A Fully Verified Container Library

Nadia Polikarpova

joint work with

Julian Tschannen

Carlo A. Furia

Department of Computer Science
Chair of Software Engineering

How is data structure verification still a thing?

Precise reasoning for programs using containers [POPL'11]

An

"... data structures are a natural candidate for full functional verification" [Zee, Kuncak, Rinard, PLDI'08]

Full functional verification of linke

paration logic [PLDI'14]

... so why aren't real container libraries verified?

Natural pr

Type-based data structure verification [PLDI'09]

Specification of red-black tr

Formalized verification of snapshotable trees [VSTTE'12]

Three challenges of realistic libraries

1. They use complex object structures

   how to reason about them in a sound and modular fashion?

2. They need clean interfaces

   how to provide abstract yet precise interface specs?

3. They are large and use nontrivial algorithms

   how to make the verifier scale?

AutoProof to the rescue!

Linked list: interface specs

```
class LINKED_LIST [T] model values

  feature {public}
    ghost values: MML_SEQUENCE [T]

    first: T -- First element.
      require not values.is_empty
      ensure Result = values [1]

    extend_back (v: T) -- Insert v at the back.
      modify model Current [values]
      ensure values = old values + <v>
...
```

Model-based contracts specify behavior in terms of abstract state

Three challenges of realistic libraries

1. Complex object structures

   how to reason about them in a sound and modular fashion?

2. Clean interfaces

   Model-based contracts: abstract yet precise interface specifications

3. Size and nontrivial algorithms

   how to make the verifier scale?

Linked list: invariants

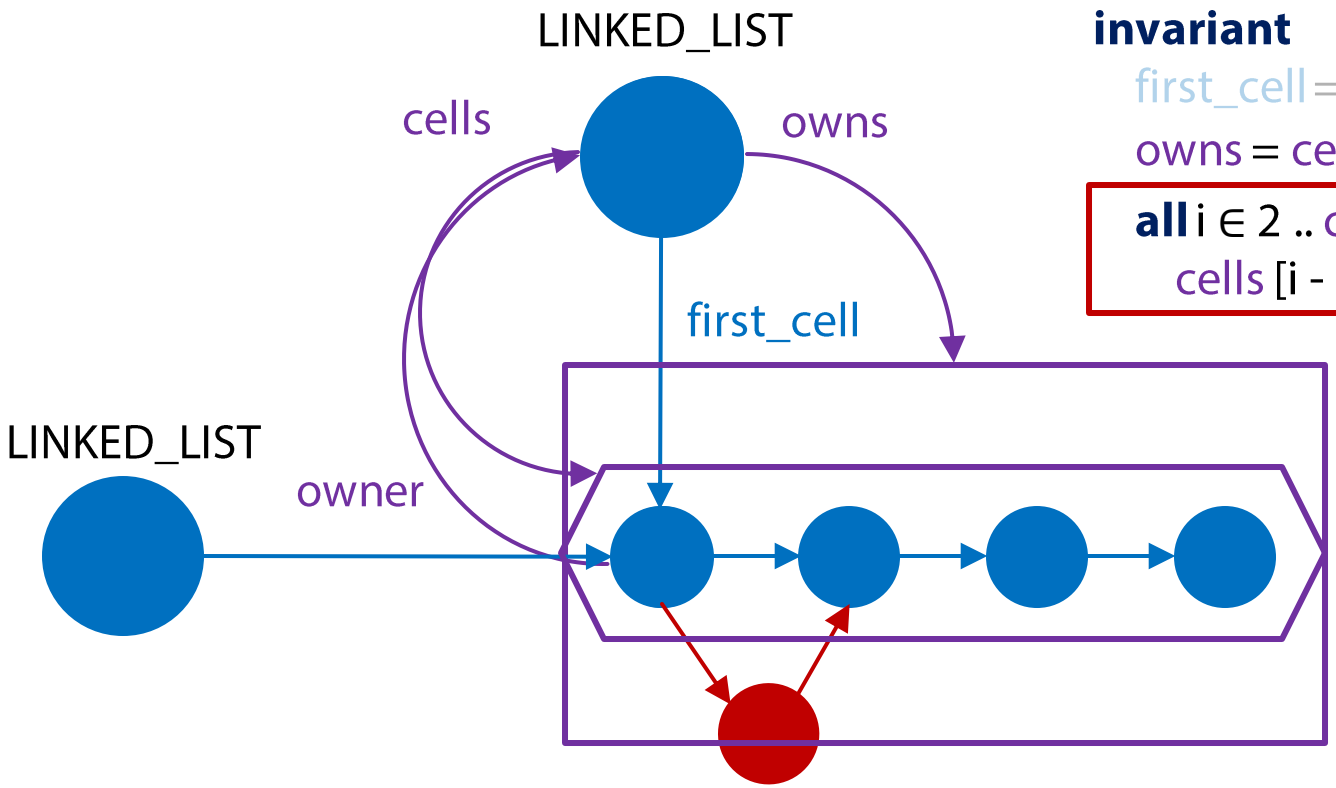
```
class LINKED_LIST [T] model values
...
  first: T -- First element.
      require not values.is_empty
      do
        Result := first_cell.item
      ensure Result = values [1]

feature {private}
   first_cell: CELL [T]
   ghost cells: MML_SEQUENCE [CELL [T]]

invariant
   all i ∈ 1 .. cells.count : values [i] = cells [i].item
   cells.count > 0 implies first_cell = cells [1]
   all i ∈ 2 .. cells.count : cells [i - 1].next = cells [i]
```

