$ as endpoints, and because the endpoints may be used in comparisons fulfilling
the level 2 rule $\infty = \infty$ and
the level 1 rule $\forall_{x \in \mathbb{R}}(-\infty < x \text{ and } x < +\infty)$,
we apply these formulas for unbounded intervals as well.

Remark 5 *For unbounded intervals we obtain*

$$[\underline{a}, \infty] \circledcirc [\underline{b}, \infty] = \begin{cases} \text{finishedBy} & : \quad \underline{a} < \underline{b} \\ \text{equal} & : \quad \underline{a} = \underline{b} \\ \text{finishes} & : \quad \underline{a} > \underline{b} \end{cases}$$

$$[-\infty, \overline{a}] \circledcirc [-\infty, \overline{b}] = \begin{cases} \text{starts} & : \quad \overline{a} < \overline{b} \\ \text{equal} & : \quad \overline{a} = \overline{b} \\ \text{startedBy} & : \quad \overline{a} > \overline{b} \end{cases}$$

$$[-\infty, \infty] \circledcirc [-\infty, \infty] = \{ \text{ equal}$$

Note that these rules actually ought to depend on the decorations of the operands which are not treated in this paper, see [3]. In this bare format the semantics may defer from what is expected in topological view.

3 Standard Interval Comparisons

Remark 6 (Notation) *From now on we carefully distinguish between*

- *(atomic)* ***operations*** *to check the state of overlapping,*
- ***comparisons*** *to be called for two intervals, or*
- ***functions*** *or* ***relations*** *that have no specified meaning.*

We have at least three alternatives to define interval comparisons based on the overlapping relation.

1. Define a data type `IntervalOverlapping` providing access to the 13 states.
2. Define a comparison for each of the 13 atomic operations.
3. Define a basic set of commonly used comparisons.

The first alternative has the advantage that the precise information about the relative positions of the two interval operands may be exploited to speed up evaluations of related comparisons.
The disadvantage is the new kind of interface defining an object instead of a set of boolean functions or operators.
We will discuss that approach in section 5. A hardware implementation is introduced in [6].

3.1 Atomic Operations

The second alternative provides 13 operations, denoted by the name of the state. By Remark 1 it is sufficient to provide 7 operations corresponding to the first 7 rows in the tables.
If these atomic operations are considered as independent, there treatment of empty sets has to be specified according to Table 1

Remark 7 *Atomic operations for empty arguments*
According to Table 1

- "before" *and* "after" *return true, if at least one operand is empty.*
- "meets" , "overlaps" , "starts" *and* "finishes" *return false, if at least one operand is empty.*
- "equal" *returns true for two empty operands.*
- "containedBy" *returns true, if the first operand is empty.*
- "contains" *returns true, if the second operand is empty.*

Remark 8 *Properties of atomic operations*

- "equal" *is an equivalence relation.*
- *All others are asymmetric* ($a \sqsubset b \implies b \not\sqsubset a$).
- "before", "starts", "containedBy", *and* "finishes" *are transitive.*

These 7 atomic operations can be used as a base for generating a binary relation algebra (BRA) of $2^7 = 128$ interval comparisons, as proposed in [5]. We think that this opportunity is a challenge for the sophisticated programmer, but the every day user shall be given a simpler interface consisting of functions or operators.
However, the 7 atomic operations are obviously not the best candidates for comparisons of the standard, because they are not partial orders. (A partial order is reflexive, antisymmetric and transitive.) If we consider the isolated operations, we lose the context of the overlapping relation, that tells us to reuse the result for several related comparisons.
On one hand the overlapping relation is too detailed, but on the other hand it keeps the context in order to make related comparisons simpler.

3.2 Standard Comparisons

Hence, starting with the overlapping relation, we will now derive the interval comparisons. We perform the overlapping relation and then put together several related states.
Interval arithmetic is containment arithmetic. Hence, a comparison based on the operation

- A containedBy B
  checking whether A is contained in the interior of B is mandatory.

