Industrial and Open Source Case Studies

Deliverable D4-1

ANR project VECOLIB

March 2015

This deliverable presents the case studies selected for the VECOLIB project. A first category of case studies includes libraries of containers in different programming languages: Ada, C, C++, and Java. The presentation classify these containers on the kind of data structures used to implement the containers (e.g., sequence, tree, associative arrays) and on their use of the dynamic storage facility.

A second category of case studies includes programs using libraries of containers. These programs are representative for the use containers in Ada and C programs.

Contents

1 Libraries of Containers

1.1 Containers by Languages

Most of the existing imperative programming languages provide libraries of containers. We present below a commented list of the libraries freely available (with GPL or LGPL license) for Ada, C, C++, and Java.

1.1.1 Containers in Ada and SPARK

The Ada 2012 standard defines a new set of containers comparing with Ada 2005. The source code is available to download at [2] or to browse only at [1]. These libraries are documented in [3, 6].

All the containers are generic, i.e., parameterized by the signature of the data stored. However, some of them allows only instantiation with data of fixed size.

With respect to the size of the container, the Ada containers belong to tree classes:

1. variable size but storing data of definite[1] type,
2. variable size and data of indefinite[2] type, and
3. bounded size storing data of definite type.

In the first two categories, the size is not bounded. The last category is mainly used for high integrity applications, where the dynamic memory allocation is forbidden. The containers are stored in bounded arrays and the library encodes the data structure (vector, tree, map) inside the array.

With respect to the collection organization, the Ada containers belong to tree categories:

- Sequential containers:
  - vectors (files `a-convec.ad*` for unbounded, `a-cobove.ad*` for bounded),
  - doubly linked lists (files `a-cdlili.ad*` for unbounded, `a-cbdlli.ad*` for bounded).
- Multiway trees: the container encodes trees with an unbounded number of children and a reference to the parent in each children. The implementation of the unbounded version uses alias types[3]. The bounded container is encoded using arrays.
- Associative containers:
  - hashed maps (files `a-cohama.ad*` for unbounded, `a-cbhama.ad*` for bounded),
  - sets (files `a-cohase.ad*` for unbounded, `a-cbhase.ad*` for bounded),
  - ordered maps (files `a-coorma.ad*` for unbounded, `a-cborma.ad*` for bounded),
  - ordered sets (files `a-coorse.ad*` for unbounded, `a-cborse.ad*` for bounded),
  - and multisets (files `a-coormu.ad*` for unbounded).

[1] A definite type in Ada has fixed size.
[2] An indefinite type in Ada has variable size
[3] Alias types in Ada are equivalent to reference types.

The hash maps and sets are implemented using hash tables with closed addressing (tables of lists). The ordered maps, sets and multisets are implemented using implemented using red-black trees.

- Queue containers over definite types: synchronized and priority queues. These containers are mainly used for concurrent (tasking) applications.

These libraries are not formally specified. However, some of them are very similar to the SPARK containers that come with a formal specification (see the next paragraph).
SPARK provides the following *formal containers*:

- formal doubly linked lists (files `a-cfdlli.ad*`),
- formal hashed maps and sets (files `a-cfhama.ad*` resp. `a-cfhase.ad*`),
- formal vectors (files `a-cofove.ad*`),
- formal indefinite vectors (files `a-cfinve.ad*`) implemented using a special form of dynamic arrays,
- formal maps and sets (files `a-cforma.ad*` resp. `a-cforse.ad*`).

They are all implemented using bounded arrays. The following site [5] provides the code and the axiomatization in WHY3 of these containers.

1.1.2 Containers in C

The standard library of C does not provide containers.

The GNOME project provide a rich library of containers called `glib` [22]. The containers are made generic by using as type of elements the `void*` type. The following containers are implemented:

- singly linked lists (files `gslist.*`),
- doubly linked lists (files `glist.*`),
- arrays of elements or of pointers (files `garray.*`),
- hash tables (files `ghash.*`),
- queues implemented by doubly linked lists (files `gqueue.*`),
- sequences implemented by balanced trees (files `gsequence.*`),
- multi-way trees (files `gtree.*` and `gnode.*`).

