intervalset Documentation

Millian Poquet

Feb 22, 2019

Contents

intervalset is a C++ library to manage sets of closed intervals of integers. This is a simple wrapper around Boost.Icl.

CHAPTER 1

Install

1.1 From Nix

Nix is a package manager with amazing properties.
It can be installed on any Linux or Unix-like (e.g., macOS) system.

```
# Released version
nix-env -iA intervalset \
        -f 'https://github.com/oar-team/kapack/archive/master.tar.gz'

# Latest version
git clone https://framagit.org/batsim/intervalset.git
nix-env -f intervalset/ci/default.nix -i
```

1.2 Build It Yourself

intervalset uses the Meson build system (and therefore Ninja).

```
# Get the code
git clone https://framagit.org/batsim/intervalset.git
cd intervalset

# Prepare build (call meson)
meson build # install prefix can be changed with --prefix
cd build

# Build the library (and the unit tests if google-test is found)
ninja

# Run the tests
```

(continues on next page)

(continued from previous page)

```
ninja test

# Install
ninja install
```

Note:

You first need to install **intervalset**'s dependencies. Notably:

- decent C++ compiler
- decent Meson
- Boost.Icl
- gtest

Up-to-date dependencies are defined in the CI Nix package.
The CI package is probably based on the release Nix package.
Give a look at the CI build output for versions.

CHAPTER 2

Usage

2.1 How to link?

intervalset must be linked if one wants to use it.

- Either with `-lintervalset`
- Or `-lintervalset_static` (to bundle **intervalset** with your program)

2.2 Quick example

```
#include <intervalset.hpp>

void usage_example()
{
    // Creation
    IntervalSet a; // empty
    IntervalSet b = IntervalSet::ClosedInterval(0,4); // [0,4] AKA {0,1,2,3,4}
    IntervalSet c = IntervalSet::from_string_hyphen("3-5,8"); // [3,5][8] AKA {3,4,5,
↪8}

    // Format
    c.to_string_brackets(); // "[3,5][8]"
    c.to_string_hyphen(); // "3-5,8"
    c.to_string_elements(); // "3,4,5,8"

    // Set operations
    IntervalSet intersection_set = (a & c); // {}
    IntervalSet union_set = (b + c); // [0,5][8]
    IntervalSet difference_set = (b - c); // [0,2]

    // In-place set operations
```

(continues on next page)

(continued from previous page)

```
    a += IntervalSet::empty_interval_set(); // a remains empty
    // -= and &= are also defined.

    // Common operations
    int number_of_elements = c.size(); // 4
    int first_element = c.first_element(); // 3
    IntervalSet first_two_elements = c.left(2); // [3-4]
    IntervalSet two_random_elements = c.random_pick(2); // Two different random␣
↪values from c
    c.contains(0); // false
    a.is_subset_of(c); // true
}
```

CHAPTER 3

API showcase

3.1 Constructors

```
#include <intervalset.hpp>

void constructors_example()
{
    // Create an empty IntervalSet
    IntervalSet s1;

    // Copy an existing IntervalSet
    IntervalSet s2 = s1;

    // Create an IntervalSet from one interval
    IntervalSet s3 = IntervalSet::ClosedInterval(0,1);

    // Create an IntervalSet from one integer
    IntervalSet s4 = 2;
}
```

3.2 Set operations

```
#include <intervalset.hpp>

void set_operations_example()
{
    IntervalSet s1 = IntervalSet::from_string_hyphen("3,4-7,10-20,22,24-28");
    IntervalSet s2 = IntervalSet::from_string_hyphen("4,19-21,23");

    // Classical set operations
    IntervalSet s_union = (s1 + s2);
```

(continues on next page)

(continued from previous page)

```
    IntervalSet s_intersection = (s1 & s2);
    IntervalSet s_difference = (s1 - s2);

    // In-place operations
    s1 += s2; // s1 = s1  s2
    s1 &= s2; // s1 = s1  s2
    s1 -= s2; // s1 = s1 \ s2
}
```

