

# Construction and Verification of Software

2017 - 2018

**MIEI - Integrated Master in Computer Science and Informatics**  
Consolidation block

**Lecture 6 - Arrays in Separation Logic**

**João Costa Seco** ([joao.seco@fct.unl.pt](mailto:joao.seco@fct.unl.pt))

based on previous editions by **Luís Caires** ([lcaires@fct.unl.pt](mailto:lcaires@fct.unl.pt))



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

# Verifast Example - Bag

The screenshot shows the Verifast IDE interface with the following details:

- File Menu:** File, Edit, View, Verify, Window(Top), Window(Bottom), Help.
- Status Bar:** Breakpoint reached.
- Code Editor:** The code for the `Bag` class is displayed. It includes:
  - A predicate annotation: `/*@ predicate BagInv(int n) = store |-> ?s && nelems |-> n && s != null && 0<=n && n <= s.length && s[0..n] |-> ?elems && s[n..s.length] |-> ?others ; */`
  - A constructor: `public Bag(int size) //@ requires size >= 0; //@ ensures BagInv(0); { store = new int[size]; nelems ≡ 0; }`
  - A method: `boolean add(int v) //@ requires BagInv(_);`
- Local Variables Table:** Shows `size` and `this`.
- Assumptions Table:** Shows:
  - 0 <= size
  - !(this = 0)
  - length(elems) = size
  - all\_eq(elems, 0)
  - !(array = 0)
  - array.length(array) = size
- Heap Chunks Table:** Shows:
  - Bag\_nelems(this, 0)
  - Bag\_store(this, array)
  - java.lang.array\_slice<int32>(array, 0, s
- Steps Panel:** Shows the verification process:
  - Verifying call
  - Consuming assertion
  - Producing assertion
  - Executing statement
  - Executing statement

# Verifast Example - Bag

---

- Fields must be considered in separate heap chunks, pure conditions can be added to assertions and predicates.

```
public class Bag {  
  
    int store[];  
    int nelems;  
  
/*@  
 predicate BagInv(int n) =  
     store |-> ?s  
     &*& nelems |-> n  
     &*& s != null  
     &*& 0<=n &*& n <= s.length  
     &*& array_slice(store,0,n,?elems)  
     &*& array_slice(store,n,s.length,?others)  
     ;  
@*/  
...  
}
```

# Verifast Example - Bag

---

- Fields must be considered in separate heap chunks, pure conditions can be added to assertions and predicates.

```
public class Bag {  
  
    int store[];  
    int nelems;  
  
/*@  
 predicate BagInv(int n) =  
     store |-> ?s  
     &*& nelems |-> n  
     &*& s != null  
     &*& 0<=n &*& n <= s.length  
     &*& s[0..n] |-> ?elems  
     &*& s[n..s.length] |-> ?others  
;  
@*/  
...  
}
```

# Verifast Example - Bag

---

- Fields must be considered in separate heap chunks, pure conditions can be added to assertions and predicates.

```
int get(int i)
  //@ requires BagInv(?n) && 0 <= i && i < n;
  //@ ensures BagInv(n);
{
  return store[i];
}

int size()
  //@ requires BagInv(?n);
  //@ ensures BagInv(n) && result>=0 ;
{
  return nelems;
}
```

# Verifast Example - Bag

---

```
public Bag(int size)
    //@ requires size >= 0;
    //@ ensures BagInv(0);
{
    store = new int[size];
    nelems = 0;
}

boolean add(int v)
    //@ requires BagInv(_);
    //@ ensures BagInv(_);
{
    if(nelems<store.length) {
        store[nelems] = v;
        nelems = nelems+1;
        return true;
    } else {
        return false;
    }
}
```

# Verifast Example - Bag

---

```
public Bag(int size)
    //@ requires size >= 0;
    //@ ensures BagInv(0);
{
    store = new int[size];
    nelems = 0;
}

boolean add(int v)
    //@ requires BagInv(?n);
    //@ ensures BagInv(n+1); // Does not hold, why?
{
    if(nelems<store.length) {
        store[nelems] = v;
        nelems = nelems+1;
        return true;
    } else {
        return false;
    }
}
```

# Verifast Example - Bag

The screenshot shows the Verifast IDE interface with the following components:

- Code Editor:** Displays the Java code for a `Bag` class. The code includes a `BagInv` predicate, an `add` method, and a `get` method. A red error bar at the top indicates "Cannot prove dummy == (dummy + 1)".
- Local Variables Table:** Shows local variables and their values. It includes `n` (value: `(dummy + 1)`), `s` (value: `s`), and `this` (value: `this`).
- Global Variables Table:** Shows global variables and their values. It includes `n` (value: `dummy`), `this` (value: `this`), and `v` (value: `v`).
- Steps History:** Shows the execution history with steps like "Executing statement", "Executing second branch", "Executing statement", "Executing statement", "Consuming assertion", and "Consuming assertion".
- Assumptions:** Lists assumptions made during verification: `!(this = 0)`, `!(this = 0)`, `!(s = 0)`, `0 <= dummy`, `dummy <= arraylength(s)`, and `!n = m`.
- Heap Chunks:** Lists heap chunks: `Bag_nelems(this, dummy)`, `java.lang.array_slice<int32>(s, 0, dummy)`, and `java.lang.array_slice<int32>(s, dummy, n)`.

# Managing arrays in SL

---

```
/*@
predicate AccountInv(Account a;int b) = a.balance |-> b &*& b >= 0;
@*/

public class Account {

    int balance;

    public Account()
    //@ requires true;
    //@ ensures AccountInv(this,0);
    {
        balance = 0;
    }
    ...
}
```

# Managing arrays in SL

---

- The bank holds an array of accounts...

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    Bank(int max)  
    {  
        nelems = 0;  
        capacity = max;  
        store = new Account[max];  
    }  
    ...  
}
```

# Managing arrays in SL

---

- And implements a couple of operations...

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    ...  
    Account retrieveAccount()  
    {  
        Account c = store[nelems-1];  
        store[nelems-1] = null;  
        nelems = nelems-1;  
        return c;  
    }  
    ...  
}
```

# Managing arrays in SL

---

- And implements a couple of operations...

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    ...  
    void addnewAccount()  
    {  
        Account c = new Account();  
        store[nelems] = c;  
        nelems = nelems + 1;  
    }  
    ...  
}
```

# Managing arrays in SL

---

```
/*@
predicate AccountP(unit a,Account c; unit b) = AccountInv(c,?n) &*& b == unit;
@*/

public class Bank {

    /*@
predicate BankInv(int n, int m) =
    this.nelems |-> n &*&
    this.capacity |-> m &*&
    m > 0 &*&
    this.store |-> ?accounts &*&
    accounts.length == m &*&
    0 <= n &*& n<=m &*&
    array_slice_deep(accounts, 0, n, AccountP, unit, _, _) &*&
    array_slice(accounts, n, m,?rest) &*& all_eq(rest, null) == true;
@*/

}
```

# array slice assertions

---

```
predicate array_slice<T>(  
    T[] array,  
    int start,  
    int end;  
    list<T> elements);
```

- **array\_slice(a,s,l,v):**
  - represents the **footprint** of array  $a[s..l-1]$
  - $v$  is a list of the array “values”  $v_i$  such that  $a[i] \rightarrow v_i$
  - $v$  is an immutable pure value (like a OCaml list)
  - $\text{array\_slice}(a,s,l,v)$  is equivalent to the assertion
    - $v = \{ v_s, \dots, v_{l-1} \}$
    - $a[s] \rightarrow v_s \ \&*& \ a[s+1] \rightarrow v_{s+1} \ \&*& \ \dots \ \&*& \ a[l-1] \rightarrow v_{l-1}$

# array slice assertions

---

```
predicate array_slice_deep<T, A, V>(
    T[] array,
    int start,
    int end,
    predicate(A, T; V) p,
    A info;
    list<T> elements,
    list<V> values);
```

- `array_slice_deep(a, s, l, P, info, v, s):`  
as in the (simple) `array_slice`  
`v` is the list of the array “values” `v_i` such that `a[i] |-> v_i`,  
the predicate `P(info,v_i;o_i)` holds for each `v_i`  
and `s` is the list of all values `o_i`

# Managing arrays in SL

---

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    Bank(int max)  
    //@ requires max>0;  
    //@ ensures BankInv(0,max);  
    {  
        nelems = 0;  
        capacity = max;  
        store = new Account[max];  
    }  
    ...  
}
```