Although the code is very well documented, there is no formal specification of these containers in some formal language, e.g., ACSL [11].

A formal specification in ACSL of some C containers (heap and vector) is provided in [25]. A notable initiative is the *ACSL by Example* [8] which provides specification in ACSL of a part of the standard C library, mainly the algorithmic part. For the moment, only the stack container is axiomatized.

Other implementations of a standard library of containers for C exists:

- SGLIB [4, 26] provides a library to manipulate user-defined types seen as linked lists, red-black trees, or hashed sets and maps. The library uses C macros to implement the container operations on C code. This library is interesting because it provides a generic code without using pointer to functions and `void` pointers.
- CCL [20] provides a library of containers which mainly translates the C++ library STL in C. The implementation provided uses pointers to functions to obtain genericity.

1.1.3 Containers in C++

Several libraries for containers are available for C++, the most known ones being STL [15]. In 2011, the C++ standard has fixed a standard library which contains most of the existing containers. However, there are still other distributions for C++ containers, e.g., Boost [13].

In all these libraries, containers are implemented as C++ templates, so generic by the type of the elements stored.

The C++ standard library: It includes a rich set of containers mainly inspired by the STL an Boost projects described below. The documentation is available at [17]. The following containers are available:

- Sequential containers:
  - array of fixed size (`<array>`),
  - vector of dynamic size (`<vector>`),
  - deque (double ended queue) as dynamic array (`<deque>`),
  - singly linked lists (`<forward_list>`),
  - doubly linked lists (`<list>`).
- Container adaptors, by default implemented using deque:
  - stack (`<stack>`),
  - queue (`<queue>`), and
  - priority queue (`<priority_queue>`).
- Associative containers, implemented by rd-black trees:
  - set (`<set>`),
  - multiple-key set (`<multiset>`),
  - map (`<map>`), and
  - multiple-key map (`<multimap>`).
- Unordered associative containers, implemented by hash tables:
  - unordered set (`<unordered_set>`),
  - unordered multiple-key set (`<unordered_multiset>`),
  - unordered map (`<unordered_map>`), and
  - unordered multiple-key map (`<unordered_multimap>`).

The code is very well documented in the natural language, but no formal specification is available for the implementations of these containers. Some static analysis approaches, e.g. [12, 7], are using Abstract Data Types specifications for these containers. In [23] specification of STL *concepts* (aka interfaces) for the container types are formalized using first order logic. The initiative *ACSL by Example* [8] targets to provide formal ACSL specifications of these libraries.

The STL library: The Standard Template Library (STL) has been developed by SGI. The documentation [15] and the source code [16] are available under the GPL license. In addition to the standard containers, the library provides the following one:

- bit vector which is a vector with optimized representation.

The BOOST library: is a library released under the Boost Software License, which allows anyone to use, modify, and distribute the libraries for free. The source [14] and the the documentation [13] are freely available. Boost adds to the containers in the standard C++ library the following ones:

- bi-map, where a container may be seen through different interfaces (e.g., both vector and set),
- circular buffer of fixed capacity implemented by an array,
- priority queues implemented by heap-like data structures (binomial heap, Fibonacci heap, etc.),
- intrusive version of the standard containers, where the memory is not dynamically allocated and objects are stored without copy when pushed,
- multi-dimensional arrays of fixed sizes.

