Joachim Faulhaber

An Introduction to the Interval Template Library

Lecture
held at the Boost Library Conference 2009

2009-05-08

Updated version 3.1.0   2009-09-17

Lecture Outline

- Background and Motivation
- Design
- Examples
- Semantics
- Implementation
- Future Works
- Availability

Background and Motivation

- Interval containers simplified the implementation of date and time related tasks
  - Decomposing *"histories"* of attributed events into segments with constant attributes.
  - Working with time grids, e.g. a grid of months.
  - Aggregations of values associated to date or time intervals.

- … that occurred frequently in programs like
  - Billing modules
  - Therapy scheduling programs
  - Hospital and controlling statistics

Design

- Background is the date time problem domain ...
- … but the scope of the **Itl** as a generic library is more general:

an **interval_set** *is a* **set**

that is implemented as a set of intervals

an **interval_map** *is a* **map**

that is implemented as a map of interval value pairs

Aspects

- There are two aspects in the design of interval containers
- Conceptual aspect

```
interval_set<int> mySet;
mySet.insert(42);
bool has_answer = mySet.contains(42);
```

  - On the conceptual aspect an interval_set can be used just as a set of elements
  - except for . . .
  - . . . ***iteration*** *over* *elements*
  - consider **`interval_set<double>`** or **`interval_set<string>`**
- Iterative Aspect
  - **Iteration** is always done over *intervals*

Design

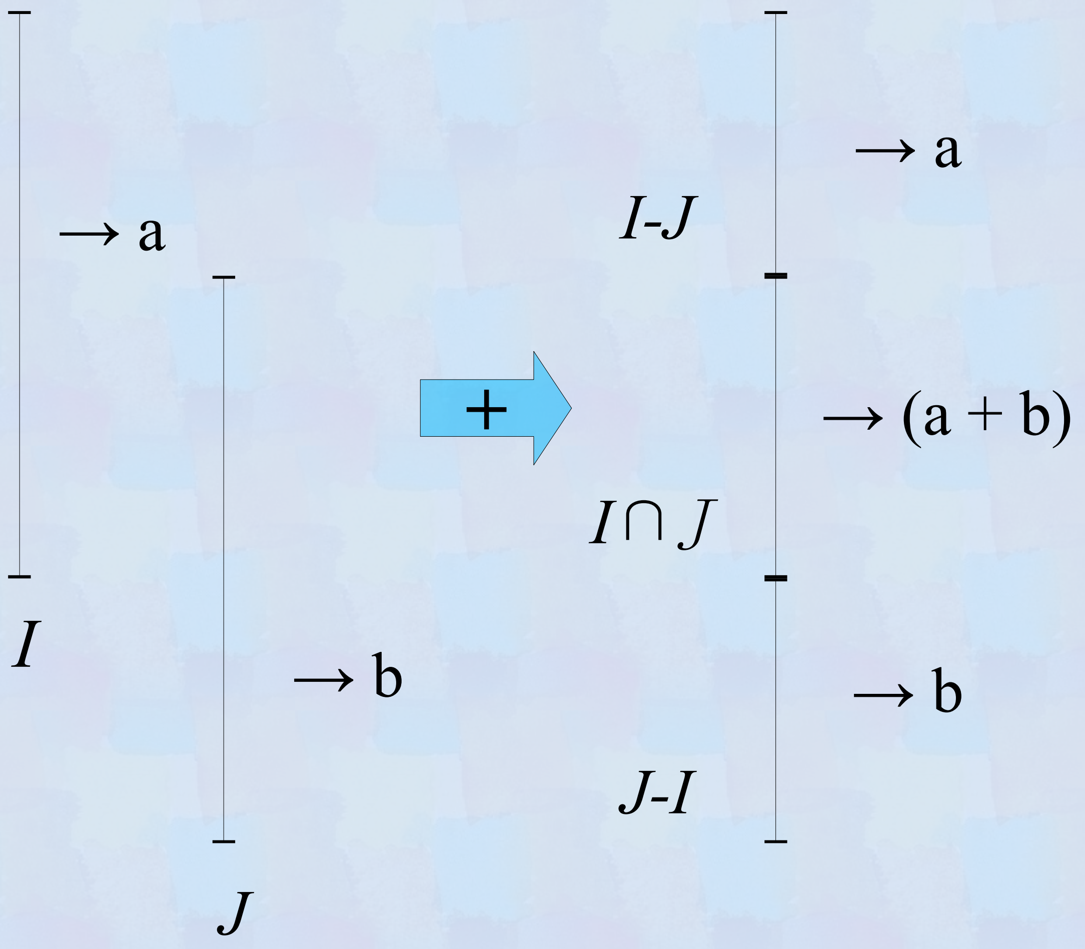
- Addability and Subtractability
  - All of itl's (interval) containers are *Addable* and *Subtractable*
  - They implement `operators +=`, `+`, `-=` and `-`

	`+=`	`-=`
sets	set union	set difference
maps	?	?

- A possible implementation for maps
  - Propagate addition/subtraction to the associated values
  - . . . or aggregate on overlap
  - . . . or aggregate on collision

Design

- Aggregate on overlap



- Decompositional effect on Intervals
- Accumulative effect on associated values

I, J: intervals, a,b: associated values

- Aggregate on overlap, a minimal example

```
typedef itl::set<strin >  !ests;
interval_map<time,  !ests> party;

party += ma"e_pair(
  interval<time>::rightopen(2#$##, 22$##),  !ests(%&ary%));

party += ma"e_pair(
  interval<time>::rightopen(2'$##, 2($##),  !ests(%)arry%));


** party now contains
+2#$##, 2'$##),>-%&ary%.
+2'$##, 22$##),>-%)arry%,%&ary%. ** !est sets a  re ated
+22$##, 2($##),>-%)arry%.
```

Design

- The Itl's class templates

Granu-larity	Style	Sets	Maps
interval		`interval`	
	joining	`interval_set`	`interval_map`
	separating	`separate_interval_set`	
	splitting	`split_interval_set`	`split_interval_map`
element		`set`	`map`

- Interval Combining Styles: ***Joining***
  - Intervals are joined on overlap or on touch
  - . . . *for maps*, if associated values are equal
  - Keeps interval_maps and sets in a minimal form

```
interval_set

  {[1      3)          }
+      [2      4)
+                 [4 5)

= {[1          4)      }

= {[1              5)}
```

```
interval_map

  {[1      3)->1          }
+      [2      4)->1
+                 [4 5)->1

={[1 2)[2 3)[3 4)         }
   ->1  ->2  ->1
={[1 2)[2 3)[3      5)    }
   ->1  ->2      ->1
```