- Another important comparison is the test whether two intervals are disjoint. It is delivered by
  A before B
  together with its reverse "after".

- The third basic comparison is the "overlaps" test.
  It is used for interval ranking in minimization problems.

These 3 operations together with their reverses are characterized in Table 1 as those states which can be described only with the "strictly less" operator $<$.
All the others compare at least one bound for equality. Hence special cases for singleton or point intervals have to be considered. In Table 1 we have carefully paid attention that the cases describe a complete partition of the argument space.
We add the related reflexive relations where $\leq$ replaces $<$.
Table 3 displays our suggestion for a basic set of interval comparisons now using operator symbols instead of names[1]. The 3rd column lists the set of overlapping states whose union describes the part of the domain that delivers "true" for the comparison.
The suggested comparisons are the most commonly used interval comparisons, any way. Another justification is given by Kulisch in [4].

Comparison	Implementation[2]	Definition
$A = B$	$\underline{a} = \underline{b} \wedge \overline{a} = \overline{b}$	equal
$A \subset B$	$\underline{b} < \underline{a} \wedge \overline{a} < \overline{b}$	containedBy
$A < B$	$\underline{a} < \underline{b} \wedge \overline{a} < \overline{b}$	(before $\wedge \neg$containedBy) $\vee$ meets $\vee$ overlaps
$A \prec B$	$\overline{a} < \underline{b}$	before $\wedge \neg$containedBy
$A \subseteq B$	$\underline{b} \leq \underline{a} \wedge \overline{a} \leq \overline{b}$	$A \subset B \vee$ equal $\vee$ starts $\vee$ finishes
$A \leq B$	$\underline{a} \leq \underline{b} \wedge \overline{a} \leq \overline{b}$	$A < B \vee$ starts $\vee$ equal $\vee$ finishedBy
$A \preceq B$	$\overline{a} \leq \underline{b}$	$A \prec B \vee$ meets $\vee$ (equal $\wedge$ isSingleton(A))

Table 3: standard comparisons

In Table 3 the states are considered as characteristic predicates introduced in Table 1 describing the subset of the set of interval pairs that evaluates to "true". Hence the $\vee$ for states (predicates) corresponds to a $\cup$ for subsets.

Remark 9 *Note that we need an additional test for a point interval.*

Remark 10 *The comparisons $<, \prec, \leq, \preceq$ deliver "false", if at least one operand is the empty interval.*

Proposition 1 *In Table 3 columns 2 and 3 are equivalent for non-empty intervals.*

Proof: We have to prove that Table 3 is correct. That means that for columns 2 and 3 the subsets S and T of $\overline{\mathbb{IR}} \times \overline{\mathbb{IR}}$ that are declared by the conditions are identical.

[1] Clearly these names are not normative but taken as a short notation in this paper.
[2] For non-empty intervals.

We outline the proof of row 3 ($A < B$).

$$\begin{aligned}
S = S_< :&= \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b}\} \\
&= \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b} \wedge true\} \\
&= \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b} \wedge (\overline{a} < \underline{b} \vee \overline{a} = \underline{b} \vee \underline{b} < \overline{a})\} \\
&= \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b} \wedge (\overline{a} < \underline{b})\} \\
&\cup \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b} \wedge (\overline{a} = \underline{b})\} \\
&\cup \{(A,B) | \underline{a} < \underline{b} \wedge \overline{a} < \overline{b} \wedge (\underline{b} < \overline{a})\} \\
&= T_{\text{before}} \cup T_{\text{meets}} \cup T_{\text{overlaps}}
\end{aligned}$$

For non-empty intervals we obviously have: before $\Rightarrow \neg$ containedBy. □

Proposition 2 *If one of the operands A or B is empty for the comparisons defined in Table 3 the result is "false" except for the two cases*

- $\emptyset = \emptyset$
- $\emptyset \subset B$ *with* $B \neq \emptyset$

which deliver "true".