1.1.4 Containers in Java

The implementation of containers in Java is freely at several sites, e.g. [21]. The containers implemented are:

- Sequence containers:
  - vector (of dynamic size `ArrayList`),
  - deque (`ArrayDeque`),
  - doubly linked lists (`LinkedList`),
  - bit set (dynamic array of longs, `BitSet`).
- Associative containers:
  - hash tables (`Hashtable`) implemented with closed addressing into an array,
  - hash maps and sets (`HashMap` resp. `HashSet`) implemented using array of lists or of tree maps,
- Ordered associative containers:

Table 1: Data structures used in container libraries

Language	Array	List	Hash tables	Binary trees	Other
Ada 2012	Bounded containers	doubly linked	maps & sets	ordered maps & (multi-)sets by RBT	multi-way trees
SPARK	formal containers				
C	array	singly & doubly linked, queues	hash tables	sequences by AVL	multi-way trees
C++	array, vector, deque	singly & doubly linked	unordered (multi-)set & (multi-)map	(multi-)set & (multi-)map by AVL or RBT	bi-map, bit vector, heap-like
Java	vector, deque	doubly linked	map & set	map & set	bit set, overlapping hash table & list

- linked hash maps (`LinkedHashMap`) implemented by linking the elements of a hash map by a doubly linked list,
- tree map and set (`TreeMap` resp. `TreeSet`).

As far as we know, there is no formal specification of these containers in some logic formalism.

1.2 Containers by Data Structure

Table 1 classified the containers by the data structures used to implement them.

1.3 Case Studies

The VECOLIB project will focus on the verification container libraries for Ada, SPARK and C languages for the following reasons:

1. The libraries of containers for these languages cover the most used containers and data structures, as proved by in Table 1.
   A notable absence are bit sets or vectors. However, these containers should be easily specified and verified using the current state of the art theories and solvers for bit vectors.

2. The data structures included are still out of the scope of the verifications tools at the present time.

3. The partners of the project (AdaCore and CEA) are interested by having verified container libraries for these languages.

4. The partners of the project (AdaCore and CEA) develop tools for the verification of the programs written in Ada and C.
   For the Ada programs and libraries, the AdaCore partner provides a translator from Ada to C. So, the final work will consist in verifying C implementation of container libraries specified using some ACL theory.

2 Programs with Containers

In addition to the verification and analysis of container implementations, the VECOLIB project will develop methods for programs using the container libraries. The properties targeted are functional properties on the content of the container.

2.1 Case Studies in Ada

The case studies for containers in Ada will come from the SPARK2014 test suite, which includes several examples using SPARK formal containers. These examples are written in SPARK and some of them have been formally verified using the SPARK2014 toolset. The git repository of SPARK2014 can be accessed anonymously using:

```
git clone --recursive https://forge.open-do.org/anonscm/git/spark2014/spark2014.git
```

The test suite can be found in the directory `spark2014/testsuite/gnatprove/tests`.

Here are a progressive sample of them, in increasing order of complexity:

- `K624-029__search_list`: a simple search function on doubly linked lists (38 lines). It has been verified automatically.
- `M607-025__rev_stack`: a program using a stack, implemented as an array, to reverse a list (67 lines dealing with lists). It has been verified automatically.
- `K624-029__amortize_queue`: a functional amortized queue made of two vectors (142 lines). Absence of run-time error has been verified automatically, but not the whole functional specification.
- `vscomp2014_partition_functional`: a partial solution to the partition refinement problem of VSCOMP in 2014 [9] using doubly linked lists, vectors and ordered maps (438 lines). This test is not completely verified by the SPARK2014 toolset.

2.2 Case Studies in C

For the C programs, two classes of applications will be considered:

Memory allocators: The memory allocators use containers to store the free or occupied list of allocated blocks. Recently, some of thee allocators (e.g., TLSF [19]) have been developed using the Event B method [24], where the data structures of the allocator are modeled using sets and maps. The translation to C uses the containers for sets and maps. The interesting part of these examples is that the data stored by these containers are related by numerical properties, e.g., the data in a set belongs to some interval of values stored as key of a map.

Problems from the Verified Software Competition (VSCOMP): The competition took place for the first time in 2010 [18] and two other editions took place in 2012 [10] and 2014 [9]. Several problems proposed require to use containers: the partition refinement problem, the patience solitaire game, the graph cloning, etc. We will consider C implementations of these problems to test the verification methods developed inside the project.

References

[1] GCC libraries for Ada. https://sourceware.org/svn/gcc/tags/gcc_4_6_3_release/gcc/ada/.