- Interval Combining Styles: ***Splitting***
  - Intervals are split on overlap and kept separate on touch
  - All interval borders are preserved (insertion memory)

```
split_interval_set

  {[1      3)          }
+      [2      4)
+                [4 5)

= {[1 2)[2 3)[3 4)     }

= {[1 2)[2 3)[3 4)[4 5)}
```

```
split_interval_map

  {[1      3)->1           }
+      [2      4)->1
+                [4 5)->1

={[1 2)[2 3)[3 4)          }
    ->1  ->2  ->1
={[1 2)[2 3)[3 4)[4 5)     }
    ->1  ->2  ->1  ->1
```

- Interval Combining Styles: ***Separating***
  - Intervals are joined on overlap but kept separate on touch
  - Preserves borders that are never crossed (preserves a hidden grid).

```
separate_interval_set

  {[1      3)         }
+       [2      4)
+                 [4 5)

= {[1           4)    }

= {[1           4)[4 5)}
```

Examples

- A few instances of intervals (interval.cpp)

```
inter/al<int> int_inter/al = inter/al<int>$$closed((,0);

inter/al<do!ble> s1rt_inter/al
  = inter/al<do!ble>$$rightopen('*s1rt(2.#), s1rt(2.#));

inter/al<std$$strin > city_inter/al
  = inter/al<std$$strin >$$leftopen(%2arcelona%, %2oston%);

inter/al<boost$$ptime> time_inter/al
  = inter/al<boost$$ptime>$$open(
                          time_from_strin (%2##3,#4,2# '5$(#%),
                          time_from_strin (%2##3,#4,2# 2($##%)
    );
```

Examples

- A way to iterate over months and weeks (month_and_wee"_ rid.cpp)

```
6incl!de <boost*itl* re orian.hpp> **boost$$ re orian pl!s adapter code
6incl!de <boost*itl*split_inter/al_set.hpp>

** 7 split_inter/al_set of  re orian dates as date_ rid.
typedef split_inter/al_set<boost$$ re orian$$date> date_ rid;

** 8omp!te a date_ rid of months !sin  boost$$ re orian.
date_ rid month_ rid(const inter/al<date>9 scope)
-
    date_ rid month_ rid;
    ** 8omp!te a date_ rid of months !sin  boost$$ re orian.
    . . .
    ret!rn month_ rid;
.

** 8omp!te a date_ rid of wee"s !sin  boost$$ re orian.
date_ rid wee"_ rid(const inter/al<date>9 scope)
-
    date_ rid wee"_ rid;
    ** 8omp!te a date_ rid of wee"s !sin  boost$$ re orian.
    . . .
    ret!rn wee"_ rid;
.
```

Examples

- A way to iterate over months and weeks

```
/oid month_and_time_ rid()
-
    date someday = day_cloc"$$local_day();
    date thenday = someday : months(2);
    inter/al<date> scope = inter/al<date>$$ri htopen(someday, thenday);

    ** 7n intersection of the month and wee"  rids ...
    date_ rid month_and_wee"_ rid
        = month_ rid(scope) 9 wee"_ rid(scope);

    ** ... allows to iterate months and wee"s. ;hene/er a month
    ** or a wee" chan es there is a new inter/al.
    for(date_ rid$$iterator it = month_and_wee"_ rid.be in();
        it <= month_and_wee"_ rid.end(); it::)
    -   . . .   .

    ** ;e can also intersect the  rid into an inter/al_map to ma"e
    ** sh!re that all inter/als are within months and wee" bo!nds.
    inter/al_map<boost$$ re orian$$date, some_type> accr!al;
    comp!te_some_res!lt(accr!al, scope);
    accr!al 9= month_and_wee"_ rid;

.
```

Examples

- Aggregating with interval_maps
  - Computing averages via implementing **operator +=** (par tys_ !est_a/era e.cpp)

```
class counted_sum
-
p!blic$
    counted_sum()$_s!m(#),_co!nt(#)-.
    co!nted_s!m(int s!m)$_s!m(s!m),_co!nt(')-.

    int s!m()const  -ret!rn _s!m;.
    int co!nt()const-ret!rn _co!nt;.
    do!ble a/era e()const
     - ret!rn _co!nt==# = #.# $ _s!m*static_cast<do!ble>(_co!nt); .

    co!nted_s!m9 operator += (const co!nted_s!m9 ri ht)
    - _s!m := ri ht.s!m(); _co!nt := ri ht.co!nt(); ret!rn >this; .

pri/ate$
    int _s!m;
    int _co!nt;
.;

bool operator == (const co!nted_s!m9 left, const co!nted_s!m9 ri ht)
- ret!rn left.s!m()==ri ht.s!m() 99 left.co!nt()==ri ht.co!nt(); .
```

Examples

- Aggregating with interval_maps
  - Computing averages via implementing **operator +=**

```
/oid partys_hei ht_a/era e()
-
    inter/al_map<ptime, counted_sum> hei ht_s!ms;

    hei ht_s!ms += (
      ma"e_pair(
        inter/al<ptime>$$ri htopen(
          time_from_strin (%2##3,#4,2# '5$(#%),
          time_from_strin (%2##3,#4,2# 2($##%)),
          counted_sum(165)) ** &ary is ',?4 m tall.
    );

    ** 7dd hei ht of more pary  !ests . . .

    inter/al_map<ptime, co!nted_s!m>$$iterator hei ht_s!m_ =
        hei ht_s!ms.be in();
    while(hei ht_s!m_ <= hei ht_s!ms.end())
    -
        inter/al<ptime> when = hei ht_s!m_,>first;
        do!ble hei ht_a/era e = (>hei ht_s!m_::).second.average();

        co!t << %+% << when.first() << % , % << when.!pper() << %)%
             << %$ % << hei ht_a/era e << % cm% << endl;
    .
.
```

Examples

- Interval containers allow to express a variety of date and time operations in an easy way.
  - Example `man_power.cpp ...`
  - Subtract weekends and holidays from an interval_set
    ```
    worktime -= weekends(scope)
    worktime -= german_reunification_day
    ```
  - Intersect an interval_map with an interval_set
    ```
    claudias_working_hours &= worktime
    ```
  - Subtract and interval_set from an interval map
    ```
    claudias_working_hours -= claudias_absense_times
    ```
  - Adding interval_maps
    ```
    interval_map<date,int> manpower;
    manpower += claudias_working_hours;
    manpower += bodos_working_hours;
    ```