Proof: See Table 1. □

Remark 11 *Let $\sqsubset$ be an asymmetric relation. Then its related reflexive partial order relation $\sqsubseteq$ is obtained by replacing the $<$ operator with the $\leq$ operator. The well known equation rule $x \sqsubseteq y \wedge x \neq y \iff x \sqsubset y$ does not hold, because we replace twice.*

Remark 12 *$\subseteq$ and $\leq$ are partial orders, whereas $\preceq$ is reflexive and antisymmetric only for point intervals.*

$$A \subseteq B \wedge B \subseteq A \implies A = B \tag{3}$$
$$A \leq B \wedge B \leq A \implies A = B \tag{4}$$
$$A \preceq B \wedge B \preceq A \implies A = B = [a,a] \tag{5}$$
$$A \preceq A \implies A = [a,a] \tag{6}$$

In Table 3 we composed the comparisons as a disjunction of the state predicates. Or, in other words sets fulfilling the state predicates were united to the set fulfilling the comparisons.
In practice, the overlapping state of two intervals provides enough information in order to determine the result of a comparison as well as a lattice operation. Table 4 displays the situation that the overlapping state out of the 16 states in the complete partition of Table 2 is known and we have to check whether the comparisons are true. Here "t" stands for true, no entry for false, "r" for reverse, and "s" means true in case of a singleton interval.
Due to remark 1 Table 4 is symmetric with respect to the center rows, t's and r's exchanged.

$A \circledcirc B$	$A \subset B$	$A \prec B$	$A < B$	$A \subseteq B$	$A \preceq B$	$A \leq B$	$A = B$
fstEmpty	t			t			
beforeP		t	t		t	t	
meets			t		t	t	
overlaps			t			t	
starts				t	$s(A)$	t	
containedByP	t			t			
finishes				t		r	
equalP				t	s	t	t
bothEmpty	t			t	t	t	t
finishedBy				r		t	
containsP	r			r			
startedBy				r	$s(B)$	r	
overlappedBy			r			r	
metBy			r		r	r	
afterP		r	r		r	r	
sndEmpty	r			r			

Table 4: standard comparisons computed with overlapping relation

Proposition 3 *Table 4 displays a correct way to implement the interval comparisons.*

Proof: We have to show that Table 4 is correct.
Let us look into row 4:
The characteristic predicate for the state "starts" is

$$\underline{a} = \underline{b} \wedge \overline{a} < \overline{b} \tag{7}$$

The first condition does not hold for the first 3 irreflexive comparisons. But it holds for the reflexive cases. So we have to check the second condition. It obviously holds for containment and the $\leq$ comparison, The $\preceq$ operator is more subtle:
We have (7) and we have to show

$$\overline{a} \leq \underline{b} \tag{8}$$

We first treat the general case where neither A nor B is a point interval.

$$\underline{a} = \underline{b} \wedge \overline{a} < \overline{b} \wedge \underline{a} < \overline{a} \Rightarrow$$
$$\underline{b} = \underline{a} < \overline{a} \Rightarrow \neg(8)$$

That proof also holds if B is a point interval. But the situation changes, if A is a singleton.

$$\underline{a} = \overline{a} = \underline{b} \tag{9}$$

and (8) is valid.
In the same manner we can prove the correctness of the other rows. □

3.3 Another set of comparisons

Another set of comparisons is used in the Fortran 95 language or in Sun's interval extensions. Those are divided into 3 groups, the "set", "possibly" and "certainly" comparisons. They can easily be obtained from the standard comparisons and thus from the overlapping relation. We show the relationship in 2 tables.