Class invariants relate abstract and concrete states

Hierarchical object structures



invariant

first_cell = cells [1]

owns = cells.range

all i ∈ 2 .. cells.count :

cells [i - 1].next = cells [j]

Linked list iterator

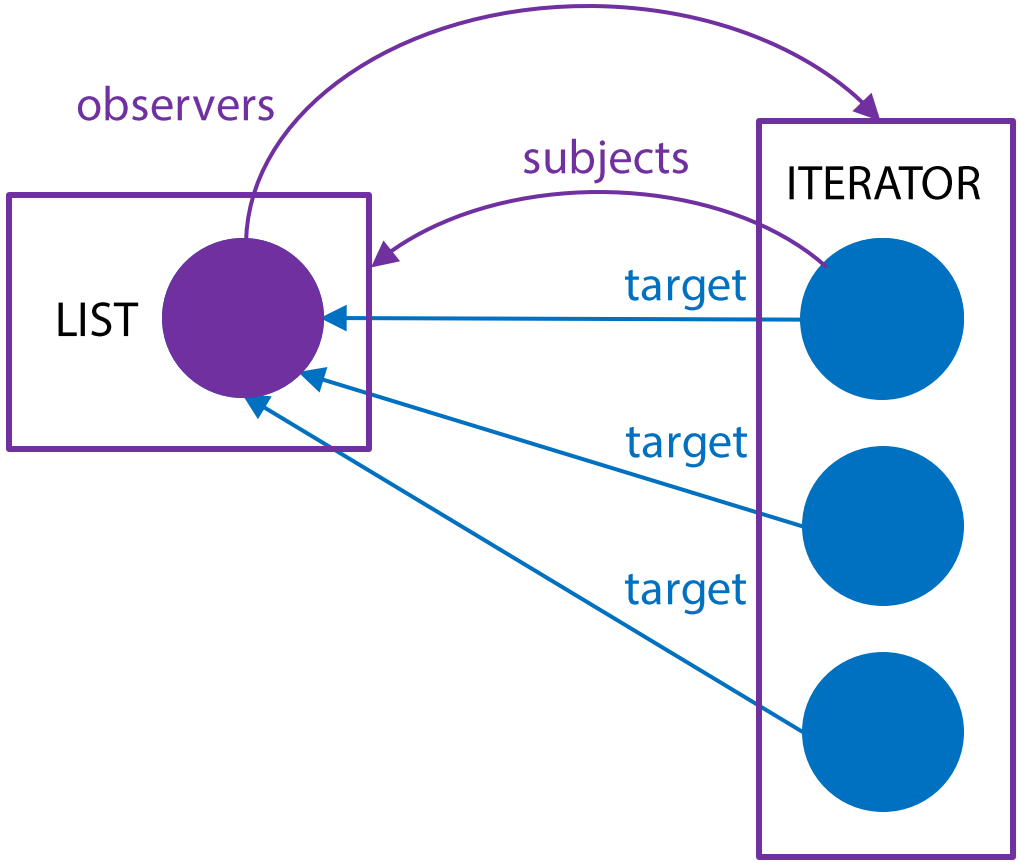
```
class LINKED_LIST_ITERATOR [T] model target, index

feature {public}
    target: LINKED_LIST [T]
    ghost index: INTEGER

feature {private}
    active: CELL [T]

invariant
    1 ≤ index ≤ target.cells.count
    active = target.cells [index]
```