Examples

- Interval_maps can also be intersected
  Example **`user_groups.cpp`**

```
typedef boost$$itl$$set<strin > &emberSet@;
typedef inter/al_map<date, &emberSet@> &embership@;

/oid !ser_ ro!ps()
-
    . . .

    &embership@ med_!sers;
    ** 8omp!te membership of medical staff
    med_!sers := ma"e_pair(member_inter/al_', &emberSet@(%Ar.Be"yll%));
    med_!sers := . . .

    &embership@ admin_!sers;
    ** 8omp!te membership of administation staff
    med_!sers := ma"e_pair(member_inter/al_2, &emberSet@(%&r.)yde%));
    . . .

    &embership@ all_!sers   = med_!sers + admin_!sers;

    &embership@ s!per_!sers = med_!sers & admin_!sers;
    . . .

.
```

Semantics

- The semantics of ***itl sets*** is based on a concept **`itl::Set`**
  - **`itl::set`**, **`interval_set`**, **`split_interval_set`** and **`separate_interval_set`** are models of concept **`itl::Set`**

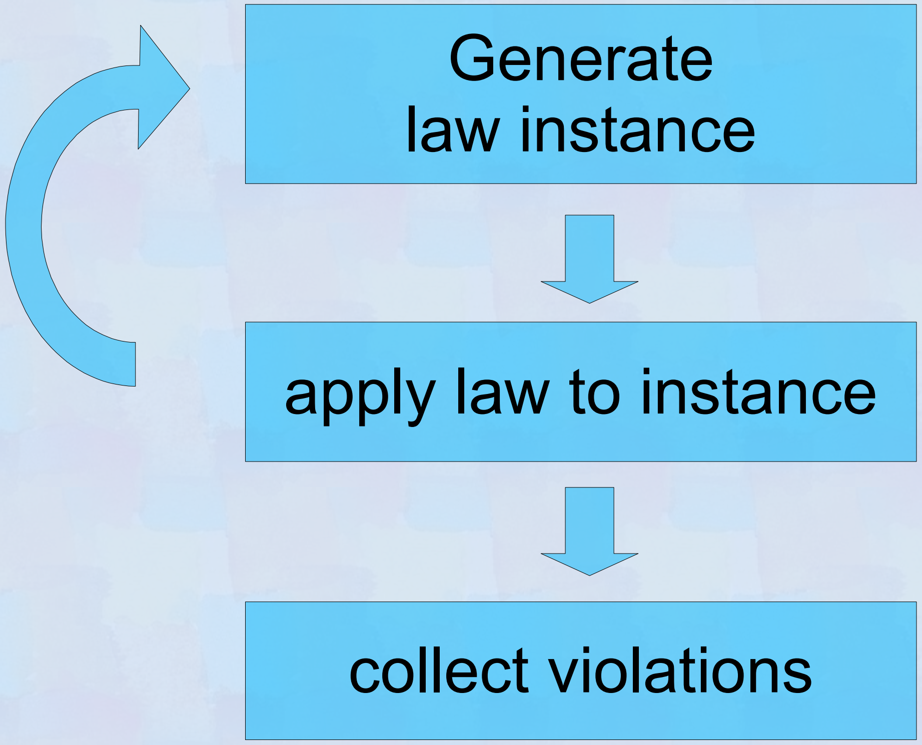
```
** 7bstract part
empty set$        Set$$Set()
s!bset relation$  bool Set$$contained_in(const Set9 s2)const
e1!ality$         bool is_element_e1!al(const Set9 s', const Set9 s2)
set !nion$        Set9 operator := (Set9 s', const Set9 s2)
                  Set  operator :  (const Set9 s', const Set9 s2)
set difference$   Set9 operator ,= (Set9 s', const Set9 s2)
                  Set  operator ,  (const Set9 s', const Set9 s2)
set intersection$ Set9 operator 9= (Set9 s', const Set9 s2)
                  Set  operator 9  (const Set9 s', const Set9 s2)

** Cart related to se1!ential orderin
sortin  order$    bool operator <  (const Set9 s', const Set9 s2)
leDico raphical e1!ality$
                  bool operator == (const Set9 s', const Set9 s2)
```

Semantics

- The semantics of ***itl maps*** is0elãc5©Ww

Semantics

- Defining semantics of itl concepts via sets of laws
  - aka c++0x axioms
- Checking law sets via automatic testing:
  - A ***La***w ***Ba***sed ***Te***st ***A***utomaton ***LaBatea***



Commutativity<T a, U b, +>:
a + b = b + a;

Semantics

- Lexicographical Ordering and Equality
  - For all itl containers `operator <` implements a ***strict weak ordering***.
  - The ***induced equivalence*** of this ordering is ***lexicographical equality*** which is implemented as `operator ==`
  - This is in line with the semantics of SortedAssociativeContainers

Semantics

- Subset Ordering and Element Equality
  - For all itl containers function `contained_in` implements a ***partial ordering***.
  - The ***induced equivalence*** of this ordering is ***equality of elements*** which is implemented as function `is_element_equal`.

Semantics

- itl::Sets
- All itl sets implement a ***Set Algebra***, which is to say satisfy a "*classical*" set of laws . . .
  - . . . using `is_element_equal` as equality
  - Associativity, Neutrality, Commutativity (for `+` and `&`)
  - Distributivity, DeMorgan, Symmetric Difference

- Most of the itl sets satisfy the classical set of laws even if . . .
  - . . . lexicographical equality: `operator ==` is used
  - The differences reflect proper inequalities in sequence that occur for `separate_interval_set` and `split_interval_set`.

Semantics

- Concept Induction / Concept Transition
  - The semantics of itl::Maps appears to be *determined* by the *codomain type* of the map

	is model of	if	example
`Map<D,Monoid>`	`Monoid`		`interval_map<int,string>`
`Map<D,Set>`	`Set`	*C1*	`interval_map<int,set<int>>`
`Map<D,CommutMonoid>`	`CommutMonoid`		`interval_map<int,unsigned>`
`Map<D,AbelianGroup>`	`AbelianGroup`	*C2*	`interval_map<int,int,total>`

  - Conditions *C1* and *C2* restrict the *Concept Induction* to specific *map traits*
    - *C1*: Value pairs that carry a *neutral element* as associated value are always deleted (Trait: ***absorbs_neutrons***).
    - *C2*: The map *is total*: Non existing keys are implicitly mapped to *neutral elements* (Trait: ***is_total***).

Implementation

- Itl containers are implemented based on std::set and std::map
  - Basic operations like *adding* and *subtracting* intervals or interval value pairs perform with a time *complexity between** *amortized O(log n)* and *O(n)*, where *n* is the number of intervals of a container.
  - Operations like *addition* and *subtraction* of whole containers are having a worst case complexity of *O(m log(n+m))*, where *n* and *m* are the numbers of intervals of the containers to combine.

    * : Consult the library documentation for more detailed information.