3.3 Accessing elements

```
#include <intervalset.hpp>

void access_example()
{
    IntervalSet s = IntervalSet::from_string_hyphen("3,10-16");

    s.first_element(); // 3
    s.left(1); // 3
    s.left(2); // {3,10}
    s.left(4); // {3,10,11,12}
    s.random_pick(2); // Two different random elements from s

    // Access can be done via operator[]
    // WARNING: This is very slow! Use iterators if performance matters.
    s[0]; // 3
    s[4]; // 13
}
```

3.4 Testing membership

```
#include <intervalset.hpp>

void membership_example()
{
    IntervalSet s1 = IntervalSet::from_string_hyphen("3,4-7,10-20,22,24-28");
    IntervalSet s2 = IntervalSet::from_string_hyphen("5-6,15,19,28");

    s1.contains(4); // Does s1 contains 4? Yup.
    s1.contains(IntervalSet::ClosedInterval(8,25)); // Nope.
    s2.is_subset_of(s1); // Yup, s2  s1
}
```

3.5 Iterating elements and intervals

```
#include <intervalset.hpp>

void traversal_example()
```

(continues on next page)

(continued from previous page)

```
{
    IntervalSet s = IntervalSet::from_string_hyphen("4,19-21,23");

    // The intervals can be traversed
    for (auto it = s.intervals_begin(); it != s.intervals_end(); ++it)
    {
        // Use operator* to retrieve the interval
        const IntervalSet::ClosedInterval & itv = *it;

        // The two bounding elements can be retrieved this way
        int interval_minimum_element = lower(itv);
        int interval_maximum_element = upper(itv);
    }

    // The individual values can also be traversed
    // Please DO note that this may be way slower than iterating over intervals
    for (auto it = s.elements_begin(); it != s.elements_end(); ++it)
    {
        // Use operator* to retrieve the element value
        int element = *it;
    }
}
```

CHAPTER 4

API reference

struct IntervalSet

Public Types

typedef boost::icl::closed_interval<int, std::less> **ClosedInterval**
A closed interval of integers.

Public Functions

IntervalSet()
Create an empty *IntervalSet*.

IntervalSet(**const** *ClosedInterval* &*interval*)
Create an *IntervalSet* made of a single interval.

IntervalSet(**const** *IntervalSet* &*other*)
Create an *IntervalSet* from another *IntervalSet* (copy constructor).

IntervalSet(int *integer*)
Create an *IntervalSet* from a single integer.

IntervalSet::element_const_iterator **elements_begin**() **const**
Iterator to beginning **element**.

Note: Iterating **intervals** is much more efficient (via *intervals_begin()* and *intervals_end()*).

IntervalSet::element_const_iterator **elements_end**() **const**
Iterator to ending **element**.

Note: Iterating **intervals** is much more efficient (via *intervals_begin()* and *intervals_end()*).

IntervalSet::const_iterator **intervals_begin**() **const**
Iterator to beginning **interval**.

IntervalSet::const_iterator **intervals_end**() **const**
Iterator to ending **interval**.

void **clear**()
Remove all the elements in the *IntervalSet*. In other words, make it empty.

void **insert**(**const** *IntervalSet* &*interval_set*)
Insert an *IntervalSet* in another. This is similar to *operator+=(const IntervalSet &)*.

void **insert**(*ClosedInterval* *interval*)
Insert a ClosedInterval in an *IntervalSet*.

void **insert**(int *integer*)
Insert an integer in an *IntervalSet*.

void **remove**(**const** *IntervalSet* &*interval_set*)
Remove an *IntervalSet* from another. This is similar to operator-=(const Intervalset &).

void **remove**(*ClosedInterval* *interval*)
Remove a ClosedInterval from an *IntervalSet*.

void **remove**(int *integer*)
Remove an integer from an *IntervalSet*.

IntervalSet **left**(unsigned int *nb_integers*) **const**
Create a sub-IntervalSet made of the nb_integers leftmost elements of the source *IntervalSet*.

Pre The source *IntervalSet* must contains nb_integers or more elements.

IntervalSet **random_pick**(unsigned int *nb_integers*) **const**
Create a sub-IntervalSet made of nb_integers randomly-picked elements from the source *IntervalSet*.

Pre The source *IntervalSet* must contains nb_integers or more elements.

IntervalSet::const_iterator **biggest_interval**() **const**
Returns a const iterator to the biggest ClosedInterval in an *IntervalSet*.

int **first_element**() **const**
Returns the value of the first element of an *IntervalSet*.