Proposition 4 *Table 5 shows how the set of Fortran 95 relations can be realized with the standard comparisons [2] and, hence, by use of the overlapping relation. Table 6 displays their realization.*

Fortran 95 relation	Level 2 definition	Standard comparisons
isDisjoint(A,B)	$\overline{a} < \underline{b} \vee \overline{b} < \underline{a}$	$A \prec B \vee B \prec A$
isInterior(A,B)	$\underline{b} < \underline{a} \wedge \overline{a} < \overline{b}$	$A \subset B$
isProperSubset(A,B)	$(\underline{b} \leq \underline{a} \wedge \overline{a} < \overline{b})$ $\vee(\underline{b} < \underline{a} \wedge \overline{a} \leq \overline{b})$	$A \subseteq B \wedge \neg(A = B)$
isSubset(A,B)	$\underline{b} \leq \underline{a} \wedge \overline{a} \leq \overline{b}$	$A \subseteq B$
setIsLess(A,B)	$\underline{a} < \underline{b} \wedge \overline{a} < \overline{b}$	$A < B$
setIsLessOrEqual(A,B)	$\underline{a} \leq \underline{b} \wedge \overline{a} \leq \overline{b}$	$A \leq B$
setIsEqual(A,B)	$\underline{a} = \underline{b} \wedge \overline{a} = \overline{b}$	$A = B$
certainlyIsLess(A,B)	$\overline{a} < \underline{b}$	$A \prec B$
certainlyIsLessOrEqual(A,B)	$\overline{a} \leq \underline{b}$	$A \preceq B$
certainlyIsEqual(A,B)	$\overline{b} \leq \underline{a} \wedge \overline{a} \leq \underline{b}$	$A \preceq B \wedge B \preceq A$
possiblyIsLess(A,B)	$\underline{a} < \overline{b}$	$\neg(B \preceq A)$
possiblyIsLessOrEqual(A,B)	$\underline{a} \leq \overline{b}$	$\neg(B \prec A)$
possiblyIsEqual(A,B)	$\underline{a} \leq \overline{b} \vee \underline{b} \leq \overline{a}$	$\neg(A \prec B \vee B \prec A)$

Table 5: Fortran 95 relations via standard comparisons

4 Lattice Operations

A lattice is a partially ordered set where each pair of elements has a *meet*[3] and a *join*. If this property holds for any (bounded) subset of elements, the lattice is called (conditionally) complete. Hence, a proper choice of *meet* and *join* as commutative, associative and absorbing binary operations provides a partial order relation. The other way round is also valid: If we have an appropriate partial order (a comparison), we can construct the corresponding complete lattice.
A direct introduction of the 7 basic comparisons that we deduced from the overlapping relation is given in [4]. The paper summarizes that interval comparisons are implemented by conditions on the endpoints.
Our two reflexive partial order relations induce the following lattices

1. The containment relation $\subseteq$ defines the complete lattice $(\overline{\mathbb{IR}}, \subseteq)$ with least element $\emptyset$ and greatest element $[-\infty\,, +\infty]$ Here the meet of two intervals, their intersection $\cap$ and the join, their interval hull $\cup$ are important and frequently used operations.

[3]Not to disturb with the overlapping state "meets".

	Containment				Set			Certainly			Possibly		
	dis	int	prop	sub	$<$	$\leq$	$=$	$<$	$\leq$	$=$	$<$	$\leq$	$=$
fstEmpty	t	t	t	t									
beforeP	t				t	t		t	t		t	t	
meets					t	t			t		t	t	t
overlaps					t	t					t	t	t
starts			t	t		t			$s(A)$		t	t	t
containedByP		t	t	t							t	t	t
finishes			t	t							t	t	t
equalP				t		s	t		s	s	ns	t	t
bothEmpty	t						t			t			
finishedBy						t			$s(B)$		t	t	t
containsP											t	t	t
equalP				t		s	t		s	s	ns	t	t
bothEmpty	t					t	t		t	t			
finishedBy						t			$s(B)$		t	t	t
containsP											t	t	t
startedBy											t	t	t
overlappedBy											t	t	t
metBy												t	t
afterP	t												
sndEmpty	t												

Table 6: Fortran comparisons computed with overlapping relation

2. The "lessThanOrEqual" operation $\leq$ does only form a conditionally complete lattice on $\overline{\mathbb{IR}} \setminus \emptyset$, because the meet $[-\infty, -\infty]$ of sequences like $(x_n) = ([-\infty, -n])$ is in $(\mathbb{R}^* \times \mathbb{R}^*, \leq)$, but not in $\overline{\mathbb{IR}}$. Similar with the join of growing sequences.