Future Works

- Implementing interval_maps of sets more efficiently

- Revision of features of the extended itl (itl_plus.zip)
  - Decomposition of histories: k histories h_k with attribute types $A_1, \ldots, A_k$ are "*decomposed*" to a product history of tuples of attribute sets:
    $(h_1<T,A_1>,\ldots, h<T,A_k>) \rightarrow h<T, (set<A_1>,\ldots, set<A_k>)>$

  - Cubes (generalized crosstables): Applying *aggregate on collision* to maps of tuple value pairs in order to organize hierachical data and their aggregates.

Availability

- Itl project on sourceforge (version 2.0.1)

httphttp://sourceforge.net/projects/itl ys

- Latest version on boost vault/Containers (3.1.0)

http://www.boostpro.com/vault/ → containers

- itl_3_1_0.zip → [illegible]

itlbau3_11_0.zip → [illegible]

Availability

- Boost sandbox
  https://svn.boost.org/svn/boost/sandbox/itl/
  - Core itl: Interval containers in prepartion for boost
    https://svn.boost.org/svn/boost/sandbox/itl/boost/itl/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/itl/
  - Extended itl_xt: “histories” and cubes
    https://svn.boost.org/svn/boost/sandbox/itl/boost/itl_xt/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/itl_xt/
  - Validater LaBatea:
    Compiles with msvc-8.0 or newer, gcc-4.3.2 or newer
    https://svn.boost.org/svn/boost/sandbox/itl/boost/validate/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/validate/

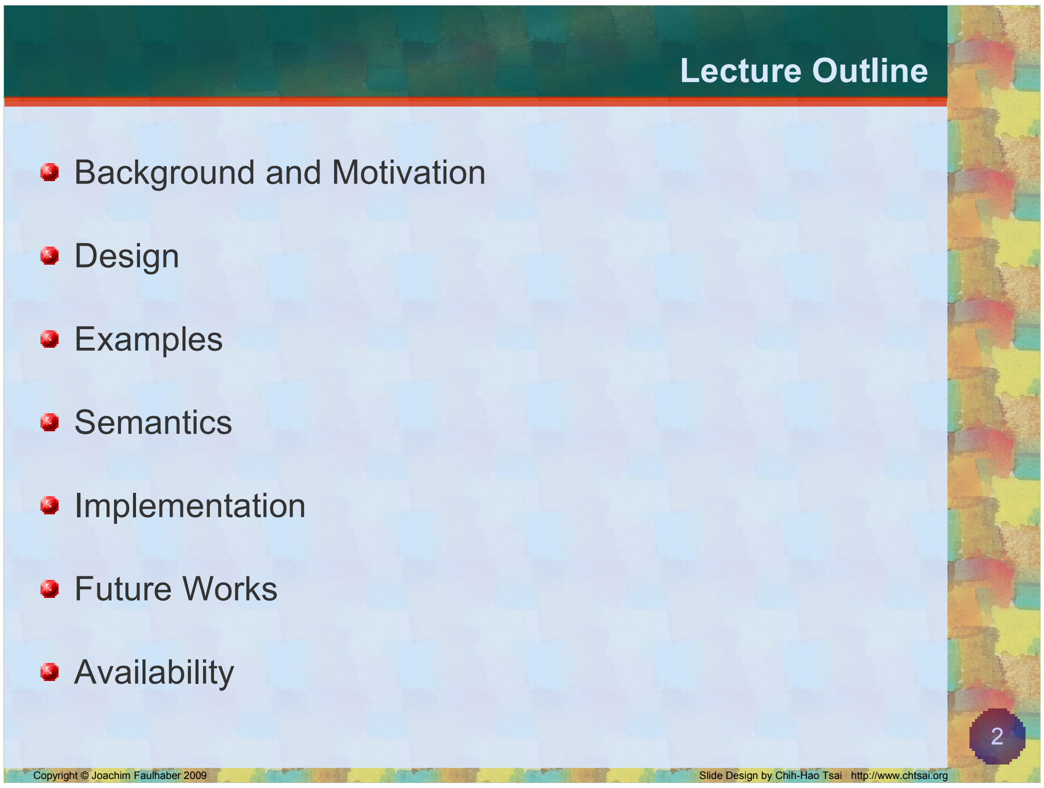
Joachim Faulhaber

An Introduction to the Interval Template Library

Lecture
held at the Boost Library Conference 2009

2009-05-08

Updated version 3.1.0   2009-09-17



Lecture Outline

- Background and Motivation
- Design
- Examples
- Semantics
- Implementation
- Future Works
- Availability

Background and Motivation

- Interval containers simplified the implementation of date and time related tasks
  - Decomposing *“histories”* of attributed events into segments with constant attributes.
  - Working with time grids, e.g. a grid of months.
  - Aggregations of values associated to date or time intervals.

- … that occurred frequently in programs like
  - Billing modules
  - Therapy scheduling programs
  - Hospital and controlling statistics

Design

- Background is the date time problem domain ...
- … but the scope of the **Itl** as a generic library is more general:

an **interval_set** *is a* **set**

that is implemented as a set of intervals

an **interval_map** *is a* **map**

that is implemented as a map of interval value pairs

Aspects

- There are two aspects in the design of interval containers
- Conceptual aspect

```
interval_set<int> mySet;
mySet.insert(42);
bool has_answer = mySet.contains(42);
```

  - On the conceptual aspect an interval_set can be used just as a set of elements
  - except for . . .
  - . . . ***iteration*** *over elements*
  - consider `interval_set<double>` or `interval_set<string>`
- Iterative Aspect
  - **Iteration** is always done over *intervals*

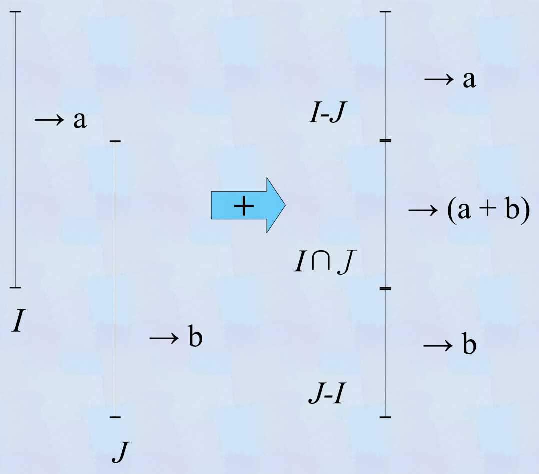
Design

- Addability and Subtractability
  - All of itl's (interval) containers are *Addable* and *Subtractable*
  - They implement `operators +=`, `+`, `-=` and `-`

	`+=`	`-=`
sets	set union	set difference
maps	?	?