# Managing arrays in SL

---

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    Account retrieveLastAccount()  
    //@ requires BankInv(?n,?m) &*& n>0;  
    //@ ensures  BankInv(n-1,m) &*& AccountInv(result,_);  
    {  
        Account c = store[nelems-1];  
        store[nelems-1] = null;  
        // code does not compile without this! Why ?  
        nelems = nelems-1;  
        return c;  
    }  
}
```

# Managing arrays in SL

---

```
public class Bank {  
  
    Account store[];  
    int nelems;  
    int capacity;  
  
    void addnewAccount()  
    //@ requires BankInv(?n,?m) &*& n < m;  
    //@ ensures  BankInv(n+1,m);  
    {  
        Account c = new Account();  
        store[nelems] = c;  
        //@ array_slice_deep_close(store, nelems, AccountP, unit);  
        nelems = nelems + 1;  
    }  
}
```

# array slice “lemmas”

---

```
lemma void array_slice_deep_close<T, A, V>(  
    T[] array, int start, predicate(A, T; V) p, A a);  
  
requires  
  
    array_slice<T>(array, start, start+1, ?elems) &*&  
    p(a, head(elems), ?v);  
  
ensures  
  
    array_slice_deep<T,A,V>(array, start, start+1, p, a, elems, cons(v,nil));
```

- incorporates the spec of an array element in a (singleton) slice spec into a (singleton) slice\_deep spec
- there are other lemmas, that join together slices
- verifast is usually able to apply lemmas automatically, but not always, in that case the programmer needs to “help”, by calling the needed lemmas.

# array slice “lemmas”

---

```
lemma void array_slice_split<T>(T[] array, int start, int start1);  
requires  
    array_slice<T>(array, start, ?end, ?elems) &&  
    start <= start1 && start1 <= end;  
ensures  
    array_slice<T>(array, start, start1, take(start1 - start, elems)) &&  
    array_slice<T>(array, start1, end, drop(start1 - start, elems)) &&  
    elems == append(take(start1 - start, elems), drop(start1 - start, elems));
```

- this “lemma” splits one array slice assertion into two (sub) array slice assertions.

# array slice “lemmas”

```
← → C ⌂ Secure | https://people.cs.kuleuven.be/~bart.jacobs/verifast/examples/rt/Object.javaspec.html

package java.lang;

import java.util.*;

/*@

inductive unit = unit;

inductive pair<a, b> = pair(a, b);

fixpoint a fst<a, b>(pair<a, b> p) {
    switch (p) {
        case pair(x, y): return x;
    }
}

fixpoint b snd<a, b>(pair<a, b> p) {
    switch (p) {
        case pair(x, y): return y;
    }
}

fixpoint t default_value<t>();

inductive boxed_int = boxed_int(int);
fixpoint int unboxed_int(boxed_int i) { switch (i) { case boxed_int(value): return value; } }

inductive boxed_bool = boxed_bool(boolean);
fixpoint boolean unboxed_bool(boxed_bool b) { switch (b) { case boxed_bool(value): return value; } }

predicate array_element<T>(T[] array, int index; T value);
predicate array_slice<T>(T[] array, int start, int end; list<T> elements);
predicate array_slice_deep<T, A, V>(T[] array, int start, int end, predicate(A, T; V) p, A info; list<T> elements)

lemma_auto void array_element_inv<T>();
    requires [?f]array_element<T>(?array, ?index, ?value);
    ensures [f]array_element<T>(array, index, value) && array != null && 0 <= index && index < array.length;
```

# Construction and Verification of Software

2017 - 2018

**MIEI - Integrated Master in Computer Science and Informatics**  
Consolidation block

**Lab Assignment 5 - Introduction to Verifast**

**João Costa Seco** ([joao.seco@fct.unl.pt](mailto:joao.seco@fct.unl.pt))

based on previous editions by **Luís Caires** ([lcaires@fct.unl.pt](mailto:lcaires@fct.unl.pt))



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

# Verifast

---

- Download the binaries of Verifast from
- <https://github.com/verifast/verifast>
- Run **vfide** from the **bin** directory and experiment the examples in the slides. Run the verifier in intermediate points in the code and examine the heap chunks available.

# Exercise 17

---

- Verify classes Bank and BankAccount

```
// Download the zip archive from CLIP with an  
// implementation of a bank account and a bank  
// (store for bank accounts).
```

```
// Write and verify the appropriate representation  
// invariants, pre-conditions and post-conditions to  
// make sure that the BankAccount and Bank abide by the  
// expected business rules.
```

```
// To verify the whole project Open the file Bank.jarsrc
```