Pre The *IntervalSet* must **NOT** be empty.

unsigned int **size**() **const**
Returns the number of **elements** of an *IntervalSet*.

bool **is_empty**() **const**
Returns whether an *IntervalSet* is empty. An empty *IntervalSet* does not contain any element.

bool **contains**(int *integer*) **const**
Returns whether an *IntervalSet* contains an integer.

bool **contains**(**const** *ClosedInterval* &*interval*) **const**
Returns whether an *IntervalSet* fully contains a ClosedInterval.

bool **is_subset_of**(**const** *IntervalSet* &*other*) **const**
Returns whether an *IntervalSet* is a subset of another *IntervalSet*.

std::string **to_string_brackets**(**const** std::string *&union_str* = "", **const** std::string *&opening_bracket* = "[", **const** std::string *&closing_bracket* = "]", **const** std::string *&sep* = ",") **const**

Returns a string representation of an *IntervalSet*.

This is the classical representation used in mathematics. For example, {1,2,3,7} is represented as [1,3][7].

std::string **to_string_hyphen**(**const** std::string *&sep* = ",", **const** std::string *&joiner* = "-") **const**

Returns a string representation of an *IntervalSet*.

This is a compact representation where {1,2,3,7} is represented as 1-3,7. Use sep=' ' to get a Batsim-compatible representation (see Batsim documentation about Interval sets representation).

string **to_string_elements**(**const** std::string *&sep* = ",") **const**

Returns a string representation of an *IntervalSet*.

This is the set representation of an *IntervalSet*. For example, {1,2,3,7} is represented as 1,2,3,7

IntervalSet &**operator=**(**const** *IntervalSet* *&other*)

Assignment operator. Reset an *IntervalSet* content to the one of another *IntervalSet*.

IntervalSet &**operator=**(**const** *IntervalSet*::*ClosedInterval* *&interval*)

Assignment operator. Reset an *IntervalSet* content to the one of a ClosedInterval.

bool **operator==**(**const** *IntervalSet* *&other*) **const**

Returns whether two *IntervalSet* exactly contain the same elements.

bool **operator!=**(**const** *IntervalSet* *&other*) **const**

Returns whether the content of two *IntervalSet* is different.

IntervalSet &**operator-=**(**const** *IntervalSet* *&other*)

Difference + assignment operator. This is similar to *remove(const IntervalSet &)*.

a -= b; means "Set a's value to be a without the elements of b".

IntervalSet &**operator+=**(**const** *IntervalSet* *&other*)

Union + assignment operator. This is similar to *insert(const IntervalSet &)*.

a += b; means "Set a's value to be the union of a and b".

IntervalSet **operator-**(**const** *IntervalSet* *&other*) **const**

Difference operator. a - b returns an *IntervalSet* of the elements that are in a but are not in b.

IntervalSet **operator+**(**const** *IntervalSet* *&other*) **const**

Union operator. a + b returns an *IntervalSet* of the elements that are in a, in b, or both in a and b.

int **operator[]**(int *index*) **const**

Subscript operator.

Returns the index-th element of the *IntervalSet*.

Pre index must be positive and strictly smaller than *size()*

Public Static Functions

IntervalSet **from_string_hyphen**(**const** std::string *&str*, **const** std::string *&sep* = ",", **const** std::string *&joiner* = "-")

Parse an *IntervalSet* string representation and return the corresponding *IntervalSet*.

See *IntervalSet::to_string_hyphen* for representation details.

IntervalSet **empty_interval_set**()
Returns an empty *IntervalSet*.

CHAPTER 5

Changelog

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog and intervalset adheres to Semantic Versioning. The public API of intervalset is simply the public C++ functions and types defined by the library.

5.1 Unreleased

- Commits since v1.2.0

5.2 v1.2.0

- Commits since v1.1.0
- Release date: 2019-02-22

5.2.1 Added

- New `is_empty` method, that returns whether an intervalset is empty.
- Full API is now documented on readthedocs.

5.3 v1.1.0

- Commits since v1.0.0
- Release date: 2018-11-09

5.3.1 Changed

- Build system changed from CMake to Meson.

5.4 v1.0.0

- Release date: 2018-10-16

Index

I