- A possible implementation for maps
  - Propagate addition/subtraction to the associated values
  - . . . or aggregate on overlap
  - . . . or aggregate on collision

- Aggregate on overlap



- Decompositional effect on Intervals
- Accumulative effect on associated values

I, J: intervals, a,b: associated values

Design

- Aggregate on overlap, a minimal example

```
typedef itl::set<strin >  !ests;
interval_map<time,  !ests> party;

party += ma"e_pair(
  interval<time>::rightopen(2#$##, 22$##),  !ests(%&ary%));

party += ma"e_pair(
  interval<time>::rightopen(2'$##, 2($##),  !ests(%)arry%));


** party now contains
+2#$##, 2'$##),>-%&ary%.
+2'$##, 22$##),>-%)arry%,%&ary%. ** !est sets a  re ated
+22$##, 2($##),>-%)arry%.
```

Design

- The Itl's class templates

Granu-larity	**Style**	**Sets**	**Maps**
interval		`interval`	
	joining	`interval_set`	`interval_map`
	separating	`separate_interval_set`	
	splitting	`split_interval_set`	`split_interval_map`
element		`set`	`map`

Design

- Interval Combining Styles: ***Joining***
  - Intervals are joined on overlap or on touch
  - . . . *for maps*, if associated values are equal
  - Keeps interval_maps and sets in a minimal form

```
interval_set

  {[1      3)           }
+       [2      4)
+                 [4 5)

= {[1           4)      }

= {[1                5)}
```

```
interval_map

  {[1      3)->1         }
+      [2      4)->1
+                [4 5)->1

={[1 2)[2 3)[3 4)        }
   ->1  ->2  ->1
={[1 2)[2 3)[3     5)   }
   ->1  ->2     ->1
```

Design

- Interval Combining Styles: ***Splitting***
  - Intervals are split on overlap and kept separate on touch
  - All interval borders are preserved (insertion memory)

```
split_interval_set

  {[1      3)          }
+       [2     4)
+                 [4 5)

= {[1 2)[2 3)[3 4)     }

= {[1 2)[2 3)[3 4)[4 5)}
```

```
split_interval_map

  {[1      3)->1          }
+      [2      4)->1
+                [4 5)->1

={[1 2)[2 3)[3 4)         }
   ->1  ->2  ->1
={[1 2)[2 3)[3 4)[4 5)    }
   ->1  ->2  ->1  ->1
```

Design

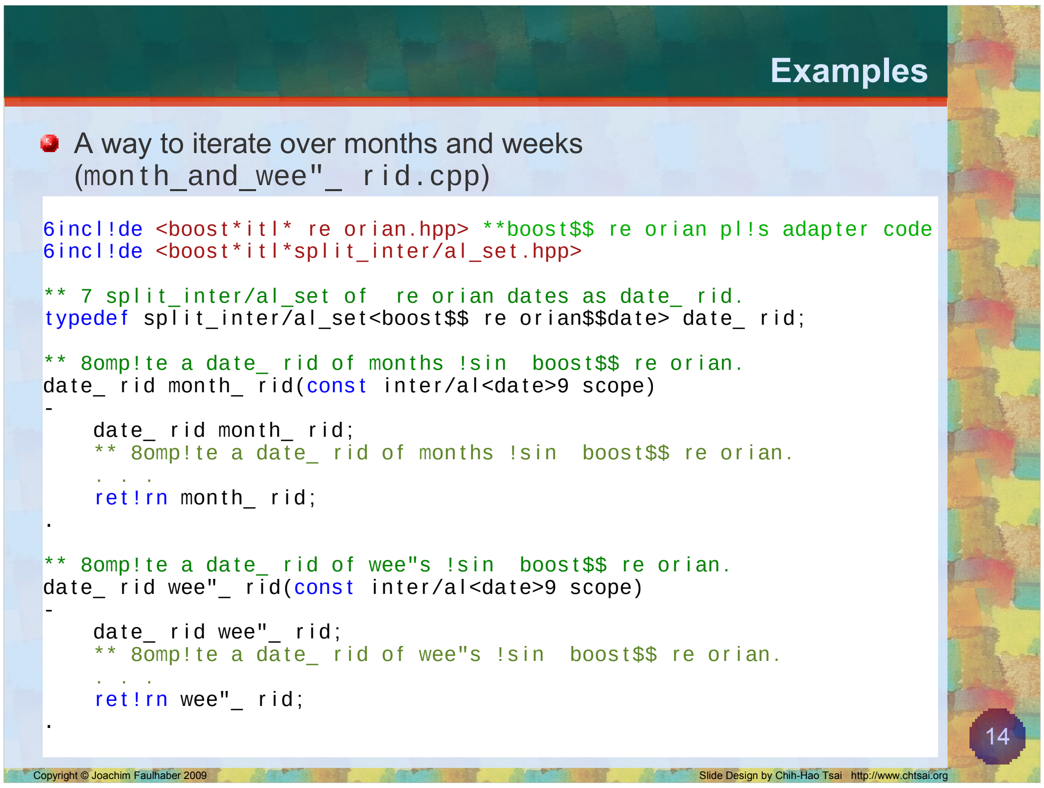
- Interval Combining Styles: ***Separating***
  - Intervals are joined on overlap but kept separate on touch
  - Preserves borders that are never crossed (preserves a hidden grid).

```
separate_interval_set

  {[1      3)          }
+       [2      4)
+                 [4 5)

= {[1           4)     }

= {[1           4)[4 5)}
```

Examples

- A few instances of intervals (interval.cpp)