3. On level 2, however, since all sequences are finite, $\overline{\mathbb{IF}} \setminus \emptyset$ is a complete lattice with smallest element $[-\infty, minreal]$, and greatest element $[maxreal, \infty]$.

   The meet or join of 2 intervals are $glb(A, B) = [min(\underline{a}, \underline{b}), min(\overline{a}, \overline{b})]$ or $lub(A, B) = [max(\underline{a}, \underline{b}), max(\overline{a}, \overline{b})]$, respectively.

4. The third relation, the "meets" or "precedes" comparison is not a partial order, therefore we do not construct a corresponding complete lattice.

Note that the switch to level 2 is necessary, because in our definition $\overline{\mathbb{IR}}$ does not contain unbounded singleton intervals.
The lattice operations can be determined with the information obtained from the interval overlapping relation in the same way as the comparisons, see Table 7. In the states concerning the empty sets, we apply the rule that *glb* as well as *lub* are empty, if either of the operands is empty.

Proposition 5 *Table 7 is correct. Note that the intersection is empty, if one of the operands is empty, whereas the intervall hull is the other operand.*

$A \circledcirc B$	$A \cap B$	$A \underline{\cup} B$	$glb(A,B)$	$lub(A,B)$
	$[max(\underline{a},\underline{b}),$ $min(\overline{a},\overline{b})]$	$[min(\underline{a},\underline{b}),$ $max(\overline{a},\overline{b})]$	$[min(\underline{a},\underline{b}),$ $min(\overline{a},\overline{b})]$	$[max(\underline{a},\underline{b}),$ $max(\overline{a},\overline{b})]$
fstEmpty	$\emptyset$	B	$\emptyset$	$\emptyset$
beforeP	$\emptyset$	$[\underline{a},\overline{b}]$	A	B
meets	$[\overline{a},\overline{a}]$	$[\underline{a},\overline{b}]$	A	B
overlaps	$[\underline{b},\overline{a}]$	$[\underline{a},\overline{b}]$	A	B
starts	A	B	A	B
containedByP	A	B	$[\underline{b},\overline{a}]$	$[\underline{a},\overline{b}]$
finishes	A	B	B	A
equalP	A	A	A	A
bothEmpty	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
finishedBy	B	A	A	B
containsP	B	A	$[\underline{a},\overline{b}]$	$[\underline{b},\overline{a}]$
startedBy	B	A	B	A
overlappedBy	$[\underline{a},\overline{b}]$	$[\underline{b},\overline{a}]$	B	A
metBy	$[\overline{b},\overline{b}]$	$[\underline{b},\overline{a}]$	B	A
afterP	$\emptyset$	$[\underline{b},\overline{a}]$	B	A
sndEmpty	$\emptyset$	A	$\emptyset$	$\emptyset$

Table 7: lattice operations computed with overlapping relation

5 The Object-Oriented API

In section 3 we introduced 3 alternatives for the definition of interval comparisons. The first alternative always computes the interval overlapping relation for the two operands and then looks-up the result of the comparison in Table 4 or Table 7 for lattice operations. It has the advantage that the precise information about the relative positions of the two interval operands may be exploited to speed up evaluations of related comparisons.
A disadvantage may be the new kind of interface defining an object instead of a set of boolean functions or operators. Furthermore one may argue that specific comparisons like $A \subseteq B$ or $A = B$ can already be checked by 2 (parallel) floating-point comparisons whereas the overlapping relation needs up to 8 floating-point comparisons. That has to be checked for the used algorithms, the specific hardware design, or the applicaton itself.
The class IOV manages the result of the overlapping test between 2 intervals like an abstract data type. Its range is a pair of intervals (the operands) together with the state information. Its operations are given in Table 8.
The state and the operands are initialized by the constructor and then never changed anymore. For each interval comparison and lattice operation there is a method without parameters, the operands are taken from the calling object. The most frequently occuring test for disjoint intervals is supplied as a utility function. The operands and the current state can be accessed.
In such a setting the predefined comparisons can be evaluated, nothing more. When we, however, open the access to the overlapping states, we can compose new efficient comparisons. Hence, additionally to the abstract data type IOV the atomic operations shall be available as comparisons, see Table 9. For convenience all 13 states should be provided. According to Remark 4 we provide