Collaborative object structures



invariant

subjects = { target }

active = target.cells [index]

all s ∈ subjects :

Current ∈ s.observers

Complex structures in the real world

java.util.LinkedList: "*... if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. [...] Note that the fail-fast behavior of an iterator cannot be guaranteed [...] Therefore, it would be wrong to write a program that depended on this exception for its correctness*"

java.util.Map: "*... great care must be exercised if mutable objects are used as map keys. The behavior of a map is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is a key in the map*"

Meanwhile in the magic world of formal methods

EiffelBase2 LINKED_LIST: *if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's ownmethods, an attempt to use the iterator* ***will not verify****. This behavior is 100% guaranteed and incurs no runtime overhead.*

EiffelBase2 MAP: ***no care*** *must be exercised if mutable objects are used as map keys. After modifying a key, the client has to prove that the modification did not affect equals comparison, or else further attempts to use the map* ***will not verify****.*

Three challenges of realistic libraries

1. Complex object structures

Semantic Collaboration: a flexible invariant methodology for reasoning about multi-object invariants

[Polikarpova, Tschannen, Furia, Meyer, FM'14]

2. Clean interfaces

Model-based contracts: abstract yet precise interface specifications

[Polikarpova, Tschannen, Furia, FM'15]

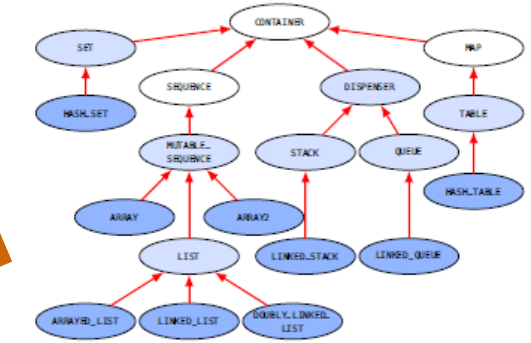
3. Size and nontrivial algorithms

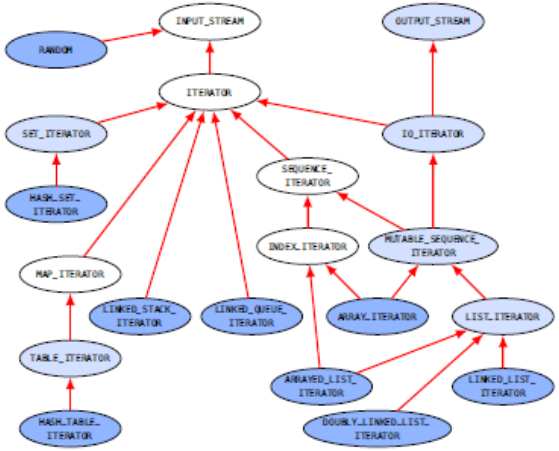
Optimized translation to Boogie to achieve predictable performance on larger problems

EiffelBase2

- 135 public methods in 46 classes
- Provides arrays, lists, stacks, queues, sets, tables
- Features iterators, hash tab… content equality fo… …oad balancing
- Used in teaching since 2011, distributed with EiffelStudio since 2012

Realistic





EiffelBase2: verification results

github.com/nadia-polikarpova/eiffelbase2

46 classes
135 public methods
8.4 kLOC

spec / code
1.4 lines (2.7 tokens)

all methods in 7.2 min
longest method 12 s
avg method < 1 s

7 person months

3 bugs

+ 2 kLOC of client code

Lessons Learned

- verifying a linked list and verifying a linked list from a real library are not the same thing
- boring operations can be the most challenging to verify (content equality, copy constructors, hash table rebalancing)
- you will need all the flexibility but also useful defaults
- light-weight formal methods eliminate most bugs, verification needed to find the most subtle few
- verification promotes good design