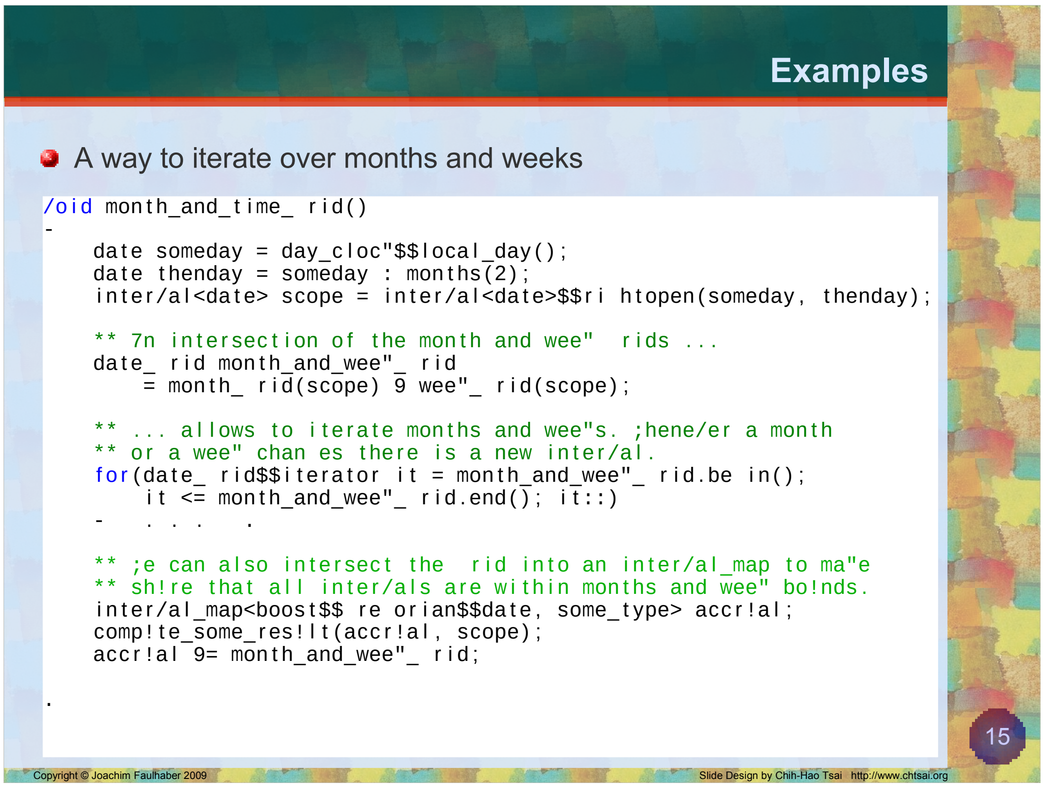
```
inter/al<int> int_inter/al = inter/al<int>$$closed((,0);

inter/al<do!ble> s1rt_inter/al
  = inter/al<do!ble>$$rightopen('*s1rt(2.#), s1rt(2.#));

inter/al<std$$strin > city_inter/al
  = inter/al<std$$strin >$$leftopen(%2arcelona%, %2oston%);

inter/al<boost$$ptime> time_inter/al
  = inter/al<boost$$ptime>$$open(
                         time_from_strin (%2##3,#4,2# '5$(#%),
                         time_from_strin (%2##3,#4,2# 2($##%)
    );
```



Examples

- A way to iterate over months and weeks (month_and_wee"_ rid.cpp)

```
6incl!de <boost*itl* re orian.hpp> **boost$$ re orian pl!s adapter code
6incl!de <boost*itl*split_inter/al_set.hpp>

** 7 split_inter/al_set of  re orian dates as date_ rid.
typedef split_inter/al_set<boost$$ re orian$$date> date_ rid;

** 8omp!te a date_ rid of months !sin  boost$$ re orian.
date_ rid month_ rid(const inter/al<date>9 scope)
-
    date_ rid month_ rid;
    ** 8omp!te a date_ rid of months !sin  boost$$ re orian.
    . . .
    ret!rn month_ rid;
.

** 8omp!te a date_ rid of wee"s !sin  boost$$ re orian.
date_ rid wee"_ rid(const inter/al<date>9 scope)
-
    date_ rid wee"_ rid;
    ** 8omp!te a date_ rid of wee"s !sin  boost$$ re orian.
    . . .
    ret!rn wee"_ rid;
.
```



Examples

- A way to iterate over months and weeks

```
/oid month_and_time_ rid()
-
    date someday = day_cloc"$$local_day();
    date thenday = someday : months(2);
    inter/al<date> scope = inter/al<date>$$ri htopen(someday, thenday);

    ** 7n intersection of the month and wee"  rids ...
    date_ rid month_and_wee"_ rid
        = month_ rid(scope) 9 wee"_ rid(scope);

    ** ... allows to iterate months and wee"s. ;hene/er a month
    ** or a wee" chan es there is a new inter/al.
    for(date_ rid$$iterator it = month_and_wee"_ rid.be in();
        it <= month_and_wee"_ rid.end(); it::)
    -   . . .   .

    ** ;e can also intersect the  rid into an inter/al_map to ma"e
    ** sh!re that all inter/als are within months and wee" bo!nds.
    inter/al_map<boost$$ re orian$$date, some_type> accr!al;
    comp!te_some_res!lt(accr!al, scope);
    accr!al 9= month_and_wee"_ rid;

.
```

Examples

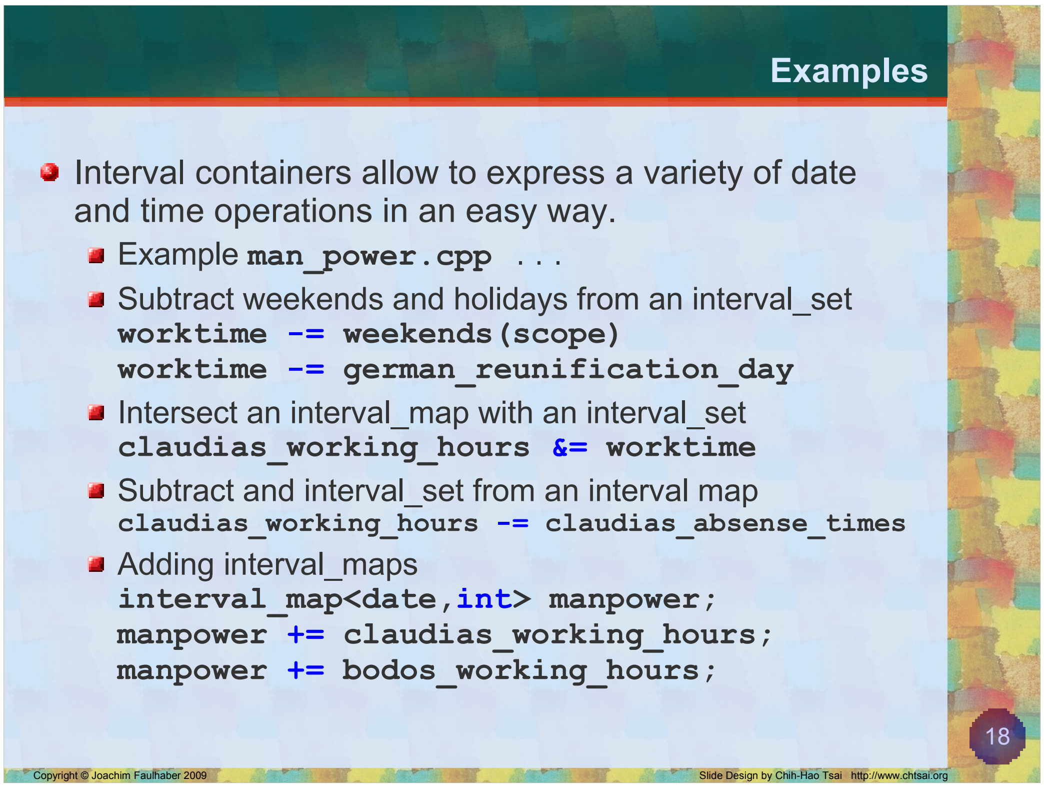
- Aggregating with interval_maps
  - Computing averages via implementing `operator +=` (partys_ !est_a/era e.cpp)

```
class counted_sum
-
p!blic$
    counted_sum()$_s!m(#),_co!nt(#)-.
    co!nted_s!m(int s!m)$_s!m(s!m),_co!nt(')-.

    int s!m()const  -ret!rn _s!m;.
    int co!nt()const-ret!rn _co!nt;.
    do!ble a/era e()const
    - ret!rn _co!nt==# = #.# $ _s!m*static_cast<do!ble>(_co!nt); .

    co!nted_s!m9 operator += (const co!nted_s!m9 ri ht)
    - _s!m := ri ht.s!m(); _co!nt := ri ht.co!nt(); ret!rn >this; .

pri/ate$
    int _s!m;
    int _co!nt;
.;

bool operator == (const co!nted_s!m9 left, const co!nted_s!m9 ri ht)
- ret!rn left.s!m()==ri ht.s!m() 99 left.co!nt()==ri ht.co!nt(); .
```