Result	Operation	Parameters	Explanation
IOV	construct	(interval A, interval B)	computes state $A \circledcirc B$
bool	disjoint	()	test for empty intersection
bool	equals	()	$A = B$
bool	subset	()	$A \subset B$ interior
bool	precedes	()	$A \prec B$, before
bool	less	()	$A < B$
bool	subseteq	()	$A \subseteq B$
bool	preceq	()	$A \preceq B$
bool	leq	()	$A \leq B$
interval	intersect	()	intersection
interval	hull	()	interval hull
interval	glb	()	greatest lower bound
interval	lub	()	lest upper bound
float	first	()	first operand A
float	second	()	second operand B
Q	state	()	set of states Q, see Def 1

Table 8: The abstract data type IOV

13 states for the end user, but internally the 3 empty-states should be made available.
The function `disjoint` can be implemented in 2 ways:

```
bool disjoint() { //  13 states
    return (state() == before) || (state() == after)
}

bool disjoint() { //  16 states
    switch (state()){
      case beforeP : return true
      case afterP  : return true
      case fstEmpty: return true
      case sndEmpty: return true
      case bothEmpty: return true
      default: return false
    }
}
```

When all 16 states are accessable, we use a case statement to implement the table look-up. More efficient implementations can be provided when the representation is fixed.
As an example for the use of the interval overlapping relation we discuss the extended interval Newton method.
The classical algorithm 1 uses 6 interval comparisons and 2 intersections.
If we transform it into a relational algorithm 2, the information with 2 calls of the interval overlapping relation is sufficient to control the flow of the method. Note that the resulting state of overlapping is stored and re-used by methods or properties.

Result	Operation	Parameters	Explanation
bool	before	(interval A, interval B)	atomic op for state “before”
bool	meets	(interval A, interval B)	atomic op for state “meets”
bool	overlaps	(interval A, interval B)	atomic op for state “overlaps”
bool	starts	(interval A, interval B)	atomic op for state “starts”
bool	containedBy	(interval A, interval B)	atomic op for state “containedBy”
bool	finishes	(interval A, interval B)	atomic op for state “finishes”
bool	equalP	(interval A, interval B)	atomic op for state “equalP”
bool	bothEmpty	(interval A, interval B)	atomic op for state “bothEmpty”
bool	finishedBy	(interval A, interval B)	atomic op for state “finishedBy”
bool	contains	(interval A, interval B)	atomic op for state “contains”
bool	startedBy	(interval A, interval B)	atomic op for state “startedBy”
bool	overlappedBy	(interval A, interval B)	atomic op for state “overlappedBy”
bool	metBy	(interval A, interval B)	atomic op for state “metBy”
bool	after	(interval A, interval B)	atomic op for state “after”

Table 9: Atomic comparisons

6 Conclusion

In this position paper we have outlined how the different sets of interval comparisons, like the one proposed in motion 13 [4] or the Fortran95 set can be defined and evaluated with the help of the interval overlapping relation. Newly combined interval comparisons can easily be defined, if the states of the interval overlapping relation are known.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26/11/1983:832–843, 1983.

[2] N. Hayes. IEEE Interval Standard Working Group - P1788 e-mail of April 11, 2010. `http://grouper.ieee.org/groups/1788/`.