[2] GNAT web site. https://www.gnu.org/software/gnat/.

[3] Online documentation of GNAT libraries for Ada. https://gcc.gnu.org/onlinedocs/gnat_rm/The-GNAT-Library.html#The-GNAT-Library.

[4] SGLIB - A simple generic library for C. http://sglib.sourceforge.net/.

[5] Specification and code for SPARK containers. https://forge.open-do.org/scm/?group_id=39.

[6] John Barnes. *Programming in Ada 2012*. Cambridge University Press, 2014.

[7] N. Blanc, A. Groce, and D. Kroening. Verifying c++ with stl containers via predicate abstraction. In *ASE*, pages 521–524. ACM, 2007.

[8] Jochen Burghardt and Jens Gerlach. ACSL by example: Towards a verified c standard library, March 2015.

[9] Ernie Cohen, Marcelo Frias, Peter Müller, and Natarajan Shankar. Verified software competition (VSCOMP). http://vscomp.org/.

[10] Jean-Christophe Filliâtre, Andrei Paskevich, and Aaron Stump. The 2nd Verified Software Competition: Experience Report. In *COMPARE 2012, Comparative Empirical Evaluation of Reasoning Systems, 1st Intl. Workshop*, volume 873, pages 36–49, Manchester, United Kingdom, June 2012. CEUR Workshop Proceedings.

[11] Frama-C team. *ACSL: ANSI/ISO C Specification Language, version 1.6*.

[12] Douglas Gregor and Sibylle Schupp. Stllint: lifting static checking from languages to libraries. *Softw., Pract. Exper.*, 36(3):225–254, 2006.

[13] BOOST group. BOOST: a C++ library. https://theboostcpplibraries.com/containers.

[14] BOOST group. BOOST code. https://www.boost.org.

[15] SGI group. STL: a C++ library. https://www.sgi.com/tech/stl/.

[16] SGI group. STL code. https://www.sgi.com/tech/stl/download.html.

[17] ISO. The C++ standard library. http://www.cplusplus.com/reference/stl/, 2011.

[18] Vladimir Klebanov, Peter Müller, Natarajan Shankar, Gary T. Leavens, Valentin Wüstholz, Eyad Alkassar, Rob Arthan, Derek Bronish, Rod Chapman, Ernie Cohen, Mark Hillebrand, Bart Jacobs, K. Rustan M. Leino, Rosemary Monahan, Frank Piessens, Nadia Polikarpova, Tom Ridge, Jan Smans, Stephan Tobies, Thomas Tuerk, Mattias Ulbrich, and Benjamin Weiß. The 1st Verified Software Competition: Experience report. In Michael Butler and Wolfram Schulte, editors, *Proceedings, 17th International Symposium on Formal Methods (FM)*, volume 6664 of *LNCS*. Springer, 2011. Materials available at www.vscomp.org.

[19] Miguel Masmano, Ismael Ripoll, Alfons Crespo, and Jorge Real. TLSF: A new dynamic memory allocator for real-time systems. In *ECRTS*, pages 79–86. IEEE Computer Society, 2004.

[20] Jacob Navia. A container library for C. http://www.cs.virginia.edu/~lcc-win32/ccl/ccl.html.

[21] OpenJDK. The Java containers. http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util.

[22] GNOME project. GLIB library. https://git.gnome.org/browse/glib/tree/glib.

[23] Alexander Stepanov and Paul McJones. *Elements of Programming*. Addison-Wesley Professional, June 2009. ISBN-13: 978-0-321-63537-2.

[24] W. Su, J.-M. Abrial, G. Pu, and B. Fang. Building memory allocators using B, 2015. submitted.

[25] Asma Tafat. *Preuves par raffinement de programmes avec pointeurs*. PhD thesis, University of Paris 11, 2013.

[26] Marian Vittek, Peter Borovansky, and Pierre-Etienne Moreau. A simple generic library for c. In *Reuse of Off-the-Shelf Components: , Proceedings of 9th International Conference on Software Reuse, Tur in, Italy*, pages 423–426. Springer, 2006.