Examples

- Aggregating with interval_maps
  - Computing averages via implementing `operator +=`

```
/oid partys_hei ht_a/era e()
-
    inter/al_map<ptime, counted_sum> hei ht_s!ms;

    hei ht_s!ms += (
      ma"e_pair(
        inter/al<ptime>$$ri htopen(
          time_from_strin (%2##3,#4,2# '5$(#%),
          time_from_strin (%2##3,#4,2# 2($##%)),
          counted_sum(165)) ** &ary is ',?4 m tall.
    );

    ** 7dd hei ht of more pary  !ests . . .

    inter/al_map<ptime, co!nted_s!m>$$iterator hei ht_s!m_ =
        hei ht_s!ms.be in();
    while(hei ht_s!m_ <= hei ht_s!ms.end())
    -
        inter/al<ptime> when = hei ht_s!m_,>first;
        do!ble hei ht_a/era e = (>hei ht_s!m_::).second.average();

        co!t << %+% << when.first() << % , % << when.!pper() << %)%
             << %$ % << hei ht_a/era e << % cm% << endl;
    .
.
```



Examples

- Interval containers allow to express a variety of date and time operations in an easy way.
  - Example `man_power.cpp` . . .
  - Subtract weekends and holidays from an interval_set
    ```
    worktime -= weekends(scope)
    worktime -= german_reunification_day
    ```
  - Intersect an interval_map with an interval_set
    ```
    claudias_working_hours &= worktime
    ```
  - Subtract and interval_set from an interval map
    ```
    claudias_working_hours -= claudias_absense_times
    ```
  - Adding interval_maps
    ```
    interval_map<date,int> manpower;
    manpower += claudias_working_hours;
    manpower += bodos_working_hours;
    ```

Examples

- Interval_maps can also be intersected
  Example **`user_groups.cpp`**

```
typedef boost$$itl$$set<strin > &emberSet@;
typedef inter/al_map<date, &emberSet@> &embership@;

/oid !ser_ ro!ps()
-
    . . .

    &embership@ med_!sers;
    ** 8omp!te membership of medical staff
    med_!sers := ma"e_pair(member_inter/al_', &emberSet@(%Ar.Be"yll%));
    med_!sers := . . .

    &embership@ admin_!sers;
    ** 8omp!te membership of administation staff
    med_!sers := ma"e_pair(member_inter/al_2, &emberSet@(%&r.)yde%));
    . . .

    &embership@ all_!sers   = med_!sers + admin_!sers;

    &embership@ s!per_!sers = med_!sers & admin_!sers;
    . . .

.
```

Semantics

- The semantics of ***itl sets*** is based on a concept **`itl::Set`**
  - **`itl::set`**, **`interval_set`**, **`split_interval_set`** and **`separate_interval_set`** are models of concept **`itl::Set`**

```
** 7bstract part
empty set$        Set$$Set()
s!bset relation$  bool Set$$contained_in(const Set9 s2)const
e1!ality$         bool is_element_e1!al(const Set9 s', const Set9 s2)
set !nion$        Set9 operator := (Set9 s', const Set9 s2)
                  Set  operator :  (const Set9 s', const Set9 s2)
set difference$   Set9 operator ,= (Set9 s', const Set9 s2)
                  Set  operator ,  (const Set9 s', const Set9 s2)
set intersection$ Set9 operator 9= (Set9 s', const Set9 s2)
                  Set  operator 9  (const Set9 s', const Set9 s2)

** Cart related to se1!ential orderin
sortin  order$    bool operator <  (const Set9 s', const Set9 s2)
leDico raphical e1!ality$
                  bool operator == (const Set9 s', const Set9 s2)
```

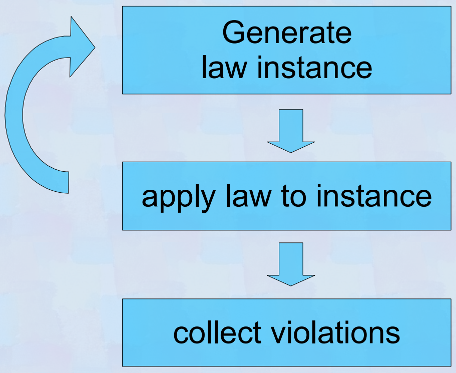
Semantics

- The semantics of ***itl maps*** is based on a concept **`itl::Map`**
  - **`itl::map`**, **`interval_map`** and **`split_interval_map`** are models of concept **`itl::Map`**

```
** 7bstract part
empty map$        &ap$$&ap()
s!bmap relation$  bool &ap$$contained_in(const &ap9 m2)const
e1!ality$         bool is_element_e1!al(const &ap9 m', const &ap9 m2)
map !nion$        &ap9 operator := (&ap9 m', const &ap9 m2)
                  &ap  operator :  (const &ap9 m', const &ap9 m2)
map difference$   &ap9 operator ,= (&ap9 m', const &ap9 m2)
                  &ap  operator ,  (const &ap9 m', const &ap9 m2)
map intersection$ &ap9 operator 9= (&ap9 m', const &ap9 m2)
                  &ap  operator 9  (const &ap9 m', const &ap9 m2)

** Cart related to se1!ential orderin
sortin  order$    bool operator <  (const &ap9 m', const &ap9 m2)
leDico raphical e1!ality$
                  bool operator == (const &ap9 m', const &ap9 m2)
```

Semantics

- Defining semantics of itl concepts via sets of laws
  - aka c++0x axioms
- Checking law sets via automatic testing:
  - A ***La***w ***Ba***sed ***Te***st ***A***utomaton ***LaBatea***



Commutativity<T a, U b, +>:
a + b = b + a;

Semantics

- Lexicographical Ordering and Equality
  - For all itl containers `operator <` implements a ***strict weak ordering***.
  - The ***induced equivalence*** of this ordering is ***lexicographical equality*** which is implemented as `operator ==`
  - This is in line with the semantics of SortedAssociativeContainers

Semantics

- Subset Ordering and Element Equality
  - For all itl containers function `contained_in` implements a ***partial ordering***.
  - The ***induced equivalence*** of this ordering is ***equality of elements*** which is implemented as function `is_element_equal`.