[3] N. Hayes. IEEE Interval Standard Working Group - P1788 Motion 8 Exception Handling, Sept 2009. `http://grouper.ieee.org/groups/1788/`.

[4] U. Kulisch. IEEE Interval Standard Working Group - P1788 Motion 13 Version 4. `http://grouper.ieee.org/groups/1788/`.

[5] D. Lohez. Comparison Relations for Intervals : A Framework. `http://grouper.ieee.org/groups/1788/`.

[6] M. Nehmeier, S. Siegel, and J. Wolff von Gudenberg. Parallel Detection of Interval Overlapping. In *Proceedings of Para2010, to appear*, Apr. 2010. submitted to para2010 `http://www.yourhost.is/para2010/home.html`.

Algorithm 1: INewton (classical)

Input:
f : function
Y : interval
ϵ : epsilon
$yUnique$: flag
$Zero$: list of enclosing intervals
$Info$: flag vector
N : number
Output: $[Zero, Info, N]$
begin
  if $0 \notin f(Y)$ **then**
    return *(Zero,Info,N)*
  $c \leftarrow \mu(Y)$;
  `/* extended division */`
  $[Z_1, Z_2] \leftarrow f(c)/f'(Y)$;
  $[Z_1, Z_2] \leftarrow c - [Z_1, Z_2]$;
  $V_1 \leftarrow Y \cap Z_1$;
  $V_2 \leftarrow Y \cap Z_2$;
  if $V_1 = Y$ **then**
    $V_1 \leftarrow [\underline{y}\,, c]$;
    $V_2 \leftarrow [c\,, \overline{y}]$;
  if $V_1 \neq \emptyset$ **and** $V_2 = \emptyset$ **then**
    $yUnique \leftarrow yUnique$ **or** $V_1 \subset Y$;
  foreach $i = 1, 2$ **do**
    if $V_i = \emptyset$ **then**
      continue;
    if $drel(V_i) < \epsilon$ **then**
      $N = N + 1$;
      $Zero[N] = V_i$;
      $Info[N] = yUnique$;
    else
      $INewton(f, V_i, \epsilon,$
      $yUnique, Zero, Info, N)$;
  end
  return *(Zero,Info,N);*
end

Algorithm 2: INewtonRel (relational)

Input:
f : function
Y : interval
ϵ : epsilon
$yUnique$: flag
$Zero$: list of enclosing intervals
$Info$: flag vector
N : number
Output: $[Zero, Info, N]$
begin
  if $0 \notin f(Y)$ **then**
    return *(Zero,Info,N)*
  $c \leftarrow \mu(Y)$;
  `/* extended division */`
  $[Z_1, Z_2] \leftarrow f(c)/f'(Y)$;
  $[Z_1, Z_2] \leftarrow c - [Z_1, Z_2]$;
  $R_1 \leftarrow Y \circledcirc Z_1$;
  $R_2 \leftarrow Y \circledcirc Z_2$;
  if $R_1.subseteq()$ **then**
    $V_1 \leftarrow [\underline{y}\,, c]$;
    $V_2 \leftarrow [c\,, \overline{y}]$;
    $bisected \leftarrow true$;
  else if not $R_1.disjoint()$
  and $R_2.disjoint()$ **then**
    $yUnique \leftarrow yUnique$ **or**
    $R_1.state() ==$
    containedBy;
  foreach $i = 1, 2$ **do**
    if not *bisected* **then**
      if $R_i.disjoint()$ **then**
        continue;
      $R_i \leftarrow Y \circledcirc Z_i$;
      $V_i \leftarrow R_i.intersect()$;
    if $drel(V_i) < \epsilon$ **then**
      $N = N + 1$;
      $Zero[N] = V_i$;
      $Info[N] = yUnique$;
    else
      $INewtonRel(f, V_i, \epsilon,$
      $yUnique, Zero, Info, N)$;
  end
  return *(Zero,Info,N);*
end