U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master in Informatics and Computing Engineering (M.EIC)

M.EIC037 | Formal Methods for Critical Systems 2021/22

Exercises on Program Verification with arrays with Dafny

Note: At least exercises 1 and 2 should be completed in the class.

1. Binary Search

Assume the following implementation of the binary search algorithm in Dafny:

```
// Finds a value 'x' in a sorted array 'a', and returns its index,
// or -1 if not found.
method binarySearch(a: array<int>, x: int) returns (index: int) {
    var low, high := 0, a.Length;
    while low < high {
        var mid := low + (high - low) / 2;
        if {
            case a[mid]  < x => low := mid + 1;
            case a[mid]  > x => high := mid;
            case a[mid] == x => return mid;
        }
    }
    return -1;
}
```

a) Identify adequate pre and post-conditions for this method, and encode them as “requires” and “ensures” clauses in Dafny. You can use the predicate below if needed.

```
// Checks if array 'a' is sorted.
predicate isSorted(a: array<int>)
  reads a
{
    forall i, j :: 0 <= i < j < a.Length ==> a[i] <= a[j]
}
```

If your post-conditions are adequate, the test cases below should be checked successfully by Dafny.

```
// Simple test cases to check the post-condition.
method testBinarySearch() {
    var a := new int[] [1, 4, 4, 6, 8];
    assert a[..]  == [1, 4, 4, 6, 8];
    var id1 := binarySearch(a, 6);
    assert a[3] == 6; // added
    assert id1 == 3;
    var id2 := binarySearch(a, 3);
```

```
    assert id2 == -1;
    var id3 := binarySearch(a, 4);
    assert a[1] == 4 && a[2] == 4; // added
    assert id3 in {1, 2};
}
```

b) Identify an adequate loop variant and loop invariant, and encode them as "decreases" and "invariant" clauses in Dafny.

2. Insertion Sort

Assume the following implementation of the insertion sort algorithm in Dafny:

```
// Sorts array 'a' using the insertion sort algorithm.
method insertionSort(a: array<int>) {
    var i := 0;
    while i < a.Length {
        var j := i;
        while j > 0 && a[j-1] > a[j] {
            a[j-1], a[j] := a[j], a[j-1];
            j := j - 1;
        }
        i := i + 1;
    }
}
```

a) Identify adequate pre and post-conditions for this method, and encode them as "requires" and "ensures" clauses in Dafny. (<u>Suggestion</u>: See SelectionSort.dfy).

   If your post-conditions are adequate, the test cases below should be checked successfully by Dafny.

```
// Simple test case to check the postcondition
method testInsertionSort() {
  var a := new int[] [ 9, 4, 3, 6, 8];
  assert a[..] == [9, 4, 3, 6, 8];
  insertionSort(a);
  assert a[..] == [3, 4, 6, 8, 9];
}
```

b) Identify adequate variants and invariants for the two loops, and encode them as "decreases" and "invariant" clauses in Dafny.

3. **Sorting algorithms**. [Home work]

Implement and verify in Dafny one of the following sorting algorithms: quick sort or merge sort.

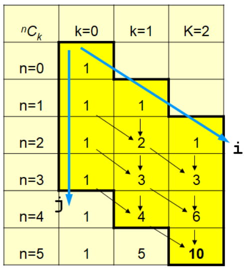
4. **Combinations (^nC_k)** [Home Work]

a) Encode in Dafny a function to define ${}^{n}C_{k}$ according to the Pascal rule ${}^{n}C_{k} = {}^{n-1}C_{k} + {}^{n-1}C_{k-1}$ $(0<k<n)$, with boundary cases ${}^{n}C_{k}=1$, if $k=0 \lor k=n$.

b) A method to calculate ${}^{n}C_{k}$ efficiently in terms of time and space using dynamic programming is reproduced below (from "Conceção e Análise de Algoritmos"). Encode it in Dafny and prove its correctness with respect to the definition in (a).

${}^{n}C_{k}$ - Programação dinâmica

Para economizar memória, passa-se a abordagem *bottom-up*.



Calculando da esquerda para a direita, basta memorizar uma coluna.

ou

Calculando de cima para baixo, basta memorizar uma linha (diagonal).

Implementação

```
long combDynProg(int n, int k) {
  int maxj = n - k;
  long c[1 + maxj];
  for (int j = 0; j <= maxj; j++)
    c[j] = 1;                          n-k+1 vezes
  for (int i = 1; i <= k; i++)
    for (int j = 1; j <= maxj; j++)
      c[j] += c[j-1];                  k(n-k) vezes
  return c[maxj];
}
```

Tempo: T(n,k) = O(k(n-k))
Espaço: S(n,k) = O(n-k)
(0<k<n, senão O(1))