Semantics

- itl::Sets
- All itl sets implement a ***Set Algebra***, which is to say satisfy a "*classical*" set of laws . . .
  - . . . using **`is_element_equal`** as equality
  - Associativity, Neutrality, Commutativity (for + and &)
  - Distributivity, DeMorgan, Symmetric Difference
- Most of the itl sets satisfy the classical set of laws even if . . .
  - . . . lexicographical equality: **`operator ==`** is used
  - The differences reflect proper inequalities in sequence that occur for **`separate_interval_set`** and **`split_interval_set`**.

Semantics

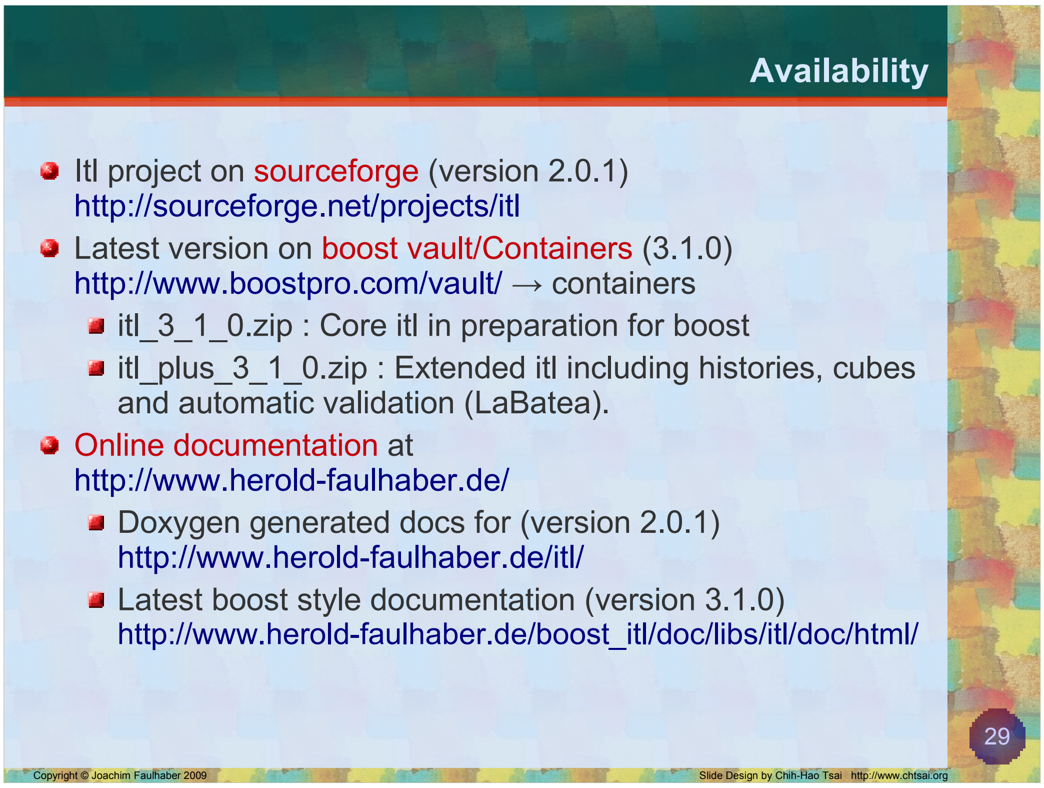
Implementation

- Itl containers are implemented based on std::set and std::map
  - Basic operations like *adding* and *subtracting* intervals or interval value pairs perform with a time *complexity between** *amortized O(log n)* and *O(n)*, where *n* is the number of intervals of a container.
  - Operations like *addition* and *subtraction* of whole containers are having a worst case complexity of *O(m log(n+m))*, where *n* and *m* are the numbers of intervals of the containers to combine.

    * : Consult the library documentation for more detailed information.

Future Works

- Implementing interval_maps of sets more efficiently
- Revision of features of the extended itl (itl_plus.zip)
  - Decomposition of histories: k histories h_k with attribute types $A_1, ..., A_k$ are "*decomposed*" to a product history of tuples of attribute sets:
    $(h_1<T,A_1>,..., h<T,A_k>) \rightarrow h<T, (set<A_1>,\ldots, set<A_k>)>$
  - Cubes (generalized crosstables): Applying *aggregate on collision* to maps of tuple value pairs in order to organize hierachical data and their aggregates.



Availability

- Itl project on sourceforge (version 2.0.1)
  http://sourceforge.net/projects/itl
- Latest version on boost vault/Containers (3.1.0)
  http://www.boostpro.com/vault/ → containers
  - itl_3_1_0.zip : Core itl in preparation for boost
  - itl_plus_3_1_0.zip : Extended itl including histories, cubes and automatic validation (LaBatea).
- Online documentation at
  http://www.herold-faulhaber.de/
  - Doxygen generated docs for (version 2.0.1)
    http://www.herold-faulhaber.de/itl/
  - Latest boost style documentation (version 3.1.0)
    http://www.herold-faulhaber.de/boost_itl/doc/libs/itl/doc/html/

Availability

- Boost sandbox
  https://svn.boost.org/svn/boost/sandbox/itl/
  - Core itl: Interval containers in prepartion for boost
    https://svn.boost.org/svn/boost/sandbox/itl/boost/itl/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/itl/
  - Extended itl_xt: “histories” and cubes
    https://svn.boost.org/svn/boost/sandbox/itl/boost/itl_xt/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/itl_xt/
  - Validater LaBatea:
    Compiles with msvc-8.0 or newer, gcc-4.3.2 or newer
    https://svn.boost.org/svn/boost/sandbox/itl/boost/validate/
    https://svn.boost.org/svn/boost/sandbox/itl/libs/validate/